PALASM® 4 Reference Guide

1992

Advanced
Micro
Devices

PALASM 4 release 1.5

PALASM® 4 USER'S MANUAL

VOLUME 2 - PALASM 4 REFERENCE GUIDE

901 Thompson Place
P.O. Box 3453
Sunnyvale, CA 94088

TEL: 408-732-2400
TWX: 910339-9280
TELEX: 34-6306
TOLL FREE: 800-538-8450

APPLICATIONS HOTLINE: 800-222-9323

Advanced Micro Devices reserves the right to make changes in specifications at any time and without notice. The information furnished by Advanced Micro Devices is believed to be accurate and reliable. However, no responsibility is assumed by Advanced Micro Devices for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No !icense is granted under any patents or patent rights of Advanced Micro Devices.

PALASM 4 USER'S MANUAL

DISCONTINUED

DISCONTINUED

TABLE OF CONTENTS

VOLUME 1 - Getting Started and MACH Workbook

SECTION I: GETTING STARTED

CHAPTER 1: INSTALLATION GUIDE

CHAPTER 2: DESIGN ENTRY

CHAPTER 3: SCHEMATIC-BASED MACH DESIGN DEMONSTRATION

SECTION II: DESIGNER'S GUIDE

CHAPTER 4: ENTRY

CHAPTER 5: COMPILATION / FITTING

CHAPTER 6: SIMULATION

MACH Design Workbook

VOLUME 2 - Reference Guide

SECTION III: LIBRARY REFERENCE

CHAPTER 7: INTRODUCTION

CHAPTER 8: MACRO AND SCHEMATIC DATASHEETS

SECTION IV: SOFTWARE REFERENCE

CHAPTER 9: MENUS AND COMMANDS

CHAPTER 10: LANGUAGE REFERENCE

CHAPTER 11: DEVICE PROGRAMMING REFERENCE

SECTION V: APPENDICES

APPENDIX A: PLD TEXT EDITOR

SECTION VI: GLOSSARY / INDEX

GLOSSARY

INDEX

SECTION III
LIBRARY REFERENCE

CHAPTER 7

INTRODUCTION

CONTENTS

7 INTRODUCTION

This chapter introduces the MACH library[1] and its functional elements, called macros, in four discussions.

- The overview, 7.1, introduces the categories of macros contained within the AMD-supplied library.
- The annotated schematic design discussion, 7.2, illustrates considerations for capturing a schematic and points to the corresponding descriptions in discussion 7.3.
- The capturing a schematic discussion, 7.3, describes how to use the library macros effectively to create a design using OrCAD/SDT III.
- The annotated datasheet, 7.4, provides a sample datasheet from Chapter 8, in this section, with a description of each information field.

[1] Only the MACH family of devices is supported in the library. The library must be purchased separately from the PALASM 4 software.

7.1 MACH LIBRARY OVERVIEW

AMD provides over 100 commonly used logic functions as macros in the MACH library. A summary of the available macros is shown next.[2]

BUFFER MACROS	COMBINATORIAL MACROS	OTHER MACROS	STORAGE MACROS	TTL-EQUIVALENT MACROS
BUF INV NTRST TRST	AND NAND NOR OR XNOR XOR	AINIT NC NODE PDWN PUP	FD FT LD LDX DFF DLAT DLATX TFF	See Chapter 8

[2] Refer to Chapter 8, in this section, for a complete list of macros, symbols, and datasheets.

7.2 ANNOTATED SCHEMATIC

This discussion presents an annotated schematic to illustrate the features and considerations of capturing a MACH schematic design. Each lettered paragraph below corresponds to a circled letter in the next figure.

A. You use the AINIT macro to specify the asynchronous preset and reset for all three terminal storage macros in a design.

   In this case, the T-type flip-flops in the 74163 are set and reset by the signals ASYNC_SET and ASYNC_RST. Use five-terminal storage macros to specify multiple asynchronous preset and reset functions.

B. You use the NODE macro on a wire connected to a module port to specify a package pin number. In this case, the clock is connected to pin 13.

C. You use the PDWN and PUP macros to disable unused TTL-equivalent macro inputs, preventing the unused input signals from being connected to the package pins due to lower-level module ports.

   Use the PDWN macro to disable an active-high input and the PUP macro to disable an active-low input.

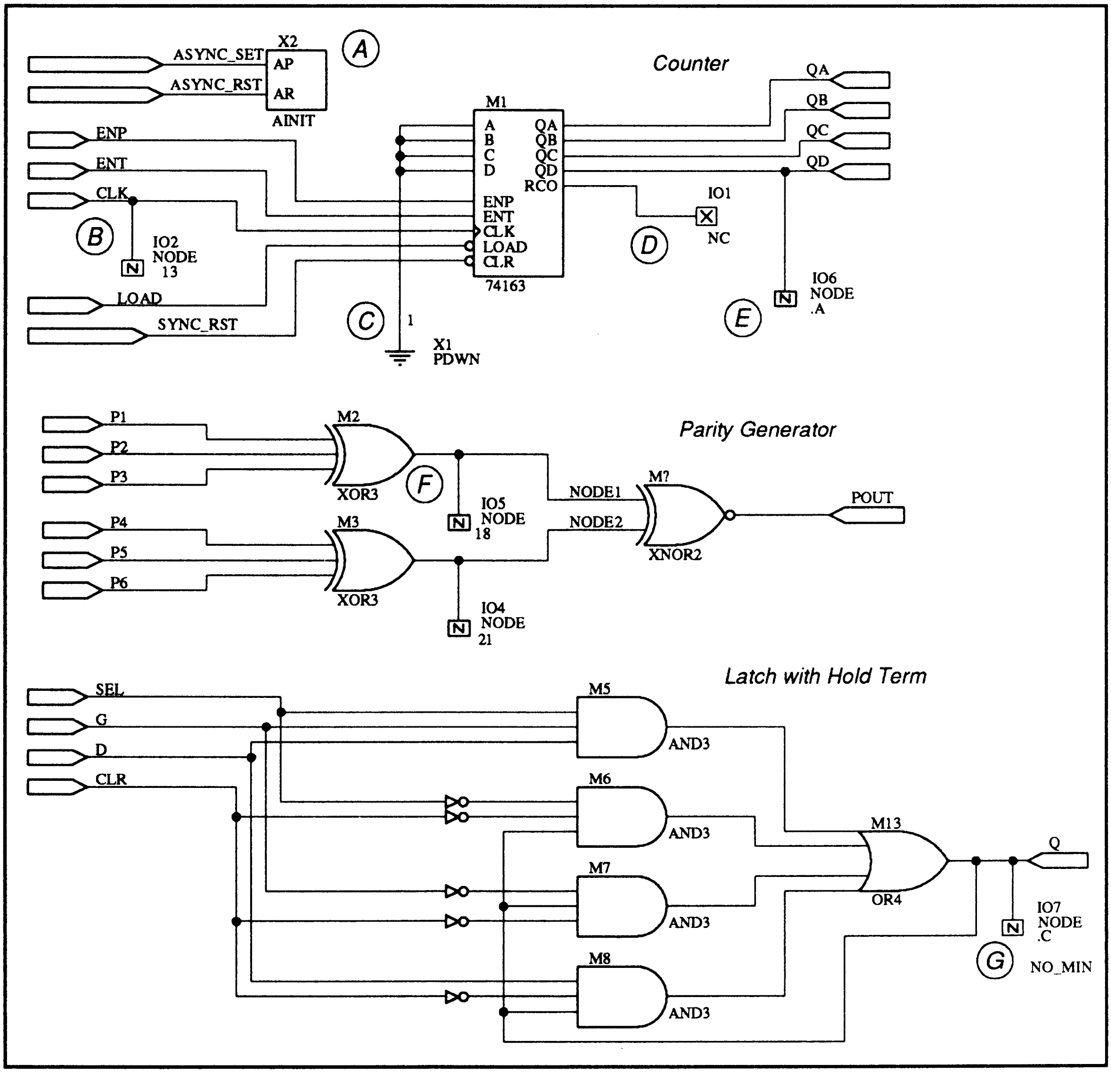
D. You use the NC macro to terminate unused TTL-equivalent macro outputs. Any unused logic associated with the designated output is then removed during later processing.

E. You use the NODE macro to assign logic to specific blocks in the MACH device. In this case, the equation driving QD is grouped into block A.

F. You use the NODE macro to specify the node number of a buried node signal. In this case, the signal NODE1 is placed at location 18.[3]

G. You use the NODE macro to prevent minimization of logic. In this case, inhibiting minimization preserves the redundant hold term of the latch design.

[3] Refer to Section IV, Chapter 11, for details on node numbers for each MACH device.



Annotated Schematic

7.3 CAPTURING A SCHEMATIC

You capture a MACH schematic using OrCAD/SDT III and the optional MACH macro library. PALASM 4 provides an interface to the OrCAD/SDT III software.[4]

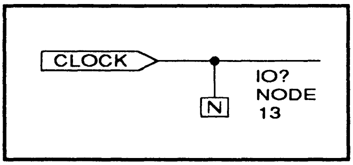
The features and considerations for capturing a MACH schematic design are presented in the following discussions.

- 7.3.1, Specifying Pin and Node Numbers
- 7.3.2, Grouping Signals into a Block
- 7.3.3, Turning Minimization Off
- 7.3.4, Manually Splitting Product Terms
- 7.3.5, Terminating Unused Inputs and Outputs
- 7.3.6, Defining Preset and Reset Functions
- 7.3.7, Interpreting Reference Designators
- 7.3.8, Naming Signals

7.3.1 SPECIFYING PIN AND NODE NUMBERS

I/O pins in a MACH design are automatically defined by unresolved module ports, ones that do not have a complementary connection. All module ports in the design above are unresolved.

Note: A module port is automatically linked to an I/O pin when it is unresolved.



To specify an I/O pin or a node location, you connect a NODE macro to the corresponding signal wire and edit part field 1 to indicate the desired number. You enter only the pin or node number; no alpha characters are allowed.

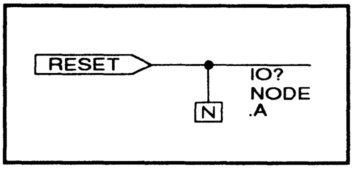
[4] Refer to Section I, Chapter 1, for details on installing the interface to OrCAD/SDT III.

Important: Assign fixed I/O package pin locations only when necessary. Letting pins float allows more flexibility during the fitting process and results in a higher probability of success.

Also: You must connect the NODE macro to an individual signal wire; connections to bus wires are ignored.

7.3.2 GROUPING SIGNALS INTO A BLOCK

To group the logic associated with a signal into a specific block, you can connect a NODE macro to the corresponding signal wire and edit part field 2. The block letter must be preceded by a period, for example, .A or .B.



Important: You must use the NODE macro to assign attributes to buffer, combinatorial, and module port nets. Storage and TTL-equivalent macros have inherent attribute fields.

7.3.3 TURNING MINIMIZATION OFF

During compilation, logic is replaced with its DeMorgan equivalent if the latter implementation requires fewer product terms.

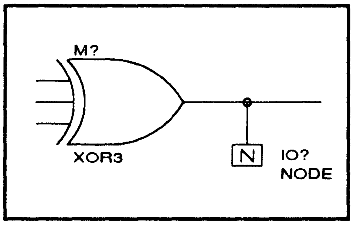
Important: When DeMorganized equivalents are substituted for clocked macros, polarity inversion occurs after the storage element. As a result, the operation of the set/reset logic is reversed when viewed at the pin.

Inhibiting minimization also preserves the redundant hold used in latch designs to prevent timing glitches.

To prevent minimization for the logic associated with a signal, you connect a NODE macro to the corresponding signal wire and edit the name of part field 3 to NO_MIN.

7.3.4 MANUALLY SPLITTING PRODUCT TERMS

The 6-input parity generator, shown in the annotated schematic discussion, is an example of a circuit that requires gate splitting. To implement the complete function requires 32 product terms. However, a MACH 110 device can support only 12 product terms for a single macrocell.[5]

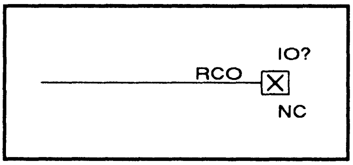


You can split the product terms of this function by placing a NODE macro at the output of each XOR3, thereby creating two buried nodes. The buried node results are fed back into the array to produce the result POUT. Each of the smaller XOR functions fits into a single macrocell; the final implementation of the parity generator fits into a MACH 110 device.

Important: Splitting product terms requires feedback through the array that introduces an additional unit of delay in the signal path. You must account for this added delay.

7.3.5 TERMINATING UNUSED INPUTS AND OUTPUTS

You must disable unused TTL-equivalent macro inputs by connecting them to supply or ground. This prevents the unused input signals from being connected to package pins due to lower-level module ports.

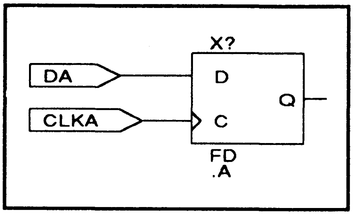


Similarly, you must terminate unused TTL-equivalent macro outputs with an NC macro. Any unused logic associated with the designated output is then removed during later processing.

[5] Refer to Section IV, Chapter 11, for details on steering product terms.

7.3.6 DEFINING PRESET AND RESET FUNCTIONS

The MACH library contains storage macros with either implied or explicit asynchronous set and reset functions. You specify these functions to define when asynchronous set or reset signals are generated after the device powers up. All flip-flops and latches are automatically reset when the device powers up, independent of the asynchronous control logic.



The AINIT macro represents the implied asynchronous set and reset functions for all of the three-terminal storage macros in a design. The following storage macros require the AINIT macro for asynchronous set and reset definition.

- FD
- FT
- LD
- LDX

Note: Only one AINIT macro can be specified per design. Any additional reset functions must be defined explicitly.

The following storage macros have five terminals, thereby allowing you to explicitly connect the appropriate asynchronous set and reset function.

- DFF
- DLAT
- DLATX
- TFF

Important: Only one asynchronous set and reset signal can be implemented in a single block within the MACH device. Each time you specify a unique set or reset function, the new logic is placed in a separate block.

7.3.7 INTERPRETING REFERENCE DESIGNATORS

Each instance of a macro in a schematic must have a unique reference designator. Reference designators are automatically assigned during the compilation process.

Note: Each reference designator must be unique across all levels of hierarchy, including the subsheets of MACH macros.

The following types of reference designators are used in the AMD-supplied macro libraries.

- **M?** appears with combinatorial, TTL-equivalent, and non-three-state buffer macros.
- **X?** appears with storage and three-state macros.
- **IO?** appears with the NODE and NC macros.

Tip: Error and warning messages include the reference designator and net name of the flagged logic; therefore, it is a good idea to name all nets in the design.

7.3.8 NAMING SIGNALS

Use an alphanumeric string no longer than nine characters for each signal name. Signal names are not case sensitive.

Important: A forward slash, /, does **not** affect the polarity in a schematic signal name.

Note: When a schematic-based design is compiled, a <design>.PDS file is generated. The signal names in this file may have prefixes or suffixes to guarantee uniqueness in the design hierarchy. For example, a forward slash is converted to a _FS_ prefix.

7.4 ANNOTATED DATASHEET

This discussion presents an annotated datasheet to illustrate the layout and information contained therein. Chapter 8, in this section, contains datasheets for each of the TTL-equivalent macros available in the MACH library. Each lettered paragraph below corresponds to a circled letter in the next two figures.

A. The feature summary is a bulleted list that summarizes the logical features of the macro.

B. The logic symbol illustrates the macro pin names and how they appear on the symbol.

C. The MACH resource summary lists the number of resources consumed by a stand-alone macro. This summary does not account for potential minimization or conflicts with other logic in a design. The following utilized resources are listed.

- Macrocell count
- Array inputs
- Product terms used
- Product terms allocated
- MACH-family restrictions, if applicable

D. The functional description explains the logical operation of the macro, and corresponds to the information presented in the function table. It also indicates if the macro is a non-standard TTL implementation.

E. The sample PDS equivalent represents the PDS file generated when the macro schematic is processed as a stand-alone design.

F. The function table illustrates the functional operation of the macro. Logic states are represented as follows.

- H represents a logical 1, or HIGH, state.
- L represents a logical 0, or LOW, state.
- X represents a don't-care condition.
- An up arrow represents a rising edge.

G. The schematic is a logical representation of the macro circuit as created in OrCAD/SDT III. The following graphic changes have been made to enhance the clarity of the schematics. No changes have been made to the logic.

- Reference designators have been removed.
- Part values have been removed from non-storage macros.
- Module-port names have been relocated to a position above the module-port symbol.
- Some pin names have been removed to prevent crowding.

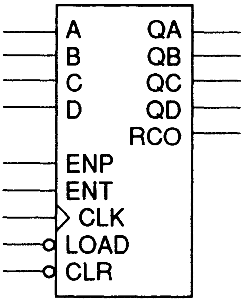
74163 4-Bit Binary Counter w/ Synchronous Reset 74163

Ⓐ

- Synchronous 4-bit binary-loadable up counter
- Synchronous reset
- Carry look-ahead output for making wider counters

Ⓑ

Logic Symbol



Ⓒ

Macrocell count:	5
Array inputs:	12
Product terms used:	17
Product terms allocated:	20

Ⓓ

Functional Description

The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count of 15 has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

Ⓔ

Sample PDS Equivalent

```
QA.T = ((/CLR * QA) + (CLR * /LOAD
  * /A * QA) + (CLR * /LOAD * /QA
  * A) + (CLR * ENP * LOAD * ENT))
QA.clkf = CLK
QB.T = ((/CLR * QB) + (CLR * /LOAD
  * /B * QB) + (CLR * /LOAD * /QB
  * B) + (CLR * QA * (CLR * ENP
  * LOAD * ENT)))
QB.clkf = CLK
QC.T = ((/CLR * QC) + (CLR * /LOAD
  * /C * QC) + (CLR * /LOAD * /QC
  * C) + (CLR * QB * QA * (CLR * ENP
  * LOAD * ENT)))
QC.clkf = CLK
QD.T = ((/CLR * QD) + (CLR * /LOAD
  * /D * QD) + (CLR * /LOAD * /QD
  * D) + (CLR * QC * QB * QA * (CLR
  * ENP * LOAD * ENT)))
QD.clkf = CLK
RCO = (QD * QC * QB * QA * ENT)
```

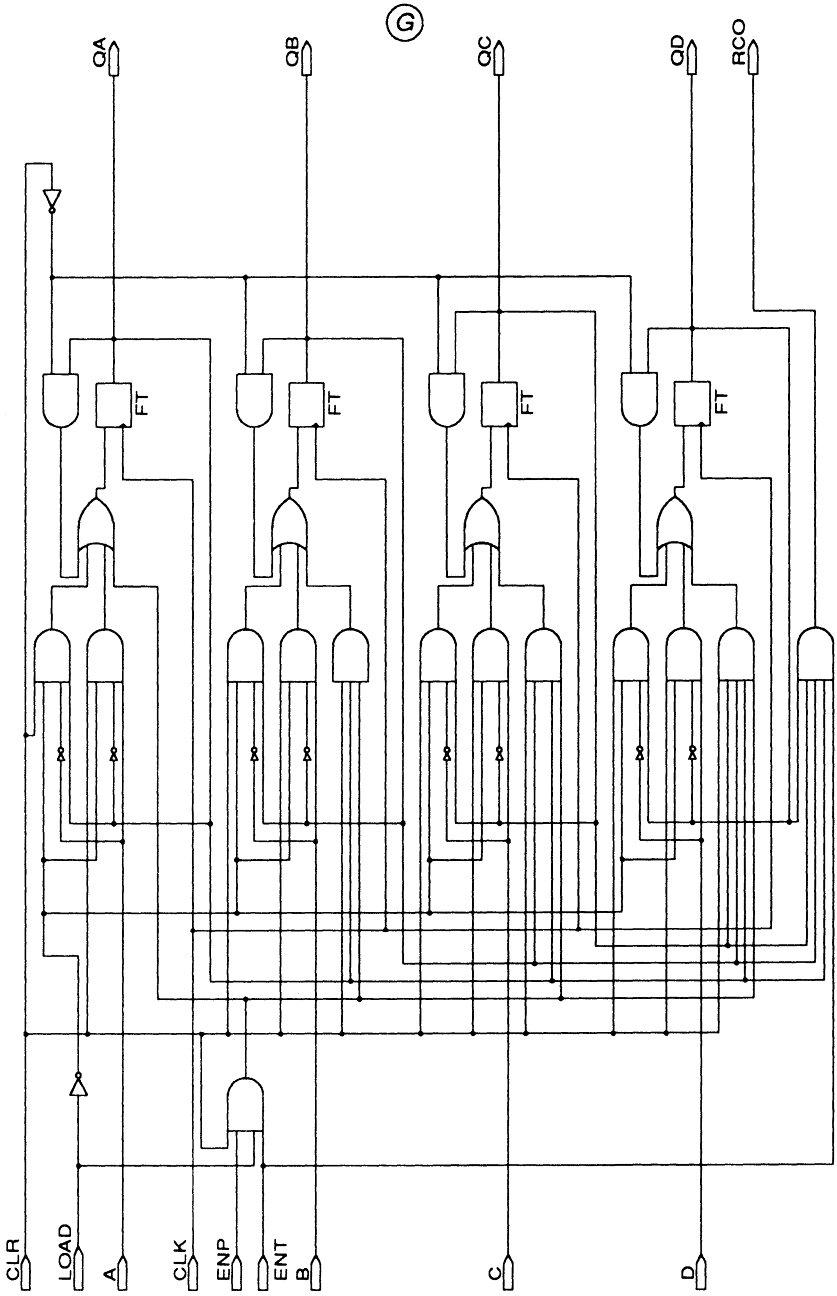
Ⓕ

Function Table

	Inputs					Outputs			
Mode	**CLK**	**CLR**	**Load**	**ENP**	**ENT**	**QD**	**QC**	**QB**	**QA**
Clear	L	L	X	X	X	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 15 and ENT is HIGH. Otherwise, it stays LOW.

4-Bit Binary Counter w/ Synchronous Reset



7.2.2 LATCH MACROS

MACH 2 series devices have special resources to implement latch macros directly. MACH 1 series devices implement the latch macros as combinatorial functions.

7.2.3 MINIMIZATION

During compilation, logic is replaced with its DeMorgan equivalent if the latter implementation is more economical.

Important: When DeMorganized equivalents are substituted for clocked macros, polarity inversion occurs after the storage element. As a result, the operation of the Set/Reset logic is reversed when viewed at the pin.

To prevent this reversal, you use the NODE macro and specify the NO_MIN attribute. Inhibiting minimization also preserves the redundant hold used in latch designs to prevent timing glitches.

7.2.4 REFERENCE DESIGNATORS

Each instance of a macro in a schematic must have a unique reference designator. Reference designators are automatically assigned during the compilation process.

Note: Each reference designator must be unique across all levels of hierarchy, including the subsheets of MACH macros.

The following types of reference designators are used in the AMD-supplied macro libraries.

- **M?** appears with combinatorial, TTL-equivalent, and non-three-state buffer macros.

- **X?** appears with storage and three-state macros.

- **IO?** appears with the NODE and NC macros.

Tip: Error and warning messages include the reference designator and net name of the flagged logic; therefore, it is a good idea to name all nets in the design.

7.2.5 SIGNAL NAMES

Use an alphanumeric string no longer than nine characters for each signal name. Signal names are not case sensitive.

Important: A slash, /, does **not** affect the polarity in a schematic signal name.

7.3 ANNOTATED DATASHEET

This discussion presents an annotated datasheet to illustrate the layout and information contained therein. Chapter 8 contains datasheets for each of the TTL-equivalent macros available in the MACH library. Each lettered paragraph below corresponds to a circled letter in the next two figures.

A. The feature summary is a bulleted list that summarizes the logical features of the macro.

B. The logic symbol illustrates the macro pin names and how they appear on the symbol.

C. The MACH resource summary lists the number of resources consumed by a stand-alone macro. This summary does not account for potential minimization or conflicts with other logic in a design. The following utilized resources are listed.

   - Macrocell count
   - Array inputs
   - Product terms used
   - Product terms allocated
   - MACH-family restrictions, if applicable

D. The functional description explains the logical operation of the macro, and corresponds to the information presented in the function table. It also indicates if the macro is a non-standard TTL implementation.

E. The sample PDS equivalent represents the PDS file that is generated when the macro schematic is processed as a stand-alone design.

F. The function table illustrates the functional operation of the macro. Logic states are represented as follows.

- 1 represents a logical 1, or HIGH state
- 0 represents a logical 0, or LOW state
- X represents a don't-care condition
- An arrow represents a rising edge

G. The schematic is a logical representation of the macro circuit as created in OrCAD/SDT III. The following graphic changes have been made to enhance the clarity of the schematics. No changes have been made to the logic.

- Reference designators have been removed.

- Part values have been removed from non-storage macros.

- Module-port names have been relocated to a position above the module-port symbol.

- Some pin names have been removed to prevent crowding.

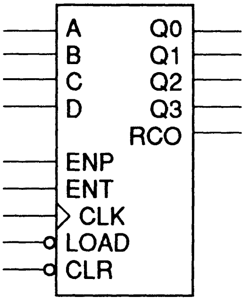
74163 4-Bit Binary Counter w/ Synchronous Reset 74163

Ⓐ

- Synchronous 4-bit binary-loadable up counter
- Synchronous reset
- Carry look-ahead output for making wider counters

Ⓑ

Logic Symbol



Ⓒ

Macrocell count:	5
Array inputs:	12
Product terms used:	17
Product terms allocated:	20

Ⓓ

Functional Description

The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count of 15 has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

Ⓔ

Sample PDS Equivalent

```
QA.T = ((/CLR * QA) + (CLR * /LOAD
  * /A * QA) + (CLR * /LOAD * /QA
  * A) + (CLR * ENP * LOAD * ENT))
QA.clkf = CLK
QB.T = ((/CLR * QB) + (CLR * /LOAD
  * /B * QB) + (CLR * /LOAD * /QB
  * B) + (CLR * QA * (CLR * ENP
  * LOAD * ENT)))
QB.clkf = CLK
QC.T = ((/CLR * QC) + (CLR * /LOAD
  * /C * QC) + (CLR * /LOAD * /QC
  * C) + (CLR * QB * QA * (CLR * ENP
  * LOAD * ENT)))
QC.clkf = CLK
QD.T = ((/CLR * QD) + (CLR * /LOAD
  * /D * QD) + (CLR * /LOAD * /QD
  * D) + (CLR * QC * QB * QA * (CLR
  * ENP * LOAD * ENT)))
QD.clkf = CLK
RCO = (QD * QC * QB * QA * ENT)
```

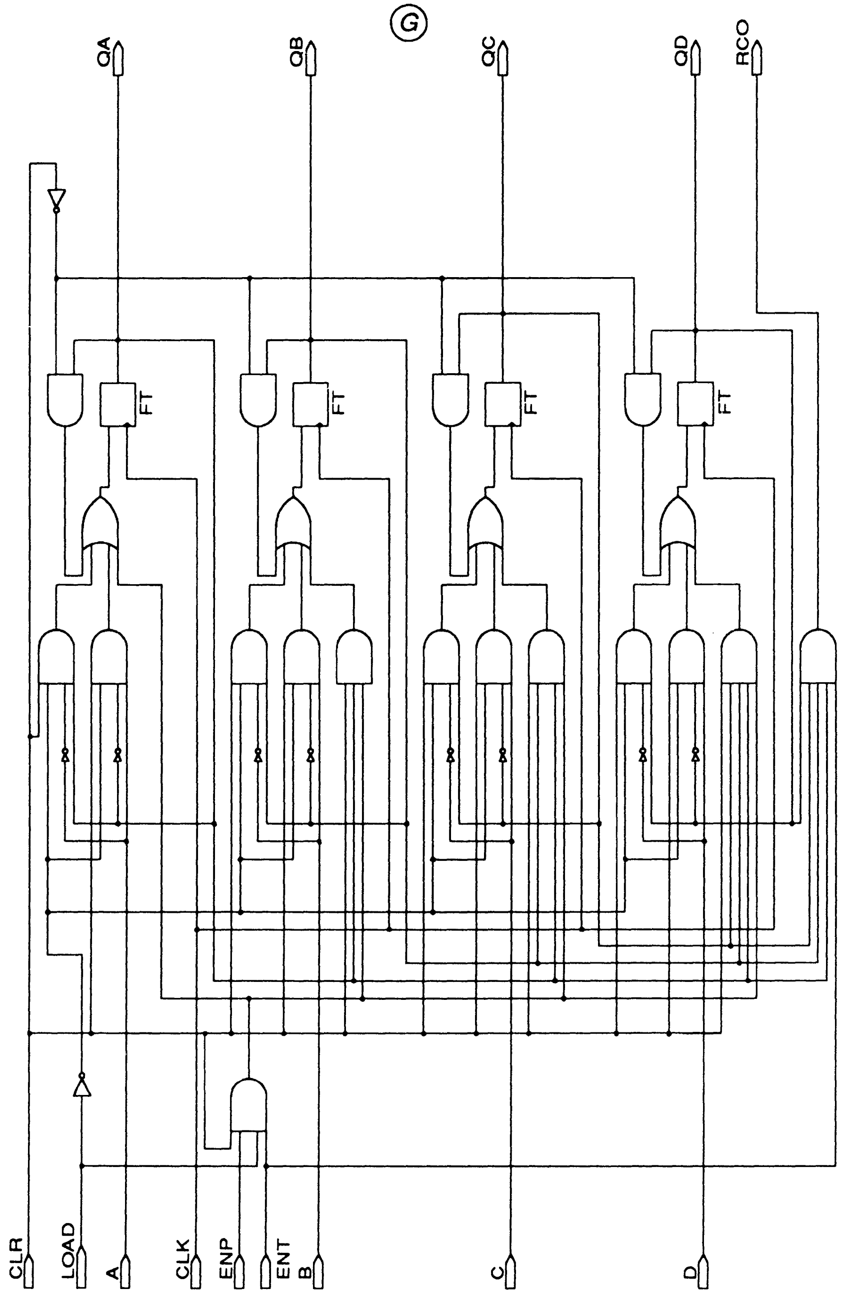
Ⓕ

Function Table

	Inputs					Outputs			
Mode	**CLK**	**CLR**	**Load**	**ENP**	**ENT**	**QD**	**QC**	**QB**	**QA**
Clear	L	L	X	X	X	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 15 and ENT is HIGH. Otherwise, it stays LOW.

74163 4-Bit Binary Counter w/ Synchronous Reset 74163

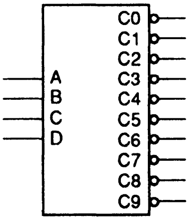


CHAPTER 8

MACRO AND SCHEMATIC DATASHEETS

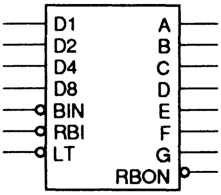
CONTENTS

7442 BCD-to-Decimal Decoder 7442



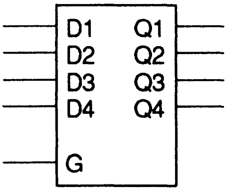
The 7442 macro decodes a 4-bit BCD input into a single active-LOW output. All outputs remain HIGH for invalid inputs.

7448 BCD-to-7-Segment Decoder 7448



The 7448 macro decodes a 4-bit BCD input into a 7-segment-display format.

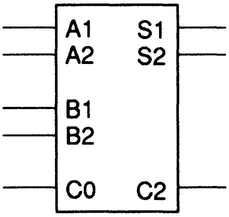
7477 4-Bit Latch 7477



The 7477 macro is a 4-bit latch. The Q1 – Q4 outputs follow the D1 – D4 input data when the G latch enable is set HIGH. When G is set LOW, the outputs latch the input data and further activity at D1 – D4 is ignored.

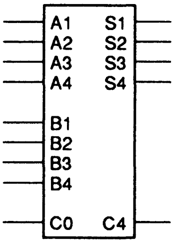
Note: The TTL version contains two 4-bit latches.

7482 2-Bit Full Adder 7482



The 7482 macro adds two 2-bit numbers using carry look-ahead logic to generate the carry out, C2.

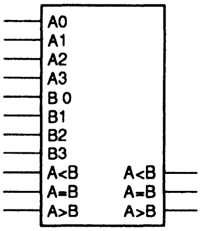
7483 4-Bit Full Adder 7483



The 7483 macro adds two 4-bit numbers using carry look-ahead logic to generate the internal 2-bit carry, C2, and the final carry out, C4.

Note: The first 2-bit addition is performed as shown in the 7482 data sheet.

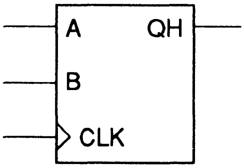
7485 4-Bit Magnitude Comparator 7485



The 7485 macro compares two 4-bit numbers, then activates one of the three outputs: A=B, A>B, or A<B. For cascaded 7485 macros, the least-significant stage must have the A=B input HIGH and the A>B and A<B inputs LOW.

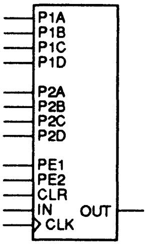
Note: The TTL version functions differently when more than one cascading input is HIGH or when all cascaded inputs are LOW.

7491 8-Bit Shift Register 7491



The 7491 macro is an 8-bit serial-in serial-out shift register. The serial-input stream is the result of logically ANDing inputs A and B.

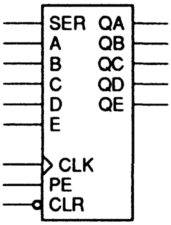
7494 4-Bit Shift Register 7494



The 7494 macro is a 4-bit serial-in serial-out shift register with asynchronous clear and synchronous preset logic.

Note: The TTL version has asynchronous preset logic.

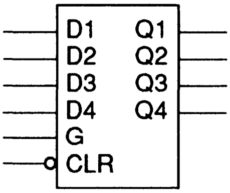
7496 5-Bit Shift Register 7496



The 7496 macro is a 5-bit shift register that offers access to each flip-flop's input and output. The 5-bit value at inputs A – E is preloaded into the shift register on the rising-edge of CLK when PE is HIGH. The preset function overrides the shift operation.

Note: The TTL version has asynchronous preset logic.

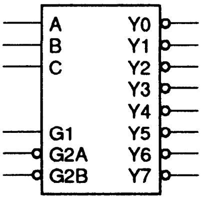
74116 4-Bit Latch w/ Clear 74116



The 74116 macro is a 4-bit latch with an asynchronous reset.

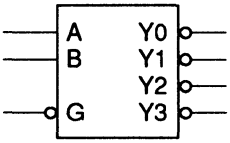
Note: The TTL version contains two 4-bit latches.

74138 3-to-8 Line Decoder 74138



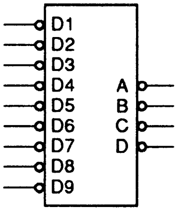
The 74138 macro decodes a 3-bit binary input into a single active-LOW output. You can cascade these macros to implement a decoder with up to 24 outputs via the three enable inputs: G1, G2A, and G2B.

74139 2-to-4 Line Decoder 74139



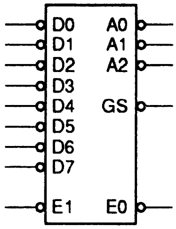
The 74139 macro decodes one of four active-LOW outputs depending on two data inputs. The active-LOW enable input, G, can be used as an input when decoding more output lines.

74147 10-to-4 Priority Line Encoder 74147



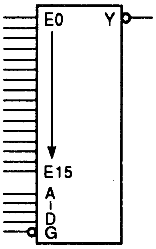
The 74147 macro generates a 4-bit BCD output code that represents the highest-order-LOW data input. Priority encoding of the inputs ensures that only the highest-order data-input line is encoded.

74148 8-to-3 Priority Line Encoder 74148



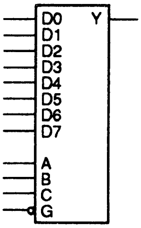
The 74148 macro generates a 3-bit binary output code that represents the highest-order-LOW data input. You can use the input enable, EI, and output enable, EO, to expand priority encoding.

74150 16-to-1 Multiplexer w/ Enable 74150



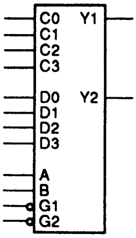
The 74150 macro decodes four data-input lines to select one of 16 data sources. The enable input, G, must be LOW to enable the Y output.

74151 8-to-1 Multiplexer w/ Enable 74151



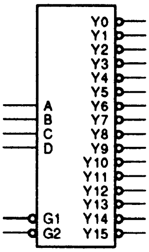
The 74151 macro decodes three data-input lines to select one of eight data sources. The enable input, G, must be LOW to enable the Y output.

74153 Dual 4-to-1 Multiplexer w/ Enable 74153



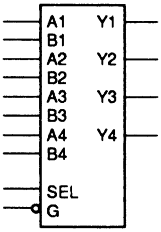
The 74153 macro consists of two 4-to-1 multiplexers with common data-select lines. Each 4-to-1 multiplexer has an active LOW strobe input line to enable the output.

74154 4-to-16 Line Decoder 74154



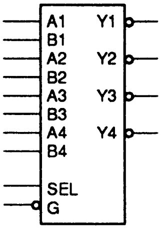
The 74154 macro decodes four data-input lines to select one of 16 active LOW outputs. You can use the enable inputs, G1 and G2, to cascade multiple macros.

74157 Quad 2-to-1 Multiplexer 74157



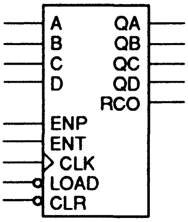
The 74157 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced LOW regardless of the inputs.

74158 Quad 2-to-1 Multiplexer 74158



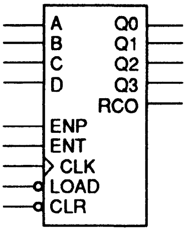
The 74158 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced HIGH regardless of the inputs.

74162 4-Bit BCD/Decade Counter w/ Synchronous Reset 74162



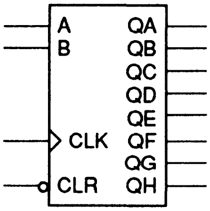
The 74162 macro is a 4-bit BCD-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded.

74163 4-Bit Binary Counter w/ Synchronous Reset 74163



The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded.

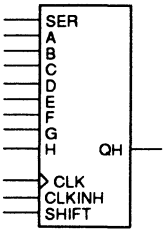
74164 8-Bit Serial-In Parallel-Out Shift Register 74164



The 74164 macro is an 8-bit serial-in parallel-out shift register with synchronous reset. The two serial inputs, A and B, are logically ANDed.

Note: The TTL version has asynchronous reset logic.

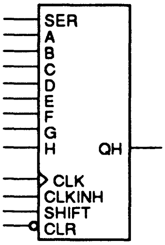
74165 Parallel-Load 8-Bit Shift Register 74165



The 74165 macro is an 8-bit parallel-in serial-out shift register. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values. A parallel-load operation overrides the inhibit function.

Note: The TTL version has two clock lines and asynchronous load logic.

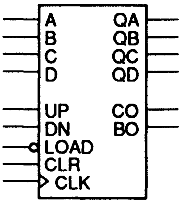
74166 8-Bit Parallel-In Serial-Out Shift Register 74166



The 74166 macro is an 8-bit parallel-in serial-out shift register with synchronous reset. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values.

Note: The TTL version has asynchronous reset logic.

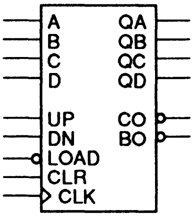
74192 4-Bit Up/Down BCD Counter 74192



The 74192 macro is a 4-bit up/down BCD counter with synchronous parallel load and asynchronous reset logic.

Note: The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

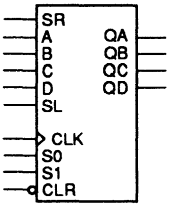
74193 4-Bit Up/Down Binary Counter 74193



The 74193 macro is a 4-bit up/down binary counter with synchronous load and asynchronous reset logic.

Note: The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

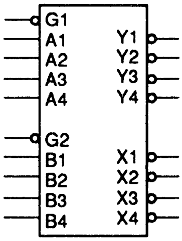
74194 4-Bit Bidirectional Universal Shift Register 74194



The 74194 macro is a 4-bit bidirectional universal shift register with synchronous reset logic.

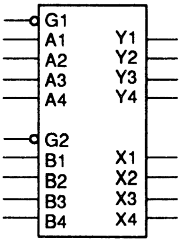
Note: The TTL version has asynchronous reset logic.

74240 Octal Inverting Buffers w/ 3-State Outputs 74240



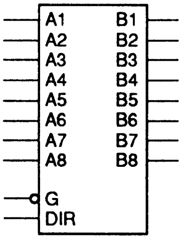
The 74240 macro contains two groups of four inverting buffers. Each group is enabled by an active-LOW input control line.

74244 Octal Non-Inverting Buffers w/ 3-State Outputs 74244



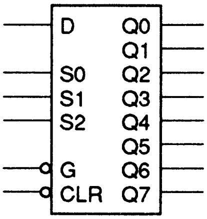
The 74244 macro contains two groups of four non-inverting buffers. Each group is enabled by an active-LOW input control line.

74245 Octal Bus Transceivers w/ 3-State Outputs 74245



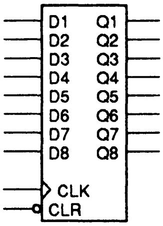
The 74245 macro implements an 8-bit bus transceiver. You can transmit data from bus A to bus B or from bus B to bus A. The data-transfer direction is controlled by the DIR control line. If the enable input, G, is set HIGH, then the buses are disabled and isolated.

74259 8-Bit Addressable Latch 74259



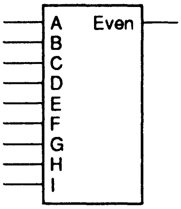
The 74259 macro is an 8-bit addressable latch with asynchronous reset logic. The following four modes of operation are selectable via the CLR and G inputs: addressable latch, memory, active-HIGH 3-to-8 demultiplexer, reset.

74273 Octal D-Type Flip-Flops 74273



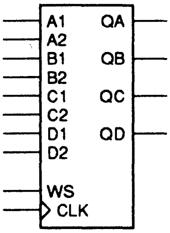
The 74273 macro is an octal D-FF bank with asynchronous reset logic.

74280 9-Bit Parity Generator/Checker 74280



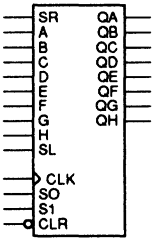
The 74280 macro is a combinatorial circuit that generates or checks for even parity on nine data lines. Odd parity is obtained by taking the inversion of the EVEN parity output.

74298 Quad 2-to-1 Multiplexer with Storage 74298



The 74298 macro selects one of two 4-bit words and stores each bit in a register on the rising edge of CLK.

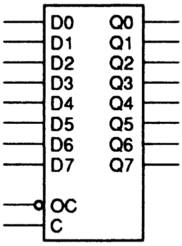
74299X 8-Bit Bidirectional Universal Shift Register 74299X



The 74299X macro is composed of two 74194s connected to form an 8-bit bidirectional universal shift register with synchronous reset logic.

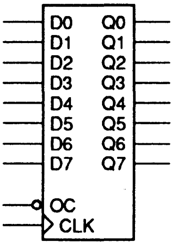
Note: The TTL version has three-state bidirectional I/Os that serve as the parallel-load inputs as well as the Q outputs.

74373 Octal D-Type Latches with 3-State Outputs 74373



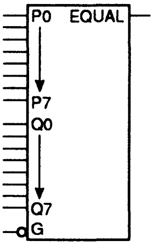
The 74373 macro is an octal D latch with an active-LOW enable input.

74374 Octal D-Type Flip-Flops with 3-State Outputs 74374



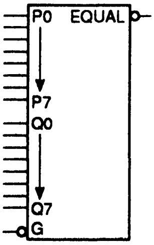
The 74374 macro is an octal D-type register with an active-LOW enable input.

74518 8-Bit Identity Comparator 74518



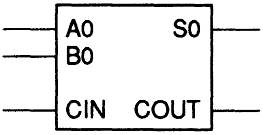
The 74518 macro compares two 8-bit numbers and sets the EQUAL output HIGH if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

74521 8-Bit Identity Comparator 74521



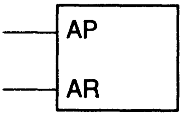
The 74518 macro compares two 8-bit numbers and sets the EQUAL output LOW if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

ADD1 1-Bit Full Adder ADD1



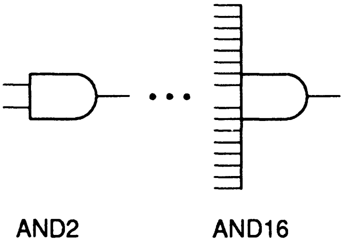
The ADD1 macro adds two 1-bit numbers. You can use the carry-out and carry-in signals to cascade multiple adders

AINIT Implicit Preset and Reset AINIT



The AINIT macro specifies the asynchronous preset and reset functions for all of the associated three-terminal flip-flops and latches.

AND AND Gates AND

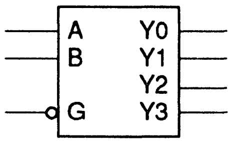


The AND2 thru AND16 macros are AND gates with two to 16 inputs.

BUF Buffer BUF

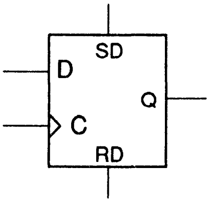
The BUF macro is a single input non-inverting buffer.

DECODE4 — 2-to-4 Line Decoder — DECODE4



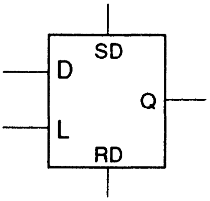
The DECODE4 macro decodes one of four active-HIGH output lines depending on the 2-bit data inputs. The enable input, G, must be LOW to activate the decoder. You can use the enable inputs to cascade multiple decoders.

DFF — D Flip-Flop — DFF



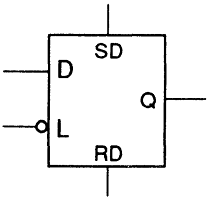
The DFF macro is a five-terminal D-type flip-flop.

DLAT — D Latch — DLAT



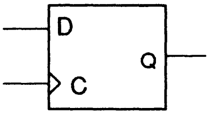
The DLAT macro is a five-terminal latch with an active-high latch input.

DLATX — D Latch — DLATX



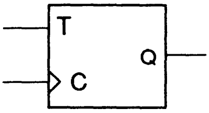
The DLATX macro is a five-terminal latch with an active-low latch input.

FD D Flip-Flop FD



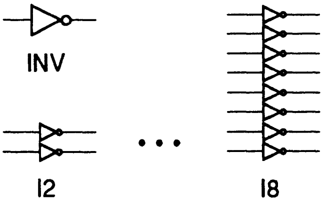
The FD macro is a three-terminal D-type flip-flop. Use the AINIT macro to specify the asynchronous preset and reset functions.

FT T Flip-Flop FT



The FT macro is a three-terminal T-type flip-flop. Use the AINIT macro to specify the asynchronous preset and reset functions.

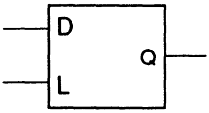
INV, I Inverters INV, I



The INV macro is a single input inverter.

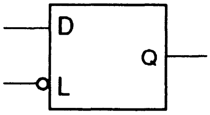
The I2 through I8 macros are banks of inverters with two to eight elements.

LD D Latch LD



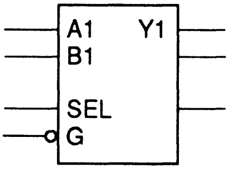
The LD macro is a three-terminal latch with an active-high latch input. Use the AINIT macro to specify the asynchronous preset and reset functions.

LDX D Latch LDX



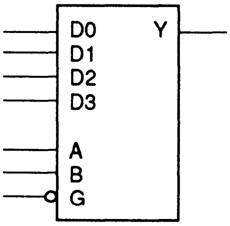
The LDX macro is a three-terminal latch with an active-low latch input. Use the AINIT macro to specify the asynchronous preset and reset functions.

MUX2 2-to-1 Multiplexer w/ Enable MUX2



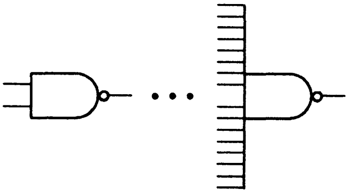
The MUX2 macro decodes one data-input line to select one of two data sources. The enable input, G, must be LOW to enable the Y1 output.

MUX4 4-to-1 Multiplexer w/ Enable MUX4



The MUX4 macro decodes two data-input lines to select one of four data sources. The enable input, G, must be LOW to enable the Y output.

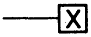
NAND NAND Gates NAND



NAND2 NAND16

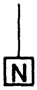
The NAND2 thru NAND16 macros are NAND gates with two to 16 inputs.

NC No Connect NC



The NC macro terminates unused outputs from other macros.

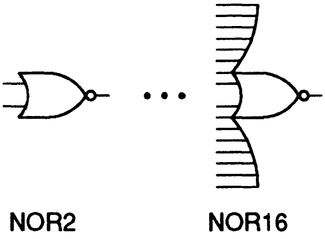
NODE Node NODE



The NODE macro forces a signal to an internal node and provides attributes for specifying node #, blockname, and no minimization.

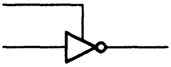
When connected to a module port, this macro does not force an internal node, but provides an attribute for specifying the pin #.

NOR NOR Gates NOR



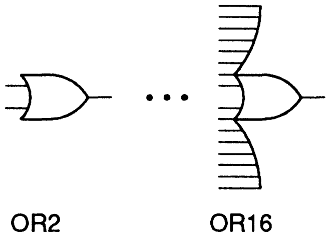
The NOR2 through NOR16 macros are NOR gates with two to 16 inputs.

NTRST Inverting Three-State Buffer NTRST



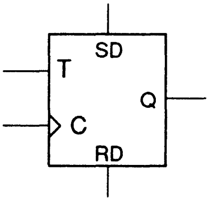
The NTRST macro is an inverting three-state buffer.

OR — OR Gates — OR



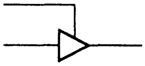
The OR2 through OR16 macros are OR gates with two to 16 inputs.

TFF — T Flip-Flop — TFF



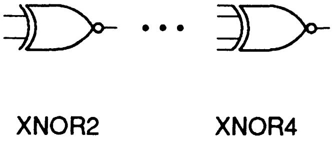
The TFF macro is a five-terminal T-type flip-flop.

TRST — Three-State Buffer — TRST



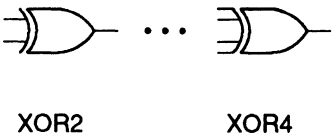
The TRST macro is a non-inverting three-state buffer.

XNOR — XNOR Gates — XNOR



The XNOR2 through XNOR4 macros are XNOR gates with two to four inputs.

XOR XOR Gates XOR

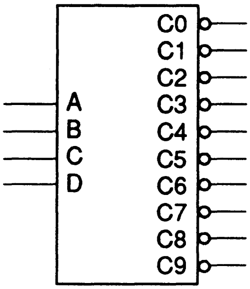


The XOR2 through XOR4 macros are XOR gates with two to four inputs.

7442 BCD-to-Decimal Decoder 7442

- Active-LOW outputs

Logic Symbol



Macrocell count:	10
Array inputs:	4
Product terms used:	10
Product terms allocated:	40

Functional Description

The 7442 macro decodes a 4-bit BCD input into a single active-LOW output. All outputs remain HIGH for invalid inputs.

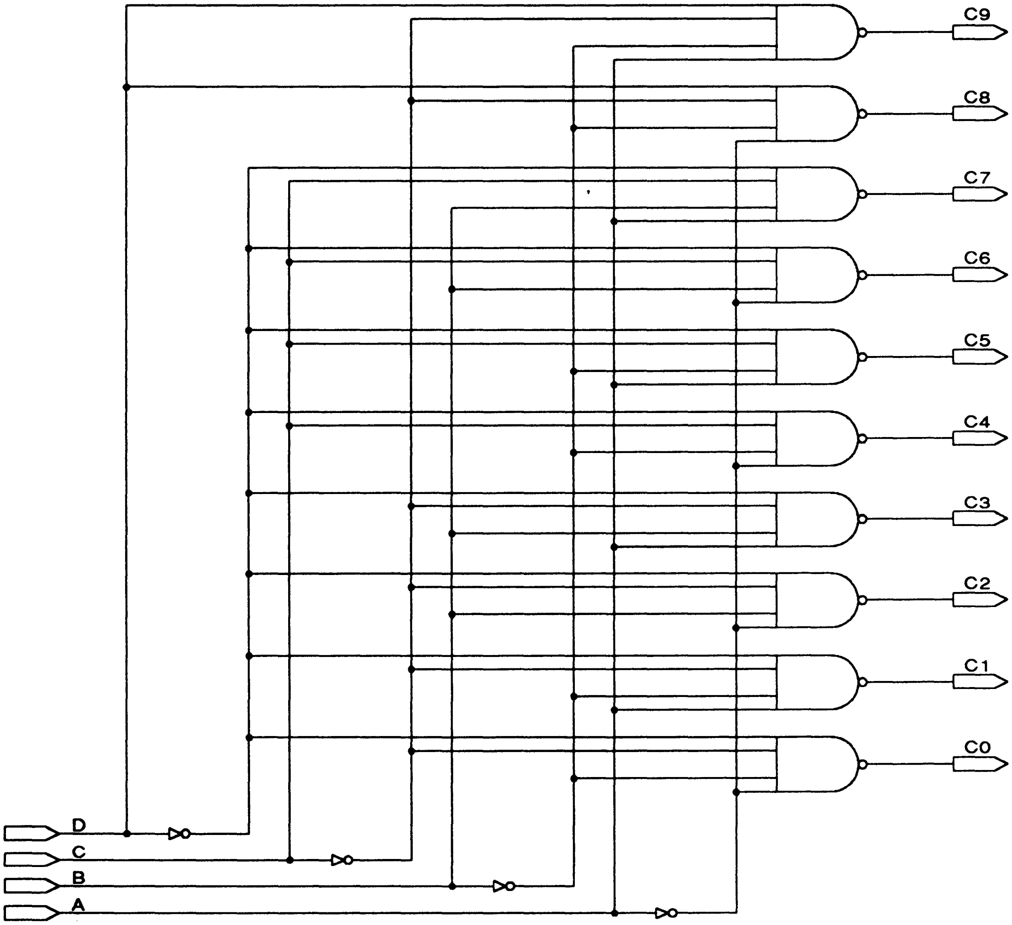
Sample PDS Equivalent

```
C9 = /(D * /C * /B * A)
C8 = /(D * /C * /B * /A)
C7 = /(/D * C * B * A)
C6 = /(/D * C * B * /A)
C5 = /(/D * C * /B * A)
C4 = /(/D * C * /B * /A)
C3 = /(/D * /C * B * A)
C2 = /(/D * /C * B * /A)
C1 = /(/D * /C * /B * A)
C0 = /(/D * /C * /B * /A)
```

Function Table

Value	BCD Inputs				Decimal Outputs									
	D	C	B	A	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L
10	H	L	H	L	H	H	H	H	H	H	H	H	H	H
11	H	L	H	H	H	H	H	H	H	H	H	H	H	H
12	H	H	L	L	H	H	H	H	H	H	H	H	H	H
13	H	H	L	H	H	H	H	H	H	H	H	H	H	H
14	H	H	H	L	H	H	H	H	H	H	H	H	H	H
15	H	H	H	H	H	H	H	H	H	H	H	H	H	H

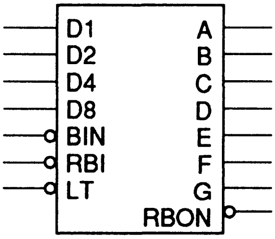
BCD-to-Decimal Decoder



7448 BCD-to-7-Segment Decoder 7448

- Lamp-test feature
- Leading- and trailing-zero blanking

Logic Symbol



Macrocell count:	8
Array inputs:	8
Product terms used:	32
Product terms allocated:	48

Functional Description

The 7448 macro decodes a 4-bit BCD input into a 7-segment-display format. To turn off all display segments, you can set the lamp-test input, LT, LOW. When the blanking input, BIN, is set LOW, all segments are turned off regardless of the BCD inputs. When the ripple-blanking input, RBI, and all BCD inputs are set LOW, and LT is set HIGH, all segments and the ripple-blanking outputs, RBO, are turned off. You can connect the RBO to the RBI of adjacent 7448 macros to turn off leading or trailing zeros.

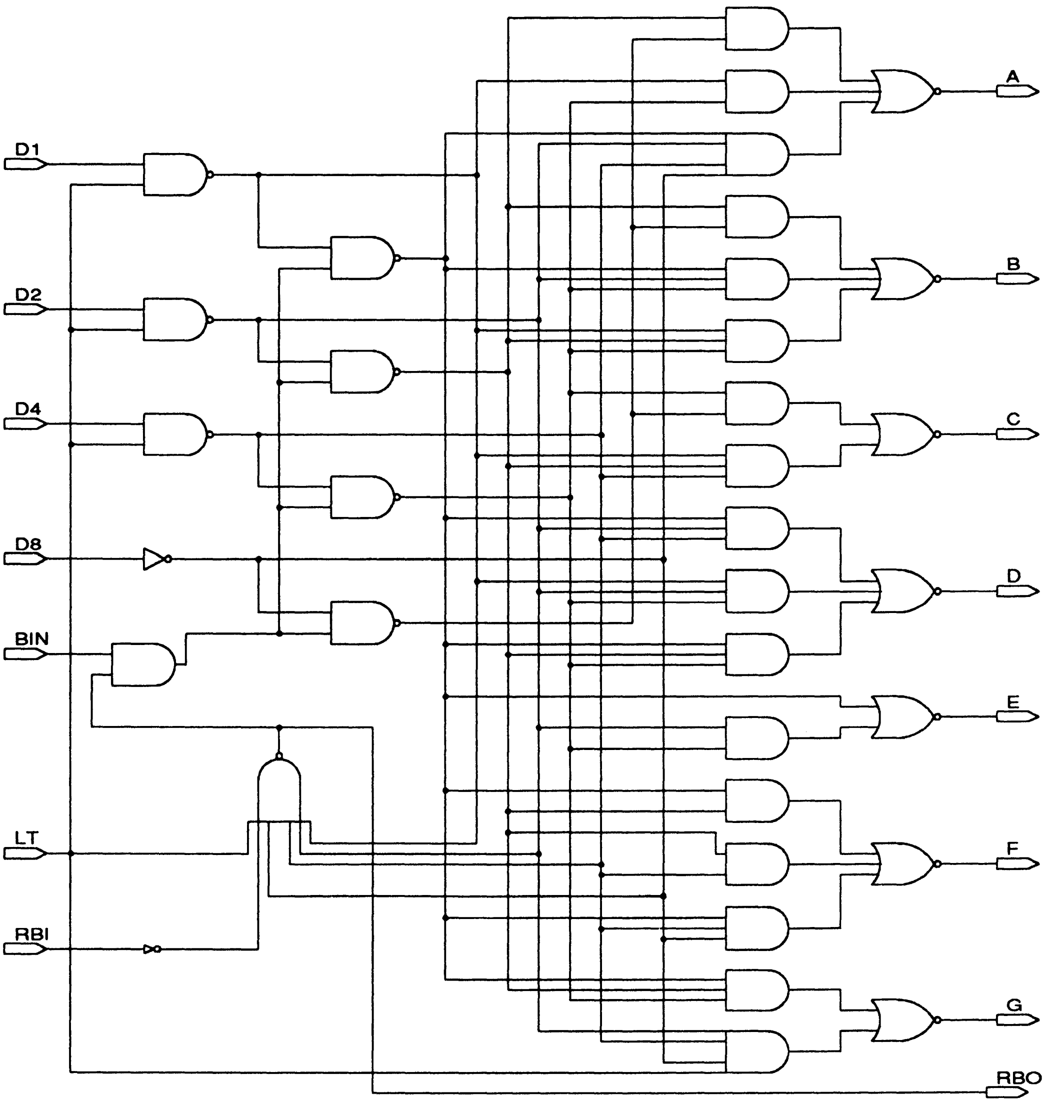
Sample PDS Equivalent

```
A = /(((/(/(D2 * LT) * (BIN * RBON)) * /(/D8
   * (BIN* RBON))) + (/(D1 * LT) * /(/(D4 * LT)
   * (BIN * RBON))) + (/(/(D1 * LT) * (BIN
   * RBON)) * /(D2 * LT) * /(D4 * LT) * /D8))
B = /(((/(/(D2 * LT) * (BIN * RBON)) * /(/D8
   * (BIN * RBON))) + (/(/(D1 * LT) * (BIN
   * RBON)) * /(D2 * LT) * /(/(D4 * LT) * (BIN
   * RBON))) + (/(D1 * LT) * /(/(D2 * LT) * (BIN
   * RBON)) * /(/(D4 * LT) * (BIN * RBON))))
C = /(((/(/(D4 * LT) * (BIN * RBON)) * /(/D8
   * (BIN * RBON))) + (/(D1 * LT) * /(/(D2
   * LT) * (BIN * RBON)) * /(D4 * LT)))
D = /(((/(/(D1 * LT) * (BIN * RBON)) * /(D2
   * LT) * /(D4 * LT)) + (/(D1 * LT) * /(D2
   * LT) * /(/(D4 * LT) * (BIN * RBON)))
   + (/(/(D1 * LT) * (BIN * RBON)) * /(/(D2
   * LT) * (BIN * RBON)) * /(/(D4 * LT)
   * (BIN * RBON))))
E = /(/(/(D1 * LT) * (BIN * RBON)) + (/(D2
   * LT) * /(/(D4 * LT) * (BIN * RBON))))
RBON = /(LT * /RBI * /D8 * /(D4 * LT)
   * /(D2 * LT) * /(D1 * LT)) F = /(((/(/(D1
   * LT) * (BIN * RBON)) * /(/(D2 * LT)
   * (BIN * RBON))) + (/(/(D2 * LT) * (BIN
   * RBON)) * /(D4 * LT)) + (/(/(D1 * LT)
   * (BIN * RBON)) * /(D4 * LT) * /D8))
G = /(((/(/(D1 * LT) * (BIN * RBON)) * /(/(D2
   * LT) * (BIN * RBON)) * /(/(D4 * LT)
   * (BIN * RBON))) + (/(D2 * LT) * /(D4 * LT)
   * /D8 * LT))
```

Function Table

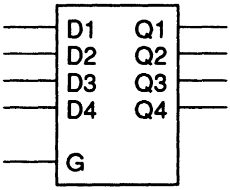
Inputs									Outputs						
Dec	LT	RBI	D8	D4	D2	D1	RBO	BIN	A	B	C	D	E	F	G
0	H	H	L	L	L	L	H	H	H	H	H	H	H	H	L
1	H	X	L	L	L	H	H	H	L	H	H	L	L	L	L
2	H	X	L	L	H	L	H	H	H	H	L	H	H	L	H
3	H	X	L	L	H	H	H	H	H	H	H	H	L	L	H
4	H	X	L	H	L	L	H	H	L	H	H	L	L	H	H
5	H	X	L	H	L	H	H	H	H	L	H	H	L	H	H
6	H	X	L	H	H	L	H	H	L	L	H	H	H	H	H
7	H	X	L	H	H	H	H	H	H	H	H	L	L	L	L
8	H	X	H	L	L	L	H	H	H	H	H	H	H	H	H
9	H	X	H	L	L	H	H	H	H	H	H	L	L	H	H
10	H	X	H	L	H	L	H	H	L	L	L	H	H	L	H
11	H	X	H	L	H	H	H	H	L	L	H	H	L	L	H
12	H	X	H	H	L	L	H	H	L	H	L	L	L	H	H
13	H	X	H	H	L	H	H	H	H	L	L	H	L	H	H
14	H	X	H	H	H	L	H	H	L	L	L	H	H	H	H
15	H	X	H	H	H	H	H	H	L	L	L	L	L	L	L
BIN	X	X	X	X	X	X	L	X	L	L	L	L	L	L	L
RBI	H	L	L	L	L	L	X	L	L	L	L	L	L	L	L
LT	L	X	X	X	X	X	H	H	H	H	H	H	H	H	H

BCD-to-7-Segment Decoder



- Enable input

Logic Symbol



Macrocell count:	4
Array inputs:	9
Product terms used:	8
Product terms allocated:	16

Functional Description

The 7477 macro is a 4-bit latch. The Q1 – Q4 outputs follow the D1 – D4 input data when the G latch enable is set HIGH. When G is set LOW, the outputs latch the input data, and further activity at D1 – D4 is ignored.

Note:
The TTL version contains two 4-bit latches.

Sample PDS Equivalent

```
Q1 = ((Q1 * VCC * /G) + (VCC * GND) + (VCC * G * D1)
  + (D1 * VCC * Q1))
Q2 = ((Q2 * VCC * /G) + (VCC * GND) + (VCC * G * D2)
  + (D2 * VCC * Q2))
Q3 = ((Q3 * VCC * /G) + (VCC * GND) + (VCC * G * D3)
  + (D3 * VCC * Q3))
Q4 = ((Q4 * VCC * /G) + (VCC * GND) + (VCC * G * D4)
  + (D4 * VCC * Q4))
```

Function Table (for 1 bit)

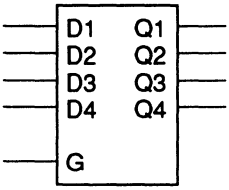
Inputs		Outputs
G	D	Q
H	H	H
H	H	H
L	X	Qo

* Qo = previous state of Q

7477 4-Bit Latch 7477

- Enable input

Logic Symbol



Macrocell count:	4
Array inputs:	9
Product terms used:	8
Product terms allocated:	16

Functional Description

The 7477 macro is a 4-bit latch. The Q1 – Q4 outputs follow the D1 – D4 input data when the G latch enable is set HIGH. When G is set LOW, the outputs latch the input data, and further activity at D1 – D4 is ignored.

Note:
The TTL version contains two 4-bit latches.

Sample PDS Equivalent

Q1 = ((Q1 * VCC * /G) + (VCC * GND) + (VCC * G * D1)
+ (D1 * VCC * Q1))
Q2 = ((Q2 * VCC * /G) + (VCC * GND) + (VCC * G * D2)
+ (D2 * VCC * Q2))
Q3 = ((Q3 * VCC * /G) + (VCC * GND) + (VCC * G * D3)
+ (D3 * VCC * Q3))
Q4 = ((Q4 * VCC * /G) + (VCC * GND) + (VCC * G * D4)
+ (D4 * VCC * Q4))

Function Table (for 1 bit)

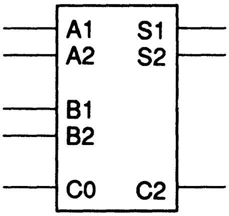
Inputs		Outputs
G	D	Q
H	H	H
H	H	H
L	X	Qo

* Qo = previous state of Q

7482 2-Bit Full Adder 7482

- Carry output and input

Logic Symbol



Macrocell count:	3
Array inputs:	5
Product terms used:	23
Product terms allocated:	24

Functional Description

The 7482 macro adds two 2-bit numbers using carry look-ahead logic to generate the carry out, C2.

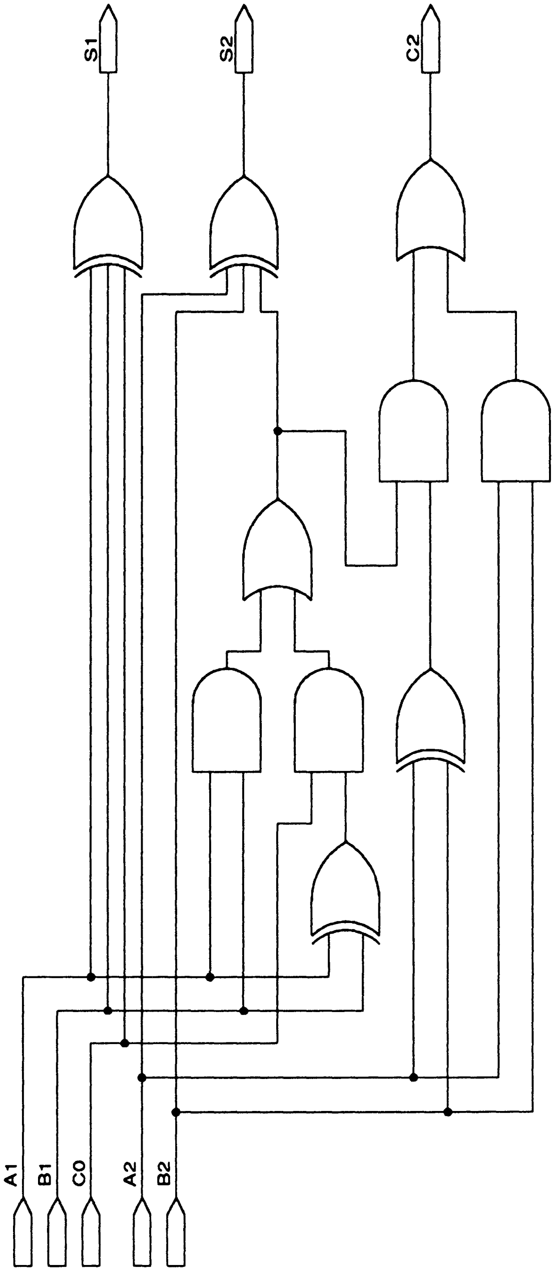
Sample PDS Equivalent

```
S1 = (A1 :+: B1 :+: C0)
S2 = (A2 :+: B2 :+: ((A1 * B1)
   + (C0 * (A1 :+: B1))))
C2 = ((((A1 * B1) + (C0 * (A1 :+: B1)))
   * (A2 :+: B2)) + (A2 * B2))
```

Function Table

Inputs				Outputs					
				C0=L			C0=H		
A2	A1	B2	B1	C2	S2	S1	C2	S2	S1
L	L	L	L	L	L	L	L	L	H
L	L	L	H	L	L	H	L	H	L
L	L	H	L	L	H	L	L	H	H
L	L	H	H	L	H	H	H	L	L
L	H	L	L	L	L	H	L	H	L
L	H	L	H	L	H	L	L	H	H
L	H	H	L	L	H	H	H	L	L
L	H	H	H	H	L	L	H	L	H
H	L	L	L	L	H	L	L	H	H
H	L	L	H	L	H	H	H	L	L
H	L	H	L	H	L	L	H	L	H
H	L	H	H	H	L	H	H	H	L
H	H	L	L	L	H	H	H	L	L
H	H	L	H	H	L	L	H	L	H
H	H	H	L	H	L	H	H	H	L
H	H	H	H	H	H	L	H	H	H

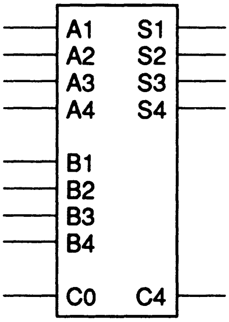
2-Bit Full Adder



7483 4-Bit Full Adder 7483

- Carry output and input

Logic Symbol



Macrocell count:	6
Array inputs:	10
Product terms used:	46
Product terms allocated:	48

Functional Description

The 7483 macro adds two 4-bit numbers using carry look-ahead logic to generate the internal 2-bit carry, C2, and the final carry out, C4.

Note: The first 2-bit addition is performed as shown in the 7482 data sheet.

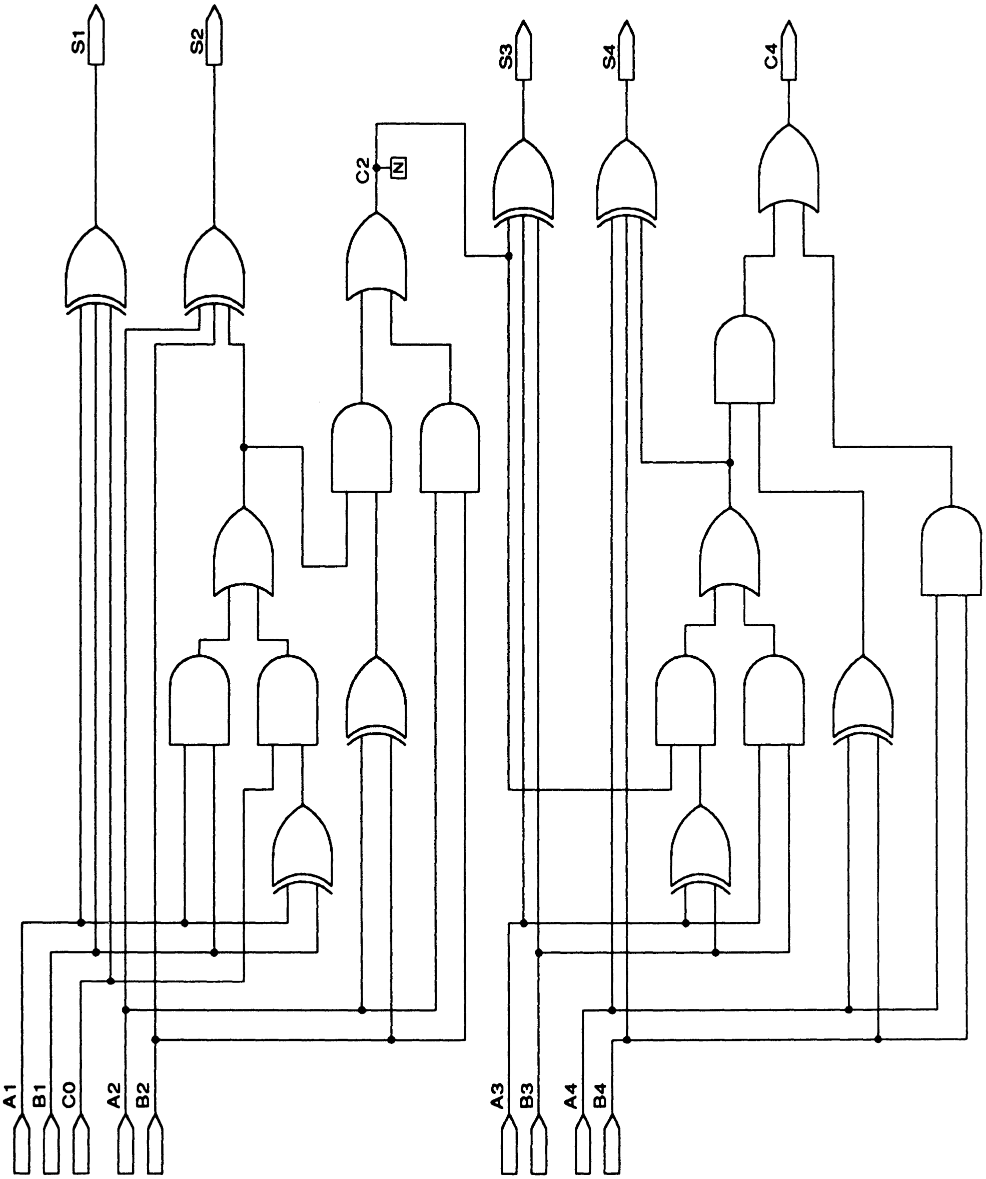
Sample PDS Equivalent

```
S1 = (A1 :+: B1 :+: C0)
S2 = (A2 :+: B2 :+: ((A1 * B1)
   + (C0 * (A1 :+: B1))))
C2 = ((((A1 * B1)
   + (C0 * (A1 :+: B1))) * (A2 :+: B2))
   + (A2 * B2))
S3 = (C2 :+: A3 :+: B3)
S4 = (A4 :+: B4 :+: ((C2 * (A3 :+: B3))
   + (A3 * B3)))
C4 = ((((C2 * (A3 :+: B3))
   + (A3 * B3)) * (A4 :+: B4)) + (A4 * B4))
```

Function Table

Inputs				Outputs (second 2-bit stage*)					
				C2=L			C2=H		
A4	**A3**	**B4**	**B3**	**C4**	**S4**	**S3**	**C4**	**S4**	**S3**
L	L	L	L	L	L	L	L	L	H
L	L	L	H	L	L	H	L	H	L
L	L	H	L	L	H	L	L	H	H
L	L	H	H	L	H	H	H	L	L
L	H	L	L	L	L	H	L	H	L
L	H	L	H	L	H	L	L	H	H
L	H	H	L	L	H	H	H	L	L
L	H	H	H	H	L	L	H	L	H
H	L	L	L	L	H	L	L	H	H
H	L	L	H	L	H	H	H	L	L
H	L	H	L	H	L	L	H	L	H
H	L	H	H	H	L	H	H	H	L
H	H	L	L	L	H	H	H	L	L
H	H	L	H	H	L	L	H	L	H
H	H	H	L	H	L	H	H	H	L
H	H	H	H	H	H	L	H	H	H

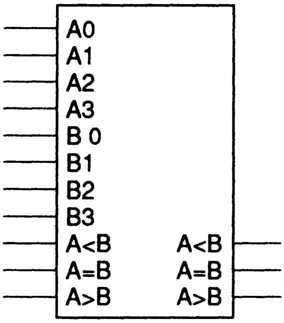
4-Bit Full Adder



7485 4-Bit Magnitude Comparator 7485

- Equal to, greater than, and less than three cascadable compare inputs

Logic Symbol



Macrocell count:	7
Array inputs:	15
Product terms used:	25
Product terms allocated:	28

Functional Description

The 7485 macro compares two 4-bit numbers, then activates one of the three outputs: A=B, A>B, or A<B. For cascaded 7485 macros, the least-significant stage must have the A=B input HIGH and the A>B and A<B inputs LOW. For intermediate stages, connect the outputs of a previous stage to the A=B, A>B, and A<B inputs.

Note:
The TTL version functions differently when more than one cascading input is HIGH or when all cascaded inputs are LOW.

Sample PDS Equivalent

```
STG32 = (/(A3 :+: B3) * /(A2 :+: B2))
AEQBO = (AEQBI * STG32 * STG10)
STGA2 = ((AGTBI + (A3 * /ALTBI * /B3))
   + (A2 * /(ALTBI + (B3 * /AGTBI * /A3))
   * /B2))
STGB2 = ((ALTBI + (B3 * /AGTBI * /A3))
   + (B2 * /(AGTBI + (A3 * /ALTBI * /B3))
   * /A2))
STG10 = (/(A1 :+: B1) * /(A0 :+: B0))
AGTBO = ((STGA2 + (A1 * /STGB2
   * /B1)) + (A0 * /(STGB2 + (B1
   * /STGA2 * /A1)) * /B0))
ALTBO = ((STGB2 + (B1 * /STGA2
   * /A1)) + (B0 * /(STGA2 + (A1
   * /STGB2 * /B1)) * /A0))
```

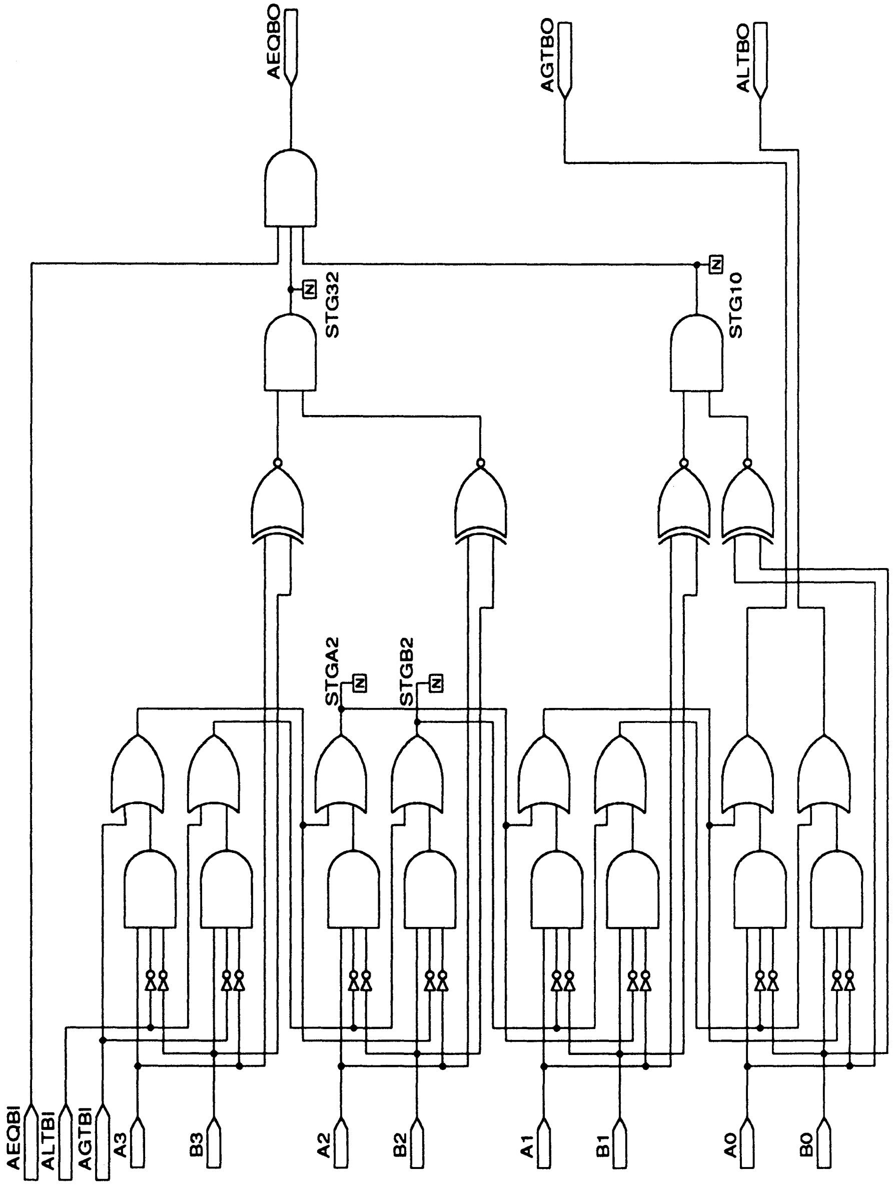
Function Table

Comparing Inputs				Comparing Inputs			Outputs		
A3,B3	A2,B2	A1,B1	A0,B0	A>B	A<B	A=B	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	H	L	L
A3<B3	X	X	X	X	X	X	L	H	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H

Invalid Conditions

A3=B3	A2=B2	A1=B1	A0=B0	X	X	H	X	X	X
A3=B3	A2=B2	A1=B1	A0=B0	H	H	L	X	X	X
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	X	X	X

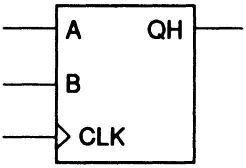
4-Bit Magnitude Comparator



7491 8-Bit Shift Register 7491

- Two serial-input lines

Logic Symbol



Macrocell count:	8
Array inputs:	9
Product terms used:	8
Product terms allocated:	32

Functional Description

The 7491 macro is an 8-bit serial-in serial-out shift register. The serial-input stream is the result of logically ANDing inputs A and B.

Sample PDS Equivalent

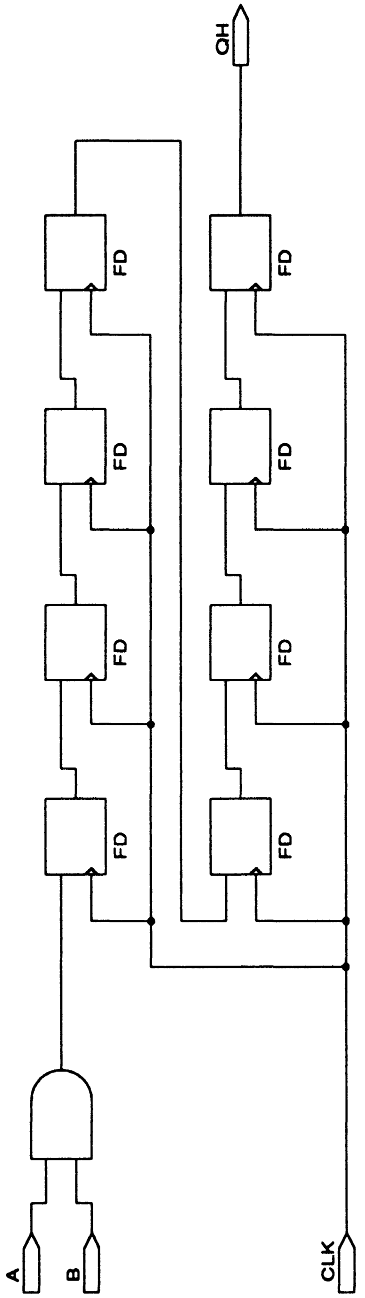
```
X2_D = (A * B)
X2_D.clkf = CLK
X3_D = X2_D
X3_D.clkf = CLK
X4_D = X3_D
X4_D.clkf = CLK
X4_Q = X4_D
X4_Q.clkf = CLK
X6_D = X4_Q
X6_D.clkf = CLK
X7_D = X6_D
X7_D.clkf = CLK
X8_D = X7_D
X8_D.clkf = CLK
QH = X8_D
QH.clkf = CLK
```

Function Table

Inputs			Outputs
CLK	A	B	Q
↑	H	H	H
↑	L	X	L
↑	X	L	L
L	X	X	Q0

• Q = (AT TIME t+8)
Qo = previous state of Q

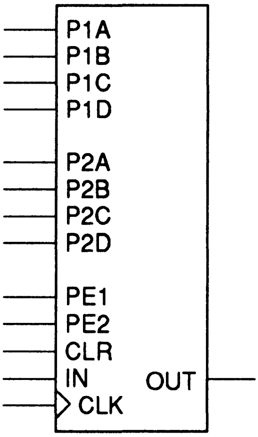
8-Bit Shift Register



7494 4-Bit Shift Register 7494

- Synchronous preset
- Dual preset inputs
- Asynchronous reset

Logic Symbol



Macrocell count:	4
Array inputs:	15
Product terms used:	12
Product terms allocated:	16

Functional Description

The 7494 macro is a 4-bit serial-in serial-out shift register with asynchronous clear and synchronous preset logic.

You can select either 4-bit preset value by setting one preset-enable input, PE1 or PE2, HIGH. If you set both HIGH, then the preset value is the OR of the two sets of data inputs. The preset function overrides the shift operation.

Note: The TTL version has asynchronous preset logic.

Sample PDS Equivalent

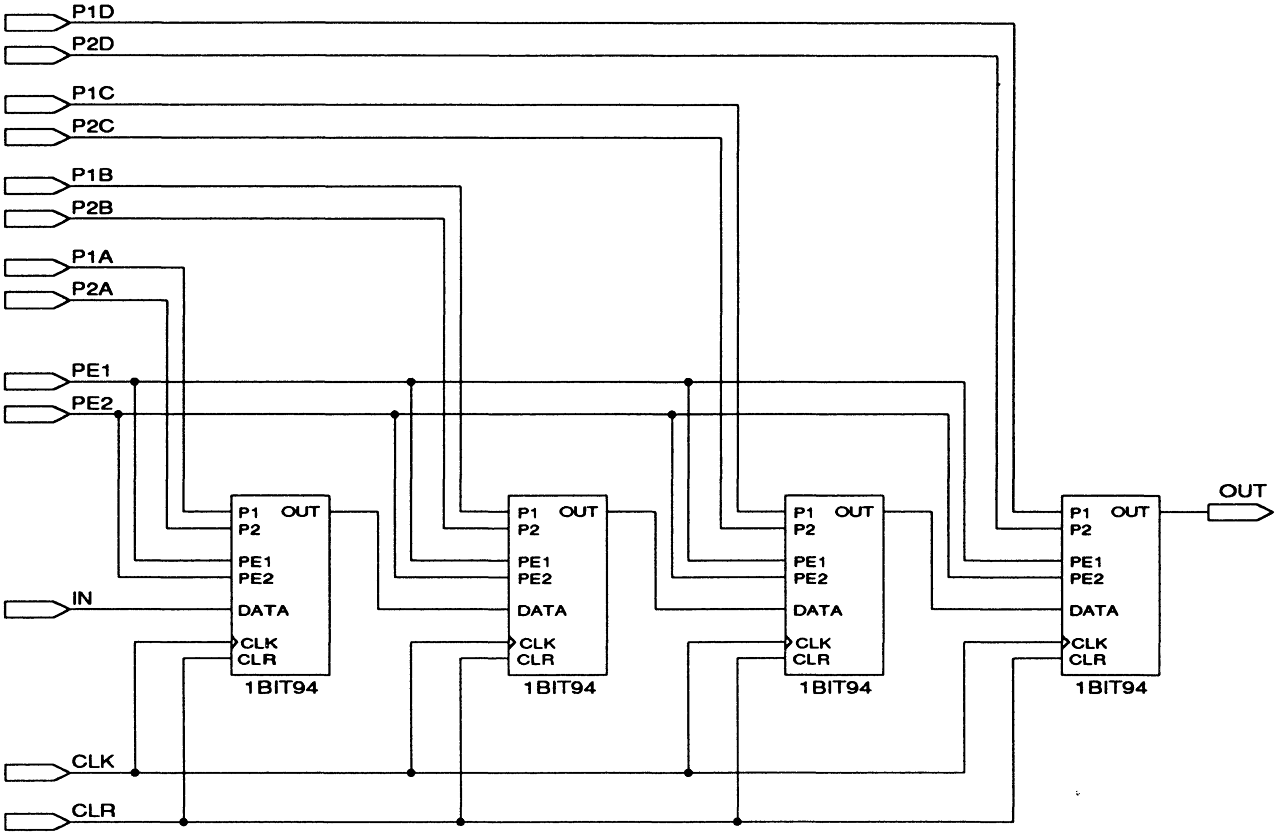
```
M1_OUT = ((P1A * PE1) + (P2A * PE2)
  + (/PE2 * /PE1 * IN))
M1_OUT.clkf = CLK
M1_OUT.rstf = CLR
M2_OUT = ((P1B * PE1) + (P2B * PE2)
  + (/PE2 * /PE1 * M1_OUT))
M2_OUT.clkf = CLK
M2_OUT.rstf = CLR
M3_OUT = ((P1C * PE1) + (P2C * PE2)
  + (/PE2 * /PE1 * M2_OUT))
M3_OUT.clkf = CLK
M3_OUT.rstf = CLR
OUT = ((P1D * PE1) + (P2D * PE2)
  + (/PE2 * /PE1 * M3_OUT))
OUT.clkf = CLK
OUT.rstf = CLR
```

Function Table

Inputs				Outputs
PE1	P1A	PE2	P2A	Resulting Preset Value
L	X	L	X	L
L	X	X	L	L
X	L	L	X	L
X	L	X	L	L
H	H	X	X	H
H	L	L	X	L
H	L	H	L	L
H	L	H	H	H
X	X	H	H	H
L	X	H	L	L
H	L	H	L	L
H	H	H	L	H

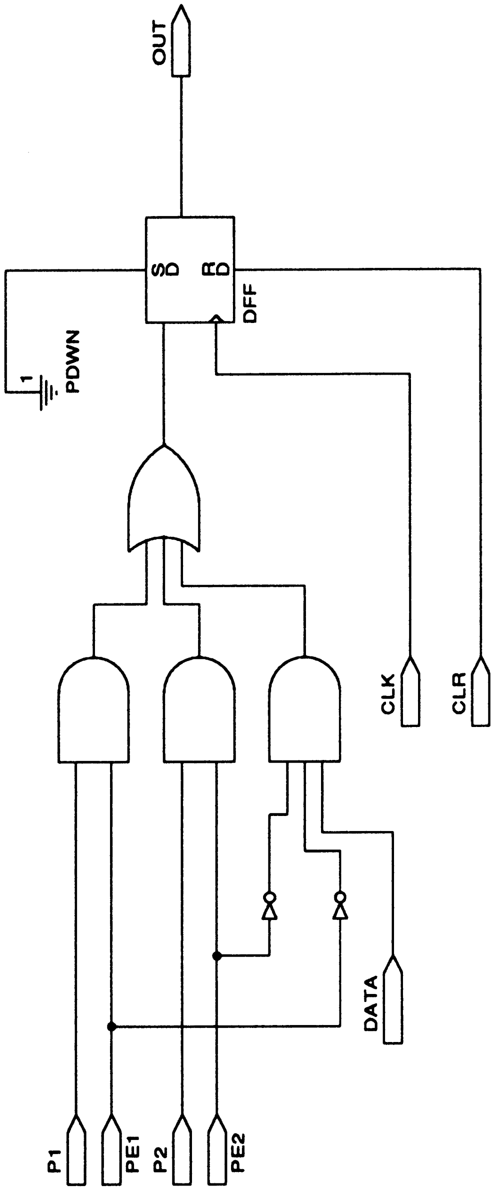
7494

4-Bit Shift Register



7494

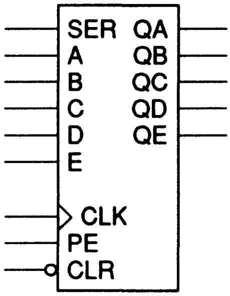
1Bit of a 7494 Shift Register



7496 5-Bit Shift Register 7496

- Synchronous preset
- Asynchronous reset
- Parallel-to-serial converter
- Serial-to-parallel converter

Logic Symbol



Macrocell count:	5
Array inputs:	12
Product terms used:	10
Product terms allocated:	20

Functional Description

The 7496 macro is a 5-bit shift register that offers access to each flip-flop's input and output. The 5-bit value at inputs A – E is preloaded into the shift register on the rising-edge of CLK when PE is HIGH. The preset function overrides the shift operation.

Note:
The TTL version has asynchronous preset logic.

Sample PDS Equivalent

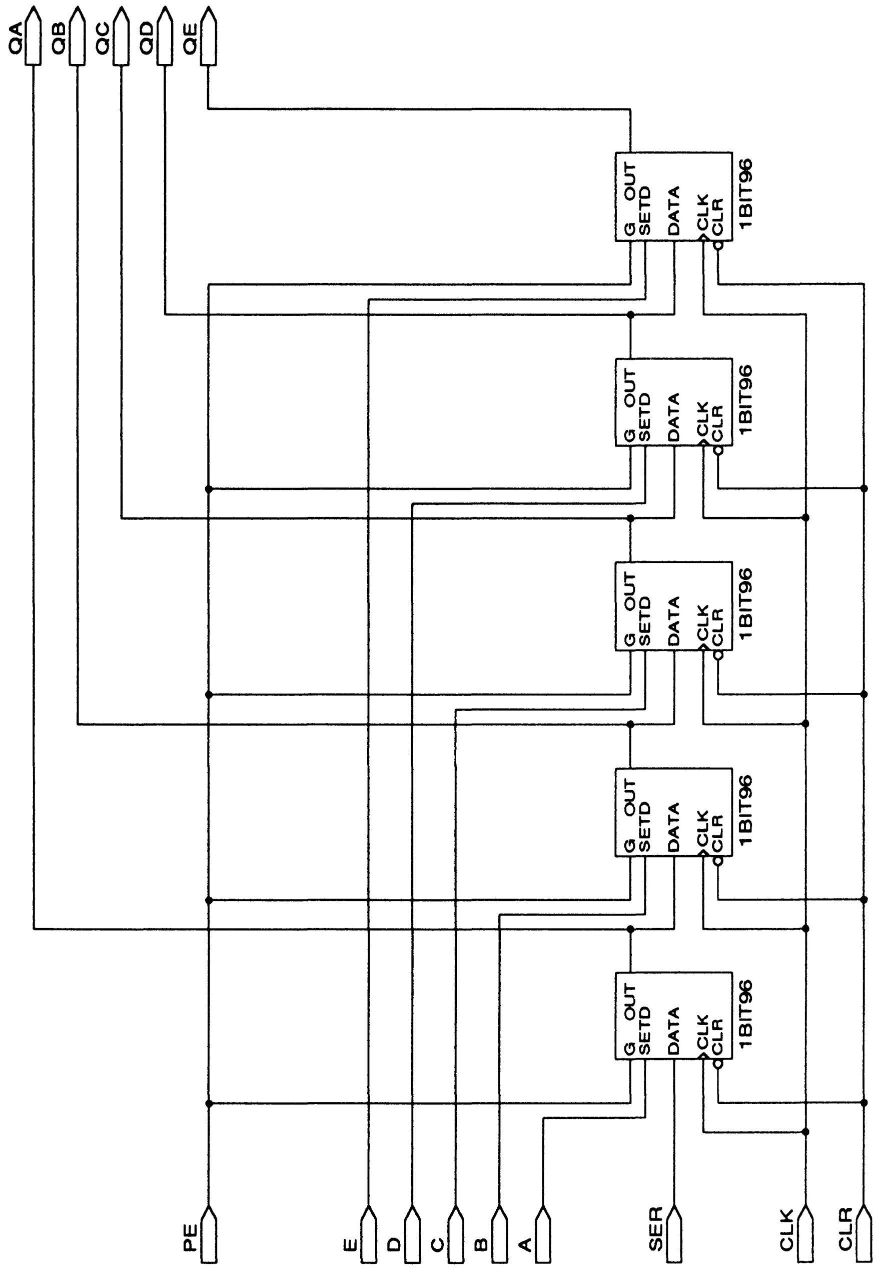
```
QA = ((A * PE) + (/PE * SER))
QA.clkf = CLK
QA.rstf = /CLR
QB = ((B * PE) + (/PE * QA))
QB.clkf = CLK
QB.rstf = /CLR
QC = ((C * PE) + (/PE * QB))
QC.clkf = CLK
QC.rstf = /CLR
QD = ((D * PE) + (/PE * QC))
QD.clkf = CLK
QD.rstf = /CLR
QE = ((E * PE) + (/PE * QD))
QE.clkf = CLK
QE.rstf = /CLR
```

Function Table

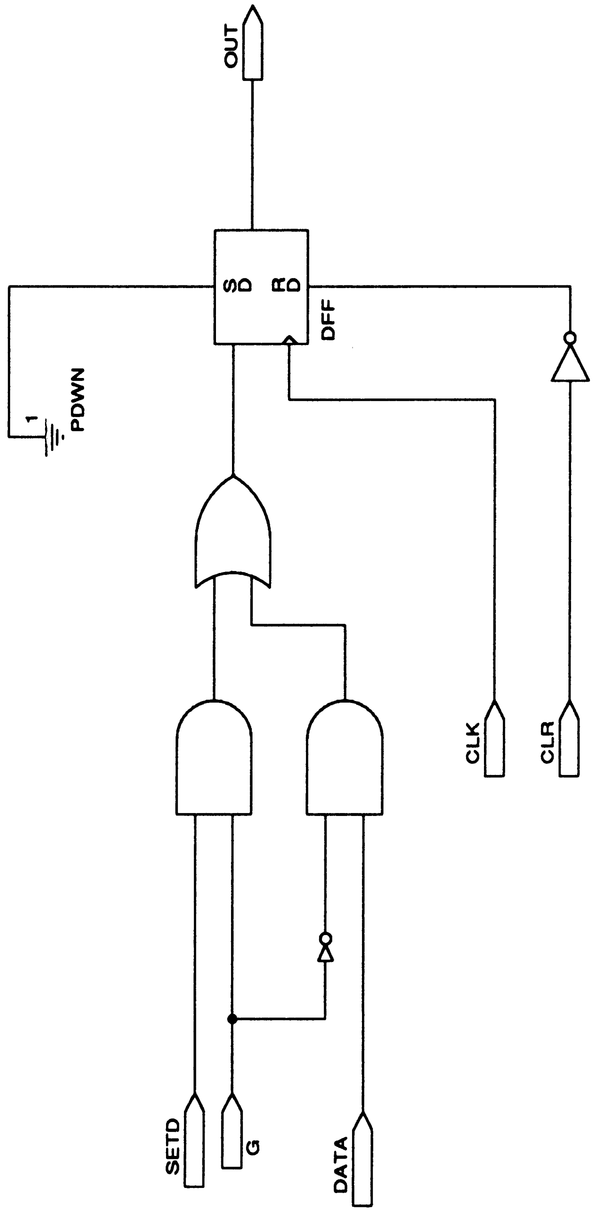
Inputs									Outputs				
CLR	PE	A	B	C	D	E	CLK	SER	QA	QB	QC	QD	QE
L	L	X	X	X	X	X	X	X	L	L	L	L	L
H	H	H	H	H	H	H	↑	X	H	H	H	H	H
H	H	L	L	H	H	L	↑	X	L	L	H	H	L
H	H	L	L	L	L	L	L	X	QA0	QB0	QC0	QD0	QE0
H	L	X	X	X	X	X	L	X	QA0	QB0	QC0	QD0	QE0
H	L	X	X	X	X	X	↑	L	L	QA0	QB0	QC0	QD0
H	L	X	X	X	X	X	↑	H	H	QA0	QBO	QC0	QD0

* QAo to QEo = previous state of registers QA to QE

7496 5-Bit Shift Register 7496



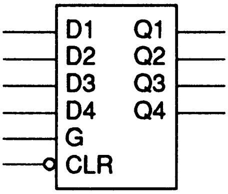
1BIT96 1Bit of a 7496 Shift Register 1BIT96



74116 4-Bit Latch w/ Clear 74116

- Asynchronous reset

Logic Symbol



Macrocell count:	4
Array inputs:	10
Product terms used:	8
Product terms allocated:	16

Functional Description

The 74116 macro is a 4-bit latch with an asynchronous reset.

Note: The TTL version contains two 4-bit latches.

Sample PDS Equivalent

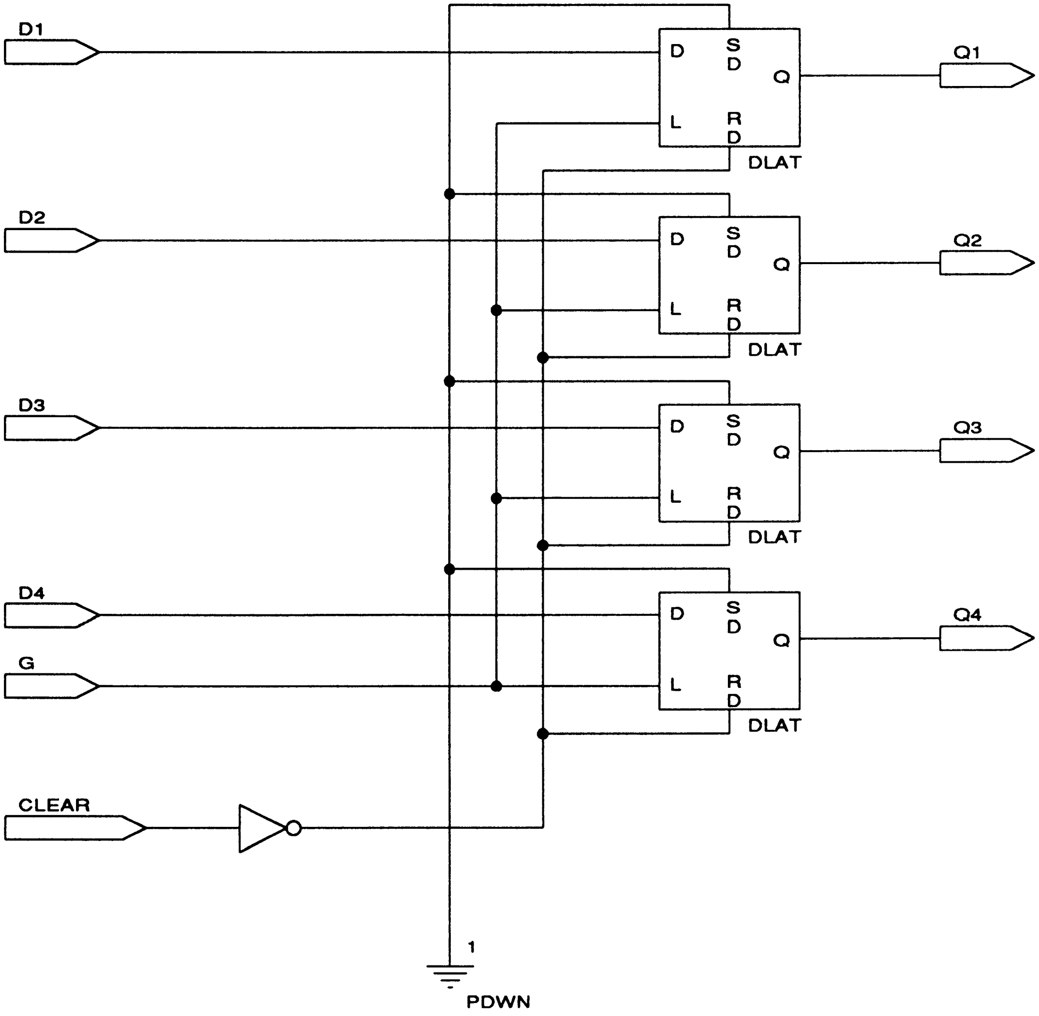
```
Q1 = ((Q1 * CLEAR * /G) + (CLEAR * GND)
   + (CLEAR * G * D1) + (D1 * CLEAR * Q1))
Q2 = ((Q2 * CLEAR * /G) + (CLEAR * GND)
   + (CLEAR * G * D2) + (D2 * CLEAR * Q2))
Q3 = ((Q3 * CLEAR * /G) + (CLEAR * GND)
   + (CLEAR * G * D3) + (D3 * CLEAR * Q3))
Q4 = ((Q4 * CLEAR * /G) + (CLEAR * GND)
   + (CLEAR * G * D4) + (D4 * CLEAR * Q4))
```

Function Table

Inputs			Outputs
CLR	G	D	Q
H	L	X	L
L	H	H	H
L	H	L	L
L	L	X	Qo*

* Qo = previous state of Q

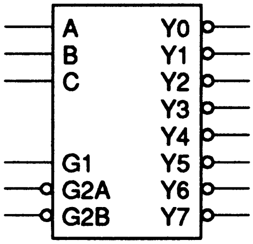
74116 4-Bit Latch w/ Clear 74116



74138 3-to-8 Line Decoder 74138

- Active LOW outputs
- Three enable inputs

Logic Symbol



Macrocell count:	8
Array inputs:	6
Product terms used:	8
Product terms allocated:	32

Functional Description

The 74138 macro decodes a 3-bit binary input into a single active-LOW output.

You can cascade these macros to implement a decoder with up to 24 outputs via the three enable inputs, G1, G2A, and G2B.

Sample PDS Equivalent

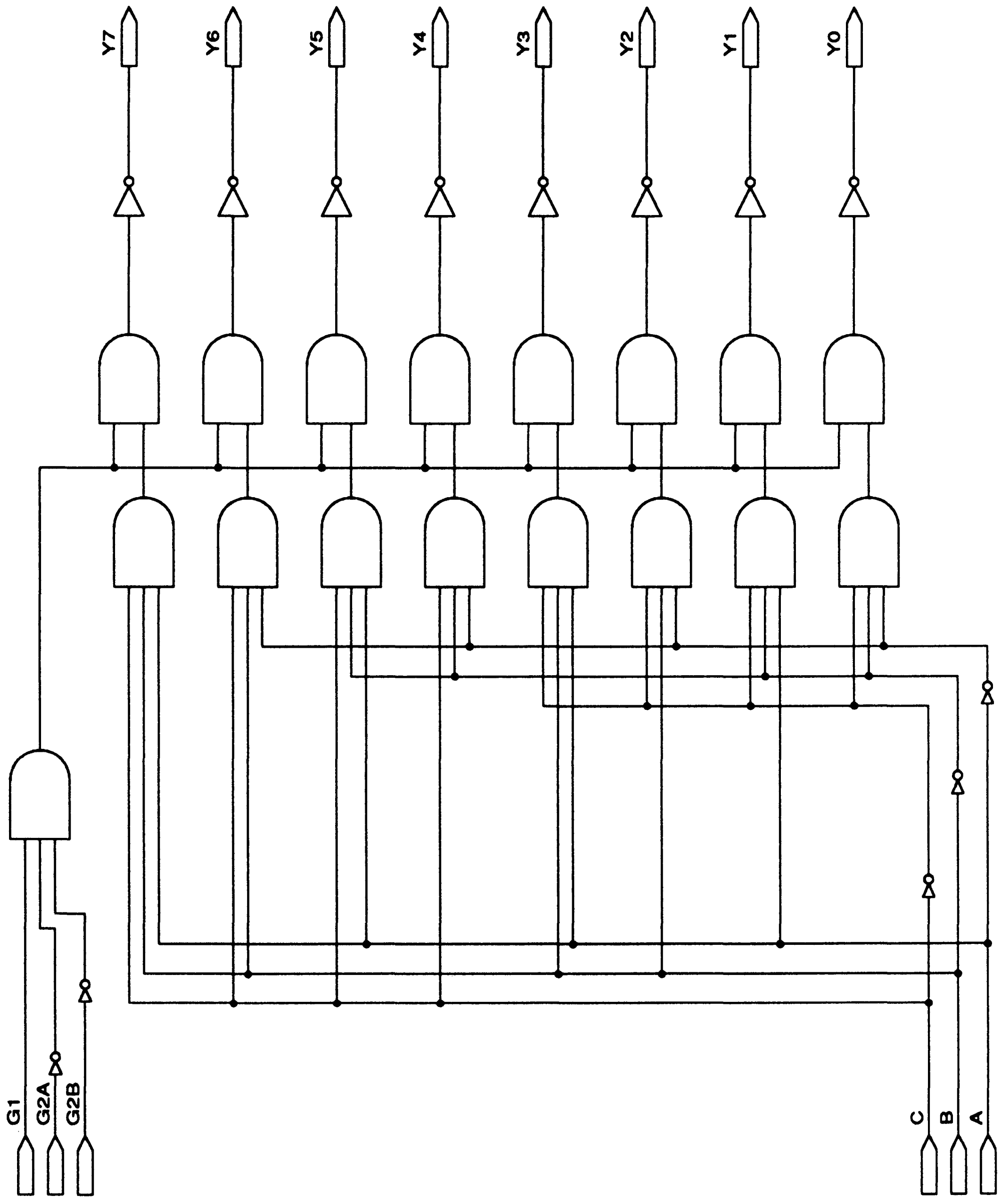
```
Y7 = ((G1 * /G2A * /G2B) * (C * B * A))
Y6 = ((G1 * /G2A * /G2B) * (C * B * /A))
Y5 = ((G1 * /G2A * /G2B) * (C * /B * A))
Y4 = ((G1 * /G2A * /G2B) * (C * /B * /A))
Y3 = ((G1 * /G2A * /G2B) * (/C * B * A))
Y2 = ((G1 * /G2A * /G2B) * (/C * B * /A))
Y1 = ((G1 * /G2A * /G2B) * (/C * /B * A))
Y0 = ((G1 * /G2A * /G2B) * (/C * /B * /A))
```

Function Table

Inputs					Outputs							
Enable		Select										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

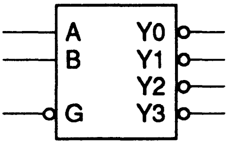
* G2 = G2A + G2B

3-to-8 Line Decoder



- Enable input

Logic Symbol



Macrocell count:	4
Array, inputs:	3
Product terms used:	4
Product terms allocated:	16

Functional Description

The 74139 macro decodes one of four active-LOW outputs depending on two data inputs. The active-LOW enable input, G, can be used as an input when decoding more output lines.

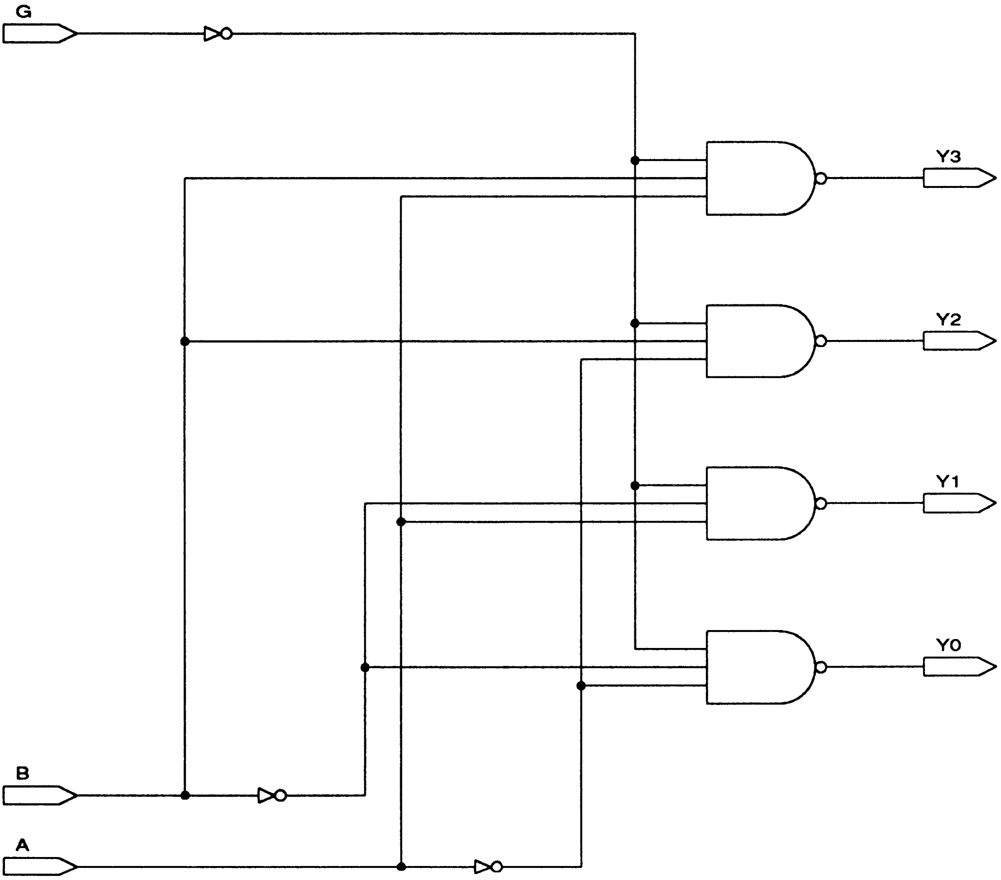
PDS Equivalent

Y3 = /(/G * B * A)
Y2 = /(/G * B * /A)
Y1 = /(/G * /B * A)
Y0 = /(/G * /B * /A)

Function Table

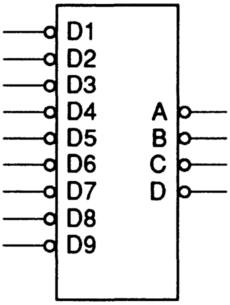
Inputs			Outputs			
Enable	Select					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

2-to-4 Line Decoder



74147 10-to-4 Priority Line Encoder 74147

Logic Symbol



Macrocell count:	4
Array inputs:	9
Product terms used:	13
Product terms allocated:	20

Functional Description

The 74147 macro generates a 4-bit BCD-output code that represents the highest-order-LOW data input. Priority encoding of the inputs ensures that only the highest-order data-input line is encoded.

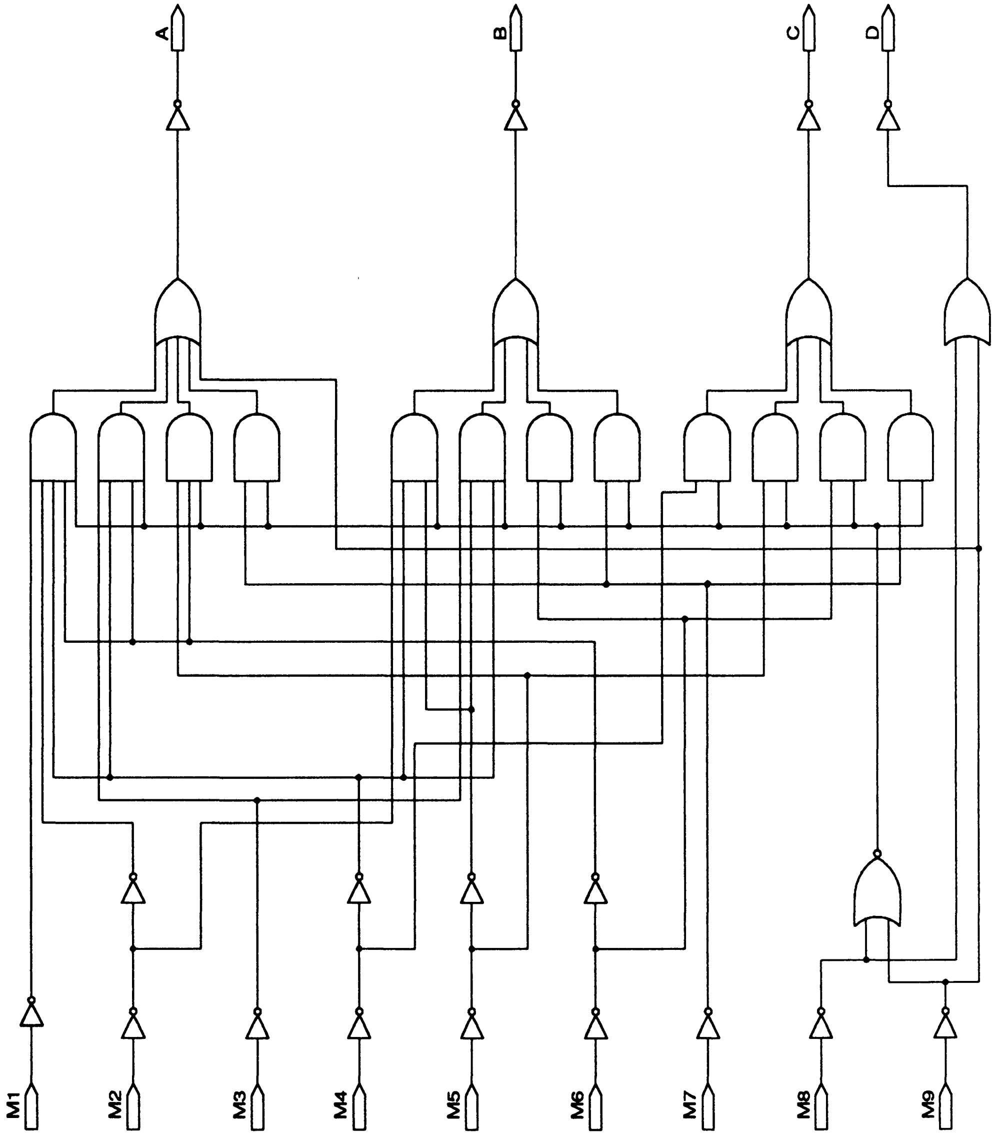
Sample PDS Equivalent

```
A = ((/D1 * D2 * D4 * D6 * /(/D8 + /D9))
    + (/D3 * D4 * D6 * /(/D8 + /D9)) + (/D5
    * D6 * /(/D8 + /D9)) + (/D7 * /(/D8
    + /D9)) + /D9)
B = ((/D2 * D4 * D5 * /(/D8 + /D9)) + (/D3
   * D5 * D4 * /(/D8 + /D9)) + (/D6 * /(/D8
    + /D9)) + (/D7 * /(/D8 + /D9)))
C = ((/D4 * /(/D8 + /D9)) + (/D5 * /(/D8
    + /D9)) + (/D6 * /(/DM8 + /D9)) + (/DM7
    * /(/D8 + /D9)))
D = (/D8 + /D9)
```

Function Table

Inputs									Outputs			
D1	D2	D3	D4	D5	D6	D7	D8	D9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

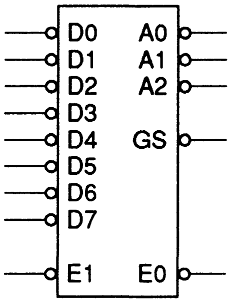
74147 10-to-4 Priority Line Encoder 74147



74148 8-to-3 Priority Line Encoder 74148

- Enable input and output for cascading

Logic Symbol



Macrocell count:	5
Array inputs:	10
Product terms used:	12
Product terms allocated:	20

Functional Description

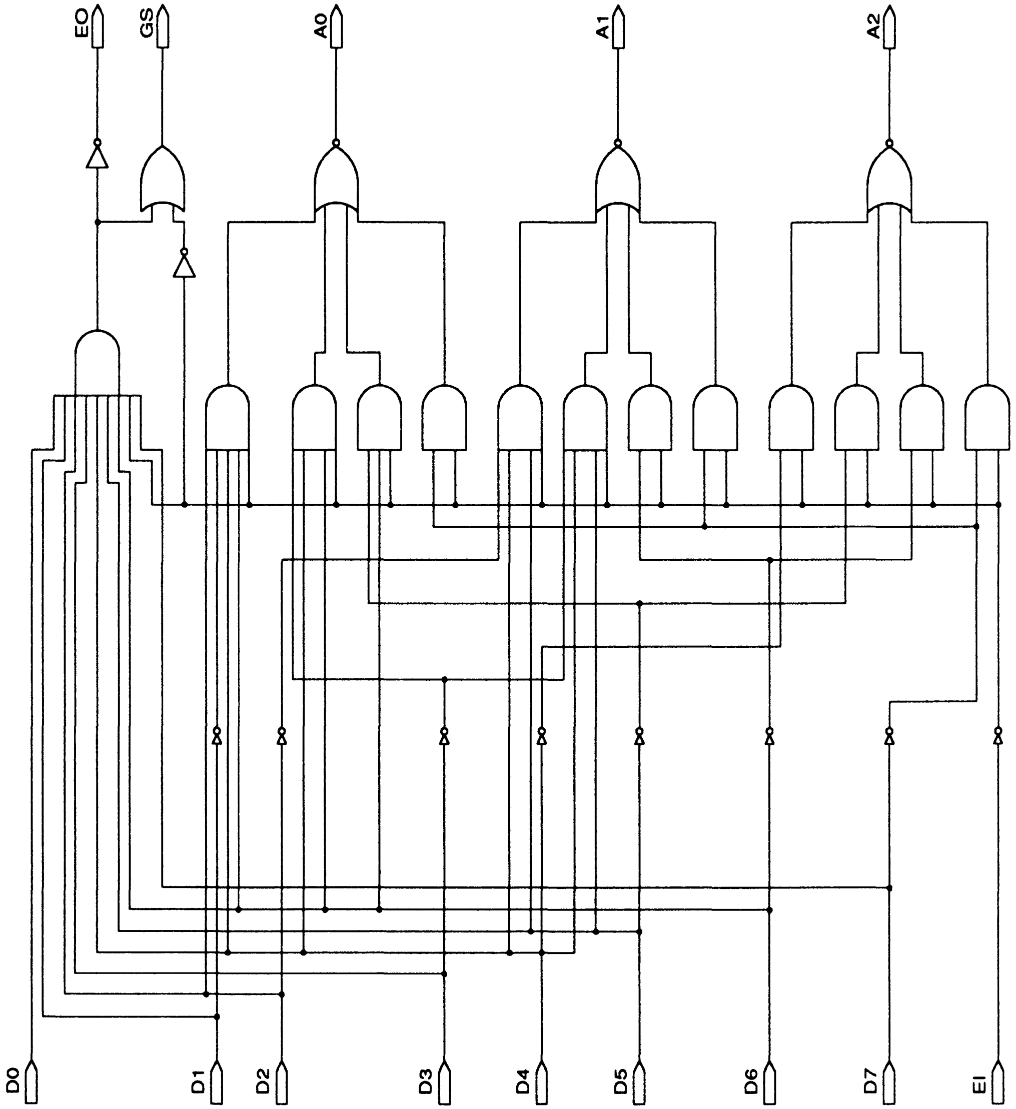
The 74148 macro generates a 3-bit binary output code that represents the highest-order-LOW data input. You can use the input enable, EI, and output enable, EO, to expand priority encoding.

PDS Equivalent

EO = (D0 * D1 * D2 * D3 * D4 * D5
  * D6 * /EI * D7)
GS = (EO + EI)
A0 = /((D2 * /D1 * D4 * D6 * /EI) + (/D3
  * D4 * D6 * /EI) + (/D5 * D6 * /EI)
  + (/D7 * /EI))
A1 = /((/D2 * D4 * D5 * /EI) + (/D3 * D4
  * D5 * /EI) + (/D6 * /EI) + (/D7 * /EI))
A2 = /((/D4 * /EI) + (/D5 * /EI) + (/D6
  * /EI) + (/D7 * /EI))

Function Table

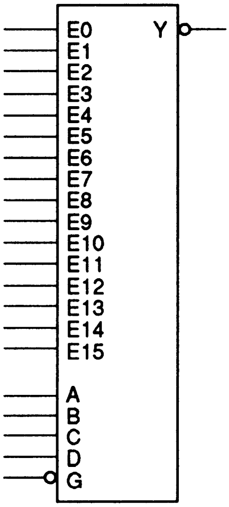
Inputs									Outputs				
E1	D0	D1	D2	D3	D4	D5	D6	D7	A2	A1	A0	GS	E1
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H


EO
GS
A0
A1
A2
D0
D1
D2
D3
D4
D5
D6
D7
EI

74150 16-to-1 Multiplexer w/ Enable 74150

- Enable input
- Inverted outputs

Logic Symbol



Macrocell count:	3
Array inputs:	23
Product terms used:	17
Product terms allocated:	20

Functional Description

The 74150 macro decodes four data-input lines to select one of 16 data sources. The enable input, G, must be LOW to enable the Y output.

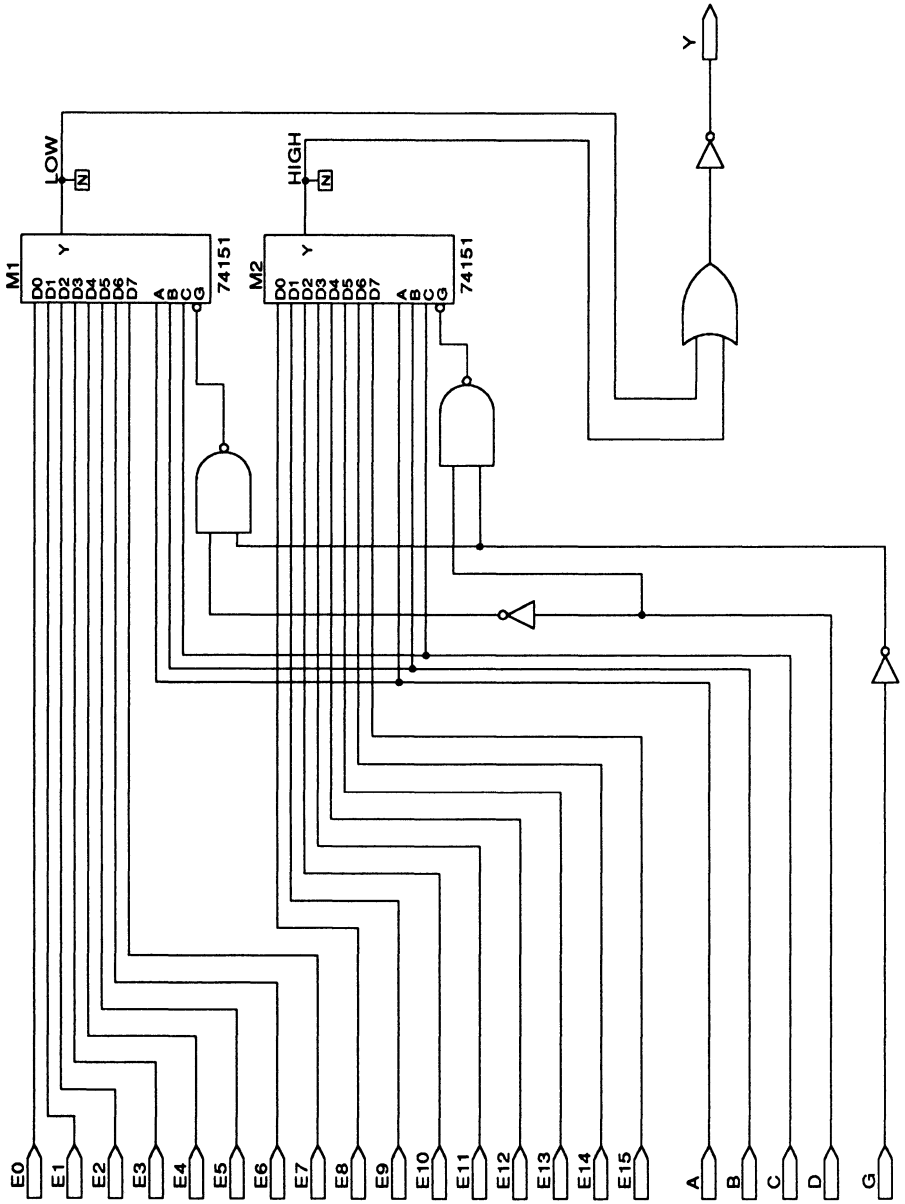
Sample PDS Equivalent

```
LOW = ((E7 * A * B * C * (/D * /G)) + (E6 * /A
   * B * C * (/D * /G)) + (E5 * A * /B * C * (/D
   * /G)) + (E4 * /A * /B * C * (/D * /G)) + (E3
   * A * B * /C * (/D * /G)) + (E2 * /A * B * /C
   * (/D * /G)) + (E1 * A * /B * /C * (/D * /G))
   + (E0 * /A * /B * /C * (/D * /G)))
HIGH = ((E15 * A * B * C * (D * /G)) + (E14 * /A
   * B * C * (D * /G)) + (E13 * A * /B * C * (D
   * /G)) + (E12 * /A * /B * C * (D * /G)) + (E11
   * A * B * /C * (D * /G)) + (E10 * /A * B * /C
   * (D * /G)) + (E9 * A * /B * /C * (D * /G)) + (E8
   * /A * /B * /C * (D * /G)))
Y = (LOW + HIGH)
```

Function Table

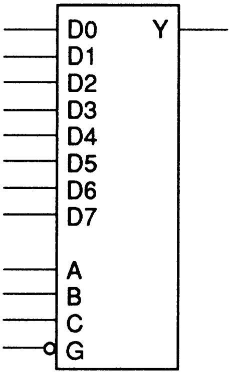
Inputs					Outputs
Select				Strobe	
D	C	B	A	G	Y
X	X	X	X	H	H
L	L	L	L	L	E0
L	L	L	H	L	E1
L	L	H	L	L	E2
L	L	H	H	L	E3
L	H	L	L	L	E4
L	H	L	H	L	E5
L	H	H	L	L	E6
L	H	H	H	L	E7
H	L	L	L	L	E8
H	L	L	H	L	E9
H	L	H	L	L	E10
H	L	H	H	L	E11
H	H	L	L	L	E12
H	H	L	H	L	E13
H	H	H	L	L	E14
H	H	H	H	L	E15

16-to-1 Multiplexer w/ Enable



- Enable input

Logic Symbol



Macrocell count:	1
Array inputs:	12
Product terms used:	8
Product terms allocated:	8

Functional Description

The 74151 macro decodes three data-input lines to select one of eight data sources. The enable input, G, must be LOW to enable the Y output.

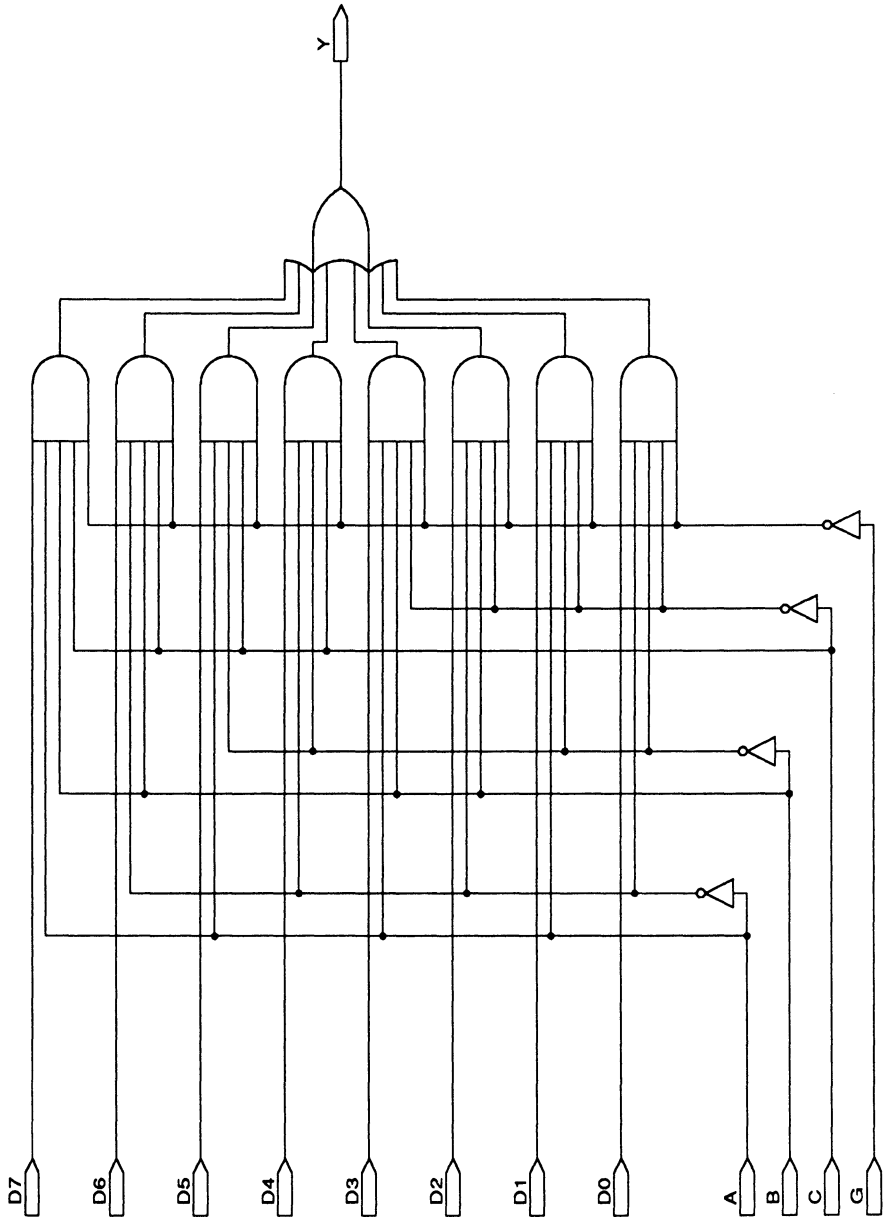
Sample PDS Equivalent

```
Y = ((D7 * A * B * C * /G) + (D6 * /A * B * C * /G)
  + (D5 * A * /B * C * /G) + (D4 * /A * /B * C * /G)
  + (D3 * A * B * /C * /G) + (D2 * /A * B * /C * /G)
  + (D1 * A * /B * /C * /G) + (D0 * /A * /B * /C * /G))
```

Function Table

Inputs				Outputs
Select			Strobe	
C	B	A	G	Y
X	X	X	H	L
L	L	L	L	D0
L	L	H	L	D1
L	H	L	L	D2
L	H	H	L	D3
H	L	L	L	D4
H	L	H	L	D5
H	H	L	L	D6
H	H	H	L	D7

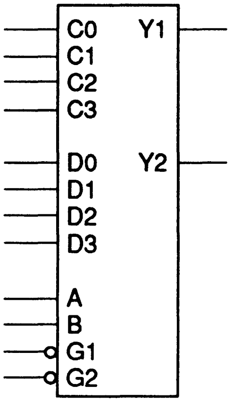
74151 8-to-1 Multiplexer w/ Enable 74151



74153 Dual 4-to-1 Multiplexer w/ Enable 74153

- Individual enable inputs
- Common data-select inputs

Logic Symbol



Macrocell count:	2
Array inputs:	12
Product terms used:	8
Product terms allocated:	8

Functional Description

The 74153 macro consists of two 4-to-1 multiplexers with common data-select lines. Each 4-to-1 multiplexer has an active LOW strobe input line to enable the output.

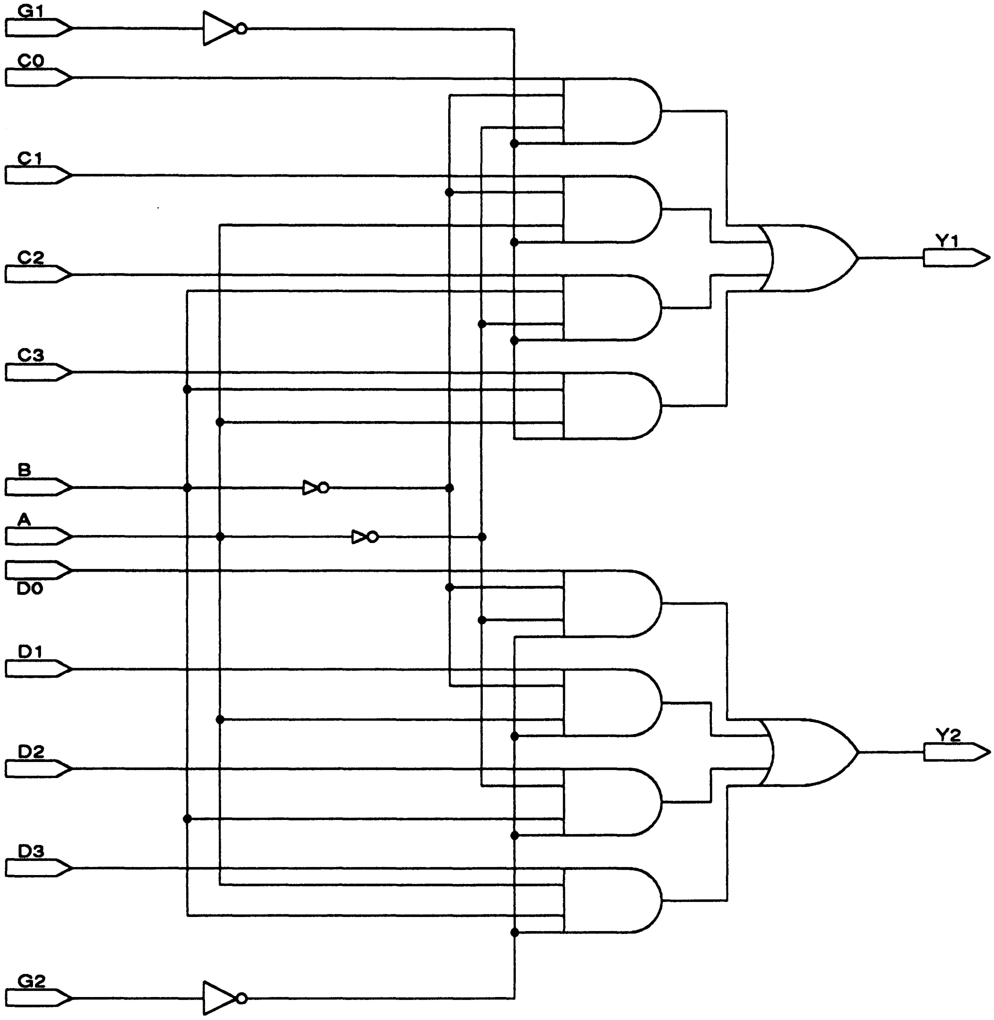
Sample PDS Equivalent

```
Y1 = ((C0 * /B * /A * /G1) + (C1 * /B * A * /G1)
  + (C2 * B * /A * /G1) + (C3 * B * A * /G1))
Y2 = ((D0 * /B * /A * /G2) + (D1 * /B * A * /G2)
  + (D2 * /A * B * /G2) + (D3 * A * B * /G2))
```

Function Table

Inputs							Outputs
Select		Data				Strobe	
B	A	C0	C1	C2	C3	G	Y
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

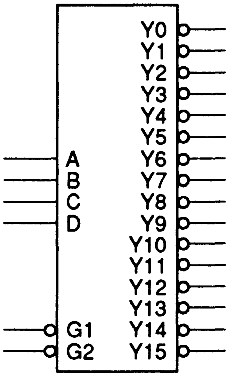
Dual 4-to-1 Multiplexer w/ Enable



74154 4-to-16 Line Decoder 74154

- Active LOW outputs
- Two enable inputs

Logic Symbol



Macrocell count:	16
Array inputs:	6
Product terms used:	16
Product terms allocated:	64

Functional Description

The 74154 macro decodes four data-input lines to select one of 16 active LOW outputs. You can use the enable inputs, G1 and G2, to cascade multiple macros.

Function Table

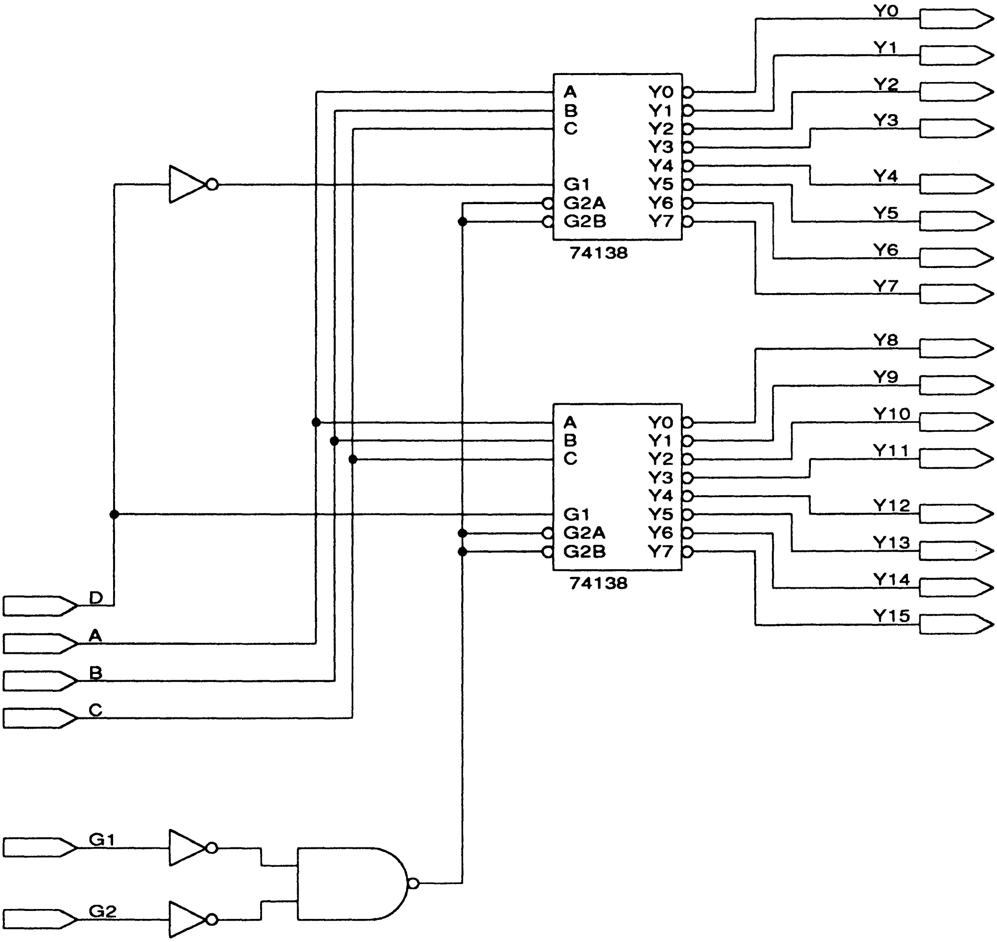
Inputs						Outputs															
Enable		Select																			
G1	G2	D	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15
X	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	X	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L

Sample PDS Equivalent

```
Y7 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (C * B * A))
Y6 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (C * B * /A))
Y5 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (C * /B * A))
Y4 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (C * /B * /A))
Y3 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (/C * B * A))
Y2 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (/C * B * /A))
Y1 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (/C * /B * A))
Y0 = ((/D * (/G1 * /G2) * (/G1 * /G2)) * (/C * /B * /A))
```

```
Y15 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (C * B * A))
Y14 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (C * B * /A))
Y13 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (C * /B * A))
Y12 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (C * /B * /A))
Y11 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (/C * B * A))
Y10 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (/C * B * /A))
Y9 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (/C * /B * A))
Y8 = ((D * (/G1 * /G2) * (/G1 * /G2)) * (/C * /B * /A))
```

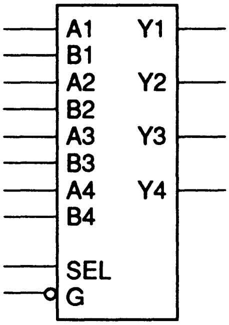
4-to-16 Line Decoder



Quad 2-to-1 Multiplexer

- Enable input

Logic Symbol



Macrocell count:	4
Array inputs:	10
Product terms used:	8
Product terms allocated:	16

Functional Description

The 74157 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced LOW regardless of the inputs.

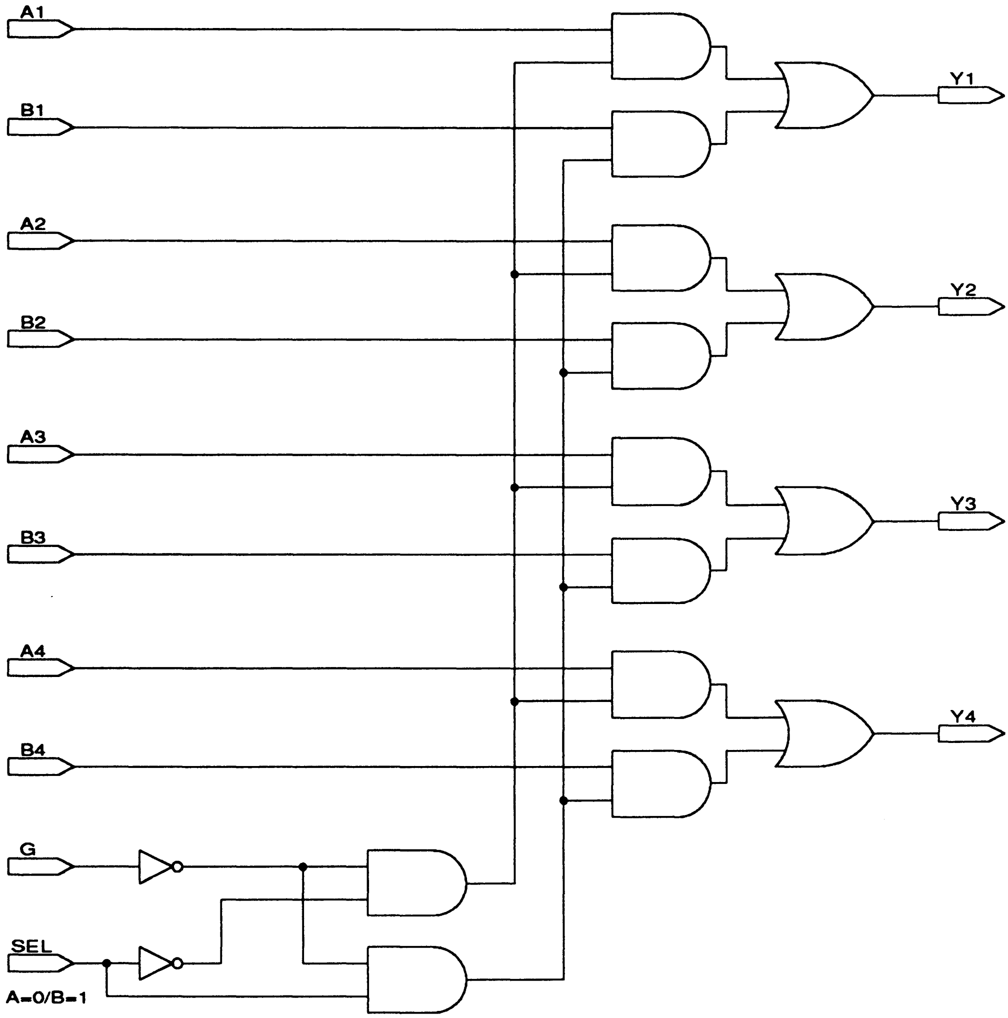
Sample PDS Equivalent

```
Y1 = ((A1 * (/G * /SEL)) + (B1 * (/G * SEL)))
Y2 = ((A2 * (/G * /SEL)) + (B2 * (/G * SEL)))
Y3 = ((A3 * (/G * /SEL)) + (B3 * (/G * SEL)))
Y4 = ((A4 * (/G * /SEL)) + (B4 * (/G * SEL)))
```

Function Table

Inputs		Outputs			
G	SEL	Y1	Y2	Y3	Y4
H	X	L	L	L	L
L	L	A1	A2	A3	A4
L	H	B1	B2	B3	B4

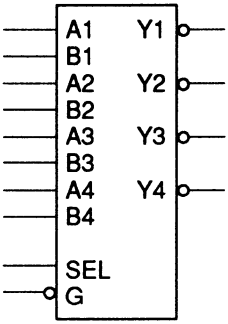
Quad 2-to-1 Multiplexer



74158 Quad 2-to-1 Multiplexer 74158

- Enable input
- Inverted outputs

Logic Symbol



Macrocell count:	4
Array inputs:	10
Product terms used:	8
Product terms allocated:	16

Functional Description

The 74158 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced HIGH regardless of the inputs.

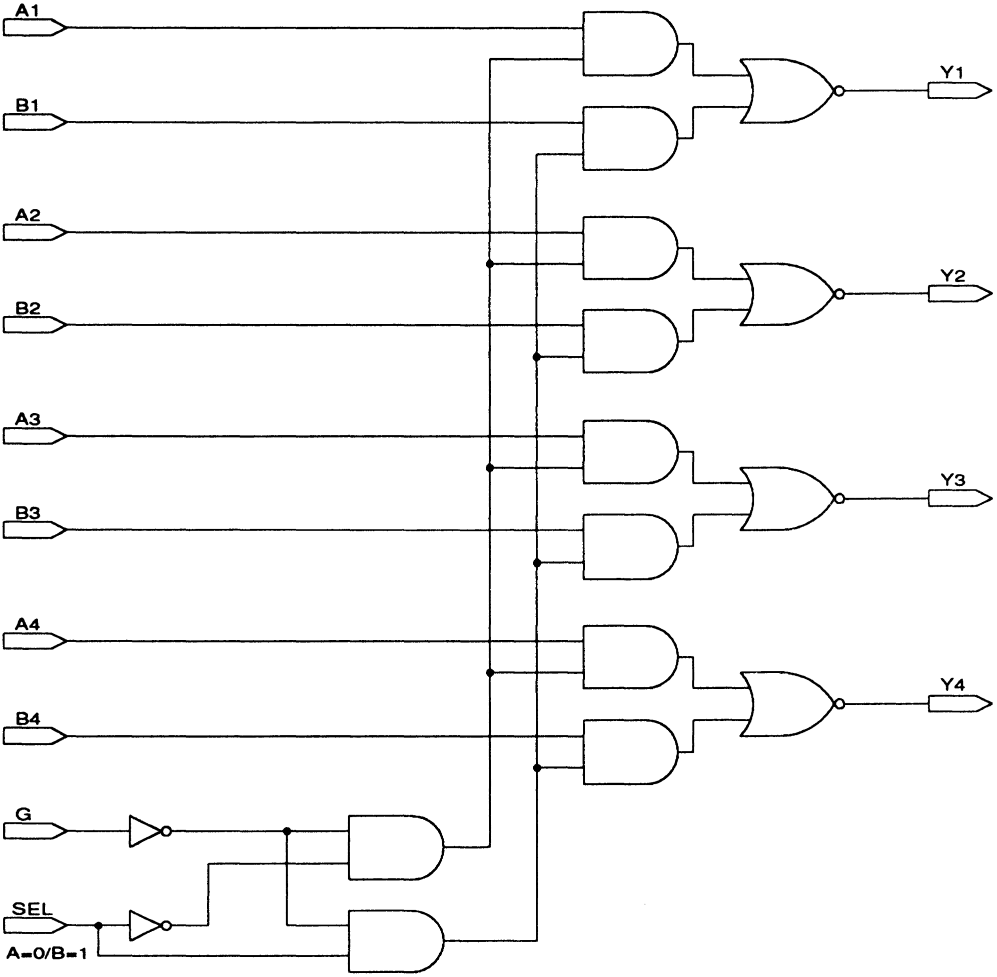
Sample PDS Equivalent

```
Y1 = /((A1 * (/G * /SEL)) + (B1 * (/G * SEL)))
Y2 = /((A2 * (/G * /SEL)) + (B2 * (/G * SEL)))
Y3 = /((A3 * (/G * /SEL)) + (B3 * (/G * SEL)))
Y4 = /((A4 * (/G * /SEL)) + (B4 * (/G * SEL)))
```

Function Table

Inputs		Outputs			
G	SEL	Y1	Y2	Y3	Y4
H	X	H	H	H	H
L	L	/A1	/A2	/A3	/A4
L	H	/B1	/B2	/B3	/B4

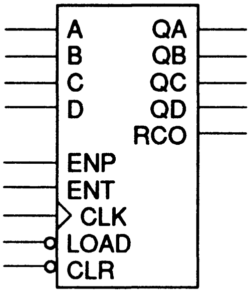
Quad 2-to-1 Multiplexer



74162 4-Bit BCD/Decade Counter w/ Synchronous Reset 74162

- Synchronous load
- Synchronous reset
- Carry look-ahead output
- Two enable inputs

Logic Symbol



Macrocell count:	5
Array inputs:	12
Product terms used:	18
Product terms allocated:	24

Functional Description

The 74162 macro is a 4-bit BCD loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count, 9, has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

Sample PDS Equivalent

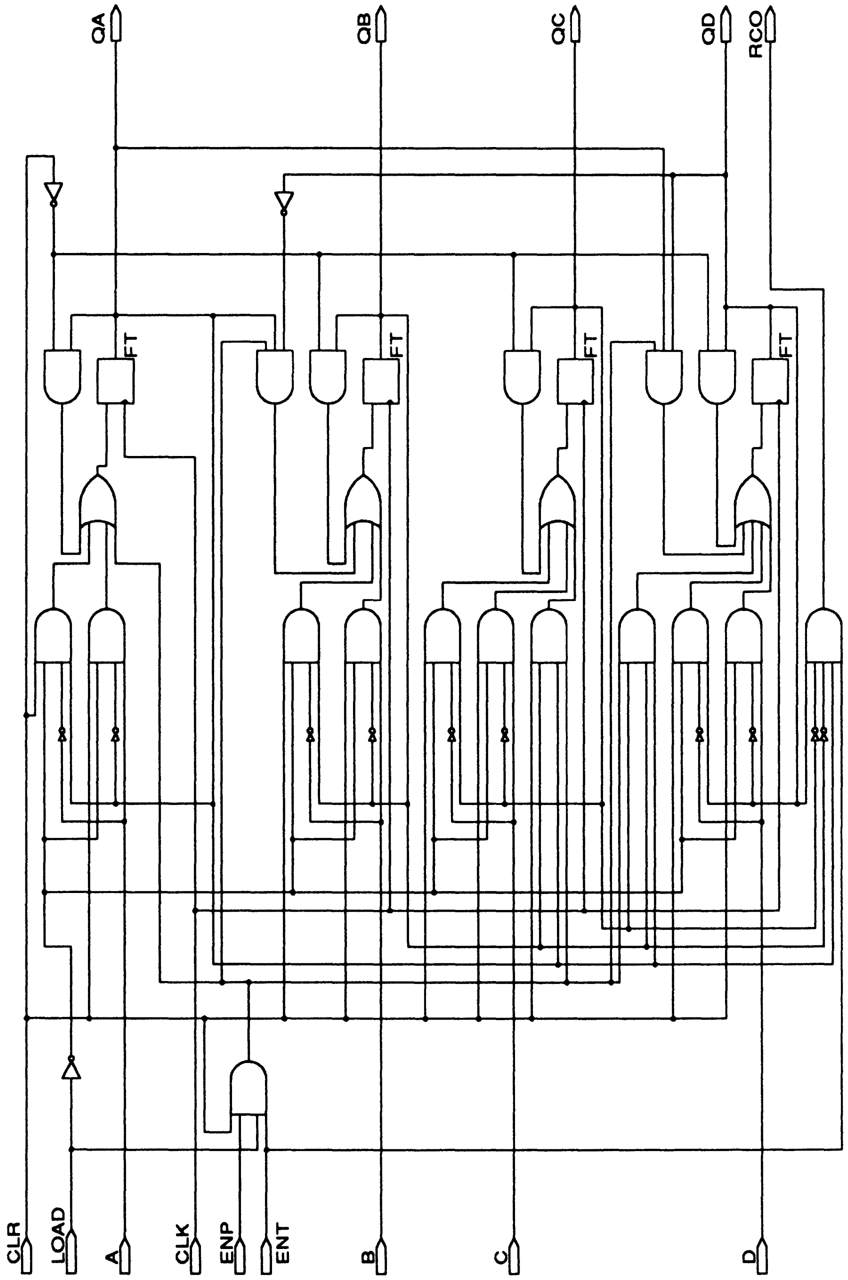
```
QA.T = ((/CLR * QA) + (CLR
  * /LOAD * /A * QA) + (CLR
  * /LOAD * /QA * A) + (CLR
  * ENP * LOAD * ENT))
QA.clkf = CLK
QB.T = ((/CLR * QB) + ((CLR
  * ENP * LOAD * ENT) * QA
  * /QD) + (CLR * /LOAD * /B
  * QB) + (CLR * /LOAD * /QB * B))
QB.clkf = CLK
QC.T = ((/CLR * QC) + (CLR
  * /LOAD * /C * QC) + (CLR
  * /LOAD * /QC * C) + (CLR
  * QB * QA * (CLR * ENP
  * LOAD * ENT)))
QC.clkf = CLK
QD.T = ((/CLR * QD) + ((CLR
  * ENP * LOAD * ENT) * QA
  * QD) + ((CLR * ENP * LOAD
  * ENT) * QC * QB * QA)
  + (CLR * /LOAD * /D * QD)
  + (CLR * /LOAD * /QD * D))
QD.clkf = CLK
RCO = (QD * /QC * /QB * QA * ENT)
```

Function Table

Inputs						Outputs			
Mode	CLK	CLR	LOAD	ENP	ENT	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 9 and ENT is HIGH. Otherwise, it stays LOW.

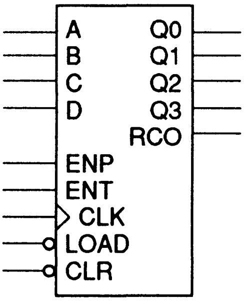
74162 4-Bit BCD/Decade Counter w/ Synchronous Reset 74162



74163 4-Bit Binary Counter w/ Synchronous Reset 74163

- Synchronous 4-bit binary-loadable up counter
- Synchronous reset
- Carry look-ahead output for making wider counters

Logic Symbol



Macrocell count:	5
Array inputs:	12
Product terms used:	17
Product terms allocated:	20

Functional Description

The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count of 15 has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

Sample PDS Equivalent

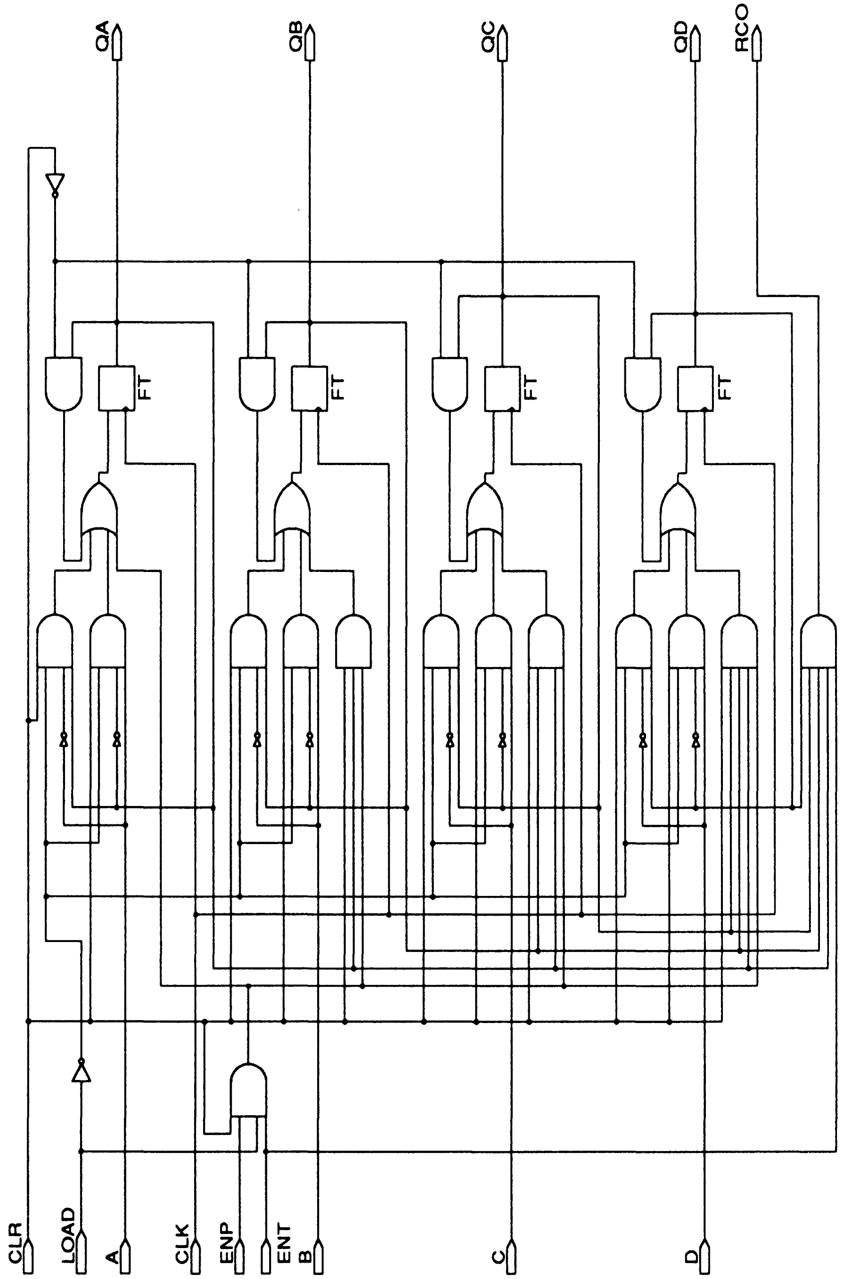
```
QA.T = ((/CLR * QA) + (CLR * /LOAD
  * /A * QA) + (CLR * /LOAD * /QA
  * A) + (CLR * ENP * LOAD * ENT))
QA.clkf = CLK
QB.T = ((/CLR * QB) + (CLR * /LOAD
  * /B * QB) + (CLR * /LOAD * /QB
  * B) + (CLR * QA * (CLR * ENP
  * LOAD * ENT)))
QB.clkf = CLK
QC.T = ((/CLR * QC) + (CLR * /LOAD
  * /C * QC) + (CLR * /LOAD * /QC
  * C) + (CLR * QB * QA * (CLR * ENP
  * LOAD * ENT)))
QC.clkf = CLK
QD.T = ((/CLR * QD) + (CLR * /LOAD
  * /D * QD) + (CLR * /LOAD * /QD
  * D) + (CLR * QC * QB * QA * (CLR
  * ENP * LOAD * ENT)))
QD.clkf = CLK
RCO = (QD * QC * QB * QA * ENT)
```

Function Table

	Inputs					Outputs			
Mode	CLK	CLR	Load	ENP	ENT	QD	QC	QB	QA
Clear	L	L	X	X	X	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 15 and ENT is HIGH. Otherwise, it stays LOW.

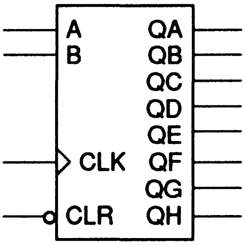
74163 4-Bit Binary Counter w/ Synchronous Reset 74163



74164 8-Bit Serial-In Parallel-Out Shift Register 74164

- Synchronous reset
- ANDed serial inputs

Logic Symbol



Macrocell count:	8
Array inputs:	10
Product terms used:	8
Product terms allocated:	32

Functional Description

The 74164 macro is an 8-bit serial-in parallel-out shift register with synchronous reset. The two serial inputs, A and B, are logically ANDed.

Note:
The TTL version has asynchronous reset logic.

Sample PDS Equivalent

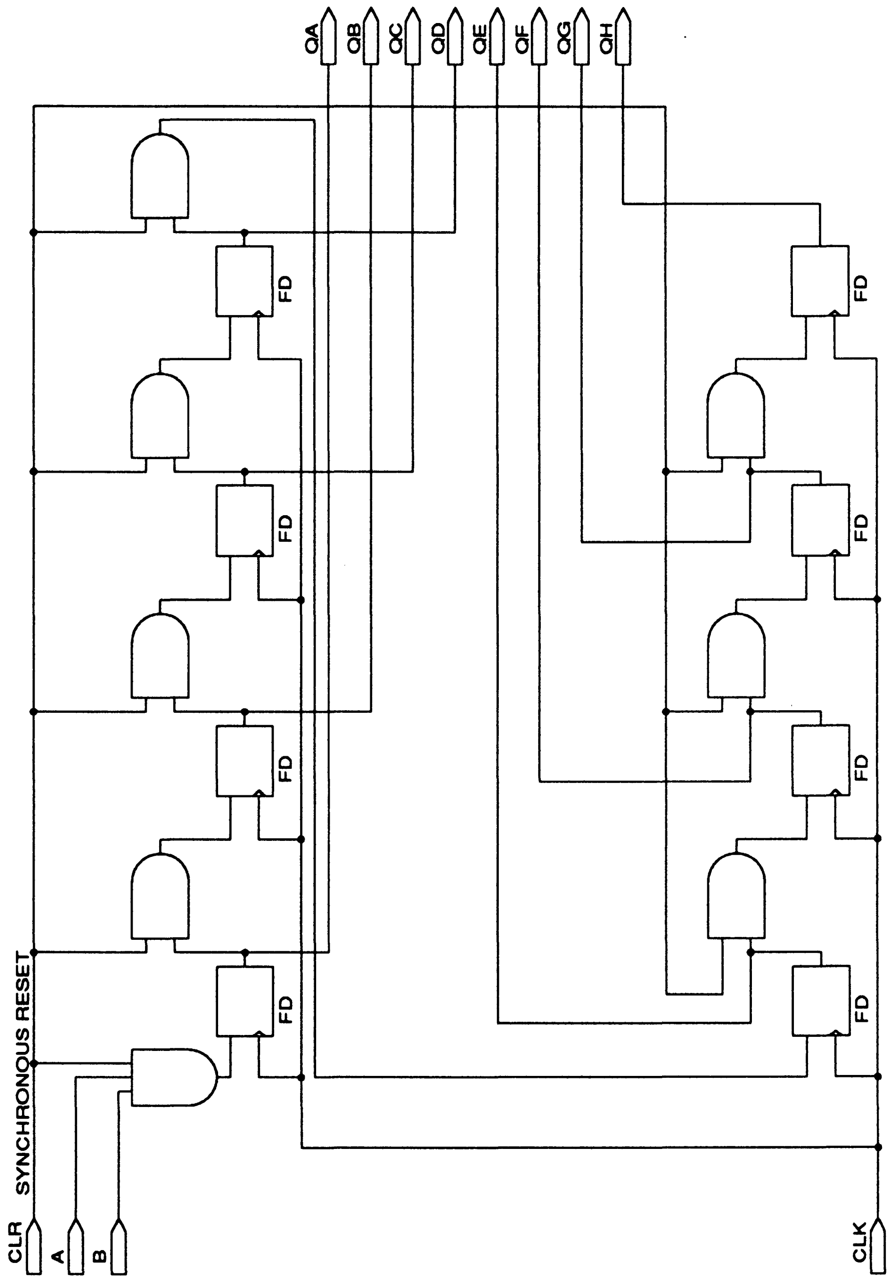
```
QA = (B * A * CLR)
QA.clkf = CLK
QB = (CLR * QA)
QB.clkf = CLK
QC = (CLR * QB)
QC.clkf = CLK
QD = (CLR * QC)
QD.clkf = CLK
QE = (CLR * QD)
QE.clkf = CLK
QF = (CLR * QE)
QF.clkf = CLK
QG = (CLR * QF)
QG.clkf = CLK
QH = (CLR * QG)
QH.clkf = CLK
```

Function Table

Inputs				Outputs							
CLK	CLR	A	B	QA	QB	QC	QD	QE	QF	QG	QH
L	L	X	X	QAo	QBo	QCo	QDo	QEo	QFo	QGo	QHo
↑	L	X	X	L	L	L	L	L	L	L	L
↑	H	H	H	H	QAn	QBn	QCn	QDn	QEn	QFn	QGn
↑	H	L	X	L	QAn	QBn	QCn	QDn	QEn	QFn	QGn
↑	H	X	L	L	QAn	QBn	QCn	QDn	QEn	QFn	QGn

* QAo to QHo = previous state of QA to QH
QAn to QGn = level of QA to QG before the most recent rising transition of the CLK, and indicates a 1-bit shift.

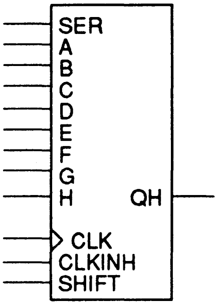
8-Bit Serial-In Parallel-Out Shift Register



74165 Parallel-Load 8-Bit Shift Register 74165

- Synchronous load
- Shift inhibit

Logic Symbol



Macrocell count:	8
Array inputs:	19
Product terms used:	24
Product terms allocated:	32

Functional Description

The 74165 macro is an 8-bit parallel-in serial-out shift register. To synchronously load the registers, you set the SHIFT input LOW. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values. A parallel-load operation overrides the inhibit function.

Note:
The TTL version has two clock lines and asynchronous load logic.

Function Table

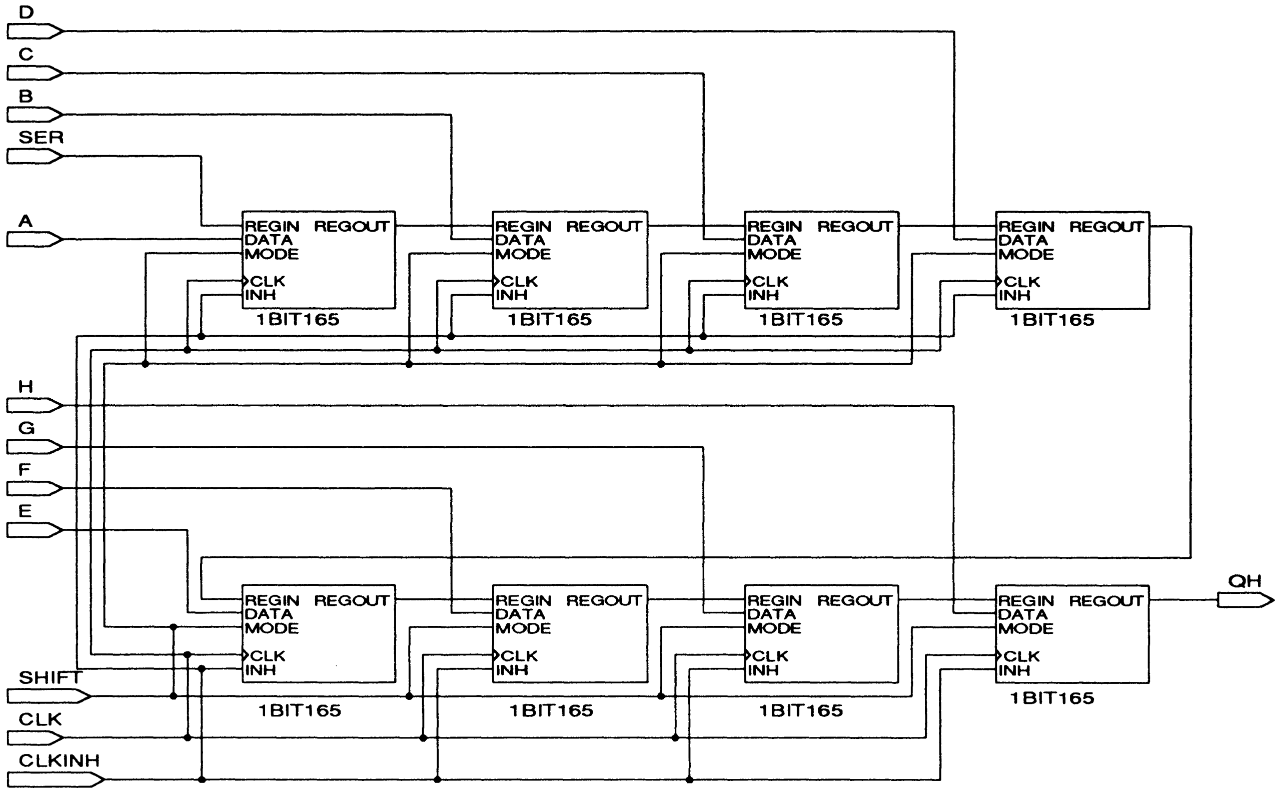
Inputs				Outputs / Internal Registers															
SHIFT	INH	CLK	SER	A	B	C	D	E	F	G	H	QA	QB	QC	QD	QE	QF	QG	QH
L	X	↑	X	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h
L	X	L	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEo	QFo	QGo	QHo
H	L	L	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEo	QFo	QGo	QHo
H	L	↑	L	X	X	X	X	X	X	X	X	L	QAo	QBo	QCo	QDo	QEo	QFo	QGo
H	L	↑	H	X	X	X	X	X	X	X	X	H	QAo	QBo	QCo	QDo	QEo	QFo	QGo
H	H	↑	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEo	QFo	QGo	QHo

* QAo to QHo = previous state of registers QA to QH

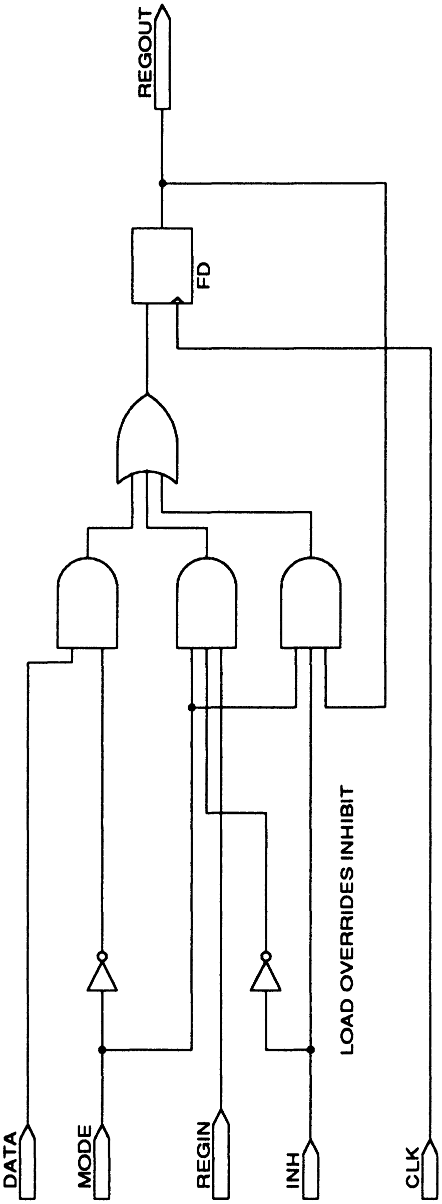
Sample PDS Equivalent

```
M1_REGOUT = ((A * /SHIFT) + (SHIFT * /CLKINH * SER) + (SHIFT * CLKINH * M1_REGOUT))
M1_REGOUT.clkf = CLK
M2_REGOUT = ((B * /SHIFT) + (SHIFT * /CLKINH * M1_REGOUT) + (SHIFT * CLKINH * M2_REGOUT))
M2_REGOUT.clkf = CLK
M3_REGOUT = ((C * /SHIFT) + (SHIFT * /CLKINH * M2_REGOUT) + (SHIFT * CLKINH * M3_REGOUT))
M3_REGOUT.clkf = CLK
M4_REGOUT = ((D * /SHIFT) + (SHIFT * /CLKINH * M3_REGOUT) + (SHIFT * CLKINH * M4_REGOUT))
M4_REGOUT.clkf = CLK
M8_REGOUT = ((E * /SHIFT) + (SHIFT * /CLKINH * M4_REGOUT) + (SHIFT * CLKINH * M8_REGOUT))
M8_REGOUT.clkf = CLK
M5_REGOUT = ((F * /SHIFT) + (SHIFT * /CLKINH * M8_REGOUT) + (SHIFT * CLKINH * M5_REGOUT))
M5_REGOUT.clkf = CLK
M6_REGOUT = ((G * /SHIFT) + (SHIFT * /CLKINH * M5_REGOUT) + (SHIFT * CLKINH * M6_REGOUT))
M6_REGOUT.clkf = CLK
QH = ((H * /SHIFT) + (SHIFT * /CLKINH * M6_REGOUT) + (SHIFT * CLKINH * QH))
QH.clkf = CLK
```

74165 Parallel-Load 8-Bit Shift Register 74165



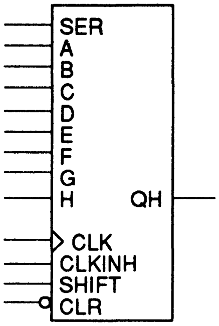
1Bit of a 74165 Shift Register



74166 8-Bit Parallel-In Serial-Out Shift Register 74166

- Parallel synchronous load
- Synchronous reset

Logic Symbol



Macrocell count:	8
Array inputs:	20
Product terms used:	24
Product terms allocated:	32

Functional Description

The 74166 macro is an 8-bit parallel-in serial-out shift register with synchronous reset. To load the registers, you set the SHIFT input LOW and apply a rising-edge clock. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values.

Note:
The TTL version has asynchronous reset logic.

Function Table

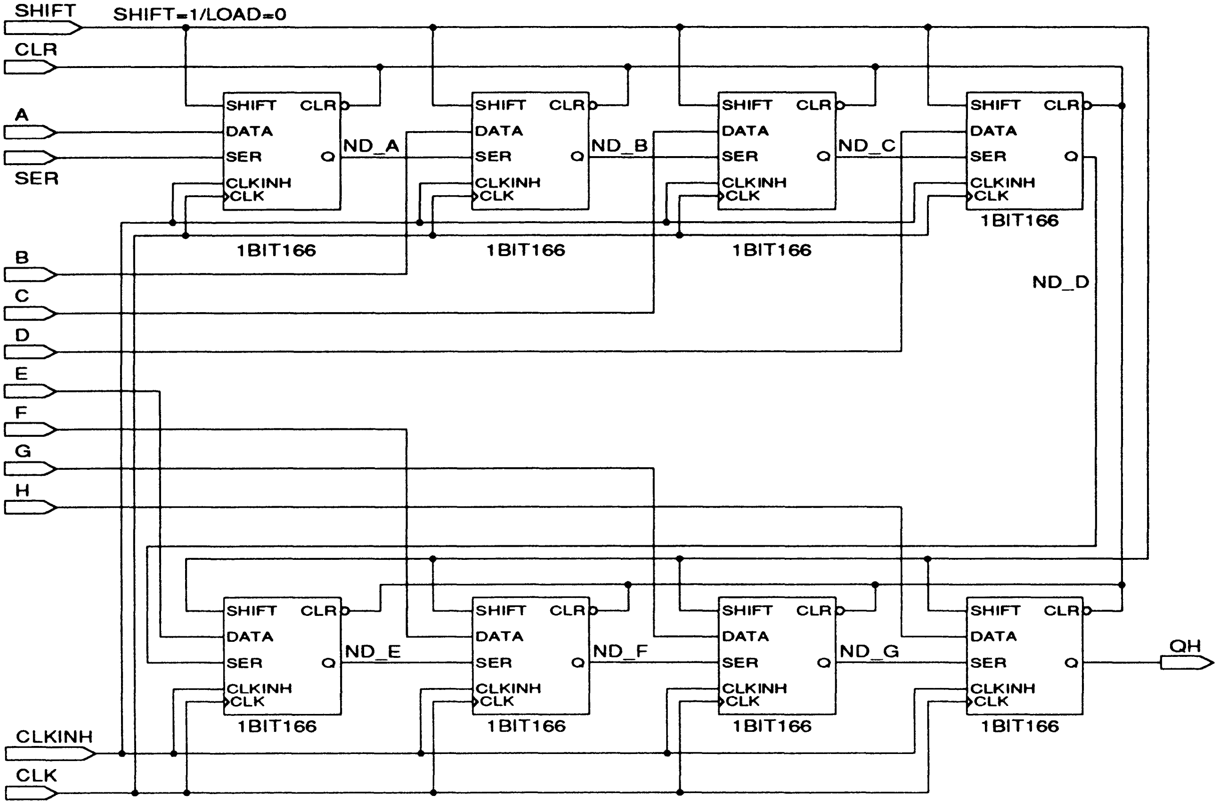
Inputs					Internal Outputs																Output
SHIFT	INH	CLK	CLR	SER	A	B	C	D	E	F	G	H	QAi	QBi	QCi	QDi	QEi	QFi	QGi	QHi	QH
X	X	L	L	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEo	QFo	QGo	QHo	QHo
X	X	↑	L	X	X	X	X	X	X	X	X	X	L	L	L	L	L	L	L	L	L
X	X	L	X	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEo	QFo	QGo	QHo	QHo
L	L	↑	H	X	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h	h
H	L	↑	H	L	X	X	X	X	X	X	X	X	L	QAn	QBn	QCn	QDn	QEn	QFn	QGn	QGn
H	L	↑	H	H	X	X	X	X	X	X	X	X	H	QAn	QBn	QCn	QDn	QEn	QFn	QGn	QGn
X	H	↑	H	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEo	QFo	QGo	QHo	QHo

* QAo to QHo = previous state of QA to QH
QAn to QGn = level of QA to QG before the most recent rising transition of the CLK, and indicates a 1-bit shift.

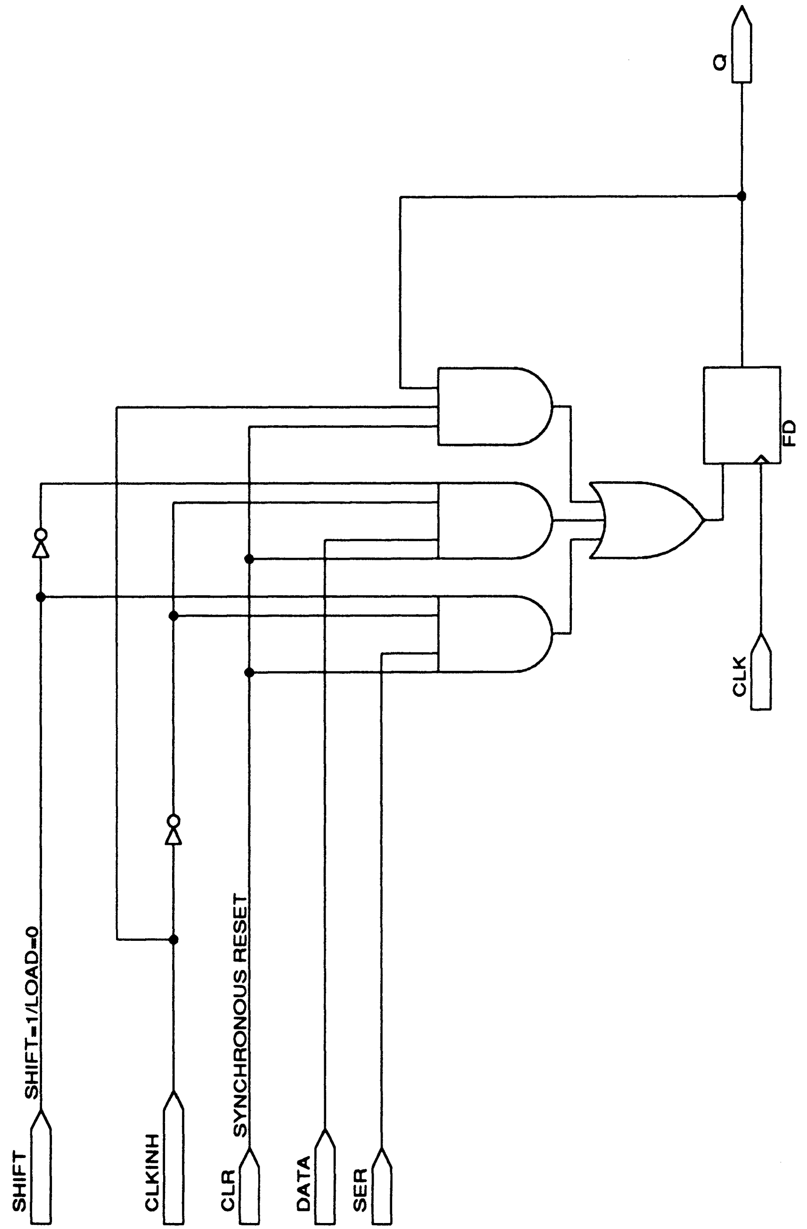
Sample PDS Equivalent

```
ND_A = ((CLR * SER * /CLKINH * SHIFT) + (CLR * A * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_A))
ND_A.clkf = CLK
ND_B = ((CLR * ND_A * /CLKINH * SHIFT) + (CLR * B * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_B))
ND_B.clkf = CLK
ND_C = ((CLR * ND_B * /CLKINH * SHIFT) + (CLR * C * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_C))
ND_C.clkf = CLK
ND_D = ((CLR * ND_C * /CLKINH * SHIFT) + (CLR * D * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_D))
ND_D.clkf = CLK
ND_E = ((CLR * ND_D * /CLKINH * SHIFT) + (CLR * E * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_E))
ND_E.clkf = CLK
ND_F = ((CLR * ND_E * /CLKINH * SHIFT) + (CLR * F * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_F))
ND_F.clkf = CLK
ND_G = ((CLR * ND_F * /CLKINH * SHIFT) + (CLR * G * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_G))
ND_G.clkf = CLK
QH = ((CLR * ND_G * /CLKINH * SHIFT) + (CLR * H * /CLKINH * /SHIFT) + (CLR * CLKINH * QH))
QH.clkf = CLK
```

74166 8-Bit Parallel-In Serial-Out Shift Register 74166



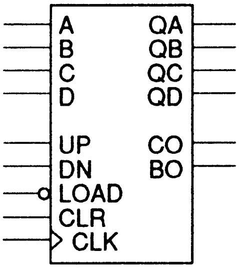
1Bit of a 74166 Shift Register



74192 4-Bit Up/Down BCD Counter 74192

- Synchronous load
- Asynchronous reset
- Carry- and borrow-out signals for expansion

Logic Symbol



Macrocell count:	6
Array inputs:	12
Product terms used:	22
Product terms allocated:	36

Functional Description

The 74192 macro is a 4-bit up/down BCD counter with synchronous parallel load and asynchronous reset logic. You can select an increasing or decreasing count sequence by setting either the UP or DN control input HIGH. An active-LOW borrow signal, BO, is generated when the count is zero and DN is HIGH. An active-LOW carry signal, CO, is generated when the count is 9 and UP is HIGH.

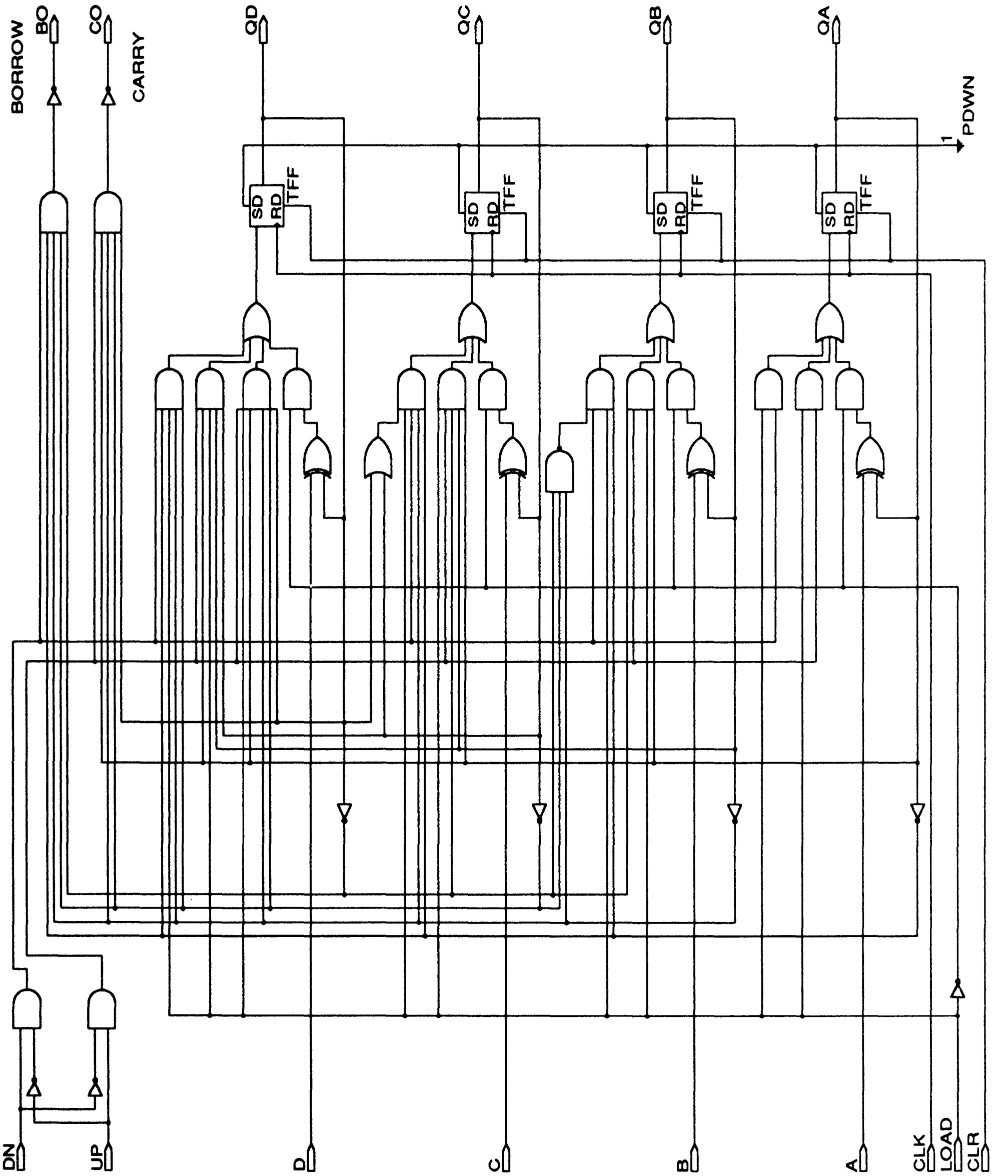
Note:
The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

Function Table

Inputs									Outputs					
CLK	CLR	LOAD	UP	DN	A	B	C	D	QA	QB	QC	QD	BO	CO
L	H	X	X	X	X	X	X	X	L	L	L	L	L	L
↑	L	L	X	X	a	b	c	d	a	b	c	d	X	X
↑	L	H	H	L	X	X	X	X	Count Up				H	H
↑	L	H	L	H	X	X	X	X	Count Down				H	H
↑	L	H	H	L	X	X	X	X	H	L	L	H	H	L
↑	L	H	L	H	X	X	X	X	L	L	L	L	L	H
↑	L	H	L	L	X	X	X	X	Hold Count				X	X
↑	L	H	H	H	X	X	X	X	Hold Count				X	X

Sample PDS Equivalent

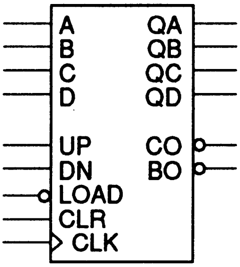
```
BO = ((DN * /UP) * /QA * /QB * /QC * /QD)
CO = ((/DN * UP) * QA * /QB * /QC * QD)
QD.T = (((DN * /UP) * /QA * LOAD * /QB * /QC) + ((/DN * UP) * QA * LOAD * QB * QC) + ((/DN * UP) * LOAD * QA
* /QB * /QC * QD) + (/LOAD * (D :+: QD)))
QD.clkf = CLK
QD.rstf = CLR
QC.T = (((QD + QC) * LOAD * (DN * /UP) * /QB * /QA) + (LOAD * (/DN * UP) * /QD * QB * QA) + (/LOAD * (C :+: QC)))
QC.clkf = CLK
QC.rstf = CLR
QB.T = ((/(/QD * /QC * /QB) * (DN * /UP) * LOAD * /QA) + (/QD * (/DN * UP) * LOAD * QA) + (/LOAD * (B :+: QB)))
QB.clkf = CLK
QB.rstf = CLR
QA.T = ((LOAD * (DN * /UP)) + (LOAD * (/DN * UP)) + (/LOAD * (A :+: QA)))
QA.clkf = CLK
QA.rstf = CLR
```


BORROW
BO
CO
CARRY
QD
QC
QB
QA
PDWN
1
SD
RD
TFF
SD
RD
TFF
SD
RD
TFF
SD
RD
TFF
DN
UP
D
C
B
A
CLK
LOAD
CLR

74193 4-Bit Up/Down Binary Counter 74193

- Synchronous load
- Carry- and borrow-out signals

Logic Symbol



Macrocell count:	6
Array inputs:	12
Product terms used:	18
Product terms allocated:	24

Functional Description

The 74193 macro is a 4-bit up/down binary counter with synchronous load and asynchronous reset logic. You can select an increasing or decreasing count sequence by setting either the UP or DN control input HIGH. An active-LOW borrow signal BO is generated when the count is zero and DN is HIGH. An active-LOW carry signal, CO, is generated when the count is 9 and UP is HIGH. A load operation overrides the count function.

Note:
The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

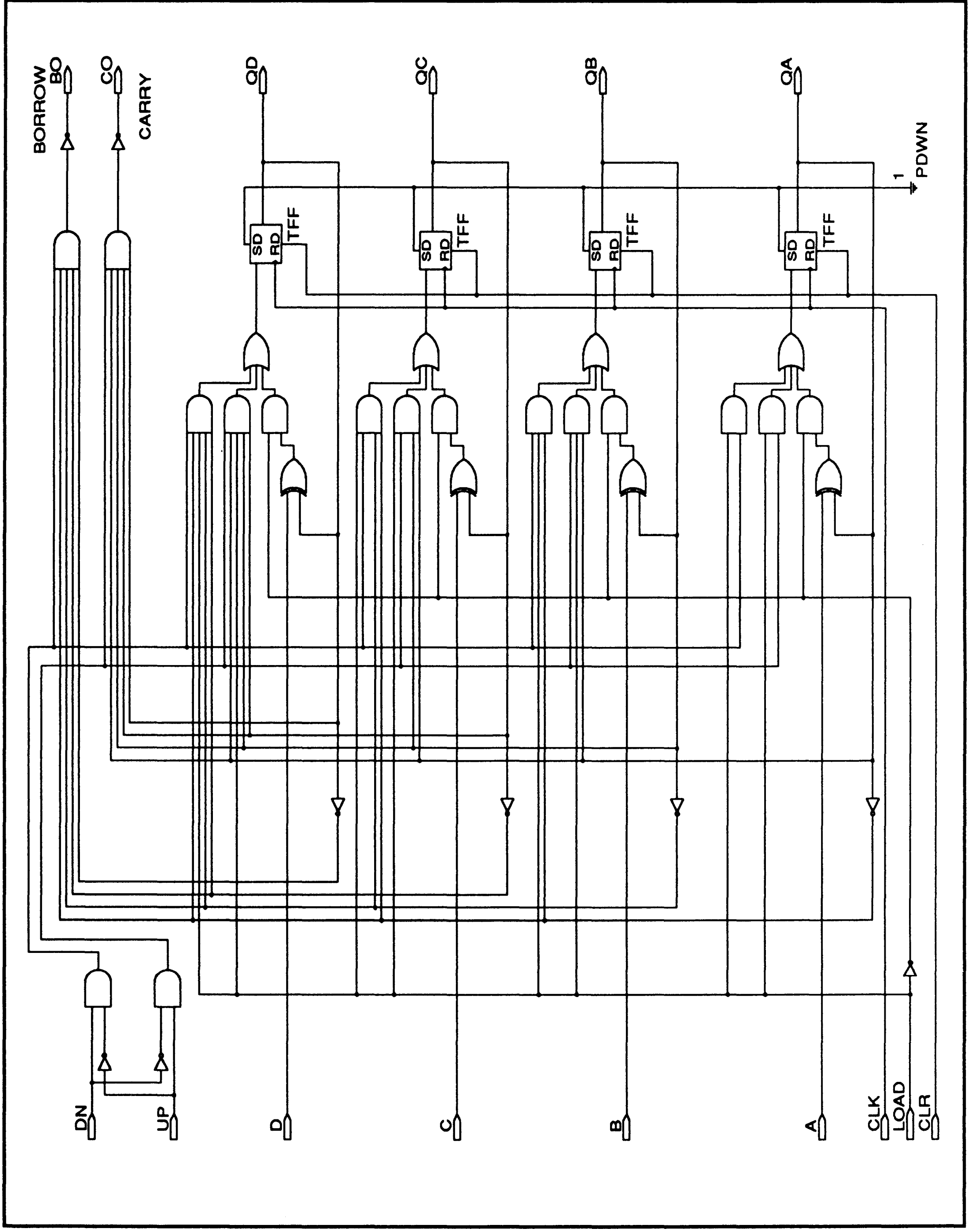
Function Table

Inputs									Outputs					
CLK	CLR	LOAD	UP	DN	A	B	C	D	QA	QB	QC	QD	BO	CO
L	H	X	X	X	X	X	X	X	L	L	L	L	L	H
↑	L	L	X	X	a	b	c	d	a	b	c	d	X	X
↑	L	H	H	L	X	X	X	X	Count Up				H	H
↑	L	H	L	H	X	X	X	X	Count Down				H	H
↑	L	H	H	L	X	X	X	X	H	H	H	H	H	L
↑	L	H	L	H	X	X	X	X	L	L	L	L	L	H
↑	L	H	L	L	X	X	X	X	Hold Count				X	X
↑	L	H	H	H	X	X	X	X	Hold Count				X	X

Sample PDS Equivalent

```
BO = ((DN * /UP) * /QA * /QB * /QC * /QD)
CO = ((/DN * UP) * QA * QB * QC * QD)
QD.T = (((DN * /UP) * /QA * LOAD * /QB * /QC) + ((/DN * UP) * QA * LOAD * QB * QC) + (/LOAD * (D :+: QD)))
QD.clkf = CLK
QD.rstf = CLR
QC.T = ((LOAD * (DN * /UP) * /QB * /QA) + (LOAD * (/DN * UP) * QB * QA) + (/LOAD * (C :+: QC)))
QC.clkf = CLK
QC.rstf = CLR
QB.T = (((DN * /UP) * LOAD * /QA) + ((/DN * UP) * LOAD * QA) + (/LOAD * (B :+: QB)))
QB.clkf = CLK
QB.rstf = CLR
QA.T = ((LOAD * (DN * /UP)) + (LOAD * (/DN * UP)) + (/LOAD * (A :+: QA)))
QA.clkf = CLK
QA.rstf = CLR
```

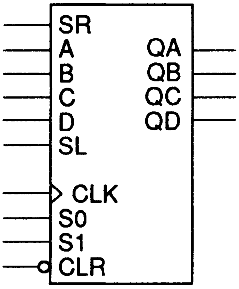
74193 4-Bit Up/Down Binary Counter 74193



74194 4-Bit Bidirectional Universal Shift Register 74194

- Parallel-to-serial converter
- Serial-to-parallel converter
- Synchronous reset
- Synchronous loading

Logic Symbol



Macrocell count:	4
Array inputs:	13
Product terms used:	16
Product terms allocated:	16

Functional Description

The 74194 macro is a 4-bit bidirectional universal shift register with synchronous reset logic. Two control lines, S1 and S0, select one of four modes of operation:

- Parallel load of four data inputs
- Right shift (in the direction QA to QD)
- Left shift (in the direction QD to QA)
- Data latch/hold register values

All operations are performed at the rising edge of CLK.

Note:
The TTL version has asynchronous reset logic.

Sample PDS Equivalent

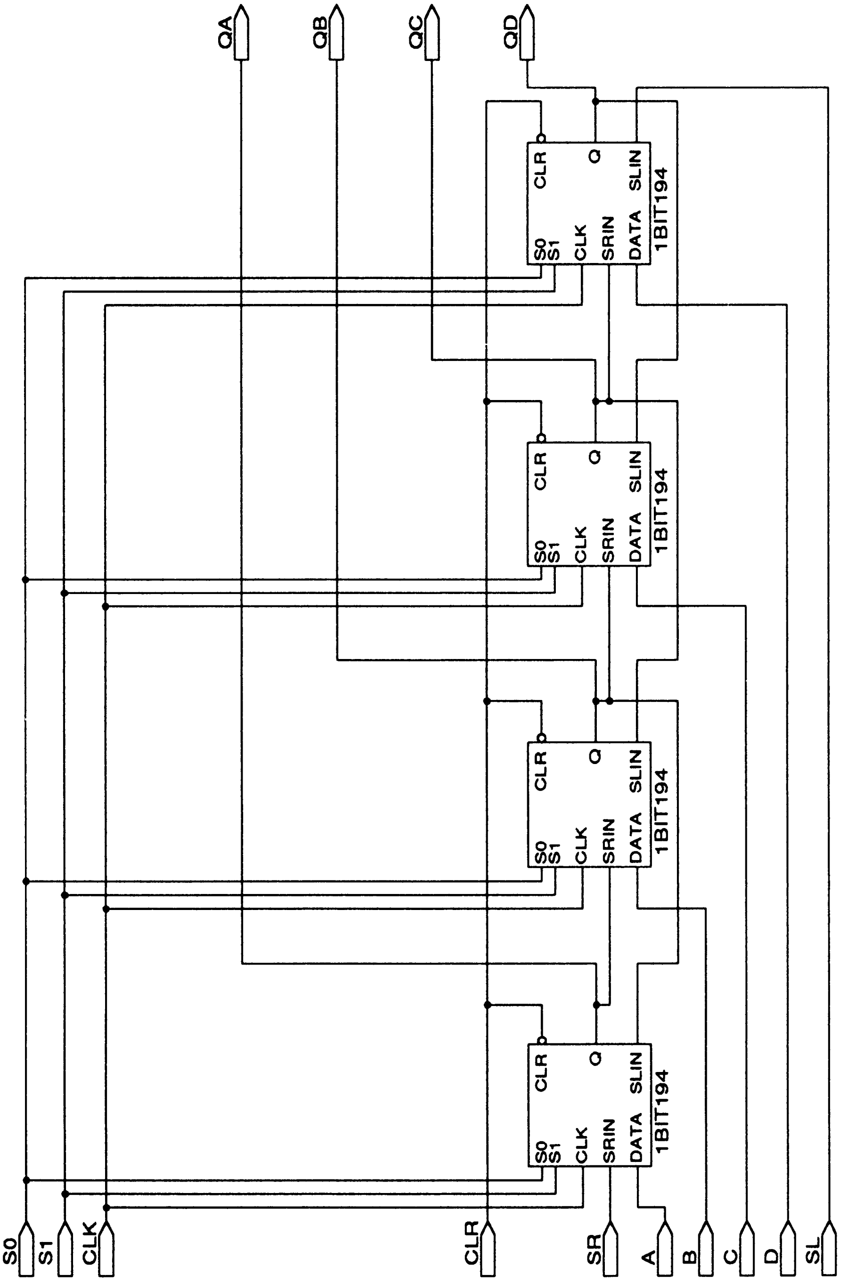
```
QA = ((CLR * SR * /S1 * S0) + (CLR
  * QB * S1 * /S0) + (CLR * S1 * S0
  * A) + (CLR * /S1 * /S0 * QA))
QA.clkf = CLK
QB = ((CLR * QA * /S1 * S0) + (CLR
  * QC * S1 * /S0) + (CLR * S1 * S0
  * B) + (CLR * /S1 * /S0 * QB))
QB.clkf = CLK
QC = ((CLR * QB * /S1 * S0) + (CLR
  * QD * S1 * /S0) + (CLR * S1 * S0
  * C) + (CLR * /S1 * /S0 * QC))
QC.clkf = CLK
QD = ((CLR * QC * /S1 * S0) + (CLR
  * SL * S1 * /S0) + (CLR * S1 * S0
  * D) + (CLR * /S1 * /S0 * QD))
QD.clkf = CLK
```

Function Table

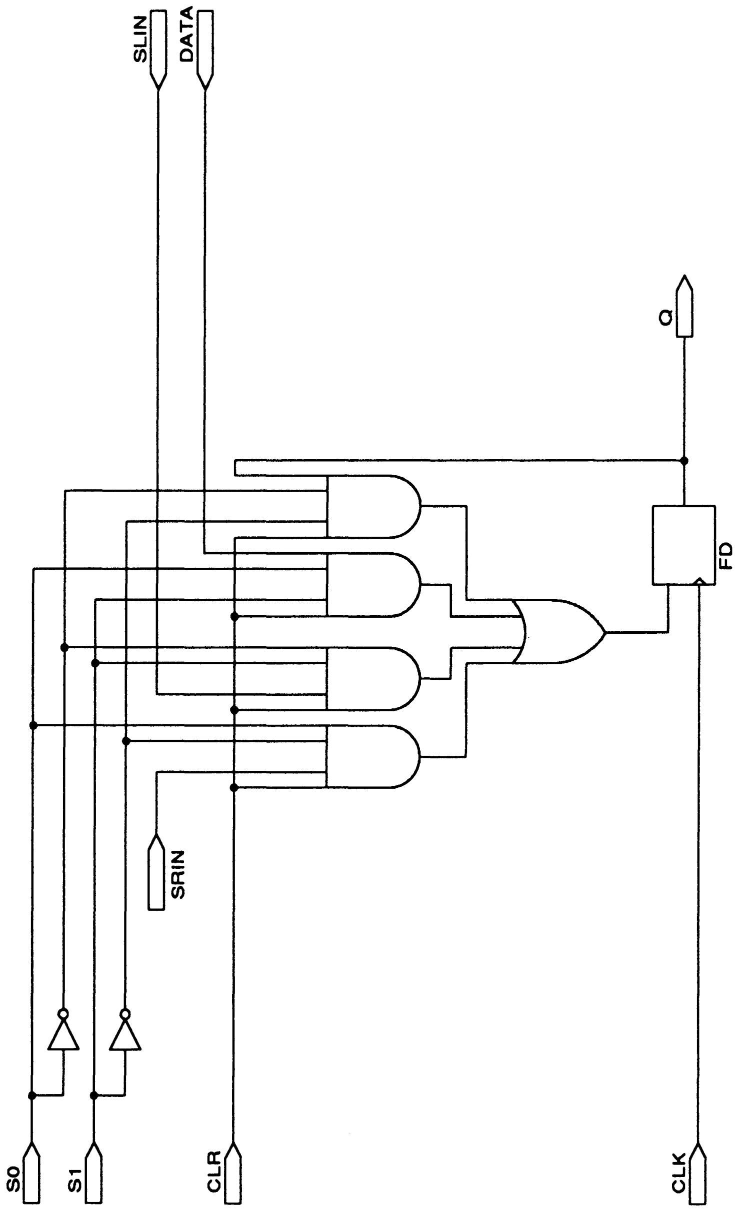
Inputs										Outputs			
Mode		Serial				Parallel							
CLK	CLR	S1	S0	SL	SR	A	B	C	D	QA	QB	QC	QD
L	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo
↑	L	X	X	X	X	X	X	X	X	L	L	L	L
X	H	L	L	X	X	X	X	X	X	QAo	QBo	QCo	QDo
L	H	L	H	X	L	X	X	X	X	L	QAn	QBn	QCn
↑	H	L	H	X	H	X	X	X	X	H	QAn	QBn	QCn
↑	H	H	L	L	X	X	X	X	X	QBn	QCn	QDn	L
↑	H	H	L	H	X	X	X	X	X	QBn	QCn	QDn	H
↑	H	H	H	X	X	a	b	c	d	a	b	c	d

* QAo to QDo = previous state of QA to QD
QAn to QDn = level of QA to QD before the most recent rising transition of the CLK, and indicates a 1-bit shift.

4-Bit Bidirectional Universal Shift Register



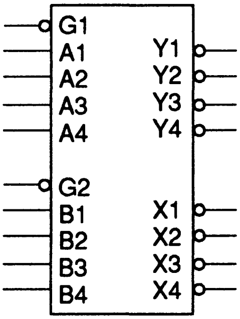
1Bit of a 74194 Shift Register



74240 Octal Inverting Buffers w/ 3-State Outputs 74240

- Two enable inputs

Logic Symbol



Macrocell count:	8
Array inputs:	10
Product terms used:	8
Product terms allocated:	32

Functional Description

The 74240 macro contains two groups of four inverting buffers. Each group is enabled by an active-LOW input control line.

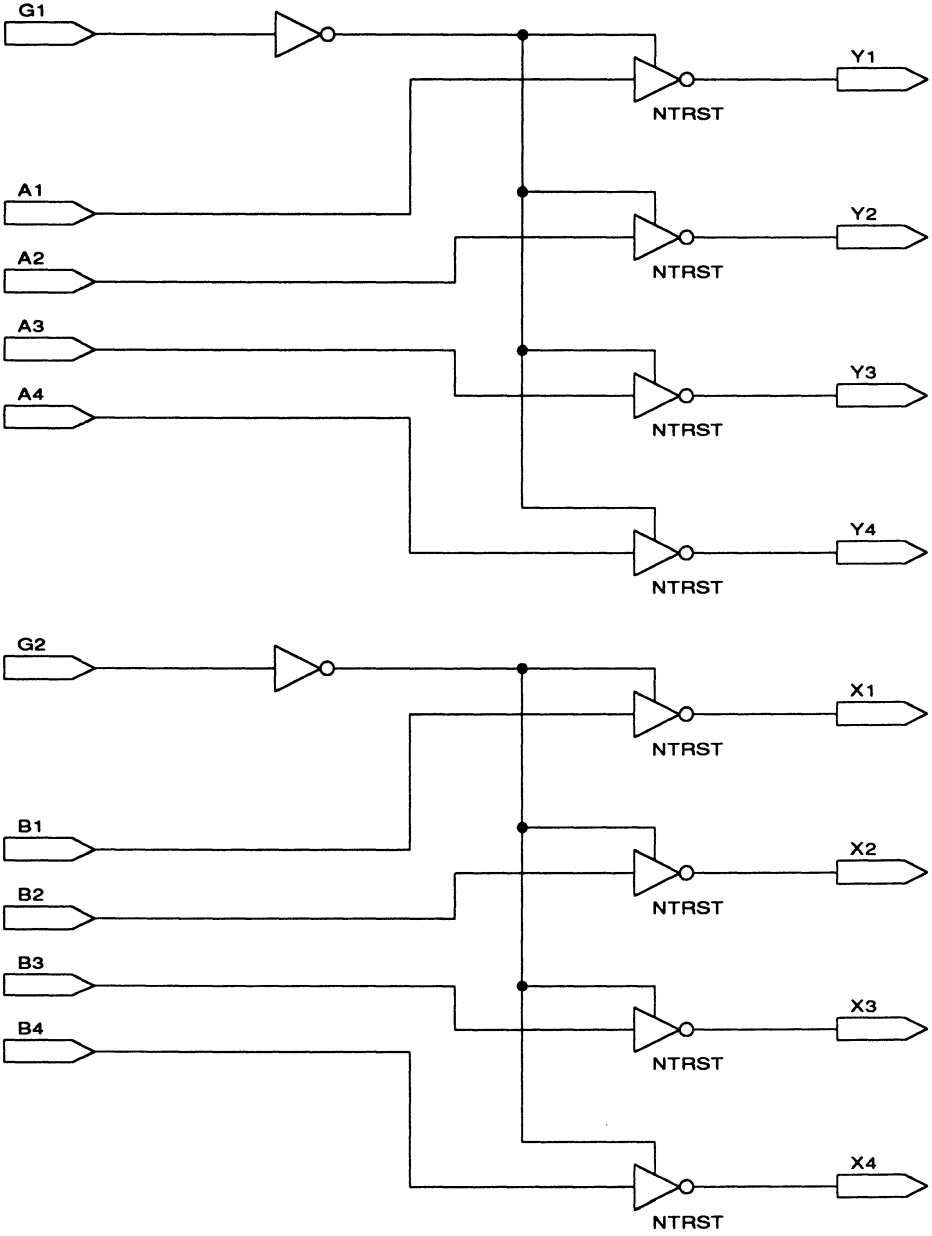
Sample PDS Equivalent

Y1.trst = /G1
Y2.trst = /G1
Y3.trst = /G1
Y4.trst = /G1
X1.trst = /G2
X2.trst = /G2
X3.trst = /G2
X4.trst = /G2
Y1 = A1
Y2 = A2
Y3 = A3
Y4 = A4
X1 = B1
X2 = B2
X3 = B3
X4 = B4

Function Table

Inputs										Outputs							
G1	A1	A2	A3	A4	G2	B1	B2	B3	B4	Y1	Y2	Y3	Y4	X1	X2	X3	X4
L	L	L	L	L	L	L	L	L	L	H	H	H	H	H	H	H	H
H	X	X	X	X	L	H	L	H	H	Z	Z	Z	Z	L	H	L	L
L	L	H	H	L	H	L	H	H	H	H	L	L	H	Z	Z	Z	Z

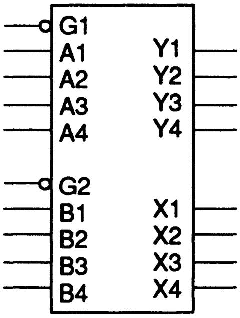
74240 Octal Inverting Buffers w/ 3-State Outputs 74240



74244 Octal Non-Inverting Buffers w/ 3-State Outputs 74244

- Two enable inputs
- 3-state outputs

Logic Symbol



Macrocell count:	8
Array inputs:	10
Product terms used:	8
Product terms allocated:	32

Functional Description

The 74244 macro contains two groups of four non-inverting buffers. Each group is enabled by an active-LOW input control line.

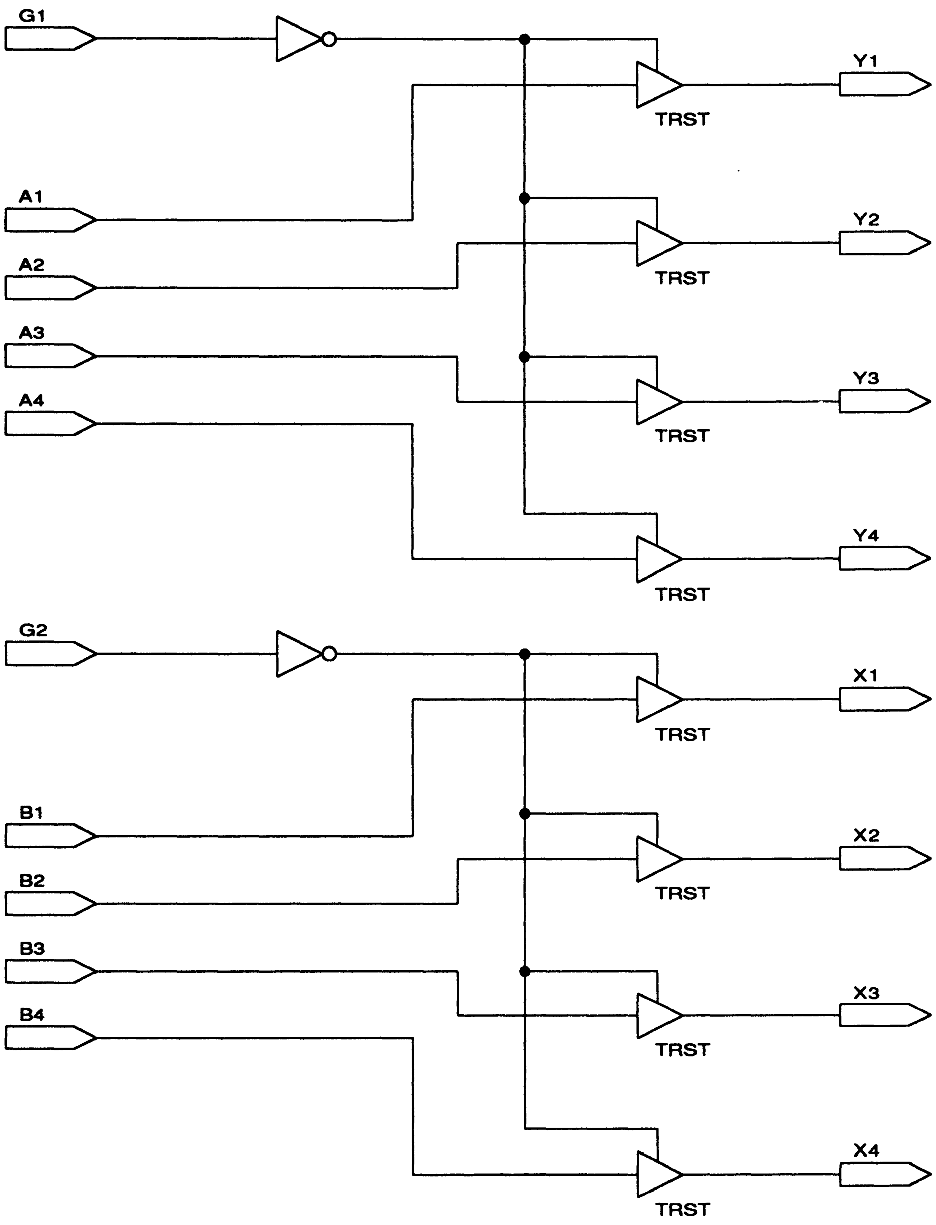
Sample PDS Equivalent

```
Y1.trst = /G1
Y2.trst = /G1
Y3.trst = /G1
Y4.trst = /G1
X1.trst = /G2
X2.trst = /G2
X3.trst = /G2
X4.trst = /G2
Y1 = A1
Y2 = A2
Y3 = A3
Y4 = A4
X1 = B1
X2 = B2
X3 = B3
X4 = B4
```

Function Table

Inputs										Outputs							
G1	A1	A2	A3	A4	G2	B1	B2	B3	B4	Y1	Y2	Y3	Y4	X1	X2	X3	X4
L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
H	X	X	X	X	L	H	L	H	H	Z	Z	Z	Z	H	L	H	H
L	L	H	H	L	H	L	H	H	H	L	H	H	L	Z	Z	Z	Z

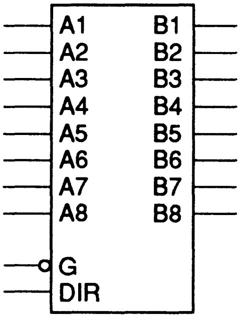
74244 Octal Non-Inverting Buffers w/ 3-State Outputs 74244



74245 Octal Bus Transceivers w/ 3-State Outputs 74245

- Enable input

Logic Symbol



Macrocell count:	16
Array inputs:	18
Product terms used:	16
Product terms allocated:	64

Functional Description

The 74245 macro implements an 8-bit bus transceiver. You can transmit data from bus A to bus B or from bus B to bus A. The data-transfer direction is controlled by the DIR control line. If the enable input, G, is set HIGH, then the buses are disabled and isolated.

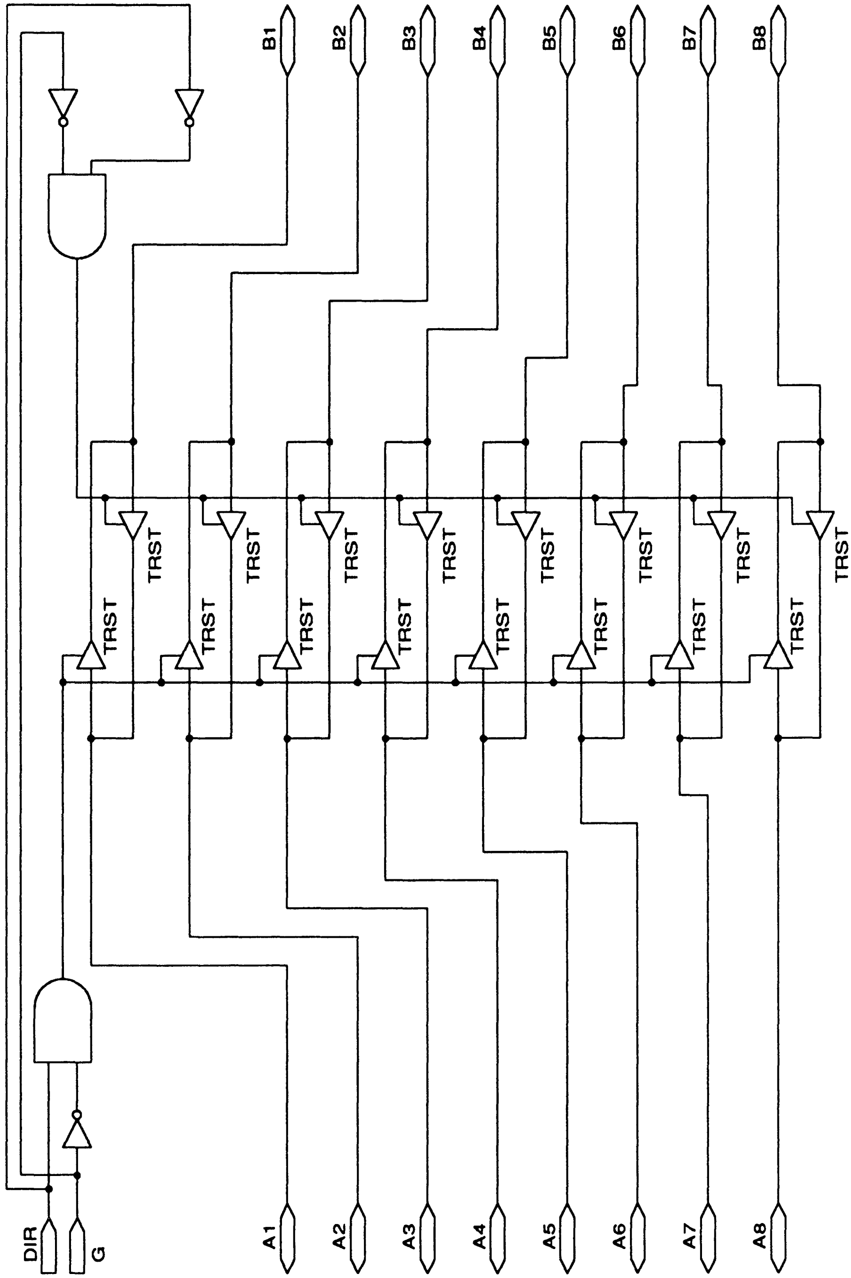
Sample PDS Equivalent

```
B1.trst = (DIR * /G)      B1 = A1
A1.trst = (/G * /DIR)     A1 = B1
B2.trst = (DIR * /G)      B2 = A2
A2.trst = (/G * /DIR)     A2 = B2
B3.trst = (DIR * /G)      B3 = A3
A3.trst = (/G * /DIR)     A3 = B3
B4.trst = (DIR * /G)      B4 = A4
A4.trst = (/G * /DIR)     A4 = B4
B5.trst = (DIR * /G)      B5 = A5
A5.trst = (/G * /DIR)     A5 = B5
B6.trst = (DIR * /G)      B6 = A6
A6.trst = (/G * /DIR)     A6 = B6
B7.trst = (DIR * /G)      B7 = A7
A7.trst = (/G * /DIR)     A7 = B7
B8.trst = (DIR * /G)      B8 = A8
A8.trst = (/G * /DIR)     A8 = B8
```

Function Table

Inputs		Operation
G	DIR	
L	L	Bus B Data to Bus A
L	H	Bus A Data to Bus B
H	X	Buses Isolated

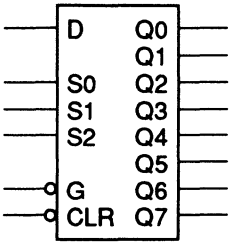
74245 Octal Bus Transceivers w/ 3-State Outputs 74245



74259 8-Bit Addressable Latch 74259

- Four modes of operation
- Asynchronous reset

Logic Symbol



Macrocell count:	16
Array inputs:	14
Product terms used:	16
Product terms allocated:	64

Functional Description

The 74259 macro is an 8-bit addressable latch with asynchronous reset logic. The following four modes of operation are selectable via the CLR and G inputs.

- Addressable latch
  Data on the D input line is written to the latch addressed by the three select lines: S2, S1, and S0. The other latches retain their values.

- Memory
  The latch outputs do not change.

- Active-HIGH 3-to-8 demultiplexer
  The addressed latch output follows the data input while the other latch outputs are held LOW.

- Reset
  All latch outputs are set LOW regardless of the value on the select and data-input lines.

Sample PDS Equivalent

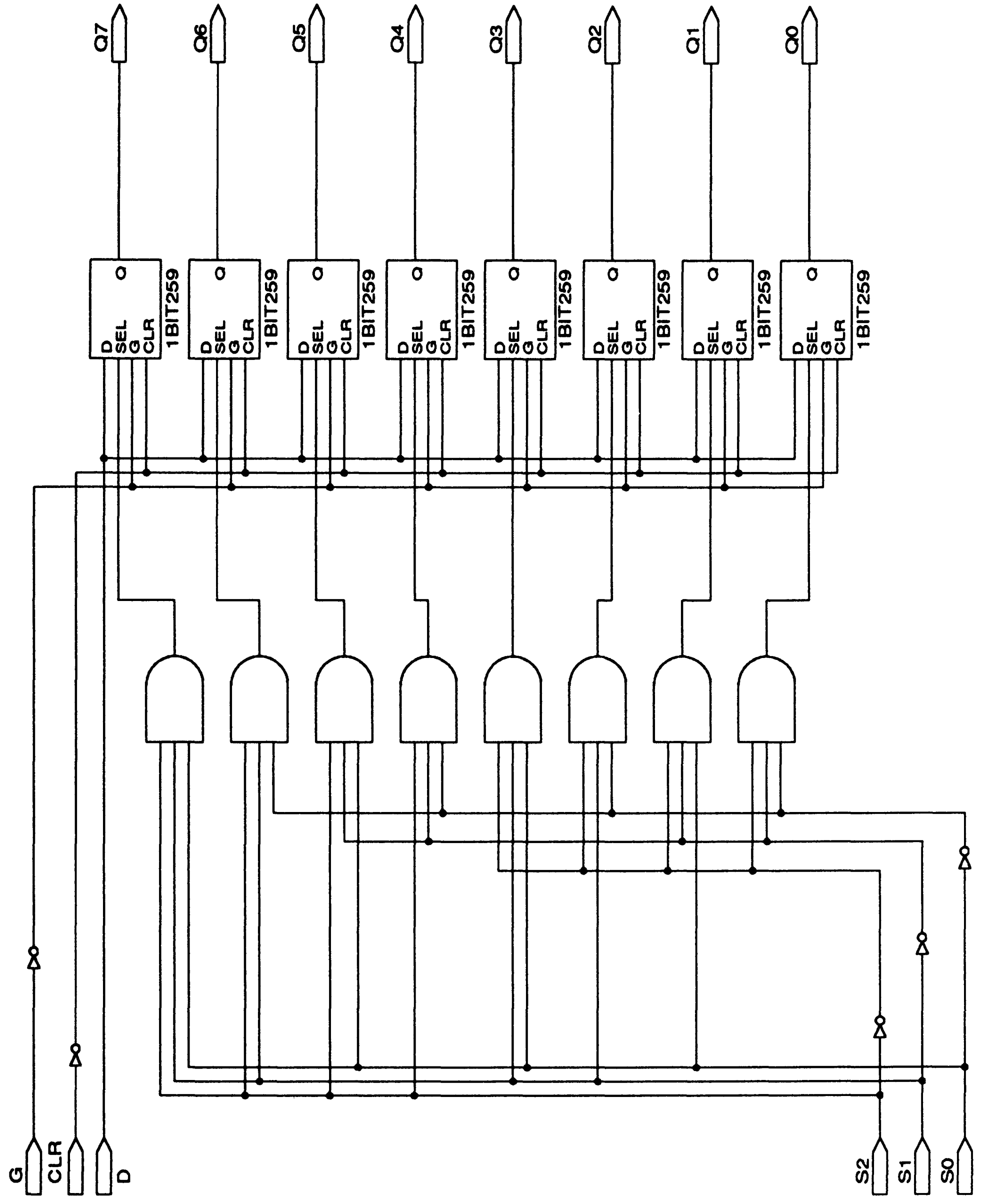
```
Q7 = (((S2 * S1 * S0) * /G * D) + (/(S2 * S1
   * S0) * CLR * Q7) + (CLR * Q7 * G) + (Q7
   * CLR * D)) Q6 = (((S2 * S1 * /S0) * /G * D)
   + (/(S2 * S1 * /S0) * CLR * Q6) + (CLR * Q6
   * G) + (Q6 * CLR * D))
Q5 = (((S2 * /S1 * S0) * /G * D) + (/(S2 * /S1
   * S0) * CLR * Q5) + (CLR * Q5 * G) + (Q5
   * CLR * D))  Q4 = (((S2 * /S1 * /S0) * /G
   * D) + (/(S2 * /S1 * /S0) * CLR * Q4) + (CLR
   * Q4 * G) + (Q4 * CLR * D))
Q3 = (((/S2 * S1 * S0) * /G * D) + (/(/S2 * S1
   * S0) * CLR * Q3) + (CLR * Q3 * G) + (Q3
   * CLR * D))  Q2 = (((/S2 * S1 * /S0) * /G
   * D) + (/(/S2 * S1 * /S0) * CLR * Q2) + (CLR
   * Q2 * G) + (Q2 * CLR * D))
Q1 = (((/S2 * /S1 * S0) * /G * D) + (/(/S2 * /S1
   * S0) * CLR * Q1) + (CLR * Q1 * G) + (Q1 * CLR
   * D))  Q0 = (((/S2 * /S1 * /S0) * /G * D) + (/(/S2
   * /S1 * /S0) * CLR * Q0) + (CLR * Q0 * G) + (Q0
   * CLR * D))
```

Function Table

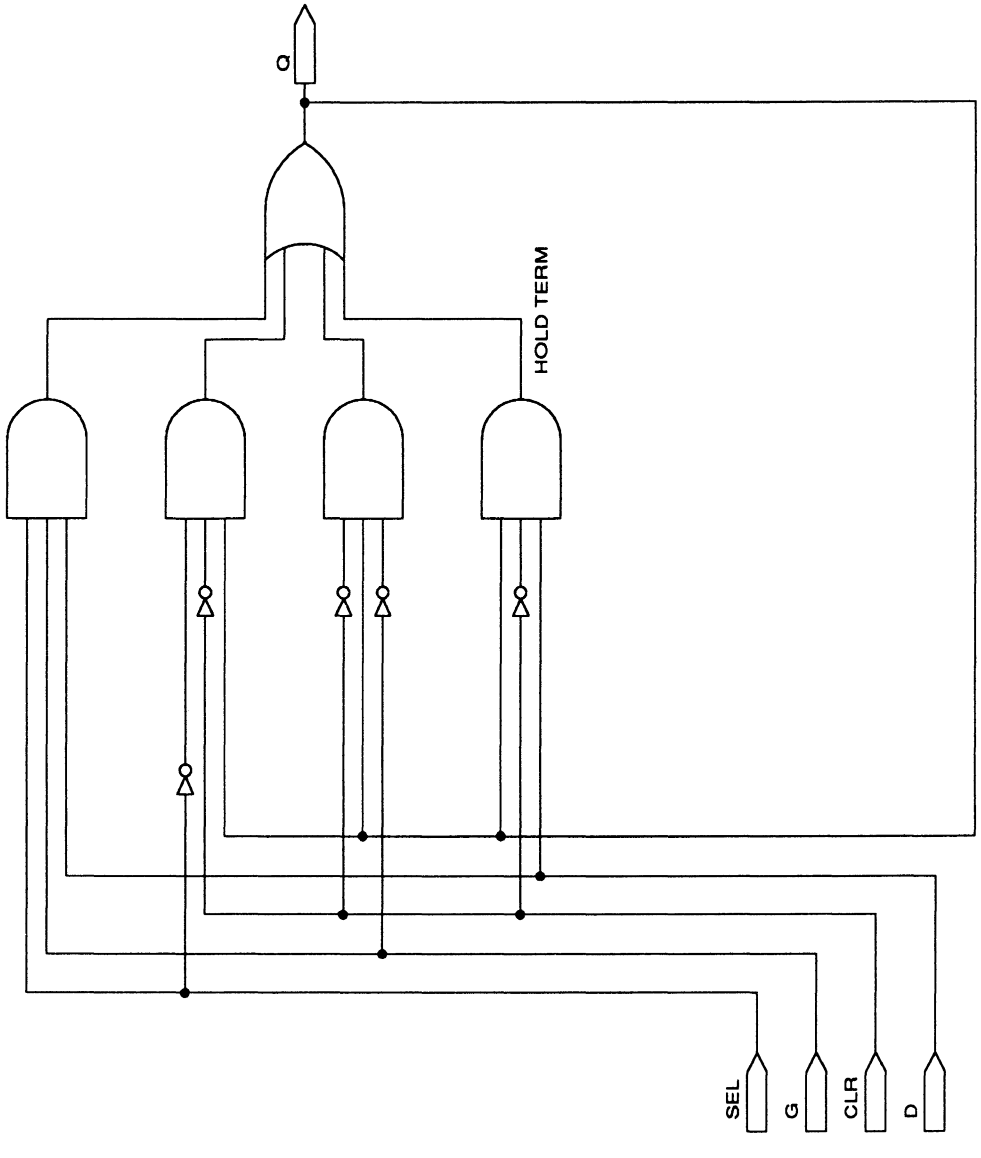
Inputs		Outputs		Functions
CLR	G	Adressed Latch	Other Latches	
H	L	D	Qo	Adressable Latch
H	H	Qo	Qo	Memory
L	L	D	L	8-line demultiplexer
L	H	L	L	Reset

* Qo = previous state of latch output Q
  D = data input D

8-Bit Addressable Latch

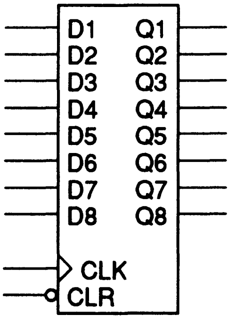


1Bit of a 74259 Latch



- Asynchronous reset

Logic Symbol



Macrocell count:	8
Array inputs:	9
Product terms used:	8
Product terms allocated:	32

Functional Description

The 74273 macro is an octal D-FF bank with asynchronous reset logic.

Sample PDS Equivalent

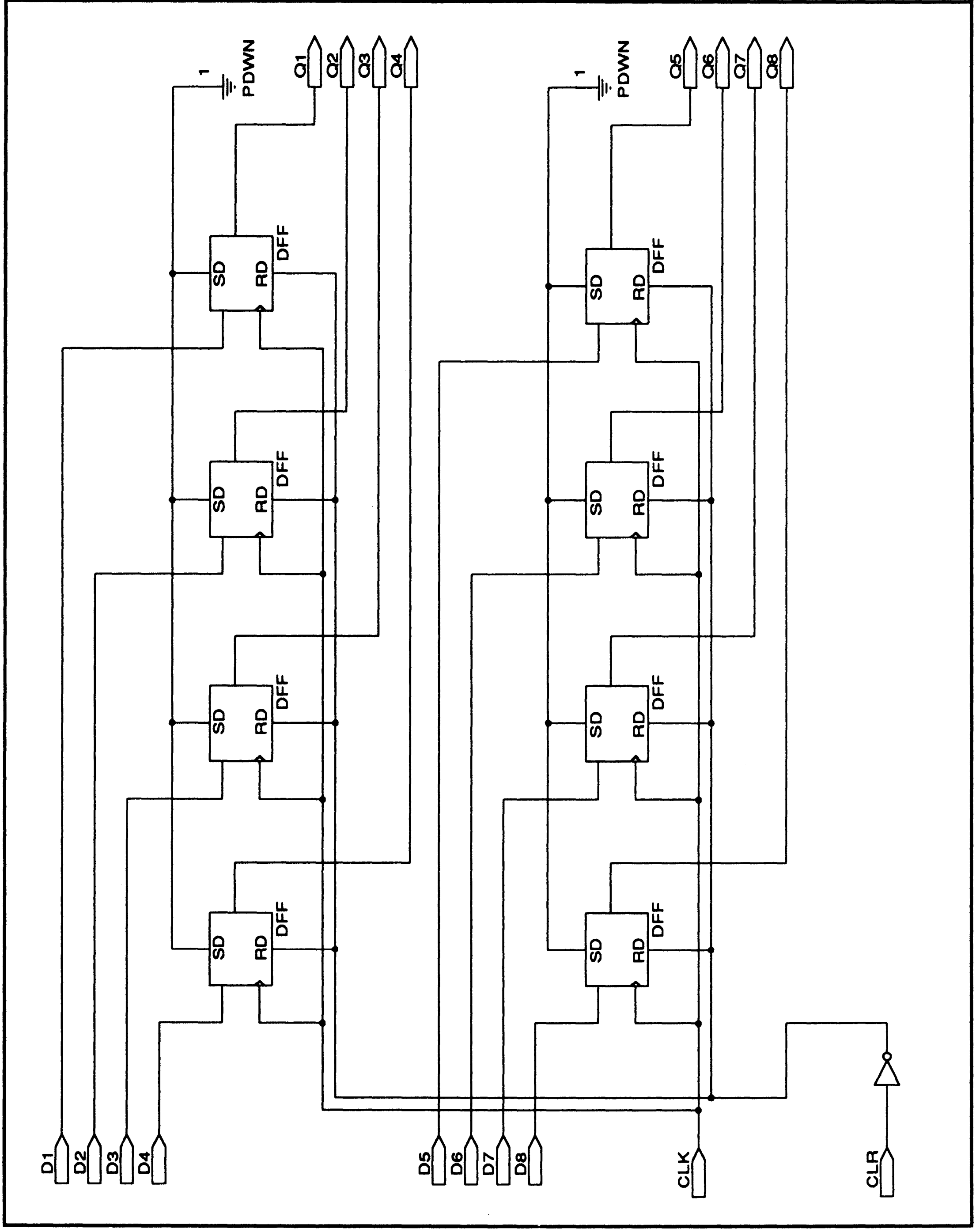
```
Q1 = D1
Q1.clkf = CLK
Q1.rstf = /CLR
Q2 = D2
Q2.clkf = CLK
Q2.rstf = /CLR
Q3 = D3
Q3.clkf = CLK
Q3.rstf = /CLR
Q4 = D4
Q4.clkf = CLK
Q4.rstf = /CLR
Q5 = D5
Q5.clkf = CLK
Q5.rstf = /CLR
Q6 = D6
Q6.clkf = CLK
Q6.rstf = /CLR
Q7 = D7
Q7.clkf = CLK
Q7.rstf = /CLR
Q8 = D8
Q8.clkf = CLK
Q8.rstf = /CLR
```

Function Table

Inputs			Outputs
CLR	CLK	D	Q
L	X	X	L
H	↑	L	L
H	↑	H	H
H	L	X	Qo

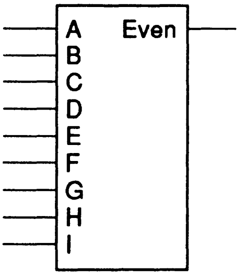
* Qo = previous state of Q

74273 Octal D-Type Flip-Flops 74273



- Active-HIGH output

Logic Symbol



Macrocell count:	4
Array inputs:	12
Product terms used:	16
Product terms allocated:	16

Functional Description

The 74280 macro is a combinatorial circuit that generates or checks for even parity on nine data lines. Odd parity is obtained by taking the inversion of the EVEN parity output.

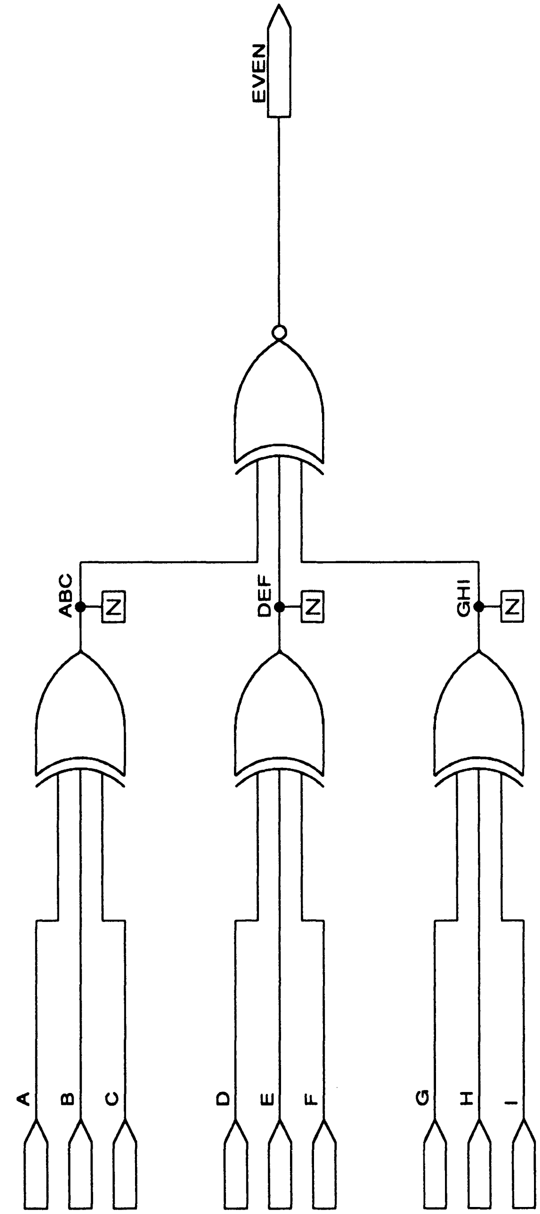
Sample PDS Equivalent

```
ABC = (A :+: B :+: C)
DEF = (D :+: E :+: F)
EVEN = /(ABC :+: DEF :+: GHI)
GHI = (G :+: H :+: I)
```

Function Table

Inputs	Outputs
Number of Inputs A - I that are High	**Even**
0, 2, 4, 6, 8,	H
1, 3, 5, 7, 9	L

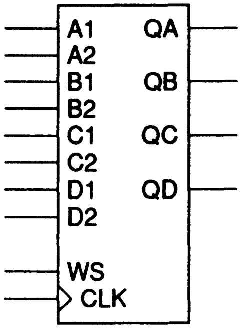
74280 9-Bit Parity Generator/Checker 74280



74298 Quad 2-to-1 Multiplexer with Storage 74298

- Synchronous storage

Logic Symbol



Macrocell count:	8
Array inputs:	21
Product terms used:	32
Product terms allocated:	32

Functional Description

The 74298 macro selects one of two 4-bit words and stores each bit in a register on the rising edge of CLK.

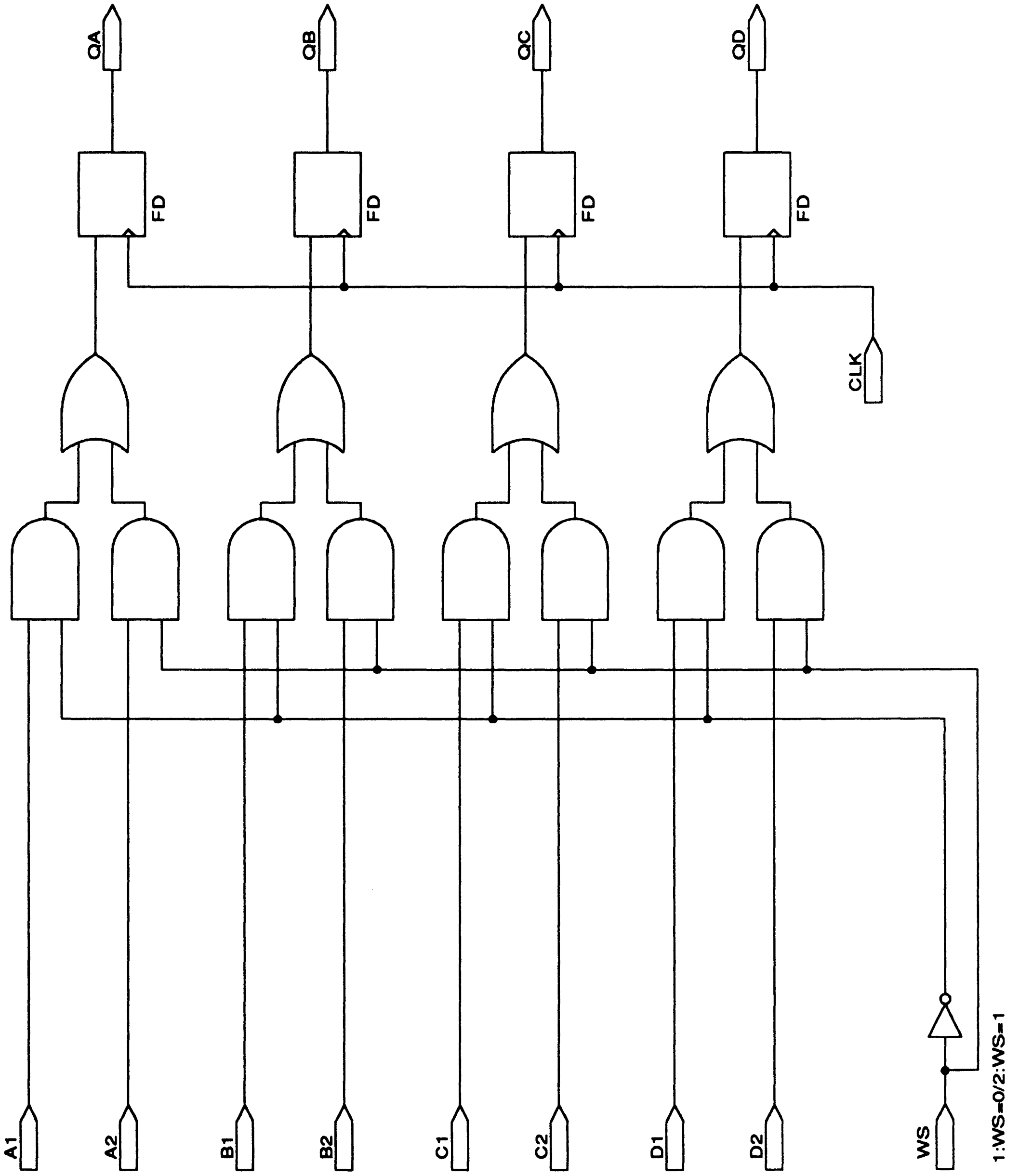
Sample PDS Equivalent

```
QA = ((A1 * /WS) + (A2 * WS))
QA.clkf = CLK
QB = ((B1 * /WS) + (B2 * WS))
QB.clkf = CLK
QC = ((C1 * /WS) + (C2 * WS))
QC.clkf = CLK
QD = ((D1 * /WS) + (D2 * WS))
QD.clkf = CLK
```

Function Table

Inputs		Outputs			
CLK	WS	QA	QB	QC	QD
L	X	QAo	QBo	QCo	QDo
↑	L	A1	B1	C1	D1
↑	H	A2	B2	C2	D2

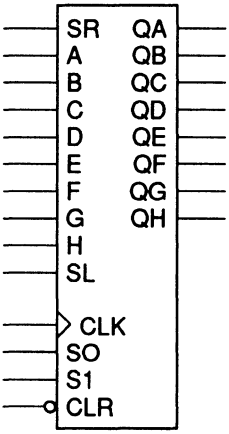
74298 Quad 2-to-1 Multiplexer w/ Storage 74298



74299X 8-Bit Bidirectional Universal Shift Register 74299X

- Parallel-to-serial converter
- Serial-to-parallel converter
- Synchronous reset
- Synchronous loading

Logic Symbol



Macrocell count:	8
Array inputs:	21
Product terms used:	32
Product terms allocated:	32

Functional Description

The 74299X macro is composed of two 74194s connected to form an 8-bit bidirectional universal shift register with synchronous reset logic.

Note:
The TTL version has three-state bidirectional I/Os that serve as the parallel-load inputs as well as the Q outputs.

Sample PDS Equivalent

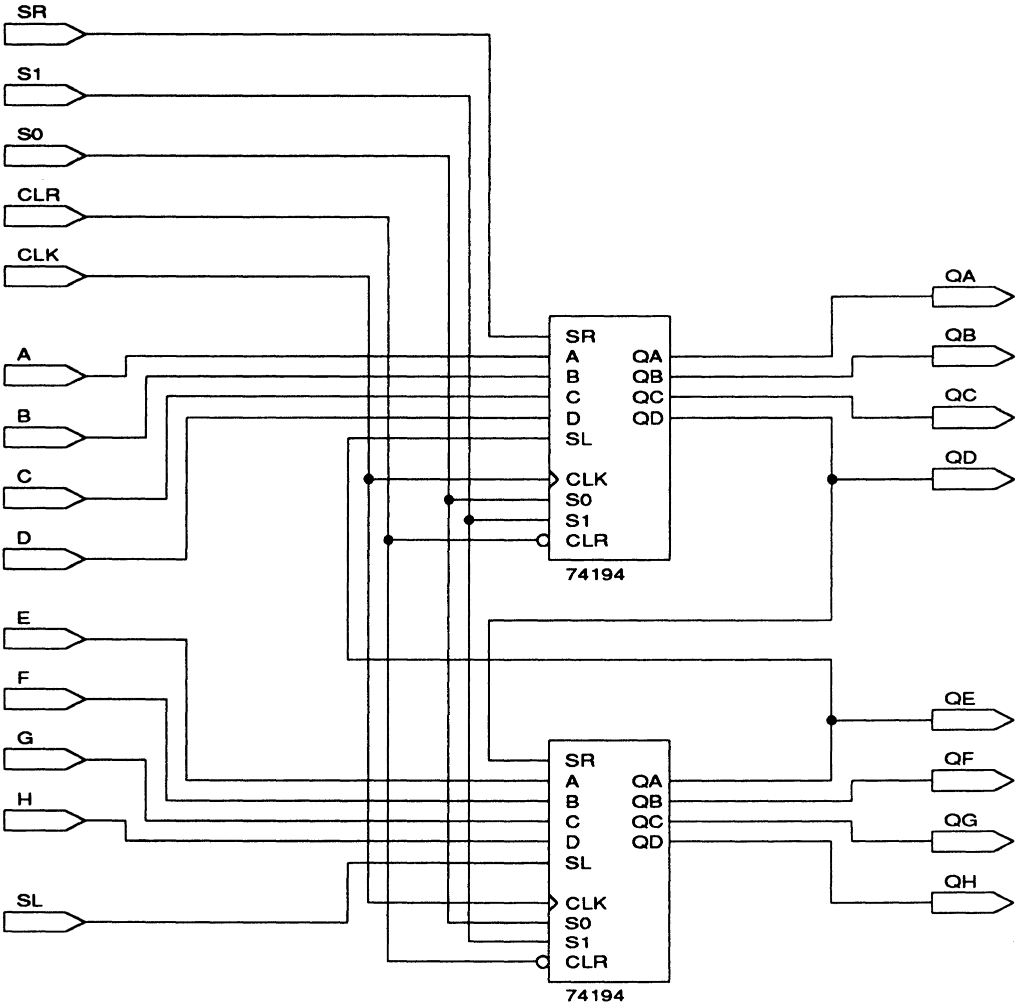
```
QA = ((CLR * SR * /S1 * S0) + (CLR * QB * S1 * /S0)
  + (CLR * S1 * S0 * A) + (CLR * /S1 * /S0 * QA))
QA.clkf = CLK
QB = ((CLR * QA * /S1 * S0) + (CLR * QC * S1 * /S0)
  + (CLR * S1 * S0 * B) + (CLR * /S1 * /S0 * QB))
QB.clkf = CLK
QC = ((CLR * QB * /S1 * S0) + (CLR * QD * S1 * /S0)
  + (CLR * S1 * S0 * C) + (CLR * /S1 * /S0 * QC))
QC.clkf = CLK
QD = ((CLR * QC * /S1 * S0) + (CLR * QE * S1 * /S0)
  + (CLR * S1 * S0 * D) + (CLR * /S1 * /S0 * QD))
QD.clkf = CLK
QE = ((CLR * QD * /S1 * S0) + (CLR * QF * S1 * /S0)
  + (CLR * S1 * S0 * E) + (CLR * /S1 * /S0 * QE))
QE.clkf = CLK
QF = ((CLR * QE * /S1 * S0) + (CLR * QG * S1 * /S0)
  + (CLR * S1 * S0 * F) + (CLR * /S1 * /S0 * QF))
QF.clkf = CLK
QG = ((CLR * QF * /S1 * S0) + (CLR * QH * S1 * /S0)
  + (CLR * S1 * S0 * G) + (CLR * /S1 * /S0 * QG))
QG.clkf = CLK
QH = ((CLR * QG * /S1 * S0) + (CLR * SL * S1 * /S0)
  + (CLR * S1 * S0 * H) + (CLR * /S1 * /S0 * QH))
QH.clkf = CLK
```

Function Table

Inputs										Outputs			
Mode		Serial				Parallel							
CLK	CLR	S1	S0	SL	SR	A	B	C	D	QA	QB	QC	QD
L	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo
↑	L	X	X	X	X	X	X	X	X	L	L	L	L
X	H	L	L	X	X	X	X	X	X	QAo	QBo	QCo	QDo
L	H	L	H	X	L	X	X	X	X	L	QAn	QBn	QCn
↑	H	L	H	X	H	X	X	X	X	H	QAn	QBn	QCn
↑	H	H	L	L	X	X	X	X	X	QBn	QCn	QDn	L
↑	H	H	L	H	X	X	X	X	X	QBn	QCn	QDn	H
↑	H	H	H	X	X	a	b	c	d	a	b	c	d

* QAo to QDo = previous state of QA to QD
QAn to QDn = level of QA to QD before the most recent rising transition of the CLK, and indicates a 1-bit shift.

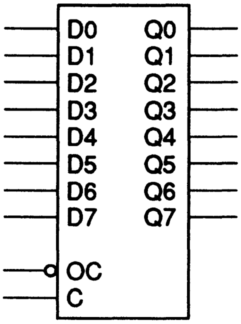
8-Bit Bidirectional Universal Shift Register



74373 Octal D-Type Latches with 3-State Outputs 74373

- Enable input

Logic Symbol



Macrocell count:	8
Array inputs:	18
Product terms used:	16
Product terms allocated:	32

Functional Description

The 74373 macro is an octal D latch with an active-LOW enable input.

Sample PDS Equivalent

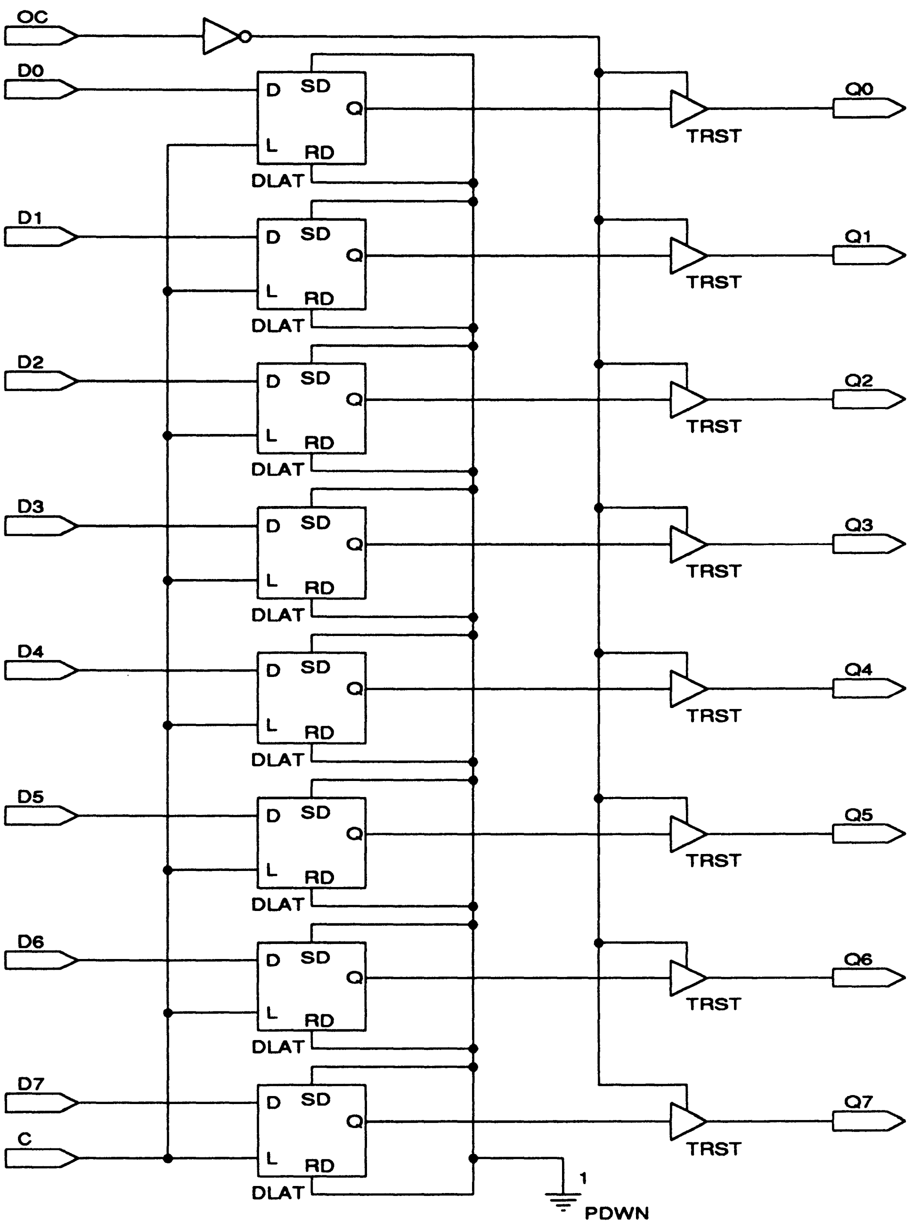
```
Q1 = ((Q1 * VCC * /C) + (VCC * GND) + (VCC * C * D1)
  + (D1 * VCC * Q1))
Q1.trst = /OC
Q2 = ((Q2 * VCC * /C) + (VCC * GND) + (VCC * C * D2)
  + (D2 * VCC * Q2))
Q2.trst = /OC
Q3 = ((Q3 * VCC * /C) + (VCC * GND) + (VCC * C * D3)
  + (D3 * VCC * Q3))
Q3.trst = /OC
Q4 = ((Q4 * VCC * /C) + (VCC * GND) + (VCC * C * D4)
  + (D4 * VCC * Q4))
Q4.trst = /OC
Q5 = ((Q5 * VCC * /C) + (VCC * GND) + (VCC * C * D5)
  + (D5 * VCC * Q5))
Q5.trst = /OC
Q6 = ((Q6 * VCC * /C) + (VCC * GND) + (VCC * C * D6)
  + (D6 * VCC * Q6))
Q6.trst = /OC
Q7 = ((Q7 * VCC * /C) + (VCC * GND) + (VCC * C * D7)
  + (D7 * VCC * Q7))
Q7.trst = /OC
Q8 = ((Q8 * VCC * /C) + (VCC * GND) + (VCC * C * D8)
  + (D8 * VCC * Q8))
Q8.trst = /OC
```

Function Table (for each D latch)

Inputs			Outputs
OC	C	D	Q
H	X	X	Z
L	H	H	H
L	H	L	L
L	L	X	Qo

* Qo = previous state of Q

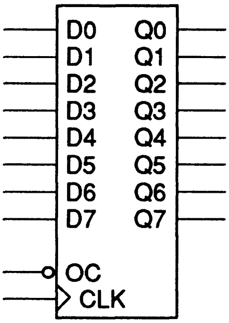
74373 Octal D-Type Latches w/ 3-State Outputs 74373



74374 Octal D-Type Flip-Flops with 3-State Outputs 74374

- Enable input
- 3-state outputs

Logic Symbol



Macrocell count:	8
Array inputs:	9
Product terms used:	8
Product terms allocated:	32

Functional Description

The 74374 macro is an octal D-type register with an active-LOW enable input.

Sample PDS Equivalent

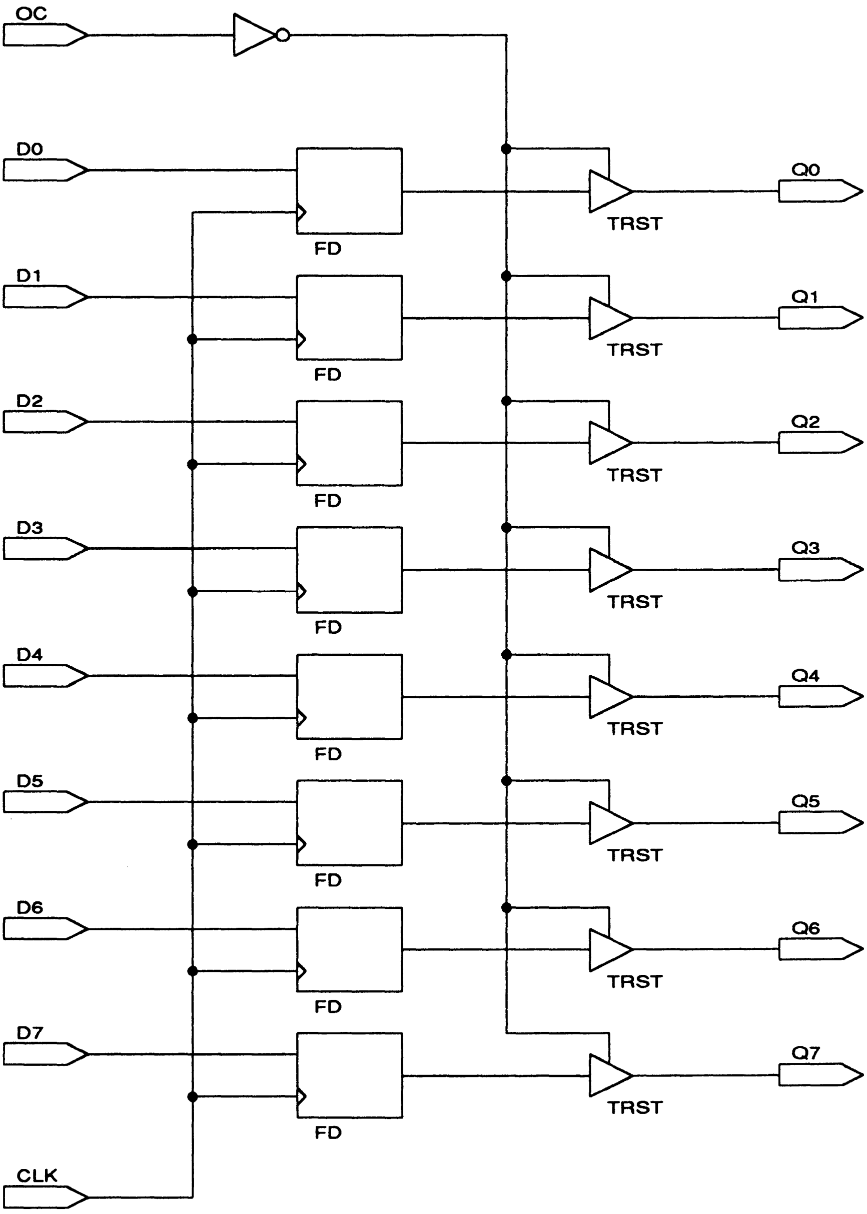
```
Q1.trst = /OC
Q1 = D1
Q1.clkf = CLK
Q2.trst = /OC
Q2 = D2
Q2.clkf = CLK
Q3.trst = /OC
Q3 = D3
Q3.clkf = CLK
Q4.trst = /OC
Q4 = D4
Q4.clkf = CLK
Q5.trst = /OC
Q5 = D5
Q5.clkf = CLK
Q6.trst = /OC
Q6 = D6
Q6.clkf = CLK
Q7.trst = /OC
Q7 = D7
Q7.clkf = CLK
Q8.trst = /OC
Q8 = D8
Q8.clkf = CLK
```

Function Table (for each D flip-flop)

Inputs			Outputs
OC	CLK	D	Q
H	X	X	Z
L	↑	H	H
L	↑	L	L
L	L	X	Qo

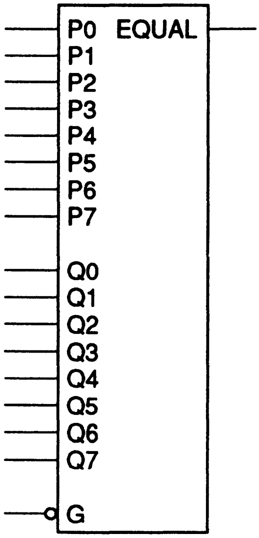
* Qo = previous state of Q

74374 Octal D-Type Flip-Flops w/ 3-State Outputs 74374



- Enable input
- Active-HIGH output

Logic Symbol



Macrocell count:	5
Array inputs:	21
Product terms used:	17
Product terms allocated:	20

Functional Description

The 74518 macro compares two 8-bit numbers and sets the EQUAL output HIGH if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

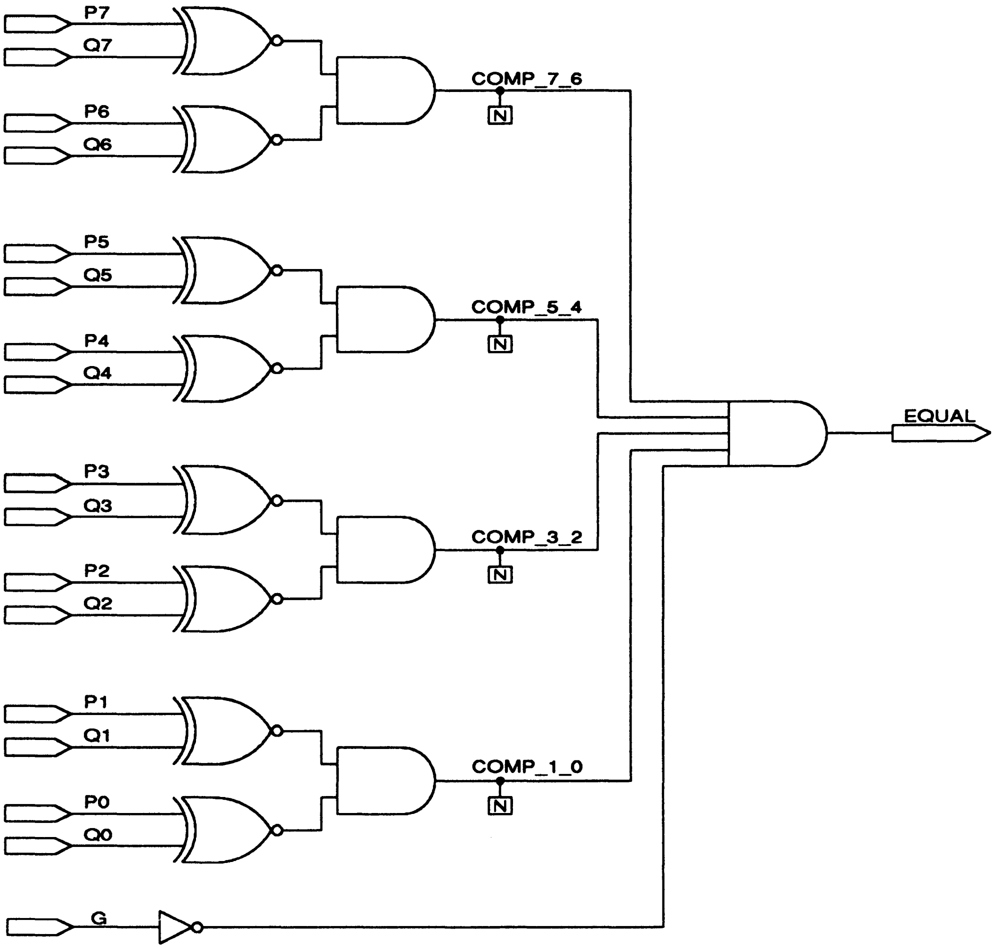
Sample PDS Equivalent

```
COMP_7_6 = (/(P7 :+: Q7) * /(P6 :+: Q6))
COMP_5_4 = (/(P5 :+: Q5) * /(P4 :+: Q4))
EQUAL = (COMP_7_6 * COMP_5_4 * COMP_3_2
  * COMP_1_0 * /G)
COMP_3_2 = (/(P3 :+: Q3) * /(P2 :+: Q2))
COMP_1_0 = (/(P1 :+: Q1) * /(P0 :+: Q0))
```

Function Table

Inputs		Outputs
DATA P, Q	G	EQUAL
P(7:0) = Q(7:0)	L	H
P(7:0) > Q(7:0)	L	L
P(7:0) < Q(7:0)	L	L
X	H	L

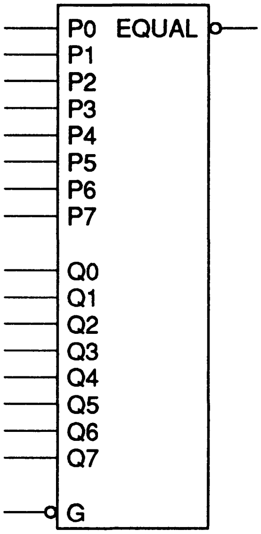
74518 8-Bit Identity Comparator 74518



74521 8-Bit Identity Comparator 74521

- Enable input
- Active-LOW output

Logic Symbol



Macrocell count:	5
Array inputs:	21
Product terms used:	17
Product terms allocated:	20

Functional Description

The 74521 macro compares two 8-bit numbers and sets the EQUAL output LOW if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

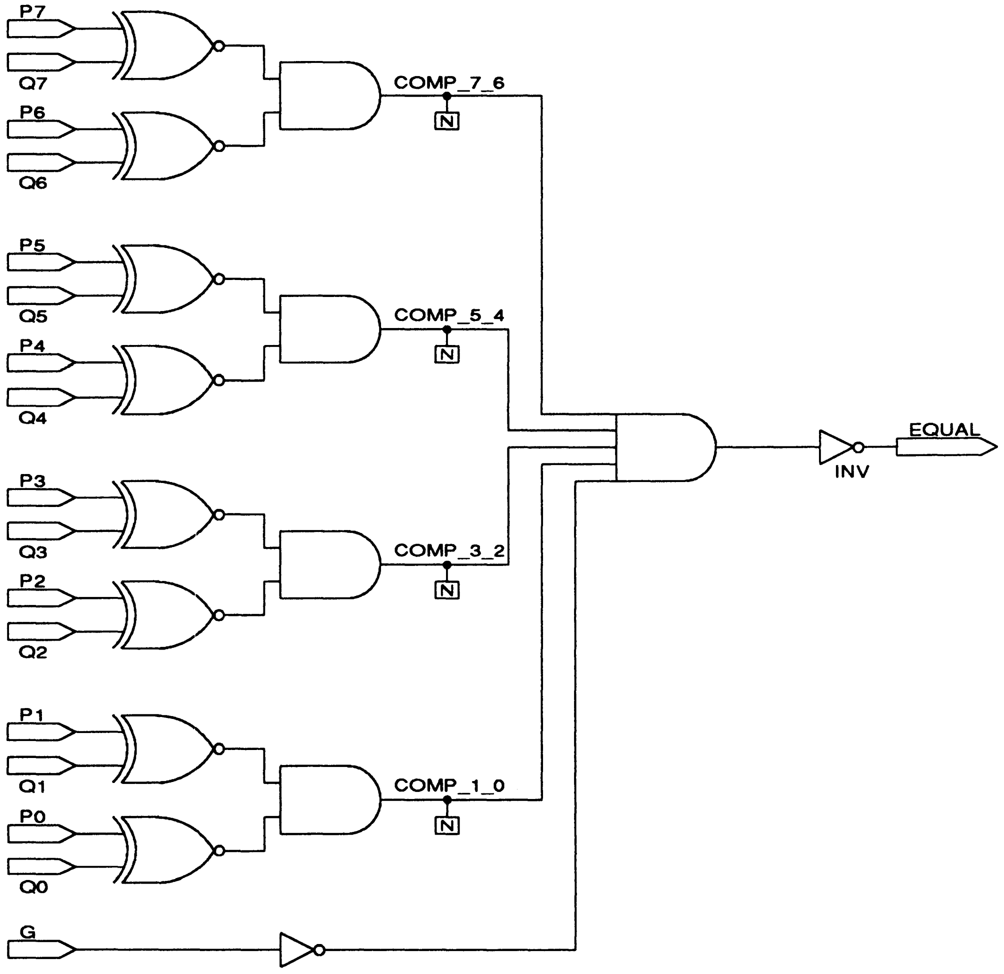
Sample PDS Equivalent

```
COMP_7_6 = (/(P7 :+: Q7) * /(P6 :+: Q6))
COMP_5_4 = (/(P5 :+: Q5) * /(P4 :+: Q4))
EQUAL = (COMP_7_6 * COMP_5_4 * COMP_3_2
  * COMP_1_0 * /G)
COMP_3_2 = (/(P3 :+: Q3) * /(P2 :+: Q2))
COMP_1_0 = (/(P1 :+: Q1) * /(P0 :+: Q0))
```

Function Table

Inputs		Outputs
DATA P, Q	**G**	**EQUAL**
P(7:0) = Q(7:0)	L	L
P(7:0) > Q(7:0)	L	H
P(7:0) < Q(7:0)	L	H
X	H	H

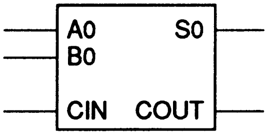
74521 8-Bit Identity Comparator 74521



ADD1 1-Bit Full Adder ADD1

- Carry input
- Carry output

Logic Symbol



Macrocell count:	2
Array inputs:	3
Product terms used:	7
Product terms allocated:	8

Functional Description

The ADD1 macro adds two 1-bit numbers. You can use the carry-out and carry-in signals to cascade multiple adders.

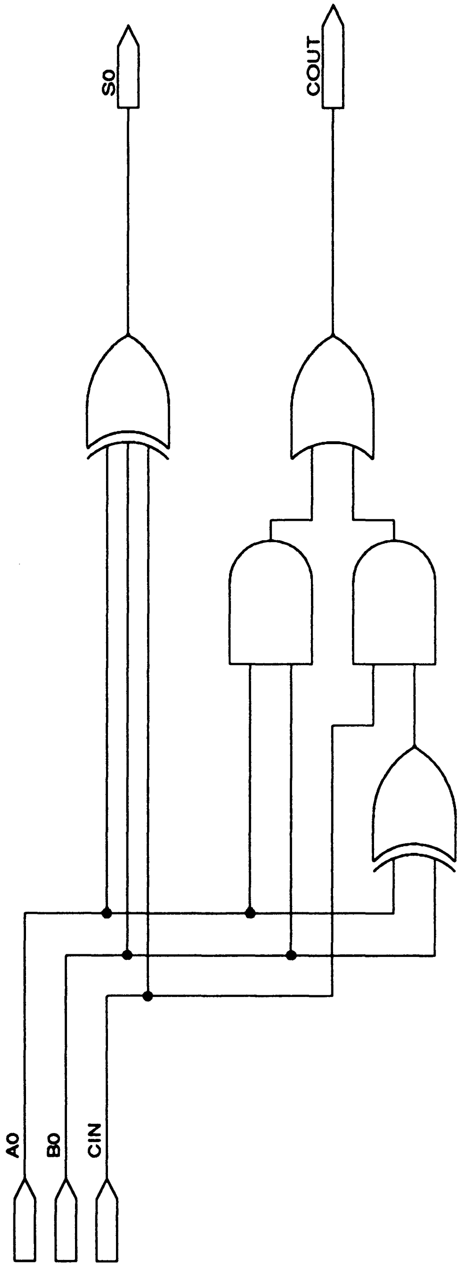
Sample PDS Equivalent

```
S0 = (A0 :+: B0 :+: CIN)
COUT = ((A0 * B0) + (CIN * (A0 :+: B0)))
```

Function Table

Inputs			Outputs	
A0	**B0**	**CIN**	**COUT**	**S0**
L	L	L	L	L
L	L	H	L	H
L	H	L	L	H
L	H	H	H	L
H	L	L	L	H
H	L	H	H	L
H	H	L	H	L
H	H	H	H	H

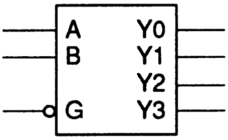
1-Bit Full Adder



2-to-4 Line Decoder

- Enable input

Logic Symbol



Macrocell count:	4
Array inputs:	3
Product terms used:	4
Product terms allocated:	16

Functional Description

The DECODE4 macro decodes one of four active-HIGH output lines depending on the 2-bit data inputs. The enable input, G, must be LOW to activate the decoder. You can use the enable inputs to cascade multiple decoders.

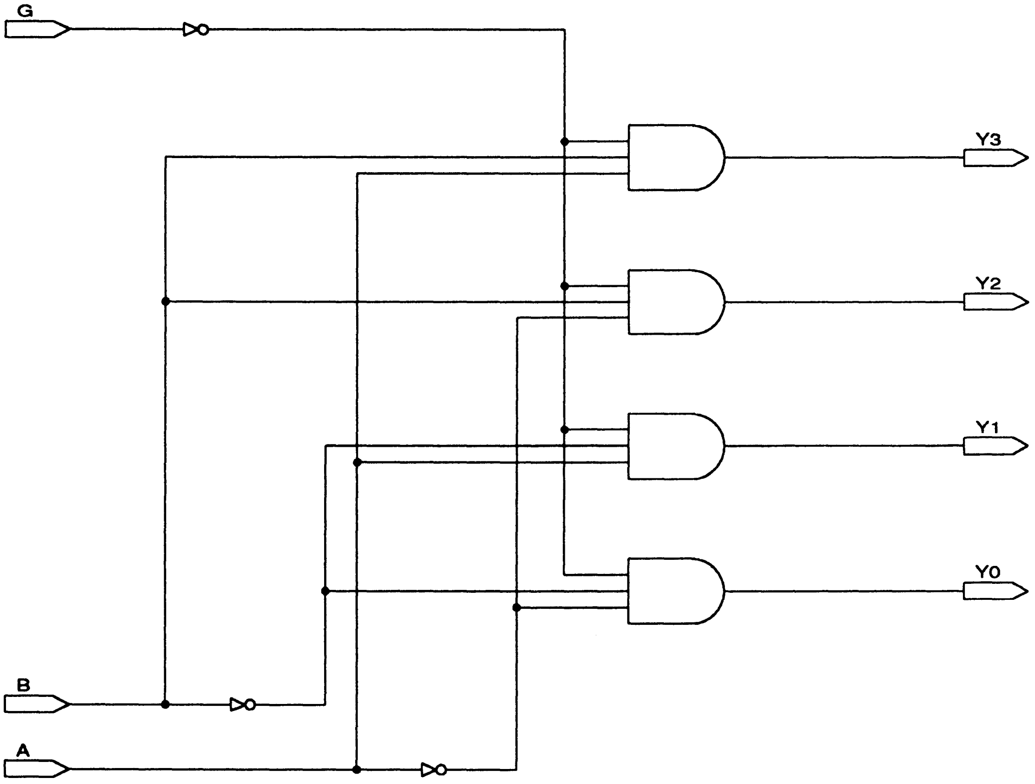
Sample PDS Equivalent

Y3 = (/G * B * A)
Y2 = (/G * B * /A)
Y1 = (/G * /B * A)
Y0 = (/G * /B * /A)

Function Table

Inputs			Outputs			
B	A	G	Y0	Y1	Y2	Y3
H	L	H	L	L	L	L
L	L	L	H	L	L	L
L	H	L	L	H	L	L
H	L	L	L	L	H	L
H	H	L	L	L	L	H

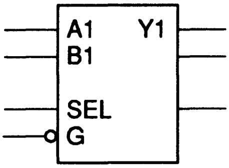
2-to-4 Line Decoder



MUX2 2-to-1 Multiplexer w/ Enable MUX2

- Enable input

Logic Symbol



Macrocell count:	1
Array inputs:	4
Product terms used:	2
Product terms allocated:	4

Functional Description

The MUX2 macro decodes one data-input line to select one of two data sources. The enable input, G, must be LOW to enable the Y1 output.

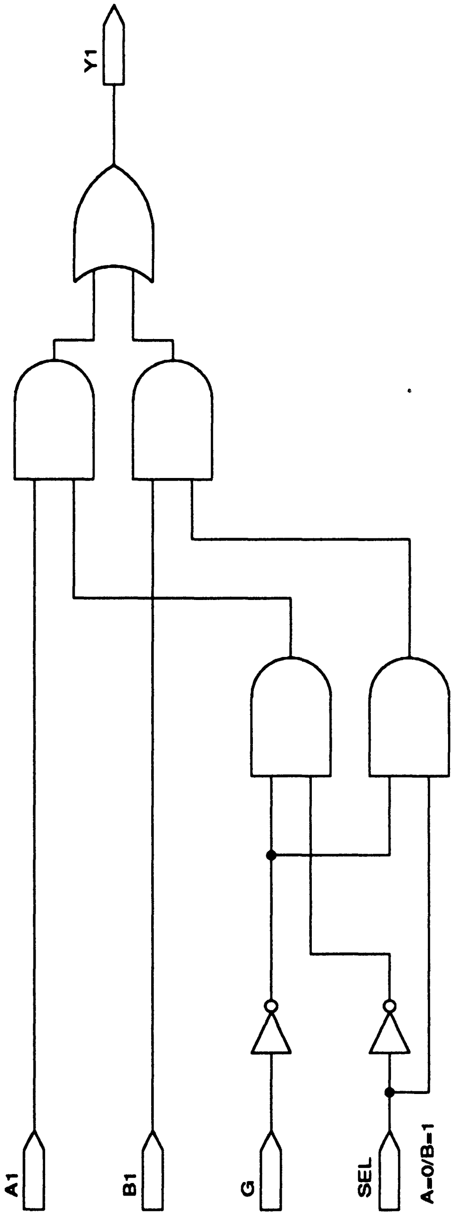
Sample PDS Equivalent

Y1 = ((A1 * (/G * /SEL)) + (B1 * (/G * SEL)))

Function Table

Inputs		Outputs
Select	Strobe	
Sel	G	Y1
X	H	L
L	L	A1
H	L	B1

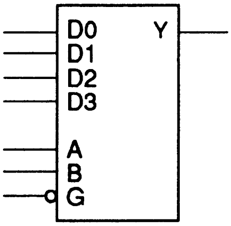
MUX2 2-to-1 Multiplexer w/ Enable MUX2



MUX4 4-to-1 Multiplexer w/ Enable MUX4

- Enable input

Logic Symbol



Macrocell count:	1
Array inputs:	7
Product terms used:	4
Product terms allocated:	4

Functional Description

The MUX4 macro decodes two data-input lines to select one of four data sources. The enable input, G, must be LOW to enable the Y output.

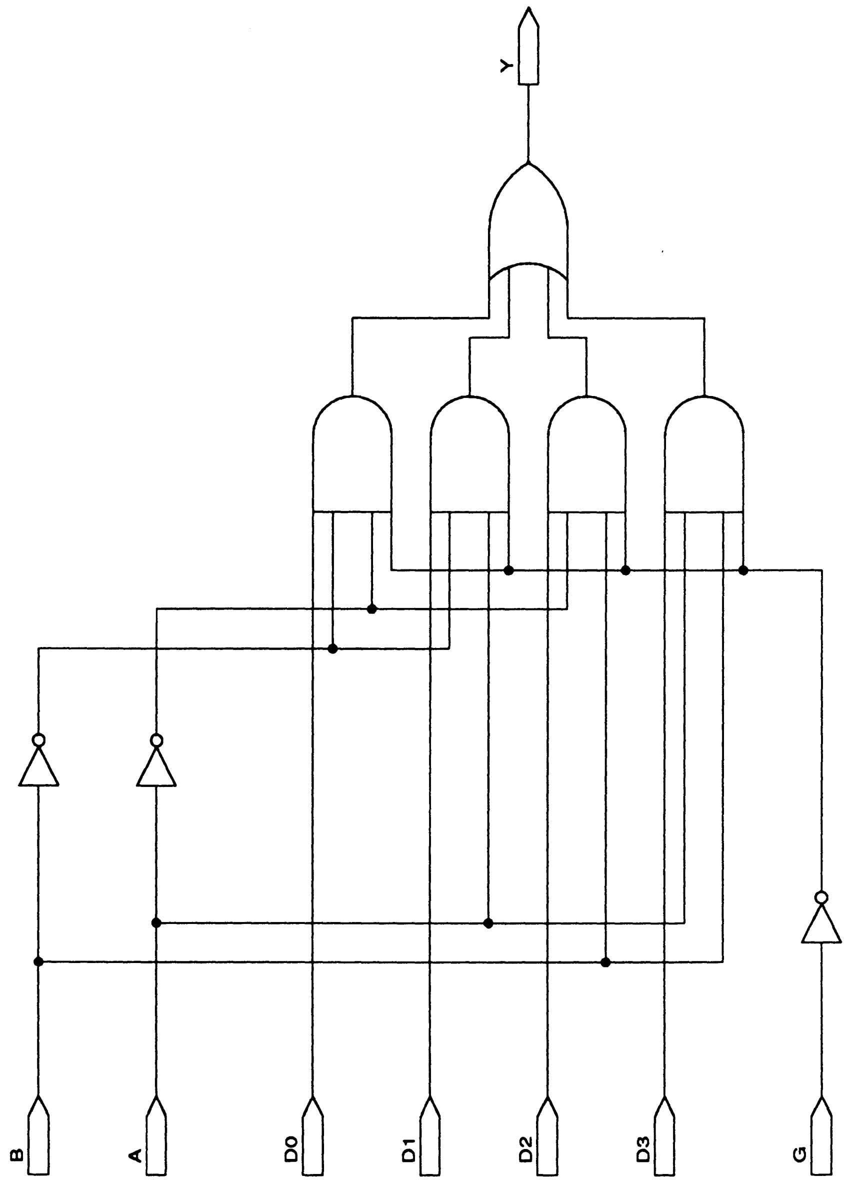
Sample PDS Equivalent

```
Y = ((D0 * /B * /A * /G) + (D1 * /B * A * /G)
   + (D2 * /A * B * /G) + (D3 * A * B * /G))
```

Function Table

Inputs			Outputs
Select		Strobe	
B	A	G	Y
X	X	H	L
L	L	L	D0
L	H	L	D1
H	L	L	D2
H	H	L	D3

4-to-1 Multiplexer w/ Enable



SECTION IV
SOFTWARE REFERENCE

Chapter 9

Menus and Commands

CONTENTS

9 MENUS AND COMMANDS

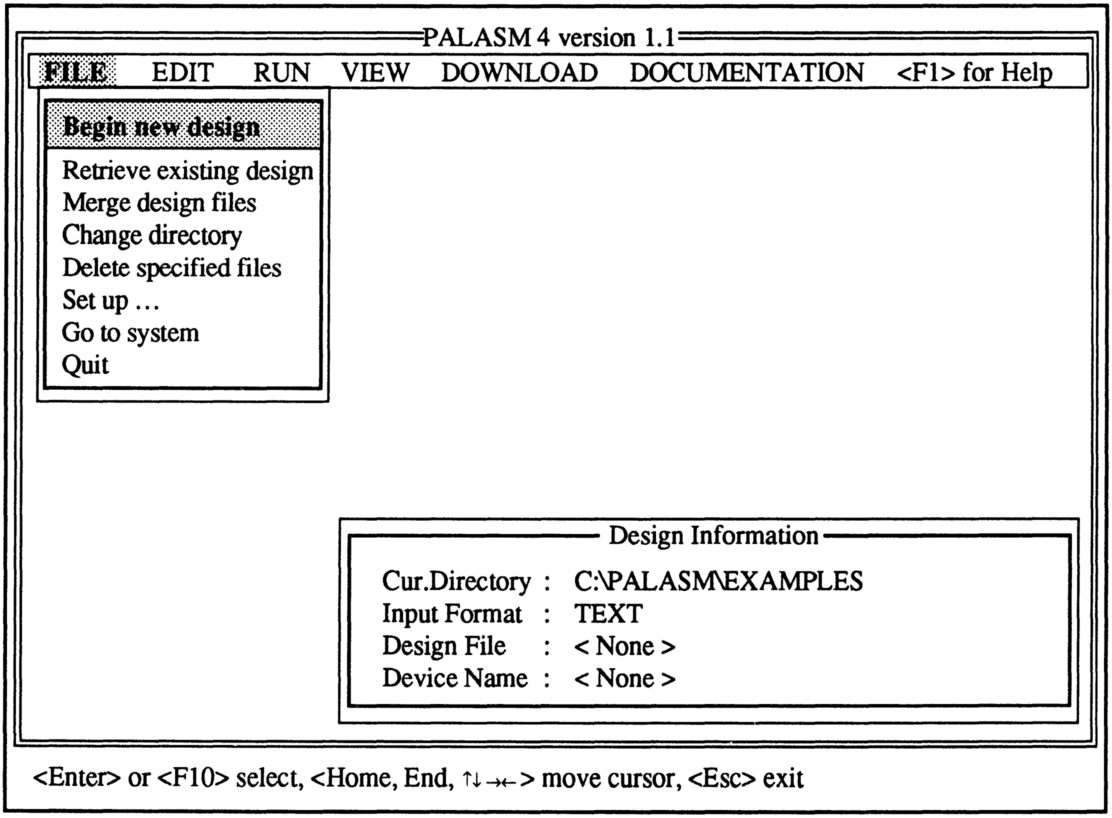
The *PALASM 4* software provides a unique environment that furnishes all commands required to develop a PLD or MACH-device design. This chapter is divided into two major topics.

- The overview, 9.1, introduces the features and conventions of the software.
- The commands and options discussion, 9.2, provides definitions and operational details for commands on the File, Edit, Run, View, Download, and Documentation menus.

9.1 OVERVIEW

Menus allow you to view and quickly select any command you need to produce and debug a PLD or MACH-device design. Using commands provided on menus, you can create or retrieve a design, process it, view and print reports, and download the JEDEC file to a device programmer.

The top-level screen is shown next. Various areas are identified and described after the figure.



Cur.Directory : C:\PALASM\EXAMPLES
Input Format : TEXT
Design File : < None >
Device Name : < None >

<Enter> or <F10> select, <Home, End, ↑↓→←> move cursor, <Esc> exit

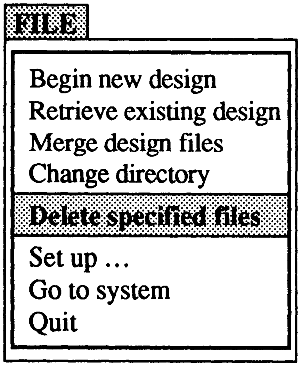
The bar across the top of the screen contains all **menu names**. A prompt on the right identifies how to access online Help. Each name reflects the kinds of commands on that menu.

- The **File** menu provides the file management, working environment, and system commands, as discussed under 9.2.1.

- The **Edit** menu provides commands to work on a particular kind of file in **either** the text editor **or** schematic editor, as described under 9.2.2.
- The **Run** menu lists all the commands you need to process a design file, as discussed under 9.2.3.
- The **View** menu includes commands to display files generated during each process, as described under 9.2.4.
- The **Download** menu provides access to the device programmer via a programmer-communication utility, as detailed under 9.2.5.
- The **Documentation** menu allows access to online reference material, as described under 9.2.6.

Depending on the working environment setup, which you define using the Set up command on the File menu, current design information appears in the lower-right corner of the screen. The status line at the bottom of the screen provides messages and prompts that change as needed.

9.1.1 FEATURES



Whether you're familiar with the environment or not, the menus and commands are easy to work with. The following features are standard.

- Drop-down-style menus like the one shown here
- Dialog-box-style forms
- Pop-up-style lists
- Keyboard invocation of commands

Commands, such as Set up, followed by ellipses, display a submenu of additional commands. When you select a command, one of three things may happen.

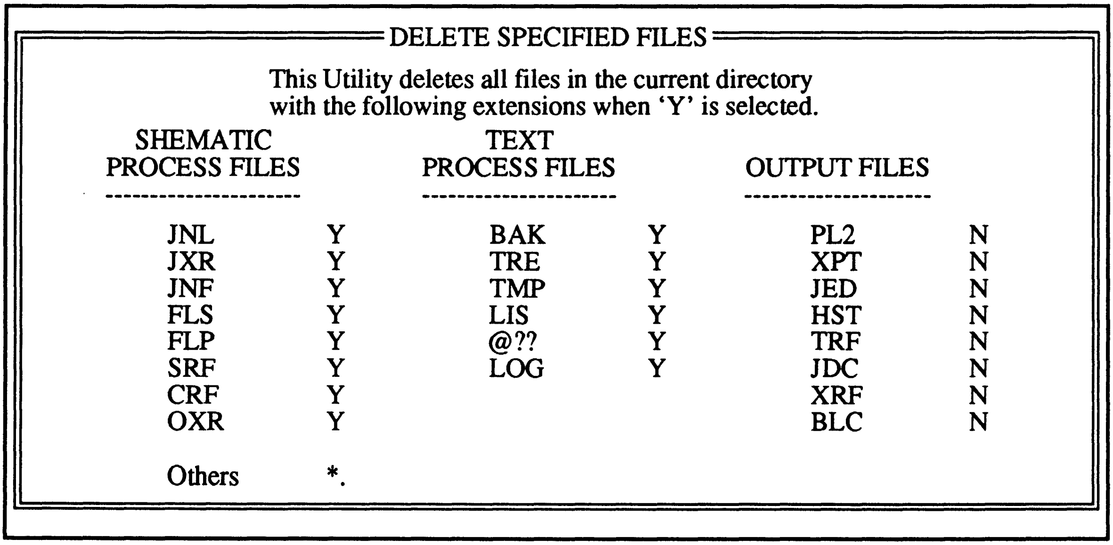
A. A process may be initiated, in which case, a window usually opens.

B. A submenu may appear listing additional commands for you to choose.

C. A form may appear where you supply additional information.

D. An option list appears only when you select an option field on a form.

A sample form is shown below. All forms presented in this chapter show the default options as they appear after first installing the software.



```
=================== DELETE SPECIFIED FILES ===================
        This Utility deletes all files in the current directory
        with the following extensions when 'Y' is selected.
   SHEMATIC                 TEXT
 PROCESS FILES          PROCESS FILES          OUTPUT FILES
 -------------          -------------          ------------

     JNL        Y           BAK        Y           PL2        N
     JXR        Y           TRE        Y           XPT        N
     JNF        Y           TMP        Y           JED        N
     FLS        Y           LIS        Y           HST        N
     FLP        Y           @??        Y           TRF        N
     SRF        Y           LOG        Y           JDC        N
     CRF        Y                                  XRF        N
     OXR        Y                                  BLC        N

     Others     *.
```

Each form provides one or more fields that typically contain information you can accept or change; the highlighted field is active. Most fields are composed of a field name and a corresponding specification. Three kinds of fields are provided: text, option, and status.

- You can type information, such as a file name, directly into the highlighted (active) **text** field.

- You press [F2] to display a list of choices for the active **option** field.

When you select an option, the list is dismissed and the specification on the form is updated.

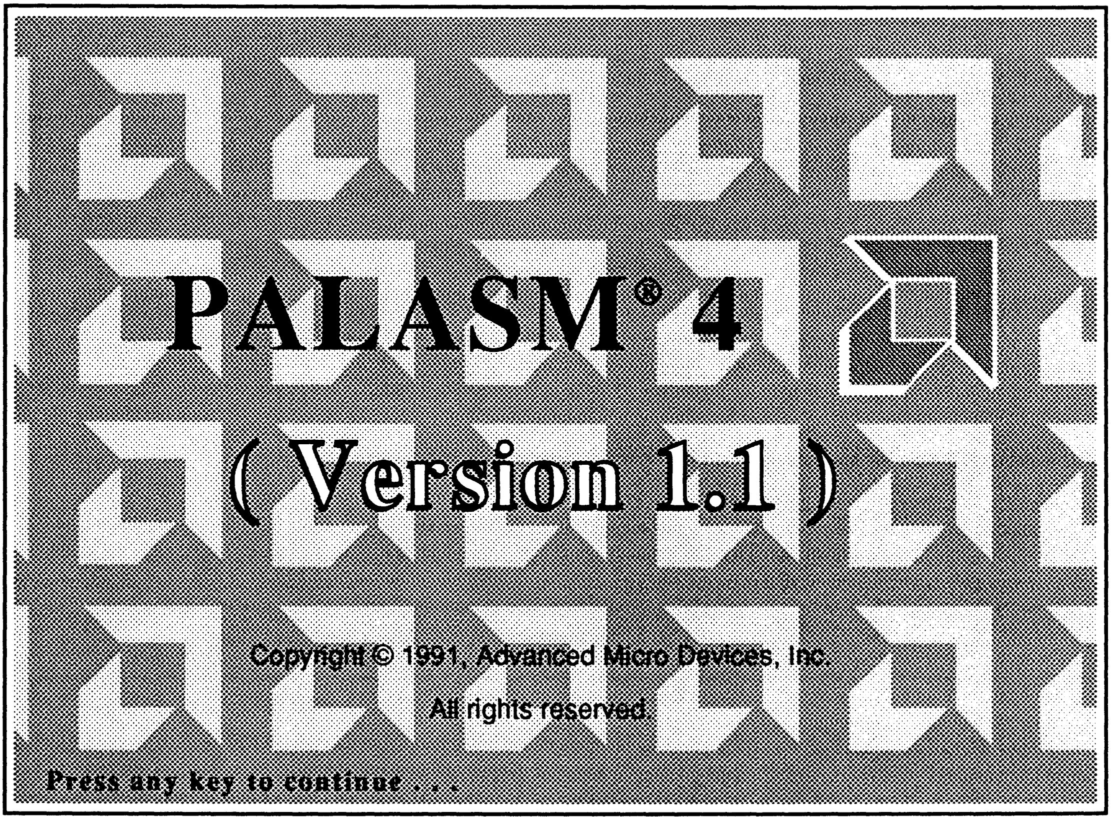
Note: An error is reported if you attempt to type into an option field.

- You cannot edit or change data in a **status** field. It's provided for information only.

9.1.2 CONVENTIONS

PALASM [Enter]

To use the software, you type the command shown in bold at left from the operating system. The copyright screen appears, as shown below.



PALASM® 4

(Version 1.1)

Copyright © 1991, Advanced Micro Devices, Inc.

All rights reserved.

Press any key to continue . . .

[Enter]

After you press a key to dismiss the copyright screen, the PALASM menus become available. At this point, you use the keyboard to select commands and options and to activate fields in forms.

Table 1 describes how you select commands from menus, submenus, and lists and how to fill in a form.

Table 1: Select a Command and Fill in a Form

TASK	KEYBOARD
Open / display a menu (select menu name)	Press arrow keys to highlight menu name.
Select a command from open menu, submenu, or list	Type the first letter of the command, which is capitalized, or press arrow keys to highlight command, then press [Enter].
Select a field / move to next or previous field	Press arrow keys to highlight field.
Display options	Press [F2].
Select option	Press arrow keys to highlight item, then press [Enter] to select item.
Enter text	Type new text.
Edit text	Move cursor and backspace or retype.
Cancel form or list / return to previous menu bar	Press [Esc].
Confirm specifications in a form	Press either [Enter] or [F10].

- When you enter a form, the first field is active unless it's a status field. You can enter data, change data, or select another field.

- When you leave a form, you're returned to the previous form, submenu, or menu. You can select another command or exit.

- When you return to a menu or submenu, the command associated with the form remains highlighted.

9.2 COMMANDS AND OPTIONS

Discussions are divided according to menu name or function, starting with the File menu at the left and moving across the menu bar to the right.

- 9.2.1, File Menu
- 9.2.2, Edit Menu
- 9.2.3, Run Menu
- 9.2.4, View Menu
- 9.2.5, Download Menu
- 9.2.6, Documentation Menu
- 9.2.7, [F1] for Help

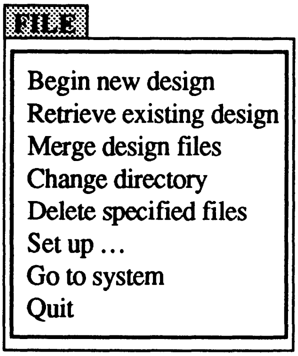
Within each discussion, commands are explained in logical order starting at the top of the menu and working through to the end.

- Any submenu or form that appears when you select a command is explained under the corresponding command discussion.

- Definitions for each field in a form are discussed in order, beginning at the top of the form and working through to the end.

- Choices for each field are discussed in order.

9.2.1 FILE MENU



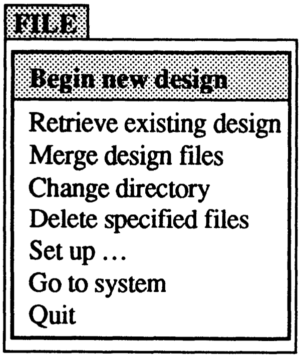
The File menu appears automatically when you enter the software environment. As shown on the left, this menu provides two kinds of commands.

- **File management commands**
  Begin new design
  Retrieve existing design
  Merge design files

- **Software-environment commands**
  Change directory
  Delete specified files
  Set up ...
  Go to system
  Quit

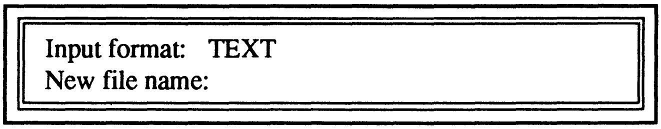
Depending on the working environment you define using the Set up command, current design information may appear in the lower-right corner of the screen.

9.2.1.1 Begin New Design



This command is automatically highlighted each time you enter the software environment. Each new file you create is stored in the current working directory.[1]

When you select the Begin new design command, a form appears so you can specify the file type and name.



Input format: TEXT
New file name:

[1] Refer to discussion 9.2.1.4, in this chapter, for details about changing the current working directory.

Input format: Text

This option field specifies the type of design you'll produce. Text refers to a text-based PDS file, which is the default.

To produce an OrCAD/SDT III schematic-based design, you must press [F2] to display the options and select Schematic.

Note: Schematic-base designs are supported only for MACH devices.

New file name:

You type the name, which must adhere to standard DOS naming conventions, in the new file name text field.

- Use any combination of upper- and/or lowercase letters, numbers, the underscore, _, and dollar sign, $, characters.

- Use up to eight, 8, characters and an optional extension: either .PDS for Boolean or state-machine designs or .SCH for schematic designs.

When you confirm your specifications, the name you specified is compared with existing file names.

- If the name corresponds to an existing file of the same type, you're asked if you want to overwrite the existing file.

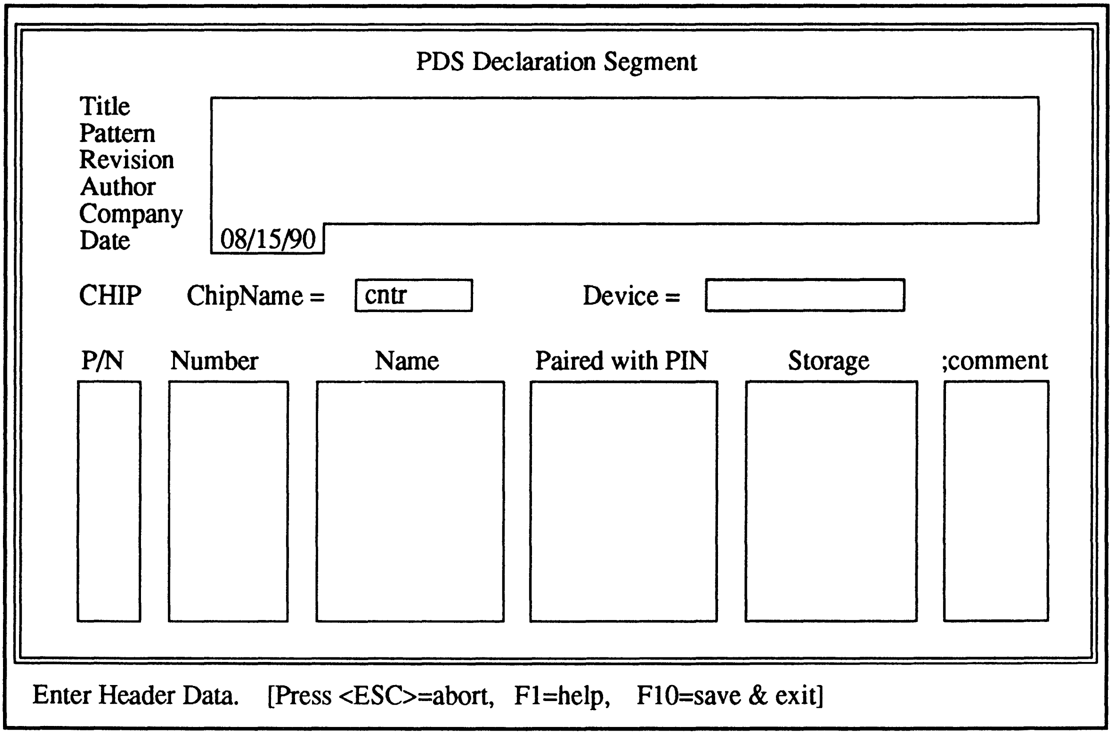
  In this case, you respond by typing the letter Y to overwrite the old file or the letter N.

- If the name is unique, one of two forms appears, depending on the kind of file you specified.

  Each form is described next.

Text-Based Design Form

After confirming a text-based format and design name, you're immediately transferred to the form shown below. The fields on this form assist you in completing the declaration segment of the PDS file.



PDS Declaration Segment

Title
Pattern
Revision
Author
Company
Date 08/15/90

CHIP ChipName = cntr Device =

P/N Number Name Paired with PIN Storage ;comment

Enter Header Data. [Press <ESC>=abort, F1=help, F10=save & exit]

The Title field is active when the form appears. You can **either** type a title **or** select a different field. The text and option fields on this form are described below.

- Title, Pattern, Revision, Author, and Company fields can contain up to fifty-nine, 59, characters, including any combination of alphanumeric characters or symbols.
- Date provides today's date automatically, as specified by the operating system, which you can change if you use the ##/##/## format.

- ChipName currently displays the design file name without the extension.

  You can specify a new chip name of up to eight, 8, alphanumeric characters.

- Device refers to the type of device for the design.

  Options include all PLD and MACH device types. You must specify a device type before saving information and leaving the form.

- P/N identifies the statement as either a pin or node statement.

  Options include a blank and two types: pin and node.

- Number requires a pin or node location, which can be **either** a whole number that fixes the location on the device **or,** for MACH devices only, a question mark, ?, that defines a floating location.

- Name requires a pin or node name.

- Paired with pin is an optional node attribute; if used, you must enter the number of the pin to which the node will be paired.[2]

- Storage refers to the optional storage type.

  Options include a blank and three types: Combinatorial, Latched, or Registered. Combinatorial is the default, which is used if this field is left blank.

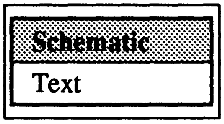
[2] Refer to Chapter 10, in this section, for details about the following topics: using the question mark to float pin and node locations, naming syntax in pin and node statements, and pairing a node with a pin.

- Comment adds an optional comment to the statement, which is preceded by a semicolon in the PDS file but not on the form.

  Options include Input, Output, IO, Clock, and Enable.

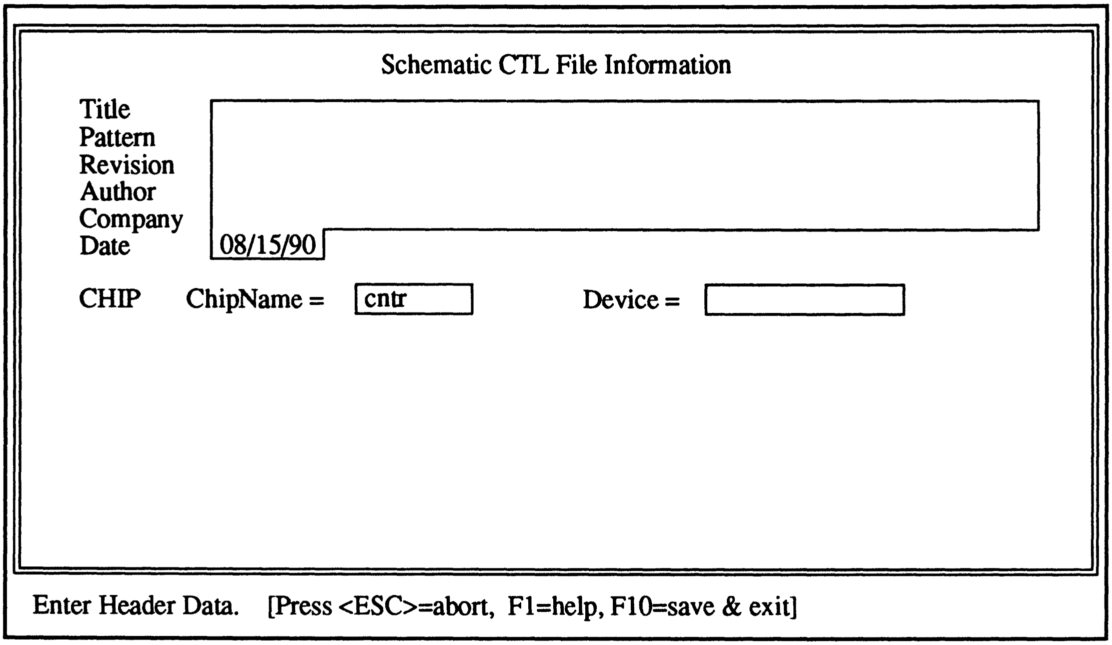
After you create and confirm specifications, you're transferred to the text editor and the PDS file is displayed.

Schematic-Based Design Form



Press [F2] and then press [Enter] to select a schematic-based format and design name. Two files are automatically created.

- An empty schematic worksheet file is created using the name you specified.
- An empty control file is created using the design name with a .CTL extension, design.CTL; then you're immediately transferred to the control-file form shown next.


Schematic CTL File Information

Title
Pattern
Revision
Author
Company
Date 08/15/90

CHIP ChipName = cntr Device =

Enter Header Data. [Press <ESC>=abort, F1=help, F10=save & exit]

When the schematic data is converted to a PDS format, the information in this form provides the declaration segment of the PDS file.

The Title field is active when the form appears. You can **either** type a title **or** select a different field. The text and option fields on this form are described next.

- Title, Pattern, Revision, Author, and Company fields can contain up to fifty-nine, 59, characters, including any combination of alphanumeric characters or symbols.
- Date provides today's date automatically, as specified by the operating system, which you can change if you use the ##/##/## format.
- ChipName currently displays the design file name without the extension.

  You can specify a new chip name of up to eight, 8, alphanumeric characters.

Important: Chip is a reserved word and cannot appear in any field, unless embedded in another word, such as ChipDate.

- Device, is provided where you specify the MACH device type for the design.[3]

You must specify a device type before saving information and leaving the form.

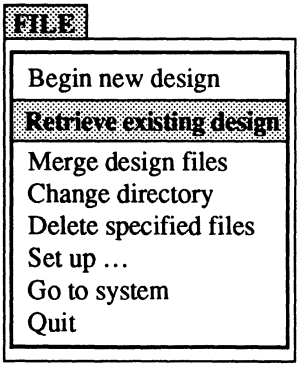
After you create and confirm specifications, you're automatically transferred to OrCAD/SDT III.[4] A blank worksheet with the name you specified earlier is available along with the AMD-supplied MACH library. You can begin placing symbols and wires to produce the schematic file.

Important: You must enter OrCAD/SDT III in this manner to use the AMD-supplied library for MACH-device designs.

[3] Refer to the *PALASM 4* online release notes for a listing of devices with JEDEC support.

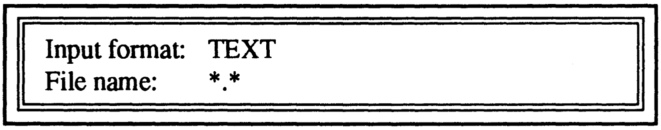
[4] Refer to the *OrCAD/SDT III Schematic Design Tools* manual for details about using the schematic editor.

9.2.1.2 Retrieve an Existing Design



You select this command and complete the form below to identify an existing design in the current working directory you want to edit or process.[5]

The form that appears is similar to the one you complete to create a new design file.



Input format: TEXT
File name: *.*

Input format: Text

This option field specifies the type of design you'll produce. Text refers to a text-based PDS file and is the default.

To edit or process an OrCAD/SDT III schematic-based design, you press [F2] to display the options and select Schematic.

File name:

You type the design name in this intelligent text field.

Note: Initially, the name field may be blank or may include *.*, however, once you create or retrieve a file, the form includes the name of the current design.

- If the field is blank, you can type a name.
- If the field contains *.*, a list of all file names appears when you press [Enter].

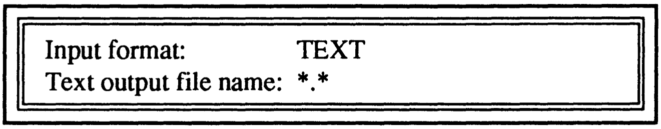
  You can enter *.PDS or *.SCH to display a list of specific files to select.

[5] Refer to discussion 9.2.1.4, in this chapter, for details about changing the current working directory.

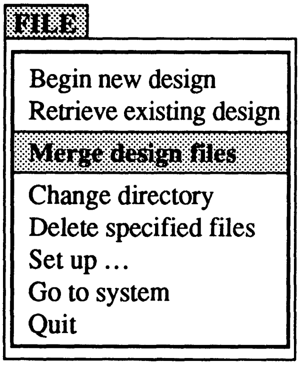
After you confirm your specifications, you can choose any command to specify the operation you want to perform. Depending on your working-environment setup, current design information may appear in the lower-right corner of the screen.

9.2.1.3 Merge Design Files

You select the Merge design files command and complete the form below to initiate a process where you can combine design files.[6] The form that appears is similar to the one you complete to create a new design.



Input format: TEXT
Text output file name: *.*



Input format: TEXT

You can combine only PDS files. Therefore, the Input format field on this form is a status field that you cannot activate or change.

Text output file name:

You type the name of the output file that will include all combined data in this field; the name must adhere to standard DOS conventions. The output file is stored in the current working directory.

Important: After you confirm the output file name, the merge process is initiated which includes compiling the design file and then the merge screen appears.

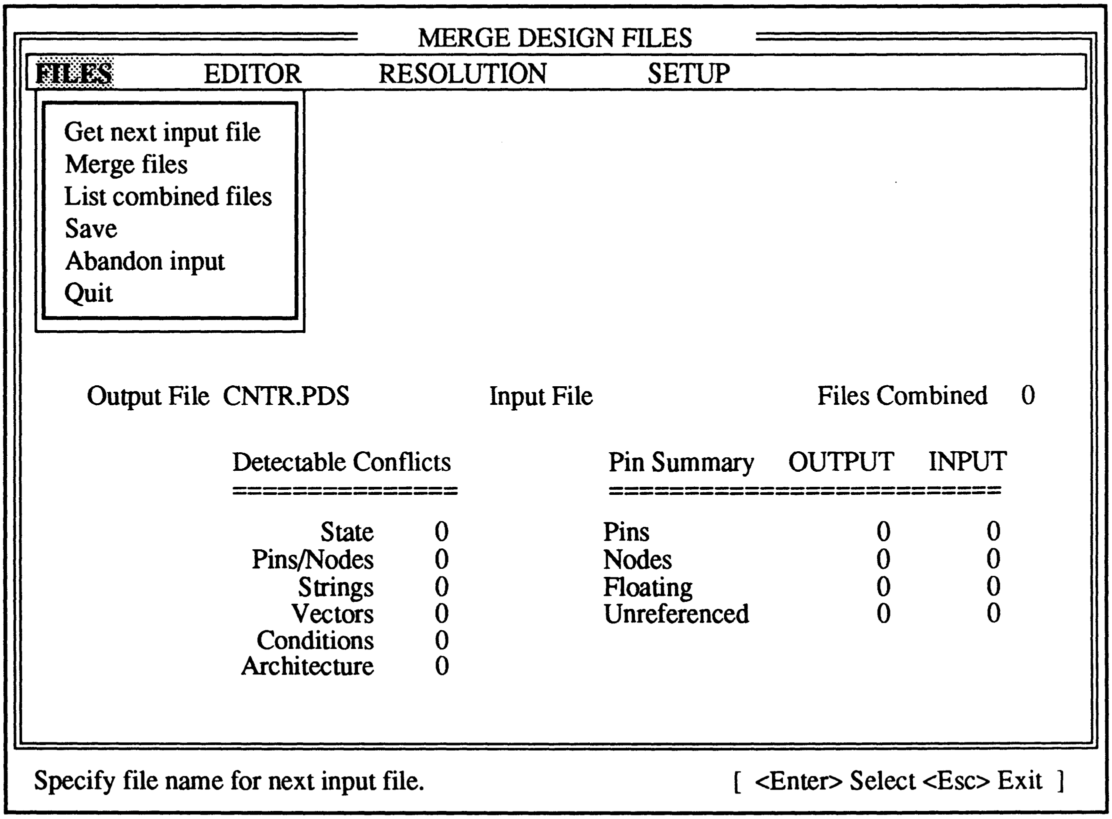
Four menus, Files, Editor, Resolution, and Setup, provide all commands for the merge process.

[6] Refer to Section II, Chapter 4, for guidelines to use when merging design files.

Status fields across the center of the screen identify the output file name, current input file name, and the number of files combined during this session.

Initially, the output file name is specified. However, the input file name is not listed because you have not yet retrieved the first input file.

The detectable-conflicts and pin-summary tables reflect the status of a comparison that's made after you retrieve an input file or resolve conflicts. Messages and prompts appear at the bottom of the screen as usual. The next figure shows the merge-process screen.



```
======================= MERGE DESIGN FILES =======================
 FILES        EDITOR        RESOLUTION        SETUP

  Get next input file
  Merge files
  List combined files
  Save
  Abandon input
  Quit

  Output File  CNTR.PDS          Input File             Files Combined   0

            Detectable Conflicts          Pin Summary    OUTPUT    INPUT
            ====================          ==========================
                   State        0         Pins               0        0
              Pins/Nodes        0         Nodes              0        0
                 Strings        0         Floating           0        0
                 Vectors        0         Unreferenced       0        0
              Conditions        0
            Architecture        0

 Specify file name for next input file.          [ <Enter> Select <Esc> Exit ]
```

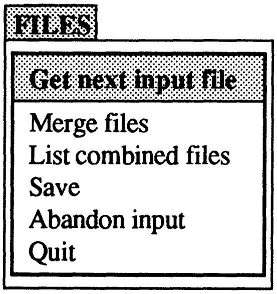
Initially, the input buffer is empty; the output buffer contains only empty declaration and equation segments.

Important: Following discussions define each merge-process command and all related forms and options. Menus are discussed in order from left to right; commands are discussed in order from first to last.

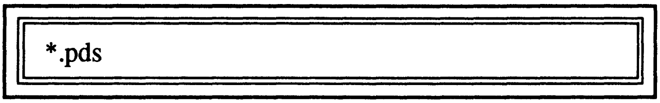
Files Menu

The merge process stores files temporarily in the input and output memory buffers. All commands on this menu, except Quit, operate on files in the memory buffers.

Get Next Input File



This command is highlighted when you begin the merge process. When you select this command, a form appears with the intelligent text field shown below.



*.pds

You can **either** type a file name **or** display a list of files and select a name from the list.

- If the form contains *.pds; press [F2] or [Enter] to display a list of all PDS files.
- If the form contains *.*, press [F2] or [Enter] to display a list of all files; however, you can only select a PDS file.

In any case, after you confirm the name, the following process is completed.

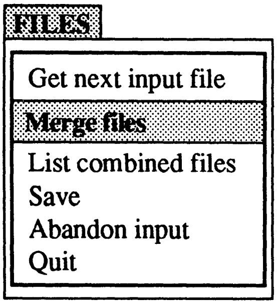
- Design data is loaded into the input buffer; the status line in the center of the screen reflects the name of the input file.
- The design is parsed, expanded, and minimized.

  If errors are detected, the input file is abandoned and the input buffer is cleared automatically.

- Data in the input buffer is compared with data in the output buffer.

  The pin-summary table reflects the status of the design in the input buffer. If design data is in the output buffer, the detectable conflicts table reflects the number of signal name or pin location conflicts between the two buffers.

Merge Files

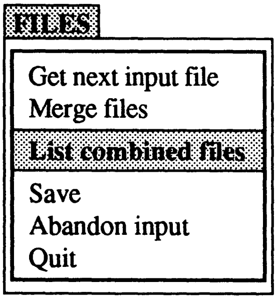


You select this command, after resolving conflicts, to move the input file into the output buffer. Data in the two buffers are combined into a single design in the output buffer; the input buffer is cleared.

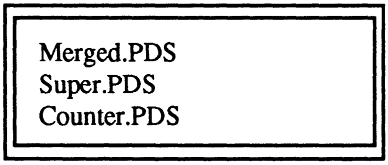
The status field in the center of the screen, which identifies the number of files combined during this session, increments by one.

> **Important:** If you initiate the Quit command before merging data in the input buffer with data in the output buffer, a warning appears and asks if you are sure you want to quit. In this case,
>
> - Y confirms you want to quit without merging data.
>
> - N cancels the quit command so you can merge and save the data.

List Combined Files



This command lists the names of all files you've combined during this session. You cannot select or edit any name in the list.

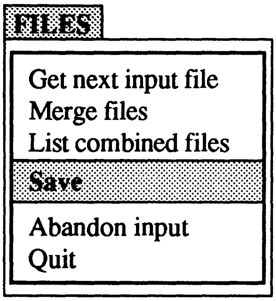


```
Merged.PDS
Super.PDS
Counter.PDS
```

Save

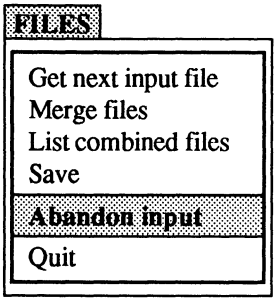
The Save command writes all data in the output buffer to the specified output file. Until you select this command, data resides only in a memory buffer.

Important: You must merge files to move data from the input buffer into the output buffer. Then save data in the output buffer to write it to the output file.



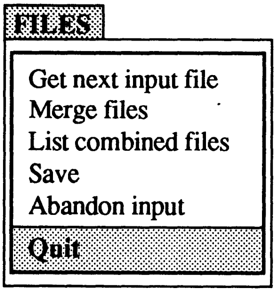
Abandon Input

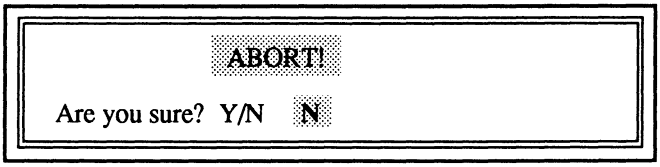
You use this command to clear the input buffer if you find the file is not appropriate to merge with data in the output buffer. The input-file status field in the center of the screen identifies the name of the input file; however, the field is cleared automatically either when the file is abandoned or after merging.



Quit

You use the Quit command to leave the merge process and return to the PALASM environment. When you select this command, you are asked to confirm ending the session.





ABORT!

Are you sure? Y/N N

- Y returns you to the PALASM environment.
- N cancels the Quit command.

Important: If you initiate the Quit command before merging data in the input buffer with data in the output buffer, a warning appears and asks if you are sure you want to quit. In this case,

- Y confirms you want to quit without merging data.

- N cancels the quit command so you can merge and save the data.

Also, if you initiate the Quit command before saving data in the output buffer, a warning states the design has changed since the last save and asks if you are sure you want to quit. In this case,

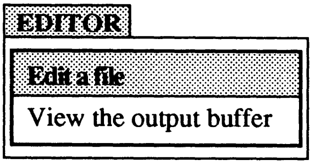
- Y confirms you want to quit without saving changes.

- N cancels the quit command so you can save the data.

Editor Menu

This menu provides two editor commands for the merge process. You cannot edit information in either the input or output buffer. However, you can edit any file and you can view information in the output buffer. The Resolution menu offers a command to edit the pin/node list in the output buffer.

Edit a File

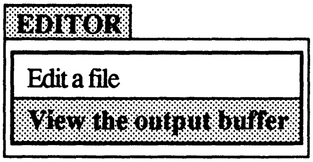


You select this command to correct design errors discovered when you retrieved the input file or to edit header data, device type, or pin locations in a combined design. When you select this command, a form appears containing an intelligent text field, as shown.

*.pds

You can **either** type a file name **or** display a list of files and select a name from the list. In either case, after you confirm the name, the text editor becomes available and the designated file is displayed on the screen.[7] To return to the merge process, you must quit from the text editor as usual.

View the Output Buffer



When you select this command, a view of the combined design in the output buffer appears on the screen. However, you cannot edit in view mode.

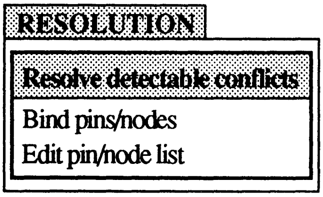
To return to the merge process from view mode, just press [Esc].

Resolution Menu

This menu provides commands to resolve conflicts between designs.

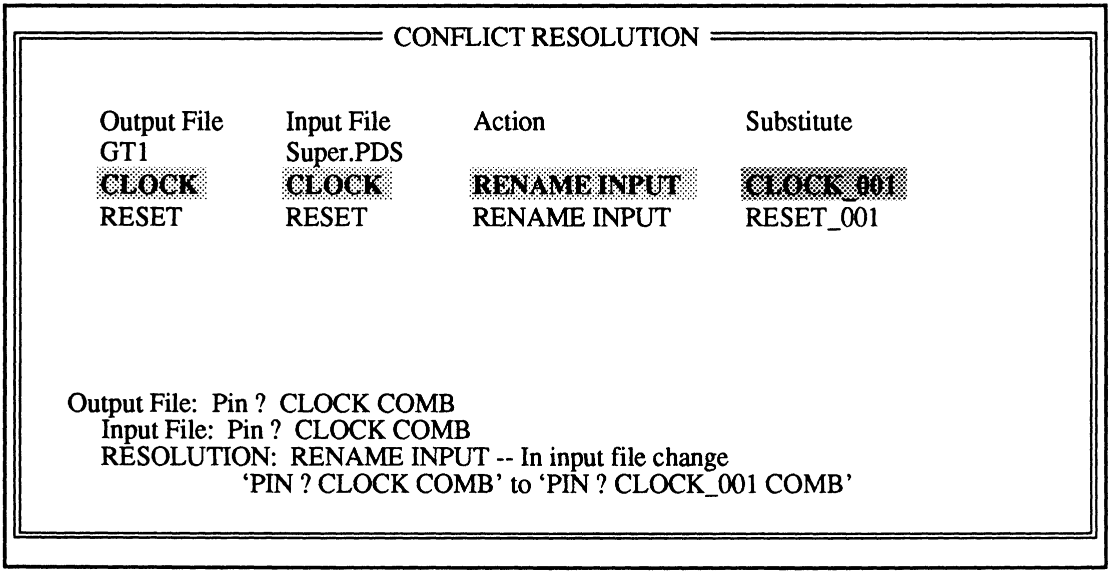
Recommendation: It's important to resolve conflicts before you merge the design in the input buffer with the design in the output buffer.

Resolve Detectable Conflicts



You use this command to display the conflict resolution form. Detectable conflicts occur when signals in the input and output buffer have the same name or pin number. These conflicts can be resolved from the conflict resolution form, shown next.

[7] Refer to Section V, Appendix A, for command definitions for the AMD-supplied text editor.



```
                    CONFLICT RESOLUTION

   Output File     Input File     Action           Substitute
   GT1             Super.PDS
   CLOCK           CLOCK          RENAME INPUT     CLOCK_001
   RESET           RESET          RENAME INPUT     RESET_001




Output File:  Pin ?  CLOCK COMB
   Input File:  Pin ?  CLOCK COMB
   RESOLUTION:  RENAME INPUT -- In input file change
               'PIN ? CLOCK COMB' to 'PIN ? CLOCK_001 COMB'
```

This form includes two columns with option fields and two columns with status fields.

- Status fields: Output File and Input File
- Option fields: Action and Substitute

Output File:

This column heading identifies the name under which combined data in the output buffer will be saved.

Each status field in this column identifies a signal name that conflicts with a signal in the input buffer. Only conflicting signals are listed. However, you cannot activate or edit status fields in this column.

Input File:

This column heading identifies the file in the input buffer.

Each status field in this column lists a signal name that conflicts with a signal in the output buffer. Again, you cannot activate or edit fields in this column.

Action

This column identifies the recovery for each signal conflict. The first item in this column is active when the form appears. Possible recovery actions include the following.

- Rename input
- Bind

The default action is to **rename** the signal in the input buffer; this option fills each row when the form appears. The name that will be used appears in the Substitute column.

> **Important**: When you intend to use separate signals, you must rename one.
>
> If you intend to use the same signal, you must bind them together using a common signal name.

To **bind** signals, you

- Press [Tab] to highlight the action field that corresponds to the pertinent conflict, then display the options as usual.

- Select Bind from the list.

  In this case, Bind appears in the action field and the name in the output buffer becomes the common name. You can change the common name as explained under Substitute.

Wildcard appears as an action in a field **when** you specify no floating input pins as a setup option **and** two pins are assigned to the same pin location on the device. In this case, a question mark is automatically assigned to the pin location in the input buffer. To restore the pin location specified in the input buffer, you must edit the pin/node list after combining the files.

Important: Wildcard is **not** available on the list of options.

Substitute

The fields in this column identify the name that will replace every instance of the signal name in the input buffer.

- If the action is to **rename** the input, the substitute is based on the naming strategy you specified using the Set renaming strategy command on the Setup menu.
- If the action is to **bind** signals together, the substitute name is taken from the design in the output buffer.

When the action is set to rename, you can change the substitute name by selecting this field and typing a new name.

Status information

Information at the bottom of the form identifies the exact pin or node statements in conflict and how the statement in the input buffer will change. For example, when you **rename** a signal the corresponding message reads as follows.

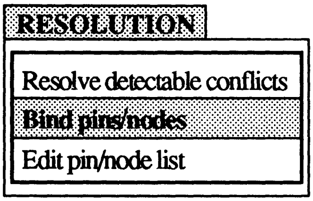
```
RESOLUTION:  RENAME INPUT -- In input file change
             'PIN ? CLOCK COMB' to 'PIN ? CLOCK_001
                  COMB'
```

When you **bind** signals together, the corresponding message reads as shown below.

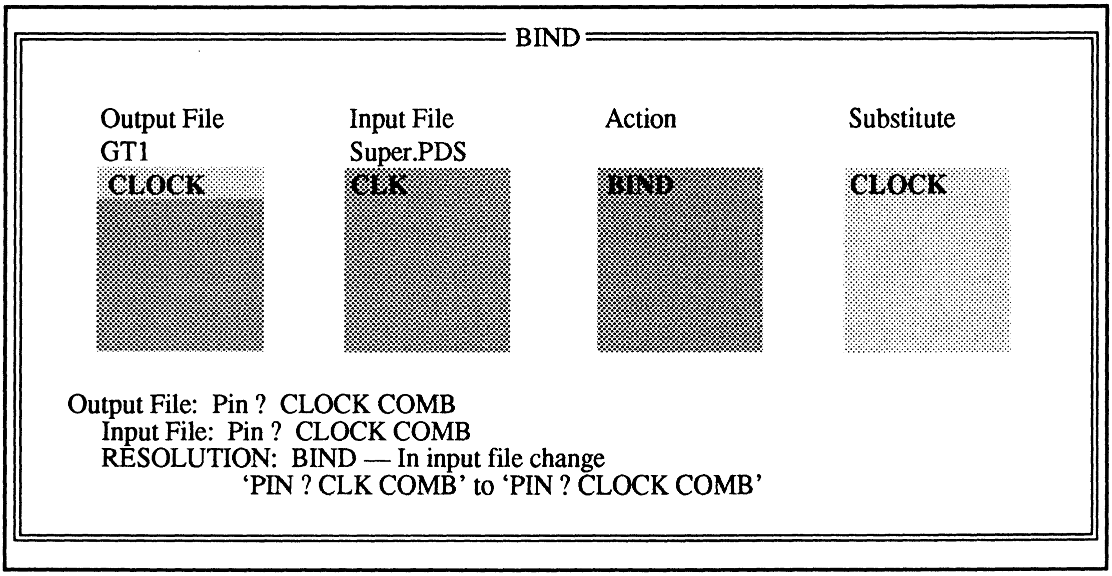
```
RESOLUTION:  BIND __ pin definitions are identical
```

In either case, the status at the bottom of the form reflects the automatic change. If you alter the action or substitute name, the status won't reflect this until you confirm, leave the field, and return to it.

Bind Pins/Nodes



You use this command to display the Bind form, shown next. You can use this form to bind signals of different names to a common signal name. Initially, this form includes only those signals you bound together using the Bind action on the conflict-resolution form.



BIND

Output File	Input File	Action	Substitute
GT1	Super.PDS		
CLOCK	CLK	BIND	CLOCK

```
Output File:  Pin ?  CLOCK COMB
   Input File:  Pin ?  CLOCK COMB
   RESOLUTION:  BIND — In input file change
                  'PIN ? CLK COMB' to 'PIN ? CLOCK COMB'
```

This form is similar to the conflict resolution form; however the field types differ as follows.

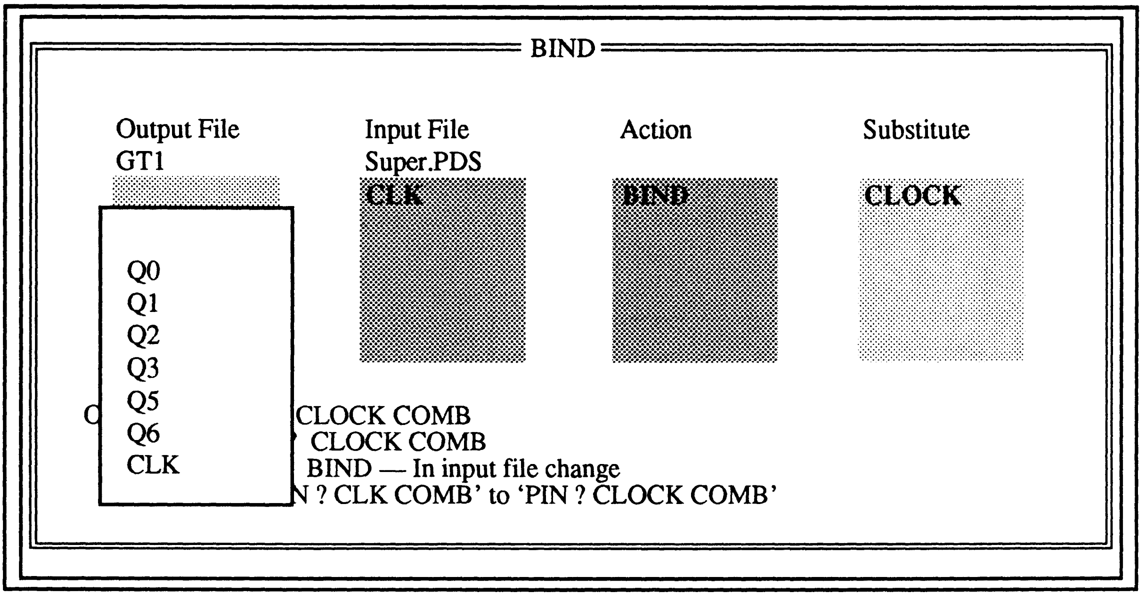
- Option fields: Output File and Input File
- Status fields: Action and Substitute

Output File

This column heading identifies the name under which combined data will be saved.

Each option field in this column identifies a signal in the output buffer that is bound by a common name to a signal in the input buffer.

Blank fields are provided so you can bind signals with different names to a common name. When a blank field is active, you press [F2] to display a list of all signals in the file, then use arrow keys and [Enter] to select a name to fill in the field. An example follows.



```
                              BIND

     Output File        Input File        Action        Substitute
     GT1                Super.PDS
                        CLK               BIND          CLOCK

     Q0
     Q1
     Q2
     Q3
     Q5
    CQ6          CLOCK COMB
     CLK          CLOCK COMB
                  BIND — In input file change
                 N ? CLK COMB' to 'PIN ? CLOCK COMB'
```

You repeat this process with the Input File column. The substitute name is taken from the output buffer.

Input File

This column heading identifies the name of the file in the input buffer.

Each option field in this column identifies a signal in the input buffer that is bound by a common name to a signal in the combined design in the output buffer.

Blank fields are provided so you can specify binding signals with different names to a common name, as described under Output File.

Action

The option field in this column lists Bind when the form first appears. Options for this field include the following.

- Bind
- No action

Bind is the default action. You can select the action field, display a list of options, and select No action to cancel the bind operation for associated signals.

Substitute

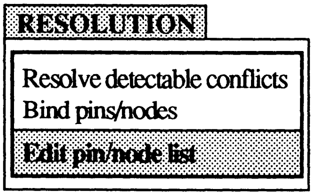
This status field lists the common name that will replace every instance of the original signal name in the input buffer. The common name is taken from the pin in the output buffer. The default substitute name is the one that's used in the output. You cannot edit the substitute name field in this form. However, you can edit the combined design later to change the common name.

Status information

Information at the bottom of the form identifies the exact pin or node statements and how the statement in the input buffer will change. For example,

```
RESOLUTION:  BIND -- In input file change
             'PIN ? CLOCK COMB' to 'PIN ? CLK COMB'
```

Edit Pin/Node List



This command displays a form that includes the header, device type, and pin statements in the output buffer. This form looks and operates like the new PDS file declaration-segment form discussed under 9.2.1.1 and shown opposite.

You can use this form to **edit** information in the **output buffer**.[8]

For example, the Bind form allows you to treat pins in the input and output buffers as the same pin; however, the common name is taken from the pin in the output buffer.

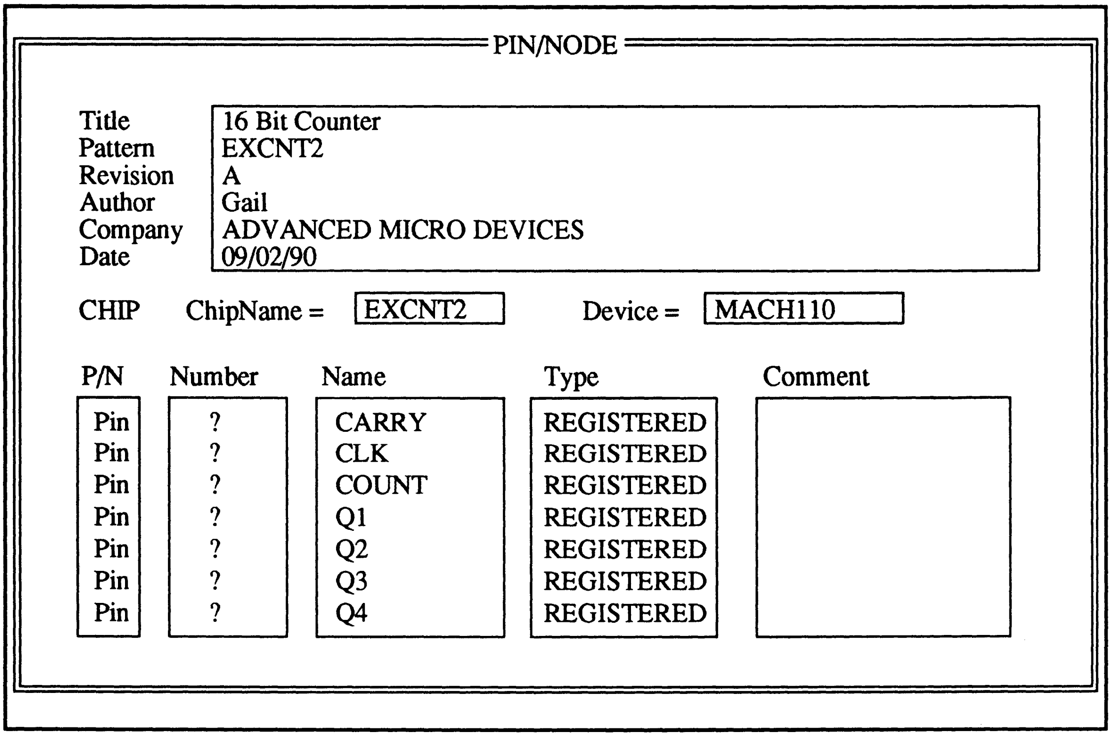
[8] Refer to discussion 9.2.1.1 for details about using the text-based design form.

You can edit the combined data in the output buffer after you merge, to choose a new name for a pin. In addition, you can edit header information or the device type.

The header information is taken from the first input file. Headers in all other input files are disregarded.

Important: You can use a question mark, ?, in the number field to specify a floating pin or node location. The storage type and comment fields are optional.

Also: If you enter new pin/node statements and run out of empty fields, just press [F10] to save the current changes and return to the merge process, then select Edit pin/node list again to display the form. Each time you enter this form, 20 empty pin/node fields become available.



PIN/NODE

Title 16 Bit Counter
Pattern EXCNT2
Revision A
Author Gail
Company ADVANCED MICRO DEVICES
Date 09/02/90

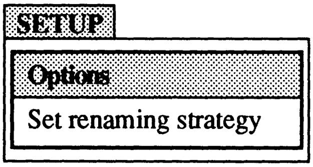
CHIP ChipName = EXCNT2 Device = MACH110

P/N	Number	Name	Type	Comment
Pin	?	CARRY	REGISTERED	
Pin	?	CLK	REGISTERED	
Pin	?	COUNT	REGISTERED	
Pin	?	Q1	REGISTERED	
Pin	?	Q2	REGISTERED	
Pin	?	Q3	REGISTERED	
Pin	?	Q4	REGISTERED	

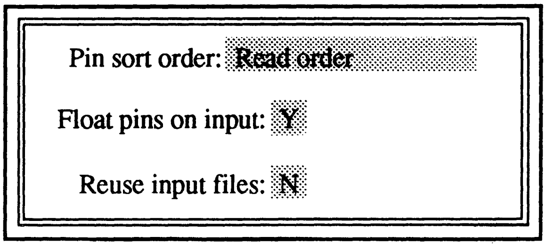
Setup Menu

This menu provides commands you use to set up the merge environment.

Options



When you select this command the form below appears listing the options you can set.



Pin sort order: Read order

Float pins on input: Y

Reuse input files: N

Pin sort order

The pin-sort order field is an option field that determines how signals are listed in pin/node statements when you want to edit the pin/node list. Available options are listed below.

OPTIONS	DEFINITIONS
Read order	List names in the order in which they are read.
Last pin first	List names in reverse read order.
Pin number, name	Sort the list by pin number, then name.
Name	Sort the list alphabetically by name.

Float pins on input

This is a Yes/No text field. Other entries are not accepted.

- Y, the default, specifies floating all pins on input.

  In this case, pin numbers specified in the design file are changed to a question mark, ?, to indicate floating.

- N specifies using the pin numbers assigned in the design file.

Recommendation: It is best to float pins on input to eliminate pin location conflicts. However, if you do not float pins on input and there are two pins assigned to the same location, the location in the input file will be floated automatically. You must then edit the pin/node list in the combined file after merging to restore the original pin location.

Reuse input files

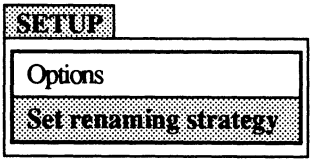
This is a Yes/No text field field. Other entries are not accepted.

- Y lists the names of all files when you use the Get next input file command.

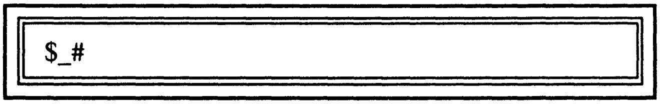
  In this case, when you type either *.* or *.PDS into the file name form, followed by [Enter], the resulting list contains the names of all files.

- N, the default, ensures the names of files merged during this session do not appear in the file name list that appears when you get the next input file.

Set Renaming Strategy



This command allows you to redefine the strategy for default substitute signal names. When you select this command, the form below appears.



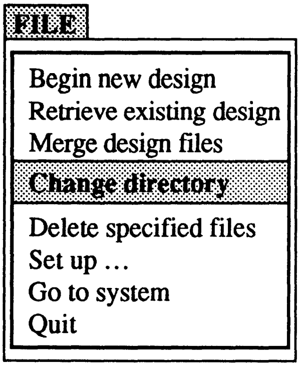
$_#

The field contains the default specification, which means that substitute signal names will be composed of all or part of the existing name, an underscore character, and a three-digit number: name_001.

- $ is replaced with the original name. When necessary, the name is truncated so the entire resulting name does not exceed 14 characters.

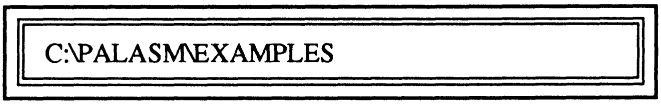
- # ensures that unique names are produced and should be included in any naming strategy. If existing signal names include numbers, these numbers are automatically skipped when substitute names are produced.
- _ allows you to quickly spot substitute names and the numbers assigned.

9.2.1.4 Change Directory



All files are stored in, and retrieved from, the current working directory; all commands operate on the files in the current working directory.

When you select this command to change the current working directory, a form appears with a text field that identifies the path to the current directory.

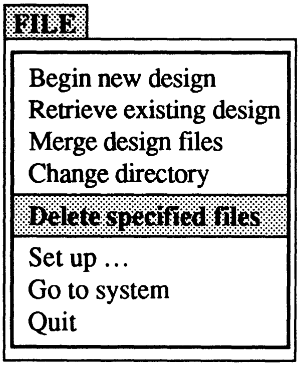


C:\PALASM\EXAMPLES

You can replace all or part of the existing path name with a new one. The new path name must include a valid drive, directory, and subdirectory.

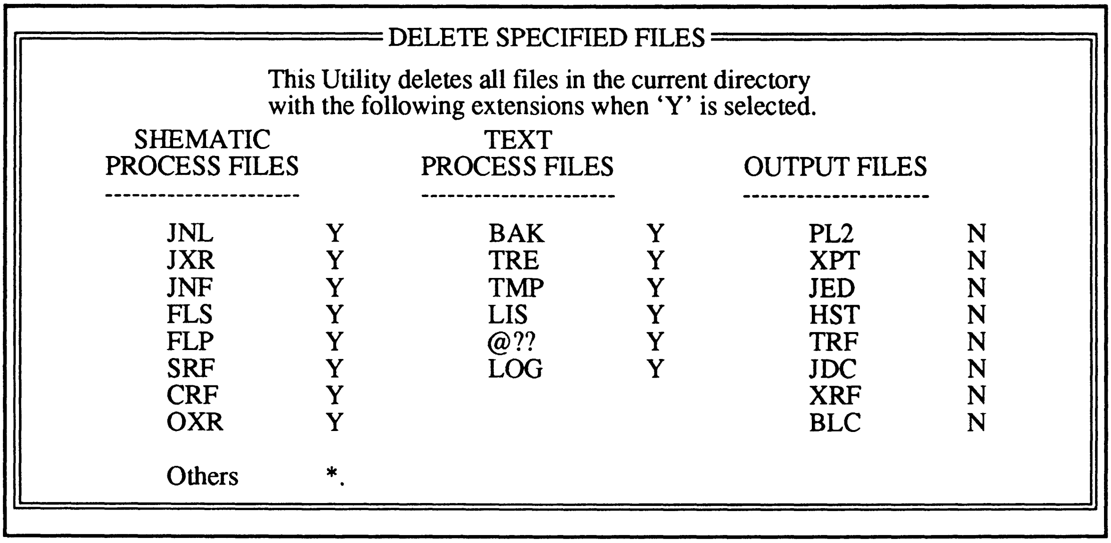
After you confirm the new path, the specified directory becomes the current working directory. Depending on the setup you've defined using the Set up and Working environment commands, the new path may appear in the lower-right corner of the screen.

9.2.1.5 Delete Specified Files



This command initiates a process to delete specified files from the current working directory. When you select this command, a form appears listing all process-related files. Design files are not listed.

- Schematic-process files are created when you convert schematic data to a PDS file.
- Text-process files are created when you compile or simulate a PDS file.
- Output files show various process results you may be interested in viewing.
- Others *., in the left column, is an option field where you can type a specific file extension that's not listed.



DELETE SPECIFIED FILES

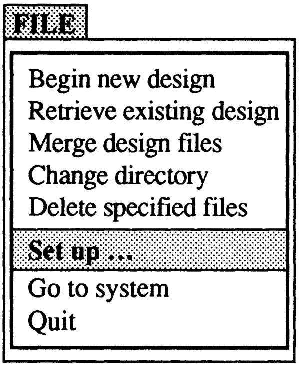
This Utility deletes all files in the current directory
with the following extensions when 'Y' is selected.

SHEMATIC PROCESS FILES		TEXT PROCESS FILES		OUTPUT FILES	
JNL	Y	BAK	Y	PL2	N
JXR	Y	TRE	Y	XPT	N
JNF	Y	TMP	Y	JED	N
FLS	Y	LIS	Y	HST	N
FLP	Y	@??	Y	TRF	N
SRF	Y	LOG	Y	JDC	N
CRF	Y			XRF	N
OXR	Y			BLC	N
Others	*.				

Names are identified by file **extension**. The text field beside each extension contains the letter Y, Yes, or N, No; all files marked with a Y will be deleted. The default is to delete all files **except** those listed under Output files, which includes results you may be interested in viewing.

When you confirm the information in this form, the designated files are deleted.

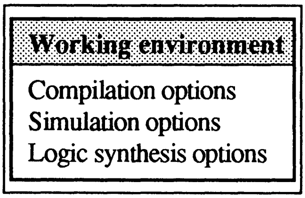
9.2.1.6 Set Up



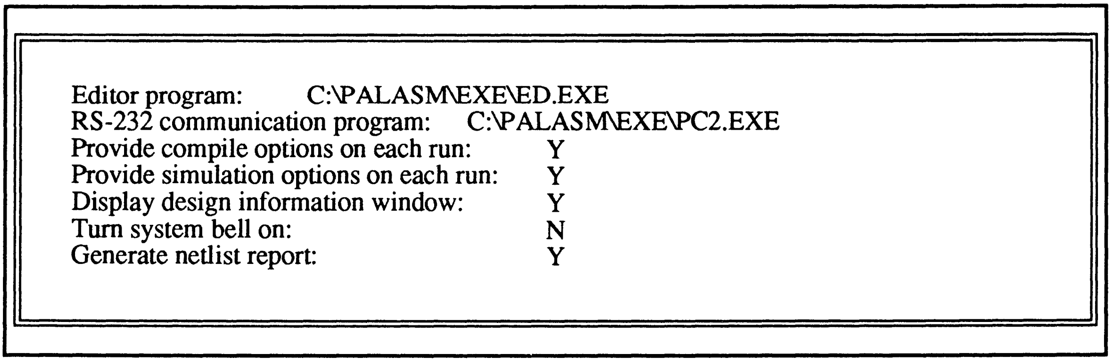
This command allows you to identify software-environment and process preferences. For example, you can suppress certain forms that might otherwise appear each time you begin compilation or simulation. In addition, you can identify a preferred editor and communication program over those supplied by AMD.

The submenu that appears when you select this command offers additional choices as explained below.

Working Environment



This command is used to specify preferences for your working environment. When you select this command, the form below appears providing text fields that display the specifications currently in effect.



```
Editor program:          C:\PALASM\EXE\ED.EXE
RS-232 communication program:    C:\PALASM\EXE\PC2.EXE
Provide compile options on each run:      Y
Provide simulation options on each run:   Y
Display design information window:        Y
Turn system bell on:                      N
Generate netlist report:                  Y
```

Editor program:

This field specifies the path name to the text editor you use to create and edit PDS, simulation, and other text files. The default path name identifies the location of the AMD-supplied text editor.[9]

- If you change the path, a preferred editor will be available for viewing and editing files.

- If the path you supply is incomplete or incorrect, the editor will not be found.

RS-232 communication ...

This field provides the path name to the software that's required to communicate with the device programmer when you download the JEDEC file. The default path name identifies the location of the AMD-supplied communication program.

- If you change the path, a preferred program will be used during the download process.

- If the path you supply is incomplete or incorrect, the program will not be found.

Provide compile options ...

This field specifies when to display the form that defines compilation options.

- Y displays the form each time you select **either** the Compile **or** Both command from the Run menu.

- N displays the form only when you select the Set up command from the File menu followed by the Compilation options command from the submenu.

Provide simulation opt ...

This field specifies when to display the form that identifies the simulation-file option.

- Y displays the form each time you select **either** the Simulation **or** Both command from the Run menu.

[9] Refer to Section V, Appendix A, for a summary of the AMD-supplied text editor commands and operations.

- N displays the form only when you select the Set up command from the File menu followed by the Simulation options command from the submenu.

Display design informa ...

Current design information includes the working directory, input format, design file name, and device type.

- Y displays current information in the lower-right corner of the screen.

- N suppresses the information.

Turn system bell on:

A bell tone can warn you of syntax errors and illegal actions while working with the software.

- Y sounds the tone.
- N suppresses the tone.

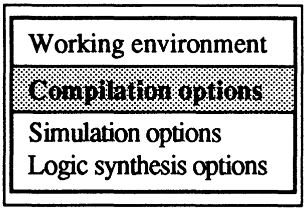
Generate netlist report:

A netlist report is generated when schematic data is converted to a PDS file.

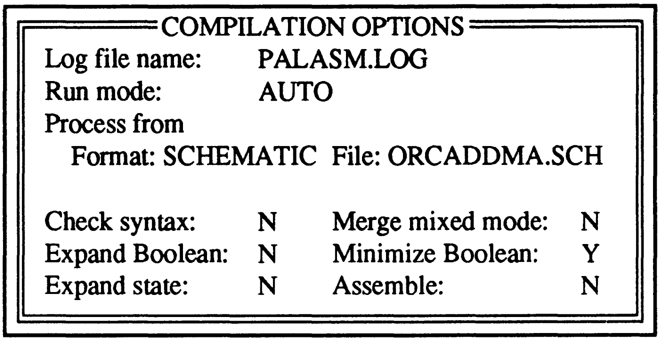
- Y generates the report.
- N suppresses the report.

Upon confirmation, you're returned to the Setup submenu. Specifications take effect as soon as you confirm them, though it may not be obvious until you take a particular action.

Compilation Options



You select this command to display the form that defines compilation options for the current design, as shown below.[10]



```
                    COMPILATION OPTIONS
Log file name:     PALASM.LOG
Run mode:          AUTO
Process from
  Format: SCHEMATIC  File: ORCADDMA.SCH

Check syntax:      N    Merge mixed mode:     N
Expand Boolean:    N    Minimize Boolean:     Y
Expand state:      N    Assemble:             N
```

The form includes status, option, and text fields.

- Two **status** fields in the center of the form identify the input format and file name.
- One **option** field, Run mode, allows you to specify either automatic or manual compilation.
- **Text** fields are provided so you can confirm or cancel options that will be used when you specify manual run mode.

Log file name:

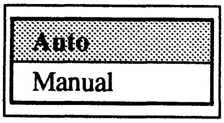
All error, warning, and status messages that scroll by during software processes are stored in the execution-log file named in this text field. The information stored in the log is replaced each time you run a new process.

The default file name is PALASM.LOG. To retain additional versions, you can assign a different name using standard DOS naming conventions. To view any but the most recent log, you must use the Other command on the View menu.

[10] Depending on the working environment setup you've specified, the compilation form may appear automatically when you select either the Compile or Both commands from the Run menu.

Run mode:

The list associated with this option field provides two choices: Auto and Manual.



Automatic mode performs all functions to process a design and ignores specifications in the lower part of the Compilation Options form.

Manual mode performs only those functions specified on the lower part of this form, though it may result in a less than optimal process and result.

> **Important**: The specifications in the following text fields apply **only** when **manual** run mode is specified.

Check Syntax:

This field specifies whether or not a syntax check is made on the PDS file. Any errors discovered during this check must be corrected before compilation can be completed.

Expand Boolean:

This field specifies expanding Boolean equations in the PDS file. Expansion means all equation definitions are expanded into individual equations.

Expand State:

A compiled PDS file contains only Boolean equations. This field specifies whether or not state-machine descriptions are expanded to Boolean equations.

Merge Mixed Mode:

This field specifies whether or not to merge a design that contains both Boolean and state-machine descriptions.

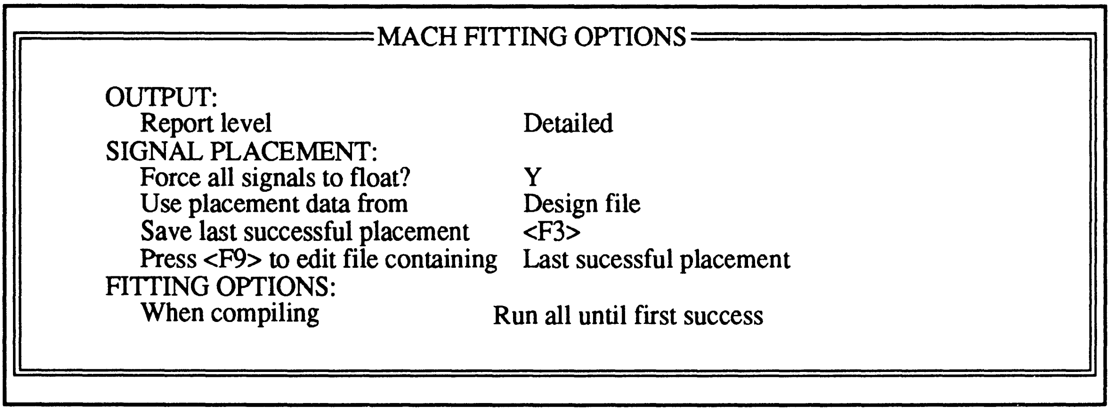
Minimize Boolean:

Minimization reduces a set of Boolean equations to a sum-of-products form that usually involves fewer product terms or literals. This field specifies minimizing Boolean equations in the PDS file.

Assemble:

Assembly translates information in a .TRE file and produces a JEDEC fuse map file for all PAL and PLS device designs and a MACH report for all MACH-device designs. This option field specifies whether or not assembly is performed.

If you're working on a MACH-device design, a MACH Fitting Options form appears after you confirm options on the Compilation Options form.



```
                    MACH FITTING OPTIONS

  OUTPUT:
     Report level                         Detailed
  SIGNAL PLACEMENT:
     Force all signals to float?          Y
     Use placement data from              Design file
     Save last successful placement       <F3>
     Press <F9> to edit file containing   Last sucessful placement
  FITTING OPTIONS:
     When compiling                    Run all until first success
```

The MACH Fitting Options form specifies options unique to fitting MACH-device designs. The form includes status, option, and text fields.

- **Option** fields allow you to specify preferences for output reports, signal placement, and fitting options.

- One **text** field, Force all signals to float, allows you to specify either yes or no.

- A **status** field in the center of the form indicates you want to save the last successful placement.

Report level

This option field provides two report choices for MACH-device designs; the default is Detailed.

OPTIONS	DEFINITIONS
Brief	Suppresses information
Detailed	Provides all available data on the fitting process

Force all signals to float?

This text field identifies whether the pin and node locations specified in the design file are used or ignored. If you type a Y in this field, the design-file placement is ignored and all pin and node locations are left floating; the software chooses locations automatically.

Use placement data from

This option field allows you to specify the source of the signal-placement data to be used during the next fitting process.

OPTIONS	DEFINITIONS
Design file	Use the pin/node statements in the PDS file.
Last successful placement	Use data in the .PLC file, from the last successful placement.
Saved placement	Use data in the .BLC file saved by pressing [F3] after an earlier successful fitting process.

Note: You can override any of these placement options by typing the letter Y in the Force all signals to float field.

Save last successful placement

Data generated during the last successful fitting process is automatically stored in a file named after the design with a .PLC extension: design.PLC. The PLC file is overwritten during each successful fitting process.

This status field indicates you can permanently store the last successful placement in a file, named after the design with a .BLC extension. Press [F3] after a successful fitting process to create this file. This field cannot be selected.

Press [F9] to edit file containing

You can edit the results of a successful placement to use during the next fitting process. For example, you can edit a pin placement to suit specific design constraints. This option field specifies which results are displayed in the text editor when you press [F9].

OPTIONS	DEFINITIONS
Last Successful	Edit the PLC file, which contains the results of the last successful placement.
Saved Placement	Edit the BLC file, which contains the results of an earlier successful placement saved by pressing [F3].

The default is to display results in the PLC file from the last successful placement. This form must be visible when you press [F9].

When compiling

This option field allows you to specify one of four fitting strategies; the default is Run until1st success: STD.

OPTIONS	DEFINITIONS
Use all fitting options	Run all possible combinations; do not stop on first success.
Run until 1st success: STD	Run all possible combinations; stop at first success. Does not execute extra macrocell iterations.
Run until 1st success: EXTRA	Run all possible combinations, including extra macrocell itterations. Stops at first success.
Select one combination.	Choose a placement or resource specification from a new form that appears.

See the ***PALASM 4 Release Notes*** that accompany this software for more information on the Run until 1st success option.

After you choose the Select one combination option, a form appears with additional specifications. All fields on this form are text fields where you can enter Y, Yes, or N, No. The default in each case is Y, which enables the corresponding item.

Maximize packing of logic blocks?	Y
Expand small PT spacing?	Y
Expand all PT spacing?	Y

Maximize packing of logic blocks?

This field specifies packing as many macrocells as possible into each logic block versus holding some macrocells in reserve.

- Y places as many macrocells as possible into each block.
- N holds some macrocells in reserve.

Expand small PT spacing

This field allows more flexibility in choosing macrocell placements and switch-matrix paths for feedback signals.

- Y leaves adjacent macrocells empty when functions with less than four product terms are placed.
- N disables the option to leave adjacent macrocells empty.

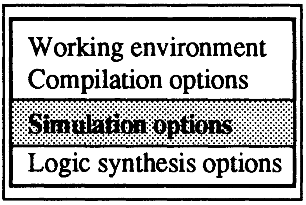
Expand all PT spacing

This specification provides extra switch-matrix resources for intrablock routing. Additional resources are needed for designs that contain many inputs that feed multiple blocks. These resources can be reserved during signal placement by leaving adjacent macrocells empty.

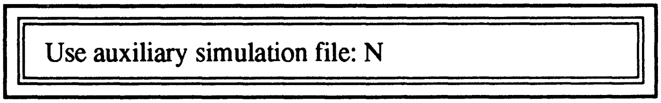
- Y skips one macrocell between each placed signal.
- N disables the option to skip one macrocell between each signal.[11]

[11] Refer to Section II, Chapter 5, for further details regarding these three fitting strategies.

Simulation Options

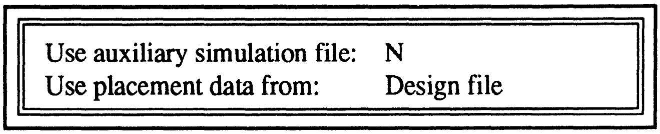


When you select this command for non-MACH-device designs, a form appears that allows you to define where simulation commands are stored.



Use auxiliary simulation file: N

When you select this command for MACH-device designs, the form that appears has an additional option field. This additional field allows you to specify the source of the signal placement data to be used during simulation and test vector generation.



Use auxiliary simulation file: N
Use placement data from: Design file

Use auxiliary simulation file:

Simulation commands can be stored in a separate auxiliary file, named after the design with a .SIM extension. Though you can store commands separately anytime, it is particularly important to do so in the following instances.

- When you merge multiple PDS files into a single MACH-device design, the simulation segments are automatically removed.

  You can use existing simulation commands within each design as a guide to produce a separate auxiliary simulation file.

- When creating schematic-based designs for MACH devices, no simulation commands appear in the resulting PDS file.

  Subsequent debugging and compilation overwrites the resulting PDS file so it's best to store simulation commands in a separate file.

Use placement data from:

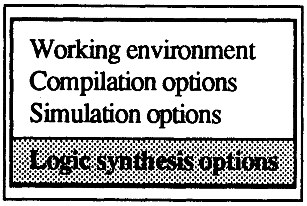
This option field allows you to identify the source of the signal placement data needed to generate test vectors during simulation. For MACH-device designs, test vectors will not be generated if signal placement data is not available. You can choose from three options.

OPTIONS	DEFINITIONS
Design file	Use the pin/node statements in the PDS file.
Last successful placement	Use data in the .PLC file, from the last successful placement.
Saved placement	Use data in the .BLC file saved by pressing [F3] after an earlier successful fitting process.

When selecting one of the above options, the following considerations apply.

- If the design file specifies any pins as floating, use this option field to select the placement data in either the .PLC or the .BLC files.
- If the design file specifies any pins as floating and you select the Design file option, test vectors will not be generated during simulation unless you first back annotate signal placement data from either the .PLC or .BLC files.

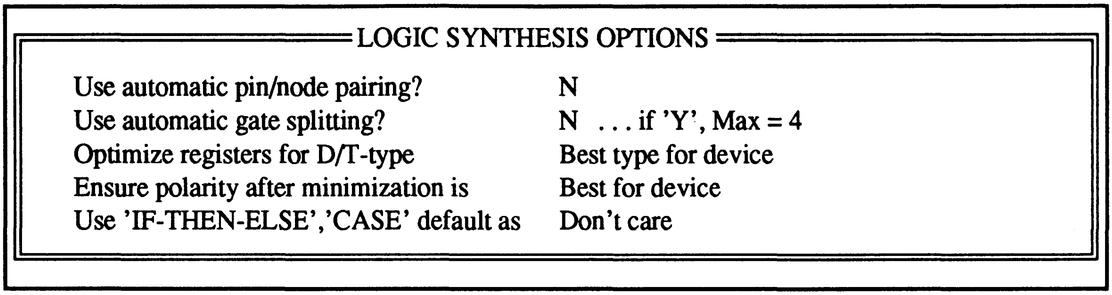
Logic Synthesis Options



Working environment
Compilation options
Simulation options
Logic synthesis options

This command allows you to specify preferences for pairing, gate splitting, register optimization, polarity, and treatment of unspecified default conditions.

When you select this command, a form appears that contains text and option fields that display specifications currently in effect.



```
═══════════════ LOGIC SYNTHESIS OPTIONS ═══════════════
  Use automatic pin/node pairing?        N
  Use automatic gate splitting?          N  . . . if 'Y', Max = 4
  Optimize registers for D/T-type        Best type for device
  Ensure polarity after minimization is  Best for device
  Use 'IF-THEN-ELSE','CASE' default as   Don't care
```

Use automatic pin/node pairing?

This text field defines whether pairing a node with a pin is enabled or not.

- Y specifies automatic input and output pairing.
- N disables automatic pairing.[12]

Use automatic gate splitting?

This text field allows you specify whether or not product terms are allocated from adjacent macrocells in a different block of the device during compilation.

- Y enables gate splitting and allows you to specify up to the maximum number of product terms that can be allocated between MACH blocks.

- N disables gate splitting.

Max =

This option field defines the total number of product terms that can be allocated between adjacent macrocells in different blocks when automatic gate splitting is enabled. Options include the following; 4 is the default.

OPTIONS	DEFINITIONS
4	Four product terms
8	Eight product terms
12	Twelve product terms
16	Sixteen product terms

Each macrocell contains four product terms. Therefore, gate width is defined as the number of product terms

[12] Refer to Chapter 10 for details about pairing a node with a pin.

rounded up to the nearest multiple of four. The maximum gate width is device dependent.

- A maximum gate width of 12 product terms is allowed for MACH 1 device designs.

- A maximum gate width of 16 product terms is allowed for MACH 2 device designs.

Optimize registers for D/T-type

This option field allows you to specify which register type is required for the design. You can choose from four options.

OPTIONS	DEFINITIONS
As specified in design file	Leave design as specified.
D, change all to D-type	Convert flip-flops to D-type.
T, change all to T-type	Convert flip-flops to T-type.
Best type for device	Convert flip-flops first to one type then to the other, compare results, then choose best type based on utilization.

Ensure polarity after...

This option field allows you to specify the polarity required for the design. You can choose from four options.

OPTIONS	DEFINITIONS
As specified in design file	Leave as specified in the design.
Best for device	Use both active high and active low, compare results, choose the one that results in fewest product terms after minimization.
Low, active low	Convert to active low polarity.
High, active high	Convert to active high polarity.

Use 'IF-THEN-ELSE', 'CASE'...

This option field allows you to specify how the software treats default values for the IF-THEN-ELSE and CASE statements. You can choose from two options; Don't care is the default.

OPTIONS	DEFINITIONS
Don't care	Unspecified default conditions are assumed to be don't care.
Off	Unspecified default conditions are assumed to be false.

The don't-care option requires you specify both the on and off sets. The off option requires you to specify only the on sets; the software assumes all other conditions to be off.

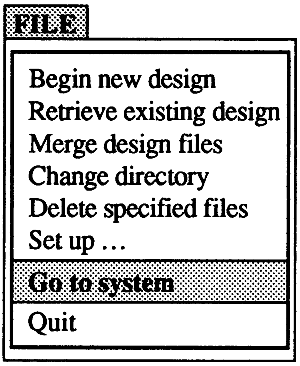
You may lose signals from the design If you select the Don't care-option and do not specify all of the default conditions. If the software treats these signals as don't care, they will be eliminated from the design during logic reduction.

Important: When translating designs created with PLPL, you must select the Off option because PLPL treats unspecified default conditions as false.

Use fast minimization?

This option allows the user to run an abbreviated version of the new Minimizer first introduced with PALASM 4 version 1.4. If it is OFF, a more exhaustive minimization is performed. Turn it ON only if an *Out of memory* error is generated or Minimizer compilation times are unusually long, change the fast Minimizer may produce equations with more product terms than the standard Minimizer did.

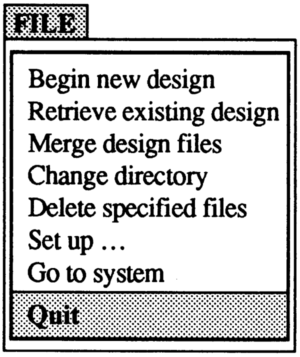
9.2.1.7 Go To System



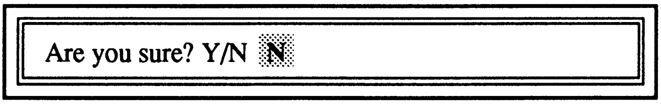
This command temporarily transfers you to the operating system. Once there, you can peruse directories and use any other operating-system commands as usual.

To leave the operating system, just type the word exit and press [Enter]. You're returned to the PALASM environment.

9.2.1.8 Quit



The Quit command transfers you to a confirmation form before exiting the PALASM environment.

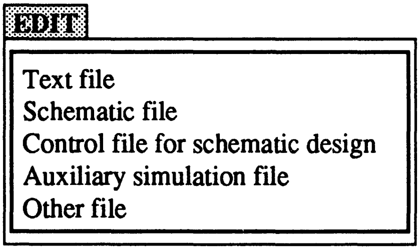


Are you sure? Y/N N

- Y exits the PALASM environment and displays the operating system.
- N returns you to the PALASM environment.

Important: Pressing [Esc] when a top-level menu is displayed initiates the Quit command automatically.

9.2.2 EDIT MENU



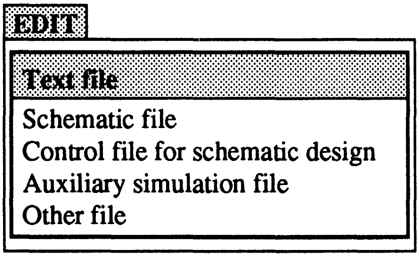
The Edit menu provides two kinds of commands that operate on the specified design in the current working directory.

- **Schematic editor command**
  Schematic file

- **Text editor commands**
  Text file
  Control file for schematic design
  Auxiliary simulation file
  Other file

Important: All commands on this menu operate on the current specified design.[13]

9.2.2.1 Text File



This command transfers you to the text editor and loads the PDS file you specified using the Retrieve existing design command on the File menu. You can edit information in this file as usual.[14]

- If this is a new PDS file, some data may have been entered using the PDS declaration-segment form when the design was created.

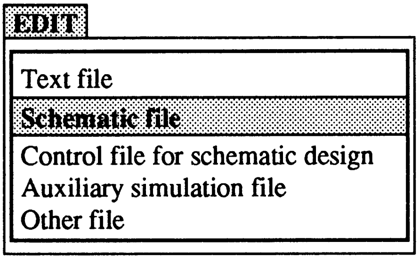
- If the PDS file was converted from schematic data, information in the declaration segment is derived from the control file; equations were produced during either compilation or conversion.

When you leave the editor, you're returned to the PALASM environment.

[13] Refer to discussions 9.2.1.1 and 9.2.1.2 for details about creating and retrieving designs. Refer to 9.2.1.4 for details about changing the current working directory.

[14] Refer to Section V, Appendix A, for details about the AMD-supplied text editor.

9.2.2.2 Schematic File

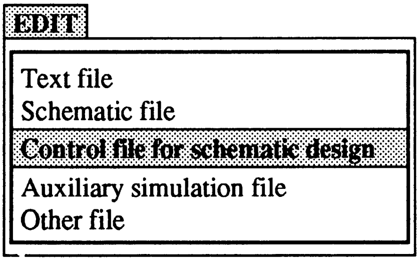


This command transfers you to the OrCAD/SDT III editor and loads the top-level schematic from the current specified design.

All commands and options in OrCAD/SDT III operate as usual.[15] When you leave OrCAD, you're returned the PALASM environment.

Important: Any device type you specify in the schematic is ignored. The device type must be specified in the control file for the schematic.

9.2.2.3 Control File for Schematic Design



This command transfers you to the control-file form for the schematic-based design, as discussed under 9.2.1.1. You can edit any field in the form to change information in the declaration segment of the resulting PDS file.

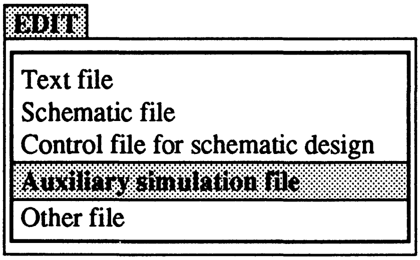
Important: The device type must be specified in the control file for the schematic. Any device type you specify in the schematic is ignored.

Also: If you change the device type in the control-file form, the information in the lower-right corner of the screen is not updated until you run the next process.

When you leave the form, you're returned to the PALASM environment.

[15] Refer to the *OrCAD/SDT III Schematic Design Tools* manual for details about using the schematic editor.

9.2.2.4 Auxiliary Simulation File



This command transfers you to the text editor and loads the auxiliary simulation file for the current design. If a file named with a .SIM extension does not exist in the current directory, a blank file becomes available so you can enter simulation commands.[16]

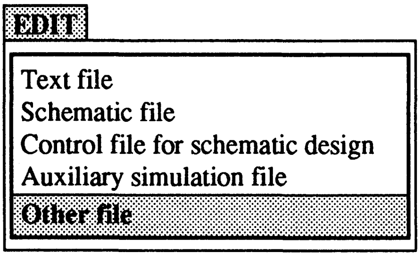
It's a good idea to produce and store simulation commands in a separate file under certain circumstances, for the following reasons.

- When you combine PDS files into a single design, the simulation segment is stripped out of the combined PDS file.
- When you enter a design as a schematic, each time you compile the design a new PDS file is produced so any simulation commands you enter are lost.

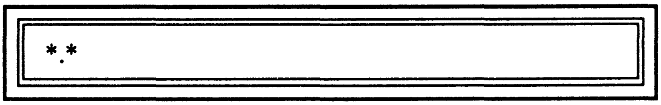
In either case, name the simulation file after the design and include a .SIM extension.

When you leave the editor, you're returned to the PALASM environment.

9.2.2.5 Other File



Use this command to identify a specific file to view or edit. When you select this command, a form appears with a text field so you can specify the name of the file.



.

The intelligent text field in this form allows you to proceed using one of two methods.

[16] Refer to Section V, Appendix A, for details about the AMD-supplied text editor.

A. Type the complete file name.

When you confirm the name, the file is loaded into the appropriate editor and made available on screen.

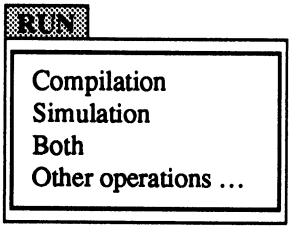
or

B. Display a list of files using one of the techniques below.

- Press [Enter] to display a list of all files in the current directory.
- Type part of a name, such as design.*, to display a list of specific files, such as all files relating to the named design.
- Type a different drive or directory path to display a list of files elsewhere.

In any case, once you select a name from the resulting list, you're transferred to the appropriate editor and the file is automatically loaded. When you leave the editor, you're returned to the PALASM environment.

9.2.3 RUN MENU



This menu provides a list of operations you can perform on the current design.

- **Primary commands**
  Compilation
  Simulation
  Both

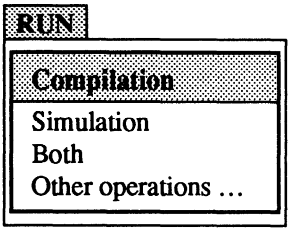
- **Secondary command**
  Other operations ...

  The Other operations command displays a submenu of commands that perform secondary tasks, as discussed under 9.2.3.4.

Important: All commands on this menu operate on the current specified design.[17]

Discussions below explain each command on the Run menu.

9.2.3.1 Compile



This command initiates the compilation process. Depending on the working environment options you specified, forms that define compilation and MACH-fitting options may appear automatically.[18]

In any case, a window opens when the process begins and messages scroll by to keep you informed.

Schematic designs

A. Certain OrCAD utilities are run to check schematics for electrical design-rule violations and the verified schematic is converted into a PDS file.

All designs

B. A syntax check is performed on the PDS file.

C. Equations are expanded, then minimized.

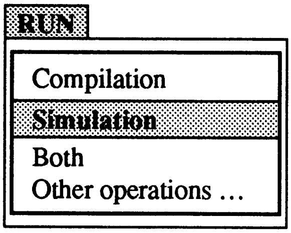
D. Assembly procedures are completed for PLD designs; the fitting process is performed for MACH-device designs.

[17] Refer to discussions 9.2.1.1 and 9.2.1.2 for details about creating and retrieving designs. Refer to 9.2.1.4 for details about changing the current working directory.

[18] Refer to discussion 9.2.1.6, Set Up, for details about compilation options.

The device pin out and JEDEC files are produced if all processes are successful. Other files are also produced.

9.2.3.2 Simulation



This command verifies design logic using commands placed in either the simulation segment of a PDS file or in an auxiliary simulation file. However, no timing verification is done.

Depending on the working environment options you've set, a form may appear asking if you're using an auxiliary simulation file.

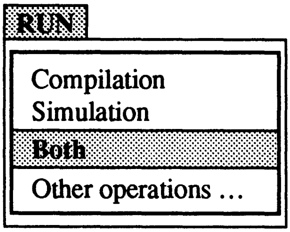
- Y indicates you are using a separate simulation file.
- N, the default, indicates simulation commands reside in the PDS file.

When the process begins, a window opens and messages scroll by to keep you informed. Two types of files are produced during simulation that can help you debug your design.

- Simulation data
- History Waveform display

If the design has not been compiled before selecting this command, it is first compiled, then simulated.

9.2.3.3 Both

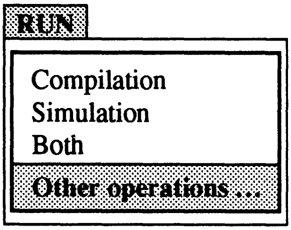


This command saves time when you want to compile and simulate the design at once. Depending on your design and set up options, several forms may appear automatically: compilation, MACH fitting, and simulation options.

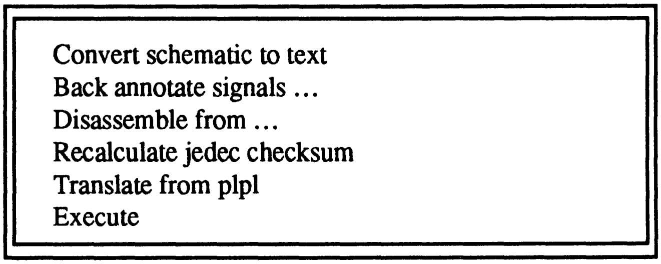
When the process begins, a window opens and messages keep you informed, as usual.

- All compilation functions are performed and output files are produced unless errors occur.
- Simulation is performed and the simulation-results files are produced.

9.2.3.4 Other Operations



This command displays a submenu that lists secondary commands you can use to **either** debug a design **or** recover a lost design.


Convert schematic to text
Back annotate signals ...
Disassemble from ...
Recalculate jedec checksum
Translate from plpl
Execute

Discussions below define each command on the submenu.

Convert Schematic to Text

Schematic conversion ordinarily occurs each time you compile the design. However, this command converts data from the current specified schematic into a PDS file without compiling the design.

When the conversion starts, a window opens and messages scroll by. A single PDS file is produced; however, a syntax check is not performed.

Back Annotate Signals

This command pertains only to MACH-device designs. You use this command to display a list of options that indicate the source for signal-placement data. The data from the specified source replaces existing signal locations in the PDS file and pin out report.[19]

OPTIONS	DEFINITIONS
Change all to floating	Ignore locations in the PDS file and float all pins and nodes.
Use last successful placement	Replace the locations in the pin/node statements in the PDS file with those from the PLC file.
Take from saved placement	Replace the locations in the pin/node statements in the PDS file with those from the BLC file.

- The PLC file is created during the last successful placement; this file is overwritten during each successful fitting process.

- The BLC file is created when you press [F3] after a successful placement and contains information from the PLC file.

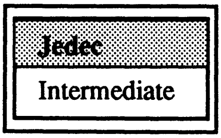
When you select a command from the submenu, the process is initiated. Upon completion, a window opens and provides status information.

Note: If the software detects an error in mapping the signal placement data into the .PDS file, the results of the back annotation will be stored in a separate file with the name design.PBK. For example, defining unused signals in the PDS file or using an illegal number will create this condition.

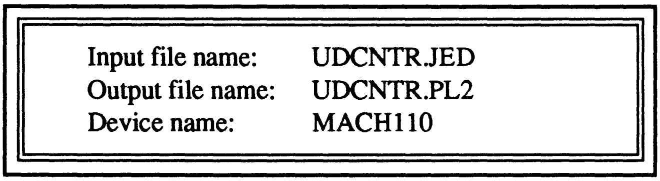
Disassemble From

This command displays a submenu with the two choices discussed below.

[19] Refer to discussion 9.2.1.6, Set Up, Compilation options, for details about signal back annotation on the MACH fitting options form.



The Jedec command converts JEDEC fuse data into Boolean equations, which is useful to reconstruct a design for which other files are missing. When you select this command, the form below appears with two text fields and an option field.



Input file name:	UDCNTR.JED
Output file name:	UDCNTR.PL2
Device name:	MACH110

Input file name:

This field provides the name of the JEDEC file, which corresponds to the currently specified design name followed by a .JED extension. A name in this field does not indicate the corresponding file exists. You can enter a new name to use a different file as input. You can enter *.JED to display a list of all JEDEC files in the current working directory.

Output file name:

This field names the Boolean equation file created during disassembly. Again, the name matches the design followed by a .PL2 extension. You can enter a new name to store the results in a different file.

Device name:

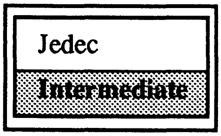
This option field allows you to select the device type corresponding to the original design. JEDEC disassembly is not supported for all devices. You can only disassemble designs created for the devices on the option list, displayed by pressing [F2] in this field.

Once you confirm specifications in the disassembly form, the process is initiated.

Note: When JEDEC fuse data is converted to Boolean equations, the pin names, comments, and simulation vectors in the original PDS file are removed.

Also: Entry formats, such as state machine, waveform, and truth tables, are converted to Boolean equations.

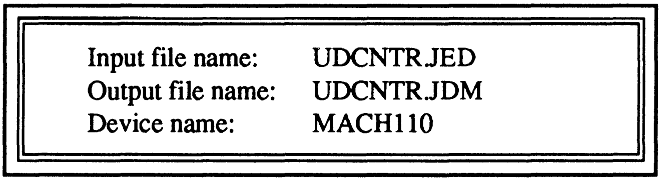
When finished, a Boolean equation output file is stored in the current directory under the name you specified.



The Intermediate command disassembles the intermediate file, PALASM2.TRE, so you can view the results of the minimization and expansion processes in a sum-of-products format.

Recalculate JEDEC Checksum

This command recalculates the checksum in the JEDEC test-data file. When you select this command, the form below appears.



Input file name: UDCNTR.JED
Output file name: UDCNTR.JDM
Device name: MACH110

Input file name:

This field contains the name of the JEDEC checksum file in the current directory, which corresponds to the current specified design name with a .JED extension. A name in this field does not indicate the corresponding file exists. You can enter a new name to use a different file as input.

Output file name:

This field contains the default name of the output file that will be created. Again, the name matches the design followed by a .PDS extension. You can enter a new name to store the results in a different file.

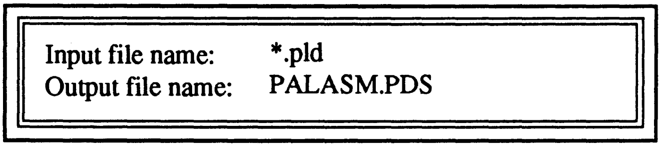
Device name

This option field allows you to select the device type corresponding to the original design. Checksum recalculation is not supported for all devices. You can only recalculate the checksum for the devices on the option list, displayed by pressing [F2] in this field.

Once you confirm specifications in the recalculation form, the process is initiated. When finished, a JDM file is stored in the current directory under the name you specified.

Translate from PLPL

This command converts a PLPL design file into a PDS file. When you select this command, a form appears so you can enter the input and output file names.



Input file name:	*.pld
Output file name:	PALASM.PDS

Input file name:

This text field provides the name of the PLPL file, which corresponds to the current specified design name followed by a .PLD extension. A name in this field does not indicate the corresponding file exists. You can enter a new name to use a different file as input.

Output file name:

This text field names the file that will be produced. Again, the name matches the design followed by a .PDS extension. You can enter a new name to store the results in a different file.

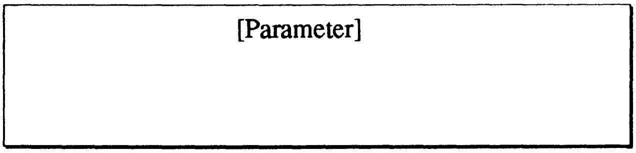
Once you confirm specifications in the translation form, the process is initiated. When finished, a new PDS file is stored in the current directory under the name you specified.

Execute

This command sets OrCAD/SDT III configuration options, such as the printer or plotter drivers, library prefix, etc. When you select this command, a form appears requesting a parameter.

```
draft.exe /c[parameter]
```

- Press [Enter] to display a blank parameter form.



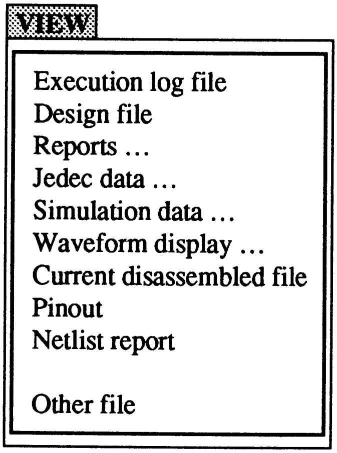
[Parameter]

- Press [Enter] a second time to invoke OrCAD's configuration program.

- Change options as usual, then update the information and quit.

When you quit, you're returned to the PALASM environment.[20]

9.2.4 VIEW MENU



The View menu provides commands to display all files related to the currently specified design. However, you cannot edit files in view mode.

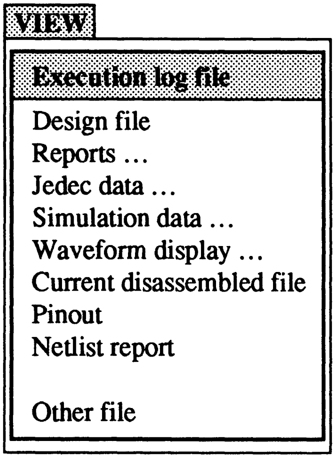
- You can press [Esc] to dismiss the file and return to the View menu.

- You can use the Go to system command on the File menu to suspend to the operating system, then use the print command to print a file. Type exit and press [Enter] to return to the PALASM environment.

Brief definitions of available commands and other information about each file are provided below.

[20] Refer to the *OrCAD/SDT III Schematic Design Tools* manual for details about specifying options using DRAFT/C.

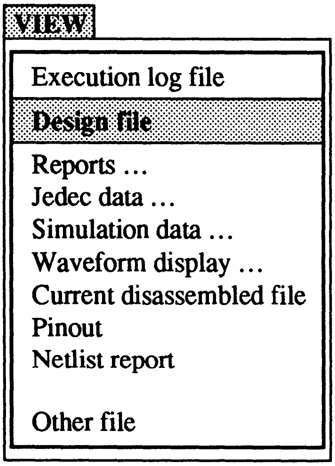
9.2.4.1 Execution Log File



This command displays the log file that contains all messages generated by the last software process run on the current design.

The log file is rewritten each time you initiate a new process. All error, warning, and status messages appear as they are recorded.

9.2.4.2 Design File

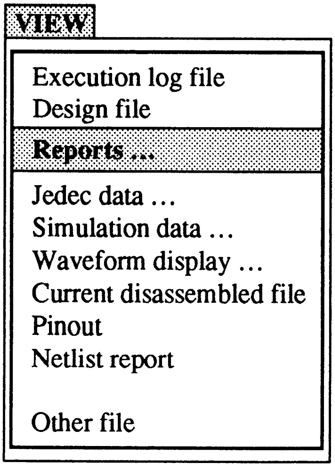


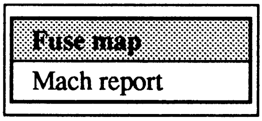
This command displays the current design file in the appropriate editor, either the text editor or the schematic editor. You can edit the file and print it using the associated editor commands.

If you retrieved a schematic design and want to view the PDS version created during compilation, you must use the Other file command on the View menu.

9.2.4.3 Reports

This command provides a submenu that lists the names of the files produced either during the PLD assembly or MACH fitting-process.





Fuse Map

This file provides both programmed and unprogrammed fuse data generated during the assembly process for non-MACH device designs.

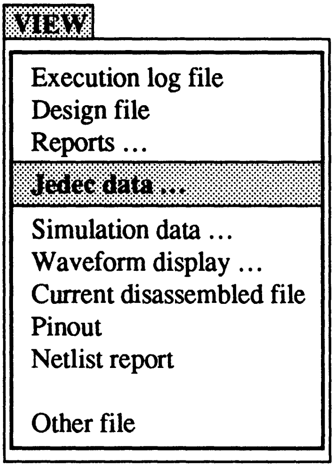
- The symbol for a programmed fuse is –.
- The symbol for an unprogrammed fuse is X.

Mach Report

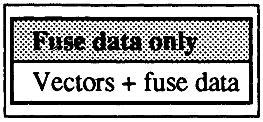
This file contains placement, block, and device pin-out information for MACH-device designs. The content of this file is controlled by the output-report specification on the MACH Fitting Options form, as shown below and discussed under 9.2.1.6.

```
...
Report level                Detailed
...
```

9.2.4.4 JEDEC Data



This command displays a submenu that lists commands to view JEDEC fuse and vector data.



Fuse Data Only

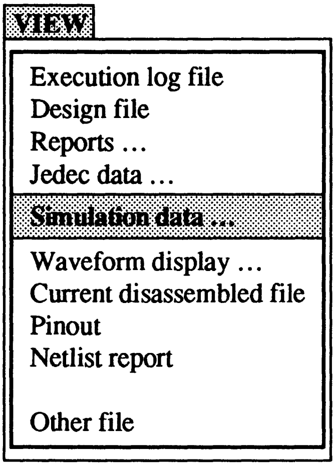
The fuse data file is created during the assembly or fitting process. The information in this file is in a machine-readable format that you can download to program a device.

Vectors + Fuse Data

Vectors are added to the fuse file after a successful compilation and simulation. Information in this file includes the following.

- Fuse data from the JEDEC fuse data file
- Test vectors added during simulation that can be used to verify a device on a programmer

9.2.4.5 Simulation Data

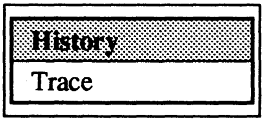


This command displays a submenu of commands to view the simulation history and trace files in a text format.

The following information is presented in each file.

- Each instance of g represents the SETF command in the simulation file.
- Each instance of c represents a complete clock cycle, which is defined by the CLOCKF command in the simulation file.

History



The history file contains the behavior of all signals defined in the pin statements. Information in this file is divided into two columns. You can track values using the cursor, which is displayed as a thick vertical bar.

- The left column lists pin names for each pin listed in the declaration segment of the PDS file.
- The right column records the simulation results in text-format waveform.

  H = high
  L = low
  X = undefined
  Z = output disabled

Trace

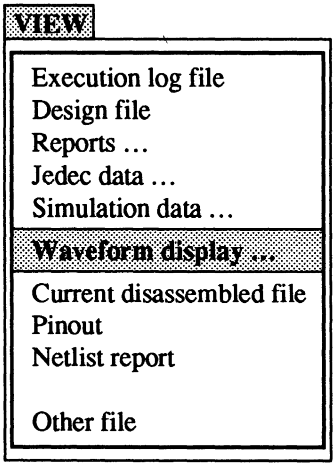
The trace file contains the behavior of only those signals specified using the Trace command in the simulation segment or file. Again, you can track values using the cursor, which is displayed as a thick vertical bar.

Information is displayed in the same text format as the history file.

- The left column provides the pin names you specified using the Trace command.
- The right column records high and low signals as a text-format waveform.

  H = high
  L = low
  X = undefined
  Z = output disabled

9.2.4.6 Waveform Display

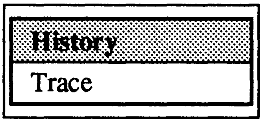


This command displays a submenu that lists the simulation waveform files.

The following information is presented in each file.

- Each instance of g represents the SETF command in the simulation file.
- Each instance of c represents a complete clock cycle, which is defined by the CLOCKF command in the simulation file.

History



This file displays the simulation history of **all signals** defined in the pin statements in a graphic format. Again, you can track values using the cursor, which is displayed as a thick vertical bar.

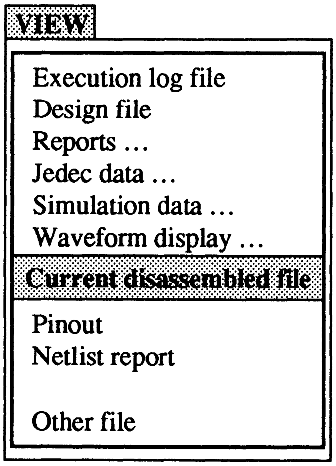
- The left column provides pin names for each pin listed in the declaration segment in a PDS file.
- The right column records high and low signals graphically.

Trace

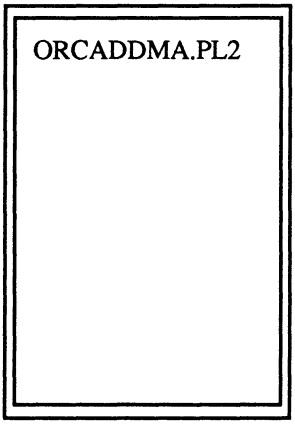
This file displays only the simulation **trace data** in a graphic format. Again, you can track values using the cursor, which is displayed as a thick vertical bar.

- The left column provides names of the pins you specified using the **Trace command** in the simulation segment or file.
- The right column displays the simulation results graphically.

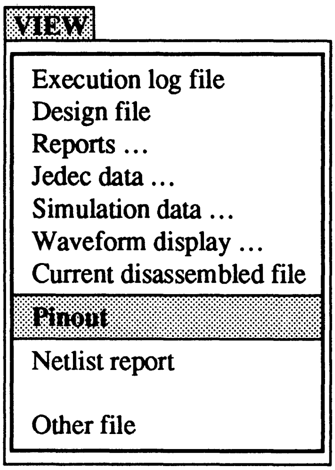
9.2.4.7 Current Disassembled File



This command lists the current disassembled file, if any, which contains the results of disassembling either the intermediate .TRE file or the JEDEC file. After selecting the name from the list, the file appears on the screen.



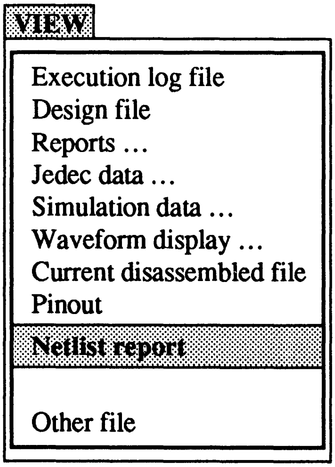
9.2.4.8 Pinout



This file provides the device pin-out information in graphic form. For MACH-device designs only, this file also includes pin and node assignments for the specified device following a successful compilation.

Note: For MACH-device designs, if you specified all pins as floating, you must first back annotate the PDS file with the signal placement that's created during the fitting process, as discussed in 9.2.3.4. Otherwise, the pinout report shows all pins as NC, no connect.

9.2.4.9 Netlist Report



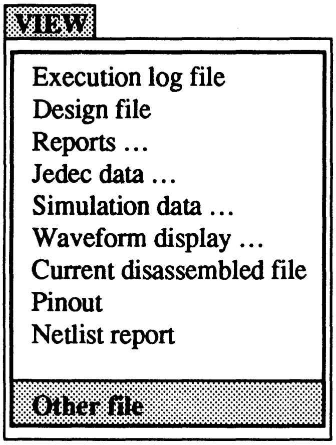
The netlist report is an intermediate file created when schematic data is converted to PDS format. This text file identifies the following.

- Signal names as they appear in the schematic
- Reference designators after annotation by OrCAD utilities
- AMD-supplied macro type
- Sheet information and X and Y locations on the sheet

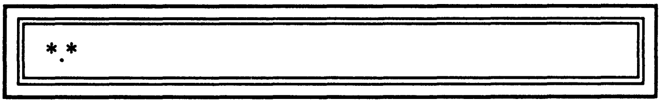
Note: This report is **not** created if you type the letter N beside the Generate netlist report field on the working-environment form.

```
...
Generate netlist report        N
```

9.2.4.10 Other File



This command allows you to view files not available through explicit commands on the View menu, including those in other directories. When you select this command, the form below appears.

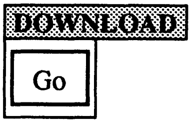


The intelligent text field in this form allows you to display a file using one of three methods.[21]

- Press [Enter] to display a list of all files in the current directory.
- Type part of a name, such as design.*, to display a list of specific files, such as all files relating to the named design.
- Type a different drive or directory path to display a list of files elsewhere.

In any case, when you select a name from the list the file is displayed.

9.2.5 DOWNLOAD MENU



The Download menu provides access to the communication program specified in the working-environment form discussed under 9.2.1.6.

You select this command to download a JEDEC file to a device programmer via the communications software. The Go command initiates loading the JEDEC file to a device programmer using one of the following packages.

[21] If the specified file is a schematic, the file is loaded and displayed in the OrCAD/SDT III editor. In this case, you can edit or print the schematic, as usual.

- The AMD-supplied PC2 programmer communications software.

```
...
RS-232 communication pro...:     C:\MACH\EXE\PC2.EXE
...
```

- The programmer communications software that's specified in the field of the working-environment form shown below.

```
...
    RS-232 communication program: C:...
...
```

When you select this command, the communications software screen appears. When you leave the communications environment, you're returned to the PALASM environment.

9.2.6 DOCUMENTATION MENU

The Documentation menu provides access to online help. Three topics are available.

Index of topics
Language reference
Help on errors

Important: You select items from a menu using arrow keys to highlight the item and the [Enter] key to select it. In addition, you can use a mouse, in which case, you highlight the desired item and double click the left mouse button. **However**, you **cannot** select a menu item by typing the first letter.

9.2.6.1 Index of Topics

When you select this command, the on-line help index screen appears. Available topics are listed in a submenu.

Topics include release notes, system configuration, troubleshooting techniques, etc.

When you select a topic from the submenu, such as release notes, the information appears and you can use PgUp or PgDn to scroll through the file.

- You press [Esc] or select Resume to return to the Online Help Index.
- You select Navigate to obtain error recovery or language reference information.
- You press [Esc] or select Quit and respond Yes to the prompt to return to the PALASM menu.

9.2.6.2 Language Reference

When you select this command, available language-construct topics are listed in a sub-menu.[22]

Important: You select items from a menu using arrow keys to highlight the item and the [Enter] key to select it. In addition, you can use a mouse, in which case, you highlight the desired item and double click the left mouse button. However, you **cannot** select a menu item by typing.

When you select a construct, a brief overview appears, including the correct syntax and supported devices. Additional topics are listed across the top of the screen; status and prompt messages appear at the bottom of the screen. To select a topic, just move the cursor to the right or left and press [Enter].

The following discussions describe each topic that's displayed.

[22] These topics are also described in Section IV, Chapter 10.

Overview

A brief overview of the construct includes a sample syntax and list of supported devices.

Syntax

A repeat of the syntax and a simple example of its use in a PDS file are shown.

Definitions

Descriptors following the keyword are usually defined in their order of appearance in the statement. Definitions include exact syntax requirements and details about the results of using the construct when applicable.

Note: If more than one screen is required to display the information, use the PgUp and PgDn buttons provided in the upper-right corner. You select a button by pressing [Enter].

Use

This topic provides details about using the construct.

Related Topics

This command displays a menu of related topics you can refer to for additional information. Press [Enter] to go to a related construct. If none are listed a message informs you.

Previous Menu

This command returns you to the language constructs menu. To return to the documentation menu, either

- Press [Esc] once and respond Yes to quit online help, or
- Select Quit and respond Yes.

9.2.6.3 Help On Errors

This command searches the execution log file that's generated during compilation for error and warning messages. The log file appears and the first message is displayed.

- Select Next message for the next error explanation and recovery.

The total number of messages and the currently displayed message number appear in the lower-right corner. When the number of messages is greater than one, two additional commands appear: Next message, Previous message.

- Select Prev message to display the previous error explanation and recovery.
- Select the Browse file command and use the PgUp and PgDn buttons to scroll through the file.
- Select Enlarge recovery to view a detailed explanation and suggested recovery.
- Select Other to view additional error files or navigate to the Index of Topics or the Language Reference.
- To return to the Documentation menu, select either the Quit command or press [Esc], then respond Yes to confirm.

9.2.7 [F1] FOR HELP

Pressing [F1] transfers you to the online help and provides information about the use or function of the currently selected menu, command, option, text, or status field.

If more than one page is required to display all of the information, PgDn and PgUp buttons are provided in the lower right and upper-left corners.

- Use the up or down arrow keys to activate a button and press [Enter] to select or use the [Page Up] [Page Down] keys.
- Press [Esc] to return to PALASM.

CHAPTER 10

LANGUAGE REFERENCE

CONTENTS

10 LANGUAGE REFERENCE

This chapter describes language elements available through the PALASM 4 software for all PLD- and MACH-device designs.[1] Language elements are organized alphabetically by name and include the following information.

- **Name** identifies a specific element and explains its purpose.
- **Syntax** identifies the segment of the PDS file where the element is used and shows required and optional syntax and variables; an example of use in a PDS file is included.
- **Definitions** describe each parameter and identifies specific syntax requirements.
- **Use** provides additional details.

Important: In the syntax boxes of this chapter, required information appears in all capital letters; variables and optional information have initial caps.

[1] All information is also available at the workstation through Online Help.

OVERVIEW

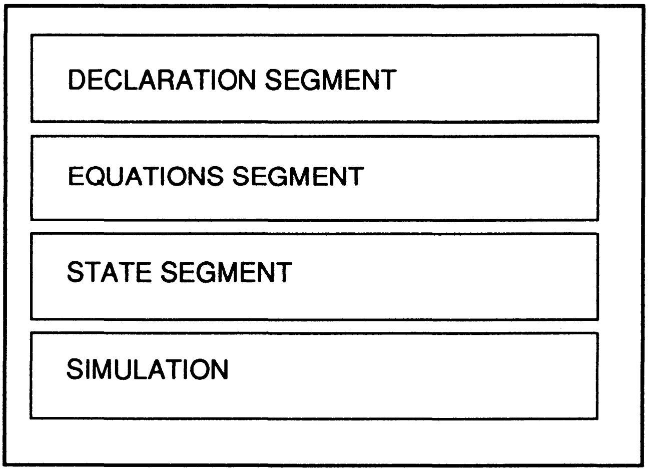
The PALASM 4 software supports three kinds of design specifications.

- Boolean equations
- State-machine descriptions
- Schematics

Only text-based designs use the language elements described in this chapter.

> **Important**: A design entered as a schematic has its logic converted to Boolean equations during the compilation process. In addition, you can manually convert a schematic-based design to Boolean equations.
>
> In any case, you can edit the resulting PALASM description specification (PDS) file to modify the design. At that point, all file and language elements relating to Boolean-equation specifications apply.

Text-based designs are described in PDS files, which are divided into several segments.



DECLARATION SEGMENT

EQUATIONS SEGMENT

STATE SEGMENT

SIMULATION

All PDS files include a declaration segment and usually a simulation segment.[2] These segments contain identical information regardless of the kind of description you produce. Depending upon the type of design description you choose, the PDS file will contain an equations segment and/or a state segment . Files that contain both an equations segment and a state segment are referred to as mixed-mode designs.[3]

The software is not case sensitive. You can enter any information using upper- or lowercase letters or a combination. For example, PALASM 4 treats the following as the same.

- ABC
- abc
- Abc

This chapter does not describe how to create a PDS file[4] using the text editor, nor how to use the declaration-segment form,[5] available when you create a new text-based design.

[2] Refer to Section II, Chapter 6, for details about when to include the simulation commands in a separate file rather than in the PDS file.

[3] Refer to Section II, Chapter 4, for details about producing mixed-mode designs.

[4] Refer to Section I, Chapter 2, for a step-by-step guide to producing a PDS file for both Boolean and state-machine designs.

[5] When you begin a new text-based design, a declaration-segment form appears, which you can complete to specify header information, PIN/NODE statements, chip name, and PLD or MACH-device type. Refer to Chapter 9, in this section, for details.

BOOLEAN-EQUATION ELEMENTS

The next figure identifies all elements available for use in each segment of a PDS file for a Boolean-equation design specification.

DECLARATION

AUTHOR
CHIP
COMBINATORIAL
COMMENT
COMPANY
DATE
DECLARATION
GROUP
LATCHED
NODE
PATTERN
PIN
REGISTERED
REVISION
SIGNATURE
STRING
TITLE
VECTOR

EQUATIONS

BOOLEAN EQUATION
EXPRESSION
CASE
.CLKF
.CMBF
COMMENT
EQUATIONS
FUNCTIONAL EQUATION
GND
IF-THEN-ELSE
.J EQUATION
.K EQUATION
OPERATOR
.PRLD
.R EQUATION
.RSTF
.S EQUATION
.SETF
.T EQUATION
.T1 EQUATION
.T2 EQUATION
.TRST
VCC
VECTOR

SIMULATION

EXPRESSION
CHECK
CLOCKF
COMMENT
FOR-TO-DO
IF-THEN-ELSE
PRELOAD
PRLDF
SETF
SIMULATION
TRACE_OFF
TRACE_ON
VECTOR
WHILE-DO

STATE-MACHINE CONSTRUCTS

The following figure identifies all elements available for each segment of a PDS file for a state-machine design specification.

DECLARATION

- AUTHOR
- CHIP
- COMBINATORIAL
- COMMENT
- COMPANY
- DATE
- DECLARATION
- GROUP
- LATCHED
- NODE
- PATTERN
- PIN
- REGISTERED
- REVISION
- SIGNATURE
- STRING
- TITLE
- VECTOR

STATE

- EXPRESSION
- CLKF
- COMMENT
- CONDITIONS
- DEFAULT_BRANCH
- DEFAULT_OUTPUT
- LOCAL DEFAULT
- MASTER_RESET
- MEALY_MACHINE
- MOORE_MACHINE
- .OUTF
- .OUTPUT_ENABLE
- OUTPUT_HOLD
- START_UP
- STATE
- STATE ASSIGNMENT
- STATE EQUATIONS
- STATE TRANSITION
- VECTOR

SIMULATION

- EXPRESSION
- CHECK
- CLOCKF
- COMMENT
- FOR-TO-DO
- IF-THEN-ELSE
- PRELOAD
- PRLDF
- SETF
- SIMULATION
- TRACE_OFF
- TRACE_ON
- VECTOR
- WHILE-DO

SPECIFYING OUTPUTS IN IF-THEN-ELSE AND CASE STATEMENTS

The following rules apply to each pin or node for outputs defined in the IF-THEN-ELSE or CASE statement.

Specify the desired output for each test condition of the IF-THEN-ELSE or CASE statement you want generated.

Note: If you do not request a specific output, the software assumes either you do not care about the output for this condition, or the output for this condition is actually defined through some other equations.

If you define the behavior of an output in an IF-THEN-ELSE or CASE statement, and then define it further using a Boolean equation or a separate IF-THEN-ELSE or CASE statement, you must obey certain rules. Specifically, you must avoid situations when both of the following occur.

- More than one of the resulting equations can be simultaneously evaluated as true.
- The output behavior defined in one equation contradicts behavior defined in the other.

If you violate these rules, the software reports the following error message: Overlapping on/off covers (check equations for completeness).[6]

Four Ways of Specifying Outputs

You can specify the behavior of each pin or node as follows.

1. Define the output as a function of the same inputs for each possible test condition.
2. Define the output as a function of different inputs for different test conditions.

[6] Techniques for avoiding this error are discussed under the heading Avoiding Overlap Errors.

3. Define the output for some of the possible test conditions and not for others.

4. Define the output two different ways for the same test condition.

The following examples are given for the IF-THEN-ELSE statement, but they apply to the CASE statement also. The IF-THEN-ELSE statement is a special instance of the CASE statement in which IF and ELSE portions replace the individual test conditions of the CASE statement.[7]

Example 1

Defining the output as a function of the same inputs for each possible test condition.

```
IF X = 0 THEN
      BEGIN
              Q = A * B
      END
ELSE
      BEGIN
              Q = 0 ;equivalent to Q = GND or /Q = 1 or /Q = VCC
      END
```

The software interprets this statement as follows.

- When X = 0, output Q is high when the expression A * B evaluates as true, and low when the expression A * B evaluates as false.

- When X = 1, output Q is always low.

The statement is reduced to the following Boolean form.

Q=A * B */X

[7] Refer to TUTOR6.PDS and TUTOR7.PDS in the PALASM\EXAMPLES directory for additional examples of IF-THEN-ELSE and CASE statements.

This statement is then combined with any other equations for Q by the minimizer.

Example 2

Defining the output as a function of different inputs for different test conditions.

```
IF X = 0 THEN
     BEGIN
            Q = A * B
     END
ELSE
     BEGIN
            Q = C * D
     END
```

The software interprets this statement as follows.

- When X = 0, output Q is high when the expression A * B evaluates as true, and low when the expression A * B evaluates as false.

- When X = 1, output Q is high when the expression C * D evaluates as true, and low when the expression C * D evaluates as false.

The statement is reduced to the following Boolean form.

Q= C * D * X + A * B */X

This statement is then combined with any other equations for Q by the minimizer.

Example 3

Defining the output for some of the possible test conditions and not for others.

```
IF X = 0 THEN
     BEGIN
            Q = A * B
     END                      ; No ELSE condition or no output
                              ; equation
                              ; in some of the CASE conditions
```

The software interprets this statement as follows.

- When X = 0, output Q is high when the expression A * B evaluates as true, and low when the expression A * B evaluates as false.
- When X = 1, output Q is undefined. In the Karnaugh map, don't-care values are used for all map locations where X = 1, unless a location is defined by other equations for Q.

This statement is reduced to the following Boolean form.

Q = A * B * /X + "don't care" * X

This is then combined with any other equations for Q by the minimizer. In the event that there is no other equation for Q, this equation reduces to Q = A * B, in which case output Q is no longer a function of input X. This occurs because of the don't-care values introduced by omitting the definition of output Q when X = 1. If you want output Q to remain a function of X, rewrite the IF-THEN-ELSE statement, adding the ELSE clause shown below.

```
IF X = 0 THEN
     BEGIN
             Q = A * B
     END
ELSE
     BEGIN
             Q = 0
     END
```

Example 4

Defining the output two different ways for the same test condition.

```
IF X = 0 THEN
     BEGIN
             Q =  A * B
            /Q = /A
     END
ELSE
     BEGIN
            Q = 0 ;equivalent to Q = GND or /Q = 1 or /Q = VCC
     END
```

Here, the designer has defined explicitly when the output should be high or low, when X = 0. The software interprets this statement as follows.

- When X = 0, output Q is high when the expression A * B evaluates as true, and low when the expression input A is low.
- When X = 1, output Q is always low.

This statement is reduced to the following Boolean form.

Q = A * /X

This statement is then combined with any other equations for Q by the minimizer. Note that output Q is no longer a function of input B. This occurs because the software assumes don't-care values for all conditions that are not covered by the equations

Q = A * B and /Q = /A, where X = 0.

Avoiding Overlap Errors

The following example shows how two separate IF-THEN-ELSE statements can interact to produce an Overlapping on/off covers (check equations for completeness) error. This error is typically caused by unrealizability, as illustrated in the following example.

```
IF A = 1 THEN
     BEGIN
            Q = C * D
     END
IF B THEN
     BEGIN
            Q = 1
     END
```

A problem occurs when both A and B are high and C or D or both C and D are low. In this situation, the first IF statement tells output Q to go low, while the second IF statement tells the same output to go high. Since Q cannot be high and low simultaneously, the equation set is unrealizable.

There are several ways to avoid problems of this nature, as described next.

Method 1

Make it impossible to have more than one IF-THEN-ELSE or CASE statement evaluate as true simultaneously.

```
IF A = 1 THEN
     BEGIN
            Q = C * D
     END

IF /A * B THEN             ;Condition /A prevents both IF
     BEGIN                 ;statements from evaluating as
            Q = 1          ;true simultaneously.
     END
```

Method 2

Make it impossible for the output specifications to conflict by always using the same output equation.

```
IF A = 1 THEN
      BEGIN
             Q = 1                      ;This output equation...
      END

IF B = 1 THEN
      BEGIN
             Q = 1                      ;...is identical to this.
      END
```

SYNTAX AND EXAMPLES

In this chapter, all language elements are presented in two forms, as shown below.

```
Syntax
             Output_pn    Assignment Operator            Expression
Example
   EQUATIONS
             /02                    =                     I1 * I2 * I3
                                                          + I4 * I5
             ...
```

- The syntax area shows the syntactical name of each element needed to complete the statement, and shows the proper order.

- The example provides a sample as it is used in a PDS file.

Important: Spacing is exaggerated in the samples shown in this chapter for readability. For example, tabs separate each element in the example to align it with the corresponding identifier above. In most cases, however, blanks can be used as separators in the PDS file.

You may not use curly braces, { }, inside IF-THEN-ELSE or CASE statements. Instead of the curly-braced shorthand, the right side of the referred-to logic equation must be copied in full as the right side of the equivalent logic equation (as in output pairing). Also, even though it is legal to do so, it is not recommended that you use curly braces outside IF-THEN-ELSE and CASE constructs to refer to an equation that is not defined inside of such constructs. The result may not what you expect or desire.

ASSIGNMENT OPERATOR

The assignment operator is a symbol that defines a specific operation interpreted by the software when processing design files. There are several assignment operators, which perform different functions depending on the software operation and where the operator appears.

Devices Supported: All PLD devices.

SYNTAX

You can use the assignment operator in equations, as shown below, and in state and conditions segments of a PDS file for either Boolean or state-machine descriptions.

Syntax	Pn	Assignment Operator	Expression
Example			
	Q0	=	EXT1
	Q1	:=	EXT2
	Q2	*=	EXT3

Definitions

The various assignment operators are defined below.

Assignment Operator

The := and *= operators are provided to support older PLD designs that include a pin list not containing a storage type.

ASSIGNMENT OPERATOR

=	This universal assignment operator agrees with any data-storage type: Combinatorial, registered, or latched. You can use this assignment operator in expressions whether or not the PIN or NODE statements contain a storage type. This operator is best suited to new designs.
:=	This assignment operator defines a registered output. The signals in the expression must be defined in either the PIN or NODE statement as registered. Otherwise, an error occurs and processing stops.
*=	This assignment operator defines a latched output. The signals in the expression must be defined in either the pin or node statement as latched or an error is reported during processing.

USE

In the equations segment, the assignment operator performs two functions.

- It defines the output on the left side of the equation as a function of the various inputs and feedback signals listed on the right side.
- It defines the output storage type.

You can assign the storage type in a PIN or NODE statement. In this case, you can use the universal assignment operator, =, in equations or expressions. If you use the := or *= operators, they must agree with the storage type defined in the PIN or NODE statement or an error occurs.

ASSIGNMENT OPERATOR

You can use assignment operators in the STATE and CONDITIONS statements in the state-machine designs, as shown below.

```
...
STATE
MOORE_MACHINE
ZERO.OUTF = /CNT2 * /CNT1 * /CNT0
CONDITIONS
CNTUP = ENABLE * CNT_UP
...
```

AUTHOR

This keyword defines the name of the design's author. This statement is useful for documenting the design.

Devices Supported: All PLD devices.

SYNTAX

You can use the AUTHOR statement only in the declaration segment, as shown below.[8]

Syntax

```
          AUTHOR                  Designer
```

Example

```
          TITLE
          PATTERN
          REVISION
          AUTHOR                  Karen Olsen
          COMPANY
          DATE
```

Definitions

Only the descriptor following the keyword, Author, is discussed.

Designer

Designer defines the name of the individual who created the design. Observe the guidelines below.

- Include the AUTHOR statement and the designer name in either Boolean or state-machine files or in the schematic-control file form.
- Place the statement and name in the order specified in the example above, following REVISION and preceding COMPANY.
- Use any combination of up to 59 alphanumeric characters and the period character.

[8] Refer to Chapter 9, in this section, for details about the declaration-segment form for new text-based designs and the schematic-control form for new schematic-based designs.

Reserved words are allowed within this statement; however, **do not** use any punctuation or symbols other than the period.

USE

The following error conditions pertain to the AUTHOR statement.

- Without the AUTHOR statement, the software issues a warning and continues processing the file.
- With multiple AUTHOR statements, the software issues an error and stops processing the file.

BOOLEAN EQUATION

These equations are used to define the desired logic functions produced at the outputs. Boolean equations form the backbone of any PDS file containing a Boolean description.[9]

Devices Supported: All PLD devices.

SYNTAX

You use Boolean equations in the equations segment of a PDS file, following the keyword EQUATIONS.

```
Syntax

             Output_pn   Assignment Operator      Expression
Example

             /02                     =            I1 * I2 * I3
                                                  + I4 * I5
```

Definitions

All parameters are described below. Additional details are provided under Use.

Output_pn

Output_pn is the name of an output pin or node defined in the declaration segment of the PDS file. Once defined, you can include the name in a Boolean equation to define how this output is to be used.[10]

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[11]

[9] Refer to BOOLEAN EQUATION and EXPRESSION, in this chapter, for additional details.

[10] Refer to NODE and PIN, in this chapter, for details about including a forward slash, /, to define polarity.

[11] Refer to ASSIGNMENT OPERATOR, in this chapter, for more information.

Expression

Expression is a group of signals or product terms, on the right side of an equation, separated by logical operators.[12] The product operator, *, denotes a logical AND function and the sum operator, +, denotes a logical OR function. You can use strings and vectors in an expression.

All values in an expression for a PLD design are signal names that must be defined before their use. The name must be defined in PIN and NODE statements in the declaration segment of the PDS file.

USE

The following conventions should be observed.

- Place a Boolean equation in any order, unless it's a substitution, in which case you must define the substitute expression before using it.
- Follow the structure shown in the syntax example; however, you can include strings and vector notation.

  a. Include either blanks or a tab character before and after the assignment operator.

  b. Use an equal sign as the assignment operator.

     The software determines storage type, either registered, combinatorial, or latched, from the PIN and NODE statements in the declaration segment.

[12] Refer to EXPRESSION, in this chapter, for additional details.

- Include up to 80 characters per line in each equation and include as many lines as necessary.

  You can place the expression on a separate line.

You can use the following high-level language statements and constructs to specify logic behavior. These are converted to Boolean equations during software processing.

- CASE
- FOR-TO-DO
- IF-THEN-ELSE
- WHILE-DO

State equations are another high-level representation of logic behavior for state-machine designs. When you compile a state-machine design, state equations are converted to Boolean equations.

Functional equations are a type of Boolean equation that defines an input. Functional equations have a similar format but a different use.[13]

[13] Refer to the following topics, in this chapter, for additional details: COMBINATORIAL, EXPRESSION, FUNCTIONAL EQUATIONS, LATCHED, REGISTERED, and STATE EQUATIONS.

CASE

This keyword provides a condition-testing structure for Boolean equations. The CASE statement more efficiently supports multiple values than the nested IF-THEN-ELSE statement.

Devices Supported: All PLD devices.

SYNTAX

You can only use the CASE statement in the equations segment of Boolean designs.

Syntax

```
    CASE (Condition Signals)
        BEGIN
        Value:          BEGIN
                            Action statement
                        END

        Value:          BEGIN
                            Action statement
                        END

        Keyword:        BEGIN
                            Action statement
                        END
        END
```

Example

```
    CASE (A,D)
        BEGIN
        0:              BEGIN
                            C = A * B
                        END

        3:              BEGIN
                            C = A + B
                        END

        2:              BEGIN
                            C = 1
                        END

        OTHERWISE:      BEGIN
                            C = 0
                        END
        END
```

Definitions

You can use groups, vector notation, and strings in a CASE statement. Parameters following the keyword, CASE, are defined below; additional details appear under Use.

Condition Signals

Condition signals define a set of signals, which are tested against a list of values, to determine subsequent actions in following statements.

The set is concatenated into a single binary value using signal values as individual bits. Each **signal** value in the set is represented by a name and has a binary value of either 0 or 1. Each **signal name** can be either an input or an output. The following rules must be observed.

- Follow the keyword in a CASE statement with the condition signals.
- Enclose signal names or vector notation in parentheses separated by commas: for example, (A,B,C[0..4])

> **Important**: Signal names or vector notation must be separated by commas.

- Define each signal name or vector notation and it's value in a PIN or NODE statement in the declaration segment.

Value

Value defines the constant in the subsequent action statement. This value is tested against the present value of the condition signals to determine whether that action is taken or skipped. The rules below apply.

- The value can be any constant; however, do not duplicate values.

The default constant is decimal. Non-decimal constants are expressed with the following prefixes.

#b Binary
#h Hexadecimal
#o Octal

All values are converted to binary during compilation.

- The value must end with a colon and precede an action statement.

Action Statement

The action statement defines the logic for a particular condition-signal value. This logic is implemented when the value preceding this statement matches the current value of the condition signals. The following rules apply.

- Precede each action statement with a value.
- Enclose each statement with BEGIN and END.

Action statements inside the BEGIN and END block can be any expression that is valid within the equations section of the design.

An action is a series of any legal PALASM statements within the equations section.

OTHERWISE

Use this keyword to identify the beginning of an optional statement indicating the action for default values of the CASE statement. When you define every possible value, you do not need this statement. However, any unspecified conditions are assumed to be don't care; they are not assumed to be false. Signals for which default conditions are not specified are eliminated from the design during the logic-reduction process.

The OTHERWISE statement generates the condition as identified and shown below.

- OR all conditions for values defined in action statements before this statement.
- Complement the entire OR term, then AND it with the right side of the OTHERWISE statement.

```
CASE (A,D)
BEGIN
   0:             BEGIN C = A * B END
   3:             BEGIN C = A + B END
   2:             BEGIN C = 1          END
   OTHERWISE:     BEGIN C = 0          END
END
```

At each clock cycle, or anytime the signals change, the condition signals (A,B) are evaluated. When the value of the signals matches the value in one of the following action statements, the logic defined in that statement is implemented; otherwise, C = 0.

The previous example is expanded during compilation into the following Boolean equation.

```
C  =      (VCC * /A * /B)
                  + /(VCC * A * B)
                  + (D * /(/A * /B + A * B))
```

USE

You can specify how the software treats default values for CASE statements by selecting one of two options on the File/Set up/Logic synthesis options form, as follows.

OPTIONS	DEFINITIONS
Don't care	Unspecified default conditions are assumed to be don't care.
Off	Unspecified default conditions are assumed to be false.

The don't-care option requires you specify both the on and off sets. The off option requires you to specify only the on sets; the software assumes all other conditions to be off.

You may lose signals from the design if you select the don't-care option and do not specify all the default conditions. If the software treats these signals as don't care, they will be eliminated from the design during logic reduction.

Important: When translating designs created with PLPL, you must select the off option because PLPL treats unspecified default conditions as false.

You can nest CASE statements within IF-THEN-ELSE statements and other CASE statements.

```
CASE (A,D)
BEGIN
   0:      BEGIN C=A*B END
   1:      CASE (C,D)
           BEGIN
           0:      BEGIN E=C*D END
           1:      BEGIN E=C+D END
           OTHERWISE:   E=1   END
           END
   OTHERWISE:      C=0 END
END
```

There is no limit to the number of nested statements you can include in a design; however, too many levels of nesting may cause you to run out of memory.[14]

If any equations are incompletely specified or ambiguous, the unspecified signals are assumed to be don't care. This increases the amount of logic reduction possible during the minimization process. However, if you have not fully evaluated the consequences of these don't-care signals, the equations resulting from compilation may be different than you intended.[15]

[14] Refer to the following topics, in this chapter, for additional details: GROUP, IF-THEN-ELSE, and VECTOR.

[15] Refer to Chapter 9, in this section, for details about default options for CASE and IF-THEN-ELSE statements on the Logic Synthesis Options form.

CHECK

This keyword in a simulation command verifies signal values at the **pin** are equal to expected values.

Devices Supported: All PLD devices.

SYNTAX

You use the CHECK command in either the simulation segment of a PDS file or in an auxiliary simulation file for Boolean, state-machine, or schematic-based designs.

Syntax

```
   CHECK      Prefix_pns
```

Example

```
   SIMULATION
   CHECK      01 /02 ^03 %04 PLAYING
```

DEFINITIONS

If the signal being tested is defined with the same polarity as in the Pin/Node declaration segment, the signal is checked to verify it is a logical 1. If the polarity is reversed, the signal is checked to verify it is a logical 0.

Note: The syntax examples are valid only if the signals are defined as active-high in the Pin/Node declaration segment.

Parameters following the keyword are defined below. Additional details are provided under Use.

Prefix

Prefix indicates the logic state of the corresponding pin, node, or state. Do not leave a blank between Prefix and pns. There are four prefixes: null, forward slash, /, caret sign, ^, and percent sign, %.

- The null prefix indicates an **active-high** signal is checked to verify it is a logical 1. In the syntax example, O1 has a null prefix.

  When used in conjunction with a state name, a null prefix indicates the specified state should be checked. In the syntax example, PLAYING has a null prefix.

- The forward slash, /, indicates an **active-high** signal is checked to verify it is a logical 0. In the syntax example, O2 has a forward-slash prefix.

- The caret sign checks the corresponding signal for a high-impedance state. High impedance occurs when a three-state buffer on an I/O pin is disabled. In this case, the letter Z appears in the simulation files to indicate the high-impedance state. In the syntax example, O3 has a caret prefix.

- The percent sign checks the corresponding signal for a don't-care state. A don't-care condition occurs when combinatorial logic is not initialized. In this case, the letter X appears in the simulation files to indicate the don't-care state. In the syntax example, O4 has a percent prefix.

Pns

Pns defines the names of the pins, nodes, or states to be verified.

- Each signal name can be up to 14 characters in length.

- Include up to 76 characters per line and use as many lines as you need.

  The screen displays up to 76 characters per line; however, all information is processed properly even if it extends beyond the 76th character.

- Include a blank between the keyword and the first pin, node, or state in the list.

  You can include **multiple** pin and node names. You can use strings or vector notation to define the signal list.

- Separate multiple prefixed pin and node names with a blank.

USE

The CHECK command verifies values of the defined signal at the pin. In contrast, the CHECKQ command verifies values at the Q output of a register.

If the signal being tested is defined with the same polarity as in the Pin/Node Declaration segment, the signal is checked to verify it is a logical 1. If the polarity is reversed, the signal is checked to verify it is a logical 0.

A conflict occurs when the value at the pin does not match the expected value. Each conflict is identified with a question mark, ?, in the simulation output files; a warning is issued and the expected value is reported in the execution-log file.[16]

The CHECK command verifies logical operations only and does not add test vectors in the JEDEC file.[17]

[16] Refer to Section II, Chapter 4, for additional details.

[17] Refer to TEST, in this chapter, for details about creating test vectors in JEDEC files.

CHECKQ

This keyword in a simulation command verifies values at the Q outputs of **registers** are equal to expected values.

Devices Supported: All PLD devices.

SYNTAX

You use the CHECKQ command in either the simulation segment of a PDS file or in an auxiliary simulation file for Boolean, state-machine, or schematic-based designs.

Syntax

```
   CHECKQ    Prefix_rns
```

Example

```
   SIMULATION
   CHECKQ    Q0 /Q1 PLAYING
```

DEFINITIONS

Parameters following the keyword are defined below. Additional details are provided under Use.

Because CHECKQ verifies signal values at the Q output of registers, you do not need to account for active-low pin declarations. This makes CHECKQ especially useful for verifying states.

Prefix

The prefix indicates the logic state of the corresponding register, node, or state. Do not leave a blank between Prefix and rns. There are two prefixes: null and forward slash.

- The null prefix indicates the register or node should be a logical 1. In the syntax example, Q0 has a null prefix.

  When used in conjunction with a state name, a null prefix indicates the specified state should be checked. In the syntax example, PLAYING has a null prefix.

- The forward slash indicates the signal should be a logical 0. In the syntax example, Q1 has a forward-slash prefix.

Rns

Rns defines the names of the output registers, nodes, or states to be verified. Each value represents both the signal name or state and the expected output value.

- Each signal name can be up to 14 characters in length.

- Include up to 76 characters per line and use as many lines as you need.

  The screen displays up to 76 characters per line; however, all information is processed properly even if it extends beyond the 76th character.

- Include a blank between the keyword and the first register, node, or state in the list.

  You can include **multiple** register and node names. You can use strings or vector notation to define the signal list.

- Separate multiple prefixed register and node names with a blank.

CHECKQ

USE

The CHECKQ command verifies that signal values at the register outputs are equal to the expected values. In contrast, the CHECK command verifies pin and node values at the output pin.

Because CHECKQ verifies signal values at the Q output of registers, you do not need to account for active-low pin declarations. This makes CHECKQ especially useful for verifying states.

A conflict occurs when the value of the output register does not match the value defined in the CHECKQ command. Each conflict is identified with a question mark in the simulation output files; a warning is issued and the expected value is reported in the execution log file.

The CHECKQ command verifies logical operations only and does not create test vectors in the JEDEC file.

CHIP

This keyword introduces the statement that defines the chip name and the PLD or MACH device for the design being created.

Devices Supported: All PLD devices.

SYNTAX

You use the CHIP statement in the declaration segment of either Boolean or state-machine designs[18]

Syntax

```
          CHIP                    Name          Device
```

Example

```
          TITLE
          PATTERN
          REVISION
          AUTHOR
          COMPANY
          DATE
          CHIP                    Counter       PALCE22V10H-25
```

Definitions

The CHIP statement must follow the DATE statement and precede PIN and NODE statements. Parameters that follow the keyword are defined below.

Name

Name assigns a chip name that conforms to the rules below.

- Use up to 14 alphanumeric characters in the PDS file. The first character cannot be numeric.

  Do not use operators, reserved words, carriage returns, tabs, or blanks.

[18] Refer to Chapter 9, in this section, for details about the declaration-segment form for new text-based or control-file form for schematic-based designs.

Device

Device assigns a valid AMD device name, such as PAL16R8.

- The name must conform to the basic device number. Power, speed, package, and operating range designators are optional.[19]

USE

The software cannot process a design file if it does not contain a CHIP statement. Errors can cause the software to misinterpret the CHIP statement and, often, the PIN and NODE statements.

[19] Refer to the specific device datasheet for the correct device name.

.CLKF

This reserved word defines a clock signal for a pin or node. The operation differs depending upon the device you've chosen.[20] For example, for devices with programmable clocks, you can use .CLKF to assert the clock on a flip-flop.

Devices Supported					
PAL16RA8	PAL20RA10	PAL26V12	PALCE29M16	PALCE29MA16	PAL32VX10
PALCE610	PLS30S16	MACH 1	MACH 2		

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state designs or in any registered-latched design.

Syntax			
	Pn.CLKF	Assignment Operator	Expression
Example			
	Q0.CLKF	=	EXT

Definitions

All parameters are defined below.

Pn.CLKF

Pn.CLKF assigns a pin or node name followed by the reserved word .CLKF. The name must be defined in an earlier PIN or NODE statement in the declaration segment. A forward slash before the pin or node name defines a falling edge clock. The absence of a forward slash defines a rising edge clock.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[21]

[20] Refer to Chapter 11, in this section, for more information regarding .CLKF statements.

[21] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

Expression

Expression includes the signal name or expression that defines the clock source.

Important: In PLDs with multiple clocks, you must specify the clock pin; don't use an expression.

All values in an expression for a PLD design are signal names that must be defined in earlier PIN and NODE statements.[22] In the syntax example, when EXT is true, the clock associated with Q0 is asserted.

USE

Multiple .CLKF statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting.

If no clock pin is defined using the .CLKF instruction, registered outputs default to the nominated default pin for the specified device.[23]

Note: For devices without a default clock pin, such as the PAL20RA10, you must define a clock pin.

You can use a group, string, or vector notation to define signals, which is an excellent way to assign a functional equation to several pins.[24]

The next example shows how to use vector notation.

[22] Refer to NODE and PIN, in this chapter, for details about including a forward slash to define polarity.

[23] Refer to Chapter 11, in this section, for details regarding default clock pins.

[24] Refer to FUNCTIONAL EQUATIONS and GROUP, in this chapter, for additional details.

```
;DECLARATION SEGMENT
    PIN                 14..16          Q[0..2]
    PIN                 8..10           CLK[0..2]

;EQUATIONS
    Q[0..2].CLKF        =       CLK[0..2]
```

This generates the following equations.

```
Q[0.CLKF        =       CLK]0]
Q1[1].CLKF      =       CLK[1]
Q[2].CLKF       =       CLK[2]
```

The following example shows how to use a STRING statement.

```
;DECLARATION SEGMENT
    STRING    IN1     'A1 * A2'

;EQUATIONS
Q0.CLKF         =       IN1
```

This generates the following equation.

```
Q0.CLKF         =       A1 * A2
```

The following example shows how to use a GROUP statement.

```
;DECLARATION SEGMENT
    GROUP     CLOCKS  A   B   C

;EQUATIONS
    CLOCKS.CLKF         =       D*E
```

This generates the following equations.

```
A.CLKF          =       D*E
B.CLKF          =       D*E
C.C.KF          =       D*E
```

CLKF

This keyword identifies which clock is used for the synchronized operation of the state machine.

Devices Supported					
PAL26V12	PALCE29M16	PALCE29MA16	PALCE610	PLS30S16	MACH 1
MACH 2					

SYNTAX

You can use this statement only in the state segment of state-machine designs.

Syntax			
	CLKF	Assignment Operator	Clock_pin
Example			
	CLKF	=	CLK1

Definitions

Parameters following the keyword are defined below.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[25]

Clock_pin

Clock_pin is the pin you define as the clock source for the state machine.[26]

- Specify the name exactly as it's defined in the PIN statement in the declaration segment.
- Place a forward slash before the equation to select a falling edge clock. For example,

  /CLKF = CLK1

[25] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

[26] Refer to the specific device datasheet to determine which pin to use.

Note: To define a falling edge clock for PAL29M16, you must use NCLKF instead of the forward slash.

USE

Without a CLKF equation, you cannot specify a non-default clock pin if you use automatic state-bit assignment.

This equation creates clock assignments for both state bits and output registers. [27]

[27] Refer to BOOLEAN EQUATION and .CLKF, in this chapter, for additional details.

CLOCKF

This command generates a clock pulse on dedicated clock pins during simulation. Three test vectors are generated: raise clock, propagate output, and lower clock. In the simulation trace and waveform files, the letter c appears in the header over the first vector for each pulse.

Devices Supported					
PAL16R4	PAL16R6	PAL16R8	PAL16V8	PAL18U8	PAL20R4
PAL20R6	PAL20R8	PAL20V8	PAL20X4	PAL20X8	PAL20X1
PAL22V10	PAL23S8	PAL24R4	PAL24R8	PAL24R10	PAL26V12
PALCE29M16	PALCE29MA16	PAL32VX10	PALCE610	PLS105	PLS167
PLS168	PLS30S16	MACH 1	MACH 2		

SYNTAX

You include this command in the simulation segment or auxiliary simulation file of Boolean and state-machine designs.

Syntax

```
        CLOCKF              Clock_pin
```

Example

```
        ...
        CLOCKF              CLOCK
        ...
```

Definitions

Only details about the clock pin are provided below.

Clock_pin

Clock_pin defines the name of the clock pin used in the PIN statement of the declaration segment. You can also use groups and strings. On some devices, this pin can be either an input or a clock, as described under Use.

Important: You use a blank to separate multiple pin names; no comma is needed.

USE

The CLOCKF command is similar to the SETF command ,[28] except that CLOCKF generates a low-to-high-to-low pulse.

Important: You cannot use a CLOCKF command for registered devices without clock pins. These devices require two SETF lines to generate the CLOCKF pulse.

- If you do not use a SETF command to set the clock signal low before clocking the register, the first clock pulse has no effect.

  Note: At the start of simulation, clock signals must be initialized low.

- If you design a system using multiple banks of independent clock pins and you connect these pins externally for synchronous clock cycles, you must list all clock pins in the CLOCKF command to synchronize these banks during simulation.
- If you do not specify a clock pin in the CLOCKF command, the nominated default clock pin for that device is used.

Some devices, such as the PAL30S16, include a pin you can configure as either an input or a clock.

- If you define the pin as a clock pin using a corresponding .CLKF command , the CLOCKF command generates a clock pulse for that pin.
- If you define the pin as a clock input on the PLS30S16, you can still use the SETF command to create a test vector for that pin.

[28] Refer to SETF, in this chapter, for additional details.

.CMBF

This reserved word allows you to customize a flip-flop for dynamic register bypass.

Devices Supported: PAL32VX10.

SYNTAX

You include this functional equation only in the equations segment of either Boolean or state designs.

Syntax			
	Pn.CMBF	Assignment Operator	Expression
Example			
	OUT1.CMBF	=	IN4 * /IN3

Definitions

All parameters are defined below.

Pn.CMBF

Pn.CMBF is a pin or node name followed by the reserved word .CMBF. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

- You cannot use negative polarity on the left side of the equation.
- You can use the group, string, and vector notation features to define signals, which is an excellent way to assign this statement to several pins and nodes.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[29]

Expression

Expression is the logic you define to be bypassed. In the syntax example, when IN4 and /IN3 are true, OUT1 is true and the flip-flop associated with OUT1 is bypassed.

[29] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

The .CMBF statement overrides the registered pin or node type you defined in the PIN or NODE statement in the declaration segment.[30] The default case is set to combinatorial.

If you have multiple functional equations for the same pin or node, an error occurs during assembly.

[30] Refer to the following topics, in this chapter, for additional information: BOOLEAN EQUATION, COMBINATORIAL, EXPRESSION, FUNCTIONAL EQUATIONS, and REGISTERED.

COMBINATORIAL

This optional reserved word defines the output type for devices with programmable outputs. When a pin or node is defined as combinatorial, the logic output is immediate; the output value is not stored.

Devices Supported: All devices **except** the PAL16R8, PAL20R8, PAL20X10.

SYNTAX

You include the optional reserved word in the PIN or NODE statement of Boolean and state-machine designs.

Syntax			
Pn	Number	Location_name	Storage
Example			
PIN	14	OUT1	COMBINATORIAL
NODE	15	OUT2	COMB

Definitions

Only the reserved word COMBINATORIAL is discussed below.[31]

Storage

Assign COMBINATORIAL as a value to define a specific pin or node storage type. Combinatorial is also the default when you do not specify a storage type.

- Place the reserved word COMBINATORIAL after the pin or node name in the corresponding statement.

- Use either the complete word COMBINATORIAL or the four-letter abbreviation, COMB.

[31] Refer to the following topics, in this chapter, for additional details: DECLARATION SEGMENT, NODE, PIN, and STORAGE.

USE

There are two ways to enter the storage type.

- Use the declaration segment form, select the Storage field, display the option list, and select an option.
- Type the storage value in the appropriate PIN or NODE statement in the PDS file using a text editor.

COMMENT

A comment can explain a statement in the PDS file to assist you or another designer when editing or debugging the design.

Devices Supported: All PLD devices.

SYNTAX

You can place comments anywhere in the design file.

Syntax

```
              ;Comment
```

Example

```
              ;State setup and defaults
              MOORE MACHINE
              DEFAULT_BRANCH HOLD_STATE ;defines default state as holding
              ;State Assignments
              ...
```

Definitions

Details about comments follow.

Comment

A comment is any combination of alphanumeric characters, symbols, or punctuation preceded by a semicolon. The software ignores everything in the line following the semicolon. A comment can be several lines in length; however, each new line must begin with a semicolon.

USE

You can use comments to separate the various segments in either a Boolean or state-machine design.

COMPANY

This keyword begins the statement to define the name of the author's company.

Devices Supported: All PLD devices.

SYNTAX

You use this keyword in the declaration segment of either a Boolean or state-machine design.

Syntax

```
            COMPANY                  Company Name
```

Example

```
            TITLE
            PATTERN
            REVISION
            AUTHOR
            COMPANY                  Advanced Micro Devices, Inc.
            DATE
            CHIP
```

Definitions

Only the descriptor following the keyword, COMPANY, is discussed.

Company Name

The company name is optional and includes any combination of up to 59 alphanumeric characters.

- You can use other symbols or punctuation; however you cannot use the dollar sign, $.
- You can use reserved words in this statement.

USE

There are two ways to enter the company name.

- Use the declaration segment form, select the company field, and type the name.
- Type the company name after the AUTHOR statement and before the date in the PDS file using a text editor.

The following error conditions pertain to the COMPANY statement.

- Without a COMPANY statement, a warning is issued and processing continues.
- With multiple COMPANY statements, an error is reported and processing stops.[32]

[32] Refer to the following topics, in this chapter, for additional details: AUTHOR, DATE, DECLARATION SEGMENT, PATTERN, REVISION, and TITLE.

CONDITIONS

This keyword identifies the beginning of the state transition equations segment.

Devices Supported: All synchronous registered PAL devices.

SYNTAX

This keyword must begin the condition-equations segment, and it must be the last part of the state segment in a PDS file.

Syntax

```
   CONDITIONS
           Name        Assignment Operator         Expression
```

Example

```
   STATE
   ...
   CONDITIONS
           QRUN                 =                  Q2 * /Q1 * /Q0
                                                   + Q3 * Q2
                                                   + Q3 * Q1
                                                   + Q3 * Q0
```

Definitions

All parameters following the keyword CONDITIONS are identified below.[33]

Name

Condition equations define the branching conditions that determine transitions and outputs.

- Ensure each name is unique.
- Include up to 14 alphanumeric characters.
- Do not include operators or reserved words.

[33] Refer to the following topics, in this chapter, for additional details: DEFAULT_BRANCH, DEFAULT_OUTPUT, LOCAL DEFAULT, OPERATOR, STATE, STATE EQUATIONS, and STATE TRANSITION EQUATION.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[34]

Expression

Assign a Boolean expression to define the condition.[35]

- All signal names in the expression must be defined in PIN and NODE statements in the declaration segment of the PDS file.

- Parentheses are allowed in the expression.

- VCC can be specified for unconditional-high signals.

- Multiple product terms are allowed in the expression.

USE

Condition equations define condition names used in state transition equations, as shown in the next example.

```
S1 := C1 -> S2
      C2 -> S3

CONDITIONS        ;begin conditions segment
C1 = I1
C2 = /I1
```

If the condition consists of a single input, the input name can be used in place of a condition name. As shown in the next example.

```
S4 := /I3 -> S5
```

[34] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

[35] Refer to EXPRESSION, in this chapter, for more information.

CONDITIONS

If the condition consists of more than a single input, you must write a condition equation and use the condition name in the state transition equation.

You must avoid conflicting conditions when branching, as shown below; the design must be changed to prevent overlapping conditions. In this example, it is impossible to determine whether S1 goes to S2, or S1 goes to S3, when A, B, and C all equal 1 or 0.

```
;conflicting conditions example
S1 := C1 -> S2
    + C2 -> S3
CONDITIONS        ;begin conditions segment
C1 = A * B
C2 = A * C
```

DATE

This keyword begins the statement that identifies the creation date of the design.

Devices Supported: All PLD devices.

SYNTAX

You use this keyword in the declaration segment of either a Boolean or state-machine design.

Syntax

```
          DATE                  Creation Date
```

Example

```
          TITLE
          PATTERN
          REVISION
          AUTHOR
          COMPANY
          DATE                  01/09/91
          CHIP
```

Definitions

Only the descriptor following the keyword DATE is discussed.

Creation Date

The creation date is a set of numbers that represent the month, day, and year the design was created or edited. The date may be entered as any 60-character string. Place the DATE statement after the COMPANY statement and before the CHIP statement.

USE

The following error conditions pertain to the DATE statement.

- Without a DATE statement, a warning is issued and processing continues.

- With multiple DATE statements, an error is reported and processing stops.[36]

[36] Refer to the following topics, in this chapter, for additional details: AUTHOR, DATE, DECLARATION SEGMENT, PATTERN, REVISION, and TITLE.

DECLARATION SEGMENT

The declaration segment is the first segment of any PDS file. It identifies basic design information, the chip name and device type and PIN and NODE statements required to process the design, and special definitions. A form automates entry of this information when you begin a design.

Devices Supported: All PLD devices.

SYNTAX

This segment is required for all designs and must be the first segment in the PDS file.

Syntax

```
;------------------------------Declaration Segment----------------------------
design information
;------------------------------Pin Declarations ----------------------------
Pin / Node statements(s)
;Special definitions
```

Example

```
;------------------------------Declaration Segment----------------------------
TITLE     16 Bit Counter (up/down); preplaced w/standard settings
PATTERN   XS.PDS
REVISION  2
AUTHOR    Gail Tiberi
COMPANY   Mystique, Inc.
DATE      08/09/90
CHIP      Counter                PAL16R6
;-----------------------------Pin Declarations -----------------------------
PIN 2 INP1 REG
NODE ? INP2 COMB
:
SIGNATURE = V2_5/90
STRING IN1 'A1 + /A2 + A3'
GROUP BANK1 OUT1, OUT2
...
```

Definitions

Groups of descriptors within the declaration segment are discussed next.[37]

[37] Refer to the following topics, in this chapter, for additional details: AUTHOR, CHIP, COMPANY, DATE, GROUP, NODE, PATTERN, PIN, REVISION, SIGNATURE, STRING, and TITLE.

DECLARATION SEGMENT

Design Information

Design information includes statements that document basic information about the design. The following statements must appear in the order shown below.

- Title
- Pattern
- Revision
- Author
- Company
- Date
- Chip Device

These statements can be useful in describing the function of the design.

Pin Declarations

PIN and NODE statements required to process the design must follow the device type. Each statement defines the name, location number, and optional storage type assigned to each device pin in the design.

Special Definitions

Special definitions are optional statements you can include after PIN and NODE statements in the declaration segment.[38]

- STRING statements can be used with any device.
- GROUP statements can be used with any device.
- SIGNATURE statements are device specific.

Though not required, it is a good idea to place each statement on a separate line. These statements can appear in any order.

[38] Refer to the following topics, in this chapter, for additional details: GROUP, SIGNATURE, and STRING.

DECLARATION SEGMENT

USE

The following error conditions can occur.

- If design information is incomplete, a warning is issued during processing.
- If multiple information statements appear, processing stops.
- If PIN and NODE statements are incomplete or incorrect, processing stops.
- For PAL and PLS devices, the signal list maps into a DIP package. For MACH devices, it maps into a PLCC package.

DEFAULT_BRANCH

This keyword begins the statement that defines the global default branch for state machines. There are three possibilities.

- A specific state
- The present state
- The next state listed on the left side of the state transition equation

The default branch will be executed if none of the conditions in the transition equation are satisfied and no local default is defined.

This statement can also utilize the complement array of certain devices.

Devices Supported: All synchronous registered PAL devices.

SYNTAX

Include the keyword once only, anywhere in the state segment of state-machine designs.

Syntax

```
    DEFAULT_BRANCH                      State         Comp
    DEFAULT_BRANCH                      HOLD_STATE
    DEFAULT_BRANCH                      NEXT_STATE
```

Example

```
    STATE                                                      ;State Setup and Defaults
    DEFAULT_BRANCH                      INIT                   ;defaults to INIT state
    DEFAULT_BRANCH                      INIT          Comp    ;uses complement array
    DEFAULT_BRANCH                      HOLD_STATE             ;defaults to present state
    DEFAULT_BRANCH                      NEXT_STATE             ;defaults to next state
    ...
```

Definitions

The next discussions explain parameters following the keyword DEFAULT_BRANCH. For more information about the various states and how they are used, refer to Use.

State

State defines the default state enabled when the next state cannot be determined from the transition equations. The state name must adhere to the following guidelines.

- It must be unique.
- It can include up to 14 alphanumeric characters.
- It cannot include operators or reserved words.

Comp

Comp defines the name of the complement array node, which must be listed in the pin and node statements. This use results in shorter equations and fewer product terms; however, it slows device performance.

- You can use a complementary array node only for the PLS105, PLS167, PLS168, and PLS3016.
- You cannot use the complement array when specifying the next or hold state branches.

HOLD_STATE

When the next state cannot be determined from the transition equations, the machine will remain in its present state.

If no DEFAULT_BRANCH or local default branch are defined, the default becomes HOLD_STATE.

NEXT_STATE

When the next state cannot be determined from the transition equations, the machine will transition to the next state listed in the PDS file.

DEFAULT_BRANCH

USE

The software recognizes two types of default states, global and local.[39]

- A local default state applies to a specific state and overrides the global default.

  Using a DEFAULT_BRANCH statement can eliminate typing a LOCAL DEFAULT statement for each state.

- The global default state applies to all states.

The following guidelines apply.

- When the next state cannot be determined from the design, the local or global default provides the state so the machine knows where to branch next.

- When you include multiple DEFAULT_BRANCH statements, only the last one is used.

Note: DEFAULT_BRANCH does not work for undefined states.

[39] Refer to the following topics, in this chapter, for additional details: DEFAULT_OUTPUT, LOCAL DEFAULT, .OUTF, OUTPUT_HOLD, and STATE.

DEFAULT_OUTPUT

This keyword begins the statement that defines a global default, allowing you to specify the next output-pin value when the value cannot be determined from the design.

Devices Supported: All registered PAL devices.

SYNTAX

You include this keyword at the beginning of the state segment in state-machine designs. It must follow setup statements and precede state-assignment equations.

```
Syntax

    DEFAULT_OUTPUT                    Output_pins

Example

    STATE                                         ;State Setup and Defaults
    ...
    DEFAULT_OUTPUT                    /OUT1       ;State Equations
```

Definitions

Only the parameter following the keyword is defined below.

Output_pins

Output_pins defines default output pin values when they are not defined in the current state. You must separate output pin values with a blank; multiple blanks are reduced to one. You can specify the values you want regardless of polarity; polarity is adjusted automatically during processing.

- If you specify the output value with nothing before the pin name, a logical 1, or high, is assumed.

- If you specify the output value by placing a forward slash before the pin name, a logical 0, or low, is assumed.

Note: If the output pin and state bit are the same, **do not** use output equations nor a DEFAULT_OUTPUT statement.

Also: You can use the reserved word OUTPUT_HOLD as a default in PLS devices. In this case, the output pin value is held to its current state when a default is used.[40]

USE

You can simplify output equations by assigning the most common output value as the default, then include an output value in the equation only when it differs from the default.[41]

[40] Refer to OUTPUT_HOLD, in this chapter, for additional details.

[41] Refer to the following topics, in this chapter, for additional details: DEFAULT_BRANCH, LOCAL DEFAULT, .OUTF, and STATE.

EQUATIONS SEGMENT

This segment of the PDS file contains the Boolean equations you create to produce the desired device behavior.

Devices Supported: All PLD devices.

SYNTAX

You use the keyword, EQUATIONS, at the beginning of the equations segment in Boolean-based designs.

Syntax

```
   ;----------------------Boolean Equations Segment------------------------
   EQUATIONS
   Boolean equations
```

Example

```
   EQUATIONS                            ;Equations Segment
   QA =  Q2 * /Q1 * /Q0
              + Q3 * Q2
              + Q3 * Q1
              + Q3 * Q0
```

Definitions

The following discussions define elements in the equations segment.

Equations

You use this keyword to define the beginning of the equations segment in a PDS file.

Boolean Equations

You assign equations that include an output pin or node name, an assignment operator, and an expression. Equations can be standard Boolean, Case, or IF-THEN-ELSE constructs.[42]

[42] Refer to BOOLEAN EQUATION, in this chapter, for additional details.

EQUATIONS SEGMENT

USE

The keyword EQUATIONS is required when Boolean equations follow.

Note: The PALASM 4 software allows both Boolean equations and state-machine descriptions in the same design file.

EXPRESSION

Expressions can be used in both Boolean and state-machine designs as part of a Boolean or functional equation, **or** in a state equation, **or** in equations in simulation commands. [43]

Devices Supported: All PLD devices.

SYNTAX

You can include expressions in the equations, state, or simulation segments of any PDS file or in a separate simulation file.[44]

```
Syntax

             Pn        Assignment Operator        Expression
Example

   EQUATIONS
             Q1                 =                 Q1 * Q0
                                                  +  /Q1 * /Q0
                                                  +  /FRM1B * SOF * SUNK
                                                  + RSTB
             ...
```

Definitions

All parameters are described next. Additional details are provided under Use.

Pn

Pn identifies the name of the output pin or node defined in the corresponding statement in the declaration segment of a PDS file. Once declared, you can include the name in an equation or expression.

[43] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, FUNCTIONAL EQUATIONS, and STATE EQUATIONS. Also, refer to NODE and PIN for details about including a forward slash to define polarity.

[44] Refer to Section II, Chapter 6, for details about using a separate simulation file.

Assignment Operator

The assignment operator is a symbol that defines the appropriate operation interpreted by the software when processing design files.[45]

Expression

Expression assigns a group of signal values or product terms on the right side of an equation, represented as product, *, and sum, +, lines. The product operator is a logical AND function and the sum operator is a logical OR function.

All values in an expression for a PLD design are signal names that must be defined in PIN and NODE statements, in the declaration segment of the PDS file, before their use elsewhere.[46]

Expressions in Boolean equations use Boolean-logic operators to combine the values of pins and nodes. **State equations** use Boolean expressions to define states and their outputs. **Functional equations** use Boolean expressions to define inputs.

USE

When using expressions, the following conventions should be observed.

- Place the expression on the right side of a Boolean, functional, or state equation.
- Use blanks or tab characters between pin or node names and logic operators.
- Place each part of the expression that uses the sum operator, +, on a separate line.

[45] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

[46] Refer to NODE and PIN, in this chapter, for details about including a forward slash to define polarity.

EXPRESSION

The following table lists the order of precedence for expressions without parentheses.

PRECEDENCE	OPERATOR/DEFINITION	
1	/	NOT
2	*	AND
3	+	OR
4	:+:	XOR
4	:*:	XNOR

FLOATING PINS AND NODES

A question mark in the location field of a PIN or NODE statement **floats** the pin or node. Each floating pin and node is automatically placed by the software during the fitting process.[47] Floating pins and nodes can increase the probability of a fit in all MACH-device designs.

Devices Supported: All MACH device designs.

SYNTAX

You can use the question mark only in the **declaration segment** of Boolean or state-machine designs.

Syntax

Keyword	Location_number	Name	Storage
Example			
Pin	?	Sensor1	Registered
Node	?	Equation1	Combinatorial
Pin	?	Output1	Latched

Definitions

Only descriptors following the keywords PIN or NODE are discussed.

Location_number

Place a question mark in the location-number field to float the corresponding pin or node. The signal is assigned to a specific pin automatically during compilation.

Important: Do not float NODE 1.

[47] You may want to group signals within the same MACH-device block. Refer to MACH_SEG, in this chapter, and to Chapter 11, in this section, for more information.

To assign a fixed location to a pin or node, you enter a number or range of numbers in the statement.

In PDS files produced from **schematics**, the question mark appears in the pin or node location field unless you assigned a fixed location. To assign a location in a schematic, you use the Part field 1 command on the OrCAD/SDT III Edit Part menu.[48]

Name

Name defines the name of the pin or node. Each name must be unique and must follow the location field.

- Begin the name with an alpha character; use any combination of up to 14 upper- or lowercase alphanumeric characters: A–Z and 0–9.

> **Important**: Keep names in a schematic less than or equal to 14 characters. Part and pin names in the schematic may be concatenated while the data is converted into PDS format. Any name longer than 14 characters is automatically truncated.

- Use underscore, _, as a connector and a forward slash to affect polarity; **no** other symbols or punctuation are allowed and no keywords, reserved words, or logic operators are allowed.

[48] Refer to Section III, Chapter 7, for more information about assigning attributes in a schematic.

FLOATING PINS AND NODES

Note: The forward slash is not supported for schematic-based designs.

Storage

Storage defines the optional storage type for a pin or node, and must follow the pin or node name.[49] Enter the reserved word or abbreviation listed below; the default is combinatorial.

• COMBINATORIAL	or	COMB
• REGISTERED	or	REG
• LATCHED	or	LAT

USE

Floating pins and nodes is recommended to assist the fitting process. You can assign a fixed location when needed; however, this may result in a no-fit situation.

Important: Do not float NODE 1.

[49] Refer to PAIR, in this chapter, for details about pairing a node with a pin.

FOR-TO-DO

This construct, also known as a FOR loop, defines the number of times to repeat a simulation block bounded by BEGIN and END statements.

Devices Supported: All PLD devices.

SYNTAX

You use this construct in either the simulation segment of a PDS file or in an auxiliary simulation file for any PLD design.

Syntax

```
FOR Variable := Start TO End DO
        BEGIN
                          Action Statement
        END
```

Example

```
SIMULATION
SETF /OE /CLOCK COUNT
FOR X := 1 TO 20 DO
        BEGIN
                          CLOCKF CLK
        END
```

Definitions

All elements of the statement are defined below.

Variable

Variable defines the index used in the FOR loop.

- A variable can include up to 14 alphanumeric characters and must end with :=, which is a convention, not the registered operator.
- A variable cannot be used elsewhere in the design file.

Start

Start defines the lower limit of the loop.

- Use only integers following the variable := and preceding the word TO.
- Ensure the number is greater than or equal to zero and less than the upper limit.

Important: If the start and end numbers are the same, the task is executed once.

End

END defines the upper limit of the loop. The number must be greater than or equal to zero.

- Use only integers following the word TO and preceding the word DO.
- Ensure the number is greater than or equal to zero and more than the lower limit.

Important: If the start and end numbers are the same, the task is executed once.

Action Statement

Action statement defines the simulation task to be repeated using this loop. The following rules apply.

- Enclose each statement with BEGIN and END.
- Use any valid simulation statements between BEGIN and END, including nested FOR loops.

USE

There is no limit to the number of constructs you can include in a design. However, minimal nesting makes the file easier to follow and faster to compile. You can nest FOR loops within other FOR loops and within the following statements and constructs.[50]

- IF-THEN-ELSE
- WHILE-DO

It's best to use the WHILE-DO construct when you do not have a fixed number of repetitions and you prefer to base repetitions on a condition.

For example, you can test the FOR loop by nesting WHILE-DO or IF-THEN-ELSE constructs within the FOR loop, as shown below.

```
FOR X := 1 to 5 DO
  BEGIN
        IF X = 1 THEN
        BEGIN
              SETF A * B
        END
  END
  ...
```

[50] Refer to the following topics, in this chapter, for additional details: CASE, IF-THEN-ELSE, SIMULATION, and WHILE-DO.

FUNCTIONAL EQUATIONS

Functional equations control signals that cannot be regulated directly through a pin or node. These are typically control signals associated with an input/output pin or node, such as set, clock, and three-state. Functional equations work like Boolean equations, in that the function is asserted when the expression is true.

Devices Supported: Refer to the specific functional-equation reserved word, such as .CLKF, for supported devices.

SYNTAX

You include functional equations anywhere in the equations segment of Boolean and state-machine designs.

Syntax

```
   EQUATIONS
   Pn.Function                Assignment Operator    Expression
```

Example

```
   EQUATIONS
   Q0.CLKF                            =              CLOCK
   Q1.SETF                            =              SET * /RST
   Q1.RSTF                            =              RST * /SET
```

Definitions

All parameters are described next. Additional details are provided under Use.

Pn.Function

Pn.Function defines the pin or node name, as defined in the PIN and NODE statements, and the specific function, which can be any of the following.

FUNCTION	DEFINITION
.CLKF	Clock control
.CMBF	Register bypass control
.PRLD	Register preload control
.RSTF	Flip-flop power up or reset control
.SETF	Flip-flop preset control
.TRST	Three-state buffer control

You **cannot** include multiple functional equations for the same pin or node. If you do, an error is reported during either assembly or fitting. However, you can use the group, string, and vector features to define signals, which is an excellent way to assign a function to several pins and nodes.

> **Note**: The .CLKF statement is the only functional equation where you can use negative polarity on the left side of the equation to define a falling edge clock on devices that support this feature.
>
> The .J, .K, .R, .S, .T, .T1, and .T2 equations are not functional equations. [51]

Assignment Operator

A symbol that defines a specific operation as interpreted by the software when processing design files.[52] Only the equal sign, =, can be used with functional equations. The registered assignment operator, :=, can be used for registered operations.

Expression

Expression is a group of signal values or product terms, on the right side of an equation, represented as product and sum lines.[53] The product operator, *, is a logical AND function and the sum operator, +, is a logical OR function. You can use strings and vectors in an expression.

[51] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, .CLKF, .CMBF, .PRLD, .RSTF, .SETF, and .TRST.

[52] Refer to ASSIGNMENT OPERATOR, in this chapter, for more information.

[53] Refer to EXPRESSION, in this chapter, for additional details.

FUNCTIONAL EQUATIONS

All values in an expression for a PLD design are signal names that must be defined before their use. The name must be defined in PIN and NODE statements in the declaration segment of the PDS file.

USE

To use negative polarity on devices that support clock polarity, just omit the forward slash after the period, ., extension, for example, Q0./CLKF. For the PALCE29M16, place the forward slash before the equation, for example, /Q0.CLKF.

The defaults for each functional equation are as follows.

- .CLKF always defaults to the default clock pin for the device.
- .CMBF defaults to combinatorial.
- .PRLD defaults to ground.
- .SETF and .RSETF both default to a bank if they are part of a bank expression. Otherwise, they default to ground.
- .TRST defaults to VCC if an output equation is defined. Otherwise, it defaults to ground.

GND

You can include this reserved word in an equation to hold a pin, node, or functional equation unconditionally low. GND is always treated as logical zero.

Devices Supported: All PLD devices.

SYNTAX

You use the reserved word, GND, in the equations segment of Boolean and state-machine designs.

```
Syntax

   Pn or
   Functional Equation   Assignment Operator      GND

Example

   OUT4                         =                  GND
   OE1.TRST                     =                  GND
```

Definitions

The element preceding the reserved word is described below.

Pn or Functional Equation

Pn or functional equation defines the element to be held low.

- The pin or node name defined in the PIN or NODE statement of the declaration segment
- The Pn.function defined in an earlier functional equation.[54]

[54] Refer to the following topics, in this chapter, for additional details: FUNCTIONAL EQUATIONS, NODE, and PIN.

USE

GND is normally used in functional equations. You can enter 0 instead of GND anywhere you want an unconditional low value.[55]

Important: You must define the GND pin in the pin statements.

[55] Refer to VCC, in this chapter, for additional details.

GROUP

This keyword clusters several pins or nodes under a single name. You can then use the group name in the equations or state segment of your design.

Devices Supported: All PLD devices.

SYNTAX

You use the keyword, GROUP, in the declaration segment of Boolean and state-machine designs.

Syntax

```
           GROUP                    Group_name                      Pn_list
```

Example

```
   PIN 15 OUT1 COMB
   PIN 16 OUT2 COMB
   NODE 5 NB   COMB
           GROUP                    BANK1                           OUT1 OUT2 NB
   :
   EQUATIONS
   :
   BANK1 = IN1 * IN2 * IN3
```

DEFINITIONS

Parameters following the keyword, GROUP, are defined below.[56]

Group_name

Group_name is the name assigned to a cluster of pins or nodes. This name can then be used in the equations or state segments of a design to refer to the entire cluster without having to list them separately. Follow the rules below.

- Assign a unique name of up to 14 alphanumeric characters.

  Do not use keywords, operators, or reserved words.

[56] Refer to MACH_SEG, in this chapter, for details about using a group name to cluster signals within a MACH device.

- Place the name after the keyword GROUP and before the PIN or NODE statements.

The syntax example shows BANK1 as a group name.

Pn_list

This is a list of the pins or nodes, defined in the pin and node statement, that are associated with the group name.

- Place this list after the group name.

- Separate pin and node names by a single blank; multiple blanks are reduced to one.

  Do not use commas or any other punctuation. You can use the range operator, [ ], to define a group of pins.

In the syntax example, the equation BANK1 = IN1 * IN2 * IN3 is automatically expanded into the three equations shown below.

```
     OUT1 = IN1 * IN2 * IN3
     OUT2 = IN1 * IN2 * IN3
     NB = IN1 * IN2 * IN3
```

USE

You can place the group statement either before or after SIGNATURE and STRING statements. You can use the group name, wherever appropriate in the design file, in place of the defined group of pins or nodes. This can simplify the equations segment of the file by reducing the number of equations required.

- You can only use the group name on the left side of an equation.

- You can also use the group name to define pins or nodes in the simulation segment.

GROUP

Note: You can use a group name when controlling registers consisting of banks of flip-flops with common reset or other control lines.

For example, use a group name to combine four outputs that share a common reset line. The software then writes four .RSTF equations.[57]

```
;DECLARATION SEGMENT
    PIN        2       A       REG
    PIN        3       B       REG
    PIN        4       C       REG
    PIN        5       D       REG
    PIN        6       E       REG
    GROUP      QRST    A   B   C   D

;EQUATIONS
    QRST.RSTF =        RES

;This expands to

    A.RSTF     =       RES
    B.RSTF     =       RES
    C.RSTF     =       RES
    D.RSTF     =       RES
```

[57] Refer to the following topics, in this chapter, for additional details: DECLARATION SEGMENT, NODE, and PIN.

IF-THEN-ELSE, EQUATIONS

This construct provides a conditional statement for Boolean logic. This construct literally means "if the condition is true, do this; if not, do that."[58]

Devices Supported: All PLD devices.

SYNTAX

You use this construct in the equations segment of Boolean and state-machine designs.

Syntax

```
   IF (Condition) THEN
             BEGIN
                                     Boolean equations
             END
   ELSE
             BEGIN
                                     Boolean equations
             END
```

Example

```
   EQUATIONS
   IF (/I1,/I2 = #b11) THEN
             BEGIN
                                     03 = A * B
             END
   ELSE
             BEGIN
                                     03 = A * /B
             END
   ...
```

Definitions

Both elements of the statement are defined next.

[58] Refer to IF-THEN-ELSE and SIMULATION, in this chapter, for details on using this construct in the simulation segment or separate simulation file of a design.

Condition

Condition defines any Boolean expression whose form consists of a pin, signal, range, or vector, an equal sign, =, and a binary, decimal, hexadecimal, or octal radix number.

- You can use more than one condition if you separate them by commas.

  The software ANDs multiple conditions together.

- You can use parentheses to enclose the condition; they are optional, but it is better to include them. You can nest parentheses.

- You cannot use group names or arithmetic expressions.

- You can use a test condition in place of any variable name in a Boolean expression as in the example, A * B * (C,D = 1).

The software ANDs conditions with vectors. For example:

```
IF (A[1..3])  becomes IF (A[1] * A[2] * A[3])
IF (/A[1..3]) becomes IF (/(A[1] * A[2] * A[3]))
```

Boolean Equation

This equation defines any Boolean equation or set of equations, as well as IF-THEN-ELSE, and CASE constructs. The equation must be enclosed by BEGIN and END statements.

IF-THEN-ELSE, EQUATIONS

USE

You can specify how the software treats default values for IF-THEN-ELSE constructs by selecting one of the following options on the File/Set up/Logic Synthesis Options form.

OPTIONS	DEFINITIONS
Don't care	Unspecified default conditions are assumed to be don't care.
Off	Unspecified default conditions are assumed to be false.

The don't-care option requires you specify both the on and off sets. The off option requires you to specify only the on sets; the software assumes all other conditions to be off.

You may lose signals from the design if you select the don't-care option and do not specify all the default conditions. If the software treats these signals as don't care, they will be eliminated from the design during logic reduction.

Important: When translating designs created with PLPL, you must select the off option because PLPL treats unspecified default conditions as false.

There is no limit to the number of constructs you can have in your design. However, minimal nesting makes the file easier to follow and faster to compile.[59]

[59] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, CASE, IF-THEN-ELSE, and SIMULATION.

IF-THEN-ELSE, EQUATIONS

You can use an IF clause without an ELSE clause, but no logic is defined when the IF clause is false. In the case of multiple or nested IF-THEN-ELSE statements, an ELSE clause always matches the last IF-THEN clause.

You can nest IF-THEN-ELSE constructs within CASE and other IF-THEN-ELSE constructs. The following examples show how the software expands IF-THEN-ELSE constructs.

IF -THEN-ELSE	EXPANDS TO						
IF (A) THEN B = 1	B	=	1 • A	that is	B	=	A
IF (A) THEN B = 1 ELSE B = 0	B	=	VCC * A	that is	B	=	A
	/B	=	VCC • /A	that is	/B	=	/A
IF (A) THEN B = 0	/B	=	VCC • A	that is	/B	=	A
IF (A) THEN /B = 1	/B	=	1 • A	that is	/B	=	A

IF-THEN-ELSE, SIMULATION

This construct provides conditional statements during simulation. This construct literally means "if the condition is true, do this; if not, do that."

Devices Supported: All PLD devices.

SYNTAX

You use this construct in Boolean and state-machine designs.

Syntax

```
IF (Condition) THEN
    BEGIN
       Task
    END
ELSE
    BEGIN
       Task
    END
```

Example

```
SIMULATION
SETF /OE /CLOCK COUNT
    IF (J = 5) THEN
       BEGIN
          CHECK Q0
       END
    ELSE
       BEGIN
          CHECK /Q0
       END
...
```

Definitions

Both elements of the statement are defined below.

Condition

Condition defines any Boolean expression.

- You can use more than one condition if you separate them by commas.

  The software ANDs multiple conditions together.

- You can use parentheses to enclose the IF condition. However, you cannot nest parentheses.

The condition can be any Boolean expression of logic signals or mathematical equality: =, >, <, >=, <=, and <>.

Task

Task defines the simulation task the software performs during the IF-THEN-ELSE loop. You use BEGIN and END statements to enclose the task and indent them to make your PDS file easier to follow.

USE

There is no limit to the number of constructs you can have in your design. However, minimal nesting makes the file easier to follow and faster to compile.[60]

You can nest IF-THEN-ELSE constructs within other IF-THEN-ELSE constructs and with the following statements.

- CASE
- FOR-TO-DO
- WHILE-DO

If the condition is false when the construct is reached, the task is not executed.[61]

Note: If you nest the IF-THEN-ELSE construct in a FOR-TO-DO construct, the condition can also be the index variable of the FOR-TO-DO construct. You cannot use an index variable outside its defining FOR-TO-DO construct.

[60] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, EQUATIONS SEGMENT, CASE, and IF-THEN-ELSE.

[61] Refer to the following topics, in this chapter, for additional details: CASE, FOR-TO-DO, and WHILE-DO.

.J EQUATION

This equation defines when to set the J input on J type flip-flops high.

Devices Supported: PALCE610.

SYNTAX

You use the .J equation in the equations segment of Boolean or state-machine designs.

```
Syntax

           Pn.J        Assignment Operator     Expression
Example

   EQUATIONS
           ...
           Q1.J                =               IN1 * /IN2
           ...
```

Definitions

All parameters are defined below.

Pn.J

Pn.J identifies the pin or node associated with the J flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

Assignment operator defines a specific operation as interpreted by the software when processing design files.[62]

Expression

Expression identifies the logic that defines when the input on .J type flip-flops is set high. In the example, when IN1 is true and IN2 is false, the flip-flop associated with the pin or node Q1 is set high.

[62] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .J equation anywhere in the equations segment. Follow the rules below.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and processing stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.J is not allowed.
- You can use group, string, and vector notation to define signals. This is an excellent way to assign the .J equation to several pins.[63]

[63] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

.K EQUATION

This equation defines when to set the K input on K-type flip-flops high.

Devices Supported: PALCE610.

SYNTAX

You use the .K equation in the equations segment of Boolean or state-machine designs.

```
Syntax

              Pn.K          Assignment Operator     Expression
Example

   EQUATIONS
   ...
              Q1.K                   =              IN1 * /IN2
   ...
```

Definitions

All parameters are defined below.

Pn.K

Pn.K identifies the pin or node associated with the K flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[64]

Expression

Expression identifies the logic that defines when the input on .K type flip-flops is set high. In the syntax example, when IN1 is true and IN2 is false, the flip-flop associated with the pin or node Q1 is set high.

[64] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .K equation anywhere in the equations segment. Follow the rules below.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.K is not allowed.
- You can use group, string, and vector notation to define signals. This is an excellent way to assign the .K equation to several pins.[65]

[65] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

LATCHED

This reserved word defines the output data storage type on devices that allow latched outputs. A latched output functions as a combinatorial output until the data is latched. Once latched, the last data present is held regardless of input changes.

Device Support					
PAL10H20EG8	PAL10H20G8	PALCE29M16	PALCE29MA16	PAL16V8HD	MACH 2

SYNTAX

You include this optional reserved word in the PIN or NODE statement of Boolean and state-machine designs.

Syntax				
	Pn	Number	Name	Storage
Example				
	PIN	3	OUT1	LATCHED
	NODE	4	OUT2	LAT

Definitions

Only the reserved word is discussed below.

Storage

Storage defines the pin or node storage type. If you do not specify a storage type, combinatorial is the default.

- Place the reserved word LATCHED after the pin or node name in the corresponding statement.

- Use either the complete word LATCHED or the three-letter abbreviation, LAT.

USE

There are two ways to enter the storage type.

- Use the declaration segment form: Select the Storage field, display the option list, and select an option.

- Type the storage value in the appropriate PIN or NODE statement in the PDS file using a text editor.

The PALCE29M16 and PALCE29MA16 have programmable latches. The polarity of the latch can be inverted by placing a forward slash before the .CLKF functional equation. The default is active-low enable. The following is an example of an active-low enable equation.

```
PIN 1    EN
PIN 3    OPIN
...
/IOPIN.CLKF = EN
```

The following equation provides an active high enable.

```
IOPIN.CLKF = EN
```

The final polarity of the latch enable as seen from outside the chip is determined by the following conditions.

- The polarity of the latch enable as defined in the PIN statement

- The polarity of the .CLKF functional equation

- The polarity of the latch enable on the right-hand side of the .CLKF functional equation

The latch enable input controls whether the flip-flop is latched or not. On the PAL10H20EG8 and PAL10H20G8 the latch is transparent when the gate pin is low. The latch is enabled when the pin is high.[66]

[66] Refer to the following topics, in this chapter, for additional details: .CLKF, COMBINATORIAL, PIN, and REGISTERED.

LOCAL DEFAULT

This branch of a state transition equation is executed if none of the conditions in the equation are satisfied. Local defaults override global defaults.[67]

Devices Supported					
PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Include the local default (state branch) in state transition equations of state-machine designs.

Syntax

```
   State1 := condition1 -> state2
            + condition2 -> state3
            +-> default state
```

Example

```
   STATE

   ;State transition equations
   RED := NOTRAFFIC -> GRN
            + RED2 -> YLW
            +-> RED
   ...
```

Definitions

Only the term Default is discussed below.

Default State

This parameter defines the next state in a state-machine transition equation when that state cannot be determined from previous conditions.

[67] Refer to DEFAULT_BRANCH and DEFAULT_OUTPUT, in this chapter, for additional details regarding global defaults.

- You must use the default operator, +->, to define a local default.
- You can define only one local default state for each state transition equation.

USE

Defaults ensure the state machine does not behave unpredictably if none of the conditions in the state transition equation is satisfied.

A local default overrides any global default.

The local default branch must be the last branch in the state-transition equation.

Global defaults are defined by the DEFAULT_BRANCH statement.

MACH_SEG_A
MACH_SEG_B
MACH_SEG_C
MACH_SEG_D

These reserved words can be used as the name in a GROUP statement to cluster signals within a single block of a MACH device. The same control logic is applied to all signals in the block.

Once declared, you can include the group name in any equation rather than writing a separate equation for each pin or node in the group.

Devices Supported: All MACH device designs.

SYNTAX

You can use the reserved word only in the GROUP statement in the **declaration segment** of Boolean or state-machine designs.

```
Constructs

              GROUP                   Group_name              Pn_list

Example

   pin/node statements ...
              GROUP                   MACH_SEG_A              R[0]    R[1]    R[2]
                                                              R[3]    O[0]    O[1]
                                                              O[2]    O[3]    O[4]
                                                              O[5]    O[6]    O[7]
```

Definitions

Only descriptors following the keyword, GROUP, are discussed.

MACH_SEG_A
MACH_SEG_B
MACH_SEG_C
MACH_SEG_D

These reserved words identify the block of a MACH device within which the named group of signals will be clustered. The block you specify must be **one** of the following.

- **MACH 110**
  MACH_SEG_A
  MACH_SEG_B

MACH_SEG_A, MACH_SEG_B, MACH_SEG_C, MACH_SEG_D

- **MACH 210**
  MACH_SEG_A
  MACH_SEG_B
  MACH_SEG_C
  MACH_SEG_D

Once declared, you can use the name **either** on the left side of an equation, as shown under Use, **or** to define pins or nodes in the simulation segment or file.

In PDS files produced from converted schematic designs, signals are clustered into one block when a common value is found in Part field 2 of certain macros.[68]

Pn_list

Pn_list identifies the pins or nodes to be included in the group. This list must follow the group name.

- Names must match those used in previous PIN or NODE statements.

  You can include a range operator, [ ], to define a group of pins or nodes if they are so defined in previous statements.

- Blanks or tab characters should be used to separate each pin or node listed; no [Return] characters are allowed.

[68] Refer to Section III, Chapter 7, for more information about assigning attributes in a schematic.

MACH_SEG_A, MACH_SEG_B, MACH_SEG_C, MACH_SEG_D

USE

Using the reserved word as a group name can be helpful when modifying a design that doesn't fit.[69] The following example shows a declared group, MACH_SEG_A, and its use in an equation in the PDS file.

```
;... pin/node statements ...
 GROUP   MACH_SEG_A R[0]  R[1]  R[2]  R[3]  O[0]  O[1]  O[2]  O[3]  O[4]  O[5]
         O[6]  O[7]

;... equations ...
 MACH_SEG_A.TRST  =  IN [1]
```

The equation above enables all outputs in block A when input IN [1] is high. The next example shows how the previous equation is automatically expanded during software processing.

```
R[0].TRST = IN[1]
R[1].TRST = IN[1]
...
R[3].TRST = IN[1]
O[0].TRST = IN[1]
O[1].TRST = IN[1]
...
O[7].TRST = IN[1]
```

[69] Refer to Section II, Chapter 5, for strategies to use when a design does not fit.

MASTER_RESET

This reserved word selects the preset function on PLS devices that provide a preset/enable pin.

Devices Supported		
PLS105	PLS167	PLS168

SYNTAX

You use this reserved word in the state segment of state-machine designs.

Syntax

```
    MASTER_RESET
```

Example

```
    MEALY_MACHINE
    MASTER_RESET
    OUTPUT_HOLD OUT1 OUT2
```

Definitions

Only the reserved word is defined below.

MASTER_RESET

This reserved word dedicates the preset/enable pin to active high preset. Conversely, the reserved word OUTPUT_ENABLE dedicates the preset/enable pin to active-low output enable.

- You can place MASTER_RESET anywhere within the setup and default statements.

  It must precede the state-assignment and transition equations.

- You cannot use both MASTER_RESET and OUTPUT_ENABLE in the same design file.

When the device is preset, the state machine goes to the state which has a value of all 1s.

USE

The software selects preset as the default if you do not use MASTER_RESET or OUTPUT_ENABLE. If you write a .SETF equation and do not use MASTER_RESET or OUTPUT_ENABLE, the software selects preset. If you write a .TRST equation, the software selects output enable.[70]

[70] Refer to OUTPUT_ENABLE and STATE, in this chapter, for additional details.

MEALY_MACHINE

This reserved word identifies the type of state machine you are designing.

Devices Supported: All PLD devices.

SYNTAX

Include the reserved word in the state segment of state-machine designs.

```
Syntax

   MEALY_MACHINE
```

```
Example

   STATE                            ;State Setup and Defaults
   MEALY_MACHINE
```

Definitions

Only the reserved word is defined below.

MEALY_MACHINE

This reserved word defines a state-machine design as one of two possible types, either Mealy or Moore. If you do not define a type in the state segment of the design, the program defaults to Mealy machine.

A Mealy machine determines its outputs from the present state and inputs.[71]

USE

You can place this reserved word statement anywhere within the STATE segment. However, for design clarity, the following guidelines are advised.

- Place the reserved word statement at the beginning of the STATE segment before the state global defaults.

[71] Refer to MOORE_MACHINE and STATE, in this chapter, for additional details.

- Use the reserved word statement only once in a file.

  The software ignores redundant state-machine definitions.

You cannot have both MEALY_MACHINE and MOORE_MACHINE statements in the same design file. If you want to include both types of state machines in the design file, only one can be written using state-machine syntax. The other must be written using Boolean equations.

MINIMIZE_OFF MINIMIZE_ON

These two keywords allow you to specify equations that will not undergo logic reduction during the minimization process.

Devices Supported: All PLD devices.

SYNTAX

You use these keywords in the equations segment of Boolean designs.

Syntax

```
   MINIMIZE_OFF
   Boolean equations
   MINIMIZE_ON
```

Example

```
   MINIMIZE_OFF
   OUT1 = A * B :+: C
   MINIMIZE_ON
```

Definitions

All parameters are defined below.

Boolean Equation

Boolean equations control storage inputs and other combinatorial functions to produce the desired device behavior. These equations form the backbone of any PDS file containing a Boolean description.[72]

MINIMIZE_OFF

This keyword prevents logic reduction from occurring on the equation or equations that follow. You can suppress logic reduction only for an entire equation. **Do not** place this keyword in the middle of an equation.

MINIMIZE_ON

This keyword reactivates logic reduction on Boolean equations after it has been suppressed using the keyword MINIMIZE_OFF.

[72] Refer to BOOLEAN EQUATION and EXPRESSION, in this chapter, for additional details.

MINIMIZE_OFF, MINIMIZE_ON

USE

During the minimization process, all Boolean equations are reduced to their simplest form. However, sometimes the results of logic reduction may produce equations that are not functionally what you intend. You can selectively skip the logic reduction on certain equations by bracketing them within MINIMIZE_ON and MINIMIZE_OFF commands.

These commands do not affect other logic conversions that occur during the minimization process. The example below shows that parentheses are expanded for all expressions regardless of whether logic reduction is suppressed or not.

BEFORE MINIMIZATION	AFTER MINIMIZATION
MINIMIZE_OFF	MINIMIZE_OFF
01 = A*B + A*/B	01 = A*B + A*/B
02 = /(A+B)	01 = /A * /B
MINIMIZE_ON	MINIMIZE_ON
03 = A*B + A*/B	03 = A
04 = /(A+B)	04 = /A * /B

You can have as many pairs of MINIMIZE_ON and MINIMIZE_OFF commands as you wish. The software ignores redundant commands.

You need not place the MINIMIZE_OFF command on its own line, but it makes the design easier to follow.

MOORE_MACHINE

This reserved word identifies the type of state machine you are designing.

Devices Supported: All PLD devices.

SYNTAX

Include the reserved word in the state segment of state machine designs.

Syntax

```
    MOORE_MACHINE
```

Example

```
    STATE                                ;State Setup and Defaults
    MOORE_MACHINE
```

Definitions

Only the reserved word is defined below.

MOORE_MACHINE

This reserved word defines a state-machine design as one of the two possible types, either Moore or Mealy. If you do not define a type in the state segment of the design, the program defaults to Mealy machine.

A Moore machine determines its outputs from the present state only.[73]

USE

You can place the reserved word statement anywhere within the STATE segment. However, for design clarity, the following guidelines are advised.

- Place the reserved word statement at the beginning of the STATE segment before the state global defaults.

[73] Refer to MEALY_MACHINE and STATE, in this chapter, for additional details.

- Use this reserved word statement only once in a file.

  The software ignores redundant state-machine definitions.

You cannot have both MOORE_MACHINE and MEALY_MACHINE statements in the same design file. If you want to include both types of state machines in the design file, only one can be written using state-machine syntax. The other must be written using Boolean equations.

NODE

This keyword is the logical name assigned to a feedback signal or internal-control product term. When used in the declaration segment of a PDS file, this keyword allows you to assign names and attributes to internal device nodes. A node can be one of the following.

- A buried flip-flop or flip-flop feedback line
- An internal-control line, such as a global reset, preset, or observability line
- A complement array term

Devices Supported					
PAL22V10	PAL32VX10	PAL18U8	PAL23S8	PAL26V12	PALCE29M16
PALCE29MA16	PALCE610	PLS30S16	PLS105	PLS167	PLS168
MACH 1	MACH 2				

SYNTAX

Include this keyword in the declaration segment of Boolean and state-machine designs.

```
Syntax

    NODE      Location_number        Name          Storage

Example

    ...
    NODE           12                ST            REG
    ...
```

Definitions

Constructs following the keyword are defined below. Additional details are provided under Use.

Number

Number identifies the node number exactly as defined in the device reference.

Name

Name defines the node name. Each name must be unique and must follow the location_number field.

- Begin the name with an alpha character; use any combination of up to 14 upper- or lowercase alphanumeric characters: A–Z and 0–9.

> **Important**: Keep names in a schematic equal to or less than 14 characters. Part and node names in the schematic may be concatenated when data is converted into PDS format. Any name longer than 14 characters is automatically truncated.

- Use underscore as a connector and a forward slash to affect polarity; **no** other symbols or punctuation are allowed and no keywords, reserved words, or logic operators are allowed.

> **Note:** The forward slash is not supported for schematic-based designs.
>
> **Also:** Polarity works differently for nodes used as inputs on the PALCE29M16 and PALCE29MA16.[74]

You can use ranges and vector notation to define node names. You must use the same number of nodes as names in ranges and vector notation. All nodes defined within a range or vector notation have the same storage type and polarity attributes.[75]

Storage

Storage defines the optional storage type for a node, which must follow the node name.[76] Enter the reserved word or abbreviation listed below; the default is combinatorial.

[74] Refer to Chapter 11, in this section, for more information.

[75] Refer to VECTOR, in this chapter, for more information on vector notation and ranges.

[76] Refer to PAIR, in this chapter, for details about pairing a node with a pin.

• COMBINATORIAL	or	COMB
• REGISTERED	or	REG
• LATCHED	or	LAT

> **Important:** COMBINATORIAL is a valid node storage attribute only for MACH devices. For non-MACH PLDs, you must specify either REG or LAT. Otherwise, the software issues an error during compilation.

USE

NODE statements must follow the CHIP statement. Use a separate line for each NODE statement. You do not have to place the NODE statements in numerical order. You can only place COMMENT statements between NODE statements, not within the NODE statement.

Declare only the nodes you are using. The software automatically assigns the name NC, no connect, to all nodes that are not declared.

PIN or NODE statements in the current version of the software differ from the pin list of previous versions. However, the old syntax is fully compatible with the new.

Use of the NODE statement is device dependent.[77]

[77] Refer to Chapter 11, in this section, and the following topics, in this chapter, for additional details: BOOLEAN EQUATION, CHIP, DECLARATION SEGMENT, FLOATING PINS AND NODES, LATCHED, OPERATOR, PIN, and REGISTERED.

OPERATOR

This is a general term that describes any symbol interpreted by the software when processing design files. It can be a mathematical term such as "+", for OR, a defining term such as ":=", for registered, or a descriptive term such as "#", for radix. The following table defines each operator and provides an example.

OPERATOR	DEFINITION	EXAMPLE
/	NOT	/A
*	AND	A * B
+	OR	A + B
:+:	XOR	A :+: B
:*:	XNOR	A :*: B
=	COMBINATORIAL	INPUT1 = A * B
:=	REGISTERED equation	INPUT1 := A * B
:=	STATE EQUATION	STATE1 := START -> END
*=	LATCHED	INPUT1 *= A * B
->	STATE TRANSITION	STATE1 := START -> END
+->	LOCAL DEFAULT	+-> RED -> WAIT
;	COMMENT	;set low before clocking
,	Literal separator	IN[1,3,4] IN[1..4,6..9]
..	Range	INPUT[0..9]
:	CASE value	0,1:
[]	Term brackets	INPUT[0..9]
()	EXPRESSION	IN1 = (A * B) (C * D * F)

{}	Substitute	OUT1 = A * B * C
		OUT2 = {OUT1} * F>

>	Greater than	IF A > 2 THEN...
<	Less than	WHILE A < 2 DO...
<>	Not equal to	IF A <> 2 THEN...
<=	Less than or equal to	WHILE A <= 2 DO...
>=	Greater than or equal	WHILE A >= 2 DO...
%	Don't care	DEFAULT_OUTPUT %OUT1
?	CHECK clash	------??????
' '	String delimiters	STRING INPUT 'A1 + /A2'
#b	Binary radix	#b101000
#d	Decimal radix	#d40
#o	Octal radix	#o50
#h	Hexadecimal radix	#h28B
Space, tab	Separator	PIN 2 IN1 REG

.OUTF

This keyword defines state output equations for Mealy and Moore machines.

Devices Supported					
PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS1	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Include .OUTF in output equations in the state segment of state-machine designs.

```
Syntax

Moore machines     Statename.OUTF   =           Output expression

Mealy machines     Statename.OUTF   =           Condition 1 -> Output 1
                                                + Condition 2 -> Output 2
                                                ...
                                                + Condition n -> Output n
                                                +-> Local default
```

```
Example

                   ...
Moore machines     TWO.OUTF         =           /CNT2 * CNT1 * /CNT0

Mealy machines     TWO.OUTF         =           RUN_UP -> /CNT2 * CNT1 * /CNT0
                                                ...
                                                TEST -> CNT2 * CNT1 * CNT0
                                                +-> /CNT2 * /CNT1 * /CNT0
                   ...
```

Definitions

The construct immediately preceding, and all constructs following, the keyword are defined below. Additional details are provided under Use.

Statename

Statename identifies the name of the state as specified in the state assignments or state-transition equations. It must be unique and it can have up to 14 alphanumeric characters.

Outputs

Outputs are pin names with their appropriate logic sense to create the desired logic values. Outputs are separated by an asterisk, *. In the syntax example, when the Moore machine is in state TWO, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

When the Mealy machine is in state TWO and the inputs match the condition defined as RUN_UP, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

You specify the output values regardless of pin polarity. The software adjusts polarity as necessary.

Conditions

In a Mealy machine, the outputs depend on the current state and the current input conditions. This field defines the condition under which the specified output will occur. The condition names must be defined in the conditions section of the state-machine design.

If the condition consists of a single input, the input name may be used in place of the condition name. You can use VCC to specify an unconditional output.

Moore machine outputs do not have conditions since their outputs are determined only by the present state.

Local Default

This output is generated if none of the conditions is satisfied. Local defaults are valid only for Mealy machines and will override global defaults.

.OUTF

USE

You can place output equations anywhere within the state segment. You may prefer to have all of the output equations after all of the state equations, or follow a state equation with its corresponding output equation.

You can use the following operators in state output equations.

OPERATOR	DEFINITION
->	Conditional output for Mealy machines
+->	Local default for a Mealy machine
=	Combinatorial assignment operator
:=	Registered assignment operator

For Mealy machines, conditions in an .OUTF equation don't have to match conditions in the state-transition equations. However, conditions that don't match are unusual.

You can omit the output equations if you use the state bits as outputs. You do this by making the output pins the same as the state bits and preforming manual state bit assignment.[78]

If you omit output equations, don't use the following constructs.

- DEFAULT_OUTPUT
- OUTPUT_HOLD

You can define some outputs with state bits and some with output equations.

If you don't use the state bits as outputs, you must specify output equations. Default output specifications are optional.

[78] Refer to Section II, Chapter 4, for additional details regarding assigning state bits.

Registered Mealy machine outputs are valid one clock cycle after reaching the new state. Combinatorial Mealy and Moore machine outputs and registered Moore machine outputs are valid upon reaching the new state. Undefined output pins have a don't-care value.[79]

[79] Refer to the following topics, n this chapter, for additional details: CONDITIONS, DEFAULT_BRANCH, DEFAULT_OUTPUT, LOCAL DEFAULT, MEALY_MACHINE, MOORE_MACHINE, OPERATOR, OUTPUT_HOLD, STATE, STATE ASSIGNMENT EQUATION, STATE EQUATION, and STATE TRANSITION EQUATION.

OUTPUT_ENABLE

This reserved word selects the output-enable function of the preset/enable pin.

Devices Supported		
PLS105	PLS167	PLS168

SYNTAX

Use this reserved word in the state segment of state-machine designs.

Syntax

```
          OUTPUT_ENABLE
```

Example

```
   STATE
          ;State Setup and Defaults
          ...
          MEALY_MACHINE
          OUTPUT_ENABLE
          OUTPUT_HOLD
          ;State Bit Assignment
          ...
```

Definitions

Only this reserved word is defined below.

OUTPUT_ENABLE

OUTPUT_ENABLE sets the preset/enable pin to output enable. You can place OUTPUT_ENABLE anywhere in the setup and default section. It must precede the state assignment and transition equations.

USE

When you use OUTPUT_ENABLE, the software dedicates the preset/enable pin to active-low output enable. Conversely, when you use MASTER_RESET, the software dedicates the preset/enable pin to active-high preset. You cannot use both OUTPUT_ENABLE and MASTER_RESET in the same design file. Follow the guidelines outlined next.

- If you do not use MASTER_RESET or OUTPUT_ENABLE, the software selects preset as the default.

- If you write a .SETF equation and do not use MASTER_RESET or OUTPUT_ENABLE, the software selects preset as the default.

- If you write a .TRST equation, the software selects output enable as the default.[80]

[80] Refer to MASTER_RESET and STATE, in this chapter, for additional details.

OUTPUT_HOLD

This keyword defines a global default that allows you to hold the present output value when there is no output defined for the current state and input condition.

Devices Supported		
PLS105	PLS167	PLS168

SYNTAX

Use this keyword in the state segment of state-machine designs.

Syntax

```
            OUTPUT_HOLD            Output_pins
```

Example

```
   STATE
   ;State Setup and Defaults
            ...
            OUTPUT_HOLD            OUT1 OUT2 OUT3
            ...
            ;State Bit Assignment
            ...
```

Definitions

The parameter following the keyword is defined below. Additional details are discussed under Use.

Output_pins

Output_pins identifies the user-defined output pins that are held when the next output value cannot be determined from the equations segment of the state-machine design. Use a blank to separate the output pins; multiple blanks are reduced to one blank.

USE

You must place the OUTPUT_HOLD statement at the beginning of the state segment. It can follow setup statements but must precede any state assignment and transition equations.

If you use the state bits as outputs, do not use OUTPUT_HOLD.

You can reduce the number of output equations required by specifying the global default as OUTPUT_HOLD. Using this technique, you only need to write equations for outputs that differ from the default.

PAIR

This keyword is an optional attribute in a PIN or NODE statement you use to direct input or output pairing.

- **Input** pairing: include the PAIR attribute in a PIN statement to logically associate an input pin with a node.
- **Output** pairing: include the PAIR attribute in a NODE statement to logically associate a node with an output pin.

Devices Supported: MACH-device designs only.

Input pairing can only be implemented in MACH devices with buried macrocells. Output pairing can be implemented in all MACH devices.

SYNTAX

You can use the PAIR keyword in the declaration segment of Boolean or state-machine designs as shown below.

Required Elements			**Optional Attributes**		
Syntax					
Keyword	Location_number	Name	Storage	Pair	Pn_name
Example, Input Pairing					
Pin	?	I1	---	Pair	R1
Node	?	R1	Reg	---	---
Example, Output Pairing					
Node	?	L2	Combinatorial	Pair	Output1
Pin	?	Output1	Comb	---	---

Definitions

The following discussions pertain only to the descriptors for NODE and PIN statements.

Location

Location defines the location of the pin or node. When both the pin and node locations are fixed, you must assign both to the same macrocell.

Name

Name defines the name of the pin or node.

Note: An optional forward slash is supported here.

Storage

Storage defines the optional storage type for the pin or node; the default is combinatorial.

Important: Combinatorial is not a valid **node** storage attribute for input pairing. When specifying an **Input** pair, use the registered or latched attribute in the **NODE** statement.

Pair

Include this optional keyword to indicate input pairing in a PIN statement or output pairing in a NODE statement. PAIR cannot be abbreviated. The keywords OPAIR and IPAIR are also valid, and denote output and input pairing, respectively.

- Output pairs are generated when there are duplicate pin/node equations.
- Input pairs are generated when a buried input node is equated to an input pin.

Pairing occurs automatically during compilation unless you enable manual pairing by typing the letter N beside the Use automatic pin/node pairing field on the Logic Synthesis Options form.

Recommendation: It is best to enable the automatic pin/node pairing option on the Logic Synthesis Options form.

Pn_name

Pn_name defines the pin or node that completes the pair. Each name must be unique and follow the keyword PAIR.

PAIR

Note: A node and its corresponding output pin should not be paired if the three-state control line is tied to ground. This permanently disables the output pin.

Also: No forward slash is allowed in the pin or node name following the keyword PAIR.

USE

When paired, the pin and node are logically associated with the same macrocell. Input pairing applies only to registered or latched inputs.

PATTERN

This keyword begins the statement defining the design's pattern, which is useful for documentation purposes.

Devices Supported: All PLD devices.

SYNTAX

Use the PATTERN keyword in the declaration segment of Boolean and state-machine designs.

Syntax

```
   PATTERN                       Design_pattern
```

Example

```
   TITLE
   PATTERN                       F00345
   REVISION
   AUTHOR
   COMPANY
   DATE
   CHIP  COUNTER  PAL16R8
```

Definitions

Only the construct following the keyword PATTERN is defined below.

Design_pattern

Design_pattern can be any combination of up to 60 alphanumeric characters. The following rules apply.

- Place the PATTERN statement after the title and before revision, as shown in the syntax example. The software assumes this order.
- Write the PATTERN statement on one line.
- Do not use a dollar sign.

  You can use reserved words within the PATTERN statement.

USE

The PATTERN statement is optional. If you do not include it, the software issues a warning and continues processing the file. If you include multiple PATTERN statements, the software issues an error and stops processing the file.[81]

[81] Refer to the following topics, in this chapter, for additional details: AUTHOR, COMPANY, DATE, DECLARATION SEGMENT, REVISION, and TITLE

PIN

This keyword begins a statement that allows you to assign names and attributes to device pins.

Devices Supported: All PLD devices.

SYNTAX

Include the PIN keyword in the declaration segment of Boolean and state-machine designs.

Syntax

```
            PIN            Location_number      Name   Storage
```

Example

```
            ...
            PIN            1                    IN1    COMB
            PIN            8                    IN2    REG
            PIN            16                   OUT
            ...
```

Definitions

Constructs following the keyword PIN are defined below. Additional details are provided under use.

Location_number

Location_number identifies the pin number, as defined in the device datasheet or Chapter 11. For MACH-device designs you can place a question mark in this field to define a floating pin.[82]

Name

Name defines the name of the pin . Each name must be unique and must follow the location_number field.

- Begin the name with an alpha character; use any combination of up to 14 upper- or lowercase alphanumeric characters: A–Z and 0–9.

[82] Refer to FLOATING PINS AND NODES, in this chapter, for additional details.

Important: Keep names in a schematic less than or equal to 14 characters. Part and pin names in the schematic may be concatenated when data is converted into PDS format. Any name longer than 14 characters is automatically truncated.

- Use the underscore as a connector and a forward slash to affect polarity; **no** other symbols or punctuation are allowed and no keywords, reserved words, or logic operators are allowed.

Note: The forward slash is not supported for schematic-based designs.

You can use ranges and vector notation to define pin names. You must use the same number of pins as names in ranges and vectors. All pins defined within a range or vector notation have the same storage type and polarity attributes.

Storage

Storage defines the pin storage type.[83] You can specify one of the following three storage types.

- COMBINATORIAL or COMB
- REGISTERED or REG
- LATCHED or LAT

Note: You need only enter the first three or four letters of the storage attribute.

If you do not select a pin type, the software defaults to combinatorial, even if the device you selected in the CHIP statement is fully registered. This helps portability of designs across all devices.

[83] Refer to Chapter 11, in this section, to determine the correct storage type for the device.

PIN

USE

PIN statements must follow the CHIP statement. Use a separate line for each PIN statement. You do not have to place the PIN statements in numerical order. You can only place comments between PIN statements, not within a PIN statement. Separate each pin attribute by one or more blanks.

Declare only the pins you are using. The software automatically assigns the name NC, no connect, to all pins that are not declared.

You must declare the VCC and GND pins.[84] You cannot use different names for VCC and GND.

Pin statements in the current version of the software differ from the pin list of previous versions. However, the old syntax is fully compatible with the new syntax.[85]

[84] Refer to the individual device datasheet or Chapter 11, in this section, for the correct VCC and GND pin numbers.

[85] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, CHIP, COMBINATORIAL, DECLARATION SEGMENT, LATCHED, NODE, OPERATOR, PAIR, REGISTERED, and VECTOR.

PRELOAD

This simulation keyword loads specified values into the register outputs. Even if a device does not physically support the preload feature, you can simulate the design as though it does, but JEDEC test-vector generation is turned off.

> **Devices Supported**: All registered PLD devices, except PAL22IP6, PAL23S8, PAL16RA10, and PAL20RA10.

SYNTAX

Use this keyword in the simulation segment of Boolean and state-machine designs.

Syntax

```
            PRELOAD Prefix_pns
```

Example

```
            SIMULATION
            PRELOAD Q0 /Q1 PLAYING
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

Prefix

The prefix specifies the logic state of the corresponding pin, node, or state. Do not leave a blank between Prefix and pns. There are two prefixes: null and forward slash.

- You specify the null prefix to load a logical 1 into the associated register output. In the syntax example, Q0 has a null prefix.

  When used in conjunction with a state name, the null prefix loads the specified state. In the syntax example, PLAYING has a null prefix.

- You specify the forward slash to load a logical 0 into the associated register output. In the syntax example, Q0 has a forward-slash prefix.

Pns

You specify the pin, node, or state to be preloaded immediately following the corresponding prefix.

You can list more than one pin or node. You can also use groups and strings.

USE

PRELOAD loads specified logic values into the corresponding device registers. Therefore, if the signal is inverted between the node and pin, the value at the pin will be the inverse of the preloaded value.[86]

Some devices provide a hardware preload feature that is activated by a dedicated pin or product term. Use the SETF command for control of these preload features.

> **Note:** Even if a device does not physically support the preload feature, you can simulate the design as though it does.

PRELOAD places a "P" in the clock field(s) of the JEDEC vector for PAL and PLS devices, as specified in the JEDEC 3A standard. For MACH devices, the buried preload vector "B" is used as defined by the JEDEC 3B standard.

[86] Refer to the specific device datasheet to determine the correct preload value for the particular flip-flop.

.PRLD

This reserved word defines the conditions in a functional equation when the preload function is activated in devices with logic-controlled preload. When the .PRLD equation is true, you can preload flip-flops with the desired value from the corresponding I/O pin.

Devices Supported	
PALCE29M16	PALCE29MA16

SYNTAX

Use the .PRLD functional equation in the equations segment of Boolean designs.

```
Syntax

            Node.PRLD    Assignment Operator     Expression

Example

   EQUATIONS
            Q0                  =               /Q0
            GLO.PRLD            =               Q0 * /Q1
            ...
```

Definitions

All parameters are defined below.

Node.PRLD

Node.PRLD identifies the node associated with the register you want to preload. On the PALCE29M16, this is the global node.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[87]

Expression

Expression identifies logic you assign to define the conditions when the preload function is activated.

[87] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .PRLD equation anywhere in the equations segment. The .PRLD equation may fit on more than one product term, depending on the device. The following rules apply.

- Do not use multiple functional equations for the same pin or node. Otherwise, an error is reported during compilation and the processing stops.
- Do not use negative polarity on the left side of the equation. For example, /Q1.PRLD is illegal.

  Logic preloads use SETF and the pin polarity defined in the pin declaration segment to determine preload polarity.

  > **Note:** Super voltage preloads evaluate a complemented pin, /A, as preload low. If there is no complement, then it is a preload high.

- Disable any product-term enables when .PRLD is low.[88]

In simulation, the SETF command asserts the product term defined by .PRLD to determine the preload values.

In the simulation history file, .PRLD is represented by the letter P, SETF by the letter H, and .RSTF by the letter L.[89]

[88] Refer to the specific device datasheet for complete information.

[89] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, PRELOAD, and SETF.

PRLDF

PRLDF assigns a value to a register output to force the specified value at the pin. Use the PRLDF keyword in the simulation segment of Boolean and state-machine designs.

Devices Supported			
PAL22IP6	PAL23S8	PAL16RA10	PAL20RA10

SYNTAX

Use this keyword in the simulation segment of Boolean and state-machine designs.

Syntax

```
   PRLDF Prefix_rns
```

Example

```
   SIMULATION
   PRLDF 01 /02
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

Prefix

The prefix specifies the logic state of the corresponding register, node, or state. Do not leave a blank between Prefix and rns. There are two prefixes: null and forward slash.

- The null prefix indicates the pin value should be a logical 1 if the polarity is not inverted in the pin declaration of the design. In the syntax example, 01 has a null prefix.

- The forward slash indicates that the pin or node should be a logical 0 if the polarity is not inverted in the pin declaration of the design. In the syntax example, 02 has a forward-slash prefix.

Rns

You specify the value to be preloaded immediately following the corresponding prefix.

You can list more than one pin or node. You can also use groups and strings.

USE

PRLDF loads a value into a register so that specified logic values appear at the pin. If an inverter exists between the register output and the pin, PRLDF compensates for the inversion by inverting the register contents.[90]

Some devices provide a hardware preload feature that is activated by a dedicated pin or product term. Use the SETF command to control preload.

For devices with a preload pin, PRLDF disables the outputs, enables preload, loads the registers with the values, disables preload, and then enables the outputs.

Note: Even if a device does not physically support the preload feature, you can simulate the design as though it does.[91]

PRLDF places a P in the clock field of the JEDEC vector for devices with supervoltage preloads.

[90] Refer to the individual device datasheet to determine the correct preload value for the particular flip-flop.

[91] Refer to the following topics, in this chapter, for additional details: CHECK, .PRLD, PRLDF, and SETF.

.R EQUATION

This equation defines when to set the R input high on S-R flip-flops.

Devices Supported					
PLS105	PLS167	PLS168	PLS30S16	PALCE610	PAL22IP6

SYNTAX

Use the .R equation in the equations segment of design files with Boolean or state-machine designs.

Syntax

```
            Pn.R        Assignment Operator      Expression
```

Example

```
   EQUATIONS
            ...
            Q1.R                 =               IN1 * /IN2
            ...
```

Definitions

All parameters are defined below.

Pn.R

Pn.R identifies the pin or node associated with the S-R flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[92]

Expression

Expression identifies the logic you define to determine when the R input in an S-R flip-flop is set high. In the example, when IN1 is true and IN2 is false, the R input in the S-R flip-flop associated with the pin or node Q1 is set high.

[92] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .R equation anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin. If you do, the software reports an error during compilation and processing stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.R is not allowed.
- You can use the group, string, and vector notation to define signals. This is an excellent way to assign a. R equation to several pins.[93]

[93] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, and .S EQUATION.

REGISTERED

This reserved word defines the output data-storage type on devices with registered outputs. A registered output stores its value regardless of any data changes. New data is placed in the register by an edge-sensitive clock signal. A register may consist of D, T, S-R, and 2-T flip-flops.

Devices Supported					
PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RA8	PAL16RP4
PAL16RP6	PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6
PAL20R8	PAL20RA10	PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8
PAL20X4	PAL20X8	PAL20X10	PAL22IP6	PAL22RX8	PAL22V10
PAL23S8	PAL24R10	PAL24R4	PAL24R8	PAL26V12	PALCE29M16
PALCE29MA16	PAL32R16	PAL32VX10	PAL64R32	PALCE610	PLS105
PLS167	PLS168	PLS30S16	MACH 1	MACH 2	

SYNTAX

You include the optional reserved word in the PIN or NODE statement of Boolean and state-machine designs.

Syntax

```
   PIN or
   NODE        Number            Location_name              Storage
```

Example

```
   ...
   CHIP COUNTER PAL16R8
   PIN         15                  OUT1                     REGISTERED
   PIN         16                  OUT2                     REG
   ...
```

Definitions

Only the reserved word is discussed below.

Storage

The storage value defines the pin or node storage type. If you do not specify a storage type, combinatorial is the default.

- Place the reserved word REGISTERED after the pin or node name in the corresponding statement.
- Use either the complete word REGISTERED or the three-letter abbreviation, REG.

USE

There are two ways to specify the storage type.

- Use the declaration segment form: you select the storage field, display the option list, and select an option.
- Type the storage value in the appropriate PIN or NODE statement in the PDS file using a text editor.

PALASM requires a data-storage type for each I/O or output pin: COMBINATORIAL, LATCHED, or REGISTERED. The software requires the registered pin or node type even if the pin or node can only be registered.[94]

[94] Refer to the following topics, in this chapter, for additional details: COMBINATORIAL, DECLARATION SEGMENT, LATCHED, NODE, and PIN.

REVISION

This keyword begins the statement that defines the revision of the current design. The design revision is useful for documentation purposes.

Devices Supported: All PLD devices.

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

```
REVISION          Design_revision
```

Example

```
TITLE
PATTERN
REVISION            2.2B
AUTHOR
COMPANY
DATE
CHIP
```

Definitions

Only the descriptor following the keyword REVISION is discussed.

Design_revision

Design_revision is optional and may be any combination of up to 59 alphanumeric characters that designates the current version of the design.

- You can use other symbols or punctuation; however, you cannot use the dollar sign.

- You can use reserved words in this statement.

USE

The following error conditions pertain to the REVISION statement.

- Without the REVISION statement, the software issues a warning and continues processing the file.
- With multiple REVISION statements, the software issues an error and stops processing the file.[95]

[95] Refer to the following topics, in this chapter, for additional details: AUTHOR, COMPANY, DATE, PATTERN, and TITLE.

.RSTF

This reserved word defines when to assert a reset signal high on devices having flip-flops or registers with a reset input.

Devices Supported					
PAL10H20EG8	PAL10H20EV8	PAL16RA8	PAL20RA10	PAL22IP6	PAL22RX8
PAL22V10	PAL23S8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state-machine designs.

```
Syntax

                Pn.RSTF        Assignment Operator     Expression
Example

   EQUATIONS
                QO                  =                  /Q1 */Q2
                QO.RSTF             =                  RST * /SET
                ...
```

Definitions

All parameters are defined below.

Assignment Operator

The assignment operator is a symbol that defines a specific operation, as interpreted by the software when processing design files.[96]

Pn.RSTF

Pn.RSTF identifies the pin or node associated with the flip-flop having a reset input.

Expression

Expression defines the logic conditions that determine when to assert the reset signal high. In the previous example, the reset pulse is initiated when RST is true and SET is false.

[96] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

Multiple .RSTF statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting. You can specify a global reset on devices with global nodes, for example, global node.RSTF.

You cannot use negative polarity on the left side of an equation. For example, /Q1.RSTF is not allowed.

You can use the GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a functional equation to several pins and nodes.

Depending on the device, the reset line is synchronous or asynchronous. On the PAL16RA8, if both preset and reset of a flip-flop are high, the flip-flop is bypassed.

In the simulation history file, .RSTF is represented by the letter L, SETF by the letter H, and .PRLD by the letter P.[97]

[97] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, PRELOAD, and SETF.

.S EQUATION

This equation defines when to set the S input on S-R flip-flops high.

Devices Supported					
PLS105	PLS167	PLS168	PLS3016S	PALCE610	PAL22IP6

SYNTAX

Use the .S equation in the equations segment of Boolean or state-machine designs.

Syntax

```
            Pn.S        Assignment Operator       Expression
```

Example

```
   EQUATIONS
            ...
            Q1.S                  =               IN1 * /IN2
            ...
```

Definitions

All parameters are defined below.

Pn.S

Pn.S is the pin or node associated with the S-R flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[98]

Expression

Expression identifies the logic you define to determine when the S input in an S-R flip-flop is set high. In the syntax example, when IN1 is true and IN2 is false, the S input in the S-R flip-flop associated with the pin or node Q1 is set high.

[98] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .S EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.S is not allowed.
- You can use GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .S equation to several pins.[99]

[99] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

.SETF

This reserved word defines when to assert a preset signal high on devices having flip-flops or registers with a preset input.

Devices Supported					
PAL10H20EG8	PAL10H20EV8	PAL16RA8	PAL20RA10	PAL22IP6	PAL22RX8
PAL22V10	PAL23S8	PAL26V12	PALCE29M16	PALCE29MA16	
PAL32VX10	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state-machine designs.

Syntax

```
            Pn.SETF      Assignment Operator     Expression
```

Example

```
   EQUATIONS
            Q0                      =            /Q1 */Q2
            Q0.SETF                 =            /RST * SET
            ...
```

Definitions

All parameters are defined below.

Pn.SETF

Pn.SETF identifies the pin or node associated with the flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[100]

Expression

Expression defines the logic conditions that determine when to assert the preset signal high on flip-flops or registers that support preset inputs.

[100] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

Multiple .SETF statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting. You can specify a global preset on devices with global nodes, for example, global node.SETF.

You cannot use negative polarity on the left side of the equation. For example, /Q1.SETF is not allowed.

You can use the GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a functional equation to several pins and nodes.

Depending on the device, the preset line is synchronous or asynchronous.[101]

If you define the outputs as COMBINATORIAL, the default value for preset is VCC, unconditional high. If you define the outputs as registered, the default value is GND, unconditional low.

In the simulation history file, .SETF is represented by the letter H, RSTF by the letter L, and .PRLD by the letter P.[102]

[101] Refer to the Chapter 11, in this section, for details on using .SETF with a specific device.

[102] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, GROUP, .RSTF, STRING, and VECTOR.

SETF

SETF assigns values to specific inputs during simulation.

Devices Supported: All PLD devices.

SYNTAX

Include the SETF command in the simulation segment or auxiliary simulation file for Boolean and state-machine designs.

Syntax

```
   SETF Prefix_pn
```

Example

```
   SIMULATION
   SETF IN1 /OE
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

If the signal being set is defined with the same polarity as in the PIN or NODE declaration segment, the signal is set to a logical 1. If the polarity is reversed, the signal is set to a logical 0.

Note: The following examples are valid only when the signals are defined as active-high in the PIN or NODE declaration segment.

Prefix

The prefix specifies the logic state of the associated input pin or node. There are two prefixes: null and forward slash.

- The null prefix sets the corresponding input to a logical 1. In the syntax example, IN1 has a null prefix.

- The forward slash sets the corresponding input to a logical 0. In the syntax example, OE has a forward-slash prefix.

Pn

Pn is the input pin or node to be set.

USE

You can list more than one pin or node. Separate multiple pins or nodes with a blank. You can also use groups and strings.

If the signal being set is defined with the same polarity as in the PIN or NODE declaration segment, the signal is set to a logical 1. If the polarity is reversed, the signal is set to a logical 0.

You cannot use the SETF command to set states. However, you can use the SETF command to set input values.

The software shows each occurrence of SETF by placing the letter g in the header of the waveform and text simulation files.

SIGNATURE

This keyword allows you to program user-defined data into devices having a SIGNATURE word function. This data can be used for such purposes as user identification, revision control, or inventory control.

Devices Supported	
PALCE16V8	PALCE20V8

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

```
             SIGNATURE    Assignment Operator     Number or String
```

Example

```
   DECLARATION
             ...
             PIN  18  /OUT8  REG
             SIGNATURE              =                   V2_5/89
             ...
```

Definitions

Only the parameters Number and String are discussed below.

Number or String

Number or string is either a base (radix) number or an alphanumeric character string.

SIGNATURE supports four number radices; the default is decimal.

• Binary	#B or #b
• Decimal	#D or #d
• Hexadecimal	#H or #h
• Octal	#O or #o

The software allows a maximum of 64 bits for the radix number. This translates to the following list of maximum digits allowed for each radix.

RADIX	MAXIMUM NUMBER OF DIGITS
Binary	64
Hex	16
Octal	21
Decimal	15

If you exceed the maximum number of digits allowed for a radix, the software issues a warning and truncates the extra most significant bits.

If a number contains a blank, non-number, a decimal number, or any other alphanumeric character except the radix operator, the software treats the entire character string as alphanumeric.

In using alphanumeric characters, observe the following guidelines.

- You can use any combination of alphanumeric characters up to a maximum of eight characters.

  If the number exceeds eight, the software issues a warning and truncates the extra characters to the right.

- You can use underscores and blanks.

The software converts alphanumeric characters to ASCII and all lowercase characters to uppercase.

USE

Place the SIGNATURE statement after the PIN or NODE statements. Observe the following rules.

- You can place the SIGNATURE statement in any order with the GROUP or STRING statements.
- You can use only one SIGNATURE statement for each device.

- You must use the assignment operator, =, in the statement.

  If you have multiple SIGNATURE statements, the software issues a warning and programs the last SIGNATURE statement.

You can access signature information even if the security fuse has been programmed.[103]

[103] Refer to DECLARATION SEGMENT and OPERATOR, in this chapter, for additional details.

SIMULATION

Use the SIMULATION keyword to start the simulation segment .

Devices Supported: All PLD devices.

SYNTAX

Syntax

```
    SIMULATION
```

Example

```
    SIMULATION
```

Definitions

No parameters are required with this keyword.

USE

Use the SIMULATION keyword to start the simulation segment or auxiliary simulation file of Boolean and state-machine designs.

START_UP

This keyword allows you to power up in a specific state or asynchronously branch to a state whenever an initialization condition occurs.

Devices Supported: All devices that support state-machine descriptions.

SYNTAX

The START_UP keyword is used in Moore machines. START_UP.OUTF is used in registered Mealy machines.

Syntax

```
          START_UP      :=      POWER_UP -> State1
                                + Condition1 -> State2

          START_UP.OUTF :=      POWER_UP -> Outputs
                                + Condition1 -> Outputs
```

Example

```
   STATE
          ;State Setup and Defaults
          ...
Moore machines
          START_UP                := POWER_UP -> S1
                                  + INIT -> S1
                                  ...
          ;Powers up and initializes to S1

Mealy machines
          START_UP.OUTF           :=            O1 * O2
                                  + INIT  -> O1*O2

          ;State Assignments
          S1 = /STATE BIT1 * /STATE BIT2
          ;S1 value is 00
          ...
          ;State Equations
          S1 = FC -> S3
             + FCC -> S7
          ...
```

Definitions

Parameters following the keyword are defined below. Additional details are discussed under Use.

State1

When power is applied to the device, the device goes to state1.

- In devices that initialize with all flip-flops high or all flip-flops low, the START_UP command assigns the appropriate all-high or all-low state-bit code to the specified state.
- In devices with programmable power up, the START_UP command programs the device to power up in the specified state. If you specify a particular state-bit code using the manual state-bit assignment syntax, the software programs the flip-flops to initialize with the specified values.

Condition1

Condition1 is a user-defined condition that specifies when an initialization occurs. When the condition is true, the device is initialized asynchronously to state2.

A condition must be defined in the condition section of the state-machine design. If the condition consists of a single input, the input name can be used in place of the condition name.

State2

This state occurs as a result of the initialization condition. This state may differ from the power-up state.

USE

Use initialization routines to ensure the state machine powers up in a known state or branches to a known state whenever initialization occurs.

If you do not include a START_UP statement, the device will power up in the state that appears in the first transition equation in the PDS file.

The first line of the example contains the power-up state. When power is applied to the device, it goes to this state. It may be helpful to think of power up as a "cold boot."

The second line of the START_UP statement defines the initialization state. Only devices with programmable initialization support this function of the START_UP statement.[104]

STARTUP.OUTF allows you to define the outputs at power-up and initialization. This is especially useful for synchronous Mealy machines where outputs are delayed by one cycle from their respective states.[105]

You cannot use a default branch in the START_UP statement.

[104] Refer to Chapter 11, in this section, for information on specific devices.

[105] Refer to .OUTF and STATE EQUATIONS, in this chapter, for additional details.

STATE

Use the STATE keyword to identify the state segment of state-machine designs. The state segment contains setups, defaults, and state equations.

Devices Supported					
PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R10
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Use the state keyword in the PDS file after the declaration segment. If your design contains a mix of state-machine and Boolean equations, it can appear before or after the Boolean segment.

Syntax

```
STATE

State Setup and Defaults
State Equations
          Transition Equations
          Output Equations
          State Assignment Equations
          Condition Equations
```

Example

```
   STATE
            ;State Setup and Defaults
            MOORE MACHINE
            START_UP := POWER_UP -> S1
            DEFAULT_BRANCH HOLD_STATE
            ;State  Transition Equations
            S1 := COND1-> S3
                 + COND2 -> S7
            ...
            ;State  Output Equations
            S1.OUTF = OUT1*/OUT2
            ...
            ;State Assignment Equations
            S1 = /STATE BIT1 * /STATE BIT2
            ...
            ;State Condition Equations
            CONDITIONS
            COND1 = IN1 * /IN2 * IN3
            ...
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

Setup and Defaults

Statements at the beginning of the state segment identify the state machine. CLKF, MASTER_RESET, MEALY_MACHINE, MOORE_MACHINE, START_UP, and START_UP.OUTF appear in this section.

STATE

State-machine designs can have global and local defaults. Global defaults are defined by DEFAULT_BRANCH, DEFAULT_OUTPUT, and OUTPUT_HOLD. Local defaults are defined in the state equations with the local default operator, +->.

State Equations

There are four types of state-machine equations. They have the following functions.

- Transition equations (required)

  For each state, these conditions specify what the next state will be under various conditions. See Condition equations below.

- Output equations (optional)

  These equations specify the outputs of the state machine. No output equations are required in cases where the state bits themselves are the outputs.

- State assignment equations (optional)

  These equations specify the bit code to be assigned to each state name used in the design. If these equations are omitted, the software assigns the bit codes automatically.

- Condition equations (normally required)

  These equations specify a condition name for each set of input values used to determine a transition. You can use input names directly only if a single input controls the transition; otherwise, you must use condition names.

USE

The state segment follows the declaration segment of the PDS file. If the design has an equations segment, the state segment can precede or follow it.

> **Important:** The state segment typically replaces the equations segment. It is possible to modify state equations with Boolean equations by including both equation and state segments, in any order. In this case, you must select the Merge Mixed Mode option from the Compile Setup menu.

The STATE segment supports the following reserved words.

- CLKF
- CONDITIONS
- DEFAULT_BRANCH
- DEFAULT_OUTPUT
- HOLD_STATE
- MASTER_RESET
- MEALY_MACHINE
- MOORE_MACHINE
- NEXT_STATE,
- .OUTF
- OUTPUT_ENABLE
- OUTPUT_HOLD
- START_UP
- STATE

STATE ASSIGNMENT EQUATION

State assignment equations define the unique bit codes to be assigned to each state name used in the design. The bit codes are composed of state bits that are stored in flip-flops.

Devices Supported					
PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Use state assignment equations in the state segment of state-machine designs.

Syntax

```
          State name    Assignment Operator    State Bits
```

Example

```
  STATE

          ;State Assignments
          S1                      =             /STATE BIT1 * /STATE BIT2
          S2                      =             /STATE BIT1 * STATE BIT2
          S3                      = S           STATE BIT1 * /STATE BIT2
          S4                      =             STATE BIT1 * STATE BIT2
          ...
          ;State Equations
          S1 := FC -> S3
             + FCC -> S7
          ...
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

State Name

The user-defined state name must be unique and can have up to 14 alphanumeric characters. It cannot contain operators or reserved words.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[106]

Statebits

The state bit name composes the bit code. The state bit name is register name, as defined in the PIN or NODE statements. Use an asterisk, *, to separate state bits. Use polarity notation to indicate the value of each state bit.

USE

State assignments follow state defaults and precede condition equations. Use the assignment operator, =, to define state assignments.

Each state assignment must include the complete set of state bits.[107] If you use three state bits for eight states, each assignment must include all three bits.

The example uses two state bits, State Bit1 and State Bit2. When both are low, the device is in state S1. When State Bit1 is low and State Bit2 is high, the device is in state S2. When State Bit1 is high and State Bit2 is low, the device is in state S3; when both are high, the device is in state S4.

For large designs in which you use many state bits (six or more), you may not want to list all possible state bit combinations. However, not defining all possible state bit combinations leaves some undefined or illegal states.

[106] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

[107] Refer to Section II, Chapter 4, for information on choosing state bit assignments.

STATE ASSIGNMENT EQUATION

The present state is defined by the contents of the state register, which consists of *n* bits capable of defining 2^n possible states. Before you can determine the present state, you must know the contents of all *n* bits. For example, three-state register bits define up to 2^3, or eight, possible states.

To define a state machine with 2^n states, you need a device with *n* registered outputs or buried registers (nodes) to use as state register bits. For example, the PAL16R8 has eight registered outputs and accommodates up to 2^8, or 256, states, provided you do not use any of the registers for other purposes such as independent outputs.[108]

If you do not assign state bits, the software assigns them automatically. The software assigns state bits to outputs that aren't defined by PIN or NODE statements. Look at the Execution log file to determine the automatic state bit assignments made by the software.[109]

[108] Refer to STATE and STATE EQUATIONS, in this chapter, for additional details.

[109] Refer to Chapter 9, in this section, for more information on execution log file.

STATE EQUATIONS

STATE EQUATIONS control the transitions, outputs, state assignment and conditions of state machines.

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PAL64R32	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

Use these equations in the state segment of the design to define the behavior of the state machine.

Syntax

```
   State Equations
            Transition Equations
            Output Equations
            State Assignment Equations
            Condition Equations
```

Example

```
   STATE
            ;State Setup and Defaults
            MOORE MACHINE
            START_UP := POWER_UP -> S1
            DEFAULT_BRANCH HOLD_STATE
            ;State  Transition Equations
            S1 := COND1-> S3
                 + COND2 -> S7
            ...
            ;State  Output Equations
            S1.OUTF = OUT1*/OUT2
            ...
            ;State Assignment Equations
            S1 = /STATE BIT1 * /STATE BIT2
            ...
            ;State Condition Equations
            CONDITIONS
            COND1 = IN1 * /IN2 * IN3
            ...
```

Definitions

Parameters following the keywords STATE EQUATIONS are defined below. Additional details are discussed under Use.

Transition Equations

These equations specify, for each state, what the next state will be under various conditions.

Output Equations

Output equations specify the output of the state machine. For a Moore machine, they define the outputs for a given state. For a Mealy machine, they define the outputs for a given state and input condition.

State Assignment Equations

These assignments define states as unique combinations of register bits.

STATE EQUATIONS

Condition Equations

Condition equations specify a condition name for each set of input values used to determine a transition. You can use input names directly only if a single input controls the transition; otherwise, you must use condition names.

USE

The order of state equations is not important except for condition equations which have their own section under the CONDITIONS keyword. The conditions segment must terminate the state segment.

Setups and defaults are not equations and must appear at the beginning of the state segment.[110]

[110] Refer to the following topics, in this chapter, for additional details: CONDITIONS, .OUTF, STATE, STATE ASSIGNMENT EQUATION, and STATE TRANSITION EQUATION.

STATE OUTPUT EQUATION

STATE OUTPUT EQUATIONS control state-machine outputs.[111]

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS1	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Include output equations in the state segment of state-machine designs after setup and defaults and before the condition equations.

```
Syntax

Moore machines   Statename.OUTF  =          Output expression

Mealy machines   Statename.OUTF  =          Condition 1 -> Output 1
                                            + Condition 2 -> Output 2
                                            ...
                                            + Condition n -> Output n
                                            +-> Local default
```

```
Example

                  ...
Moore machines   TWO.OUTF        =          /CNT2 * CNT1 * /CNTO

Mealy machines   TWO.OUTF        =          RUN_UP -> /CNT2 * CNT1 * /CNTO
                                            ...
                                            TEST -> CNT2 * CNT1 * CNTO
                                            +-> /CNT2 * /CNT1 * /CNTO
                  ...
```

Definitions

The construct immediately preceding, and all constructs following, the keyword are defined below. Additional details are provided under Use.

[111] Refer to .OUTF, in this chapter, for a description of state output equations.

STATE OUTPUT EQUATION

Statename

Statename identifies the name of the state as specified in the state assignments or state transition equations. It must be unique and can have up to 14 alphanumeric characters.

Outputs

Outputs are pin names with appropriate logic sense to create the desired logic values. Outputs are separated by an asterisk, *. In the syntax example, when the Moore machine is in state TWO, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

When the Mealy machine is in state TWO and the inputs match the condition defined as RUN_UP, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

You specify the output values regardless of pin polarity. The software adjusts polarity as necessary.

Conditions

In a Mealy machine, the outputs depend on the current state and the current input conditions. This entry specifies the condition under which the specified output will occur. The condition names must be defined in the conditions section of the state-machine design.

If the condition consists of a single input, the input name may be used in place of the condition name. You can use VCC to specify an unconditional output.

Moore machine outputs do not have conditions since their outputs are determined only by the present state.

Local Default

This output is generated if none of the conditions is satisfied. Local defaults are valid only for Mealy machines. Local defaults override global defaults.

STATE OUTPUT EQUATION

USE

You can place output equations anywhere within the state segment. You may prefer to have all of the output equations after all of the state equations or following each state equation with its corresponding output equation.

You can use the following operators in STATE OUTPUT EQUATIONS.

OPERATOR	DEFINITION
->	Conditional output for Mealy machines
+->	Local default for a Mealy machine
=	Combinatorial assignment operator
:=	Registered assignment operator

For Mealy machines, conditions in .OUTF don't have to match conditions in the state-transition equations. Typically, however, these conditions match.

You can use the state bits as outputs by making the output pins the same as the state bits and performing manual state bit assignment.[112] In this case, you can omit the output equations. If you do this, don't use the following constructs.

- DEFAULT_OUTPUT
- OUTPUT_HOLD

You can define some outputs with state bits and some outputs with output equations.

If you don't use the state bits as outputs, you must specify output equations. Default output specifications are optional.

[112] Refer to Section II, Chapter 4, for additional details regarding assigning state bits.

STATE OUTPUT EQUATION

Registered Mealy machine outputs are valid one clock cycle after reaching the new state. Combinatorial Mealy and Moore machine outputs and registered Moore machine outputs are valid on reaching the new state. Undefined output pins have a don't-care value.[113]

[113] Refer to the following topics, in this chapter, for additional details: CONDITIONS, DEFAULT_BRANCH, DEFAULT_OUTPUT, LOCAL DEFAULT, MEALY_MACHINE, MOORE_MACHINE, OPERATOR, OUTPUT_HOLD, STATE, STATE ASSIGNMENT EQUATION, STATE EQUATIONS, and STATE TRANSITION EQUATION.

STATE TRANSITION EQUATION

For each state, transition equations specify what the next state will be under various conditions.

Devices Supported					
PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PAL64R32	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

Include state-transition equations in the state segment of state-machine designs.

Syntax

```
   STATE
              ...
              Present state          :=            Condition1 -> State1
                                                   + Condition2 -> State2
                                                   +-> Local default
```

Example

```
   STATE
              ...
              ;State Equations
              ...
              TWO                    :=            COUNT_UP -> THREE
                                                   + COUNT_DWN -> ONE
                                                   +-> TWO
              ...
```

Definitions

Parameters following the keyword STATE are defined below. Additional details are discussed under Use.

Present State

The present state name is defined in the state diagram.

- Use any combination of up to 14 upper- or lowercase alphanumeric characters, A-Z and 0-9.

- Do not use keywords, reserved words, or logic operators.

Condition

Condition identifies condition names as defined in the conditions section of the state-machine design.

State

State identifies the next state, as defined in the state diagram.

Local Default

Local default defines the state the machine will go to if none of the specified conditions is satisfied.

USE

You can place state transition equations anywhere within the state segment except in the CONDITIONS section.

When you create transition equations, use the state-equation operator, :=, to separate the present state from the transition. Use the transition operator, ->, to identify the condition and corresponding transition state. Use the OR operator (+) to indicate additional transitions. Use the default operator, +->, to identify the local default.[114]

In the example, when the state machine is in state TWO, it will transition to state THREE if the input conditions specified for COUNT_UP are satisfied. It will transition to state ONE if the conditions for COUNT_DWN are satisfied. If neither of these conditions is satisfied, it will remain in state TWO.

[114] Refer to the following topics, in this chapter, for additional details: CONDITIONS, DEFAULT_BRANCH, DEFAULT_OUTPUT, LOCAL DEFAULT, STATE, STATE ASSIGNMENT EQUATION, and STATE EQUATIONS.

STRING

This keyword allows you to assign a name to frequently used character strings such as equations and expressions. You can then use the string name anywhere in the remainder of your design file. The software substitutes the character string for the name during processing.

Devices Supported: All PLD devices.

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

```
        STRING        String Name         'String'
```

Example

```
        ...
        STRING          IN1               'A1 + /A2 + A3'
        STRING          IN2               '(A1 + /A2 + /A3)'
        ...
   EQUATIONS
             ...
             BANK1 = IN1
             05 = /IN1
             06 = /IN2
             ...
```

Definitions

Parameters following the keyword STRING are defined below.

String Name

This is the name assigned to a cluster of equations, expressions or other parameters. The name can then be used in the equations or state segments of a design to refer to the entire cluster, without having to list all the characters separately. Follow the rules below.

- Assign a unique name of up to 14 alphanumeric characters.
- Do not use keywords, operators, or reserved words.

- Place the name after the keyword STRING and before the PIN or NODE statements.

String

String identifies the cluster of characters defined by the string name. Single quotes are delimiters and identify the characters to be substituted. The software substitutes these characters literally. However, functional strings, must conform to the rules of the function, such as in an expression. The software does not limit the number of characters you can substitute.

USE

Place at least one blank between the keyword STRING, the string name, and the cluster of characters comprising the string. Extra blanks or tabs are reduced to one blank.

In the syntax example, the software processes the string substitution as follows:

```
BANK1 = A1 + /A2 + A3
05 = /(A1 + /A2 +A3)
06 = /(A1 + /A2 +/A3)
```

To DeMorganize an expression when you complement a string name. Use parentheses to enclose the expression. This is only true for expressions. For example, you cannot complement an .OUTF statement.

The following table illustrates the effect of complementing string names.

STRING	EXPRESSION
IN1	A1 + /A2 + A3
/IN1	/A1 + /A2 + A3
IN2	(A1 + /A2 + /A3)
/IN2	/A1 + A2 + A3

Complementing string IN1 causes only the first pin in the string to invert polarity. In contrast, complementing string IN2 DeMorganizes the entire string.

The keywords GROUP and STRING have distinctly different uses. Use GROUP only for clustering pins. Use STRING to substitute any string of characters.[115]

[115] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, DECLARATION SEGMENT, and GROUP.

TEST

This keyword verifies that values at the Q outputs of **registers** are equal to expected values, and creates "T" test vectors per the JEDEC 3B standard.

The Test command changes the simulation results to match the specified signal values, and generates corresponding test vectors in the JEDEC file.[116]

Devices Supported: MACH-device designs only.

If you use the TEST command with non-MACH PLDs, it is converted to a CHECKQ command automatically.

SYNTAX

You use the TEST command in either the simulation segment of a PDS file or in an auxiliary simulation file for Boolean, state-machine, or schematic-based designs.

Syntax

```
   TEST       Prefix_rns
```

Example

```
   SIMULATION
   TEST       /Q1 Q2
```

Definitions

Because the TEST command verifies signal values at the Q output of registers, you do not need to account for active-low pin declarations. This makes TEST especially useful for verifying states.

Parameters following the command TEST are defined below. Additional details are provided under Use.

[116] Refer to the JEDEC *JESD3-B Standard* for additional details regarding test-vector generation.

Prefix

The prefix indicates the logic state of the corresponding register, node, or state. Do not leave a blank between Prefix and rns. There are two prefixes: null and forward slash.

- The null prefix indicates the register or node should be a logical 1. In the syntax example, Q0 has a null prefix.

  When used in conjunction with a state name, a null prefix indicates the specified state should be checked. In the syntax example, PLAYING has a null prefix.

- The forward slash indicates that the signal should be a logical 0. In the syntax example, Q1 has a forward-slash prefix.

Note: If the simulated value does not match the expected value, the TEST command forces the expected value. The expected value appears in the test vectors, and a clash is indicated in the simulation results.

Rns

Rns defines the names of the output registers, nodes, or states to be verified. Each value represents both the signal name or state and the expected output value.

- Each signal name can be up to 14 characters in length.

- Include up to 76 characters per line and use as many lines as you need.

  The screen displays up to 76 characters per line; however, all information is processed properly even if it extends beyond the 76th character.

TEST

- Include a blank between the keyword and the first register, node, or state in the list.

  You can also include **multiple** register and node names. You can also use strings or vector notation to define the signal list.

- Separate multiple prefixed register and node names with a blank.

USE

A conflict occurs when the value of the output register does not match the value defined in the TEST command. Each conflict is identified with a question mark in the simulation output files; a warning is issued and the expected value is reported in the execution log file.

.T EQUATION

This equation defines when to set the T input on T-type flip-flops high.

Devices Supported: PALCE610.

SYNTAX

Use the .T EQUATION in the equations segment of Boolean or state-machine designs.

```
Syntax

             Pn.T          Assignment operator      Expression
Example

   EQUATIONS
             ...
             Q1.T                  =                IN1 * /IN2
             ...
```

Definitions

All parameters are defined below.

Pn.T

Pn.T is the pin or node associated with the T flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[117]

Expression

Expression identifies the logic defining when the input on .T-type flip-flops is set high. In the syntax example, when IN1 is true and IN2 is false, the flip-flop associated with the pin or node Q1 is set high.

[117] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .T EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin. If you do, the software reports an error during compilation and processing stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.T is not allowed.
- You can use the GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .T EQUATION to several pins.[118]

[118] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, and .S EQUATION.

.T1 EQUATION

This equation defines when to set the T1 input on dual toggle, 2-T, flip-flops high.

Devices Supported: PAL22IP6.

SYNTAX

You use the .T1 equation in the equations segment of Boolean or state-machine designs.

```
Syntax
             Pn.T1        Assignment Operator      Expression
Example
   EQUATIONS
             ...
             Q1.T1                  =              IN1 * /IN2
             ...
```

Definitions

All parameters are defined below.

Pn.T1

Pn.T1 is the pin or node associated with the dual toggle flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[119]

Expression

Expression identifies the logic you define to determine when to set the T1 input on dual toggle flip-flops high. In the example, when IN1 is true and IN2 is false, the T1 input in the dual toggle flip-flop associated with the pin or node Q1 is set high.

[119] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .T1 EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.T1 is not allowed.
- You can use GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .T1 equation to several pins.[120]

[120] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

.T2 EQUATION

This equation defines when to set the T2 input on dual toggle, 2-T, flip-flops high.

Devices Supported: PAL22IP6.

SYNTAX

You use the .T2 EQUATION in the equations segment of Boolean or state-machine designs.

Syntax

```
            Pn.T2         Assignment Operator      Expression
```

Example

```
   EQUATIONS
            ...
            Q1.T2                  =               IN1 * /IN2
            ...
```

Definitions

All parameters are defined below.

Pn.T2

Pn.T2 is the pin or node associated with the dual toggle flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.[121]

Expression

Expression identifies the logic you define to determine when to set the T2 input on dual toggle flip-flops high. In the syntax example, when IN1 is true and IN2 is false, the T2 input in the dual toggle flip-flop associated with the pin or node Q1 is set high.

[121] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .T2 EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.

- You cannot use negative polarity on the left side of the equation. For example, /Q1.T2 is not allowed.

- You can use GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .T2 equation to several pins.[122]

[122] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

TITLE

This keyword begins the statement that defines the title of the design. Including the design title is useful for documentation purposes.

Devices Supported: All PLD devices.

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

```
          TITLE                         Design Title
```

Example

```
          TITLE                         Traffic Controller
          PATTERN
          REVISION
          AUTHOR
          COMPANY
          DATE
          CHIP
```

Definitions

Only the descriptor following the keyword TITLE is discussed.

Design Title

Design title is an optional name that includes any combination of up to 59 alphanumeric characters indicating the company's name.

- You can use other symbols or punctuation; however you cannot use the dollar sign.
- You can use reserved words in this statement.

USE

There are two ways to enter the design title.

- Use the declaration segment form, select the TITLE field, and type the name.
- Type the design title first, followed by the pattern in the PDS file using a text editor.

The following error conditions pertain to the TITLE statement.

- Without a design TITLE statement, a warning is issued and processing continues.
- With multiple design TITLE statements, an error is reported and processing stops.[123]

[123] Refer to the following topics, in this chapter, for additional details: AUTHOR, COMPANY, DATE, DECLARATION SEGMENT, PATTERN, and REVISION.

TRACE_OFF

TRACE_OFF defines the end of the simulation section being traced by TRACE_ON.

Devices Supported: All PLD devices.

SYNTAX

Use the reserved word in the simulation segment or auxiliary simulation file of Boolean and state-machine designs.

Syntax

```
    TRACE_OFF
```

Example

```
    SIMULATION
    TRACE_ON
    TRACE_OFF
```

Definitions

No parameters are required with this keyword.

TRACE_OFF

This reserved word indicates when to conclude the simulation section being traced by TRACE_ON.

USE

If you do not conclude a trace section with TRACE_OFF, the software terminates the trace at the end of the file and displays a warning. If you have multiple traces but do not conclude one, the software does so by assuming a TRACE_OFF before the next TRACE_ON statement.[124]

[124] Refer to SIMULATION and TRACE_ON, in this chapter, for additional details.

TRACE_ON

TRACE_ON defines which signal values to record in the trace file during simulation. Tracing allows you to reorder and group signals however you wish. Tracing also resolves reverse polarity problems because you can define polarity in the TRACE statement.

Devices Supported: All PLD devices.

SYNTAX

Use the keyword in an auxiliary simulation file or the simulation segment of Boolean and state-machine designs.

Syntax

```
   TRACE_ON Pn
```

Example

```
   SIMULATION
   TRACE_ON O1 O2 /IN2 /IN3
```

Definitions

Only the parameter following the keyword TRACE_ON is defined below.

Pin, Node

Pin, node is a list of pin or node names, as defined in the PIN and NODE statements of the declaration segment.

USE

Observe the following usage conventions when using TRACE_ON.

- You can repeat TRACE_ON statements.
- You cannot nest TRACE_ON statements. If the software finds a new TRACE_ON statement, it abandons the previous statement and continues with the new statement.
- You can list multiple pins or nodes with TRACE_ON. Separate pins and nodes with only a blank.

- You can reverse pin or node polarity by placing or removing a forward slash before the pin or node name. You cannot use polarity with states.

- You can use strings and groups with TRACE_ON.

- Conclude a trace section with a TRACE_OFF command. If you do not conclude a trace with TRACE_OFF, the software terminates the trace at the end of the file and displays a warning.

During simulation, the software generates a simulation history file. You view this file as either an ASCII table or as a history waveform.

Tracing causes the software to create the trace file, which contains the simulation results of the pins and nodes selected by TRACE_ON. The signals are listed in the same order and with the same polarity as listed in the TRACE_ON statement. If TRACE_ON is not used, then no trace file is created.

The software converts state names to state bits and appends them to the TRACE_ON statement for simulation.[125]

[125] Refer to SIMULATION and TRACE_ON, in this chapter, for additional details.

.TRST

This reserved word defines when to enable three-state outputs on devices with programmable enable.

Devices Supported					
PAL10H20EG8	PAL10H20EV8	PAL16L8	PAL16P8	PAL16R4	PAL16R6
PAL16R8	PAL16RA8	PAL16RP4	PAL16RP6	PALCE16V8	PAL18P8
PALC18U8	PAL20L10	PAL20L8	PAL20R4	PAL20R6	PAL20RA10
PAL20RS4	PAL20RS8	PAL20S10	PAL20X4	PAL20X8	PALCE20V8
PAL22IP6	PAL22RX8	PAL22V10	PAL23S8	PALCE29M16	PALCE29MA16
PAL32VX10	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state designs.

```
Syntax

            Pn.TRST      Assignment Operator      Expression
Example

   EQUATIONS
            Q0                     =               /Q0
            Q1.TRST                =               I1 * /I2
            ...
```

Definitions

All parameters are defined below.

Pn.TRST

Pn.TRST is associated with the three-state buffer. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation, as interpreted by the software when processing design files.[126]

[126] Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

Expression

Expression defines the logic conditions that determine when to enable three-state outputs on devices with programmable enable.

USE

Multiple .TRST statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting.

You can use more than one product term for a three-state buffer depending on the device. Some devices using product terms also have a pin for controlling the buffer which overrides the logic.

You **cannot** complement the pin name. For example, if a signal is defined as Q1 in the PIN statement, /Q1.TRST is not permitted. You **can** complement the Boolean expression, if supported by the device, as follows:

```
Q1.TRST = I1 * I2 * I3 for active high
Q1.TRST = /(/I1 * I2 * I3) for active low
```

.TRST provides several logical means of enabling the outputs on some devices, such as the PALCE29M16. Examples follow.[127]

- A dedicated output:

  Q[1..4].TRST = VCC

- A dedicated input:

  O[5..8].TRST = GND

[127] Refer to Chapter 11, in this section, for additional details regarding devices that support programmable enable.

- A product-term enable, XOR:

  O[9..12].TRST = I1 * I2 :*: I3

- OE pin enable, where there is a choice between a product term and a dedicated output enable pin:

  O[1..4].TRST = IOE

- Unconditional high, where an output equation exists for a pin and no .TRST equation exists:

  Q.TRST = VCC

- Unconditional low, where no output equations or .TRST equation is defined:

  Q.TRST = GND

Many PAL devices have dedicated enable pins to control some or all three-state outputs. For these outputs, no .TRST equation is needed.[128]

[128] Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, GROUP, STRING, and VECTOR.

VCC

You can include this reserved word in an equation to hold a pin, node, or functional equation unconditionally high.

Devices Supported: All PLD devices.

SYNTAX

You use the reserved word, VCC, in the equations segment of Boolean and state-machine designs.

Syntax

```
            Pn or
            Functional Equation   Assignment Operator    VCC
```

Example

```
  EQUATIONS
  ...
            OUT5                           =                        VCC
            OE1.TRST                       =                        VCC
  ...
```

Definitions

The element preceding the reserved word is described below.

Pn or Functional Equation

Pn or functional equation defines the element to be held high.

- The pin or node name defined in the PIN or NODE statement of the declaration segment

- The pin or node function defined in an earlier functional equation.[129]

[129] Refer to the following topics, in this chapter, for additional details: FUNCTIONAL EQUATIONS, NODE, and PIN.

USE

VCC is normally used for functional equations. You can use 1 instead of VCC anywhere you want an unconditional high value.[130]

Important: You must define the VCC pin in a PIN statement.

[130] Refer to GND, in this chapter, for additional details.

VECTOR

VECTOR notation allows you to assign a set of descriptions to a set of pins or nodes. It also allows you to compare the value of a set of pins or nodes to a radix in a CASE statement.

Devices Supported: All PLD devices.

SYNTAX

Use vector notation in Boolean and state-machine designs.

Syntax

```
             PIN                  x1..xn         NUM[y1..yn]
```

Example

```
   DECLARATION

             PIN 1 INA COMB
             ...
             PIN                  5..2           OUT[4..1]
             PIN                  6, 10..8       DATA[4..1]
             PIN                  18..11         ADD[7..0]
             ...
   EQUATIONS
   ...
   CASE                                          (ADD[7..0])
                                  BEGIN
                                         #hOF:
                                  BEGIN
                                  ...
                                  END
             END
   ...
```

Definitions

Parameters following the keyword PIN are defined below. Additional details are discussed under Use.

x1..xn

This notation specifies the user-defined pin number range. You must have the same number of pin numbers as pin names.

y1..yn

This notation specifies the user-defined pin name range. You must have the same number of pin names as pin numbers.

VECTOR is essentially a dual range syntax: "for this range of pins use this range of names."[131]

USE

To use VECTOR notation, you must do the following.

- Declare all pins in the vector in one PIN statement.
- Use subscripted pin or node names with the format NAME[1] rather than NAME 1.

You can include input and output pins in the same vector if your application calls for it, but you cannot include pins and nodes in the same vector.

In the syntax example, the software converts the pin definitions as follows.

```
PIN 2   OUT[1]
PIN 3   OUT2]
...
PIN 18  ADD[7]
```

In the CASE statement, the software tests the value on pins ADD[7] through ADD[0]. If they match the hex value 0F, the instructions between the BEGIN and END statements are executed.

[131] Refer to OPERATOR, in this chapter, for additional details.

WHILE-DO

This is a simulation construct that looks at a logical condition and performs a specified task as long as the condition is true.

Devices Supported: All PLD devices.

SYNTAX

Use the WHILE-DO construct in an auxiliary simulation file or the simulation segment of Boolean and state-machine designs.

Syntax

```
   WHILE (Condition) DO
           BEGIN
                                    Task
           END
```

Example

```
   SIMULATION
           ...
           WHILE (/BIT2 * /BIT3) DO
                                    BEGIN
                                                CLOCKF CLOCK
                                    END
           ...
```

Definitions

The following structures are part of the WHILE-DO construct.

Condition

Condition is any Boolean expression. You can use more than one condition if you separate them by commas; the software ANDs multiple conditions together. If the WHILE-DO construct is nested in a FOR-TO-DO construct, the condition can also be the index variable of the FOR-TO-DO construct. You cannot use an index variable outside its defining FOR-TO-DO construct.

Use parentheses to enclose the WHILE condition. However, you cannot nest parentheses.

Task

The simulation task the software performs during the WHILE-DO loop. Use BEGIN and END statements to enclose the task; indent the statements to make your program easier to follow.

USE

You can nest WHILE-DO constructs within CASE, FOR-TO-DO, IF-THEN-ELSE, and other WHILE-DO constructs. There is no limit to the number of constructs you can include in your design. However, using a minimal number of nests may make your program easier to follow and faster to compile.

The condition can be any Boolean expression of logic signals or mathematical equality: =, >, <, >=, <=, and <>.[132]

[132] Refer to the following topics, in this chapter, for additional details: FOR-TO-DO, IF-THEN-ELSE, and SIMULATION.

CHAPTER 11

DEVICE PROGRAMMING REFERENCE

CONTENTS

11 DEVICE PROGRAMMING REFERENCE

This chapter provides datasheets on the PALASM language syntax, examples of language use, and information related to programming PLD and MACH devices.

- The PLD introduction, 11.1, discusses the purpose of, and guidelines for using, this device programming reference.
- The PLD cross-reference table, 11.2, tabulates programming features for each device and indicates where you can find a language syntax example for each feature.
- The general PLD language syntax, 11.3, explains general device features and shows the language syntax needed to use them.
- The PLD device syntax datasheets, 11.4, provide information relating to the PALASM language for devices with special features.

> **Important:** Standard devices do not have a PLD device syntax datasheet.

- The MACH 1 and MACH 2 series, 11.5., furnishes device, language, and programming information.

11.1 PLD INTRODUCTION

This chapter is organized so you can quickly and easily familiarize yourself with PALASM language constructs and the syntax required for designing with different programmable devices.

- Review the feature cross-reference table for programming features that apply to the device you want to design with.
- Read the general syntax for programming features marked with an X in the cross-reference table.
- Read the corresponding device syntax for features marked with an asterisk, *, in the cross-reference table.
- For features marked with an ampersand, @, in the device feature cross-reference table, read the general syntax datasheets for the language syntax and the corresponding device syntax datasheet for node locations and descriptions.[1]

11.1.1 PLD NAMING CONVENTIONS

The naming of devices follows the convention used in the PAL Device Data Book. For clarity, only the number of a device is addressed. For example, 22V10 includes the devices PAL22V10, AmPAL22V10, and PALCE22V10; 16R8 includes the family of devices PAL16L8, PAL16R8, PAL16R6, and PAL16R4. The actual devices are listed under the alphanumeric device reference index. For each device syntax datasheet, the actual devices are listed under the device number heading.

[1] Refer to Chapter 10, in this section, for an in-depth discussion of particular language elements.

11.1.2 STANDARD PLD DEVICES VERSUS NON-STANDARD PLD DEVICES

For ease of reference, devices whose features are covered by the **general** syntax data sheet are grouped as standard programmable devices.[2] Devices that require specific language syntax are grouped as non-standard programmable devices.[3]

105	PLS105 / PLSCE105	11-42
167/168	PLS167 / PLSCE167	11-46
16RA8	PAL16RA8	11-50
16V8HD	PAL16V8HD	11-54
20EG8	PAL10H20EG8 / PAL10020EG8	11-64
20EV8	PAL10H20EV8 / PAL10020EV8	11-68
20RA8	PAL20RA8	11-72
22IP6	PAL20IP6	11-76
22V10	PAL22V10 / AmPAL22V10 / PALCE22V10	11-82
23S8	AmPAL23S8	11-86
26V12	PALCE26V12	11-92
29M16	PALCE29M16	11-98
29MA16	PALCE29MA16	11-106
30S16	PLS30S16	11-112
32VX10	PAL32VX10	11-132
610	PALCE610	11-144

[2] Standard programmable devices include 16R8 family, 16V8, 18P8, 20R8 family, 20V8, 20X10/20L10 family, 22P10, 24R10 family, and 24V10.

[3] Non-standard programmable devices include 105, 167/168, 16RA8, 16V8HD, 20EG8, 20EV8, 20RA8, 22IP6, 22V10, 23S8, 26V12, 29M16, 29MA16, 30S16, 32VX10, and 610.

11.2 DEVICE FEATURE CROSS-REFERENCE

The following table cross references the features for each programmable device. It also identifies whether a particular device requires special language constructs and where to look for that information.

PLD Device Feature Cross-Reference Table	Standard Programmable Devices													Non-Standard Programmable Devices															
Features	16L8/20L8	16R4/20R4	16R6/20R6	16R8/20R8	16V8/20V8	18P8	20L10/24L10	20X4/20X8	20X10	22P10	24R4/24R8	24R10	24V10	105	167/168	16RA8	16V8HD	*1 20EG8	*2 20EV8	20RA10	22IP6	22V10	23S8	26V12	29M16	29MA16	30S16	32VX10	610
Output-Enable Control																													
Common		X	X	X				X	X		X	X																	
Individual						X				X								X	X		X	X		X				X	
Common / individual					X								X				X									X			
Others														*	*	*				*			*		*		*		*
Clock Control																													
Common	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X	X			X	X					X	
Individual																X				X									
Others																	*							*	*	*	*		*
Preset Control																													
Individual																X				X						X			
Global																		X	X			X	X	X	X			X	
Others														*	*						*						*		
Reset Control																													
Individual																X				X						X			
Global																						X	X	X	X			X	
Others																					*						*		
Device Polarity																													
Active low	X	X	X	X			X	X	X		X	X																	
Active high														X	X												X		
Programmable					X	X				X			X			X	X	X	X	X	X	X	X	X	X	X		X	X
Output Logic Types																													
Combinatorial	X	X	X		X	X	X	X		X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D flip-flop		X	X	X	X			X	X		X	X	X			X	X		X	X		X	X	X	X	X		X	*
SR flip-flop														X	X						*						X		*
Latch																		X							X	X			
T flip-flop																					*								*
JK flip-flop																													*
Macrocells - different configs.																							*		*	*	*		*
Non-programmable Feedback																													
Comb. output - I/O feedback	X	X	X		X	X	X	X		X	X		X			X				X	X	X	X						
Reg. output - I/O feedback																X				X	X		X						
Reg. output - /Q feedback		X	X	X	X			X	X		X	X	X									X							

Notes: *1 PAL10H20G8, PAL10020G8
*2 PAL10H20V8, PAL10020V8
X See the general PLD Language Syntax for general language syntax; if necessary, see also the corresponding PLD Device Syntax datasheet for node descriptions and locations.
* See the corresponding PLD Device Syntax datasheet for device dependent features.

PLD Device Feature Cross-Reference Table (Continued)	**Standard Programmable Devices**													**Non-Standard Programmable Devices**															
Features	**16L8/20L8**	**16R4/20R4**	**16R6/20R6**	**16R8/20R8**	**16V8/20V8**	**18P8**	**20L10/24L10**	**20X4/20X8**	**20X10**	**22P10**	**24R4/24R8**	**24R10**	**24V10**	**105**	**167/168**	**16RA8**	**16V8HD**	*1 **20EG8**	*2 **20EV8**	**20RA10**	**22IP6**	**22V10**	**23S8**	**26V12**	**29M16**	**29MA16**	**30S16**	**32VX10**	**610**
Programmable Feedback																													
Output with I/O feedback																		X	X				X	X	X	X		*	*
Output with /Q feedback																							X	X	X	X		*	
Output with Q feedback																											*		*
Output with I/O and /Q feedback																									X	X		*	
Output with I/O and Q feedback																											*		
Output with I/O and latch feedback																	*												
Buried register with /Q feedback																							X		X	X		*	
Buried register with Q feedback														X	X												X		
Register input with /Q																									X	X			
Latch input																	*												
Preload Control																													
Supervoltage					X								X				X	X	X			X	X	X	X	X	X	X	X
Product term																									X	X			
Others																*				*									
Observability Product Term Control																							X		X	X	X		
Miscellaneous																													
Complement Array														X	X												X		
State Bits														*	*												*		
Electronic Signature					X								X				X												
Bypassable Register																												*	
Input Latch																	*												
Input Register																											*		
Open-Collector Output																	*												

Notes: *1 PAL10H20G8, PAL10020G8
*2 PAL10H20V8, PAL10020V8
X See the general PLD Language Syntax for general language syntax; if necessary, see also the corresponding PLD Device Syntax datasheet for node descriptions and locations.
* See the corresponding PLD Device Syntax datasheet for device dependent features.

11.3 GENERAL PLD LANGUAGE SYNTAX

Each discussion below explains a feature commonly found in most programmable devices.[4]

- 11.3.1, Output-Enable Control
- 11.3.2, Clock Control
- 11.3.3, Preset Control
- 11.3.4, Reset Control
- 11.3.5, Device Polarity
- 11.3.6, Combinatorial Logic
- 11.3.7, Registered or Latched Logic
- 11.3.8, Feedback
- 11.3.9, Preload Control
- 11.3.10, Observability Product Term Control
- 11.3.11, Complement Array
- 11.3.12, Electronic Signature

Each discussion consists of the following information.

- Description of the feature
- Pertinent, standard PALASM language syntax
- Example(s) of the language as it is used

All language elements required to design with standard programmable-device features and those elements needed for certain general features of non-standard programmable devices are included. Use this information in conjunction with the device programming syntax datasheet when designing with non-standard programmable devices.

Note: In each syntax example, *italicized* information is provided as an explanation and is not part of the actual syntax.

[4] Refer to the Device Programming Feature Cross-Reference Table for more information about individual device features.

11.3.1 OUTPUT-ENABLE CONTROL

Three major types of output-enable control are identified below and discussed next.

- Common external output
- Individual product term control
- Common external or individual product term control

11.3.1.1 Common External Output-Enable Pin

A common external output-enable pin controls all three-state buffers within the device. Since the three-state buffers are externally controlled by an input pin, **no language syntax is required.**

Devices Supported: Devices with common external output-enable control include the 16R8, 20R8, 20X10/20L10, and 24R10.

11.3.1.2 Individual Product Term Control

Each three-state buffer is controlled individually by a product term. To use these product terms, define each product term control individually in the language syntax and example, as shown next.

Devices Supported: Devices with individual product term output-enable control include the 20EG8, 20EV8, 22IP6, 26V12, and 32VX10.

Note: For the 20EG8 and 20EV8, even though the output buffer is two-state instead of three-state, the syntax you use to define the product term control is the same as shown next.

Syntax

```
Pin Statement(s)   :
                   PIN  Output_pin_location        Output_pin_name         Storage_type
                   :
Equation(s)        Output_pin_name.TRST = Boolean expression using one product term
```

Example

```
:
CHIP EXAMPLE PALCE16V8
:
PIN     2    I1     COMB        ;input
PIN     3    I2     COMB        ;input
PIN     12   01     COMB        ;output
:
01.TRST = I1 * I2               ;output enable controlled by product term (I1 * I2)
```

11.3.1.3 Common External Pin or Individual Product Term Control

Each three-state buffer can be controlled using either of the following methods.

- An external, common output-enable pin
- An individual product term

In addition, these three-state buffers can be either

- permanently enabled or
- permanently disabled.

Use the syntax shown in the following examples to define the three-state buffer control.

Devices Supported: Devices that have output enable with common external or individual product term are the 16V8, 16V8HD, 20V8, 24V10, and 29MA16.

Note: The output enable defaults are as follows.

- If the pin is defined as an output, pin.TRST = VCC
- If the pin is not defined as an output, pin.TRST = GND

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location        Output_pin_name        Storage_type
                  PIN  Output_enable_pin_location Output_enable_pin_name  Storage_type
                  :
```

```
Pin Statement(s)  :
                  Output_pin_name.TRST
                  external common output-enable pin
                  = Output_enable_pin_name
     or           individual product term
                  = Boolean expression using one product term
     or           permanently enabled
                  = VCC      ;default for pins defined as outputs
     or           permanently disabled
                  = GND      ;default for pins not defined as outputs
```

Example 1

External common-enable pin control

```
:
CHIP EXAMPLE PALCE16V8
:
PIN     11   IOE    COMB        ;input
PIN     12   01     COMB        ;output
:
01.TRST = IOE                   ;output controlled by input pin IOE
```

Example 2

Individual product term output-enable control

```
:
CHIP EXAMPLE PALCE16V8
:
PIN     2    I1     COMB        ;input
PIN     3    I2     COMB        ;input
PIN     12   01     COMB        ;output
:
01.TRST = I1 * I2               ;output enable controlled by product term (I1 * I2)
```

Example 3

Output enable permanently enabled

```
:
CHIP EXAMPLE PALCE16V8
:
PIN      12   01     COMB          ;output
:
01.TRST = VCC                      ;output permanently enabled
```

Example 4

Output enable permanently disabled

```
:
CHIP EXAMPLE PALCE16V8
:
PIN      12   01     COMB          ;output
:
01.TRST = GND                      ;output permanently disabled
```

11.3.2 CLOCK CONTROL

In general, there are two types of clock control for registered and latched programmable devices.

- Common external clock control
- Individual product term clock control

The two clock control types are discussed below.

11.3.2.1 Common External Clock Control

A dedicated clock pin is used to clock all registers in the device. Since the clock configuration is fixed for these devices, no special language syntax is required.

Devices Supported: Devices that have a common external clock pin include the 105, 167/168, 16R8, 16V8, 18P8, 20EG8, 20EV8, 20R8, 20V8, 20X10, 22P10, 22V10, 23S8, 24R10, 24V10, and 32VX10.

11.3.2.2 Individual Product Term Clock Control

The clock input for each register is individually controlled by a product term. You must use the syntax below to program each product term for the clock control.

Devices Supported: Devices that have clock control through an individual product term are the 16RA8 and 20RA10.

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location      Output_pin_name       Storage_type
                  :
Equation(s)       Output_pin_name.CLKF = Boolean expression using one product term
```

Example

Individual product term clock control

```
:
CHIP EXAMPLE 16RA8
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     12   O1    REG         ;registered output
:
O1.CLKF = I1 * /I2             ;clock input to the register of output O1 is controlled
                               ;    by the product term (I1* /I2)
```

11.3.3 PRESET CONTROL

There are two types of preset control.

- Individual product term control
- Global product term control [5]

[5] Refer to discussion 11.4 for node locations and information.

11.3.3.1 Individual Product Term Control

Each preset input is individually controlled by a programmable product term. To program the preset product terms, you use the language syntax to define each preset input, as shown next.

Devices Supported: Devices that have preset/reset control with an individual product term are the 16RA8, 20RA10, and 29MA16.

Syntax

```
Pin Statement(s)   :
                   PIN  Output_pin_location       Output_pin_name          Storage_type
                   :
Equation(s)        Output_pin_name.SETF = Boolean expression using one product term
```

Example

Preset with individual product term control

```
:
CHIP EXAMPLE PAL16RA8
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     12   O1    REG         ;registered output
:
O1.SETF = I1 * /I2             ;preset input to the register of output O1 is controlled
                               ;    by the product term (I1 * /I2)
```

11.3.3.2 Global Product Term Control

The preset inputs of all registers within the device are controlled by a common global preset product term. To program this global product term, you must declare the global node in the pin statement segment, as shown next.

Devices Supported: Devices that have preset with global product term control are the 20EG8, 20EV8, 22V10, 23S8, 26V12, 29M16, and 32VX10.

Syntax

```
Pin Statement(s)   :
                   NODE  Global_node_location    Global_node_name
                   :
Equation(s)        Global_node_name.SETF = Boolean expression using one product term
```

Example

Preset with global product term control

```
:
CHIP EXAMPLE
:
PIN     2    I1     COMB          ;input
PIN     3    I2     COMB          ;input
:
NODE    1    GLOBAL               ;internal global node
:
GLOBAL.SETF = I1 * /I2            ;preset inputs to all registers are globally controlled
                                  ;    by the product term (I1 * /I2)
```

11.3.4 RESET CONTROL

There are two types of reset control.

- Individual product term control
- Global product term control[6]

11.3.4.1 Individual Product Term Control

Each reset input is individually controlled by a programmable product term. To program the preset product terms, you use the language syntax to define each preset input, as shown next.

Devices Supported: Devices that have reset control with an individual product term are the 16RA8, 20RA10, 29MA16, and 610.

[6] Refer to discussion 11.4 for node locations and information.

Syntax

```
Pin Statement(s)    :
                    PIN  Output_pin_location        Output_pin_name          Storage_type
                    :
Equation(s)         Output_pin_name.RSTF = Boolean expression using one product term
```

Example

Reset with individual product term control

```
:
CHIP EXAMPLE PAL16RA8
:
PIN      2     I1     COMB          ;input
PIN      3     I2     COMB          ;input
PIN      12    01     REG           ;registered output
:
01.RSTF = I2                        ;reset input to the register of output 01 is controlled
                                    ;    by the product term (I2)
```

11.3.4.2 Global Product Term Control

The reset inputs of all registers within the device are controlled by a common global reset product term. To program this global product term, you must declare the global node in the pin statement segment, as shown below.

Devices Supported: Devices that have preset/reset with global product term control are the 22V10, 23S8, 26V12, 29M16, and 32VX10.

Syntax

```
Pin Statement(s)    :
                    NODE  Global_node_location    Global_node_name
                    :
Equation(s)         Global_node_name.RSTF = Boolean expression using one product term
```

Example Reset with global product term control

```
:
CHIP EXAMPLE
:
PIN      2     I1     COMB          ;input
PIN      3     I2     COMB          ;input
:
NODE     1     GLOBAL               ;internal global node
:
GLOBAL.RSTF = I2                    ;reset inputs to all registers are globally controlled
                                    ;    by the product term (I2)
```

11.3.5 DEVICE POLARITY

There are three types of device polarity.

- Active low
- Active high
- Programmable

You must be careful about both the pin and equation output polarities when you write equations for each type of device.

11.3.5.1 Active-Low Polarity

Active-low devices are those programmable devices whose outputs pass through the output pins inverted.

For these devices, the output of the equations always has the opposite polarity of the output pin. Examples below show how you can write the same equation for active-low or active-high outputs using active-low devices.

Devices Supported: Active-low devices are the 16R8, 20R8, 20X10/20L10, and 24R10.

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location      Output_pin_name        Storage_type
                  :
Equation(s)       Output_pin_name* = Boolean expression
* Output_pin_name in output equation must have the opposite polarity of the
  pin statement.
```

Example 1

Active-low output using active-low devices

```
:
CHIP EXAMPLE PAL16R8
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     4    I3    COMB        ;input
PIN     12   /01   REG         ;active-low output pin
:
01 = I1 * I2 + I3              ;equation with the opposite polarity of the pin
                               ;    statement
```

Example 2

Active-high output using active-low devices

```
:
CHIP EXAMPLE PAL16R8
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     4    I3    COMB        ;input
PIN     12   01    REG         ;active-high output pin
:
/01 = /I1 * /I3                ;same equation with active-low output using DeMorgan's
    + /I2 * /I3                ;    theorem  /01 = /((I1*I2) + I3)
```

11.3.5.2 Active-High Polarity

Active-high devices are those programmable devices whose outputs pass through the output pins without inversion. For these devices the pin output and the equation output are always the same polarity.

Devices Supported: Active-high devices are the 105, 167/168, and 30S16.

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location      Output_pin_name      Storage_type
                  :
Equation(s)       Output_pin_name* = Boolean expression
```

** Output_pin_name in output equation must have the opposite polarity of the pin statement.*

Example 1

Active-high output using active-high devices

```
:
CHIP EXAMPLE PLS105
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     4    I3    COMB        ;input
PIN     12   O1    COMB        ;active-high output pin
:
O1 = I1 * I2 + I3              ;equation with the same polarity as the pin statement
```

Example 2

Active-low output using active-high devices

```
:
CHIP EXAMPLE PLS105
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     4    I3    COMB        ;input
PIN     12   /O1   COMB        ;active-low output pin
:
/O1 = /I1 * /I3                ;same equation with active-low output using DeMorgan's
    + /I2 * /I3                ;    theorem /O1 = /((I1*I2) + I3)
```

11.3.5.3 Programmable Polarity

You can individually program the polarity of each output.

- If you want an active-high output pin, use the same output polarity for both the pin statement and the equation.
- If you want an active-low output pin, use the opposite polarity for the pin statement and the equation.

Devices Supported: Devices with programmable polarity are the 16RA8, 16V8, 16V8HD, 18P8, 20EG8, 20EV8, 20RA10, 20V8, 22IP6, 22P10, 22V10, 23S8, 24V10, 26V12, 29M16, 29MA16, 32VX10, and 610.

For consistency, all examples use an active-high polarity in each pin statement for each output pin. In this case, the output equation controls the polarity of the pin.[7]

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location      Output_pin_name*      Storage_type
                  :
Equation(s)       Output_pin_name* = Boolean expression

* Output_pin_name can be active high or active low.
```

[7] Refer to Section II, Chapter 4, for details on polarity.

Example 1

Active-high output from a device with programmable polarity

```
:
CHIP EXAMPLE PAL16R8
:
PIN     2    I1    COMB       ;input
PIN     3    I2    COMB       ;input
PIN     4    I3    COMB       ;input
PIN     12   01    COMB       ;active-high output pin
:
01 = I1 * I2 + I3             ;equation with same the polarity as the output pin
```

Example 2

Active-low output from a device with programmable polarity

```
:
CHIP EXAMPLE PAL16R8
:
PIN     2    I1    COMB       ;input
PIN     3    I2    COMB       ;input
PIN     4    I3    COMB       ;input
PIN     12   01    COMB       ;active-low output pin
:
/01 = I1 * I2 + I3            ;equation with the opposite polarity of the pin
                              ;    statement
```

11.3.6 COMBINATORIAL LOGIC

For outputs that use only combinatorial logic, you define the storage type as COMBINATORIAL or use the abbreviation, COMB, in the pin declaration segment of the PDS file. See the syntax description and example shown next.

Note : The default storage type is combinatorial, which is used if you do not define the type.

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location      Output_pin_name        COMB
                  :
Equation(s)       Output_pin_name = Boolean expression
```

Example

Combinatorial output

```
:
CHIP EXAMPLE PAL16R6
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     4    I3    COMB        ;input
PIN     12   01    COMB        ;output
:
01 = I1 * I2 + I3              ;Boolean expression
```

11.3.7 REGISTERED OR LATCHED LOGIC

For outputs that use registered or latched logic, you must define the corresponding output type in the pin declaration segment. On some programmable devices the registers can be programmed into different types, while others have a fixed hardware configuration. In general, there are three types of registers. Their corresponding language syntax is described below.

- D Flip-flop
- SR Flip-flop
- Latch

11.3.7.1 D Flip-Flop

To use registered logic with D Flip-flops, you must define the output type as REGISTERED or REG and write the corresponding equation for the registered output.

Devices Supported: Devices that use D flip-flops are the 16RA8, 16R8, 16V8, 16V8HD, 20EV8, 20RA9, 20R8, 20V8, 20X10/20L10, 22V10, 24R10, 24V10, 26V12, 29M16, 29MA16, 32VX10, and 610.

Syntax

```
Pin Statement(s)  :
                  PIN Output_pin_location        Output_pin_name          REG
                  :
Equation(s)       Output_pin_name = Boolean expression
```

Example

Registered output with D flip-flop

```
CHIP EXAMPLE PAL16R8
:
PIN     2    I1    COMB        ;input
PIN     3    I2    COMB        ;input
PIN     4    I3    COMB        ;input
PIN     12   O1    REG         ;registered output
:
O1 = I1 * I2 + I3              ;equation for registered output O1
```

11.3.7.2 SR Flip-Flop

To use registered logic with SR flip-flops, define the output type as REGISTERED or REG and provide one corresponding equation for each of the R and S inputs.

Devices Supported: Devices that use SR flip-flops are the 105, 167/168, 30S16, and 610.

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location       Output_pin_name          REG
                  :
Equation(s)       Output_pin_name.R = Boolean expression
                  Output_pin_name.S = Boolean expression
```

Example

Registered output with SR flip-flop

```
:
CHIP EXAMPLE PLS167
:
PIN     2    I1     COMB        ;input
PIN     3    I2     COMB        ;input
PIN     4    I3     COMB        ;input
PIN     9    01     REG         ;registered output
:
01.R = I1 * I2 + I3             ;equation for the R input of the register 01
01.S = /I1                      ;equation for the S input of the register 01
```

11.3.7.3 Latch

To use latched logic, define the output type as LATCHED, or LAT, and write the corresponding equation for the latched output.

Devices Supported: Devices that use latched logic are the 20EG8, 29M16, and 29MA16.

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location      Output_pin_name         LAT
                  :
Equation(s)       Output_pin_name = Boolean expression
```

Example

Latch output

```
:
CHIP EXAMPLE PALCE29M16
:
PIN     2    I1     COMB        ;input
PIN     14   I2     COMB        ;input
PIN     23   I3     COMB        ;input
PIN     9    01     LAT         ;latched output
:
01 = I1 * I2 + I3               ;equation for latched output 01
```

11.3.8 FEEDBACK

You specify feedback of logic signals by defining the signals in the pin segment and specifying the logical construction in the equations or state segments. You may then use the defined signal names in other equations to accomplish feedback. This section details feedback specifications by syntax category.

11.3.8.1 Programmable Feedback

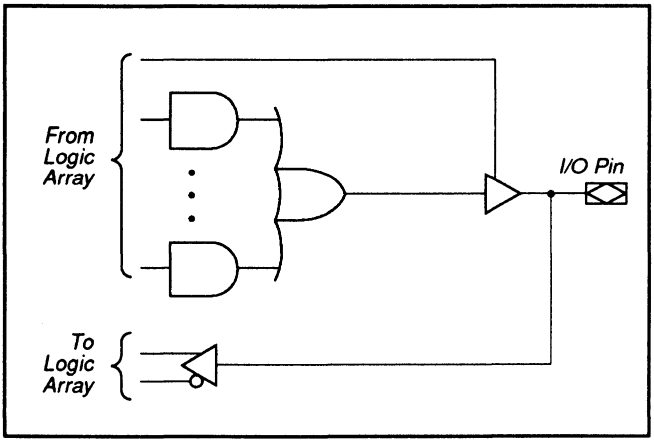
For those programmable devices that have a programmable feedback configuration, there are six possible configuration types.

- Output with I/O feedback
- Output with /Q feedback
- Output with I/O and /Q (dual) feedback
- Buried register with /Q feedback
- Buried register with Q feedback
- Registered input with /Q output.

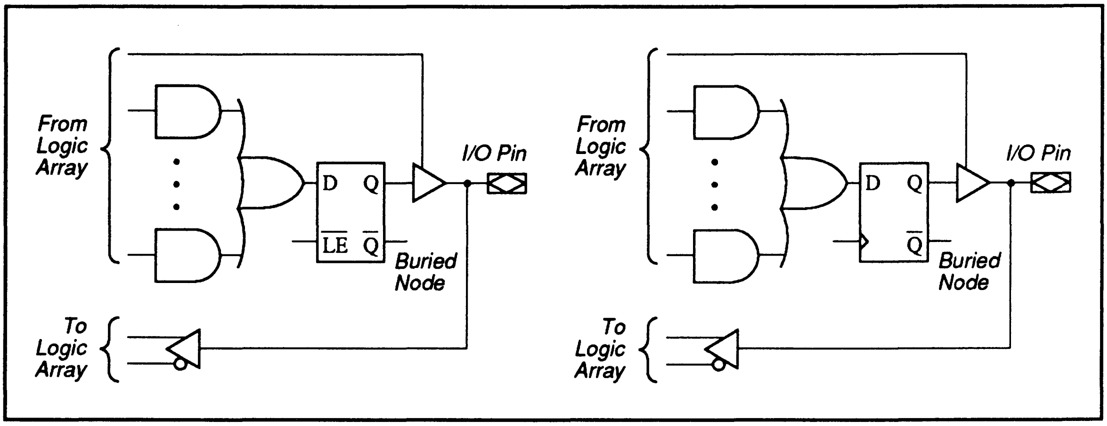
Output with I/O Feedback

The language syntax for combinatorial and registered output with the I/O feeding back to the arrays is shown next.

Devices Supported: Devices that have output with I/O feedback configuration are the 20EG8, 20EV8, 23S8, 26V12, 29M16, 29MA16, 32VX10, and 610.



Combinatorial Output with I/O Feedback



Registered/Latched Output with I/O Feedback

Syntax

```
Pin Statement(s)   :
                   PIN  I/O_pin_location   I/O_pin_name   <I/O_storage_type*>
                   :

Equation(s)        I/O_pin_name = Boolean expression
                   :
                   < Output equations(s) using I/O_pin_name as feedback >
```

* *Use the following table for the appropriate storage types.*

Output Type	*<I/O storage type>*
Combinatorial	*COMB*
D flip-flop	*REG*
Latch	*LAT*

Example 1

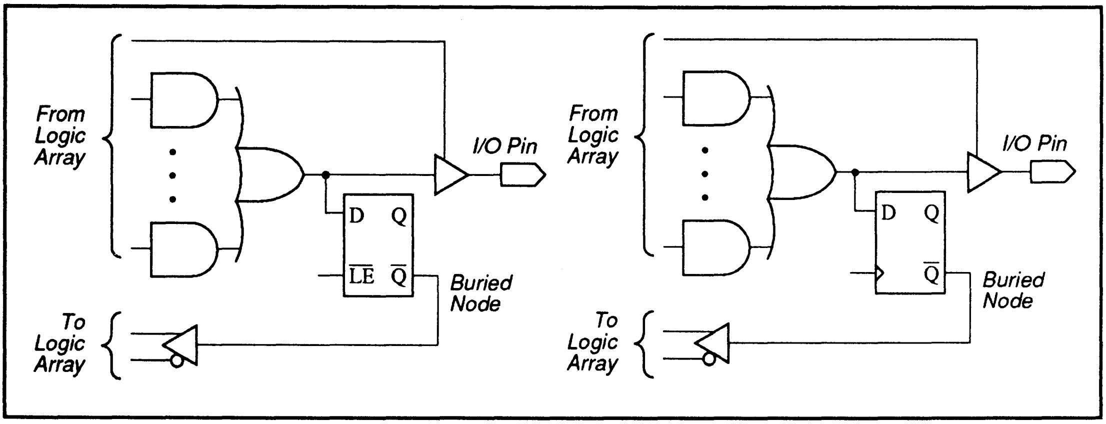
Output with I/O feedback

```
:
CHIP EXAMPLE PALCE29M16
:
PIN     2    I1     COMB       ;input
PIN     14   I2     COMB       ;input
PIN     3    IOF0   COMB       ;I/O, combinatorial
PIN     15   IOF4   REG        ;I/O, registered
IOF0 = I1 * I2 * IOF4          ;equation for IOF0 with feedback from registered
                               ;    I/O, IOF4
IOF4 = IOF0 * I1               ;equation with feedback from combinatorial I/O, IOF0
```

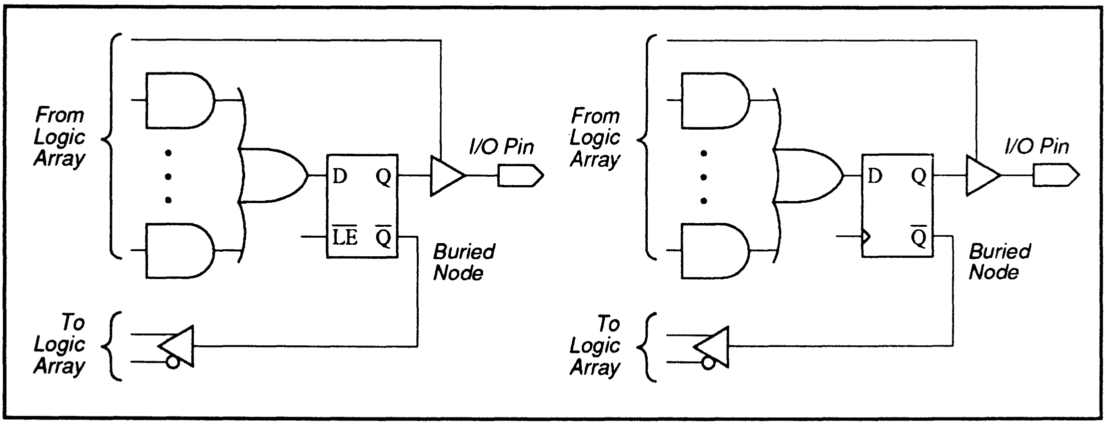
Output with /Q Feedback

The syntax for combinatorial and registered or latched output with the /Q register output feeding back to the arrays is shown below.

Devices Supported: Devices that have output with /Q feedback configuration are the 20EG8, 20EV8, 23S8, 26V10, 29M16, 29MA16, and 32VX10.



Combinatorial Output with /Q Feedback



Registered/Latched Output with /Q Feedback

Syntax

```
Pin Statement(s)  :
                  PIN  I/O_pin_location  I/O_pin_name   <I/O_storage_type*>
                  :
                  NODE  Buried_node_location  Buried_node_name < Node_storage_type*>
                  :
Equation(s)       I/O_pin_name = Boolean expression
                  /Buried_node_name = {  I/O_pin_name**  }
                  :
                  < Output equation(s) using either I/O_pin_name or Buried_node_name as
                       feedback(s) >
```

* *Use the following table for the appropriate storage types.*

Storage element	*<I/O_storage_type>*	*<Node_storage_type>*
D Flip-flop	*COMB or REG*	*REG*

** *I/O_pin_name inside curly brackets must use the same polarity as defined on the left side of the output equation.*

Example

Combinatorial output with /Q feedback

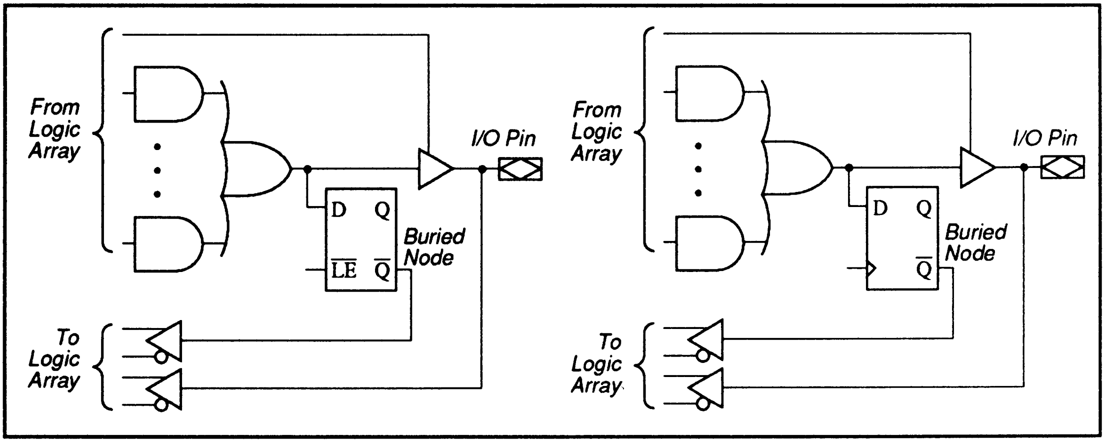
```
CHIP EXAMPLE PALCE29M16
:
PIN     2    IO    COMB       ;input
PIN     14   I1    COMB       ;input
PIN     9    IOF2  COMB       ;I/O, combinatorial
PIN     15   IOF4  COMB       ;I/O, combinatorial
NODE    9    RF2              ;buried node
:
/IOF2 = IO * I1               ;equation for IOF2
/RF2 = { /IOF2 }              ;define buried node /RF2 as /Q of IOF2's register
IOF4 = /RF2 * I1              ;equation with feedback from buried node RF2
```

Output with I/O and /Q (Dual) Feedback

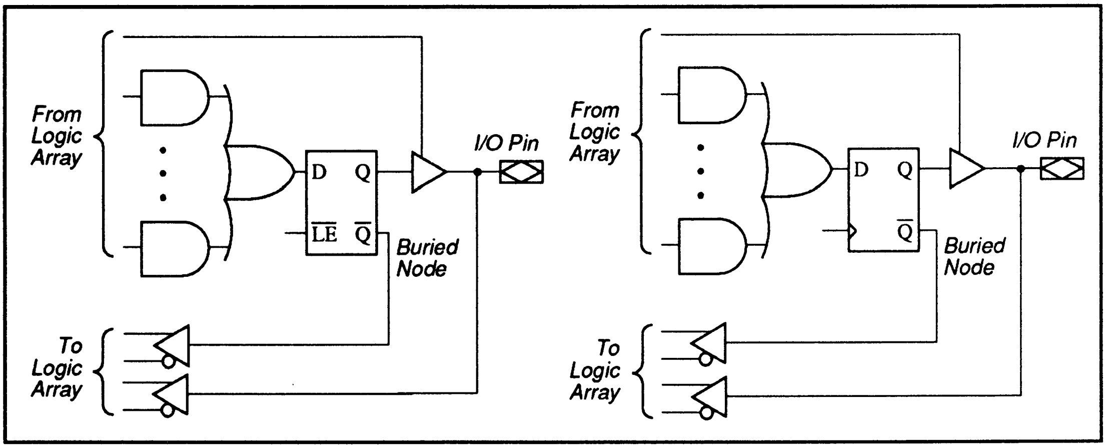
The language syntax for combinatorial and registered or latched output with both the I/O and the /Q register output feeding back to the array is shown below.

Devices Supported: Devices that have output with both I/O and /Q configuration are the 29M16 and 29MA16.

Note: This configuration is only possible in macrocells with dual feedback capability.



Combinatorial Output with I/O and /Q Feedback (Dual Feedback)



Registered/Latched Output with I/O and /Q Feedback (Dual Feedback)

Syntax

```
Pin Statement(s)   :
                   PIN  I/O_pin_location  I/O_pin_name   <I/O_storage_type*>
                   :
                   NODE  Buried_node_location  Buried_node_name < Node_storage_type*>
                   :
Equation(s)        I/O_pin_name = Boolean expression
                   /Buried_node_name = {  I/O_pin_name**  }
                   :
                   < Output equation(s) using either I/O_pin_name or buried_node_name,
                         or both, as feedback(s)>
```

* *Use the following table for the appropriate storage types.*

Storage element	*<I/O_storage_type>*	*<Node_storage_type>*
D Flip-flop	*COMB or REG*	*REG*

** *I/O_pin_name inside curly brackets must use the same polarity as defined on the left side of the output equation.*

Example

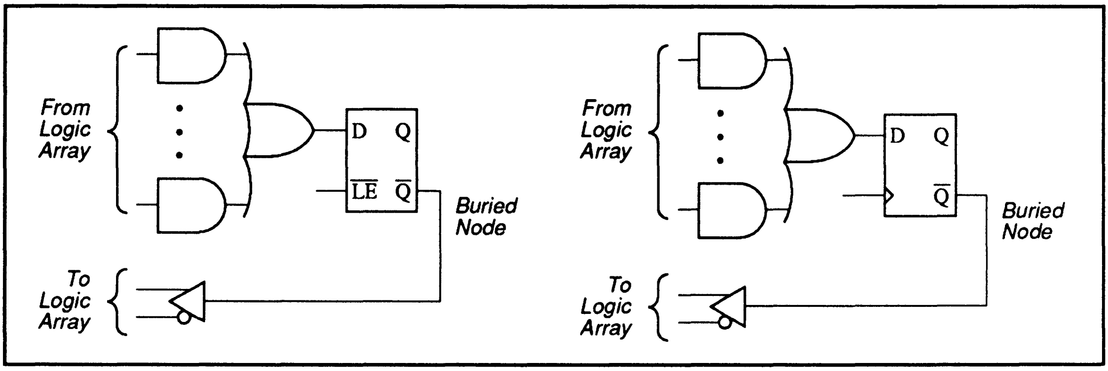
Registered output with I/O and /Q feedback

```
CHIP EXAMPLE PALCE29M16
:
PIN     2    I0     COMB        ;input
PIN     14   I1     COMB        ;input
PIN     3    IOF0   COMB        ;I/O, combinatorial
PIN     15   IOF4   REG         ;I/O, registered
PIN     16   IOF5   REG         ;I/O, registered
NODE    3    RF0                ;buried node, of pin 3's register
:
/IOF0 = I0 * /I1                ;equation for combinatorial I/O /IOF0
/RF0 = { /IOF0 }                ;define /RF0 as /Q output of IOF0
IOF4 = IOF0 * /I0               ;equation with I/O feedback IOF0
IOF5 = /RF0 * I1                ;equation with buried node feedback /RF0
```

Buried Register with /Q Feedback

To use the /Q feedback of registers or latches that are buried, you must write an equation for the buried node. The language syntax for the buried /Q output feedback is illustrated below.

Devices Supported: Devices that have buried registers with /Q feedback configuration are the 23S8, 29M16, and 29MA16.



Buried Register with /Q Feedback

Syntax

```
Pin Statement(s)   :
                   NODE Buried_node_location      Buried_node_name        REG or LAT
                   :
Equation(s)        /Buried_node_name = Boolean expression
                   :
                   < Output equation(s) using buried_node_name as feedback >
```

Example

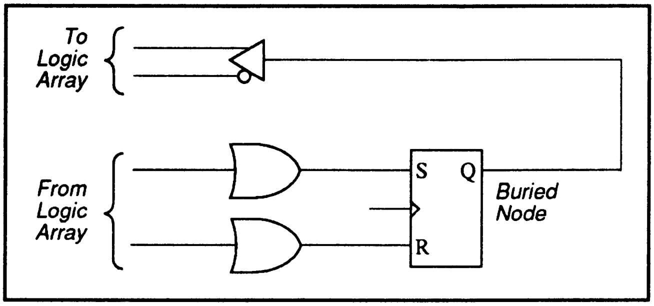
Buried register with /Q feedback

```
CHIP EXAMPLE PALCE29M16
:
PIN     2    I0    COMB       ;input
PIN     14   I1    COM        ;input
PIN     15   IOF4  REG        ;I/O, registered
NODE    6    R1    REG        ;Buried node
:
/R1 = I0 * /I1                ;equation for buried node, R1, with negative polarity
IOF4 = /R1 * I0               ;equation using R1 as feedback
```

Buried Register with Q Feedback

To use the Q feedback of registers that are buried, you must write an output equation for the buried node of the Q output. The language syntax for the buried Q feedback is illustrated on the next page.

Devices Supported: Devices that have buried registers with Q feedback configuration are the 105, 167/168, and 30S16.



Buried Register with Q Feedback

Syntax

```
Pin Statement(s)   :
                   NODE  Buried_node_location     Buried_node_name REG
                   :
Equation(s)        Buried_node_name.S = Boolean expression
                   Buried_node_name.R = Boolean expression
                   :
                   < Output equation(s) using buried_node_name as feedback >
```

Example

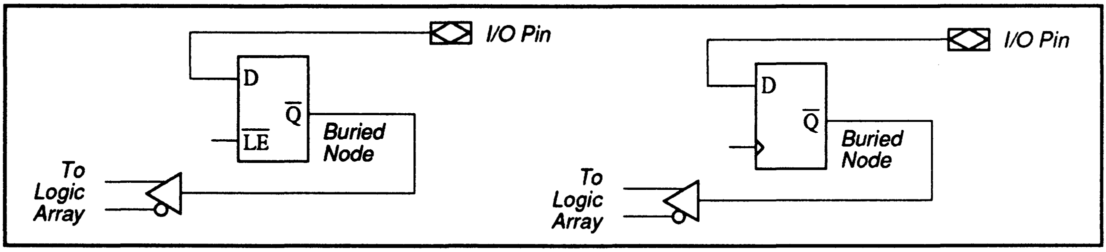
Buried registered with Q feedback

```
CHIP EXAMPLE PLS167
:
PIN      8    I0    COMB       ;input
PIN      7    I1    COMB       ;input
PIN      6    I3    COMB       ;input
PIN      14   P0    REG        ;output, registered
NODE     1    S0    REG        ;Buried node
:
S0.S = I0 * /I1                ;equation for S input of the SR flip-flop with buried
                               ;    node S0
S0.R = I3 * I1                 ;equation for R input of the SR flip-flop with buried
                               ;    node S0
P0 = S0 * I0                   ;equation using S0
```

Registered Input with /Q Output

The output macrocell can also be configured as an input register or latch with /Q output.

Devices Supported: Devices that have registered input with /Q output configuration are the 29M16 and 29MA16.



Registered/Latched Input with /Q Output

Syntax

```
Pin Statement(s)   :
                   PIN  I/O_pin_location          I/O_pin_name         Storage_type
                   :
                   NODE Buried_node_location      Buried_node_name     REG
Equation(s)        < Output equation(s) using buried_node_name as feedback* >
*   Pin name cannot appear on the left side of any equation.
```

Example

Registered input with /Q input

```
:
CHIP EXAMPLE PALCE29M16
:
PIN     2    I0     COMB        ;input
PIN     14   I1     COMB        ;input
PIN     7    I02    REG         ;I/O, registered
PIN     15   IOF4   COMB        ;I/O, combinatorial
NODE    7    R2     REG         ;Buried node
:
IOF4 = /R2 * I0 + R2 * I1       ;equation using R2
```

11.3.8.2 Non-Programmable Feedback

Some programmable devices have feedbacks that are not programmable, that is, their feedback paths are fixed. There are two types of non-programmable feedback.

- Combinatorial or registered output with I/O feedback
- Registered output with /Q feedback

Combinatorial or Registered Output with I/O Feedback

If an output is configured to be combinatorial or registered, the I/O can be used to feedback directly into the logic array. To use the combinatorial output as feedback, no special language syntax is necessary. Simply write the output equation for that I/O pin, then use the I/O pin name in the Boolean expression, as required for other output equations.

Devices Supported: Devices that have combinatorial output with I/O feedback are the 16R8, 16RA8, 16V8, 18P8, 20R8, 20RA10, 20V8, 20X10/20L10, 22IP6, 22P10, 22V10, 23S8, 24R10, and 24V10.

Registered Output with /Q Feedback

If an output is configured to be registered, the /Q output of the register can be used to feedback directly into the logic array. To use the /Q output as feedback, no special language syntax is necessary. Simply write the output equation for that I/O pin, then use the I/O pin name in the Boolean expression, as required for other output equations.

Devices Supported: Devices that have registered output with /Q feedback are the 16R8, 16V8, 20R8, 20V8, 20X10/20L10, 22V10, 24R10, and 24V10.

11.3.9 PRELOAD CONTROL

There are two types of preload control.

- Supervoltage
- Product term control

11.3.9.1 Supervoltage

The supervoltage preload control allows any arbitrary state value to be loaded into the registers or latches under supervoltage.[8] No special language syntax is required for this type of supervoltage-enabled preload.

Devices Supported: Devices that use supervoltage preload are the 16V8, 16V8HD, 20EG8, 20EV8, 20V8, 22V10, 23S8, 24V10, 26V12, 29M16, 29MA16, 30S16, 32VX10, and 610.

11.3.9.2 Product Term Control

The global preload product term is used to control the preload function. To use the preload product term, you must use the language syntax described below.

[8] Refer to the PAL Device Data Book for details.

Devices Supported: Devices that have both super-voltage enabled and preload product term control are the 29M16 and 29MA16.[9]

Syntax

```
Pin Statement(s)   :
                   NODE  Global_node_location     Global_node_name
                   :
Equation(s)        Global_node_name.PRLD = Boolean expression using one product term
```

Example

Preload control with product term

```
:
CHIP EXAMPLE PALCE29M16
:
PIN     2    I1    COMB         ;input
PIN     11   I2    COMB         ;input
:
NODE    1    GLOBAL             ;internal global node
:
GLOBAL.PRLD = I1 * /I2          ;preload the registers when (I1 * /I2) is true
```

11.3.10 OBSERVABILITY PRODUCT TERM CONTROL

The global observability product term is used to control the observability function. To use this product term, you must use the language syntax described below.

Devices Supported: Devices that have observability product term control are the 23S8, 29M16, 29MA16, and 30S16.[10]

[9] Refer to 11.4 for specific device syntax datasheets, which provide the assigned node location.

[10] Refer to the individual datasheets for the assigned node location.

Syntax

```
Pin Statement(s)  :
                  NODE  Observe_node_location     Observe_node_name
                  :
Equation(s)       Observe_node_name = Boolean expression using one product term
```

Example

```
:
CHIP EXAMPLE PALCE29M16
:
PIN     2    I1     COMB       ;input
PIN     11   I2     COMB       ;input
:
NODE    2    OBSERVE           ;internal global node
:
OBSERVE = I1 * /I2             ;observe buried registers when ( I1 * /I2 ) is true
```

11.3.11 COMPLEMENT ARRAY

Complement arrays are used in PLS devices as extra logic resources. To use a complement array, you must first define the output of the complement array as a node in the pin statement, then write the equation for the opposite output polarity. To use the complement array in an equation, you must use the same polarity as defined in the pin statement. The syntax and example for the complement array are described next.

Devices Supported: Devices that have complement arrays are the 105, 167/168, and 30S16.

Syntax

```
Pin Statement(s)  NODE  Complement_array_node_location  Complement_array_node_name
Equation(s)  Complement_array_node_name* =Boolean expression with one product term
             :
             < Output equation(s) using either Complement_array_node_name** >
```

* *Complement_array_node_name must have the opposite polarity as defined in the node statement.*

** *The complement_array_node_name used in other output equations must have the same polarity as defined in the node statement.*

Example

Complement array

```
CHIP EXAMPLE PLS167
:
PIN      7    I1     COMB         ;input
PIN      6    I2     COMB         ;input
PIN      5    I3     COMB         ;input
PIN      13   Q3     REG          ;registered output
:
NODE     13   /CA                 ;complement array node
:
CA = I1 * /I2                     ;write the equation for CA output with opposite polarity
                                  ;    as defined in the pin statement
Q3.S = /CA * I3                   ;use same polarity, /CA as defined in the pin statement
```

11.3.12 ELECTRONIC SIGNATURE

The electronic signature word contains 64 bits of programmable memory that can contain user-defined data. You can program the signature using the Signature statement, shown next.

Devices Supported: Devices that have electronic signature capability are the 16V8, 20V8, and 26V12.

Syntax

```
Pin declaration segment
                :
                SIGNATURE* = < Base(radix) number > or an alphanumeric character string
```

* *The signature can be specified in any of the following formats.*

Base (Radix) Number	*Syntax*	*Max Number of Digits*
Binary	*#B or #b*	*64*
Decimal (Default)	*#D or #d*	*15*
Hexadecimal	*#H or #h*	*16*
Octal	*#O or #o*	*21*

Example 1

Signature using a base number

SIGNATURE = 14435O

Example 2

Signature using an alphanumeric character string

SIGNATURE = V12_6

11.4 PLD DEVICE SYNTAX DATASHEETS

The information here is divided into datasheets for the following non-standard programmable devices.[11]

- 105
- 167/168
- 16RA8
- 16V8HD
- 20EG8
- 20EV8
- 20RA10
- 22IP6
- 22V10
- 23S8
- 26V12
- 29M16
- 29MA16
- 30S16
- 32VX10
- 610

Each datasheet consists of the following information for a particular device.

- Pin and Node Descriptions
- Block and Macrocell Diagrams
- Special Programming Features

11.4.1 PIN AND NODE DESCRIPTIONS

The pin and node descriptions provide the locations and names for each pin and node in the specified device. This information is needed to program specific features, such as global preset and reset, preload with product term control, observability, and feedback with buried nodes.

[11] Standard devices are not listed here and are discussed only in 11.3. MACH devices are discussed in 11.5.

11.4.2 BLOCK AND MACROCELL DIAGRAM(S)

The block diagram shows the relationship between the macrocells and pins, and the locations of the macrocells. The macrocell diagram(s), if any, show logic and fuse information of each type of macrocell, as well as node locations.

11.4.3 SPECIAL PROGRAMMING FEATURES

These discussions identify the language syntax required to program each special feature not covered in the earlier general language syntax discussion. Information here is organized into subtopics that cover each special programming feature. The corresponding language syntax and an example that illustrates its use are also included.

PLS105: PLS105 / PLSCE105

PIN AND NODE DESCRIPTIONS

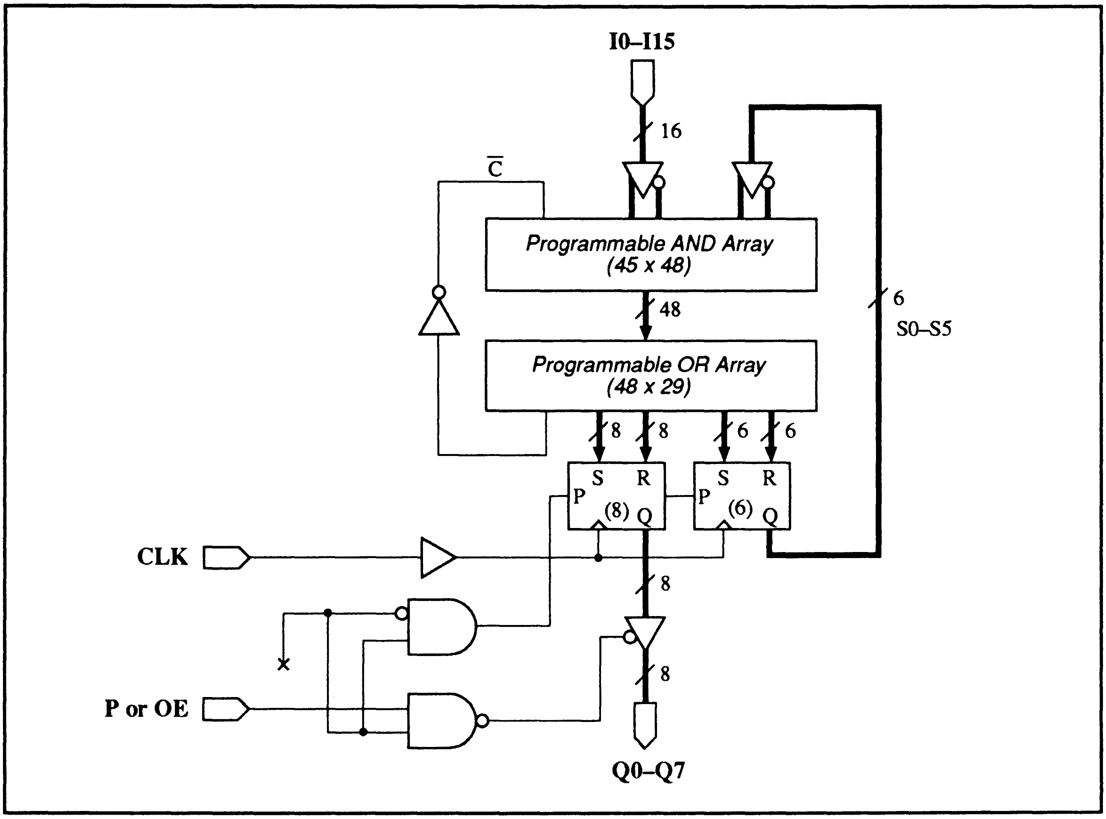
- 28 pins
- 6 buried nodes
- 1 complement array node name

105

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK	–	–	–
2 – 9	I7 – I0	–	–	–
10 – 13	Q7 – Q4	–	–	–
14	GND	–	–	–
15 – 18	Q3 – Q0	–	–	–
19	P or OE	–	–	–
20 – 27	I15 – I8	–	–	–
28	VCC	–	–	–
–	–	1 – 6	S0 – S5	Buried feedback
–	–	7	/C	Complement array

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



105: Block Diagram Showing Pin and Node Locations

SPECIAL PROGRAMMING FEATURES

- Programmable preset/output-enable pin
- State bits

Programmable Preset/Output Enable Pin

Pin 19 controls either preset or output enable for the entire device. You define the global function for pin 19 by writing either a .TRST or a .SETF functional equation for one or more outputs. If you write equations for more than one output, each must contain the same sequence of literals and operators.

105: PLS105 / PLSCE105

To eliminate confusion and reduce errors, it is a good idea to write functional equations for a group of vectors rather than for each pin; the preset or enable function controls all output pins.

Syntax 1

Using an output vector

```
Pin Statement(s)  :
                  PIN  Input_pin_location        Input_pin_name
                  PIN  < Output_pin_number(s) >   Output_vector_name
                  :
```

```
Equation(s)  using Pin 19 to control output enable
                  Output_vector_name.TRST = Input_pin_name
     or      using Pin 19 to control preset
                  Output_vector_name.SETF = Input_pin_name
```

Example 1

Output enable using an output vector

```
:
PIN     19   OE                        ;output-enable control input
PIN     10   13, 15.18  Q[7..0]        ;output vector, Q[7..0]
:
Q[7..0].TRST = OE                      ;pin OE controls the buffers of outputs Q7 to Q0
```

Syntax 2

Using group outputs

```
Pin Statement(s)  :
                  PIN  Input_pin_location    Input_pin_name
                  :
                  GROUP  Group_name    < Output_pin/Node_name(s) >
```

```
Equation(s)  using Pin 19 to control output enable
                  Group_pin_name.TRST = Input_pin_name
     or      using Pin 19 to control preset
                  Group_pin_name.SETF = Input_pin_name
```

Example 2

Preset enable using group outputs

```
:
PIN      19   PRESET                  ;preset input
PIN      18   Q7     REG              ;output
:
PIN      10   Q0     REG              ;output
:
NODE     1    S0                      ;buried node
:
NODE     6           S5               ;buried node
GROUP OUTPUTS
   Q7 Q6  Q5  04  03  02 01 00
   S5 S4  S3  S2  S1  S0              ;group all outputs and buried registers as OUTPUTS
:
OUTPUTS.SETF = PRESET                 ;pin PRESET controls the preset of output registers
                                      ;    Q7 to Q0
```

State Bits

Device 105 has six buried-state registers where bits can be stored: S0 to S5. Do not assign names to the buried-state registers when you want to assign state bits automatically.

167/168: PLS167 / PLSCE167 / PLS168 / PLSCE168

PIN AND NODE DESCRIPTIONS

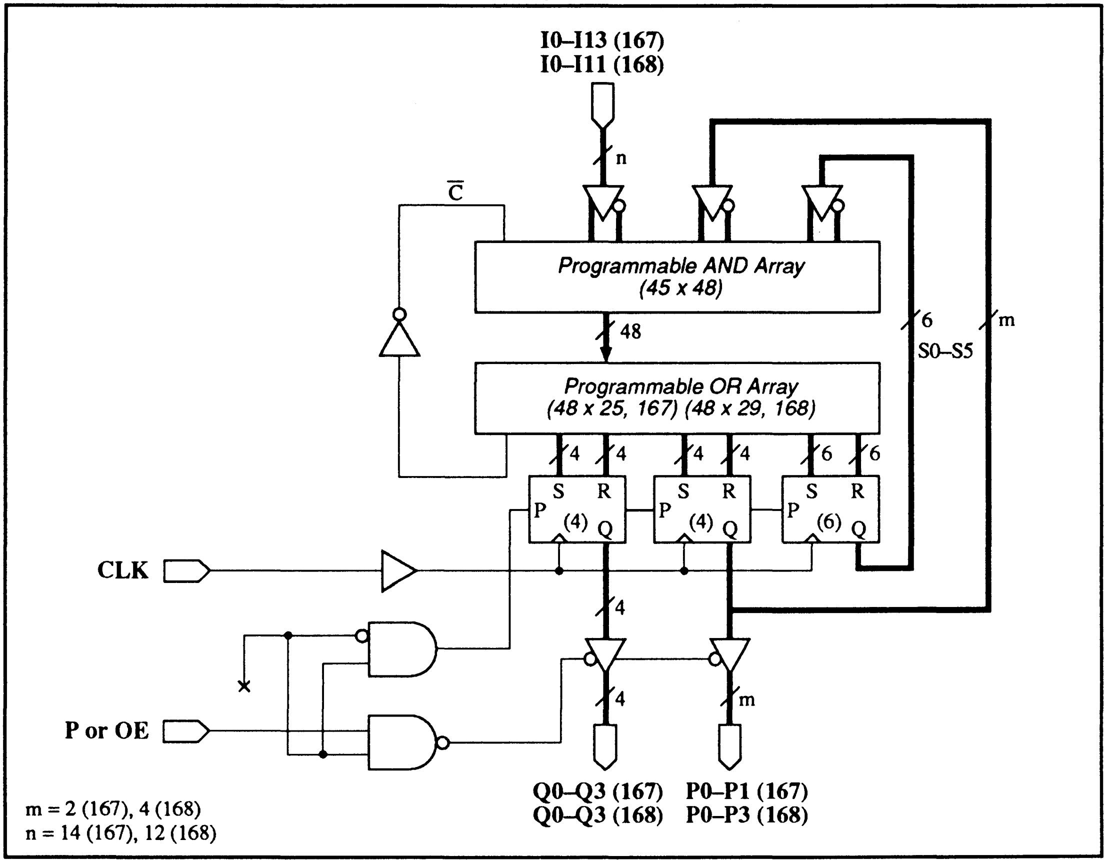
- 24 pins
- 6 buried nodes
- 1 complement array node name

167/168

Pin Location	Pin Name	Node Location	Node Name	Node Descriptions
1	CLK	–	–	–
2 – 8	I6 – I0	–	–	–
9 – 11	Q0 – Q2	–	–	–
12	GND	–	–	–
13	Q3	–	–	–
14 – 15	P0 – P1	–	–	
16	P or OE	–	–	
17 – 23	I13 – I7	–	–	–
24	VCC	1 – 6	S0 – S5	Buried feedbacks
–	–	7	/C	Complement array

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



167/168: Block Diagram Showing Pin and Node Locations

SPECIAL PROGRAMMING FEATURES

- Programmable preset/output-enable pin
- State bits

Programmable Preset/Output-Enable Pin

Pin 16, P or OE, controls either the preset or output enable globally. You define the function for pin 16 by writing either a .TRST or a .SETF functional equation for one or more outputs. If you write equations for more than one output, each must contain the same sequence of literals and operators.

To eliminate confusion and reduce errors, it's a good idea to write functional equations for a group of vectors instead of for each pin; since the preset or enable function controls all output pins.

Syntax 1

Programmable output enable/preset using an output vector

```
Pin Statement(s)  :
                  PIN   Input_pin_location          Input_pin_name
                  PIN  < Output_vector_number(s) >  Output_vector_name   Storage_type
                  :
Equation(s)   using Pin 16 to control output enable
                  Output_vector_name.TRST = Input_pin_name
     or       using Pin 16 to control preset
                  Output_vector_name.SETF = Input_pin_name
```

Example 1

Output enable using an output vector

```
:
PIN     16   OE                       ;output-enable input
PIN     14   15, 9..11  Q[0..7]       ;output vector, Q[0..7]
:
Q[0..7].TRST = OE                     ;pin OE controls the outputs Q0 to Q7
```

Syntax 2

Programmable output enable/preset using group outputs

```
Pin Statement(s)  :
                  PIN  Input_pin_location        Input_pin_name
                  :
                  GROUP  Group_name    < Output_pin/node_name(s) >

Equation(s)  using Pin 16 to control output enable
                  Group_pin_name.TRST = Input_pin_name
     or      using Pin 16 to control preset
                  Group_pin_name.SETF = Input_pin_name
```

Example 2

Preset using group outputs

```
:
PIN      16    PRESET               ;preset input
PIN      9     Q0     REG           ;output
:
PIN      11    Q2     REG           ;output
PIN      14    P0     REG           ;output
PIN      15    P1     REG           ;output
:
NODE     1     S0     REG           ;buried node
:
NODE     6     S5     REG           ;buried node
GROUP OUTPUTS
    P0 P1  02  01  00
    S5 S4  S3  S2  S1  S0           ;group all outputs and buried registers as OUTPUTS
:
OUTPUTS.SETF = PRESET               ;pin PRESET controls the preset of output registers Q2
                                    ;    to Q0, P0 and P1, and buried registers S5 to S0
```

State Bits

Devices 167/168 have six buried state registers for storing bits: S0 to S5. Do not assign names to the buried-state registers when you want to assign state bits automatically.

16RA8: PAL16RA8

PIN AND NODE DESCRIPTIONS

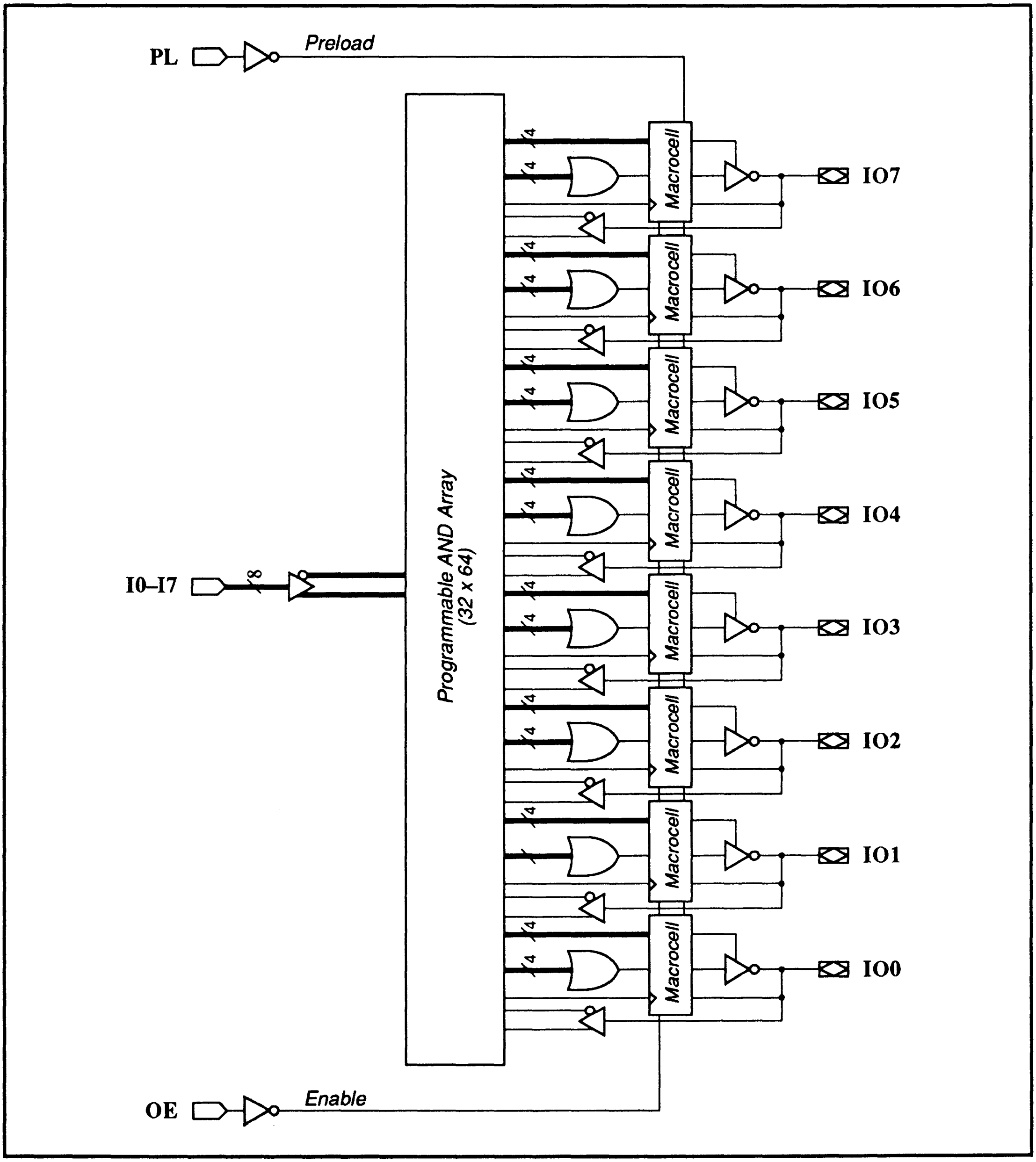
- 20 pins
- No internal nodes

16RA8

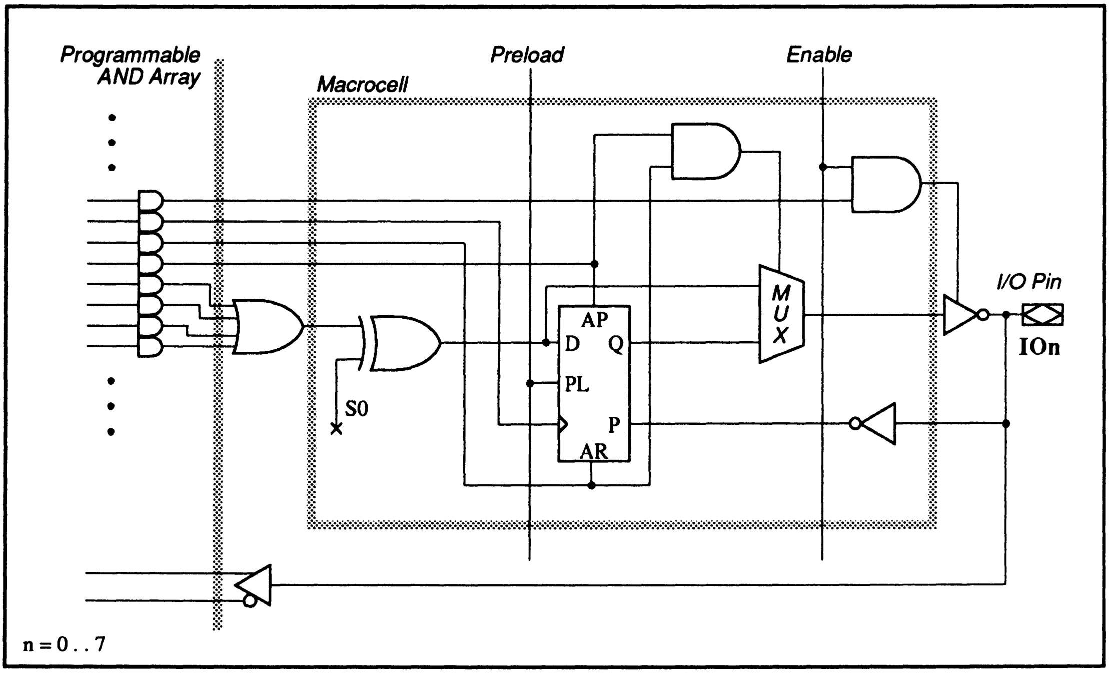
PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	PL	–	–	–
2 – 9	I0 – I7	–	–	–
10	GND	–	–	–
11	OE	–	–	–
12 – 19	IO0 – IO7	–	–	–
20	VCC	–	–	–

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



16RA8: Block Diagram Showing Pin and Macrocell Locations



16RA8: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Common external and individual product term output-enable control
- External preload control

Common External and Individual Product Term Output-Enable Control

Each three-state output buffer is controlled by both the common external output-enable pin and an individual product term. If the individual product term is used, an output buffer is enabled **only if the external output-enable pin is low and the output-enable product term is true.**

To program the product term, you write a .TRST equation for the corresponding output. Otherwise, just control the output buffer using the external output-enable pin. To program the individual product term use the syntax below.

Syntax

```
Pin Statement(s)  :
                  PIN  I/O_pin_location  Output_pin_name  Storage_type
                  :
Equations)        :
                  I/O_pin_name .TRST = Boolean expression with one product term
```

Example

```
:
PIN     11   OE                   ;output enable input
PIN     2    I0                   ;input
PIN     3    I1                   ;input
PIN     12   I00   COMB           ;output, combinatorial
:
I00.TRST = I0 * I1                ;output buffer I00 is only enabled if OE is LOW and
                                  ;    (I0*I1) is HIGH
```

External Preload Control

Register preload is controlled by a TTL-level signal through an external preload pin, pin 1. No special language syntax is required.

16V8HD: PAL16V8HD

PIN AND NODE DESCRIPTIONS

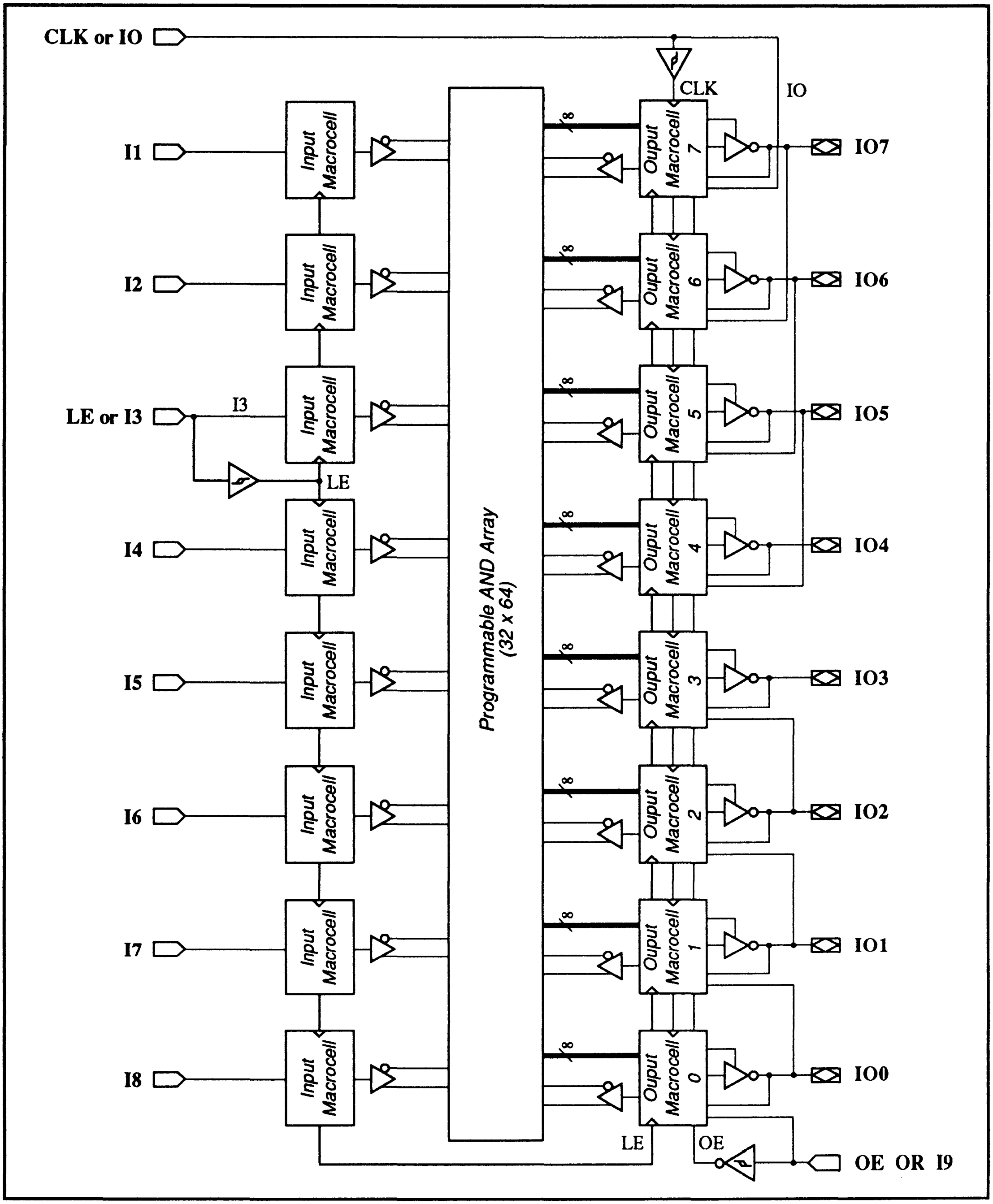
- * 24 Pins
- * 16 Buried Nodes

16V8HD

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK or I0	–	–	–
2 – 3	I1 – I2	1 – 2	IBN1 – IBN2	Buried nodes for input latches
4	LE or I3	3	IBN3	Buried nodes for input latches
5 – 9	I4 – I8	4 – 8	IBN4 – IBN8	Buried nodes for input latches
10	OE or I9	–	–	–
11	GND	–	–	–
12	VCC	–	–	–
13 – 16	IO0 – IO3	9 – 12	IOBN0 – IOBN3	Buried nodes for feedback latches
17	GND	–	–	–
18	VCC	–	–	–
19 – 20	IO4 – IO5	13 – 14	IOBN4 – IOBN5	Buried nodes for feedback latches
21	GND	–	–	–
22 – 23	IO6 – IO7	15 – 16	IOBN6 – IOBN7	Buried nodes for feedback latches
24	VCC	–	–	–

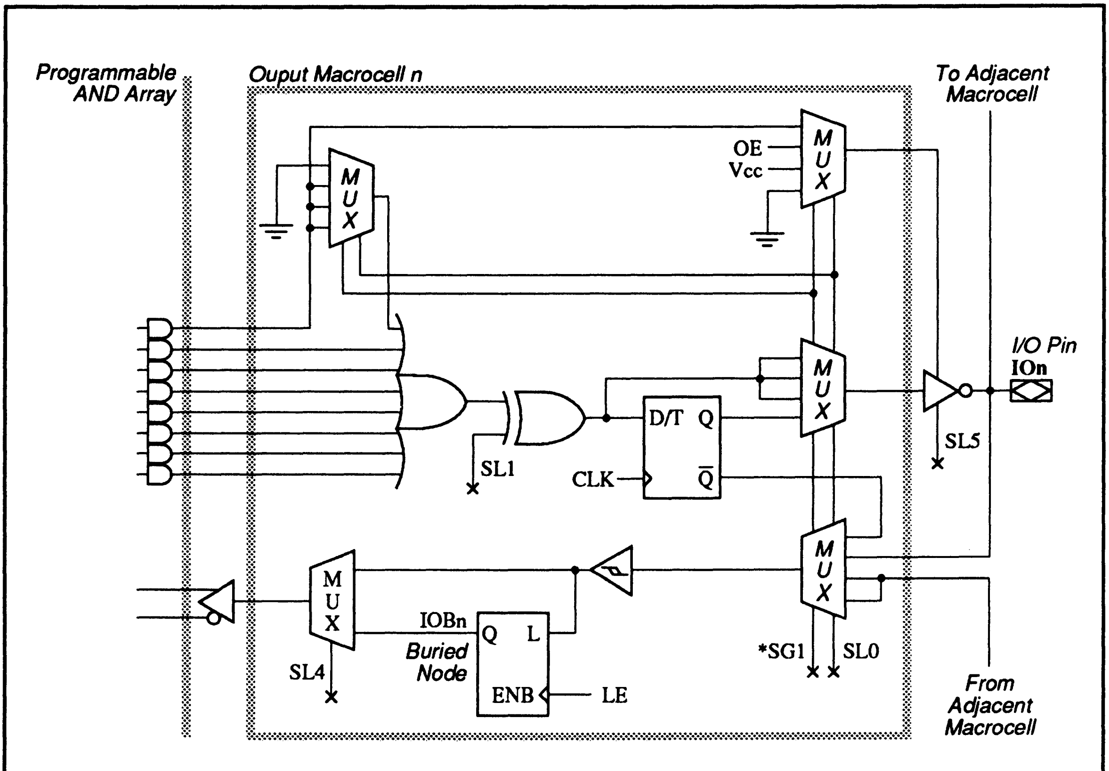
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



16V8HD: Block Diagram

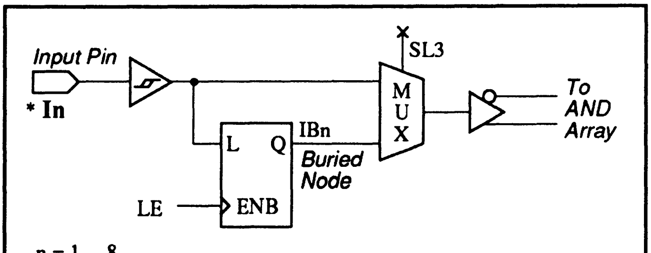
16V8HD: PAL16V8HD



n = 0 . . 7

* In macrocells MC0 and MC7, SG1 is replaced by $\overline{SG0}$ on the feedback multipexer

16V8HD: Output Macrocell



n = 1 . . 8

* I3 can also be used as dedicated latch input enable

16V8HD: Input Macrocell

SPECIAL PROGRAMMING FEATURES

- Latch and clock controls
- Output with latched feedback
- Feedback with latched input
- Input latch
- Output buffer with open collector output

Latch and Clock Controls

The clocks of all output registers are controlled by a dedicated clock input, pin 1, which can also be used as an input. The latch enable inputs of all input and feedback latches are controlled by a dedicated latch input, pin 4, which can also be used as an input. To use pin 1 as clock control, write a .CLKF equation for any I/O pin. To use pin 4 as latch control, write a .CLKF equation for any latch node name, as shown below.

Syntax 1

For clock control

```
Pin Statement(s)  :
                  PIN  1  Clock_input_name
                  PIN  I/O_pin_number    I/O_pin_name    REG
                  :
Equation(s)       I/O_pin_name.CLKF = Clock_input_name
```

Syntax 2

For latch control

```
Pin Statement(s)  PIN  4  Latch_enable_name
                  Node  Buried_node_number    Buried_node_name    LAT
                  :
Equation(s)       I/O_pin_name.CLKF = Latch_enable_name
```

Note: If you do not include any .CLKF equations, then the clock pin is routed to all registers by default and the latch enable pin is routed to all latches by default.

Note: If you write no .CLKF equations, the clock pin is routed to all registers by default and the latch enable pin is routed to all latches by default.

16V8HD: PAL16V8HD

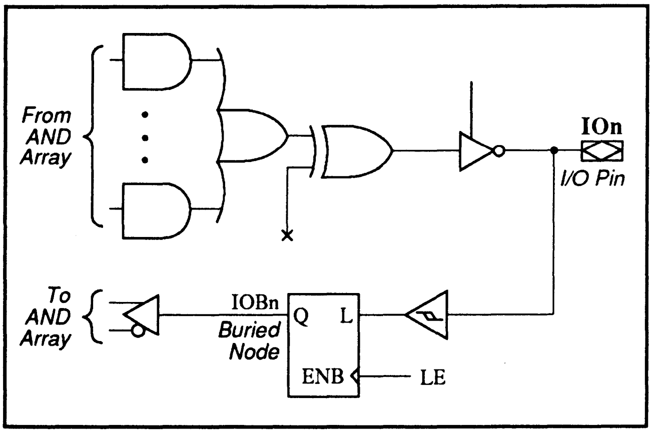
Example

Latch and clock control

```
:
PIN     1     Clock                 ;clock input
PIN     4     LE                    ;latch enable input
PIN     13    IO0    REG            ;output, registered
:
NODE    9     IOBN0 LAT             ;buried node of feedback latch IO0
:
IO0.CLKF = Clock                    ;assign clock input
IOBN0.CLKF = LE                     ;assign latch enable input
```

Output with Latched Feedback

For each output macrocell, the feedback from each I/O can be programmed to be a latched input that feeds back to the AND array. Each output macrocell can be configured to have either combinatorial or registered output with latched feedback. The PALASM syntax for both cases is shown below.



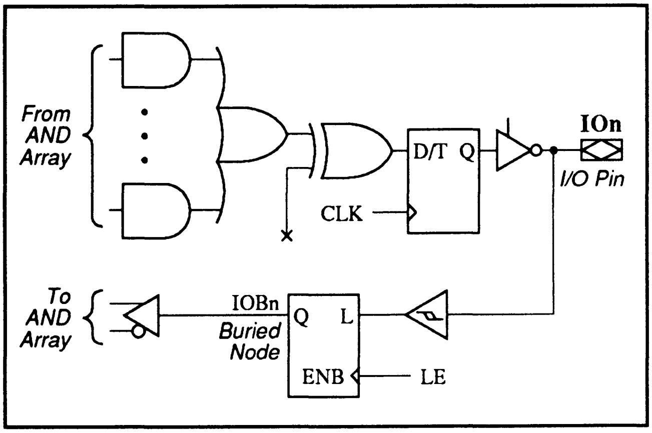
16V8HD: Combinatorial Output with Latched Feedback

Syntax

For combinatorial output with latched feedback

```
Pin Statement(s)  :
                  PIN  I/O_pin_number    I/O_pin_name    COMB
                  :
                  NODE  Buried_node_number    Buried_node_name    LAT
                  :
Equation(s)       I/O_pin_name = Boolean expression
                  Buried_node_name = I/O_pin_name*
                  :
                  < Output equation(s) using Buried_node_name as feedback >
```

* *I/O_pin_name must use the same polarity as defined in the pin statement.*



16V8HD: Registered Output with Latched Feedback

16V8HD: PAL16V8HD

Syntax

For registered output with latched feedback

Pin Statement(s)

```
PIN  I/O_pin_number    I/O_pin_name    REG
:
NODE  Buried_node_number    Buried_node_name    LAT
:
```

Equation(s)

for D flip-flop

```
I/O_pin_name = Boolean expression
```

for T flip-flop

```
I/O_pin_name.T = Boolean expression
:
Buried_node_name = I/O_pin_name*
:
< Output equation(s) using Buried_node_name as feedback >
```

* *I/O_pin_name must use the same polarity as defined in the pin statement.*

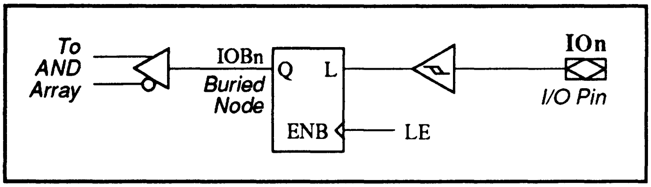
Example

Output with latched feedback

```
:
PIN     2     I1                  ;input, combinatorial
PIN     3     I2                  ;input, combinatorial
PIN     13    IO0    REG          ;I/O, registered
PIN     14    /IO1   COMB         ;I/O, combinatorial
PIN     15    IO2    COMB         ;I/O, combinatorial
:
NODE    9     IOBN0  LAT          ;buried node, feedback latch
NODE    10    IOBN1  LAT          ;buried node, feedback latch
:
IO0 = I1 * I2                     ;output equation for IO0
IO1 = I1 * /I2                    ;output equation for IO1
IOBN0 = IO0                       ;assign node IOBN0 to latch the feedback from IO0
IOBN1 = /IO1                      ;assign node IOBN1 to latch the feedback from /IO1
```

Feedback with Latched Input

Each output macrocell can be configured to be used just as an input, with no output. The PALASM syntax for this configuration is shown below.



16V8HD: Feedback with Latched Input Only

Syntax

Pin Statement(s)	`PIN  I/O_pin_number    I/O_pin_name`
	`:`
	`NODE Buried_node_number    Buried_node_name    LAT`
	`:`

Equation(s)	`Buried_node_name = I/O_pin_name*`
	`:`
	`< Output equation(s) using Buried_node_name as input >`

** I/O_pin_name must use the same polarity as defined in the pin statement.*

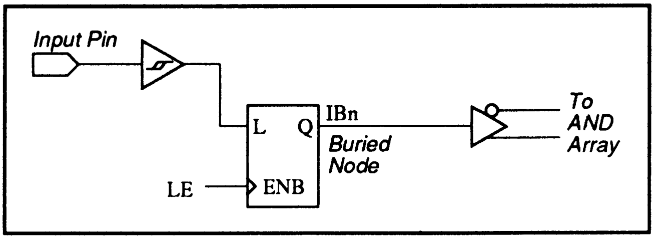
Example

```
:
PIN     2    I1                ;input, combinatorial
PIN     3    I2                ;input, combinatorial
PIN     13   IO0               ;input only
PIN     15   IO2   COMB        ;I/O, combinatorial
:
NODE    9    IOBN0 LAT         ;buried node, feedback latch
:
IOBN0 = IO0                    :assign node IOBN0 to IO0
IO2 = I1 * /I2 * /IOBN0        ;output equation for IO2 using IOBN0 as input
```

16V8HD: PAL16V8HD

Input Latch

Each input macrocell can be configured as a latched input. The PALASM syntax for this configuration is shown below.



16V8HD: Input Latch

Syntax

Pin Statement(s)	`PIN Input_pin_number Input_pin_name`
	`:`
	`NODE Buried_node_number Buried_node_name LAT`

Equation(s)	`Buried_node_name = Input_pin_name*`
	`:`
	`Output equation(s) using Buried_node_name as input`

** Input_pin_name must use the same polarity as defined in the pin statement.*

Example

Input latch

```
:
PIN     2    I1                 ;input, combinatorial
PIN     3    I2                 ;input, combinatorial
PIN     15   I02    COMB        ;I/O, combinatorial
:
NODE    1    IBN1   LAT         ;buried node, latched
:
IBN1 = I1                       ;assign node IBN1 to input I1
I02 = I1 * IBN1                 ;output equation for I02 using node IBN1 as input
```

Output Buffer with Open Collector Output

Each output buffer can be programmed to have an open-collector output by blowing the open-collector fuse, SL5. An 8-bit mask is used to control the eight open-collector fuses of the eight outputs. The MSB of the 8-bit mask controls output IO7, while the LSB controls output IO0, shown below.

Outputs	IO7	IO1	IO2	IO3	IO4	IO5	IO6	IO0
Eight-bit Mask	X	X	X	X	X	X	X	X
	MSB							LSB

X = 0 or 1
0 = No Open-collector
1 = Open-collector

If the bit for the corresponding output is set, that open-collector fuse is blown, resulting in an open-collector output. For example, the mask 00000111 means that IO0 to IO2 are open-collector outputs. In order to use the 8-bit mask to program the outputs, use the PALASM syntax below.

Syntax

```
Declaration segment      :
                         COLLECTOR = < Base(radix) number >
```

Base(radix) number	*Syntax*	*Max. No. of Digits*
Binary	*#B or #b*	*8*
Decimal	*#D or #d*	*3*
Hexadecimal	*#H or #h*	*2*
Octal	*#O or #o*	*3*

Example

Output buffer with open-collector output

```
:
;Declaration Segment
:
COLLECTOR = #B10100001          ;Outputs IO0, IO5 and IO7 are programmed to be open-
                                ;    collector outputs.
```

20EG8: PAL10H20EG8 / PAL10020EG8

PIN AND NODE DESCRIPTIONS

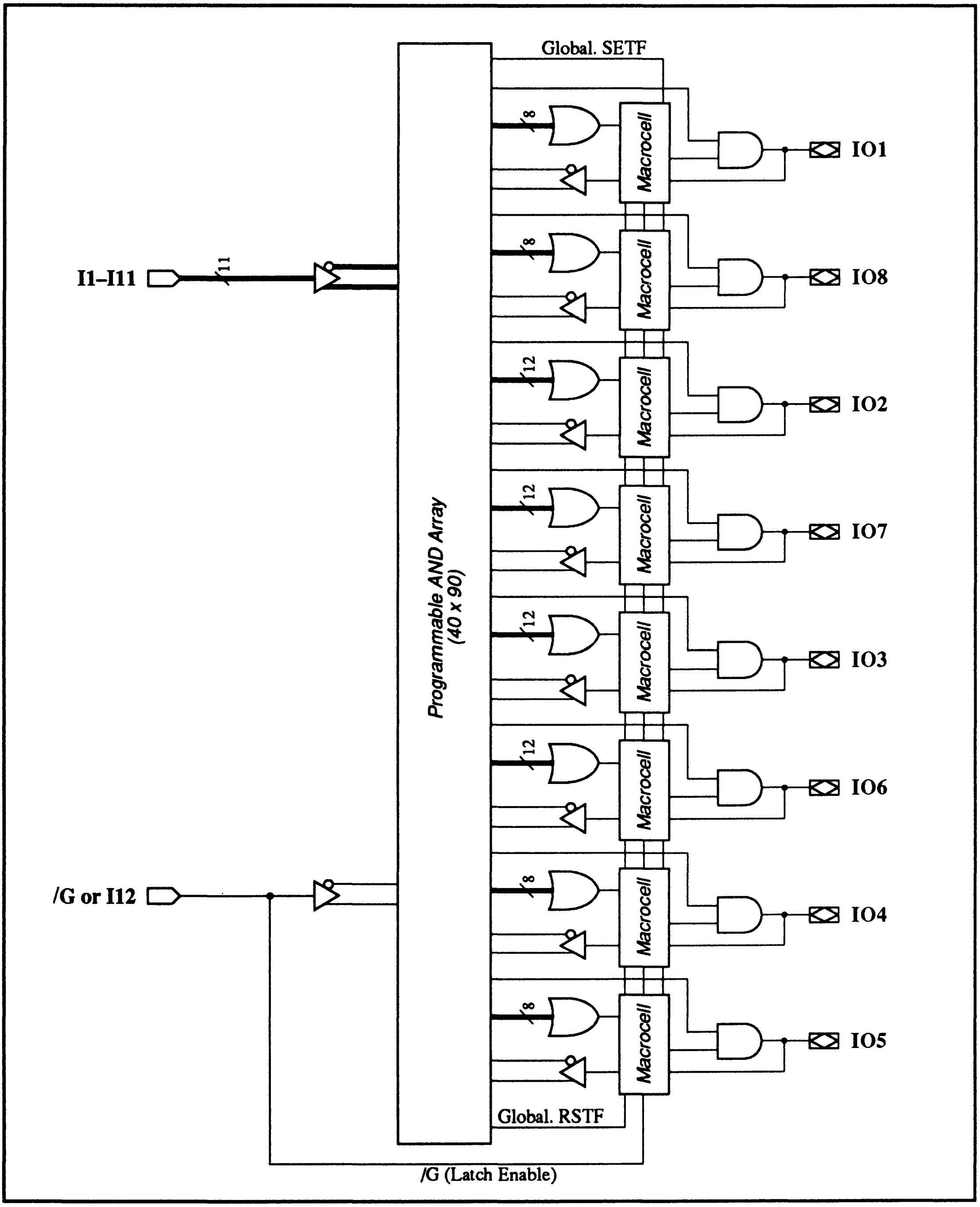
- 24 pins
- 1 global preset/reset node

20EG8

Pin Location	Pin Name	Node Location	Node Name	Node Descriptions
1 – 2	I1 – I2	–	–	–
3	/G or I12	–	–	–
4 – 5	IO1 – I2	–	–	–
6	VCO1	–	–	–
7 – 8	IO3 – I4	–	–	–
9 – 11	I3 – I5	–	–	–
12	VEE	–	–	–
13 – 16	I6 – I9	–	–	–
17 – 18	IO5 – IO6	–	–	–
19	VCO2	–	–	–
20 – 21	IO7 – I8	–	–	–
22 – 23	I10 – I11	–	–	–
24	VCC	–	–	–
–	–	1	GLOBAL	Global preset and reset

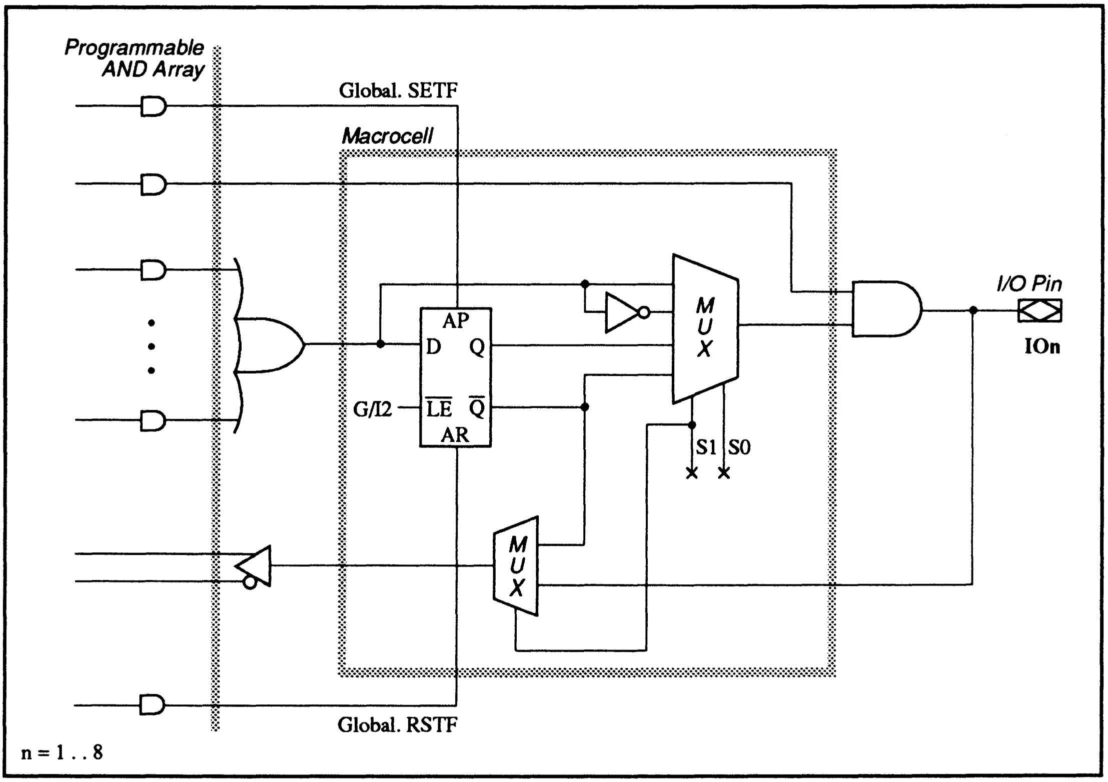
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



20EG8: Block Diagram Showing Pin and Node Locations

20EG8: PAL10H20EG8 / PAL10020EG8



20EG8: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

These devices have no additional programming features other than those discussed under 11.3.

20EV8: PAL10H20EV8 / PAL10020EV8

PIN AND NODE DESCRIPTIONS

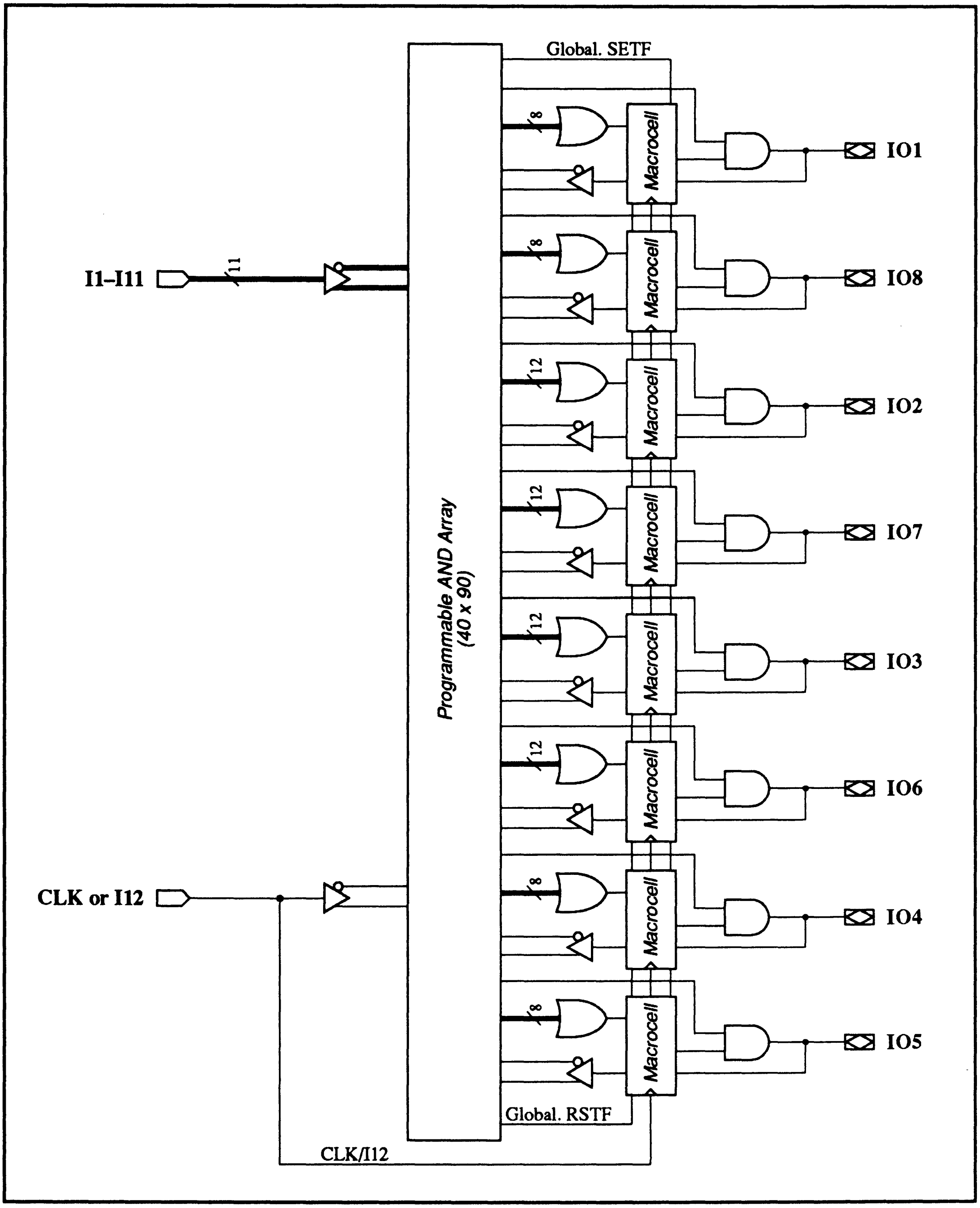
- 24 pins
- 1 global preset/reset node

20EV8

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1 – 2	I1 – I2	–	–	–
3	CLK or I12	–	–	–
4 – 5	IO1 – I2	–	–	–
6	VCO1	–	–	–
7 – 8	IO3 – I4	–	–	–
9 – 11	I3 – I5	–	–	–
12	VEE	–	–	–
13 – 16	I6 – I9	–	–	–
17 – 18	IO5 – IO6	–	–	–
19	VCO2	–	–	–
20 – 21	IO7 – I8	–	–	–
22 – 23	I10 – I11	–	–	–
24	VCC	–	–	–
–	–	1	GLOBAL	Global preset and reset

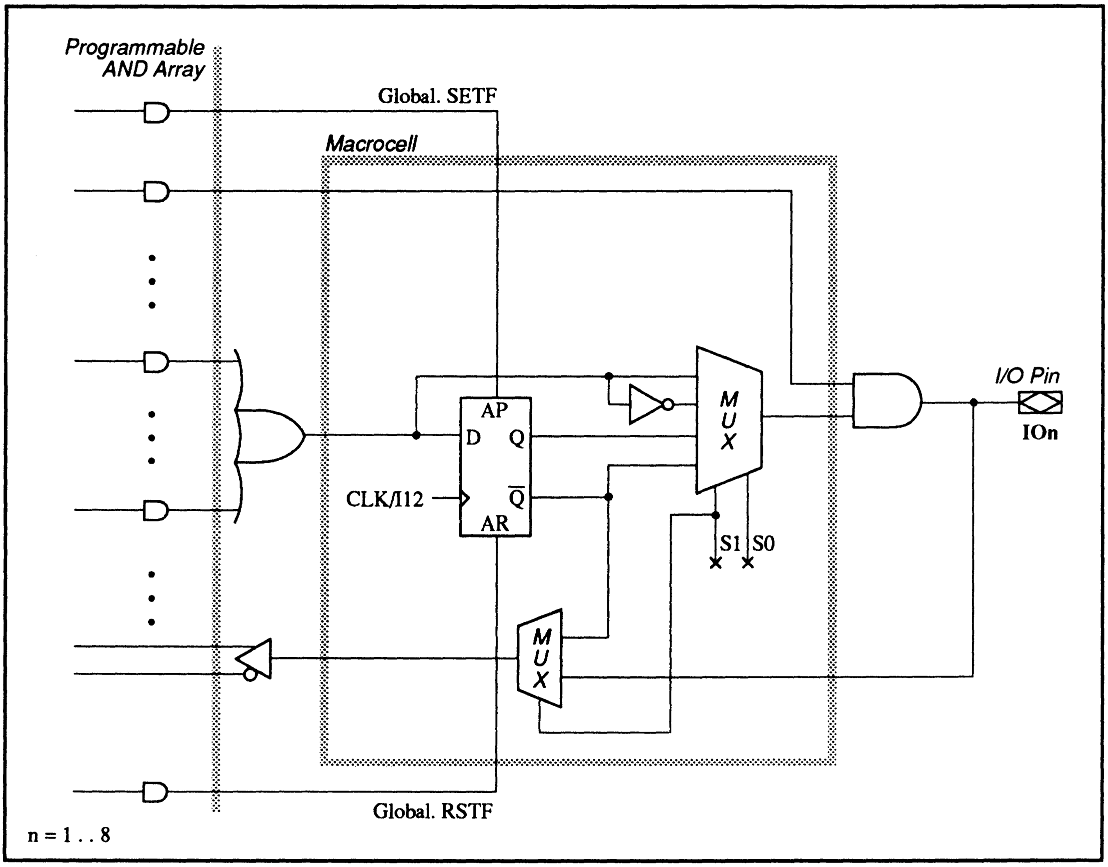
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



20EV8: Block Diagram Showing Pin and Node Locations

20EV8: PAL10H20EV8 / PAL10020EV8



20EV8: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

These devices have no additional programming features other than those discussed under 11.3.

20RA10: PAL20RA10

PIN AND NODE DESCRIPTIONS

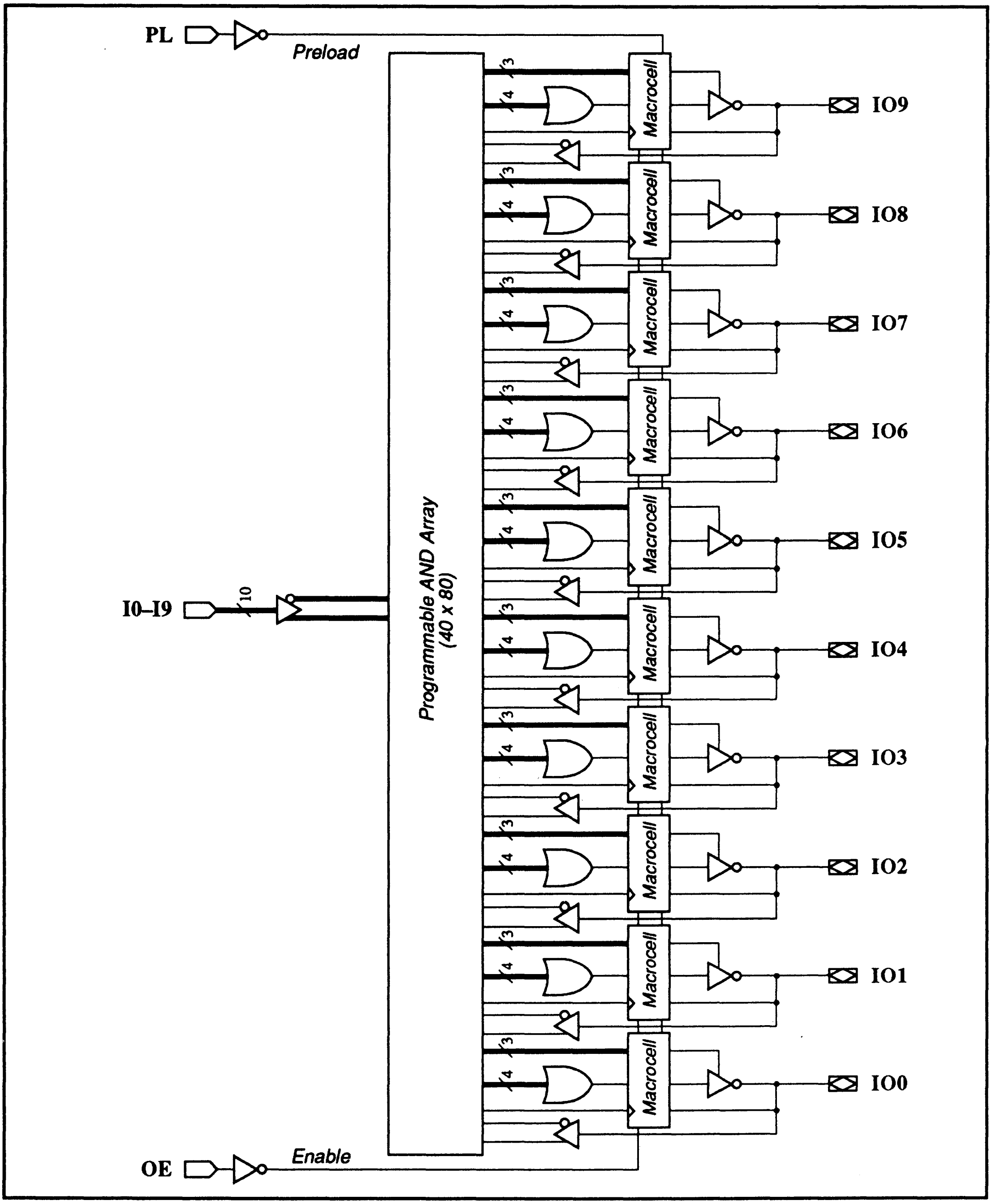
- 20 pins
- No internal nodes

20RA10

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	PL	–	–	–
2 – 11	I0 – I9	–	–	–
12	GND	–	–	–
13	OE	–	–	–
14 – 23	IO0 – IO9	–	–	–
24	VCC	–	–	–

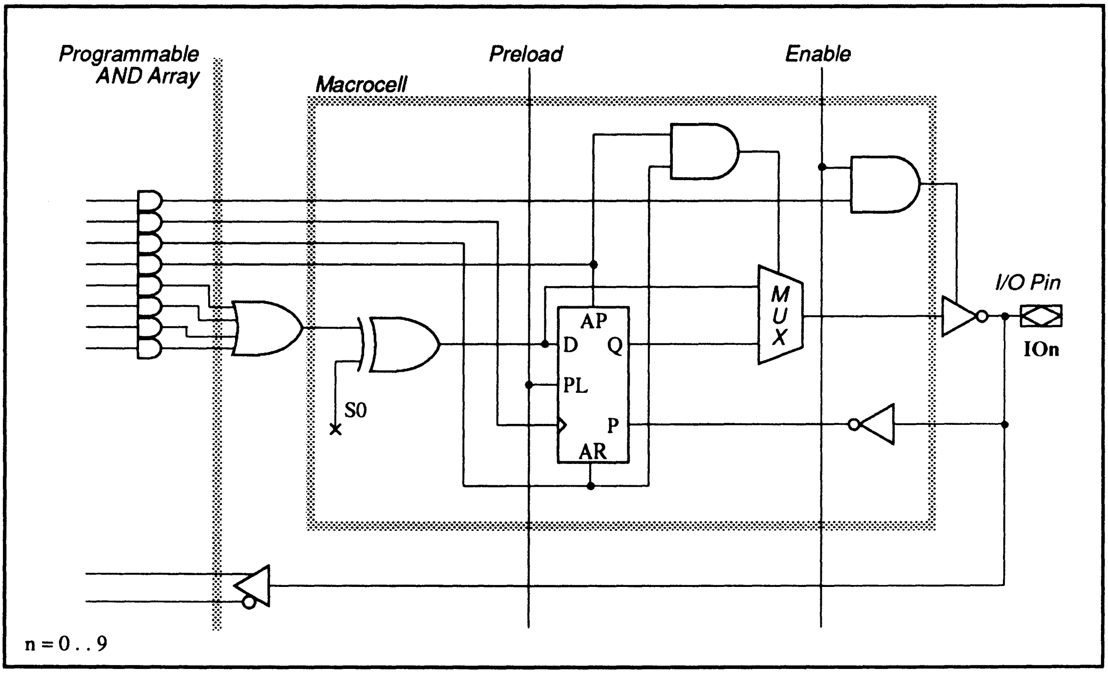
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



20RA10: Block Diagram Showing Pin and Macrocell Locations

20RA10: PAL20RA10



20RA10: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Common external and individual product term output-enable control
- External preload control

Common External and Individual Product Term Output-Enable Control

Each three-state output buffer is controlled by both the common external output-enable pin and an individual product term. If the individual product term is used, an output buffer will be enabled **only if the external output-enable pin is low and the output-enable product term is true.**

To program the product term, you write a .TRST equation for the corresponding output. Otherwise, you can control the output buffer with the external output-enable pin.

Syntax

```
Pin Statement(s)   :
                   PIN  I/O_pin_location  I/O_pin_name Storage_type
                   :
Equation(s)        I/O_pin_name .TRST = Boolean expression with one product term
```

Example

```
:
PIN     13   OE                    ;output-enable input
PIN     2    I0                    ;input
PIN     3    I1                    ;input
PIN     14   I00   COMB            ;output, combinatorial
:
I00.TRST = I0 * I1                 ;output buffer I00 is only enabled if OE is LOW and
                                   ;    (I0*I1) is HIGH
```

External Preload Control

Register preload is controlled by a TTL-level signal through an external preload pin, pin 1. No special language syntax is required.

22IP6: PALCE22IP6

PIN AND NODE DESCRIPTIONS

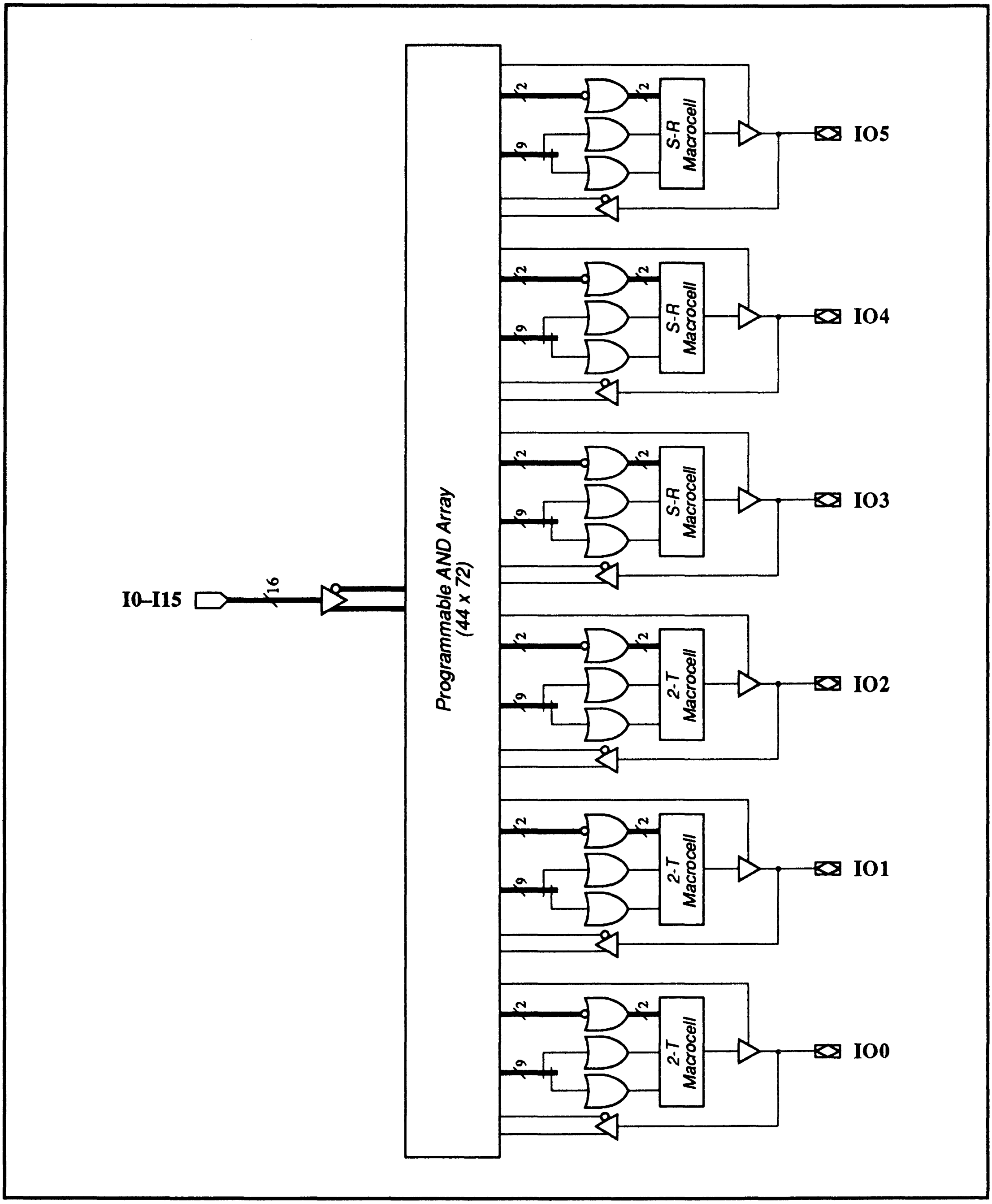
- 24 pins
- No internal nodes

22IP6

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1 – 5	I0 – I4	–	–	–
6	VCC	–	–	–
7 – 12	I5 – I0	–	–	–
13 – 14	I15 – I14	–	–	–
15 – 17	IO5 – IO3	–	–	–
18	GND	–	–	–
19 – 21	IO2 – IO0	–	–	–
22 –24	I13 – I11	–	–	–

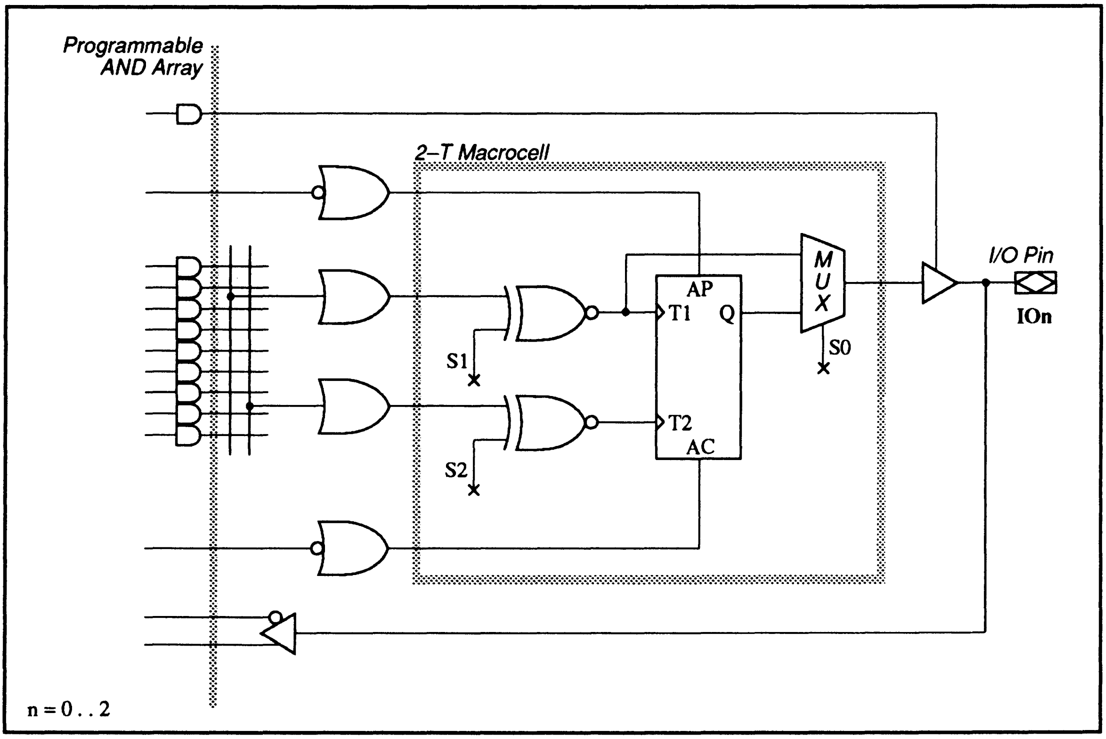
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

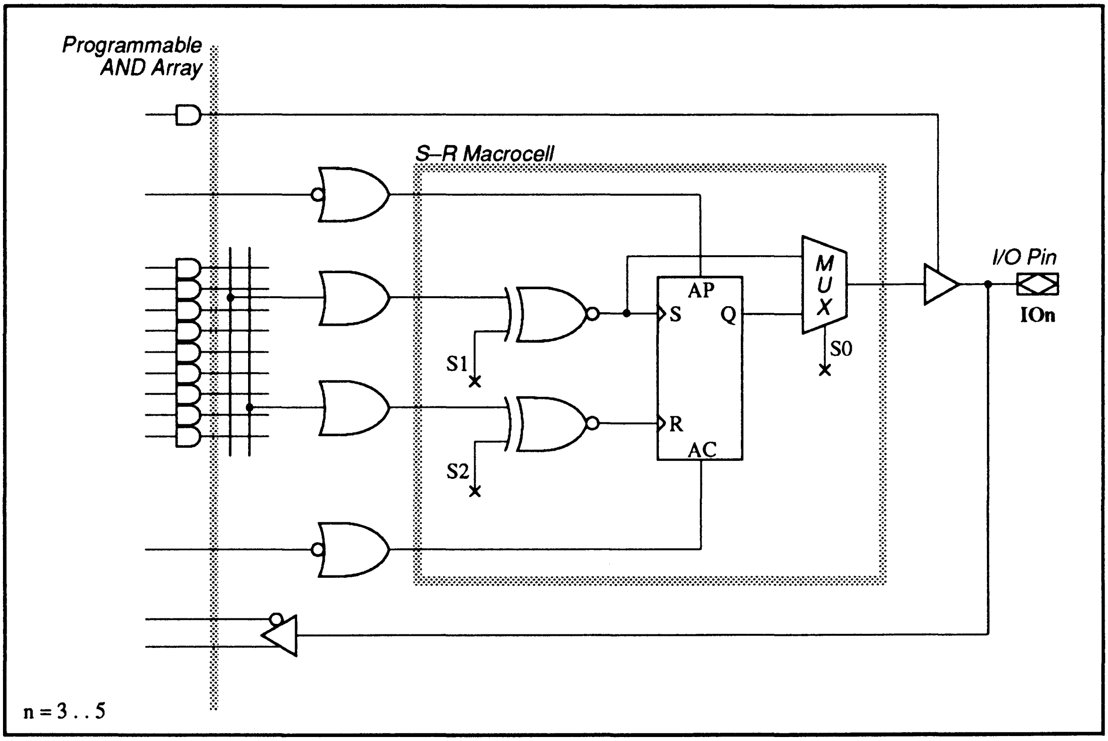


22IP6: Block Diagram Showing Pin and Macrocell Locations

22IP6: PALCE22IP6



22IP6: 2-T Macrocell Diagram



22IP6: S-R Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- 2-T flip-flops with programmable edge-activated input polarity
- SR flip-flops with programmable edge-activated input polarity
- Preset/reset controls with individual sum terms

2-T Flip-Flops

Three of the flip-flops in the 22IP6 are 2-T flip-flops. Each 2-T flip-flop has two independent inputs. Each input can be defined as rising- or falling-edge triggered. The language syntax is shown next.

22IP6: PALCE22IP6

Syntax

```
Pin Statement(s)   :
                   PIN  I/O_pin_location          I/O_pin_name          REG
                   :
Equation(s)        I/O_Pin_name.T1* = Boolean expression
                   I/O_Pin_name.T2* = Boolean expression
```

** Use active-high output_pin_name for rising-edge triggered; use active-low output_pin_name for falling-edge triggered.*

Example

```
:
PIN     1    I0                 ;input
PIN     2    I1                 ;input
PIN     15   I05    REG         ;output, registered
:
 I05.T1 = I1 * I2               ;T1 is asserted when I1 * I2 is high (rising edge)
/I05.T2 = I1 * I2               ;T2 is asserted when I1 * I2 is low (falling edge)
```

SR Flip-Flops

Three of the flip-flops in the 22IP6 are SR flip-flops. Each SR flip-flop has two inputs. Each input can be defined as rising- or falling-edge triggered. The language syntax is shown below.

Syntax

```
Pin Statement(s)   :
                   PIN  I/O_pin_location          I/O_pin_name          REG
                   :
Equation(s)        I/O_Pin_name.S* = Boolean expression
                   I/O_Pin_name.R* = Boolean expression
```

** Use active-high output_pin_name for rising-edge triggered; use active-low output_pin_name for falling-edge triggered.*

Example

```
:
PIN     1    I0                ;input
PIN     2    I1                ;input
PIN     15   I05    REG        ;output, registered
:
 I05.S = I1 * I2               ;S is asserted when I1 * I2 is high (rising edge)
/I05.R = I1 * I2               ;R is asserted when I1 * I2 is low (falling edge)
```

Preset/Reset Control with Individual Sum Term Syntax

On each flip-flop, each preset and reset function is controlled by a programmable sum term. The language syntax is shown below.

Syntax

```
Pin Statement(s)  :
                  PIN  Output_pin_location   I/O_pin_name   Storage_type
                  :
```

```
Equation(s)  for preset
                  Output_Pin_name.SETF = Boolean expression using one sum term
                                         or NAND product term
             for reset
                  Output_Pin_name.RSTF = Boolean expression using one sum term
                                         or NAND product term
```

Example

```
:
PIN     1    I0                ;input
PIN     2    I1                ;input
PIN     15   I05    REG        ;output, registered
:
I05.SETF = I1 + /I2            ;set is asserted when I1 + I2 is true
I05.RSTF = /(I1 * I2)          ;reset is asserted when /(I1* I2) is true
```

22V10: PAL22V10 / AMPAL22V10 / PALCE22V10

PIN AND NODE DESCRIPTIONS

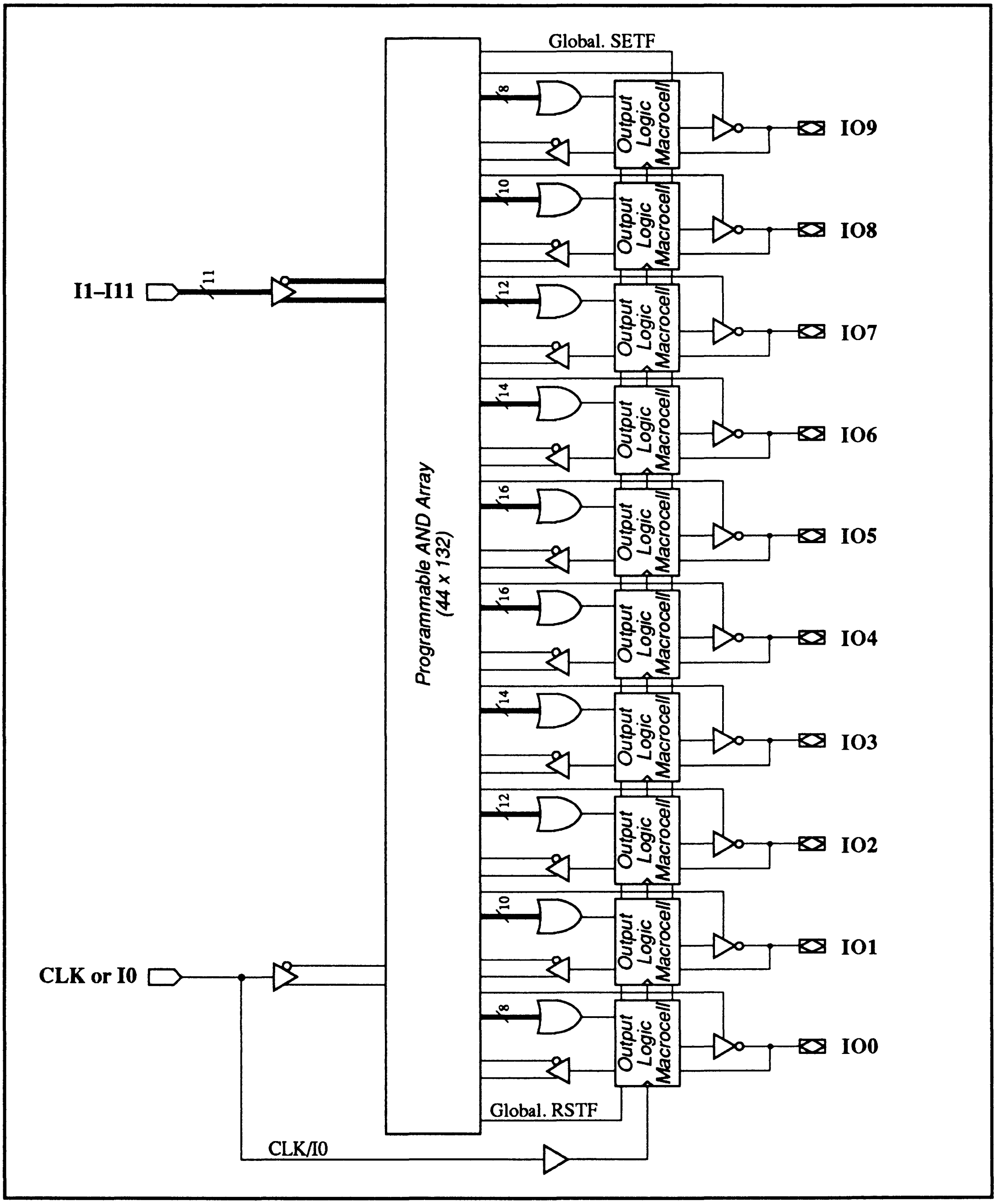
- 24 pins
- 1 global preset and reset node

22V10

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK or I0	–	–	–
2 – 11	I1 – I10	–	–	–
12	GND	–	–	–
13	I1	–	–	–
14 – 23	IO0 – IO9	–	–	–
24	VCC	–	–	–
–	–	1	GLOBAL	Global preset and reset

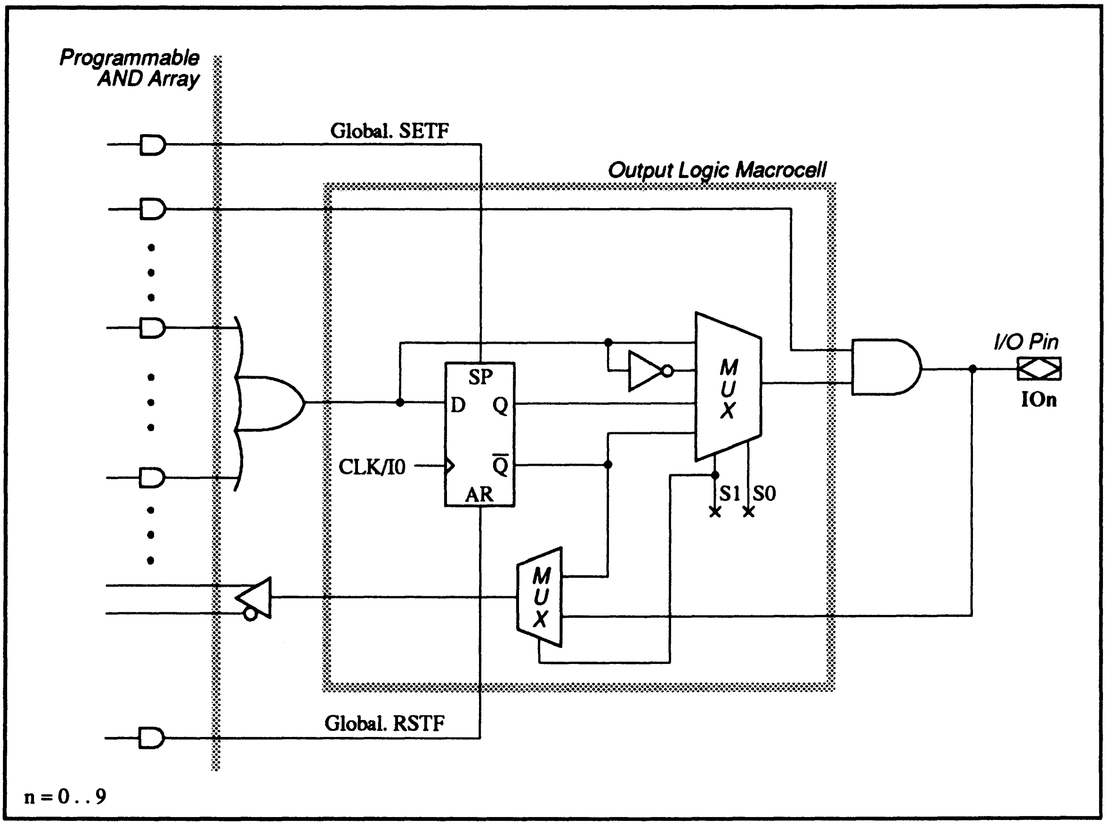
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



22V10: Block Diagram Showing Pin and Node Locations

22V10: PAL22V10 / AMPAL22V10 / PALCE22V10



22V10: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

These devices have no additional programming features other than those discussed under 11.3.

23S8: PALCE23S8

PIN AND NODE DESCRIPTIONS

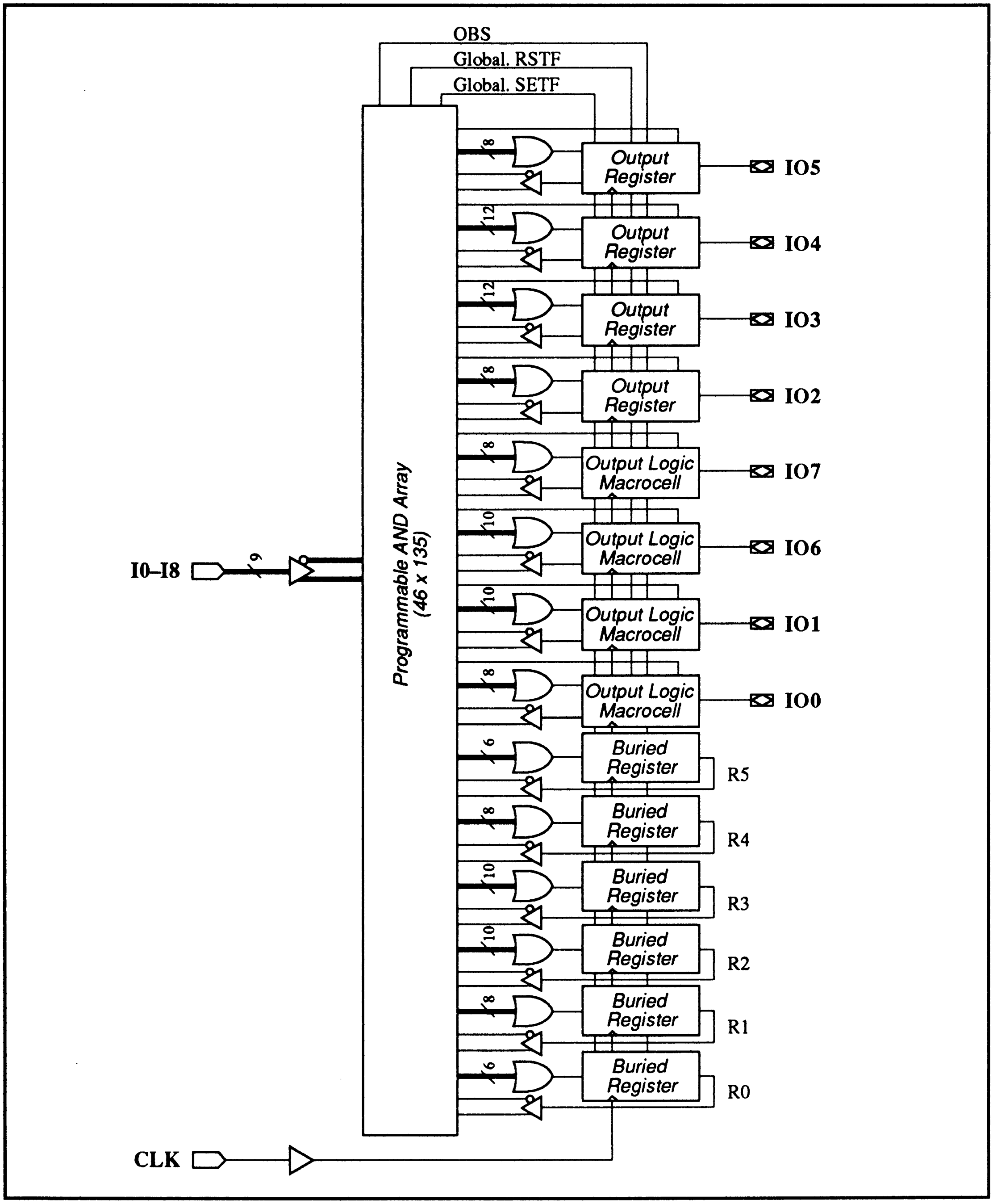
- 20 pins
- 1 global preset and reset node
- 1 observability node
- 10 buried nodes

23S8

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK	–	–	–
2 – 9	I1 – I7	–	–	–
10	GND	–	–	–
11	I8	–	–	–
12 – 13	IO0 – IO1	3 – 4	RIO0 – RIO1	Register feedbacks
14 – 17	IO2 – IO5	–	–	–
18 – 19	IO6 – IO7	11 – 12	RIO6 – RIO7	Register feedbacks
20	VCC	–	–	–
–	–	1	GLOBAL	Global preset and reset
–	–	2	OBS	Observability
–	–	5 – 10	R0 – R5	Buried registers

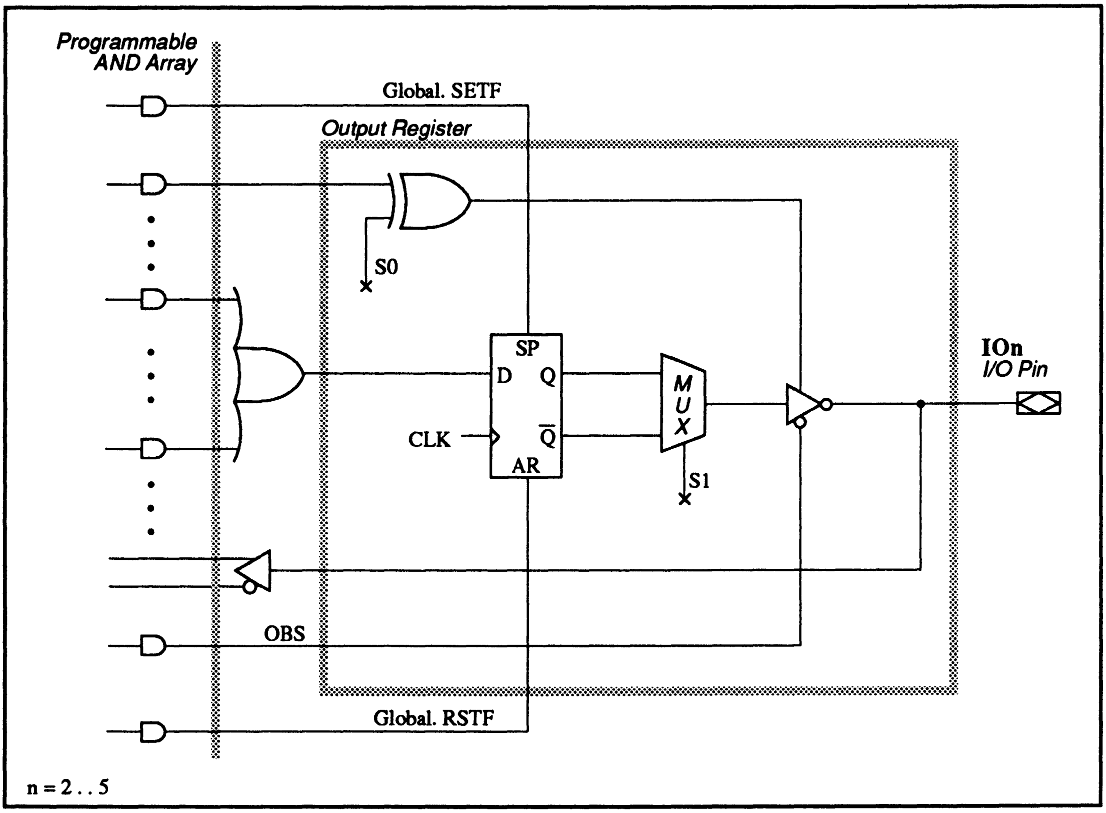
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

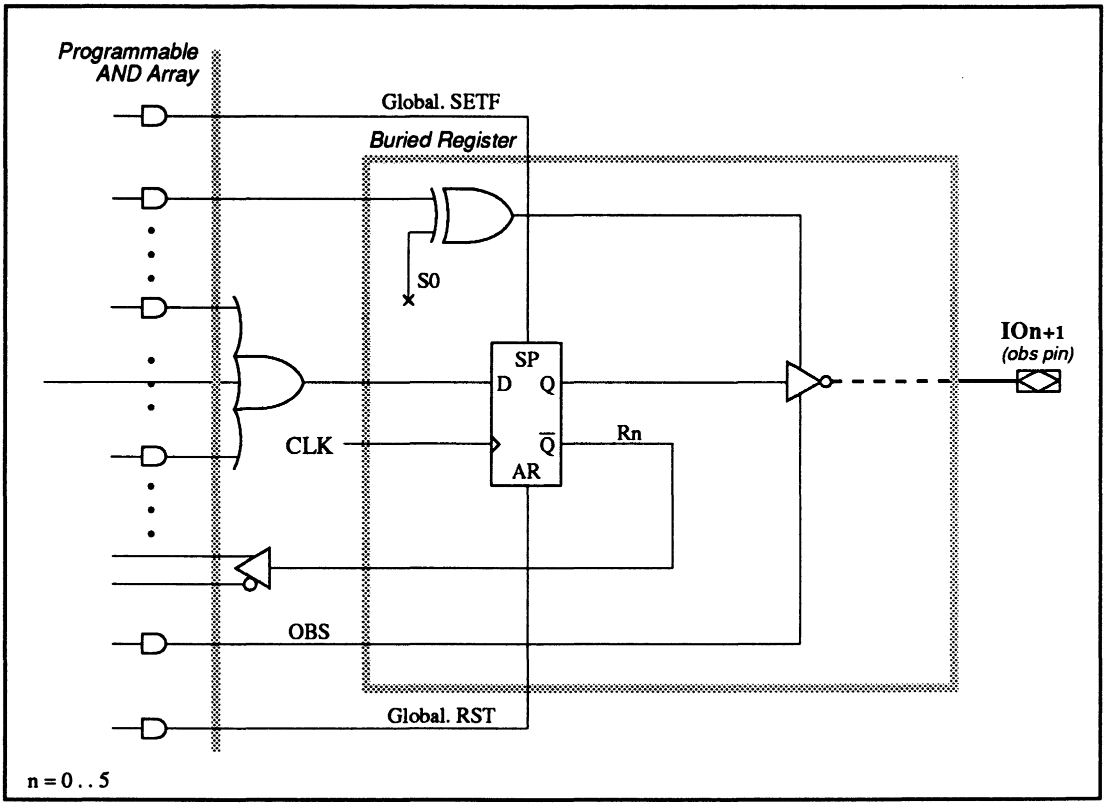


23S8: Block Diagram Showing Pin and Node Locations

23S8: PALCE23S8

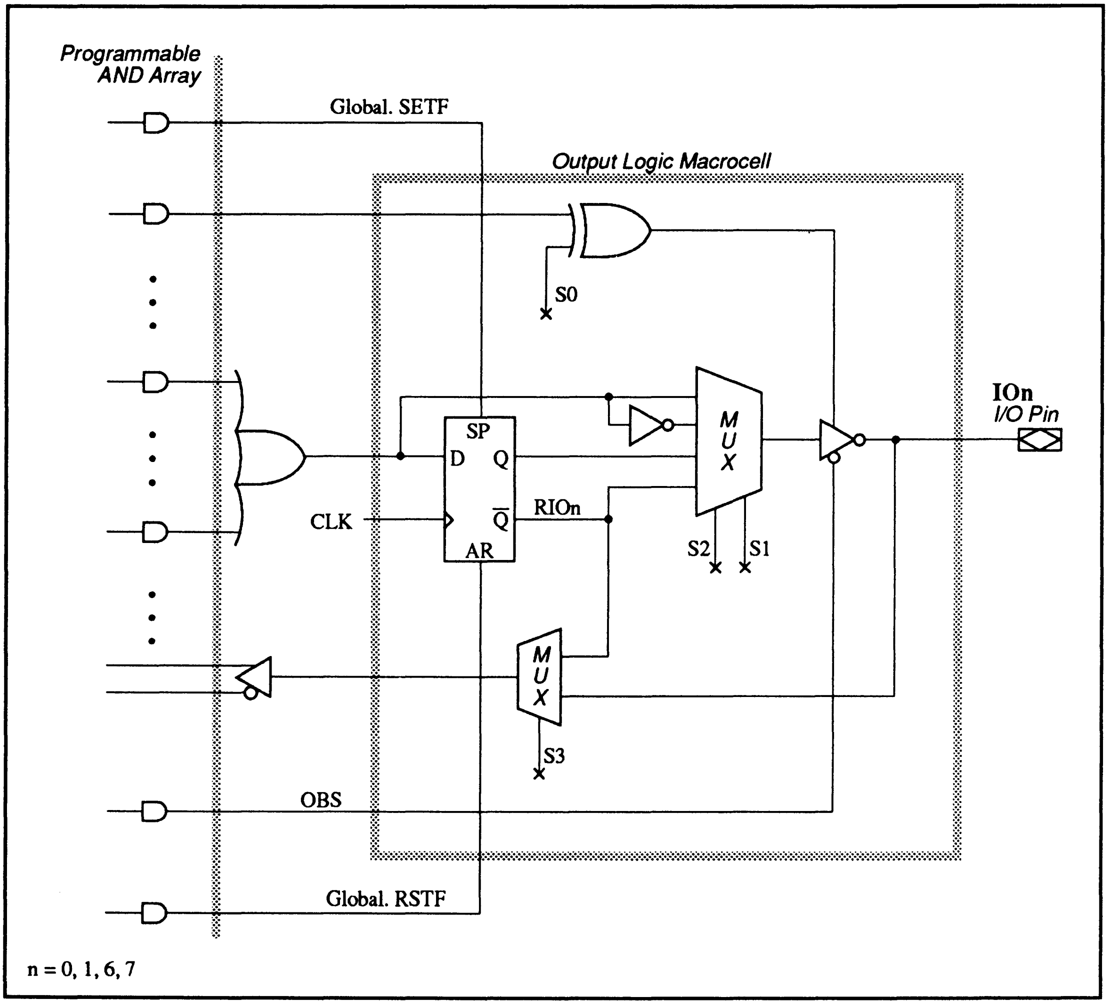


23S8: Output Register Macrocell Diagram



23S8: Buried Register Macrocell Diagram

23S8: PALCE23S8



23S8: Output Logic Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Individual output-enable product term control with programmable polarity
- Macrocells with different configurations

Individual Output-Enable Product Term Control with Programmable Polarity

Each output buffer is controlled by an individual output-enable product term which has programmable polarity. The language syntax is shown below.

Syntax

```
Pin Statement(s)  :
                  PIN Output_pin_location        Output_pin_name         Storage_type
                  :
Equation(s)  for active-high output-enable control
                  Output_pin_name.TRST = Boolean expression using one AND product
                                           term
     or      for active-low output-enable control
                  Output_pin_name.TRST = Boolean expression using one NAND product
                                           term or one SUM term
```

23S8: PALCE23S8

Example

Individual output-enable product term control with programmable polarity

```
:
PIN     2    I0              ;input
PIN     3    I1              ;input
PIN     12   I00    REG      ;output, registered
PIN     13   I01    REG      ;output, registered
PIN     14   I02    REG      ;output, registered
:
I00.TRST = I1 * I2           ;active-high output-enable control using AND product
                             ;    term
I01.TRST = /(I1 * I2)        ;active-low output-enable control using NAND product
                             ;    term
I02.TRST = /I1 + /I2         ;active-low output-enable control using SUM term
```

Macrocells with Different Configurations

The 23S8 has three different types of macrocells: output register, output logic macrocell, and buried register.

Each type of macrocell provides a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	OUTPUT REGISTER	OUTPUT LOGIC MACROCELL	BURIED REGISTER
Non-Programmable Feedback			
Combinatorial Output with I/O feedback	X	N/A	N/A
Registered Output with I/O feedback	X	N/A	N/A
Programmable Feedback			
Output with I/O feedback	N/A	X	N/A
Output with /Q feedback	N/A	X	N/A
Buried register with /Q feedback	N/A	X	X

26V12: PALCE26V12

PIN AND NODE DESCRIPTIONS

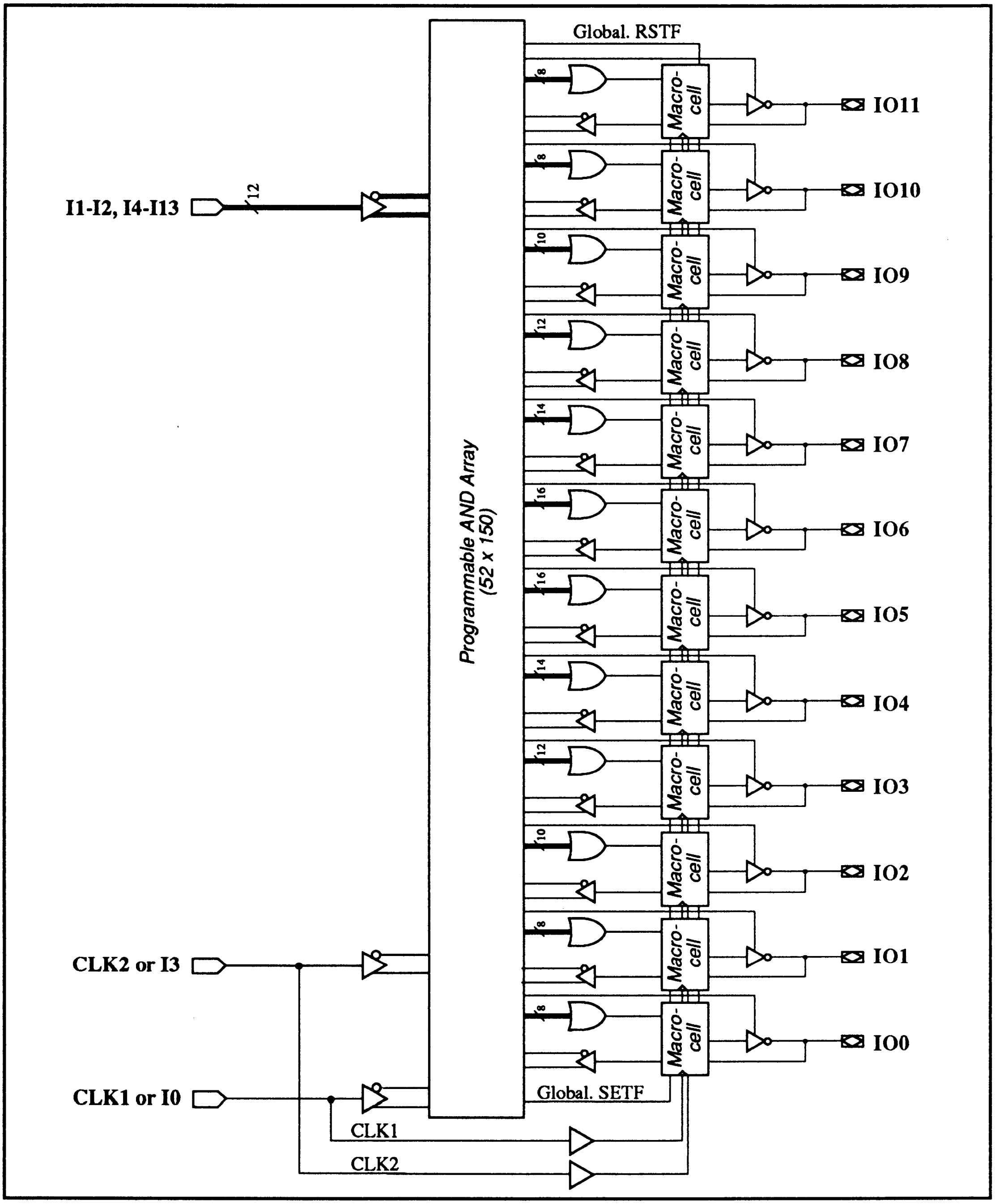
- 28 Pins
- 12 Buried nodes
- 1 Global preset, reset, and preload node

26V12

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK1 or I0	–	–	–
2 – 3	I1 – I2	–	–	–
4	CLK2 or I3	–	–	–
5 – 6	I4 – I5	–	–	–
7	VCC	–	–	–
8 – 14	I6 – I12	–	–	–
15 – 20	IO0 – IO5	2 – 7	R0 – R5	Register feedback
21	GND	–	–	–
22 – 27	IO6 – IO11	8 – 13	R6 – R11	Register feedback
28	I13	–	–	–
–	–	1	GLOBAL	Global preset and reset

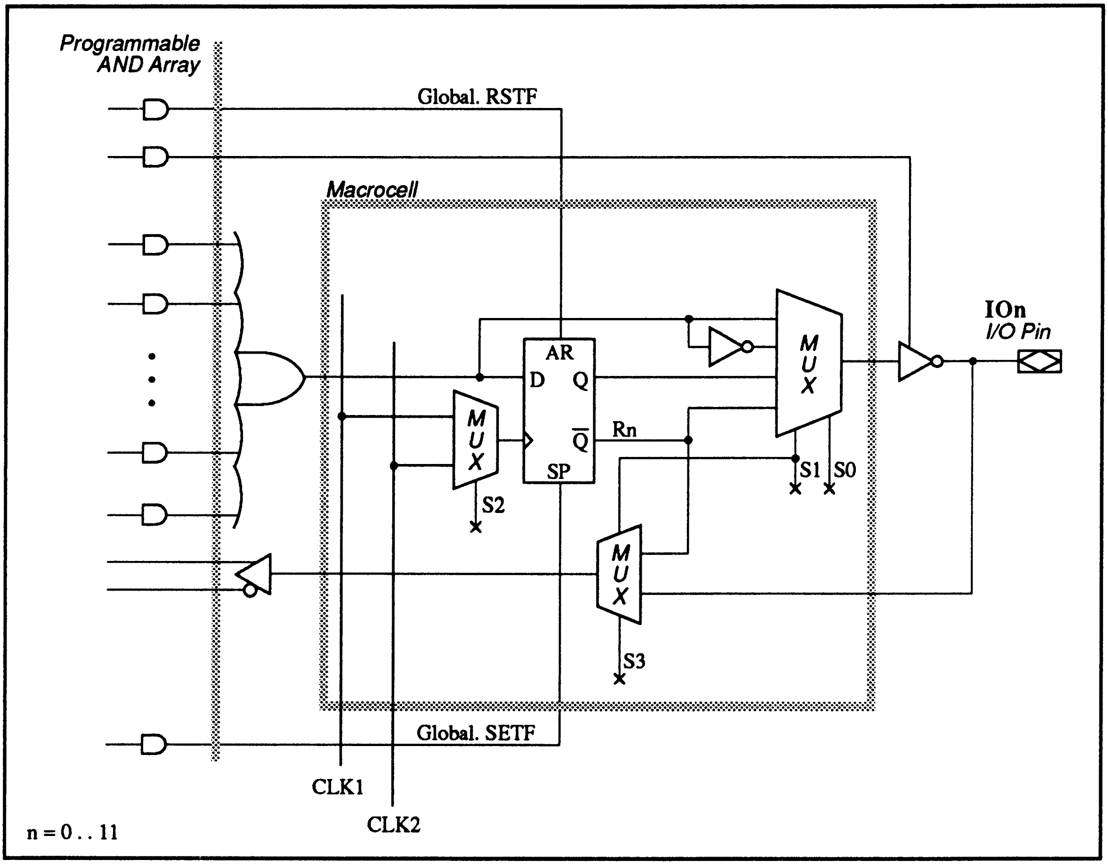
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



26V12: Block Diagram Showing Pin and Node Locations

26V12: PALCE26V12



26V12: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Two external clock pins

Two External Clock Pins

The 26V12 allows you to individually select one of two external clock pins as the the clock input for each flip-flop. You can write a .CLKF functional equation, as shown next.

Syntax

```
Pin Statement(s)  :
                  PIN  Clock_input_pin_location   Clock_input_pin_name
                  PIN  Output_pin_location        Output_pin_name       Storage_type
                  :
Equation(s)       Output_pin_name.CLKF = Clock_input_pin_name
```

Example

```
:
PIN     1    CLK1             ;clock input
PIN     4    CLK2             ;clock input
PIN     14   I2               ;input
PIN     15   I00    REG       ;output, registered
PIN     16   I01    REG       ;output, registered
:
I00. CLKF = CLK1              ;I00 flip-flop uses CLK1 as clock signal
I01. CLKF = CLK2              ;I01 flip-flop uses CLK2 as clock signal
```

Equations written for the 22V10 are mapped to the 26V12 exactly as if they were implemented on the 22V10. This means that feedback is taken from /Q by default, rather than from the I/O pin. If you want feedback from the I/O pin, you must define the corresponding buried node in the pin statement portion of the design file. However, you should not write any equations for that node. If you use the pin name on the right side of an equation, feedback will be routed from the I/O pin rather than from /Q.

29M16: PALCE29M16

PIN AND NODE DESCRIPTIONS

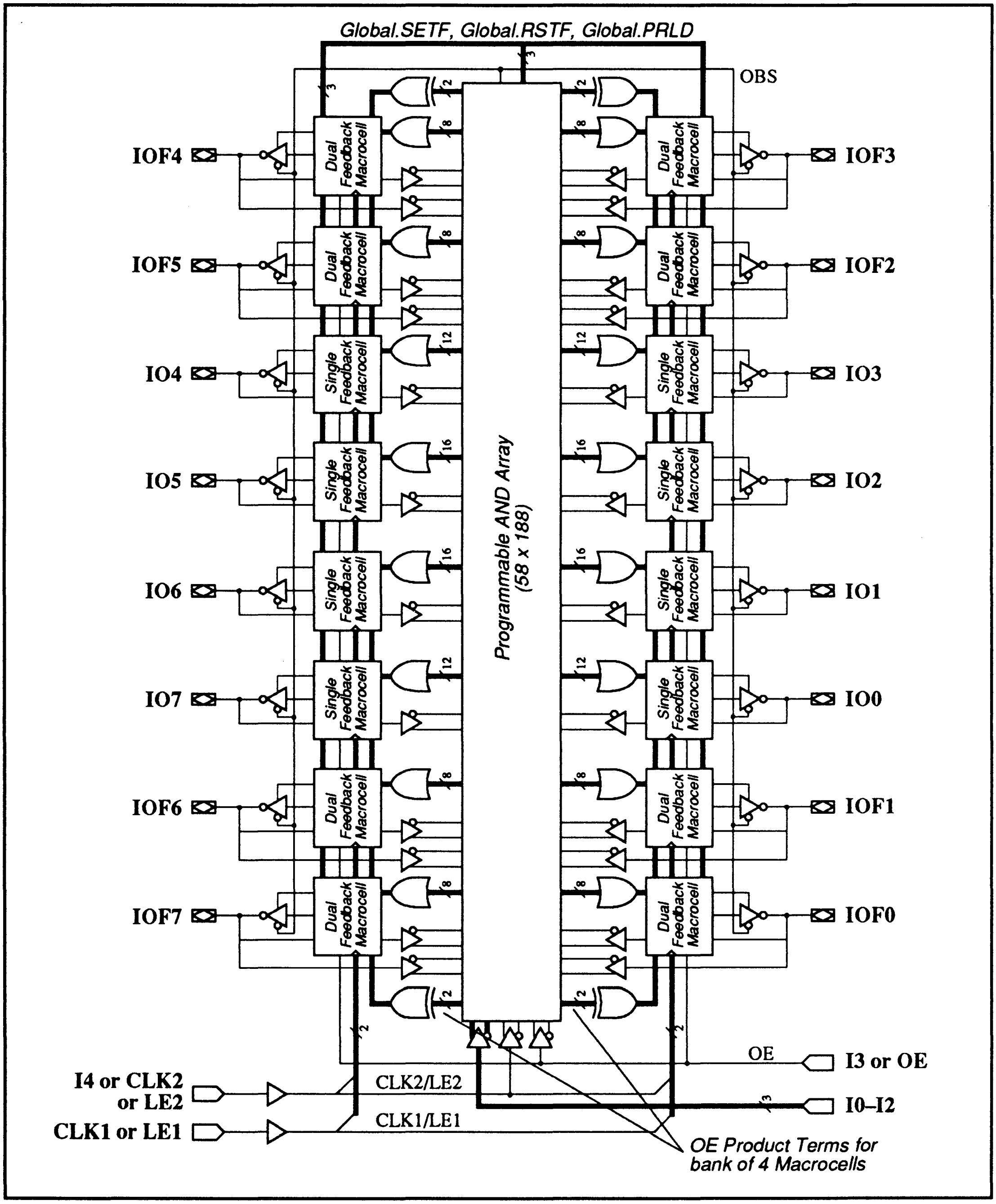
- 24 pins
- 16 buried nodes
- 1 global preset, reset, and preload node
- 1 observability node

29M16

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK1 or LE	–	–	–
2	I0	–	–	–
3 – 4	IOF0 – IOF1	3 – 4	RF0 – RF1	Buried feedback of dual feedback macrocell
5 – 8	IO0 – IO3	5 – 8	R0 – R3	Buried feedback of single feedback macrocell
9 – 10	IOF2 – IOF3	9 – 10	RF2 – RF3	Buried feedback of dual feedback macrocell
11	I3 or OE	–	–	–
12	GND	–	–	–
13	I4 or CLK2 or LE	–	–	–
14	I1	–	–	–
15 – 16	IOF4 – IOF5	11 – 12	RF4 – RF5	Buried feedback of dual feedback macrocell
17 – 20	IO4 – IO7	13 – 16	R4 – R7	Buried feedback of single feedback macrocell
21 – 22	IOF6 – IOF7	17 – 18	RF6 – RF7	Buried feedback of dual feedback macrocell
23	I2	–	–	–
24	VCC	–	–	–
–	–	1	GLOBAL	Global preset, reset, and preload
–	–	2	OBS	Observability node

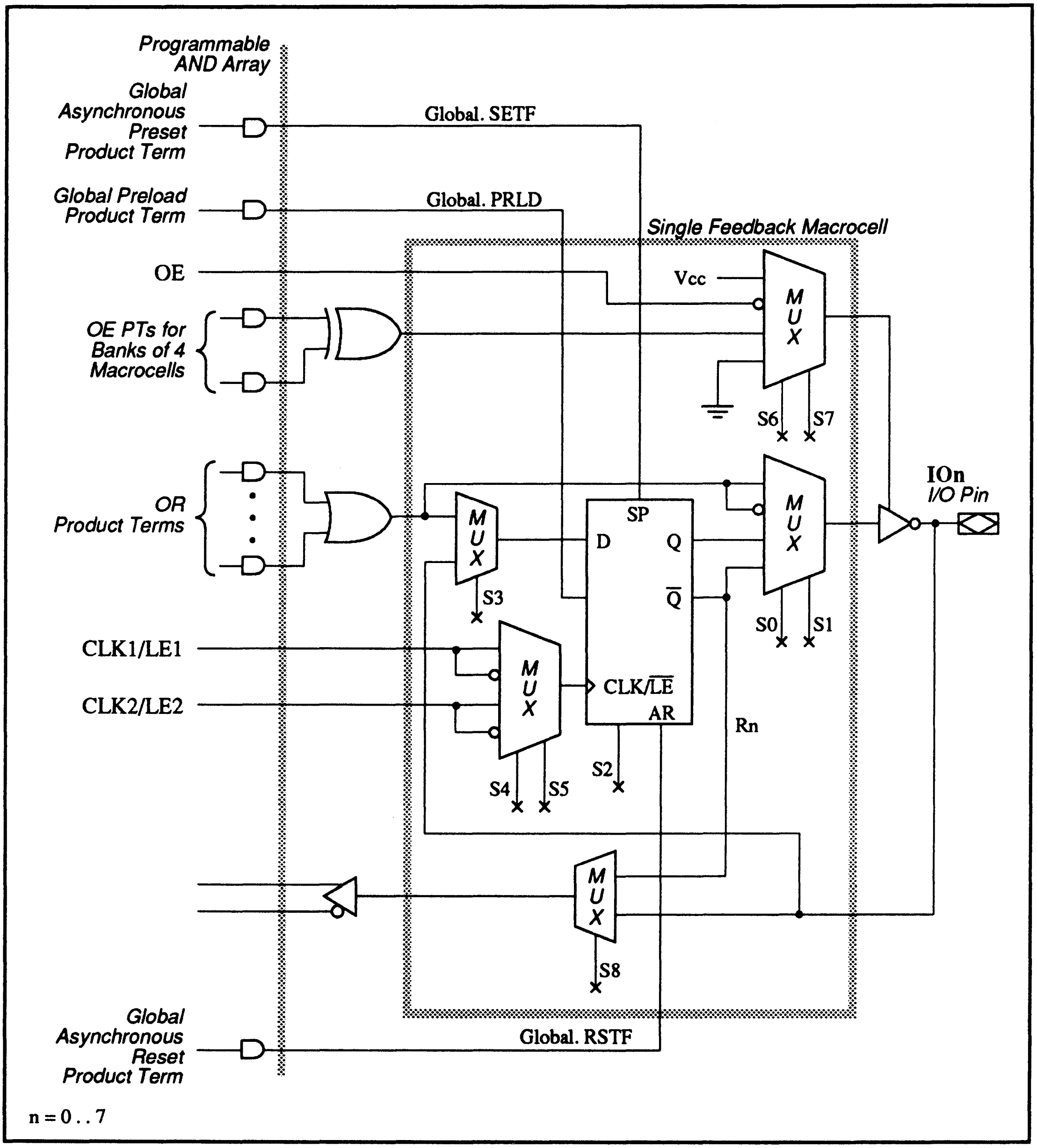
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

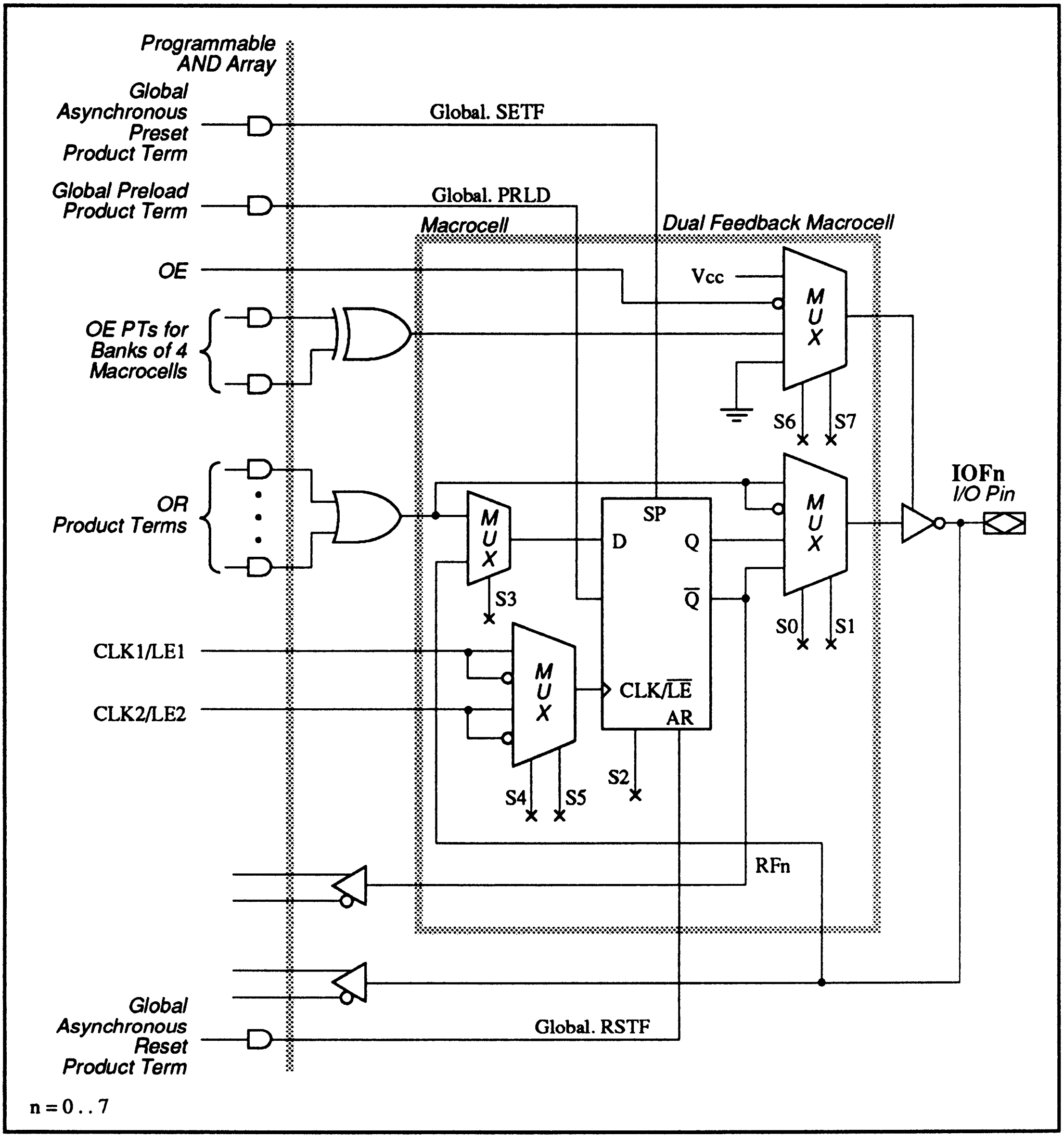


29M16: Block Diagram Showing Pin and Node Locations

29M16: PALCE29M16



29M16: Single Feedback Macrocell Diagram



29M16: Dual Feedback Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Common external or grouped XOR output-enable control
- Programmable clock polarity
- Macrocells with different configurations

Common External or Grouped XOR Output-Enable Control

Each group of four outputs shares a common output-enable control, which can be selected from a common external output pin, or an XOR function with two product terms, or permanently enabled, or permanently disabled. The four groups of outputs are arranged as follows.

group 1:	IOF0, IOF1, IO0, IO1
group 2:	IOF2, IOF3, IO2, IO3
group 3:	IOF4, IOF5, IO4, IO5
group 4:	IOF6, IOF7, IO6, IO7

To program the output-enable functions, you can write a .TRST functional equation for any output within a group, or for an output vector using the language syntax shown below.[12]

[12] Refer to VECTOR in Chapter 10, in this section, for details on output vector.

Syntax 1

```
Pin Statement(s)  PIN  Input_pin_name
                  PIN  Output_pin_location     I/O_pin_name          Storage_type
                  :
```

```
Equation(s)  for external output-enable pin
                  I/O_pin_name.TRST
                  =  Input_pin_name
     or      grouped XOR function
                  =  Boolean expression using XOR function with two product terms*
     or      permanently enabled
                  =  VCC
     or      permanently disabled
                  =  GND
```

* *If you write separate .TRST equations for each output sharing the same XOR function, each equation must list the same literals and operators in the same sequence. Otherwise, an error message occurs if used in the same design.*

Example

```
:
PIN        11      OE                        ;OE input
PIN        2       I0                        ;input
PIN        14      I1                        ;input
PIN        23      I2                        ;input
PIN        3..4, 5..6      O[1..4]           ;grouped output pins 3,4,5,8 into an
                                             ;     output vector O[1..4]
PIN        9..10,  7..8    O[5..8]
PIN        15..16, 17..18  O[9..12]
PIN        21..22, 19..20  O[13..16]
:
O[1..4].TRST = OE                            ;output buffers O1 to O4 are controlled
                                             ;     by external OE pin
O[5..8].TRST = I0 * I1                       ;output buffers O5 to O8 are controlled
             :+: /I2                         ;     by the XOR function
O[9..12].TRST = VCC                          ;output buffers O9 to O12 are permanently
                                             ;     enabled
O[13..16] = GND                              ;output buffers O13 to O16 are permanently
                                             ;     disabled
```

29M16: PALCE29M16

Programmable Clock Polarity

The 29M16 allows you to select one of two clock or latch-enable pins for each flip-flop using the .CLKF functional equation. An active-low equation produces either a falling-edge triggered clock or an active-low latch enable. An active-high equation produces either a rising-edge triggered clock or an active-high latch enable. To select which clock input, simply write a .CLKF functional equation for any output.

Syntax

```
Pin Statement(s)  :
                  PIN Clock_input_pin_location
                  Clock_input_pin_name Input_type
                  PIN Output_pin_location I/O_pin_name Storage_type
                  :
Equation(s)       Output_pin_name.CLKF*
              For common external clock input
                  =  Clock_input_pin_name
    or        for individual clock product term
                  =  Boolean expression with one product term
```

* *For rising-edge-triggered clock or active-high latch enable, use an active-high output_pin_name; for falling-edge-triggered clock or active-low latch enable, use an active-low output_pin_name.*

Example

```
 :
 PIN   1    CLK1         ;input
 PIN   3    IO0   REG    ;output, registered
 :
 /IO0  .CLKF = CLK1      ;clocks IO0 register on the falling edge of CLK1 clock
                         ;    signal
```

Macrocells with Different Configurations

The 29M16 has two types of macrocells, a single-feedback macrocell and a dual-feedback macrocell.

Each type of macrocell allows a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	SINGLE-FEEDBACK MACROCELL	DUAL-FEEDBACK MACROCELL
Programmable Feedback		
Output with I/O feedback	X	X
Output with / Q feedback	X	X
Output with /Q and I/O feedback	N/A	X
Buried register with /Q feedback	X	X
Register input with /Q	X	X

29MA16: PALCE29MA16

PIN AND NODE DESCRIPTIONS

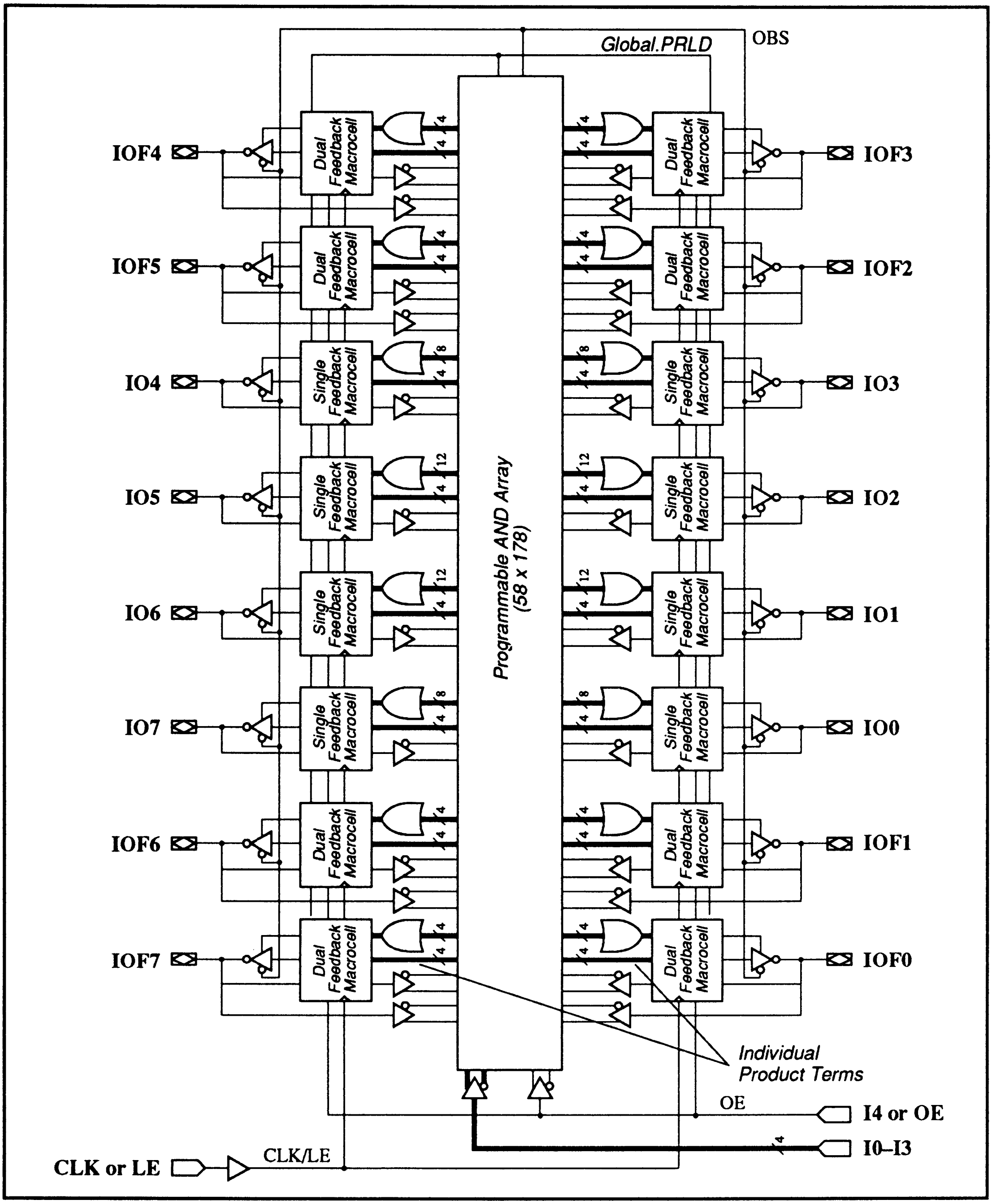
- 24 pins
- 16 buried nodes
- 1 global preset, reset, and preload node
- 1 observability node

29MA16

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK or LE	–	–	–
2	I0	–	–	–
3 – 4	IOF0 – IOF1	3 – 4	RF0 – RF1	Buried feedback of dual feedback macrocells
5 – 8	IO0 – IO3	5 – 8	R0 – R3	Buried feedback of single feedback macrocells
9 – 10	IOF2 – IOF3	9 – 10	RF2 – RF3	Buried feedback of dual feedback macrocells
11	I4 or OE	–	–	–
12	GND	–	–	–
13	I1	–	–	
14	I2	–	–	–
15 – 16	IOF4 – IOF5	11 – 12	RF4 – RF5	Buried feedback of dual feedback macrocells
17 – 20	IO4 – IO7	13 – 16	R4 – R7	Buried feedback of single feedback macrocells
21 – 22	IOF6 – IOF7	17 – 18	RF6 – RF7	Buried feedback of dual feedback macrocells
23	I3	–	–	–
24	VCC	–	–	–
–	–	1	GLOBAL	Global preload
–	–	2	OBS	Observability node

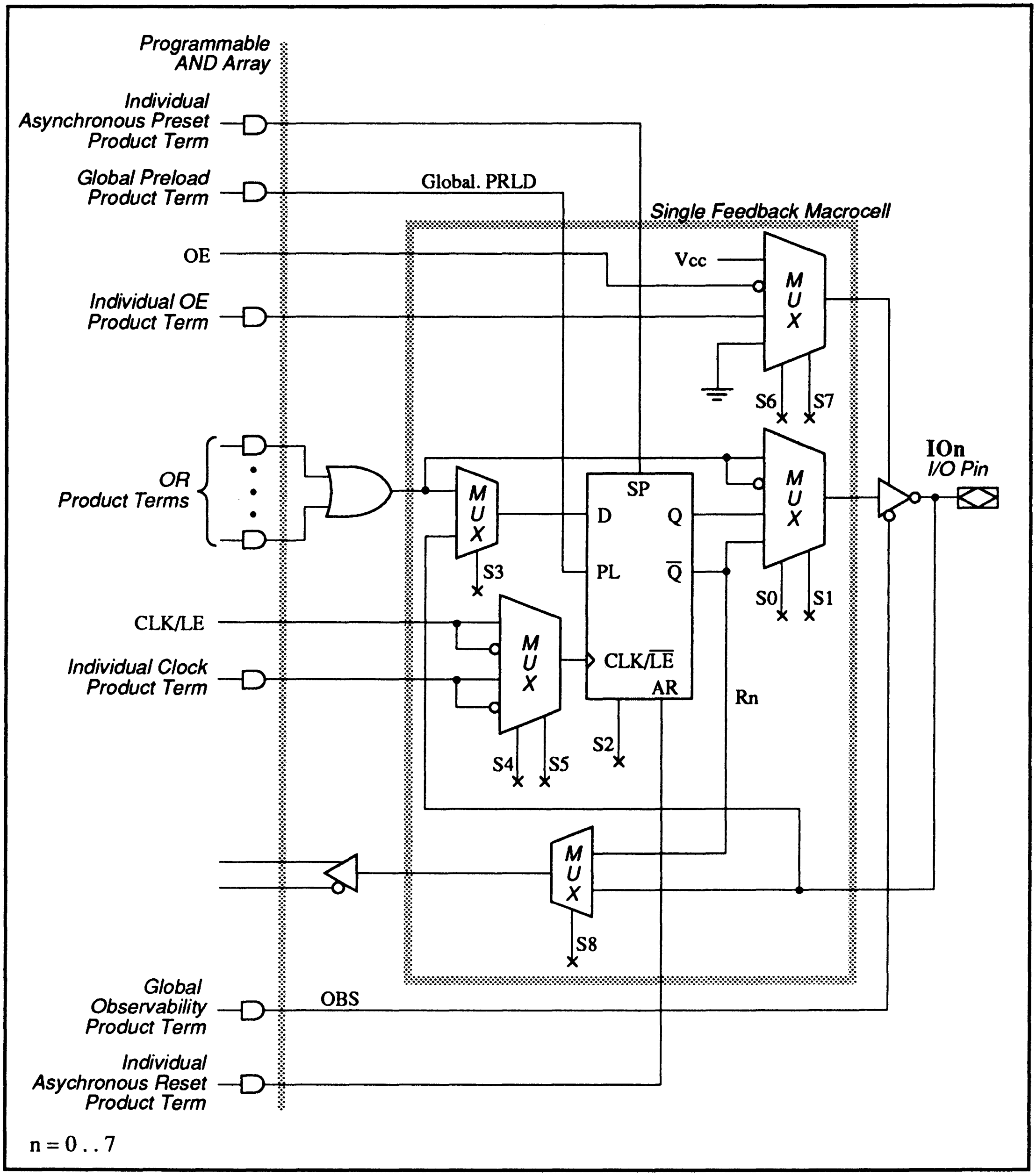
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

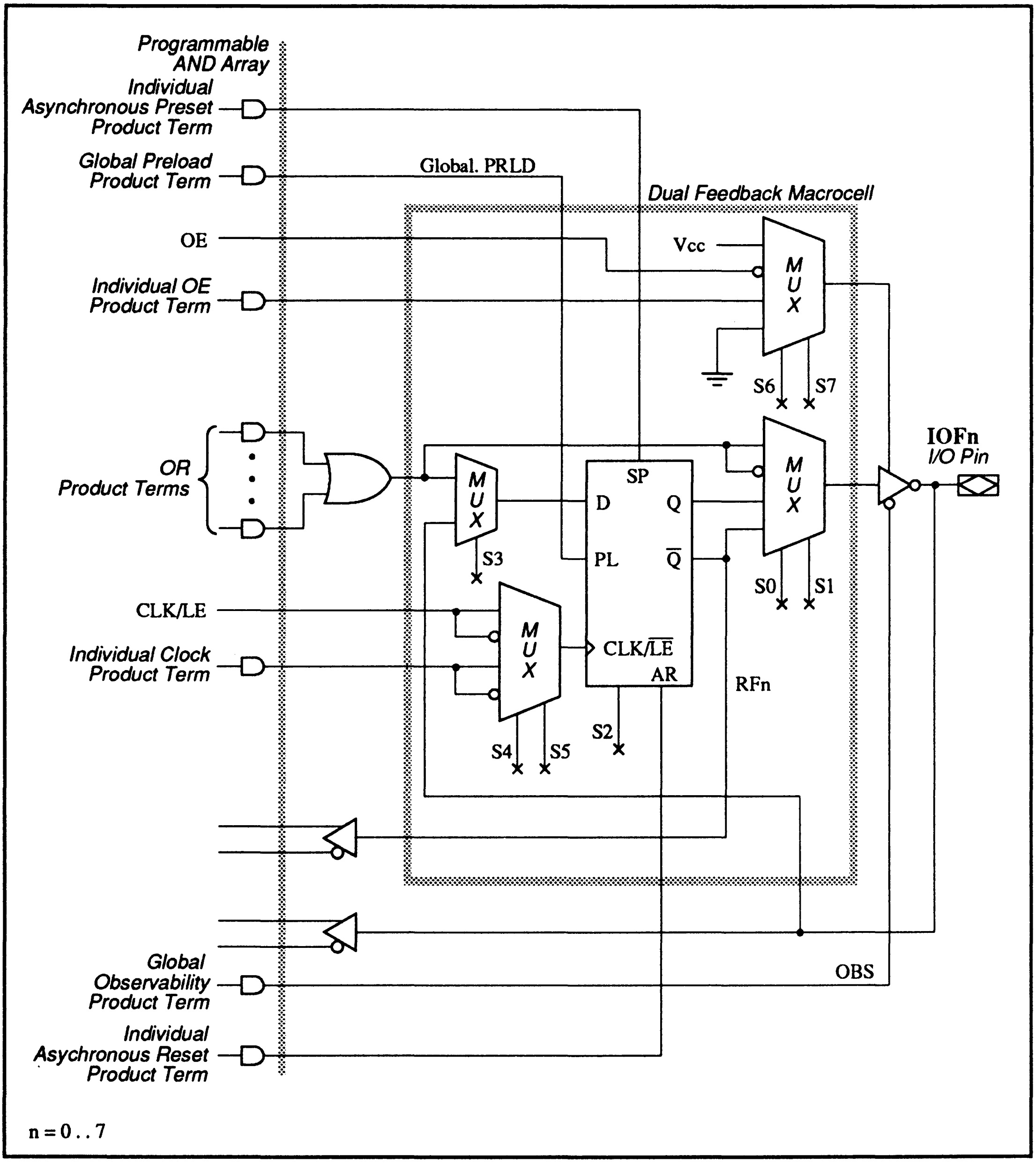


29MA16: Block Diagram Showing Pin and Node Locations

29MA16: PALCE29MA16


Programmable AND Array
Individual Asynchronous Preset Product Term
Global Preload Product Term
Global. PRLD
Single Feedback Macrocell
OE
Vcc
M U X
Individual OE Product Term
S6
S7
IOn
I/O Pin
OR Product Terms
SP
D
Q
PL
Q̅
S3
S0
S1
CLK/LE
Individual Clock Product Term
CLK/L̅E̅
AR
Rn
S2
S4
S5
S8
Global Observability Product Term
OBS
Individual Asychronous Reset Product Term
n = 0 . . 7

29MA16: Single Feedback Macrocell Diagram


Programmable AND Array
Individual Asynchronous Preset Product Term
Global Preload Product Term
Global. PRLD
Dual Feedback Macrocell
OE
Vcc
Individual OE Product Term
M U X
S6
S7
OR Product Terms
M U X
D
SP
Q
PL
Q̅
S3
S0
S1
IOFn
I/O Pin
CLK/LE
Individual Clock Product Term
M U X
CLK/L̅E̅
AR
RFn
S2
S4
S5
Global Observability Product Term
OBS
Individual Asychronous Reset Product Term
n = 0 . . 7

29MA16: Dual Feedback Macrocell Diagram

29MA16: PALCE29MA16

SPECIAL PROGRAMMING FEATURES

- Clock control with individual clock product term or common external clock pin with programmable clock polarity
- Macrocells with different configurations

Clock Control

The 20MA16 allows you to select either a common external clock pin or an individual clock product term for each flip-flop using the .CLKF functional equation. An active-low equation produces either a falling edge triggered clock or an active-low latch enable. An active-high equation produces either a rising edge triggered clock or an active-high latch enable. To select a clock input, simply write a .CLKF functional equation for any output.

Syntax

```
Pin Statement(s)  :
                  PIN  Input_pin_location*    Input_pin_name**
                  PIN  Output_pin_location I/O_pin_name REG or LAT
                  :

Equation(s)       Output_pin_name .CLKF
              for common external clock input
                  = Input_pin_name
     or       for individual clock product term
                  = Boolean expression with one product term

*   Clock or latch input
**  For rising-edge-triggered clock or active-high latch enable, use an active-high
    output_pin_name; for falling-edge-triggered clock or active-low latch enable,
use
    an active-low output_pin_name.
```

Example

```
:
PIN      1    CLK               ;clock input
PIN      2    I0                ;input
PIN      14   I2                ;input
PIN      3    I00    REG        ;output, registered
PIN      4    I01    REG        ;output, registered
PIN      5    I0     REG        ;output, registered
PIN      6    I1     REG        ;output, registered
:
/I00.CLKF = CLK                 ;clocks I00 register on the falling edge of CLK
                                ;    clock signal
I01.CLKF = CLK                  ;clocks I01 register on the rising edge of CLK1
                                ;    clock signal
I0.CLKF = I0 *  I2              ;output register clock is asserted when (I0 * I2)
                                ;    is true
/I1.CLKF = I0 * I2              ;output register clock is asserted when /(I0 * I2) is
                                ;    true
```

Macrocells with Different Configurations

The 29MA16 has two types of macrocells, a single-feedback macrocell and a dual-feedback macrocell.

Each type of macrocell provides a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	SINGLE-FEEDBACK MACROCELL	DUAL-FEEDBACK MACROCELL
Programmable Feedback		
Output with I/O feedback	X	X
Output with / Q feedback	X	X
Output with /Q and I/O feedback	N/A	X
Buried register with /Q feedback	X	X
Register input with /Q	X	X

30S16: PLS30S16

PIN AND NODE DESCRIPTIONS

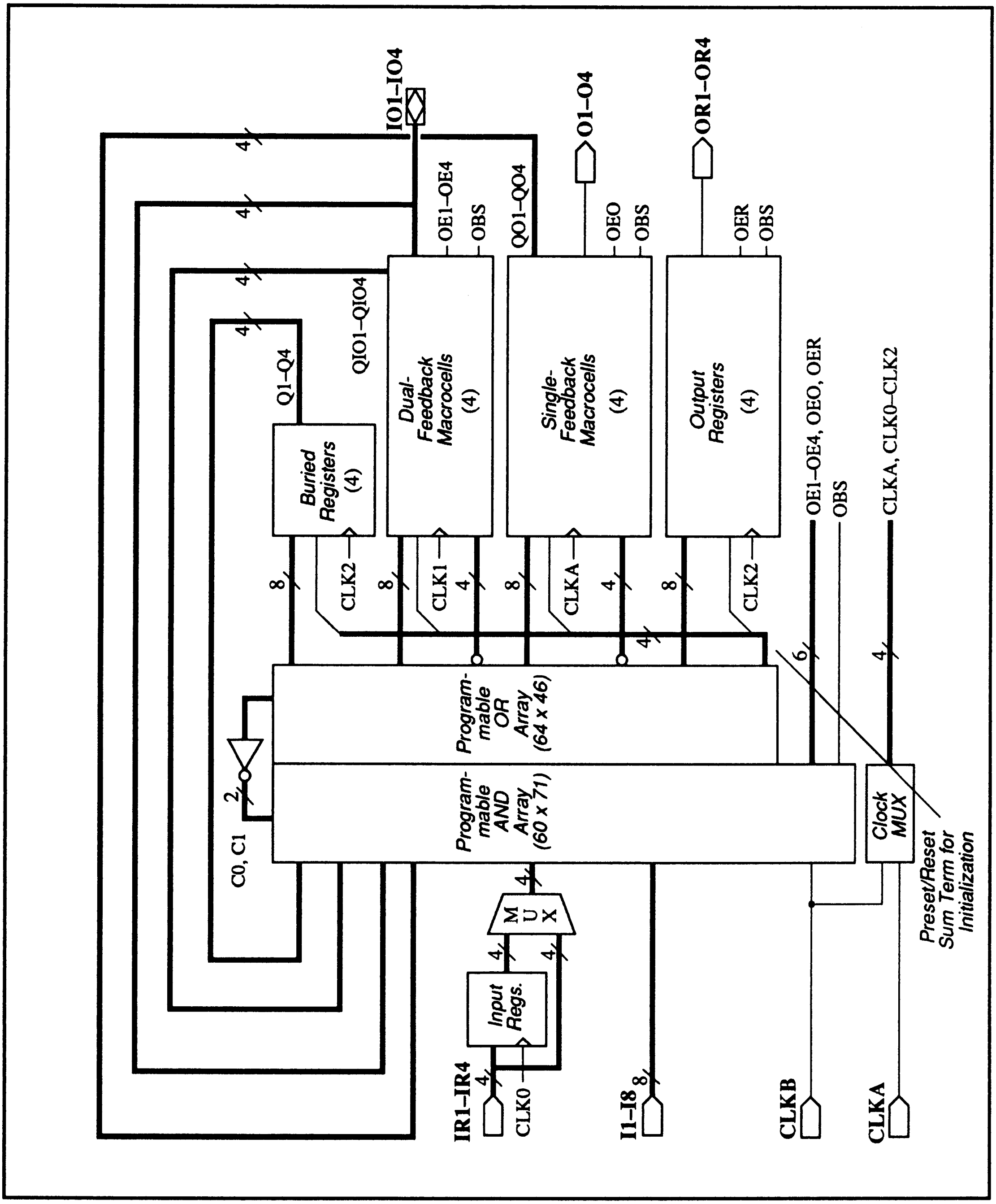
- 28 pins
- 14 buried nodes
- 2 complement array nodes
- 1 observability nodes

30S16

Pin Location	Pin Name	Node Location	Node Name	Node Descriptions
1	CLKA	–	–	–
2 – 3	I1 – I2	–	–	–
4	CLKB	–	–	–
5 – 7	I3 – I5	–	–	–
8 – 9	IO1 – IO2	5 – 6	QIO1 – QIO2	Q feedback from I/O
10 – 13	OR1 – OR4	–	–	–
14	GND	–	–	–
15 – 18	O1 – O4	1 – 4	Q01 – Q04	Q feedback from output
19 – 20	IO3 – IO4	7 – 8	QIO3 – QIO4	Q feedback from I/O
21 – 24	IR1 – IR4	–	–	–
25 – 27	I6 – I8	–	–	–
28	VCC	–	–	–
–	–	9 – 12	Q1 – Q4	Buried Q feedback
–	–	13 – 14	C0 – C1	Complement arrays
–	–	15	OBS	Observability

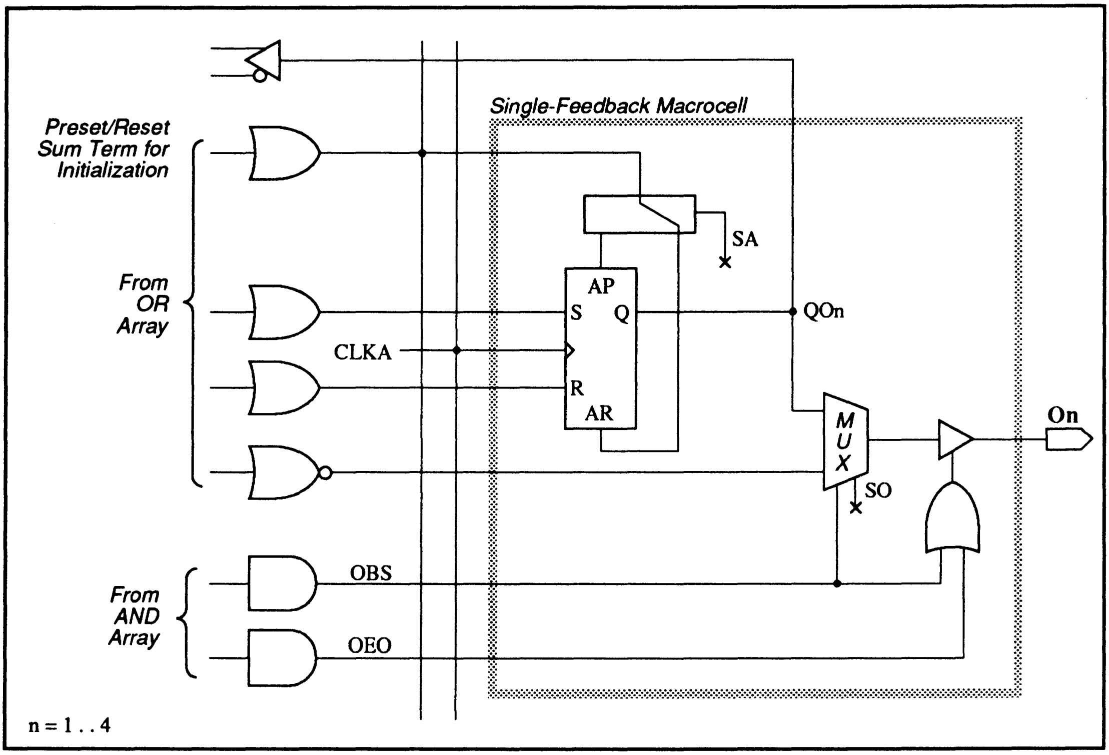
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

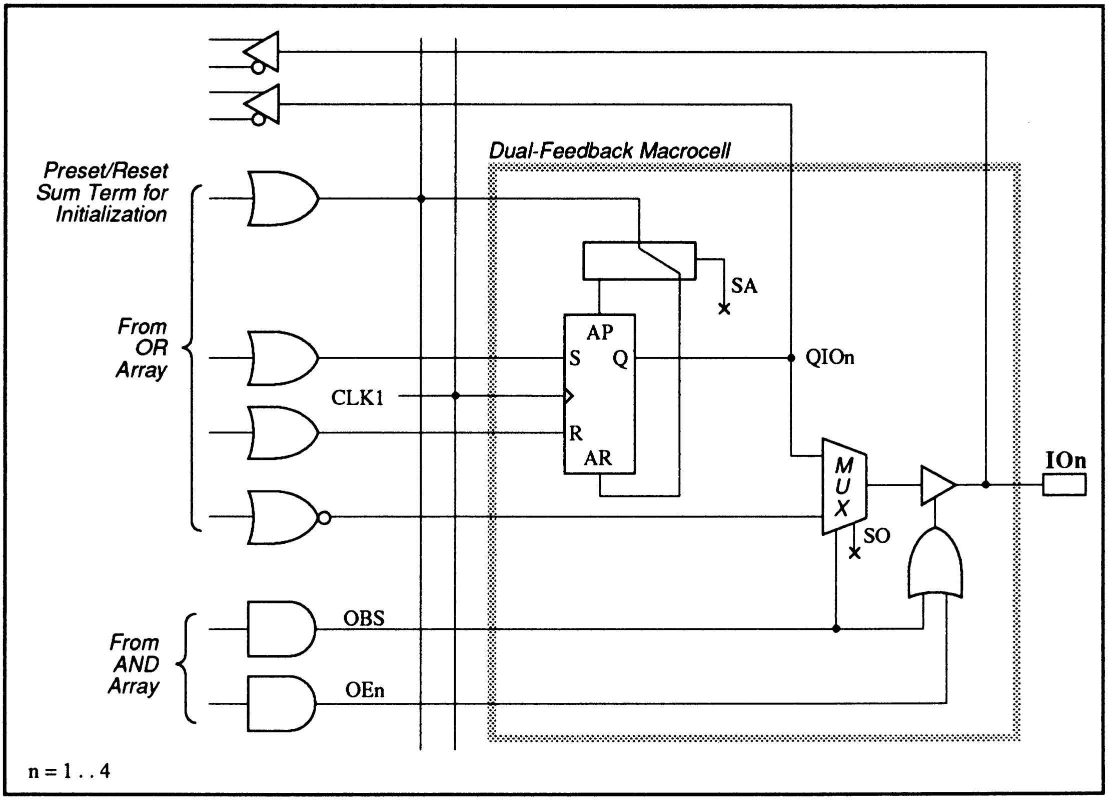


30S16: Block Diagram Showing Pin and Node Locations

30S16: PLS30S16

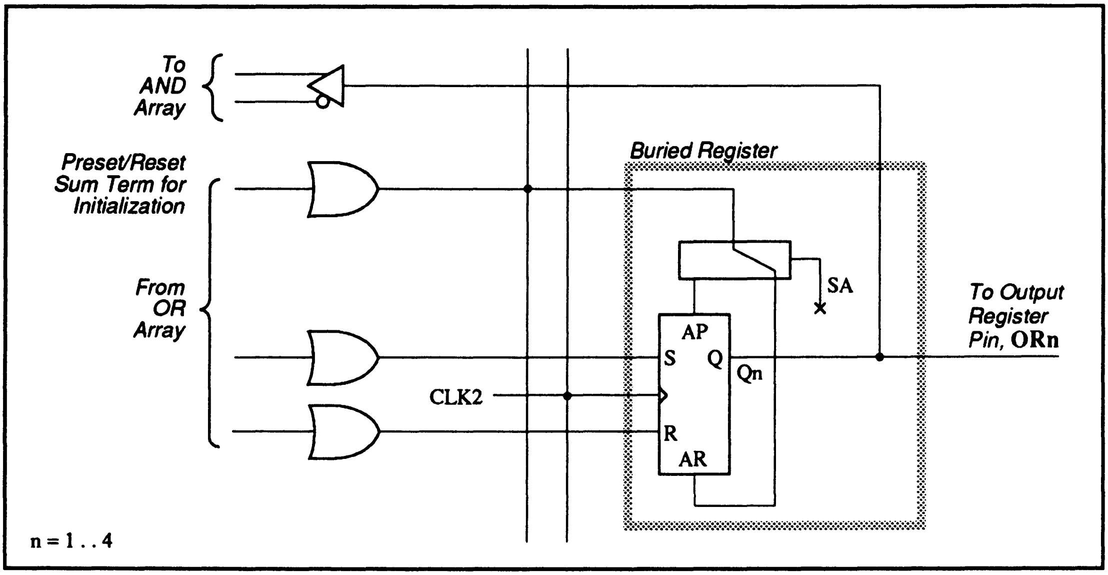


30S16: Single Feedback Macrocell Diagram

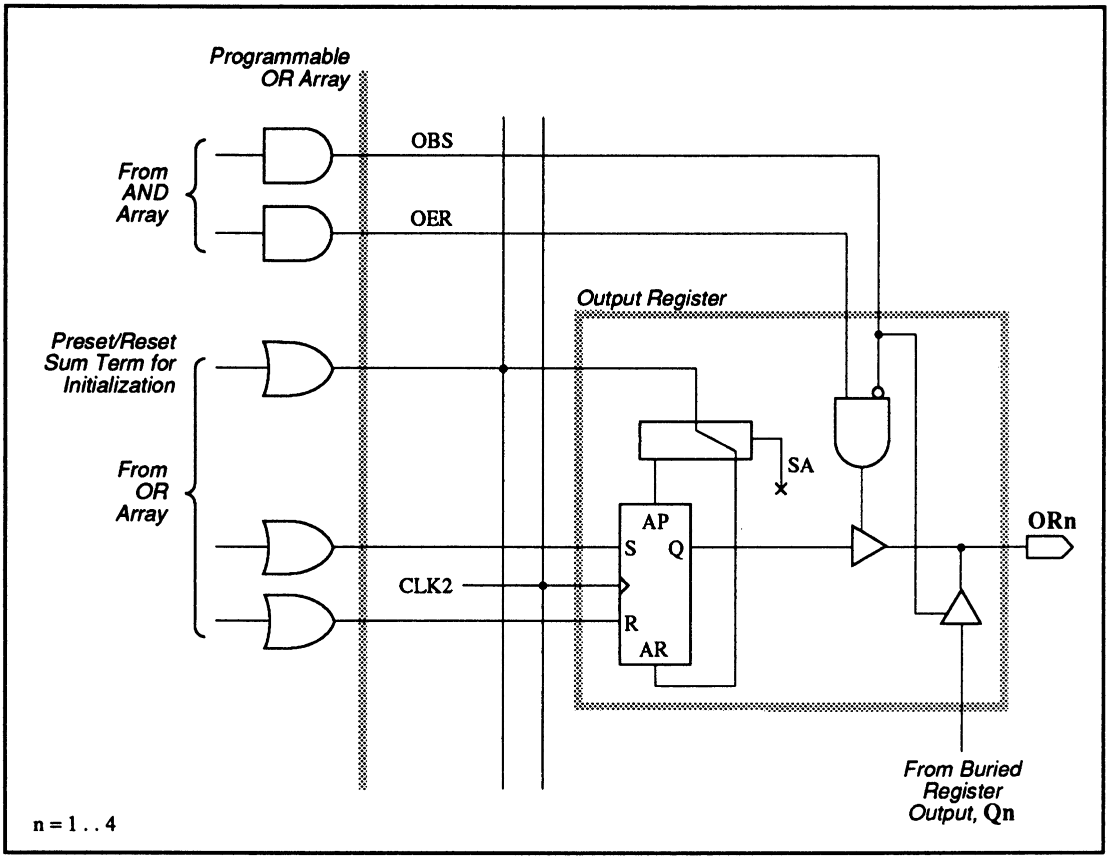


30S16: Dual Feedback Macrocell Diagram

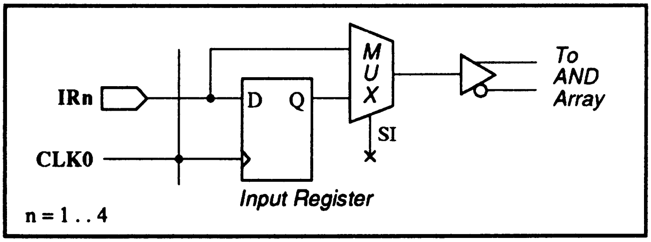
30S16: PLS30S16


To AND Array
Preset/Reset Sum Term for Initialization
From OR Array
Buried Register
SA
AP
S
Q
Qn
CLK2
R
AR
To Output Register Pin, ORn
n = 1 . . 4

30S16: Buried Register Macrocell Diagram

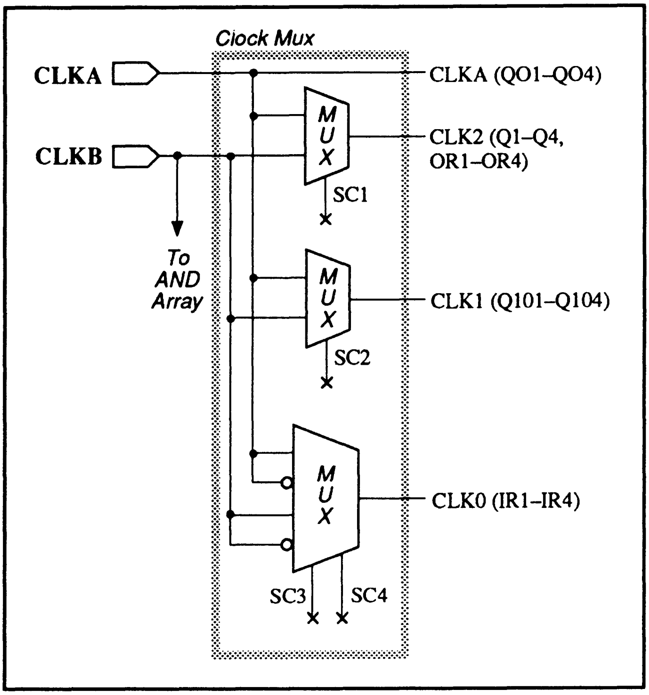


30S16: Output Register Macrocell Diagram



30S16: Input Register Macrocell Diagram

30S16: PLS30S16



30S16: Clock MUX Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Individual/group output-enable product terms
- Multiple clock controls
- Preset/reset sum terms for initialization
- Macrocells with different configurations
- Outputs with Q feedback
- Outputs with I/O and Q feedbacks
- Registered Inputs
- State bits

Individual and Group Output-Enable Product Terms

There are two types of output-enable control.

- Individual output-enable product term
- Group product term

Outputs	Product Terms	Type of Product
IO1 - IO4	OE1 - OE4	Individual
O1 - O4	OEO	Group
OR1 - OR4	OER	Group

Individual Output-Enable Product Term

Outputs IO1 to IO4 each have an individual product term to control the output buffer. You use the syntax below to program each product term.

Syntax

```
Pin Statement(s)  PIN Output_pin_location I/O_pin_name Storage_type
                  :
Equation(s)       Output_pin_name.TRST = Boolean expression with one product term
```

Example

```
:
PIN     2    I1    COMB      ;input, combinatorial
PIN     3    I2    COMB      ;input, combinatorial
:
PIN     8    IO1   REG       ;I/O, registered
:
IO1.TRST = I1 * /I2          ;IO1 is individually controlled by (I1 * /I2)
```

Group Product Term

The output buffers for outputs O1 to O4 are controlled by one product term, OEO. The output buffers for outputs OR1 to OR4 are controlled by another product term, OER. To program these groups of product terms, write a .TRST equation for any output within a group using the syntax shown above. You can also group the outputs together and write a .TRST equation for the assigned group name using the GROUP statement, or write a .TRST equation for an output vector.[13]

Example

```
:
PIN     2    I1    COMB       ;input, combinatorial
PIN     3    I2    COMB       ;input, combinatorial
:
PIN     15   01    REG        ;output, registered
PIN     16   02    REG        ;output, registered
PIN     17   03    REG        ;output, registered
PIN     18   04    REG        ;output, registered
:
PIN     10   OR1 REG          ;output, registered
:
GROUP  Group0  01 02 03 04
:
Group0.TRST = I1 * /I2        ;01 to 04 is controlled by ( I1 * /I2 )
OR1.TRST = I2                 ;OR1 to OR4 is controlled by ( I2 )
```

[13] Refer to VECTOR in Chapter 10, in this section, for details.

Multiple Clock Controls

The 30S16 has four clock controls. Each controls a group of registers and can be programmed into a selection of clock signals, as shown in the clock MUX macrocell diagram. The clock arrangement is summarized in the table below.

OUTPUTS	CLOCK CONTROLS	CLOCK SIGNALS
IR1 - IR4	CLK0	CLKA (default) /CLKA CLKB /CLKB
IO1 - IO4	CLK1	CLKA (default) CLKB
OR1 - OR4, Q1 - Q4	CLK2	CLKA (default) CLKB
O1 - O4	CLKA	CLKA

You can assign a clock to a clock control by writing a .CLKF equation for any output within the clock bank. You can also use the VECTOR or GROUP statements to assign a group of registers.[14] If you do not assign any clock signal to a bank of registers, the default clock signal is used.

> **Note:** You may only assign the identical clock signal to the output registers within a clock bank.

Syntax

```
Pin Statement(s)   :
                   PIN  I/O_pin_location    I/O_pin_name    Storage_type
                   :
Equation(s)        Output_pin_name.CLKF = Clock_signal
```

[14] Refer to Chapter 10, in this section, for details.

30S16: PLS30S16

Example

```
:
PIN   1    CLKA                          ;clock input
PIN   4    CLKB                          ;clock input
:
PIN   8..9, 19..20  IO[1..4]             ;output vector
PIN   21..24        IR[1..4]             ;output vector
PIN   10   OR1                REG        ;output, registered
:
IO[1..4].CLKF  = CLKB                    ;IO1 to IO4 registers are controlled by CLKB
OR1.CLKF = CLKA                          ;OR1 to OR4 registers are controlled by CLKA
:
                                         ;if no clock is assigned to outputs IR1 to
                                         ;   IR4, by default they are controlled by
                                         ;   CLKA
```

Preset/Reset Sum Term for Initialization

All registers in the 30S16 can be initialized to a predefined state under certain conditions. Each flip-flop contains a preset and a reset line, only one of which can be selected. There are four sum terms. Each sum term defines the conditions under which the selected set or reset line is activated within a bank of registers with the same clock control.[15] While the initialization condition must be the same for each flip-flop in a register bank, you can individually program the preset/reset sum term to set or reset each flip-flop.

You can initialize registers using either state-machine language or Boolean equations. The PALASM language syntax for both follows.

[15] See the multiple clock-control feature for the register-bank arrangements.

```
Pin Statement(s)  :
                  I/O_pin_location      I/O_pin_name       Storage_type
                  :
                  Buried_node_location   Buried_node_name   Storage_type
Equation(s)  for initialization to one
                  I/O_pin_name.SETF     = Boolean expression
                  Buried_node_name.SETF = Boolean expression
             or
                  for initialization to zero
                  I/O_pin_name.RSTF     = Boolean expression
                  Buried_node_name.RSTF = Boolean expression
```

Note: The Boolean expression used on the right side of the functional equations must use the same sequence of literals and operators for all outputs in the same register bank, since all use the same sum term.

State-Machine Language

For a state machine design, follow the steps below to initialize a state machine to a known starting state using the programmable, asynchronous preset/reset sum term for each bank of registers.

1. Initialize the state machine in the conditions segment, by defining the initialization condition as a set of input and/or feedback value(s).

   CONDITIONS
   INIT = Boolean expression

2. Define the starting state and outputs as a function of the INIT condition in the Setup and Defaults portion of the state segment using the syntax below.

   START_UP := Condition -> Desired_state

   or for a Mealy state machine design

   START_UP.OUTF := Condition -> Desired_outputs

The next example shows initialization to a known state using a state-machine design for a 30S16 device.

30S16: PLS30S16

Example

```
:
PIN       2     I1                        ;input
PIN       3     I2                        ;input
PIN       5     I3                        ;input
PIN       6     I4                        ;input
PIN       15    01      REG               ;output,registered
PIN       16    02      REG               ;output,registered
:
NODE      9     Q1      REG               ;buried node
:
STATE
MEALY_MACHINE
DEFAULT_BRANCH  HOLD_STATE
START_UP := INIT -> NEXT_STATE
START_UP.OUTF :=  INIT -> 01 * Q2         ;when INIT is true, outputs 01 and 02 will be
                                          ;   set to high, while other outputs will be
                                          ;   reset to low
CONDITIONS
INIT = I1 * I2 * I3 * I4                  ;INIT is defined as condition
                                          ;   (I1 * I2 * I3 * I4)
```

Boolean Equations

You can initialize the registers using Boolean equations. Defining the programmable initialization function consists of writing a .SETF or .RSTF functional equation to set or reset for each flip-flop. The Boolean expression using sum term defines the conditions under which initialization occurs. The .SETF or .RSTF equation defines the value (set or reset) of each flip-flop. If all flip-flops within a bank of registers are set or reset to the same value, you can write a single equation for an output vector.[16]

[16] Refer to Chapter 10, in this section, for details.

Syntax

```
Pin Statement(s) I/O_pin_location   I/O_pin_name    Storage_type
                 :
                 Buried_node_location  Buried_node_name  Storage_type

Equation(s)
                 for initialization to one
                 I/O_pin_name.SETF      = Boolean expression
                 Buried_node_name.SETF  = Boolean expression

             or  for initialization to zero
                 I/O_pin_name.RSTF      = Boolean expression
                 Buried_node_name.RSTF  = Boolean expression
```

Note: *The Boolean expression used on the right side of the functional equations must use the same sequence of literals and operators for all outputs in the same register bank, since all use the same sum term.*

Example

```
:
PIN     2       I1                           ;input
PIN     3       I2                           ;input
PIN     5       I3                           ;input
PIN     6       I4                           ;input
PIN     8..9,   19..20  IO[1..4]  REG        ;output, registered
PIN     15..16, 17..18  O[1..4]   REG        ;output, registered
:
NODE    9  Q1                     REG        ;buried node
IO[1..4].SETF = I1 * I2  * /I4               ;when (I1*I2*/I4) is true, registers IO1 to
             +I3 * /I2                       ;   IO4 are set to high
O[1..2].SETF = I1 * I2 * /I4                 ;when ((I1*I2*/I4)+(I3*/I2)) is true,
                                             ;   registers O1 & O2 are set to high
O[3..4].RSTF = I1 * I2 * /I4                 ;registers O3 & O4 are set to low under the
            +I3 * /I2                        ;   same condition as O1 and O2
Q1.RSTF = I1 * I2                            ;buried register Q1 is set to low when
                                             ;   (I1 * I2) is true
```

30S16: PLS30S16

Macrocells with Different Configurations

The 30S16 has four types of macrocells: output register, single-feedback macrocell, dual-feedback macrocell, and buried register. Each type of macrocell provides a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	OUTPUT REGISTER	SINGLE-FEEDBACK MACROCELL	DUAL-FEEDBACK MACROCELL	BURIED REGISTER MACROCELL
Non-Programmable Feedback				
Registered Output with I/O feedback	X	N/A	N/A	N/A
Programmable Feedback				
Output with Q feedback	N/A	X	X	N/A
Output with Q and I/O feedback	N/A	N/A	X	N/A
Buried register with Q feedback	N/A	X	X	X

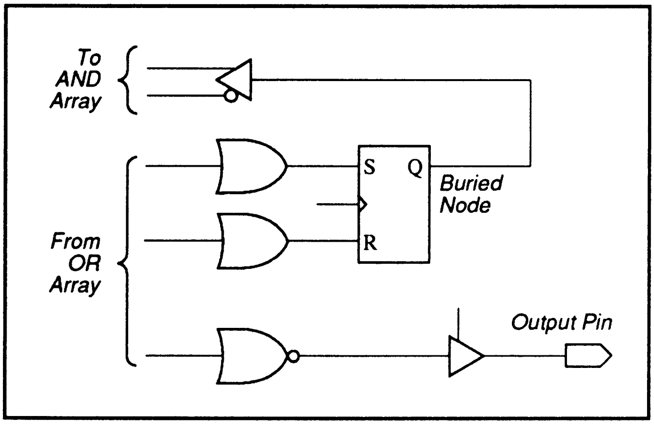
Output with Q Feedback

The single-feedback macrocell in a 30S16 can be configured as either a combinatorial or registered output with the Q register output feeding back to the array. For a combinatorial output, the output and the register each have separate logic using separate product terms. The output uses active-low logic; the buried register uses active-high logic.

Example

Combinatorial output with a feedback

```
:
PIN     2    I1    COMB       ;input
PIN     3    I2    COMB       ;input
PIN     8    I01   COMB       ;I/O, combinatorial
PIN     15   01    REG        ;I/O, registered
:
/II01 = I * /I2               ;defines I01's comb output
I01.S = I2
```

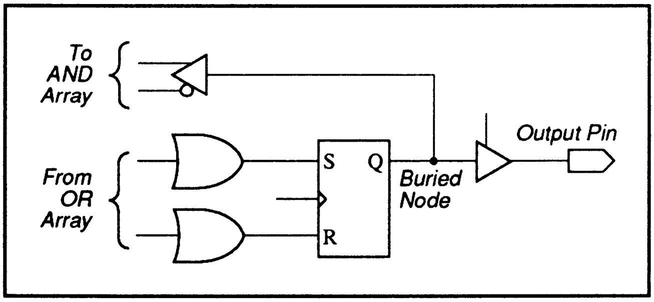


30S16: Combinatorial Output with Q Feedback in Single-Feedback Macrocell

Syntax

Combinatorial output with Q feedback

```
Pin Statement(s)  :
                  PIN  I/O_pin_location         I/O_pin_name          COMB
                  :
                  NODE Buried_node_number    Buried_node_name REG
                  :
Equation(s)       /I/O_pin_name       = Boolean expression
                  Buried_node_name.S = Boolean expression
                  Buried_node_name.R = Boolean expression
                  :
                  < Equations using buried_node_name feedback >
```



30S16: Registered Output with Q Feedback in Single-Feedback Macrocell

30S16: PLS30S16

Syntax

Registered output with Q feedback

```
Pin Statement(s)    PIN  I/O_pin_location I/O_pin_name REG
                    :
                    NODE Buried_node_number Buried_node_name REG
                    :           :
Equation(s)         I/O_pin_name .S = Boolean expression
                    I/O_pin_name .R = Boolean expression
                    :
                    < Equations using buried_node_name as feedback >
```

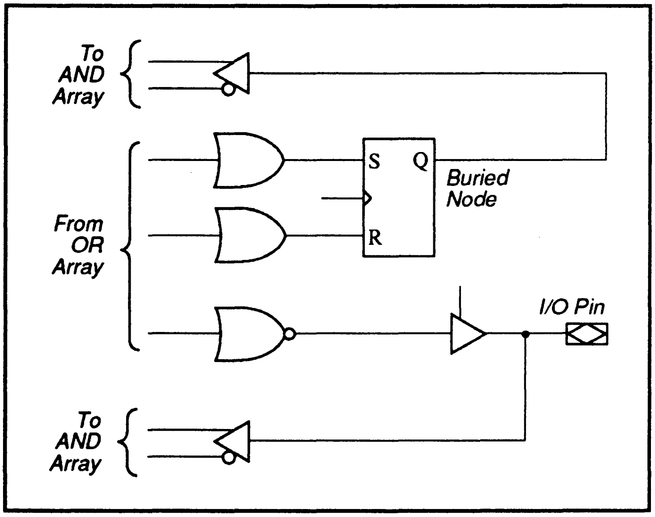
Example

Registered output with Q feedback

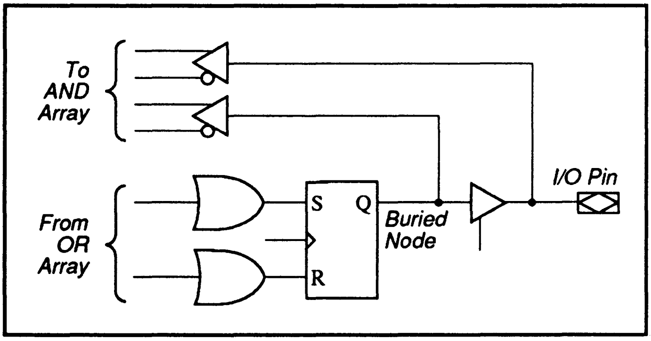
```
:
PIN      2     I1     COMB        ;input, combinatorial
PIN      3     I2     COMB        ;input, combinatorial
:
PIN      15    01     REG         ;output, registered
PIN      16    02     COMB        ;output, combinatorial
:
NODE     1     Q01    REG         ;buried node for register of 01
:
01.S = I1 * I2                    ;equation for registered output I01, input S
01.R = /I2                        ;equation for registered output I01, input R
Q01.S = { 01.S }                  ;S equation for 01's buried node
Q01.R = { 01.R }                  ;R equation for 01's buried node
/02 = Q02                         ;equation using buried Q output, Q01, as feedback
```

Output with Q and I/O Feedbacks

The dual macrocell in a 30S16 can be configured to have both the I/O and Q register outputs feeding back to the logic array for either combinatorial or registered outputs. The language syntax for such arrangements is similar to those described above with just the Q feedback except that you can now include both the Q and I/O feedbacks in an output equations, as illustrated in the example below.



30S16: Combinatorial Output with Q and I/O Feedbacks



30S16: Registered Output with Q and I/o Feedbacks in Dual-Feedback Macrocell

30S16: PLS30S16

Example

Combinatorial and registered output with Q and I/O Feedback

```
          :
PIN       2     I1     COMB         ;input
PIN       3     I2     OMB          ;input
PIN       8     IO1    REG          ;I/O, registered
PIN       9     IO2    COMB         ;I/O, combinatorial
PIN       19    IO3    COMB         ;I/O, combinatorial
:
NODE      5     QIO1   REG          ;buried node for register of IO1
NODE      6     IO2    REG          ;buried node for register of IO2
NODE      7     QIO3   REG          ;buried node for register of IO3
:
IO1.S = I2 * I1                     ;equation for registered output IO1, input S
IO1.R = /I2                         ;equation for registered output IO1, input R
:
QIO2.S = I2 * / I1             ;equation for buried register of output IO2, input S
QIO2.R = /I2 * /I1                  ;equation for buried register of output IO2, input R
/IO2 = I2                           ;equation for combinatorial output IO2
:
QIO3.S = /I2 * QIO2                 ;equation using buried Q output, QIO2, as feedback
QIO3.R = /I2 * / QIO1               ;equation using registered Q output, QIO1,as feedback
IO3 = IO1 * /IO2                    ;equation using registered and combinatorial output,
                                    ;    IO1 and IO2 as feedback
```

Registered Inputs

Inputs IR1 to IR4 can be used as registered D-type or combinatorial inputs. The language syntax is shown below.

Syntax

```
Pin Statement(s)  :
          for combinatorial inputs
               PIN Input_pin_location          Input_pin_name          COMB
               :
     or   for registered Inputs
               PIN Input_pin_location          Input_pin_name          REG
```

> **Note**: Combinatorial is the default, which is used when you do not assign a specific storage type.

Example

```
:
PIN      21   IR1   COMB        ;input, combinatorial
PIN      22   IR2   REG         ;input, registered
PIN      23   IR3               ;input, combinatorial
```

State Bits

The buried registers and all output registers with feedback paths to the AND array can be used for state bit storage. The register associated with the outputs below can be used for state bit storage.

Ql to Q4
O1 to O4
IO1 to IO4

Note: Do not assign names to the outputs and/or buried state registers when you want to assign state bits automatically.

32VX10: PAL32VX10

PIN AND NODE DESCRIPTIONS

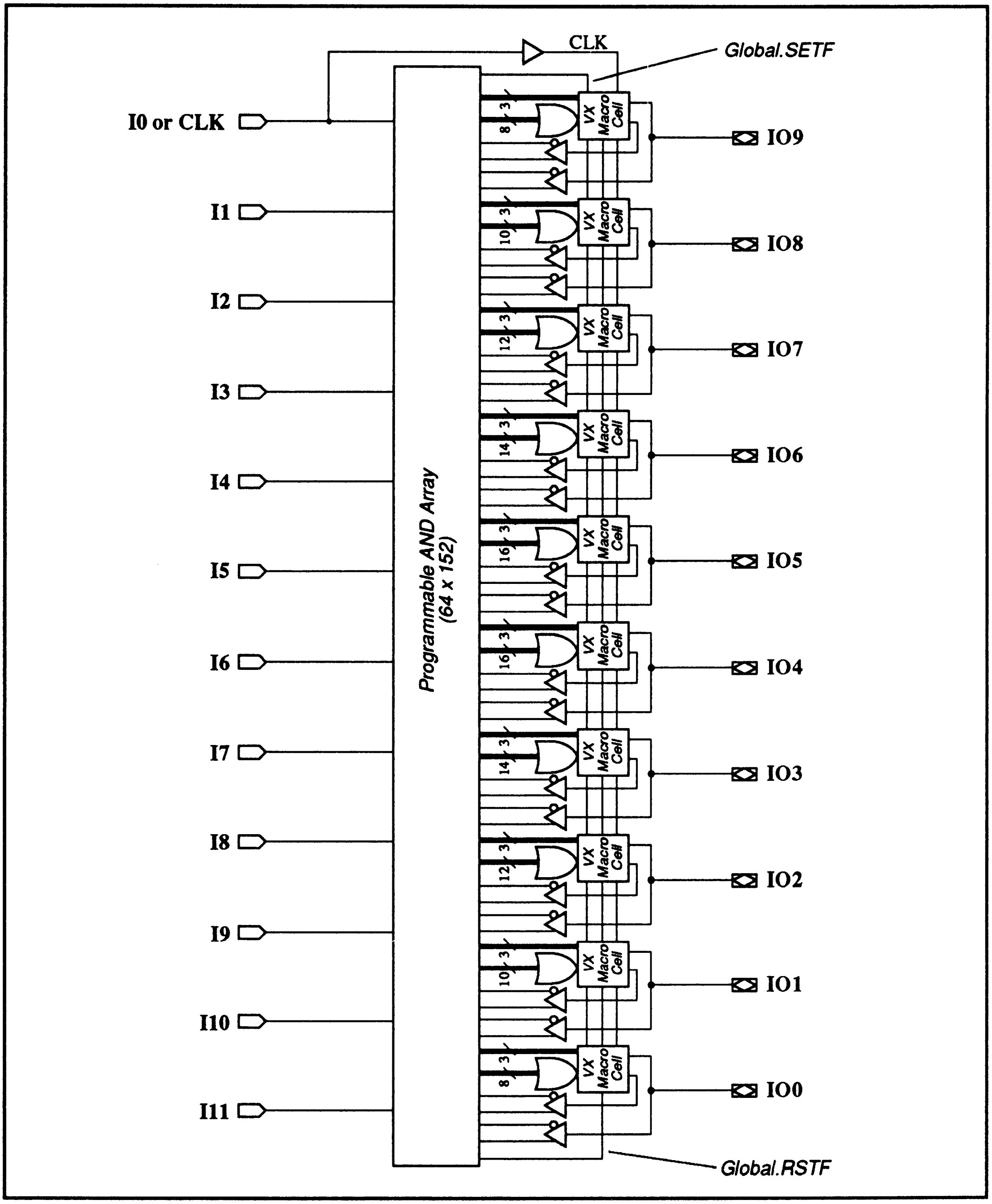
- 24 pins
- 10 buried nodes
- 1 global preset/reset node

32VX10

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	I0/CLK	–	–	–
2 – 11	I1 – I10	–	–	–
12	GND	–	–	–
13	I11	–	–	–
14	IO0 – IO9	2 – 11	R0 – R10	Buried nodes
24	VCC	–	–	–
–	–	1	GLOBAL	Global Preset and reset

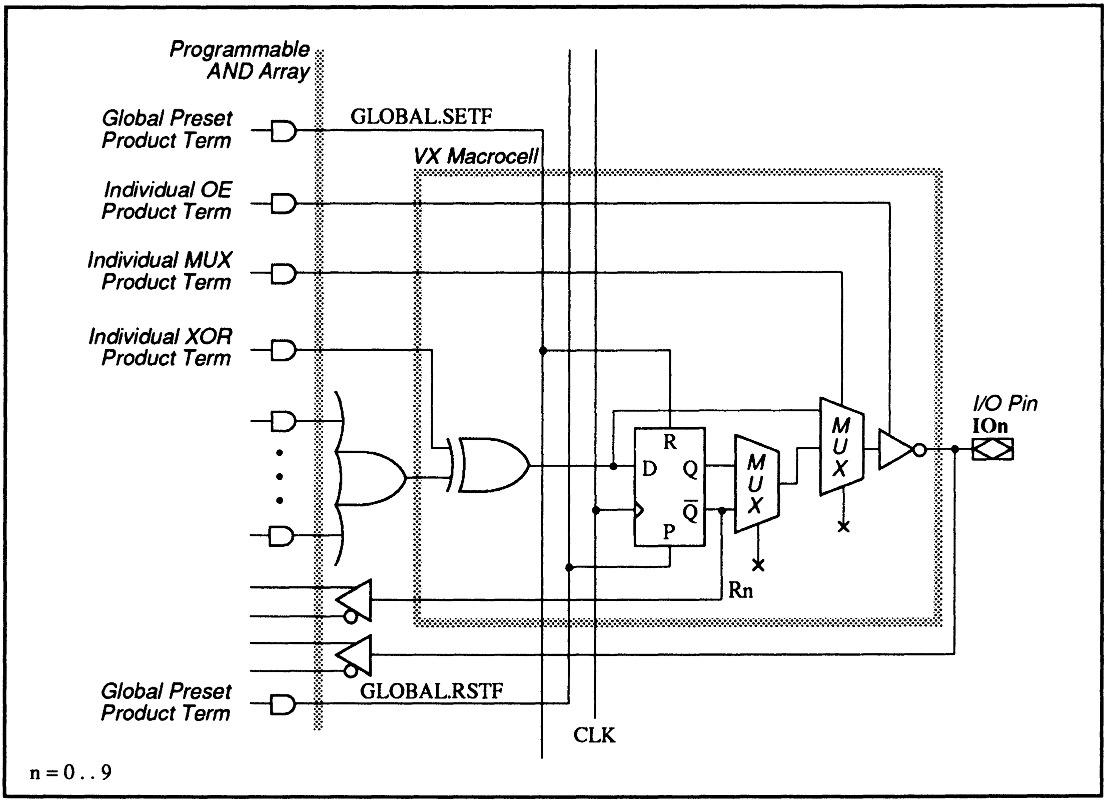
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



32VX10: Block Diagram Showing Pin and Macrocell Locations

32VX10: PAL32VX10


Programmable AND Array
Global Preset Product Term
GLOBAL.SETF
VX Macrocell
Individual OE Product Term
Individual MUX Product Term
Individual XOR Product Term
R
D
Q
Q̅
P
MUX
MUX
I/O Pin
IOn
Rn
Global Preset Product Term
GLOBAL.RSTF
CLK
n = 0 . . 9

32VX10: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Bypassable register
- Output with I/O feedback
- Output with /Q feedback
- Output with I/O and /Q feedback
- Buried /Q feedback

Bypassable Register

In each macrocell, there is a MUX product term available to control the dynamic multiplexing between the combinatorial and registered output. When the product term is asserted, the output register is bypassed. To program the MUX product term, use the PALASM syntax shown next.

Syntax

```
Pin Statement(s)  :
                  PIN  I/O_pin_location    I/O_pin_name
                  :
Equation(s)       I/O_pin_name.CMBF
                  :
             for dynamic bypassable register control
                  = Boolean expression with one product term
                  :
      or     for combinatorial output
                  = VCC
                  :
      or     for registered output
                  = GND
```

Note: The .CMBF functional equation overrides the storage type declaration in the pin statement.

Example

Dynamic bypassable register control

```
:
PIN   2      I1                ;input
PIN   3      I2                ;input
:
PIN   21     IO7               ;output
PIN   22     IO8               ;output
PIN   23     IO9               ;output
:
IO7.CMBF = VCC                 ;combinatorial output
IO8.CMBF = GND                 ;registered output
IO9.CMBF = I1 * /I2            ;dynamic MUX output
                               ;when (I1*/I2) is true, the output register is bypassed
```

Output with I/O Feedback

Each macrocell can be configured to have either a combinatorial or registered output with the output feeding back to the logic array. The language syntax for such configuration is shown next.

32VX10: PAL32VX10

Syntax

```
Pin Statement(s) PIN  I/O_pin_location    I/O_pin_location    Storage_type*

Equation(s)      I/O_pin_name** = Boolean expression
                 :
                 <Equation(s) using I/O_pin_name as feedback >
```

* *The .CMBF equation will override any storage_type declaration in the pin statement.*

** *When the output is combinatorial and active high, the XOR function is not allowed in the output equation.*

	Output Polarity	
Output Configuration	*Active-high*	*Active-low*
Combinatorial	*XOR function not used*	
Registered		

Example

Combinatorial and registered output with I/O feedback

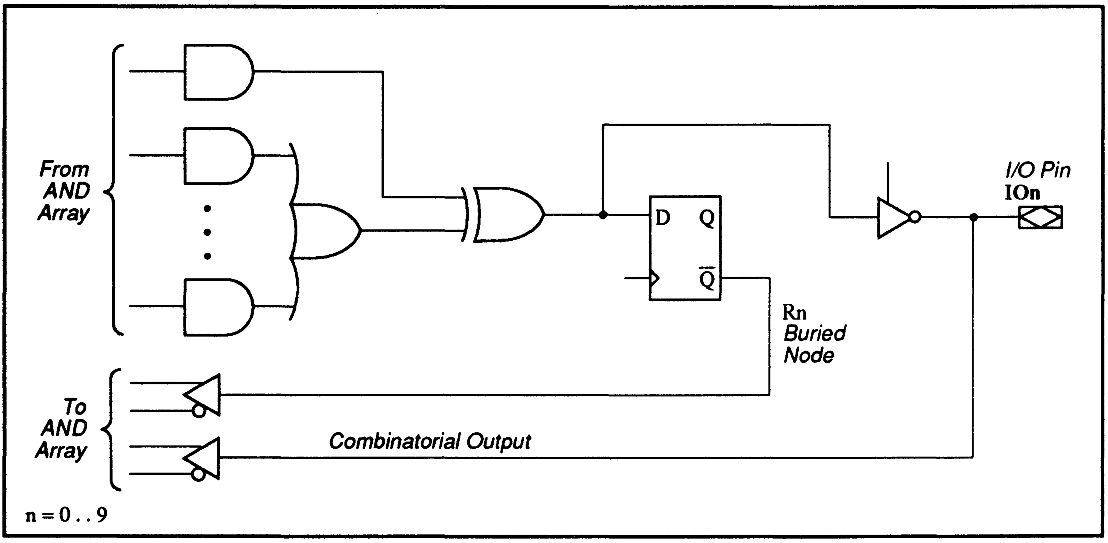
```
:
PIN     2    I1
PIN     3    I2
PIN     14   IO0
PIN     15   IO1
PIN     16   IO2
PIN     17   IO3
:
IO0.CMBF = VCC               ;combinatorial output
IO1.CMBF = VCC               ;combinatorial output
IO0 = I1 * I2                ;XOR not allowed in active-high combinatorial output
/IO1 = I1 :+: I2             ;XOR allowed in active-low combinatorial output
IO2.CMBF = GND               ;registered output
IO3.CMBF = GND               ;registered output
IO2 = / I2 :+:  I1           ;XOR allowed in active-high registered output
/IO3 = I2 :+:(/I1*IO0*/IO2)  ;XOR allowed in active-low registered output using
                             ;    combinatorial and registered outputs IO0 and
                             ;    IO2 as feedbacks
```

Output with /Q Feedback

Each macrocell can be configured to have either a combinatorial or registered output with the /Q output of the register feeding back to the logic array. The language syntax for this configuration is similar to the next configuration, output with /Q and I/O feedbacks. In this case only the /Q output and not the I/O is used in the output equations.

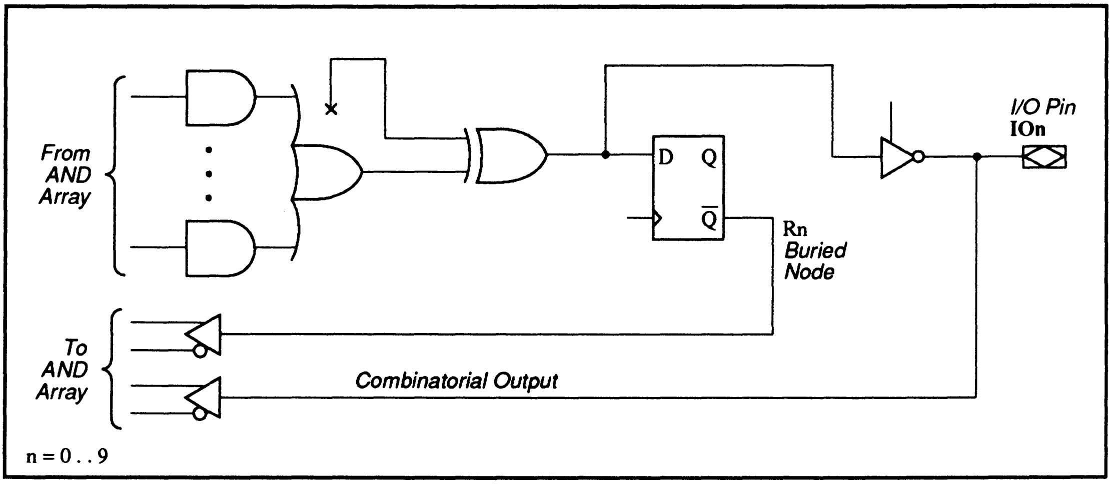
Output with I/O and /Q Feedbacks

Each macrocell can be configured to have either a combinatorial or registered output with both the /Q output of the register and the output itself feeding back to the logic array. The language syntax for both combinatorial and registered outputs is shown separately below.



32VX10: Combinatorial Output with /Q and I/O Feedback Where XOR Function is Available

32VX10: PAL32VX10



32VX10: Combinatorial Output with /Q and I/O Feedback Where XOR Function is Not Allowed

Syntax

Combinatorial output with /Q and I/O feedbacks

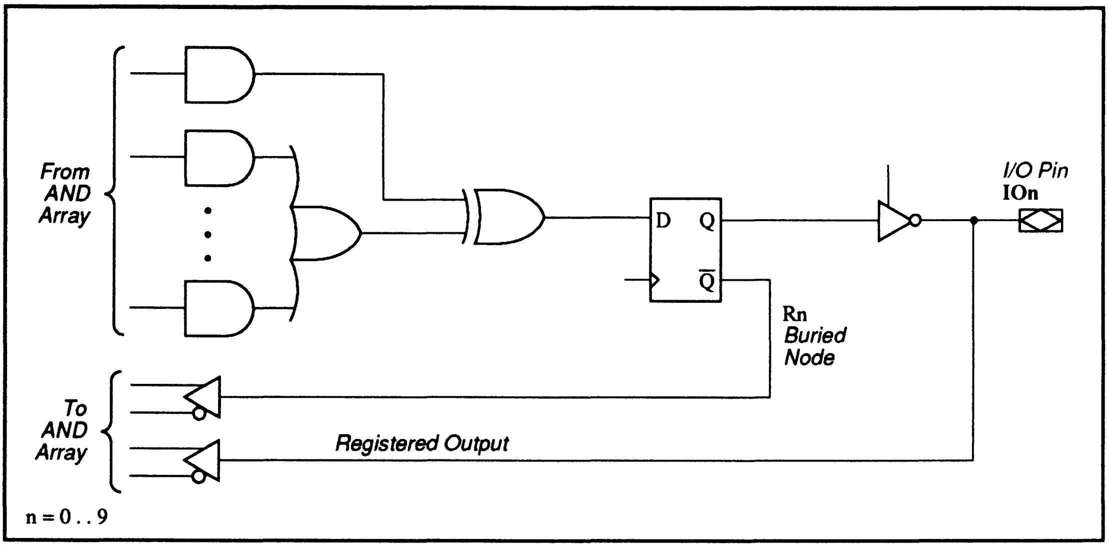
Pin Statement(s)	PIN I/O_pin_location I/O_pin_location COMB* NODE Buried_node_location Buried_node_name REG
Equation(s)	I/O_pin_name** = Boolean expression Buried_node_name*** = { I/O_pin_name } : <Equation(s) using Buried_node_name, or I/O_pin_name, or both as feedback(s)>

* *The .CMBF equation will override any storage_type declaration in the pin statement.*

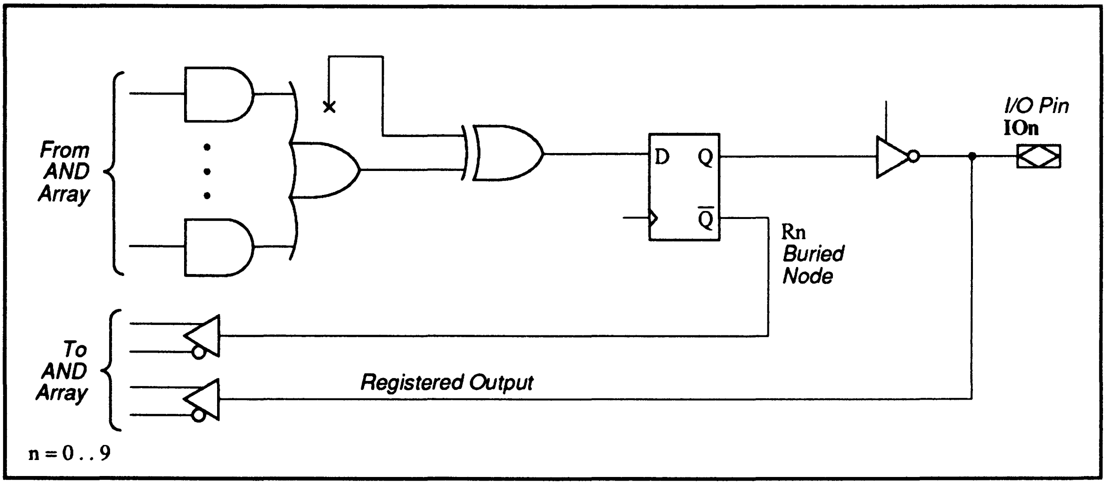
** *For combinatorial outputs, only active-high equations are allowed.*

*** *For combinatorial outputs, XOR function is only allowed when the output equation*
is active high and the Buried_node_name is active low.

	Combinatorial Output Polarity	
Buried_node_name polarity	Active-high	Active-low
Active-high		Illegal
Active-low	XOR allowed	Illegal



32VX10: Registered Output with /Q and I/O Feedback Where XOR Function is Available



32VX10: Registered Output with /Q and I/O Feedback Where XOR Function is Not Allowed

32VX10: PAL32VX10

Syntax

Registered output with /Q and I/O feedback

Pin Statement(s)	PIN I/O_pin_location I/O_pin_location REG* NODE Buried_node_location Buried_node_name REG
Equation(s)	I/O_pin_name = Boolean expression Buried_node_name** = { I/O_pin_name } : <Equation(s) using Buried_node_name, or I/O_pin_name, or both as feedback(s)>

* *The .CMBF equation will override any storage_type declaration in the pin statement.*

** *For registered outputs, XOR function is only allowed when the Buried_node_name is active low.*

	Registered Output Polarity	
Buried_node_name polarity	*Active high*	*Active low*
Active high		
Active low	*XOR allowed*	*XOR allowed*

Example

Combinatorial and registered output with /Q and I/O feedbacks

```
:
PIN     2    I1                    ;input
PIN     3    I2                    ;input
PIN     14   I00
PIN     15   I01
PIN     16   I02
PIN     17   I03
PIN     18   I04
NODE    2    R0     REG
NODE    3    R1     REG
NODE    4    R2     REG
:
I00.CMBF = VCC                     ;combinatorial output
I01.CMBF = VCC                     ;combinatorial output
I00 = I1 * I2                      ;XOR not allowed in active-high combinatorial output and
                                   ;    active-high buried node, R0
R0 = { I00 }
/I01 = I1 :+: I2                   ;XOR allowed in active-high combinatorial output and
                                   ;    active-low buried node, R1
/R1 = { I01 }
I02.CMBF = GND                     ;registered output
I03.CMBF = GND                     ;registered output
I02 = / I2 :+: I1                  ;XOR allowed in active-high registered output and
                                   ;    active-low buried node, R2
/R2 = { I02 }
/I03 = I2 :+:(/I1*I00*/R2)         ;XOR allowed in active-low registered output using
                                   ;    combinatorial output I01 and buried /Q output R2 as
                                   ;    feedbacks
/R3 = { /I03 }
I04.CMBF = VCC                     ;combinatorial output
I04 = I02 * / R1 * R3              ;using combinatorial output I02 and buried /Q outputs R1
                                   ;    and R3 as feedbacks
```

32VX10: PAL32VX10

Buried \Q Feedback

The output macrocell can be configured as a buried register with the /Q output feeding back to the logic array. The language syntax for this configuration is shown below.

Syntax

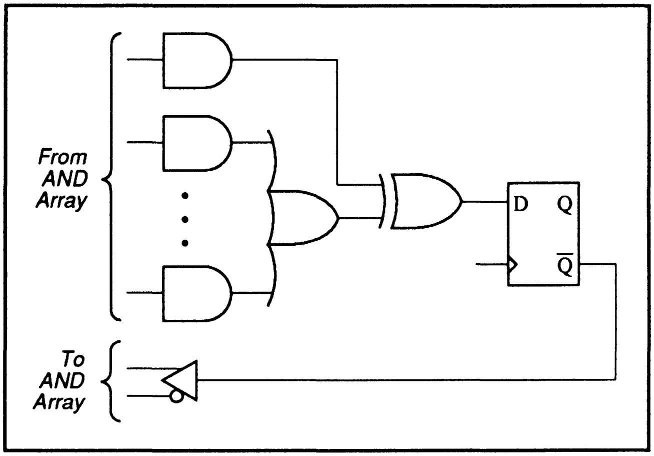
Pin Statement(s)	PIN Buried_node_location Buried_node_name REG
Equation(s)	Buried_node_name* = Boolean expression : < Equation(s) using Buried_node_name as feedback >

* *XOR function is only allowed in active-low buried node equations.*

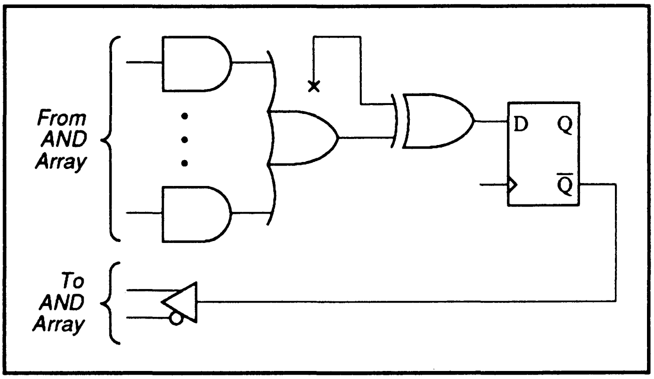
Example

Buried /Q output

```
:
PIN     2    I1                 ;input
PIN     3    I2                 ;input
PIN     16   I02
NODE    2    R0    REG
NODE    3    R1    REG
:
R0 = I1 * I2                    ;XOR not allowed in active-high node equation
/R1 = I1 :+: I2                 ;XOR allowed in active-low node equation
I02.CMBF = GND                  ;registered output
I02 = I1 * /R0 * /R1            ;using buried /Q feedbacks R0 and R1
```



32VX10: Buried /Q Feedback Where XOR Function is Available



32VX10: Buried /Q Feedback Where XOR Function is Not Allowed

610: PALCE610

PIN AND NODE DESCRIPTIONS

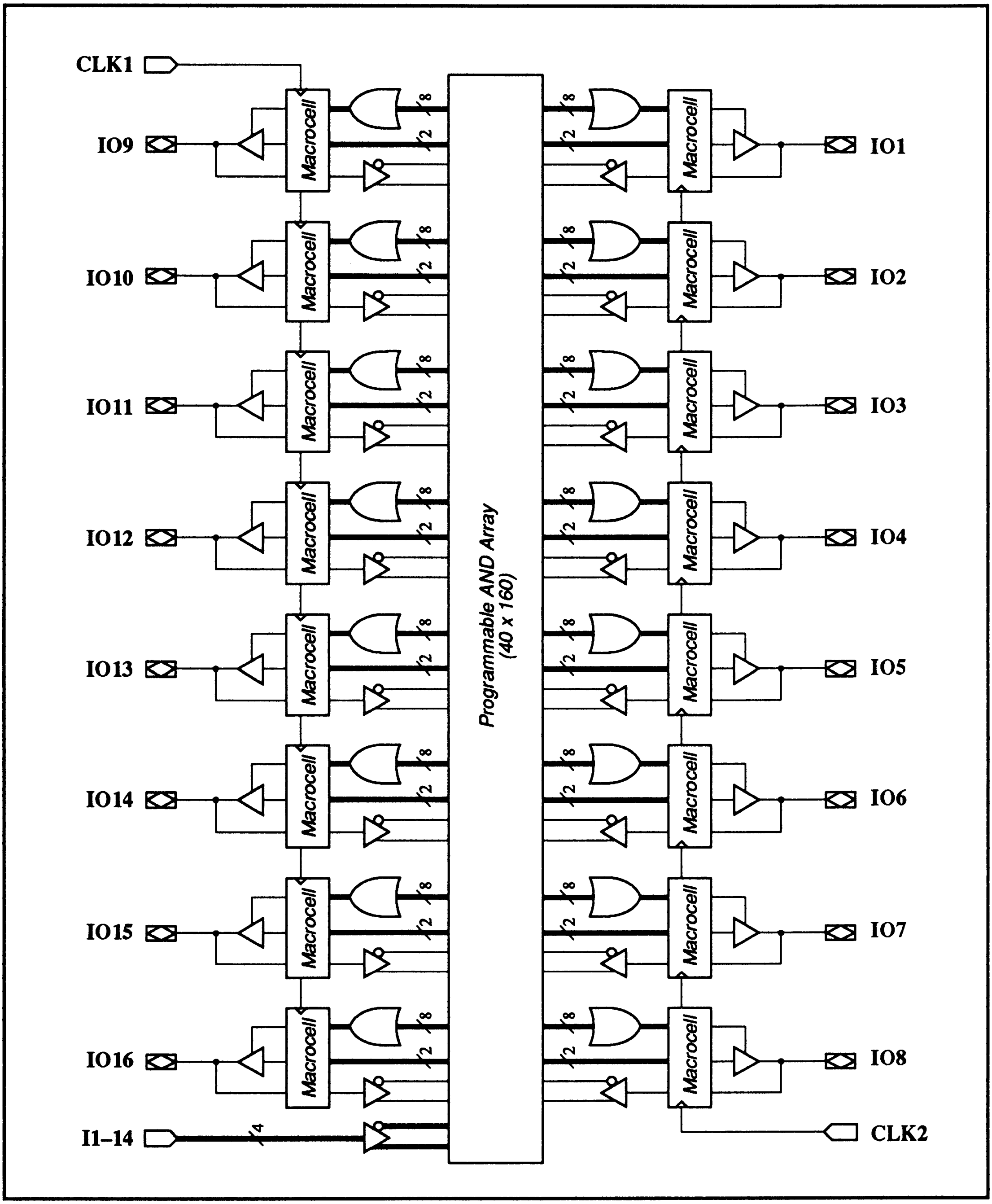
- 24 pins
- 16 buried nodes

610

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK1	–	–	–
2	I3	–	–	–
3 –10	IO9 –IO16	1 – 8	R9 – R16	Buried feedback
11	I4	–	–	–
12	GND	–	–	–
13	CLK2	–	–	–
14	I2	–	–	–
15 – 22	IO8 – IO1	9 – 16	R8 – R1	Buried feedback
23	I1	–	–	–
24	VCC	–	–	–

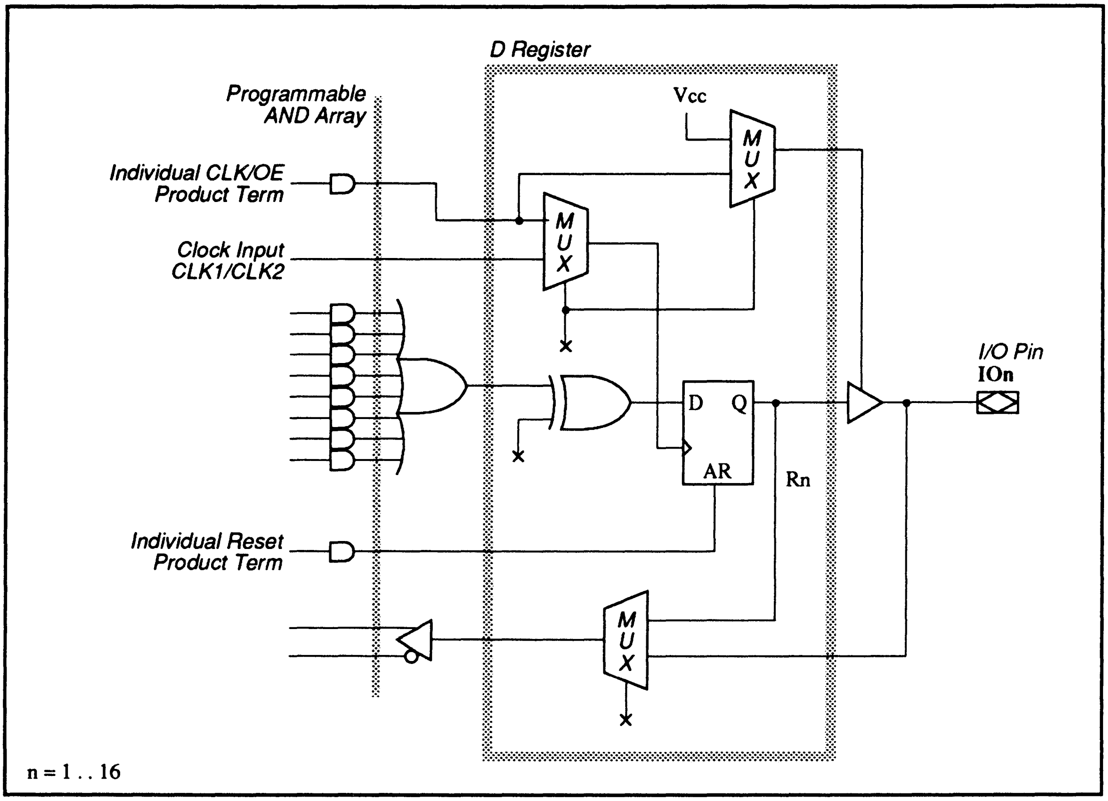
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

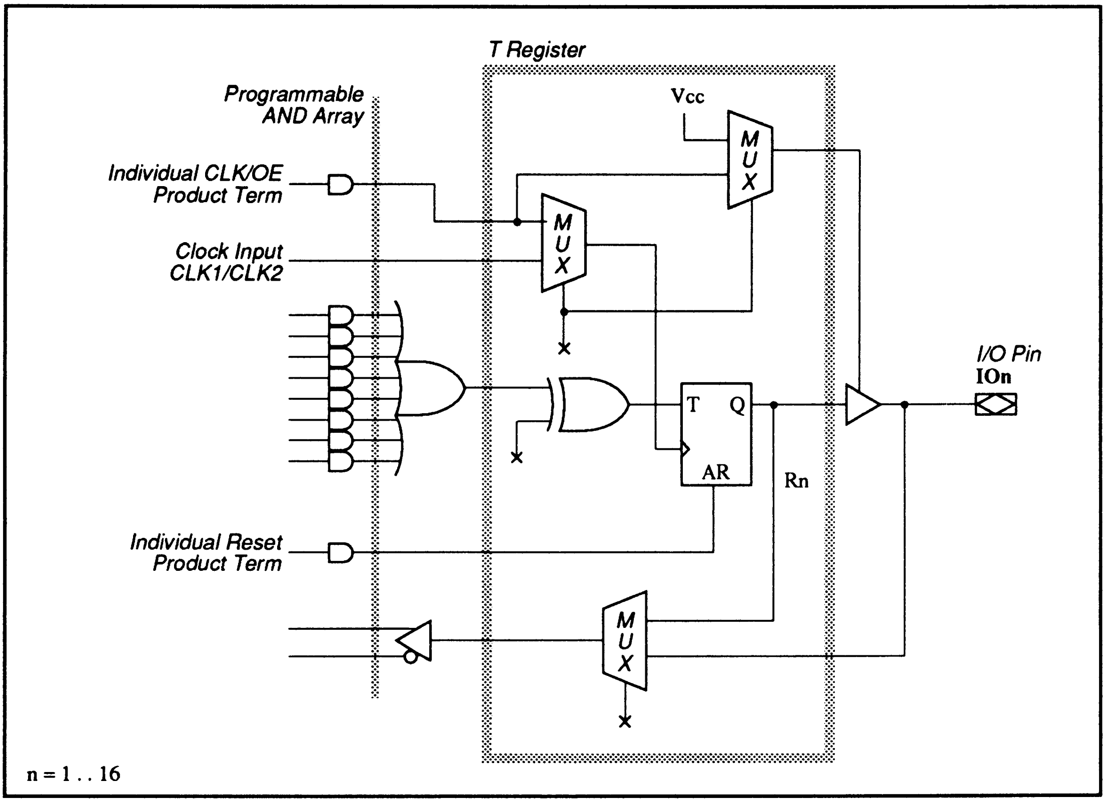


610: Block Diagram Showing Pin and Node Locations

610: PALCE610

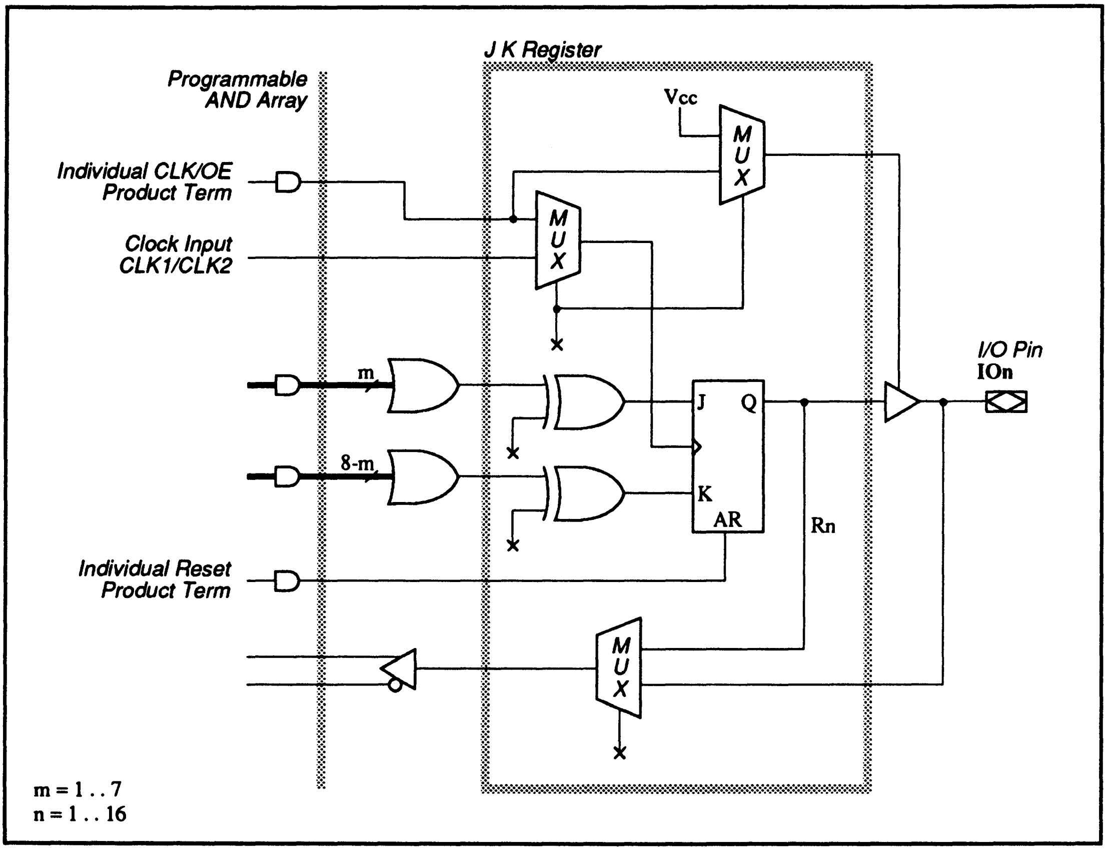


610: D Flip-Flop Macrocell Diagram

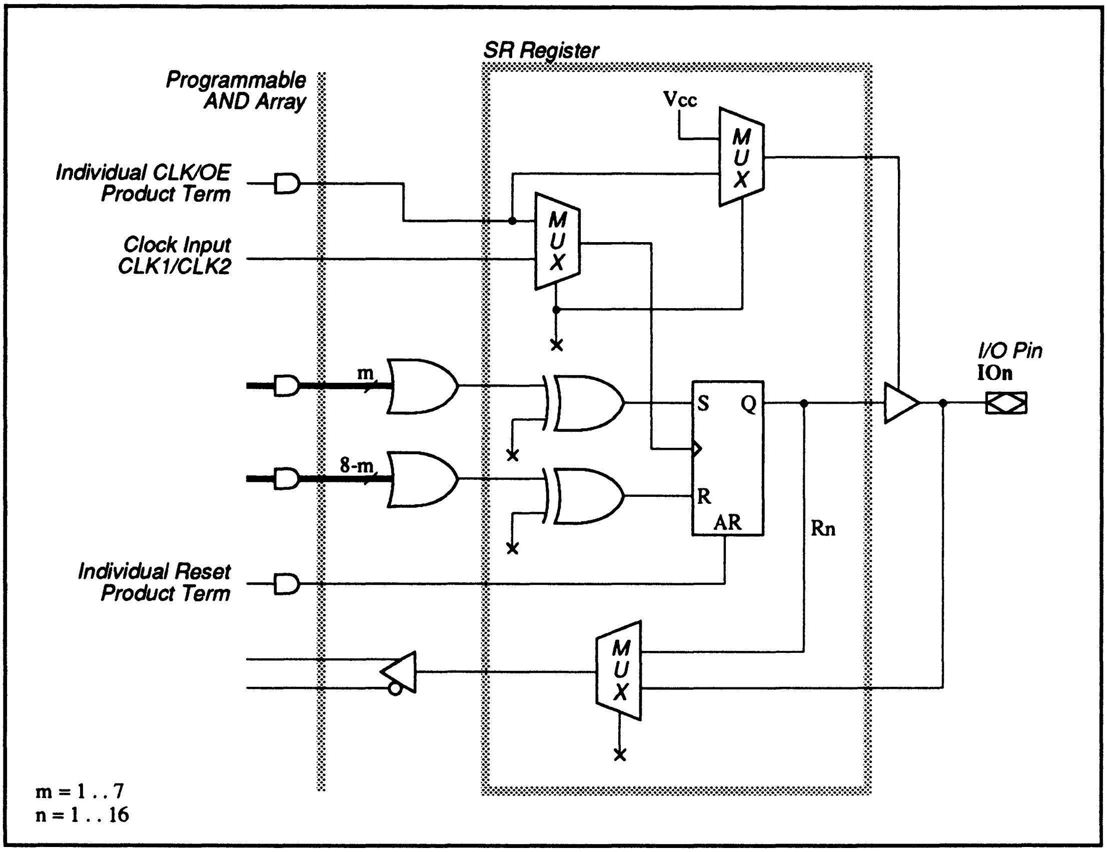


610: T Flip-Flop Macrocell Diagram

610: PALCE610



610: JK Flip-Flop Macrocell Diagram



610: SR Flip-Flop Macrocell Diagram

610: PALCE610

SPECIAL PROGRAMMING FEATURES

* Clock and output-enable product term control
* Register configurations
- Macrocells with different configurations
- Output with I/O feedback
- Output with Q feedback
- Buried register with Q feedback

Clock and Output-Enable Product Term Control

Each macrocell can select as its clock either the corresponding clock pin or the CLK/OE product term. If the clock pin is selected, the output enable is controlled by the CLK/OE product term. If the CLK/OE product term is selected to control the clock, the output is always enabled.

To select the CLK/OE product for the clock control, simply write the .CLKF equation for the corresponding output. To select the CLK/OE product term for output-enable control, write the .TRST equation instead. The syntax for both cases is shown next.

Syntax

```
Pin Statement(s)  PIN Output_pin_location Output_pin_name  Storage_type
                  :
Equation(s)  for clock control
                  Output_pin_name.CLKF = Boolean expression with one product term

     or      for output-enable control
                  Output_pin_name.TRST = Boolean expression with one product term
```

The table below summarizes how the PALASM 4 software interprets the .CLKF and .TRST equations when there is a conflict. For example, in case 3 for the same output, if the .CLKF equation exits and the .TRST equation also exits and is equal to VCC, then the clock is controlled by the CLK/OE product term, and the output enable is connected directly to VCC.

	.CLKF Equation	.TRST Equation	PALASM software's interpretation
Case 1 output	Exists	-	clock controlled by CLK/OE product term enable is enabled by VCC
Case 2	-	Exists	clock controlled by external clock pin output enable is controlled by CLK/OE product term
Case 3	Exists	Exists and = VCC	clock controlled by CLK/OE product term; output enable is enabled by VCC
Case 4	Exists	Exists and not = VCC	ERROR

Example 1

Using CLK/OE product term to control clock

```
:
PIN     2    I3           ;input
PIN     11   I4           ;input
:
PIN     3    I09   REG    ;output, registered
:
I09.CLKF = I3 * /I4       ;clock of I09's register is controlled by the product term
                          ;    (I3*/I4)
```

610: PALCE610

Example 2

Using CLK/OE product term to control output enable

```
:
PIN     2    I3            ;input
PIN     11   I4            ;input
:
PIN 3   I09  REG           ;output, registered
:
I09.TRST = I3 * /I4        ;output enable of I09's output buffer is controlled by
                           ;    product term (I3*/I4)
```

Register Configurations

Each register in a 610 can be configured as one of four types of registers with programmable polarity.

- D flip-flop
- T flip-flop
- JK flip-flop
- RS flip-flop

Syntax

```
Pin Statement(s)  :
                  PIN  I/O_pin_number    I/O_pin_name    REG
                  :

Equation(s)  for D flip-flop
                  I/O_pin_name*    = Boolean expression
             or   for T flip-flop
                  I/O_pin_name.T*  = Boolean expression
             or   for JK flip-flop
                  I/O_pin_name.J*  = Boolean expression
                  I/O_pin_name.K*  = Boolean expression
             or   for SR flip-flop
                  I/O_pin_name.R*  = Boolean expression
                  I/O_pin_name.S*  = Boolean expression
```

* *I/O_pin_name can be active high or active low.*

Example

Register configurations

```
:
PIN     23   I1                ;input
PIN     14   I2                ;input
PIN     3    I09    REG        ;output, registered
PIN     4    I010   REG        ;output, registered
PIN     5    I011   REG        ;output, registered
PIN     6    I012   REG        ;output, registered
:
/I09 = I1                      ;output equation, D FF, active low
I010.T = I2                    ;output equation, T FF, active high
/I011.S = I1 * /I2             ;output equation, S input of SRFF, active low
I011.R = /I1                   ;output equation, R input of SRFF, active high
/I012.J = I1 * /I2             ;output equation, J input of JKFF, active low
/I012.K = /I1                  ;output equation, K input of JKFF, active high
```

Macrocells with Different Configurations

The 610 output macrocell can be configured as one of the following types of output.

- Combinatorial
- D registered
- T registered
- JK registered
- RS registered

Each type of output provides a different set of feedback configurations, as tabulated next. Allowable configurations are marked with an X in the table.

610: PALCE610

FEEDBACK CONFIGURATIONS	COMBINATORIAL	D	T	JK	RS
Programmable Feedback					
Output with I/O feedback					
Combinatorial	X	N/A	N/A	N/A	N/A
Registered	N/A	X	X	N/A	N/A
Output with /Q feedback					
Registered	N/A	X	X	X	X
Buried register with /Q feedback	N/A	X	X	X	X

Output with I/O Feedback

If an output is programmed as combinatorial, it can be used as a feedback. If an output is programmed as registered, it can be used as a feedback only if the register is configured as either a D or T flip-flop. It is illegal to use the output as a feedback if the output is configured to be either a JK or SR flip-flop.

To use the output as an I/O with a feedback, just use the I/O pin name for the required output equation, no special language syntax is required.

Output with Q Feedback

The output macrocell can be configured as a registered output with the Q output of the register available as a feedback. If the output is configured as combinatorial, the Q feedback is not allowed.

To use the Q output of the register as a feedback while also using the register as an output, you must first write a transfer equation for the buried node then use the buried node name in the required equation, as shown next.

Syntax

```
Pin Statement(s) PIN  I/O_pin_location    I/O_pin_name    REG
                 :
                 NODE Buried_node_location    Buried_node_name    REG

Equation(s)  for D flip-flop
                 I/O_pin_name*        = Boolean expression
                 Buried_node_name**   = { I/O_pin_name*** }
             or  for T flip-flop
                 I/O_pin_name.T*      = Boolean expression
                 Buried_node_name.T** = { I/O_pin_name*** }
             or  for JK flip-flop
                 I/O_pin_name.J*      = Boolean expression
                 I/O_pin_name.K*      = Boolean expression
                 Buried_node_name.J** = { I/O_pin_name*** }
                 Buried_node_name.K** = { I/O_pin_name*** }
             or  for SR flip-flop
                 I/O_pin_name.R*      = Boolean expression
                 I/O_pin_name.S*      = Boolean expression
                 Buried_node_name.R** = { I/O_pin_name*** }
                 Buried_node_name.S** = { I/O_pin_name*** }
                 :
                 < Equation(s) using Buried_node_name as feedback >
```

* *I/O_pin_name can be active high or active low.*

* *Buried_node_name must be the same polarity as the I/O_pin_name defined in the output equation.*

*** *I/O_pin_name inside curly bracket must be the same polarity as the I/O_pin_name defined in the output equation.*

610: PALCE610

Example

Registered output with Q feedback

```
:
PIN     23   I1    COMB  ;input
PIN     14   I2    COMB  ;input
PIN     22   I01   REG         ;I/O, registered
:
NODE    16   R1                ;buried node
:
/I01.R = /I1 * I2              ;equation for R input of register I01 with active-low
                               ;    input
I01.S = /I1 * I2               ;equation for S input of register I01 with active-high
                               ;    input
/R1.R = {/I01.R}               ;buried node has same polarity as I01.R
R1.S = { I01.S}                ;buried node has same polarity as I01.S
I02.T = I2* / I1*R1            ;equation for T input I02 with R1 as feedback
```

Buried Register with Q Feedback

The register in the output macrocell can be configured as a buried register. To use the Q output of the buried register as a feedback, just write an equation for the buried node and use the buried node name in the required equation, shown next.

Syntax

```
Pin Statement(s)  NODE Buried_node_location    Buried_node_name    REG
Equation(s)   for D flip-flop
                  Buried_node_name.D* = Boolean expression
              or  for T flip-flop
                  Buried_node_name.T* = Boolean expression
              or  for JK flip-flop
                  Buried_node_name.J* = Boolean expression
                  Buried_node_name.K* = Boolean expression
              or  for SR flip-flop
                  Buried_node_name.R* = Boolean expression
                  Buried_node_name.S* = Boolean expression
                  :
                  < Equation(s) using Buried_node_name as feedback >
*  Buried_node_name can be active-high or active-low
```

Example

Buried register with Q feedback

```
:
PIN     23   I1    COMB      ;input
PIN     14   I2    COMB      ;input
PIN     21   IO2   COMB      ;output, combinatorial
:
NODE    16   R1    REG       ;buried node of buried register
:
/R1.J = /I2 * I1             ;equation for J input of register IO1
/R1.K = I2 * IO2             ;equation for K input of register IO1
IO2 = I2* / I1*R1            ;equation for IO2 with R1 as feedback
```

11.5 MACH 1 AND MACH 2 SERIES DEVICES

This topic provides device and language information related to programming the MACH 1 and 2 series devices.

- The overview, 11.5.1, introduces the features.
- The discussion on sample equations, 11.5.2, provides examples of how to use the PALASM language syntax to configure macrocells.

> **Important**: Unless otherwise stated, all discussions in this chapter pertain to all MACH 1 and 2 series devices.
>
> **Note:** The term MACH 1 series refers to the MACH 110, 120, and 130 devices. The term MACH 2 series refers to the MACH 210, 220, and 230 devices.

11.5.1 OVERVIEW

This overview includes three discussions.

- 11.5.1.1, Device Features
- 11.5.1.2, Pin and Node Descriptions
- 11.5.1.3, PALASM Programming Features

11.5.1.1 Device Features

The MACH 1 and 2 series devices are similar; however, the MACH 2 devices are larger and contain both output and buried macrocells.[17]

- MACH 1 series devices are generally best suited for I/O-intensive designs.
- MACH 2 series devices are generally best suited for logic-intensive designs.

[17] Refer to the *AMD High Density EE CMOS Programmable Logic MACH™ Devices Data Book* for details about each device.

11.5 MACH 1 AND MACH 2 SERIES DEVICES

This topic provides device and language information related to programming the MACH 1 and 2 series devices.

- The overview, 11.5.1, introduces the features.
- The discussion on sample equations, 11.5.2, provides examples of how to use the PALASM language syntax to configure macrocells.

Important: Unless otherwise stated, all discussions in this chapter pertain to all MACH 1 and 2 series devices.

Note: The term MACH 1 series refers to the MACH 110, 120, and 130 devices. The term MACH 2 series refers to the MACH 210, 215, 220, and 230 devices.

11.5.1 OVERVIEW

This overview includes three discussions.

- 11.5.1.1, Device Features
- 11.5.1.2, Pin and Node Descriptions
- 11.5.1.3, PALASM Programming Features

11.5.1.1 Device Features

The MACH 1 and 2 series devices are similar; however, the MACH 2 devices are larger and contain both output and buried macrocells.[17]

- MACH 1 series devices are generally best suited for I/O-intensive designs.
- MACH 2 series devices are generally best suited for logic-intensive designs.
- The MACH215 is an ayschronous device.

[17] Refer to the *AMD High Density EE CMOS Programmable Logic MACH™ Devices Data Book* for details about each device.

This difference in architecture also affects product-term steering.[18]

The following table summarizes the I/O, block, and macrocell features of each device.

	MACH DEVICE						
FEATURE	110	120	130	210	220	230	215
Input Pins	4	8	6	4	8	6	4
Clocks/Input Pins	2	4	4	2	4	4	2*
Input/Output Pins	32	48	64	32	48	64	32
Blocks	2	4	4	4	8	8	4
Output Macrocells	32	48	64	32	48	64	32
Buried Macrocells	0	0	0	32	48	64	32**

* The MACH215 has dedicated clock pins. All other input and I/O pins can drive PT clocks.

** The MACH215 buried macrocells are available for input register use only.

11.5.1.2 Pin and Node Descriptions

You can specify the pin or node location of a signal in the design file.[19] Following discussions provide illustrations of available pins and nodes for each device.

11.5.1.3 PALASM Programming Features

Discussions below illustrate the PALASM programming features for the MACH 1 and 2 series devices, and are divided as follows.

- Fixing Pin and Node Locations
- Pairing Pins and Nodes
- Steering Product Terms

[18] Refer to the discussion on steering product terms, in this chapter, for more information.

[19] Refer to Section II, Chapter 4, for details about the various design-entry methods supported for MACH-device designs. Also, refer to Chapter 10, in this section, for details about MACH-specific syntax.

- Steering Product Terms
- Splitting Functions
- Programming Flip-Flop Types
- Optimizing with D- or T-Type Flip-Flops
- Defining Preset and Reset
- Defining Clock Pins
- Defining Output Enable

Fixing Pin and Node Locations

You can assign **fixed** pin and node locations or leave them **floating**.

- You **float** a pin or node by placing a question mark, ?, in the location field of a pin or node statement, as shown below.

```
PIN      ?    <pin name>
NODE     ?    <node name>
```

- Or, you can force all locations to float using the corresponding command on the MACH Fitting Options form.

Each floating pin and node is assigned a location during compilation and fitting. The location is based on optimal use of the device.

Recommendation: It is best to float all pin and node locations.

Also: If you float some locations and fix others, a no-fit situation may occur. Consider whether your design is close to the device-resource limit. If it is, you may have to sacrifice fixed assignments to fit the design in the device.

- You assign a **fixed** pin or node location by specifying a number in the location field of a pin or node statement using the following syntax.

```
PIN      3    <pin name>
NODE     2    <node name>
```

Important: **Fixed** locations may limit available resources, which affects **both** product-term steering **and** gate splitting.[20]

The group statement allows you to assign pins and nodes to a specific block. However, the only way to preserve the order is to assign fixed locations. The following example assigns three pins and three nodes to block A of a MACH device, using the reserved word MACH_SEG_A as a group name.[21]

```
GROUP MACH_SEG_A IO0 IO1 IO2 A0 A1 A2
```

Pairing Pins and Nodes

You can use the optional Pair attribute in a pin or node statement to direct input or output pairing manually.[22] Pairing can save resources when pins and nodes are left floating. If a node and a pin have the same equation, pairing them eliminates the need to generate the same equation twice.

- **Input** pairing includes the Pair attribute in a pin statement to associate an input pin with a node logically.

  Input pairing can only be implemented in MACH 2 series devices since they have buried macrocells.

- **Output** pairing includes the Pair attribute in a node statement to associate a node with an output pin logically.

  Output pairing can be implemented in all MACH devices.

[20] Refer to discussions on steering product terms and splitting gates, in this chapter, and to Section II, Chapter 5, for more information.

[21] Refer to Chapter 10, in this section, for details about the group name MACH_SEG_*block*.

[22] Refer to Chapter 10, in this section, for information on input and output pairing.

The keywords OPAIR and IPAIR are also valid and denote output and input pairing, respectively. To enable manual pairing, enter the letter N beside the Use automatic pin/node pairing field of the Logic Synthesis Options form.

Recommendation: Use the default setting that allows the software to establish pairs automatically.

Steering Product Terms

Four product terms are available to each macrocell. Product-term steering is an automatic software process that borrows additional terms from **unused** adjacent macrocells.

Note: Product terms can be borrowed only in groups of four, so an equation that requires five product terms actually consumes two groups of four.

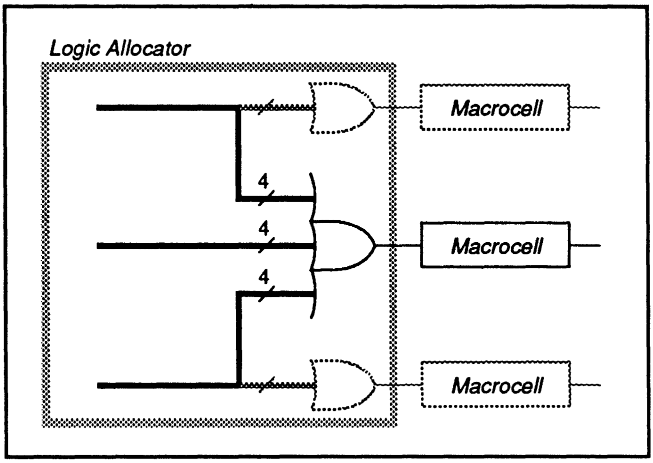
In MACH 1 series designs, up to eight product terms can be borrowed from adjacent macrocells for a total of 12. Four product terms can be borrowed from the adjacent macrocells above and below. However, the end macrocells in a block are limited to four borrowed product terms. For a MACH 110 device, an end macrocell occurs at the boundary of each bank of eight cells. For example, for block A, the nodes 2, 9, 10, and 17 originate in end macrocells. For a MACH 130 device, an end macro cell occurs at the end of each block of 16 cells. For example, nodes 2 and 17 originate in end macrocells.

In MACH 2 series designs, up to 12 product terms can be borrowed from adjacent macrocells for a total of 16. Four product terms can be borrowed from the one adjacent macrocell above and eight from the two adjacent macrocells below. However, the end macrocells in a block are limited to four or eight borrowed product terms, depending on whether it is at the bottom or top of the block, respectively. For a MACH 210 device, an end macrocell occurs at the boundary of each block of 16 macrocells. For example, for block A, the nodes 2 and 17 originate in end macrocells.

Note: Product-term steering does not result in additional delays in the signal path.

Reminder: In product-term steering, the first and last macrocells of a block cannot use product terms from the first or last macrocell of an adjacent block.

The figure below illustrates full product-term steering for a MACH 1 series device.



MACH 1 Series Full PT Steering

The next figure shows an example of full product-term steering for a MACH 2 series device. In this case, four product terms are borrowed from the macrocell above and four each from the two macrocells below.

Each loop back through the array results in an additional level of delay in the signal path.

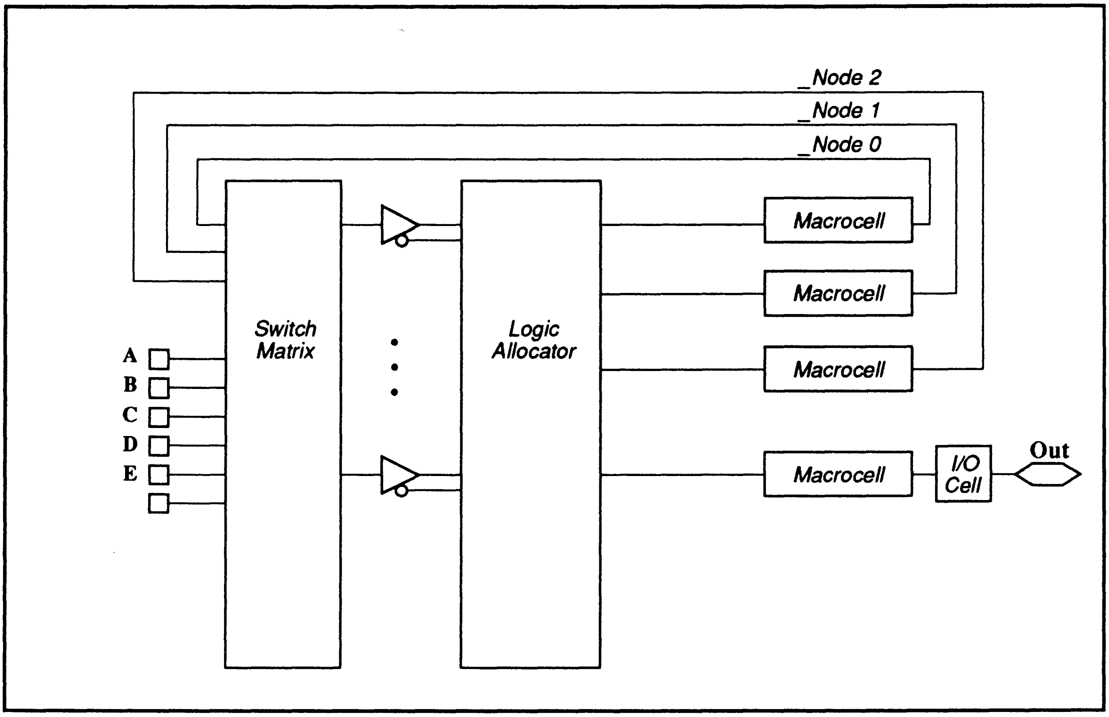
By default, the fitting process first attempts product-term steering. If this cannot be accomplished, gate splitting is used.

You enable automatic gate splitting by entering a Y in the Use automatic gate splitting field and defining the cluster size in the Max= field of the Logic Synthesis Options form.

Simulation shows no difference between product-term steering and gate splitting in waveform or trace files; it does not show propagation delays or gate widths.[23]

The next figure illustrates a design with 16 product terms implemented with automatic gate splitting enabled and a maximum cluster size of four.

[23] Refer to Section II, Chapter 6, for details about simulation.



Gate Splitting For MACH 110

Pin Declaration Segment

PIN	?	A	;INPUT
PIN	?	B	;INPUT
PIN	?	C	;INPUT
PIN	?	D	;INPUT
PIN	?	E	;INPUT
PIN	?	OUT	;OUTPUT

Equation Segment

OUT = A :+: B :+: C :+: D :+: E

After automatic gate splitting, equations are generated for the exclusive-OR function, as shown next. You can view these equations by compiling the design, disassembling the intermediate file, and then viewing the <design>.pl2 file.

Equation Segment

```
OUT     =   /A * B * /C * D * E
        +   /A * B * C * /D *E
        +   /A * B * C * D * /E
        +   A * B * C * D * E
        +   _NODE0
        +   _NODE1
        +   _NODE2
_NODE0  =   A * B * C * /D * /E
        +   A * B * /C * D * /E
        +   A * B * /C * /D * E
        +   A * /B * C * D */E
_NODE1  =   A * /B * C * /D * E
        +   A * /B * /C * D * E
        +   A * /B * /C * /D * /E
        +   /A * /B * C * D * E
_NODE2  =   /A * /B * C * /D * /E
        +   /A * /B * /C * /D * E
        +   /A * /B * /C * D * /E
        +   /A * B * /C * /D * /E
```

Programming Flip-Flop Types

Output macrocells are individually defined as either combinatorial, latched,[24] or registered by using the optional storage attribute in a pin or node statement.

If you specify registered output, the flip-flop can be programmed as **either** a D-type **or** T-type. D-type is the default. The syntax for both types is shown below.

```
IO1   = Boolean expression          ;D-type is default
IO1.T = Boolean expression          ;T-type is specified
```

Optimizing with D- or T-Type Flip-flops

The software can determine whether the design could be implemented more efficiently using one type flip-flop over the other. To accomplish this, you specify the following on the Logic Synthesis Options form.

```
Optimize registers for D/T type          Best type for device
```

The software compares the minimized results of a D-type and T-type implementation. The result with the lowest resource usage is chosen for each equation.

[24] Macrocell latch has an active-low latch enable input.

Important: You can also specify the flip-flop type in an equation. However, the Optimize registers for D/T-type option overrides the design file specifications. This option allows you to use the type(s) specified in the design file, change all to D, change all to T, or use the best type for the device.

Defining Preset and Reset

Each block in a MACH device has a single asynchronous set and reset line. You can enable each preset and reset at the macrocell level using a .SETF or .RSTF equation. For flip-flop and latch[25] configurations, both set and reset definitions must be provided to clear the "No Set/Reset initialization function found!" warning in the report file.

Note: MACH devices provide programmable polarity between the macrocell output and the pin. If you select the Ensure polarity after minimization is Best For Device option on the Logic Synthesis Options form, the logic polarity at the pin may be inverted.

To assign a preset or reset signal to the entire device, use node 1 to define the global asynchronous preset and reset product terms. In this case, you can assign a descriptive name, such as GLOBAL, to node 1, then write either the desired preset or reset equation using the following general form.

Important: You must specify node 1 in the pin declarations segment if you use the global set or reset features. When back annotating, make sure the global node has not been converted to floating.

Recommendation: Keep set/reset functions simple. Each input for these functions uses an array input that would otherwise be available for other logic.

[25] This applies to MACH 2 series latches with active-low latch inputs only.

Preset

```
NODE 1 GLOBAL
GLOBAL.SETF = <preset equation>
```

Reset

```
NODE 1 GLOBAL
GLOBAL.RSTF = <reset equation>
```

The example below initializes all registers to 1, high, when inputs I1 = 1, I2 = 0, and I3 = 1, for positive polarity in the pin and equation declaration segments.

```
GLOBAL.SETF = I1 * /I2 * I3
```

The preset and reset functions are asynchronous; therefore, the registers are initialized independent of the clock.

Defining Clock Pins

The following table summarizes the clock pins available for each MACH device.

MACH 110 AND 210	MACH 120 AND 220	MACH 130 AND 230
CLK0, pin 13 CLK1, pin 35*	CLK0, pin 15 CLK1, pin 17 CLK2, pin 49 CLK3, pin 50*	CLK0, pin 20 CLK1, pin 23 CLK2, pin 62 CLK3, pin 65*

* Default clock if you do not specify a clock pin

Defining Output Enable

Each I/O cell has a three-state buffer that can be

- permanently enabled, or
- permanently disabled, or
- controlled by a product term.

The following output-enable equation examples show the three types of output enables.

Permanently Enabled

```
<pin name>.TRST          = VCC   ;
```

Preset

NODE 1 GLOBAL
GLOBAL.SETF = <preset equation>

Reset

NODE 1 GLOBAL
GLOBAL.RSTF = <reset equation>

The example below initializes all registers to 1, high, when inputs I1 = 1, I2 = 0, and I3 = 1, for positive polarity in the pin and equation declaration segments.

GLOBAL.SETF = I1 * /I2 * I3

The preset and reset functions are asynchronous; therefore, the registers are initialized independent of the clock.

Defining Clock Pins

The following table summarizes the clock pins available for each MACH device.

MACH 110 AND 210	MACH215	MACH 120 AND 220	MACH 130 AND 230
CLK0, pin 13 CLK1, pin 35*	CLK0, pin 13* CLK1, pin 35	CLK0, pin 15 CLK1, pin 17 CLK2, pin 49 CLK3, pin 50*	CLK0, pin 20 CLK1, pin 23 CLK2, pin 62 CLK3, pin 65*

* Default clock if you do not specify a clock pin

Defining Output Enable

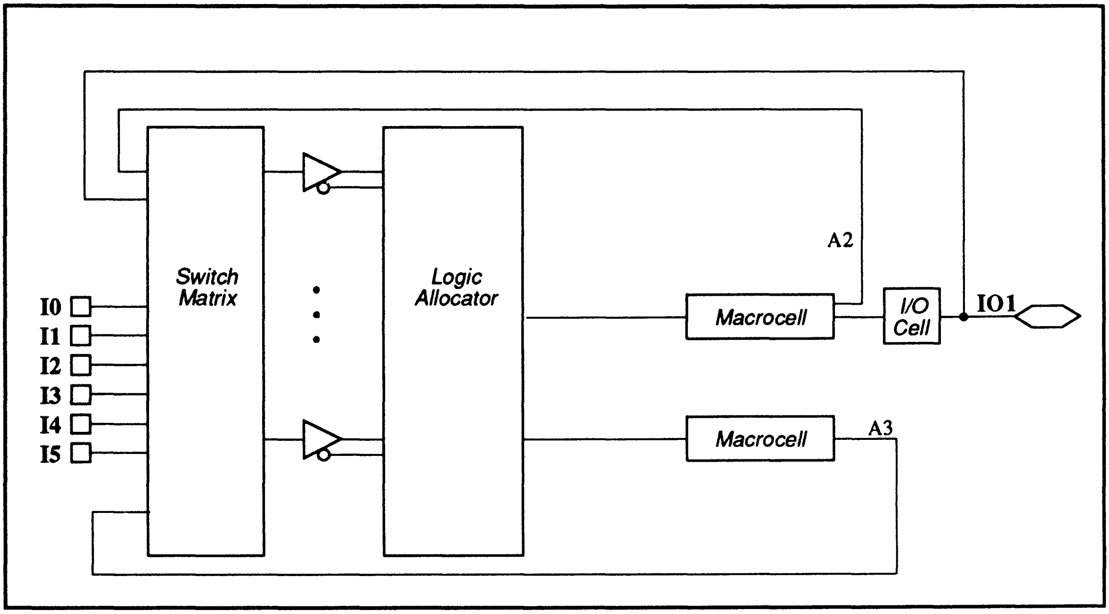
Each I/O cell has a three-state buffer that can be

- permanently enabled, or
- permanently disabled, or
- controlled by a product term.

The following output-enable equation examples show the three types of output enables.

Permanently Enabled

```
<pin name>.TRST          = VCC   ;
```



MACH Resource Organization

11.5.2.1 I/O Cell and Macrocell

You can choose one of the following types of configurations for a pin, output macrocell node, or buried macrocell node.

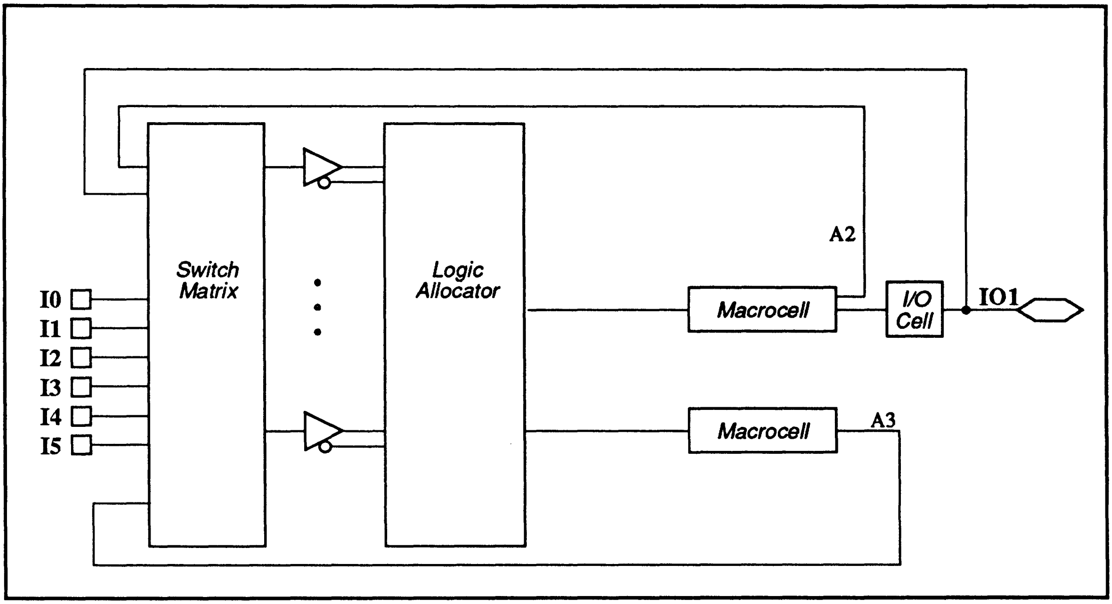
- Combinatorial
- Latched (active-low latch input)
- D-registered
- T-registered

You specify combinatorial, latched, or registered in the pin declaration segment of the design file, as follows.

PIN DECLARATION SEGMENT

```
Pin     4    IO2    COMBINATORIAL
Node    5    A3     REGISTERED
Pin     5    IO3    LATCHED
```

For **registered** nodes and pins, you must also specify the type of register in the equations segment of the design file, as follows.



MACH Resource Organization

11.5.2.1 I/O Cell and Macrocell

You can choose one of the following types of configurations for a pin, output macrocell node, or buried macrocell node.

- Combinatorial
- Latched (active-low latch input)
- D-registered
- T-registered

You specify combinatorial, latched, or registered in the pin declaration segment of the design file, as follows.

PIN DECLARATION SEGMENT

```
Pin      4    IO2   REGISTERED
Node     5    A3    REGISTERED
Pin      5    IO3   LATCHED
```

For **registered** nodes and pins, you must also specify the type of register in the equations segment of the design file, as follows.

EQUATIONS SEGMENT

```
IO2 = I1 * I2 * I3          ;D-type, active high
IO2.D = I1 * I2 * I3        ;D-type, active high
IO2.T = I1 * I2 * I3        ;T-type, active high
```

Note: For a T-registered signal, you must use the .T suffix in string definitions. For example:

```
A6.T: = {IO2.T}
```

11.5.2.2 Pin and Node Feedback

A MACH device allows you to specify feedback from a pin or a node. The following examples illustrate equations for I/O pin, output macrocell, and buried macrocell feedback.

Important: Since the MACH device emulates 22V10 behavior, feedback is normally taken from the register instead of the I/O pin, unless you specify otherwise. To realize pin feedback for a **registered** or **latched** output, you must specify an equation for the associated node, even if you do not use the node in the design. If you do not define the node, the configuration will default to feedback from the node. However, this is not true if the output pin is IPAIRed, in which case feedback defaults to I/O pin feedback, without the necessity of writing a dummy equation.

To realize pin feedback for a **combinatorial** output, you do not need to specify a node equation.

I/O Pin Feedback

The examples below are divided into two categories: combinatorial and registered, or latched outputs.

Combinatorial Output

```
IO1 = I1 * IO2 * I3         ;feedback from pin IO2
```

Registered or Latched Outputs

PIN DECLARATION SEGMENT

```
Pin     ?   IO1   REG
Node    ?   A4    PAIR    IO1     REG
```

EQUATIONS SEGMENT

```
IO1     =   I1 * IO2 * I3     ;feedback from pin IO2
A4      =   I1 * IO2 * I3     ;mandatory node eqn.
```

Node Feedback

You specify feedback from an output macrocell or buried macrocell node as shown below.

Output Macrocell Node

```
IO1 = I1 * A2 * I3        ;feedback from output macrocell
                          ; node A2
```

Buried Macrocell Node

```
IO1 = I1 * A3 * I3        ;feedback from buried macrocell
                          ;node A3
```
[27]

11.5.2.3 Registered and Latched Inputs

The following examples illustrate equations for latched and registered inputs.[28]

Registered Inputs

PIN DECLARATION SEGMENT

```
Pin     33  I4    PAIR A3
Node    5   A3    REGISTERED
```

EQUATION SEGMENT

```
A3 = I4             ;D-type registered input
```

Note: T-type registered inputs are not supported.

Latched Inputs (active-low latch input)

PIN DECLARATION SEGMENT

[27] This applies to MACH 2 series devices only.

[28] This applies to MACH 2 series devices only.

```
Pin     33    I4    PAIR A3
Node    5     A3    LATCHED
```

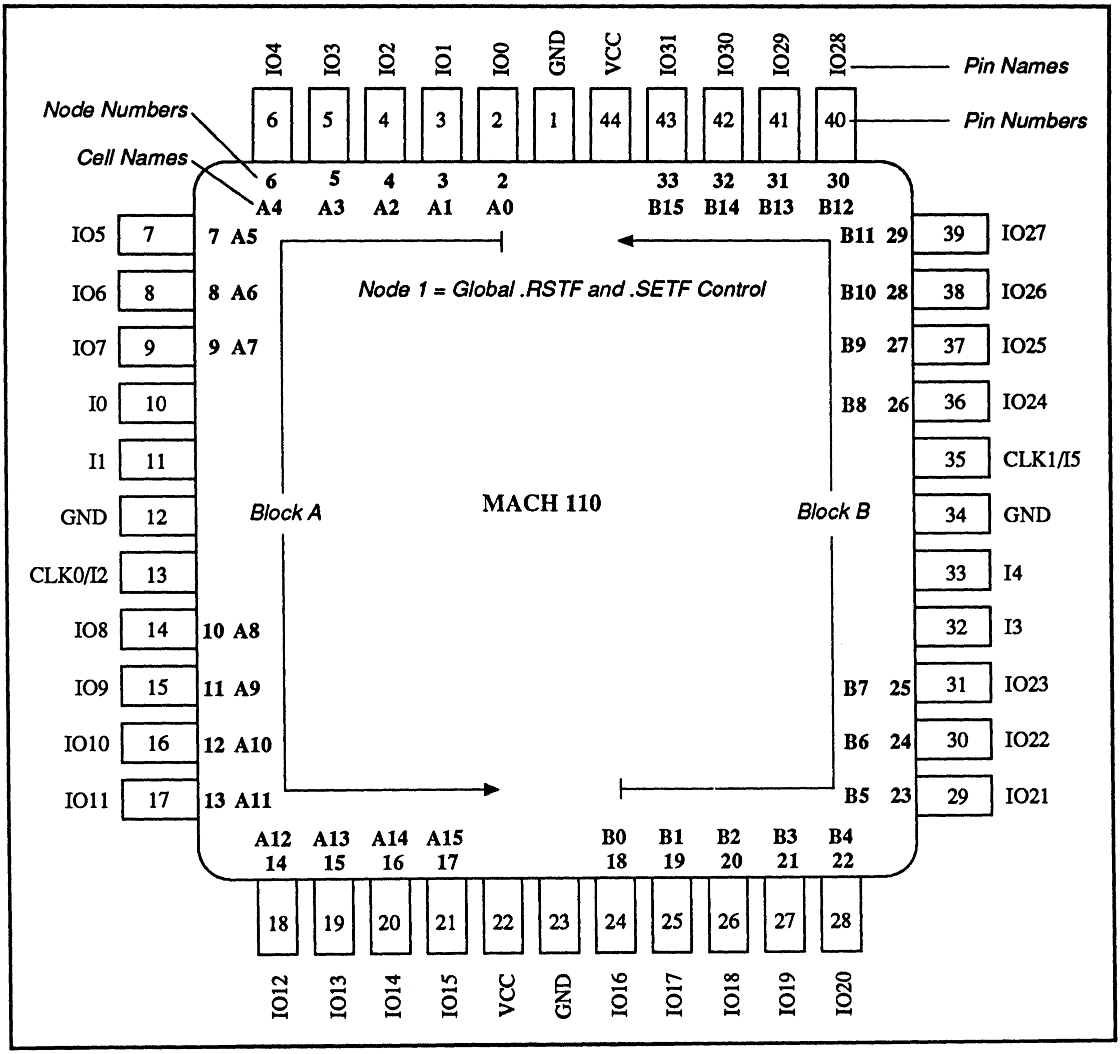
EQUATION SEGMENT

```
A3 = I4            ;latched input
```

MACH 110 DEVICE

An illustration of the MACH 110 node numbers and cell names is shown next. Each I/O pin in the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

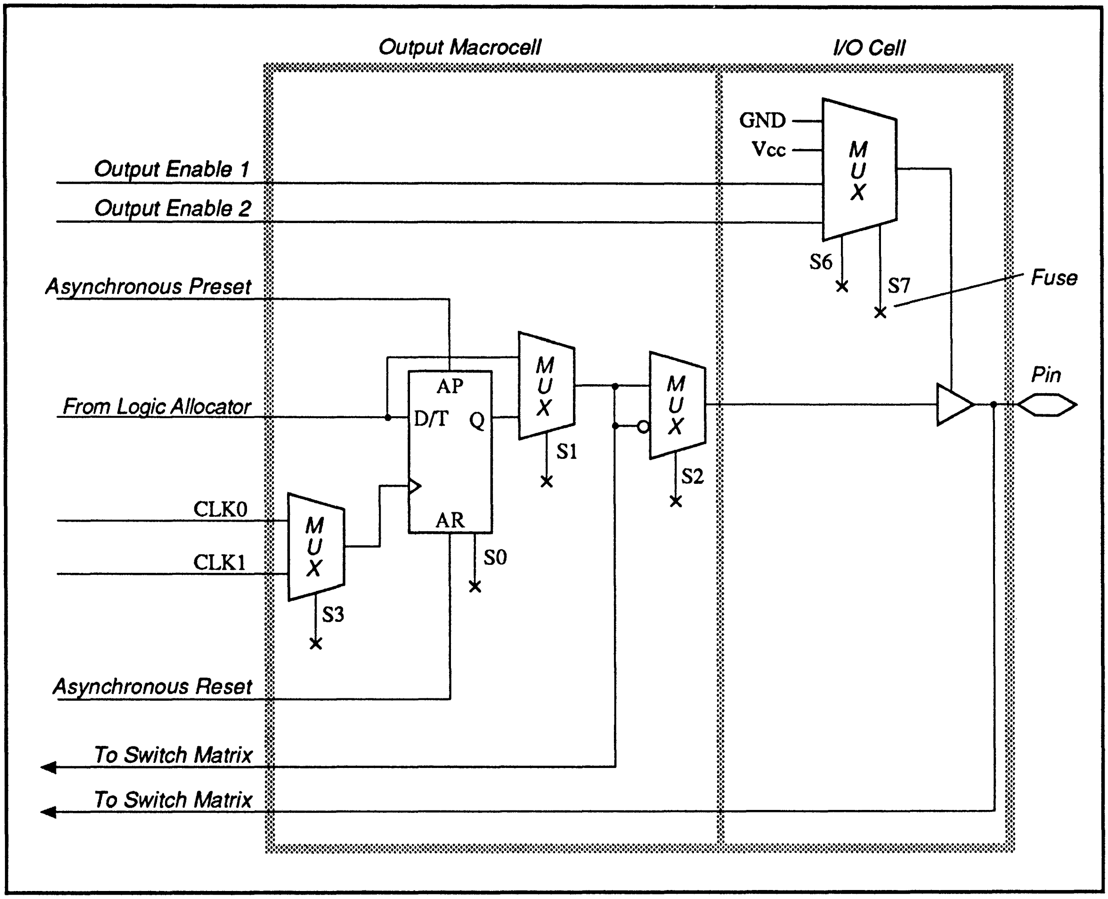
Important: Pin and cell **names** have been assigned for reference purposes and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.



MACH 110 Node Numbers and Cell Names

MACH 110 DEVICE

The logic for a MACH 110 output macrocell and I/O cell is shown next.



MACH 110 Output Macrocell and I/O Cell

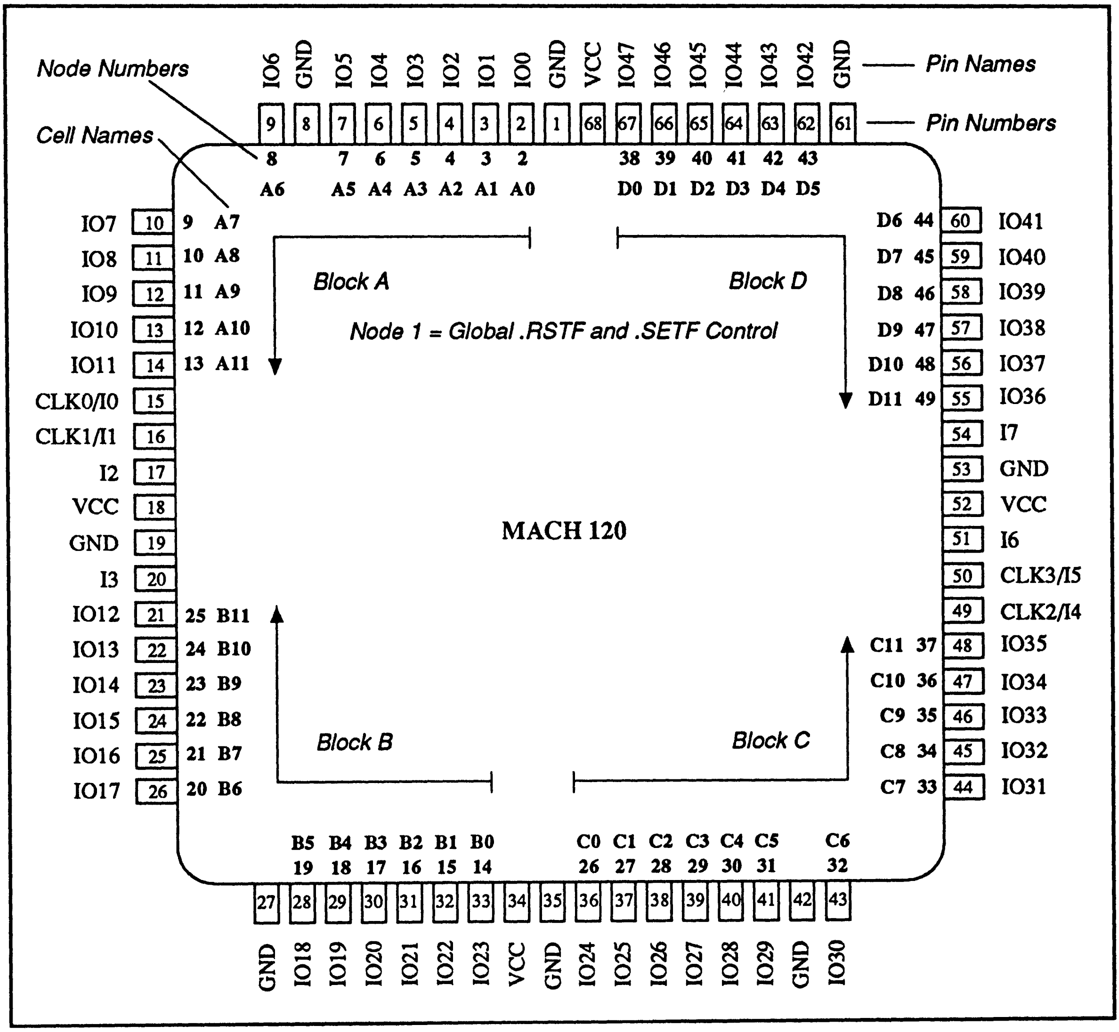
The output of the macrocell can be combinatorial, D flip-flop, or T flip-flop. You define the configuration in the pin declaration or equation segment of the design.

Note: For MACH 1 series devices, latches are implemented as combinatorial functions. MACH 2 series devices have resources that allow latches with active-low latch inputs to be configured directly.

MACH 120 DEVICE

An illustration of the MACH 120 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

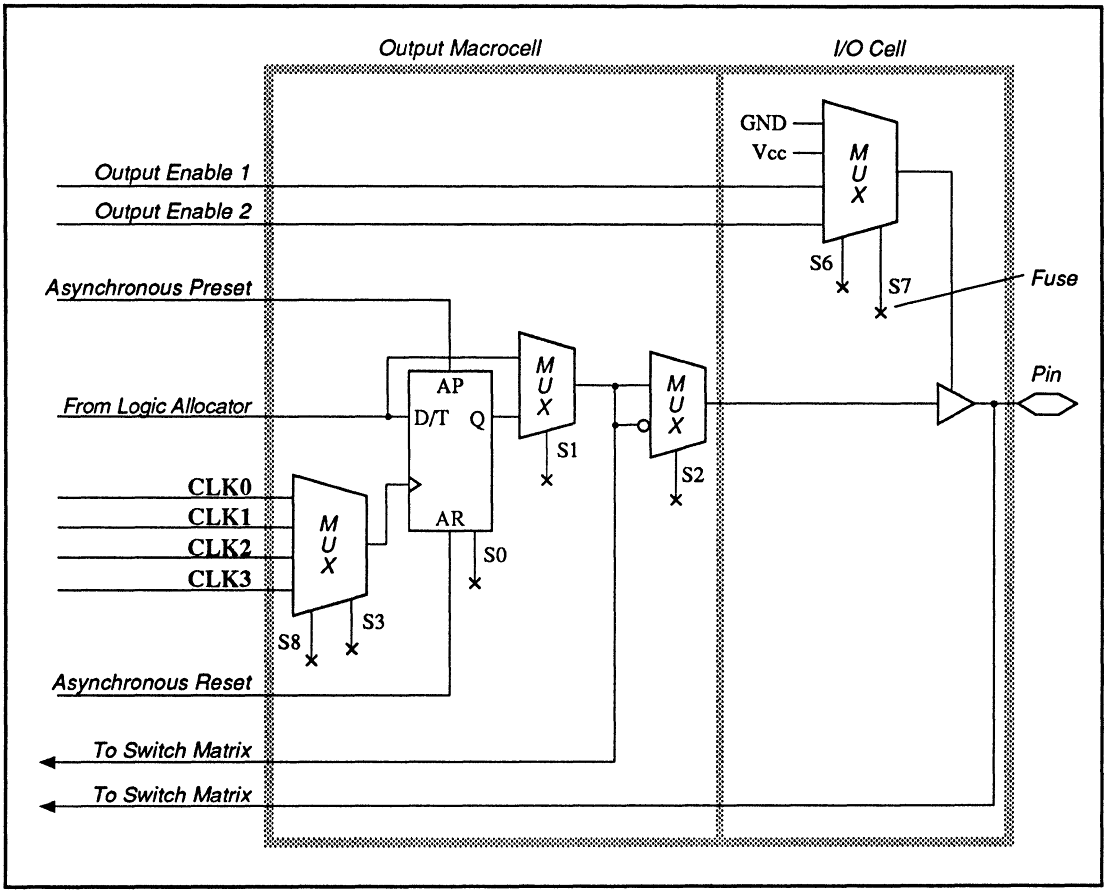
Important: Pin and cell **names** have been assigned for reference purposes and reflect functionality.when appropriate. You can assign your own names in the pin declaration segment of the design file.



MACH 120 Node Numbers and Cell Names

MACH 120 DEVICE

The logic for a MACH 120 output macrocell and I/O cell is shown next.



MACH 120 Output Macrocell and I/O Cell

The output of the macrocell can be combinatorial, D flip-flop or T flip-flop. You define the configuration in the pin declaration or equation segment of the design.

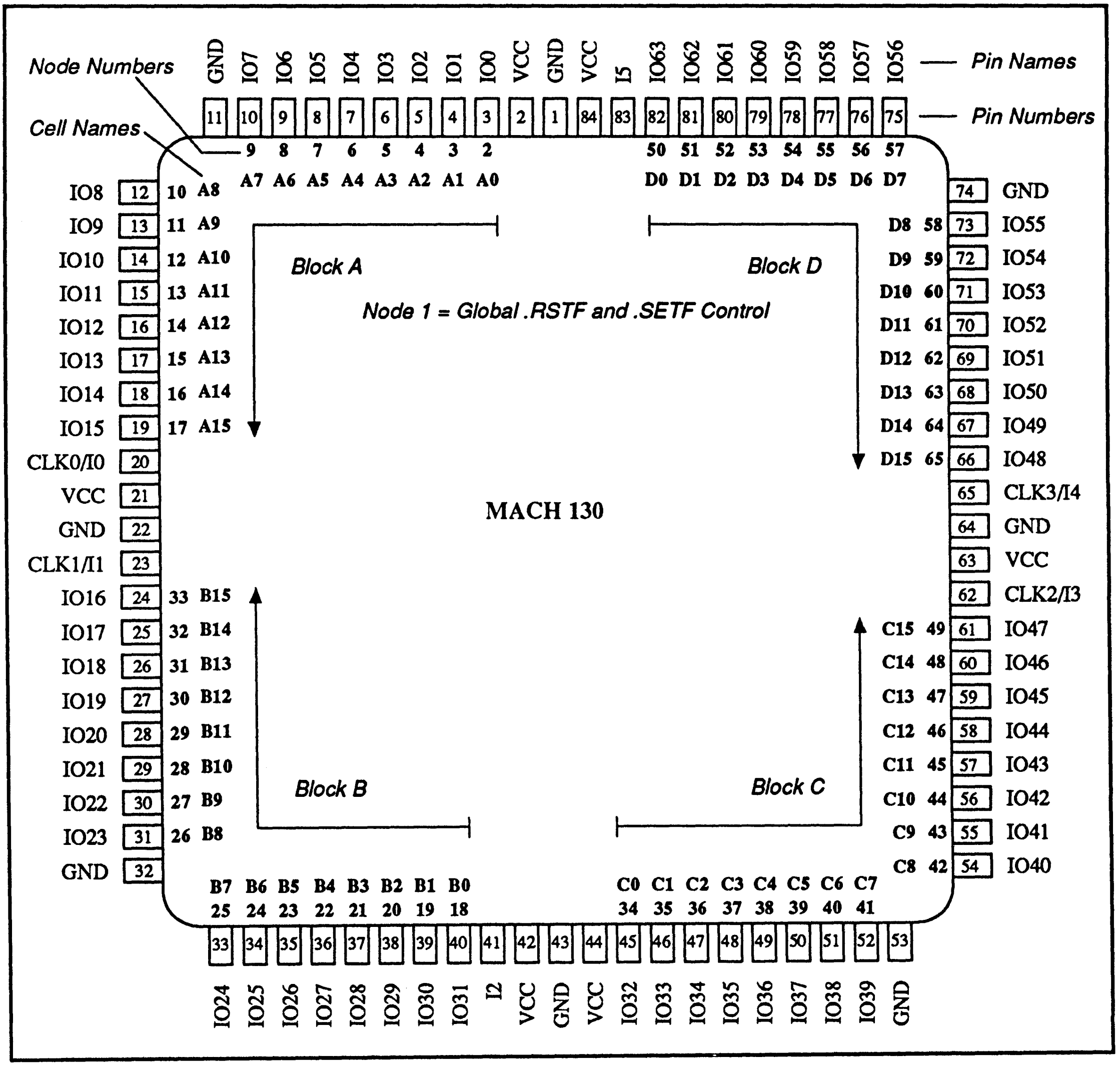
Note: For MACH 1 series devices, latches are implemented as combinatorial functions. MACH 2 series devices have resources that allow latches with active-low latch inputs to be configured directly.

MACH 130 DEVICE

An illustration of the MACH 130 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 3 corresponds to node 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell **names** have been assigned for reference purposes, and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

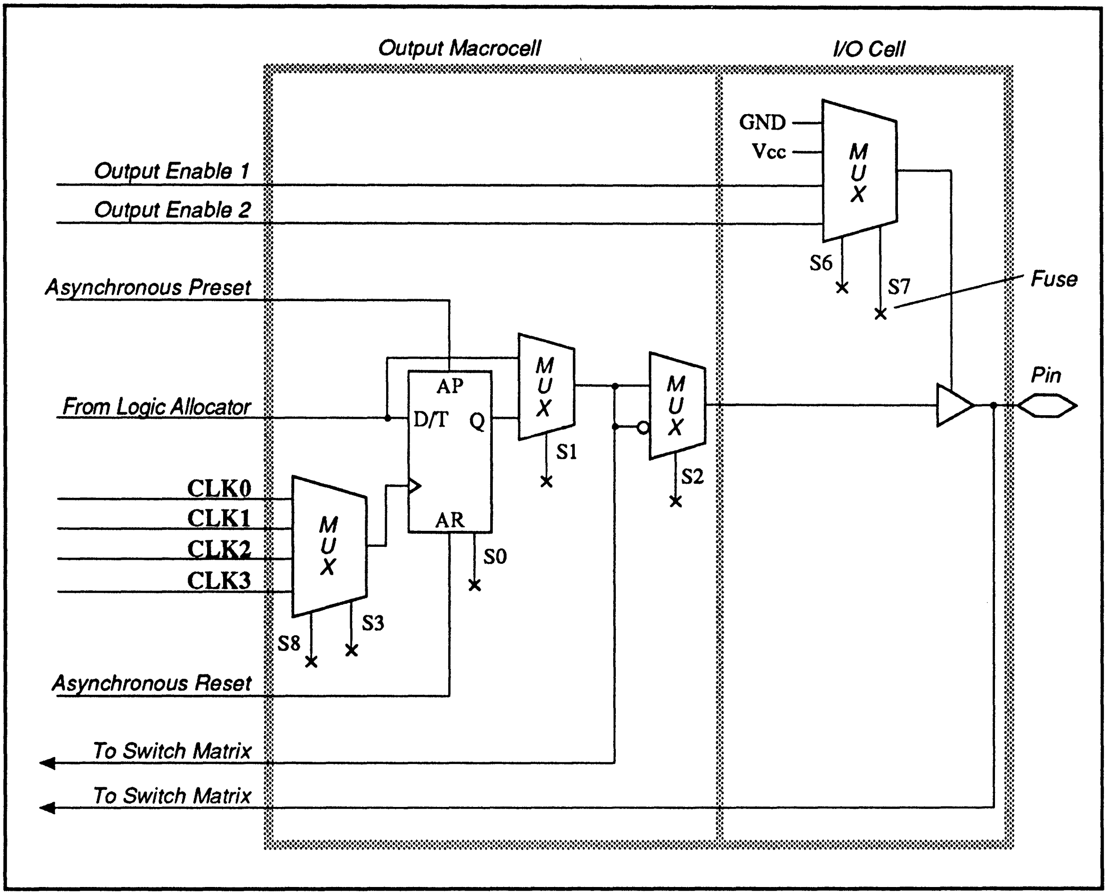
MACH 130 DEVICE



MACH 130 Node Numbers and Cell Names

MACH 130 DEVICE

The logic for a MACH 130 output macrocell and I/O cell is shown next.



MACH 130 Output Macrocell and I/O Cell

The output of the macrocell can be combinatorial, D flip-flop, or T flip-flop. You define the configuration in the pin declaration or equation segment of the design.

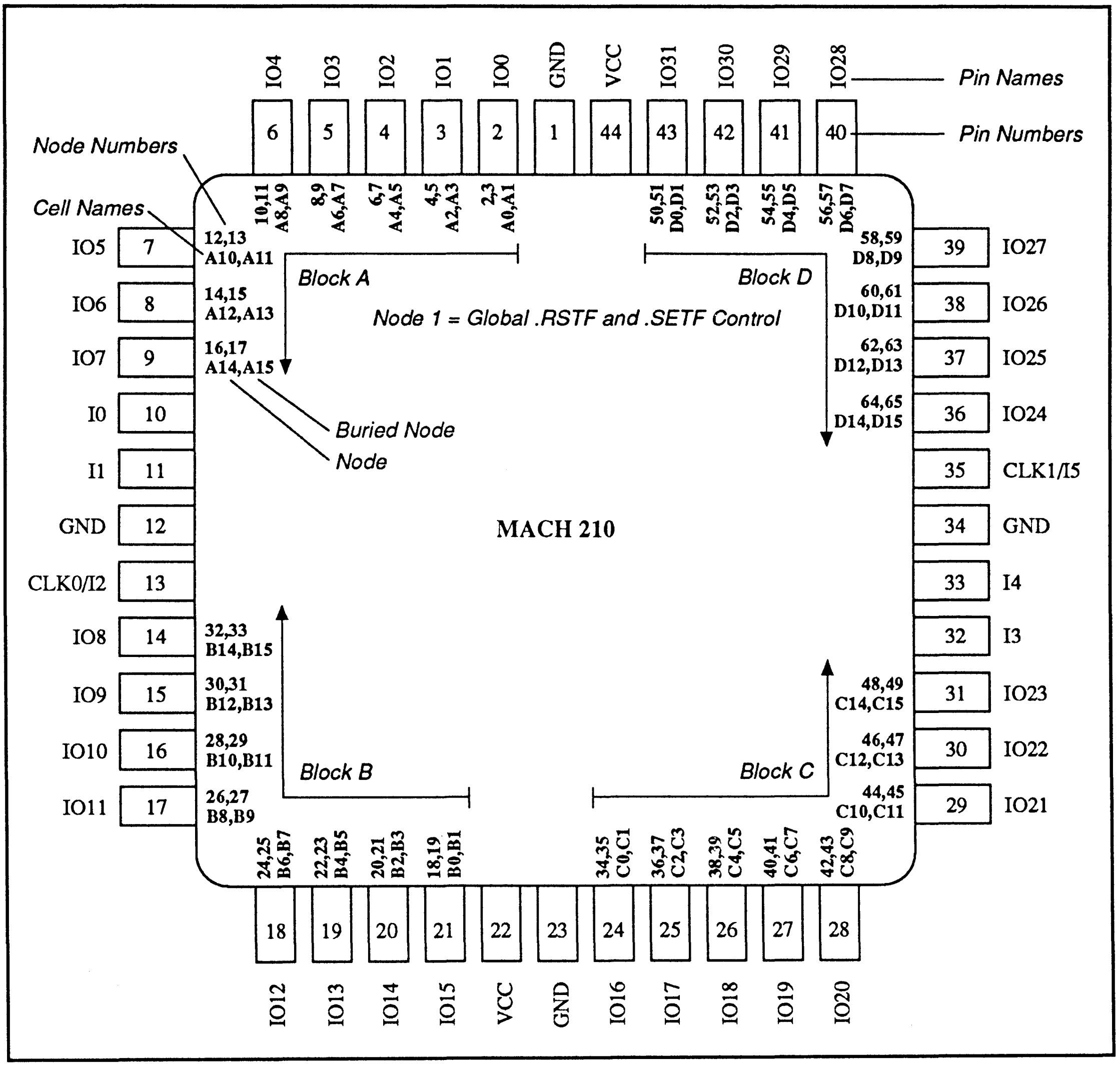
Note: For MACH 1 series devices, latches are implemented as combinatorial functions. MACH 2 series devices have resources that allow latches with active-low latch inputs to be configured directly.

MACH 210 DEVICE

An illustration of the MACH 210 node numbers and cell names is shown next. Each I/O pin in the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell **names** have been assigned for reference purposes and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

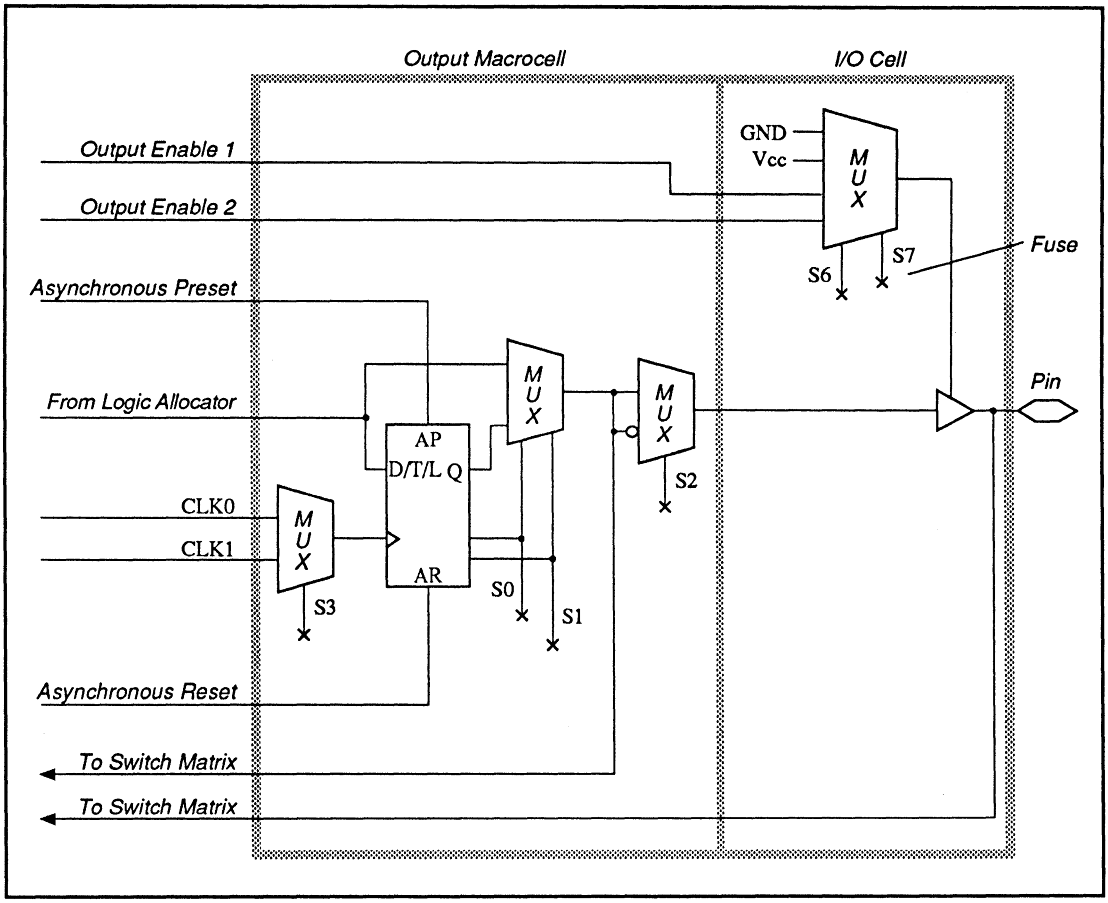
MACH 210 DEVICE



MACH 210 Node Numbers and Cell Names

MACH 210 DEVICE

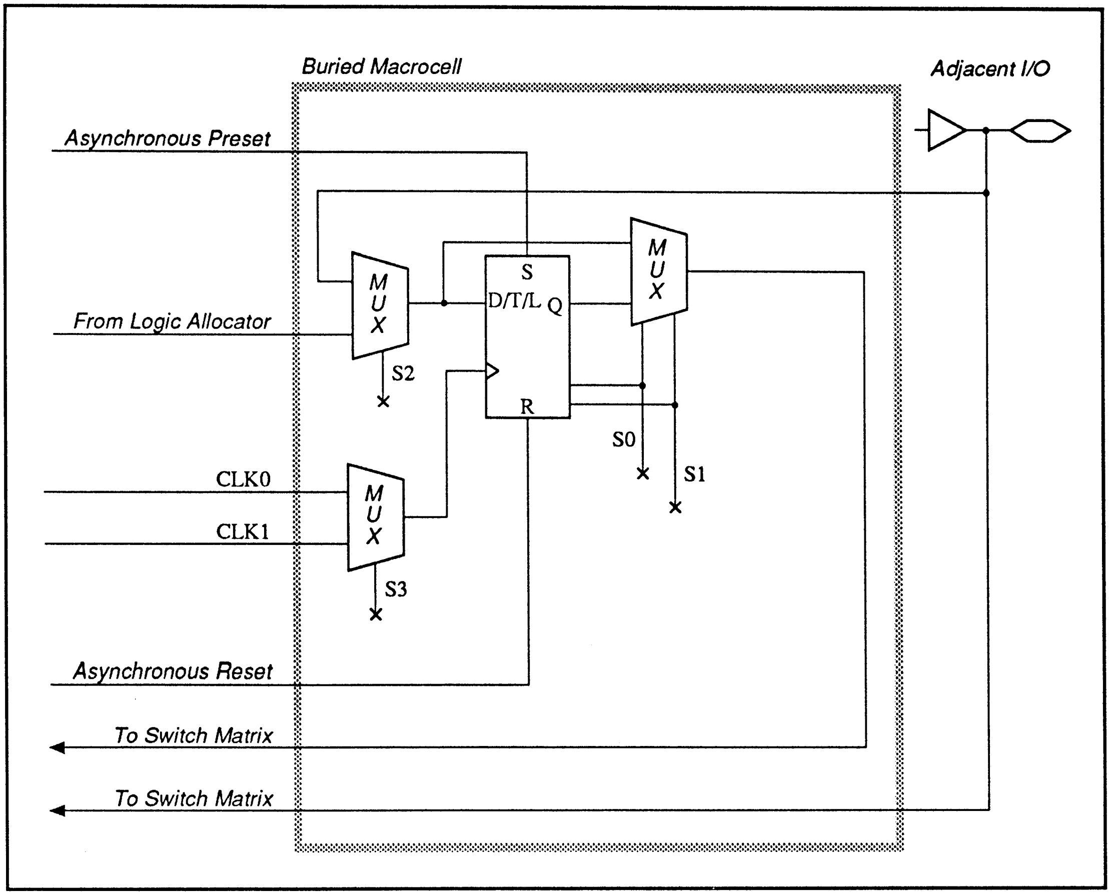
The logic for a MACH 210 output macrocell and I/O cell is shown next.



MACH 210 Output Macrocell and I/O Cell

The output of the macrocell can be combinatorial, D flip-flop, T flip-flop, or latch.[29] You define the configuration in the pin declaration or equation segment of the design.

The next figure illustrates the MACH 210 buried macrocell layout.



MACH 210 Buried Macrocell

[29] Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

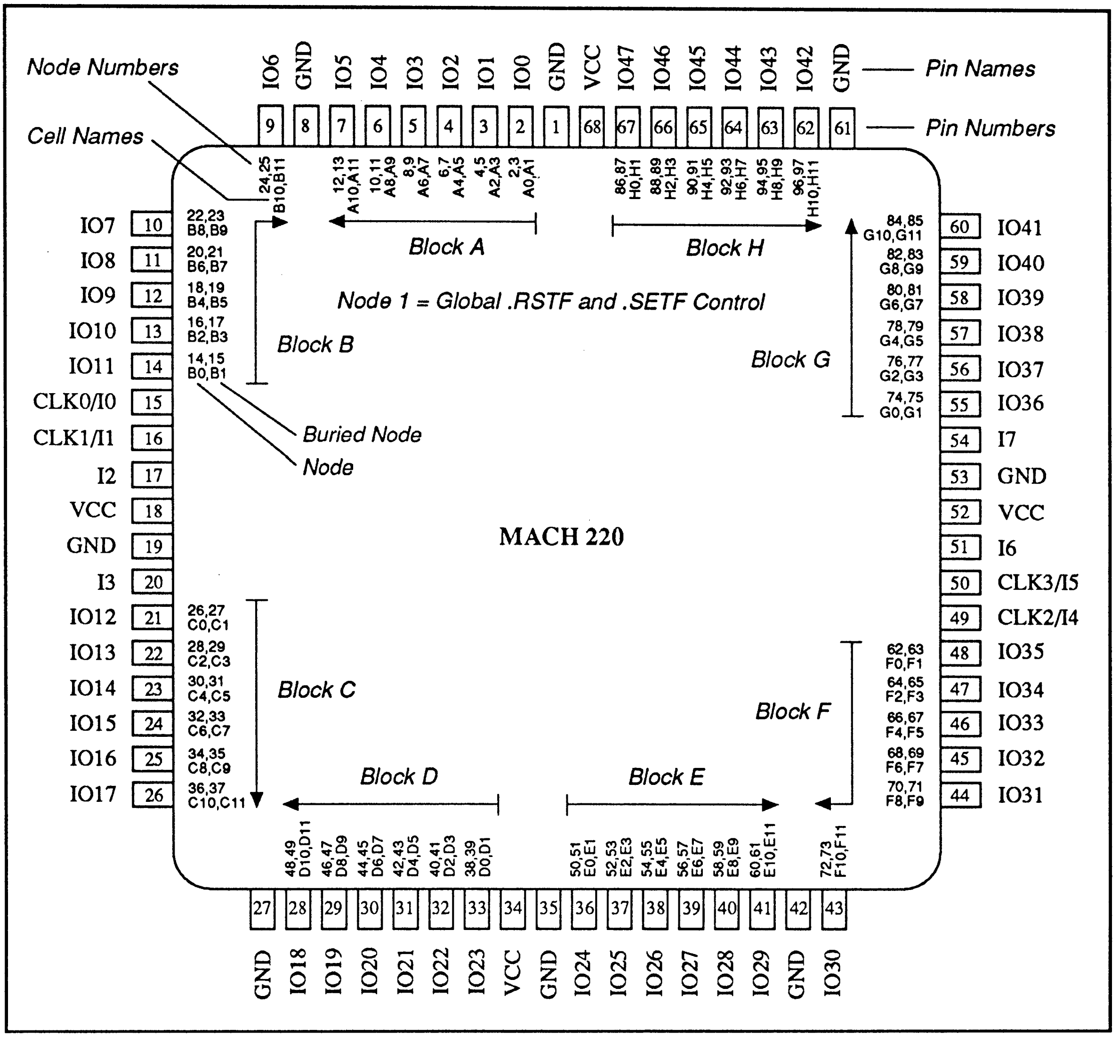
MACH 220 DEVICE

An illustration of the MACH 220 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: The illustrated node numbers for the MACH 220 device are PRELIMINARY, and may change without notice.

Also: Pin and cell **names** have been assigned for reference purposes and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

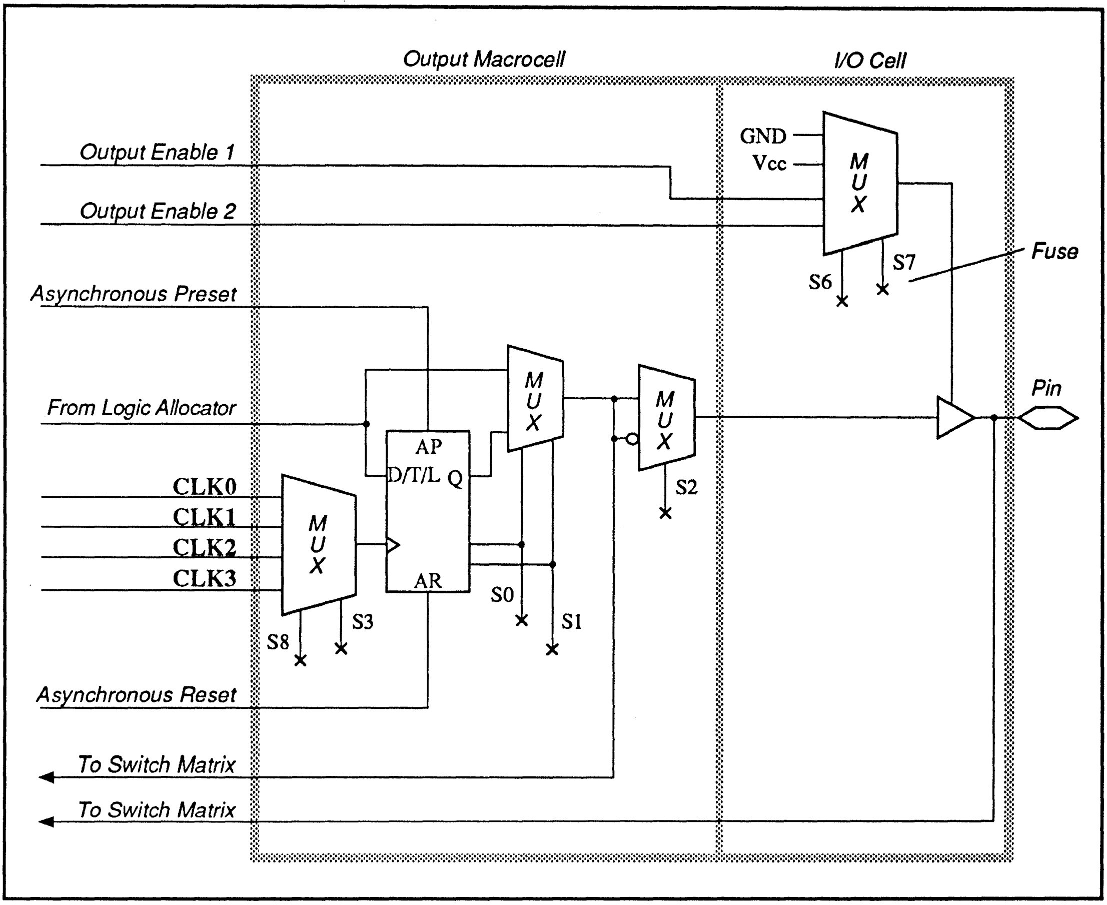
MACH 220 DEVICE



MACH 220 Node Numbers and Cell Names (**Preliminary**)

MACH 220 DEVICE

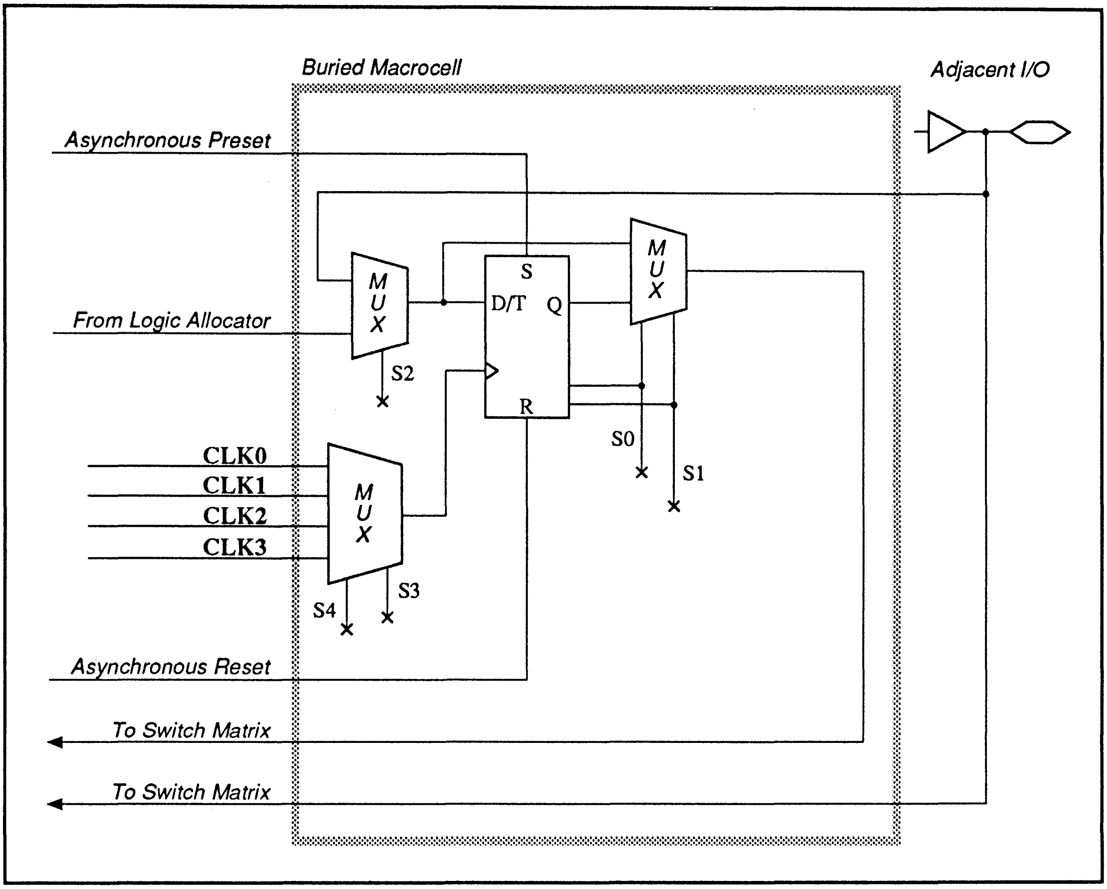
The logic for a MACH 220 output macrocell and I/O cell is shown next.



MACH 220 Output Macrocell and I/O Cell

The output of the macrocell can be combinatorial, D flip-flop, T flip-flop, or latch.[30] You define the configuration in the pin declaration or equation segment of the design.

The next figure illustrates the MACH 220 buried macrocell layout.



MACH 220 Buried Macrocell

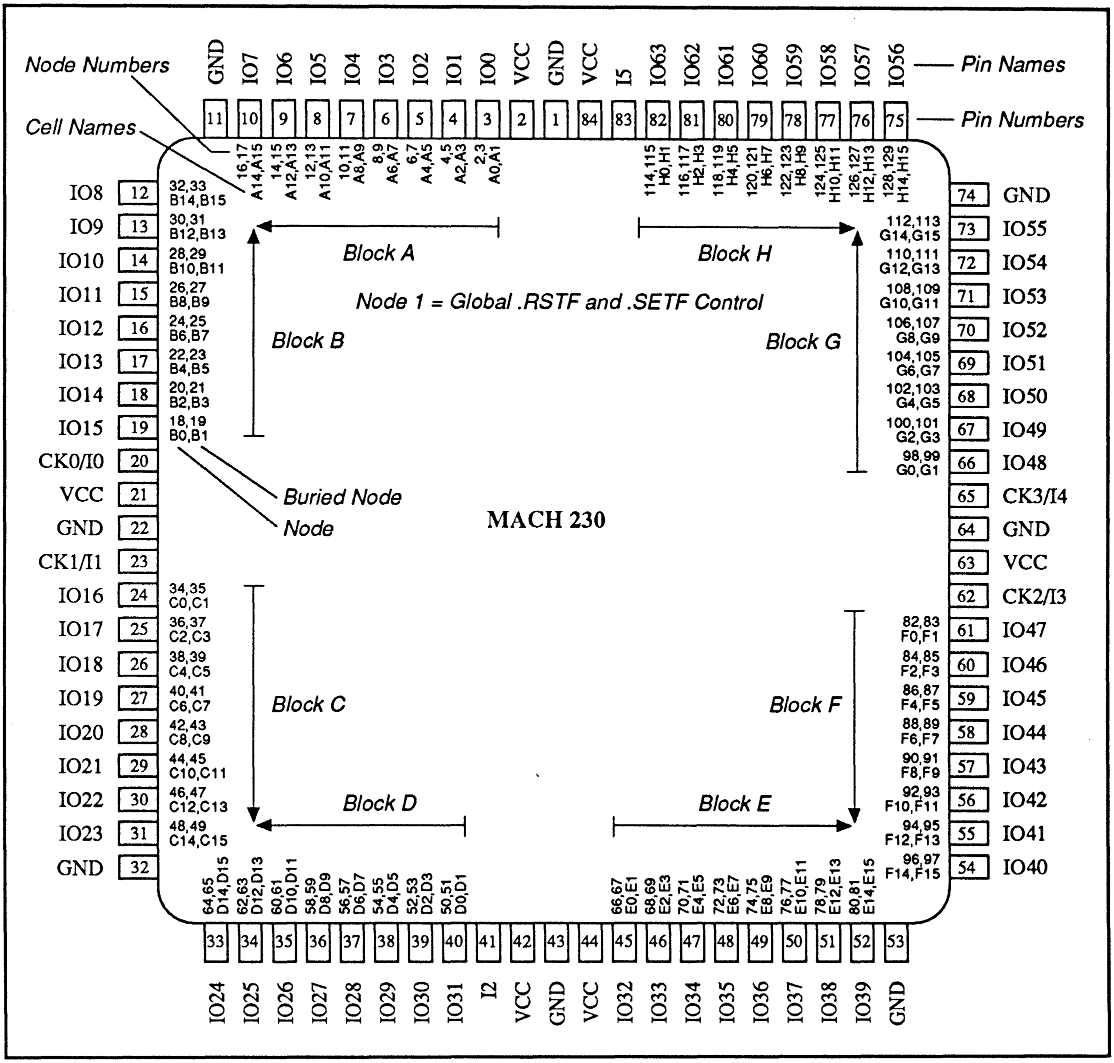
[30] Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

MACH 230 DEVICE

An illustration of the MACH 230 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 3 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 3. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell **names** have been assigned for reference purposes and, where appropriate, reflect functionality. You can assign your own names in the pin declaration segment of the design file.

MACH 230 DEVICE

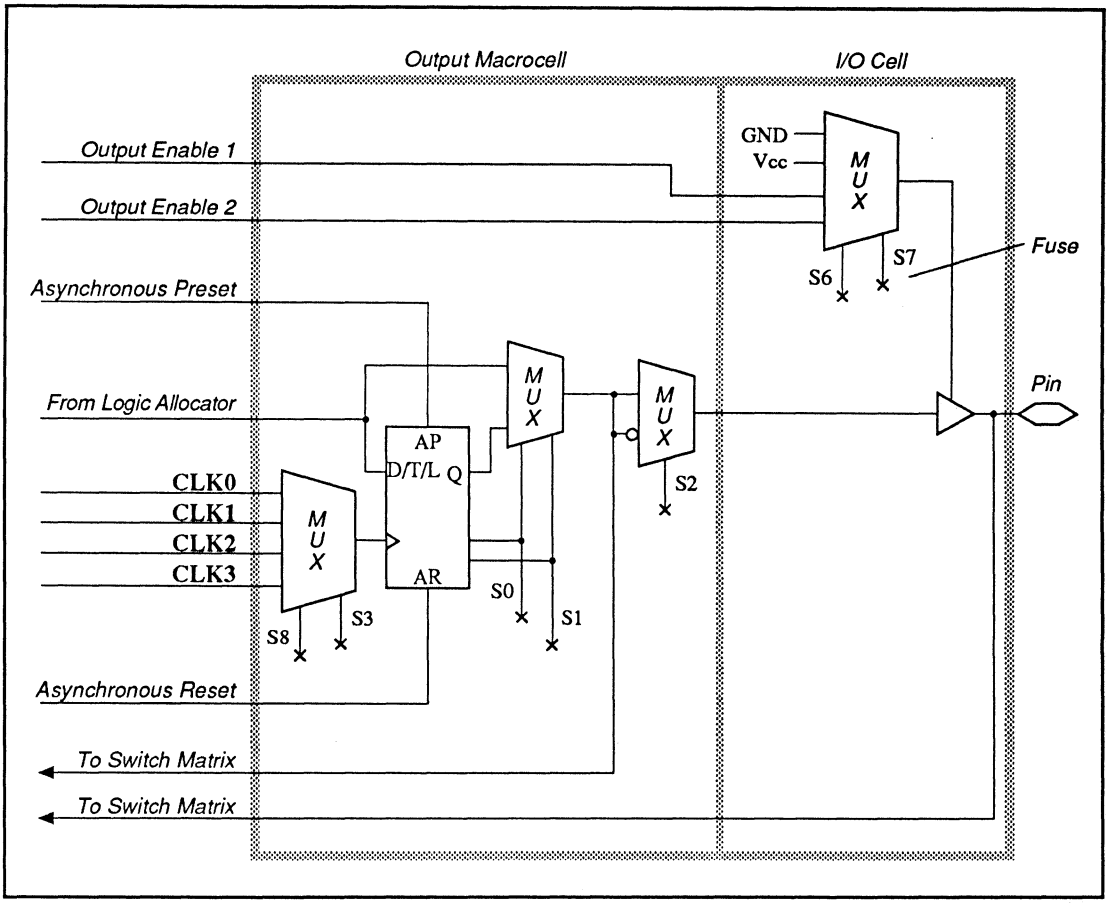


MACH 230 Node Numbers and Cell Names

MACH 230 DEVICE

The logic for a MACH 230 output macrocell and I/O cell is shown next.

The output of the macrocell can be combinatorial, D flip-flop, T flip-flop, or latch.[31] *You define the configuration in the pin declaration or equation segment of the design.*

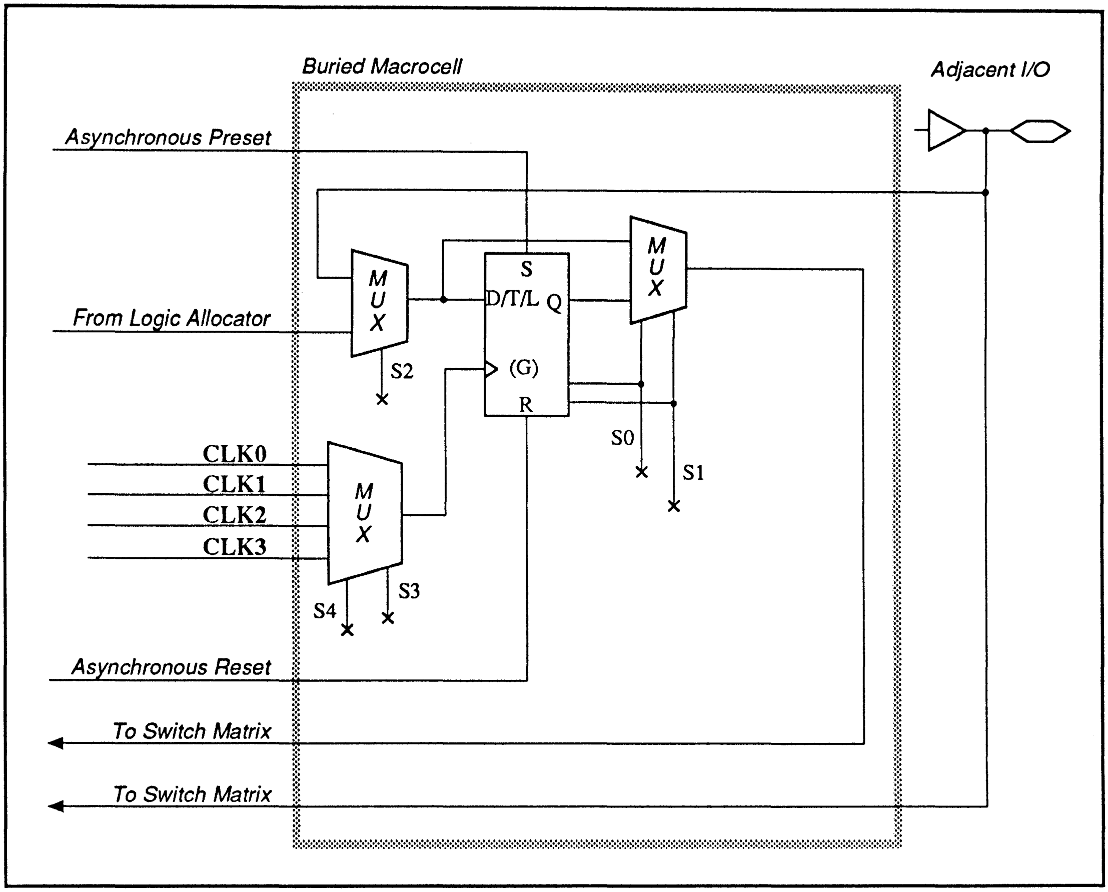


MACH 230 Output Macrocell and I/O Cell

[31] Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

The output of the macrocell can be combinatorial, D flip-flop, T flip-flop, or latch.[32] You define the configuration in the pin declaration or equation segment of the design.

The next figure illustrates the MACH 230 buried macrocell layout.



MACH 230 Buried Macrocell

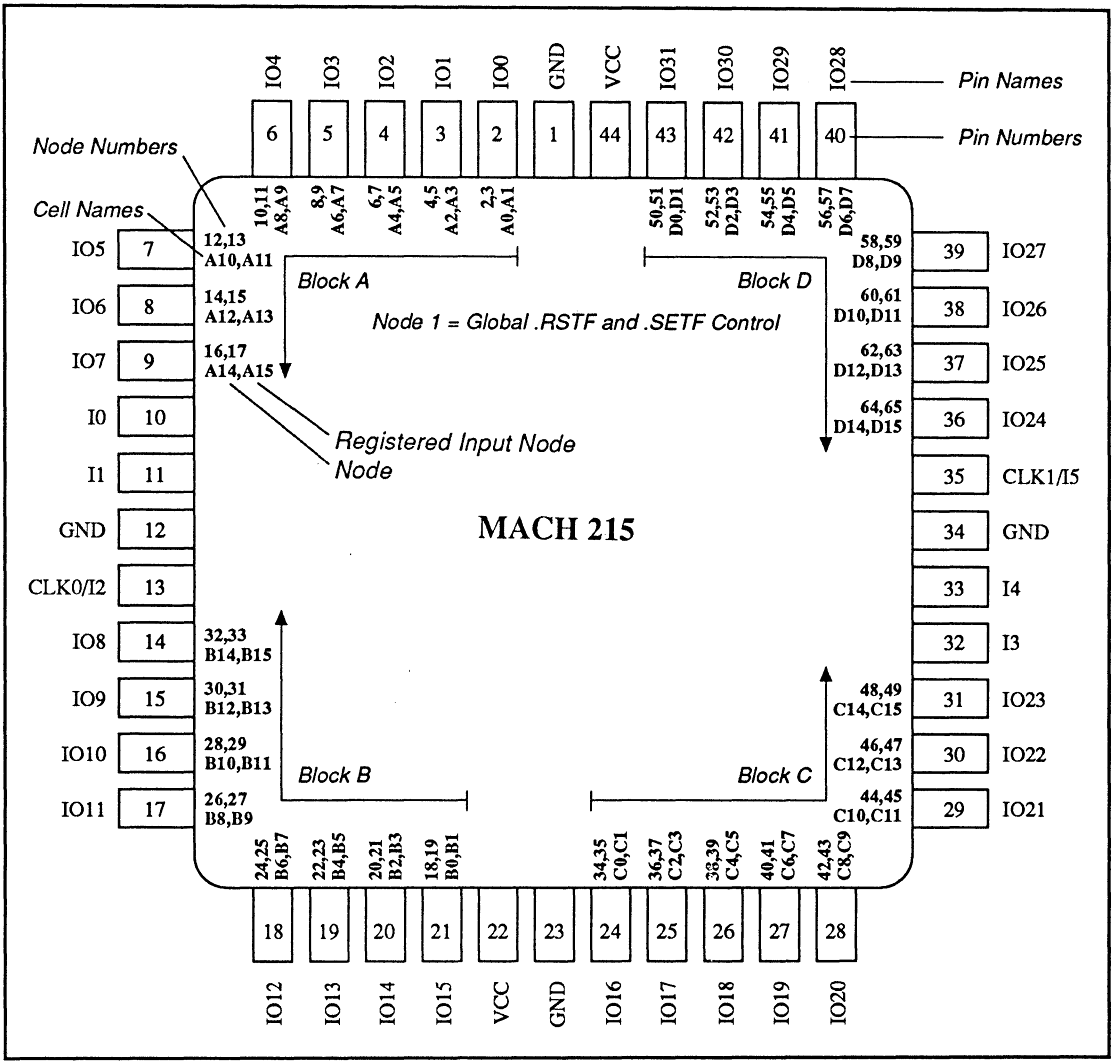
[32] Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

MACH 215 DEVICE

An illustration of the MACH 215 node numbers and cell names is shown next. *Each I/O pin of the device has an associated node, designated by a number. For example, pin 3 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 3. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.*

The MACH 215 device has no buried feedback; instead it has dedicated input registers.

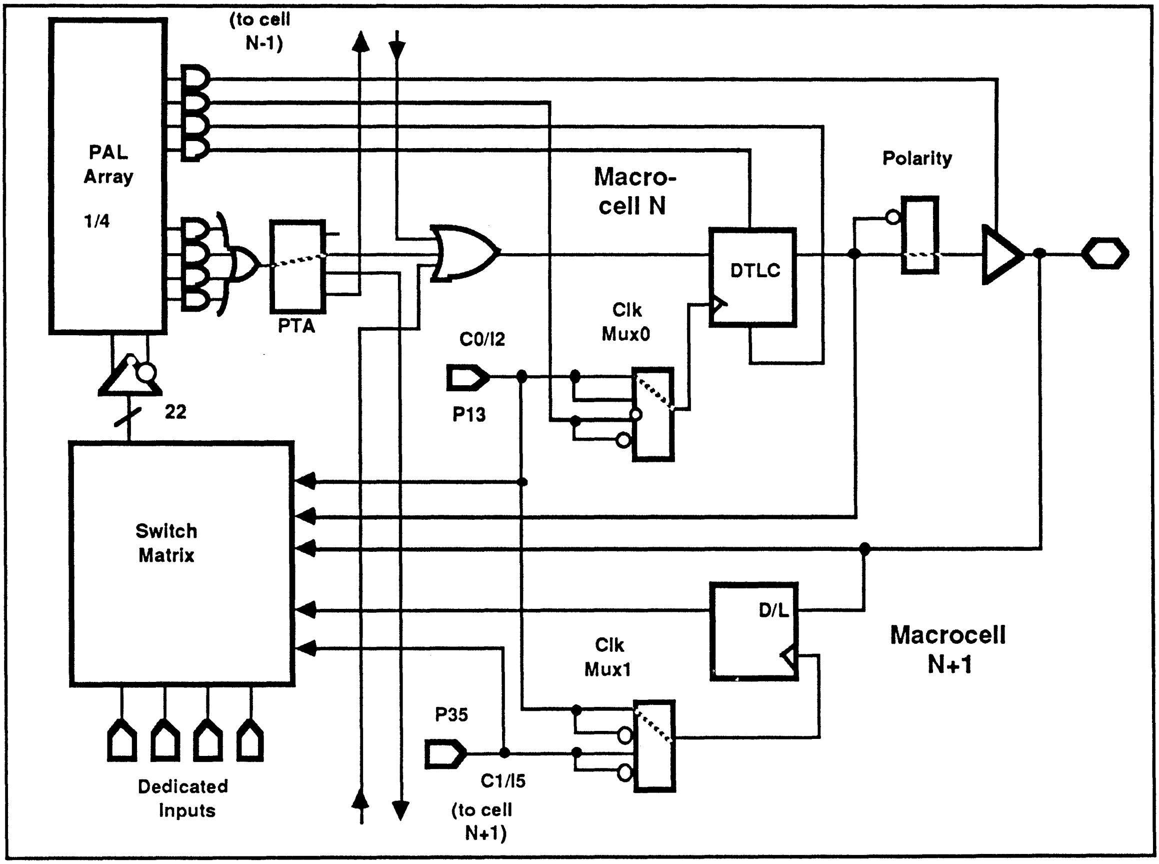
Important: Pin and cell **names** have been assigned for reference purposes and, where appropriate, reflect functionality. You can assign your own names in the pin declaration segment of the design file.



MACH 215 Node Numbers and Cell Names

MACH 215 DEVICE

The logic for a MACH 215 macrocell architecture is shown next.



MACH 215 Macrocell Architecture

SECTION V
APPENDICES

APPENDIX A

PLD TEXT EDITOR

CONTENTS

A PLD TEXT EDITOR

The software includes a text editor to create and edit PALASM design specification (PDS) files. The information presented here includes each command that's available, listed by menu.

- File menu, A.1
- Window menu, A.2
- Block menu, A.3
- Search menu, A.4
- Print menu, A.5
- Macro menu, A.6
- Editing menu, A.7
- Other menu, A.8
- Quit menu, A.9

A.1 FILE MENU

With the text editor, you can switch easily between multiple files, which are inserted into a **ring** in memory as you edit them. When you quit a file, it is deleted from the ring and the previous file in the ring becomes the new current file. Commands to switch between multiple files, and other file commands, are listed in the table below.

FILE MENU COMMAND	COMMAND DEFINITION
Load	Load the named file(s) into the ring.
File	Quit and save the current file.
Save	Write the current file to disk.
Quit file	Quit the current file without saving changes.
Next	Make the next file in the ring the new current file.
Prev	Make the previous file in the ring the new current file.
Read	Insert the specified file into the current file.
Change name	Change the name of the current file.
Write block	Write the current marked block to a specified file.
OS shell	Return to the operating system environment.
Global file	Quit and save all files which have been loaded.

A.2 WINDOW MENU

You can display up to eight windows on the screen at one time. Each window can contain a separate file, and the same file can be viewed in multiple windows. Window commands are listed in the table below.

WINDOW MENU COMMAND	COMMAND DEFINITION
Close	Close the current window, unless it is the only window on the screen, in which case do nothing.
Grow	Increase the size of the current window if there are multiple windows on the screen.
Split	Split the current window horizontally to create a new window.
Next	Make the next window the new current window.
One	Close all windows except the current window and expand it to occupy the entire screen.
Prev	Make the previous window the new current window.
Shrink	Reduce the size of the current window by expanding the window above or below it.
Zoom	Toggle the current window between occupying the entire screen and sharing the screen with other windows.

A.3 BLOCK MENU

Block commands allow you to identify a contiguous portion of text that you can copy, or cut and move, to a new area in the same file or to a different file. They are listed in the table below.

BLOCK MENU COMMAND	COMMAND DEFINITION
mark block Begin	Mark the beginning of a block of characters. The block is not shown until you also mark the end.
mark block End	Mark the end of a block of characters. The block is not shown until you also mark the beginning.
Copy block	Copy the current marked block to the new cursor position.
Move block	Copy the current marked block, delete it from its present location, and insert it in the new cursor position.
Delete block	Delete the current marked block.
Unmark	Remove the mark from the currently blocked characters.
mark Line	Mark the complete line where the cursor is located. Moving the cursor to different lines adds those lines to the block.

A.4 SEARCH MENU

You use the search commands to look for a specified string of characters in the current file. After choosing one of the commands, you are prompted for the string and given several search options. The search is then performed; you can cancel the command by pressing [Return].

The search commands are listed in the table below.

SEARCH MENU COMMAND	COMMAND DEFINITION
Find	Search for a specified string of characters within the current file.
Replace	Search for and replace a specified string of characters with a different string.
Again	Repeat the previous Find or Replace command.

A.5 PRINT MENU

You can specify how you want the current file printed. The table below lists the available print commands.

PRINT MENU COMMAND	COMMAND DEFINITION
print All	Send the entire current file to the printer.
print Block	Send the current marked block to the printer. The marked block must be in the current file.
send Formfeed	Send a form-feed character (ASCII 12) to the printer.
set Left margin	Set the number of spaces to be printed at the beginning of each line. The default is 0 spaces.
set Page size	Set the number of lines to be printed per page. A value of 0 will allow continuous printing.

A.6 MACRO MENU

You can capture keystrokes to create a macro using the Macro record command. For each macro you create, you are asked to assign a unique command key. Every time you press that key, the recorded keystrokes are repeated.

You can establish different libraries of macros by saving them under a file name and loading them into the text editor when you want to use them. The macros are then assigned to the keys to which they were bound when they were saved.

The table below lists the available macro commands.

MACRO MENU COMMAND	COMMAND DEFINITION
Macro record	Toggle Macro record between ON and OFF. You assign a command key, and all following keystrokes are recorded until you execute Macro record a second time.
Read macro	Load the named macro file, with all the macros it contains, from disk into the text editor's internal macro buffer. You are prompted for the name of the file.
Write macro	Save all current defined macros to a disk file. You are prompted to name the file.

A.7 EDITING MENU

The text editor allows you to use the editing commands listed in the table below.

EDITING MENU COMMAND	COMMAND DEFINITION
Add line	Add a blank line below the current line, and place the cursor on the new line. The cursor column does not change.
Delete line	Delete the current line and place the cursor on the following line.
delete to End of line	Delete text on the current line from the cursor position to the end of the line.
Insert line	Insert a blank line above the current line and place the cursor on that line.
Join line	Join the end of the current line to the line below it.
Split line	Split the current line at the cursor position.
Undelete line	Recover the last line deleted from the current file.

A.8 OTHER MENU

Four different operating modes and two tab-setting options are available with the text editor. Those commands, and several other function commands, are listed in the table below.

OTHER MENU COMMAND	COMMAND DEFINITION
set Autoindent OFF (ON)	Toggle between Autoindent ON and OFF. The default is ON. • In auto-indent mode, the left margin is maintained through word-wrap, paragraph-reformatting, and the [Return] key, and is aligned with the first non-space character.
set Insert OFF (ON)	Toggle between Insert ON and OFF. The default is ON. • In insert mode, text is shifted to the right as you enter new text. Backspacing shifts text to the left as it deletes the character to the left of the cursor. • In overwrite (non-insert) mode, text you enter overlays text at the cursor position. Backspacing deletes the character to the left of the cursor but does not shift the remainder of the line.
set Wordwrap OFF (ON)	Toggle between Wordwrap ON and OFF. The default is ON. • In word-wrap mode, text you enter wraps around to the next line when the cursor goes past the right margin and a non-space character is typed. The left margin is determined by the function of the auto-indent mode.

continued on next page...

...continued

OTHER MENU COMMAND	COMMAND DEFINITION
set Ptabwidth	Set the physical tab width to a value of 2, 4, or 8. This determines the width of physical tab characters found in files. You are prompted to set the value.
set Ctabwidth	Set the cursor tab width to any value from 2 through 12. This defines the actual screen position to which the cursor will move each time a tab key is pressed. You are prompted to set the value.
set Right margin	Set the right margin for use in the word-wrap mode.
set Backups OFF (ON)	Toggle between Backups ON and OFF. The default is ON. • The backup function keeps backup copies of files that are written to disk.
set Enter matching ON (OFF)	Toggle between Enter matching ON and OFF. The default is OFF. • The match function automatically enters the matching right character when the left double quote, paren, or square bracket is typed.

A.9 QUIT MENU

Leaving the text editor, and saving all the work you've done, is easily accomplished by using the commands listed in the table below.

QUIT MENU COMMAND	COMMAND DEFINITION
Quit all files	Quit all files without saving changes.
Exit all files	Conditionally quit all files. If the file has been modified, the editor prompts for whether the changes should be saved.

SECTION VI
GLOSSARY / INDEX

GLOSSARY

Active Edge

A low-to-high or high-to-low signal transition that initiates an action.

Assemble

A transparent software process that generates a JEDEC formatted fuse map and test vectors.

Back Annotation

A user-selectable software process that takes pin assignment names determined by the fitting process and writes them back into the PDS file.

Bank

A collection of I/O or buried macrocells within a block. In the MACH 110 device, each block has two banks with 8 I/O cells in each bank. In the MACH 210 device, each block has one bank with 8 I/O cells and one bank with 8 buried macrocells. See Block.

Block

A collection of PAL-like structures that function as independent PAL devices on a single chip. Each block contains a product-term array, a logic allocator, macrocells, and I/O cells. The blocks communicate with each other only through the switch matrix.

Buried Macrocell

A buried macrocell allows the designer to use registered inputs. The input register is a D-type flip-flop. Once configured as a registered input, the buried macrocell can not generate logic from the product term array. A buried macrocell does not send its output to an I/O cell. The output of a buried macrocell is a feedback signal to the switch matrix. This allows the designer to generate additional logic without requiring additional pins.

Circuit Simulation

A software breadboard to verify design functionality and performance. Software that logically emulates a circuit's functions to ensure proper design.

Combinatorial Macro
A macro that performs a logical function and has no storage capability.

Combined files
A PDS file created by merging two separate PDS files. Certain variables may be automatically renamed or reassigned to different pins.

Controllability
The degree to which signals in a part of a circuit can be made to take on specific values through manipulation of primary inputs; used in testability analysis.

Critical Path Evaluation
The identification and analysis of signal paths whose delays could limit the speed of the circuit.

Current Design File
The design file that you specified to work on.

Default Value
The value used unless you specify a different one.

Design File
A file containing the PALASM description or schematic representation of a design.

Disassemble
The process of translating a .TRE file or a JEDEC fuse map back into a PDS.

EEPROM (E^2 PROM)
See Electrically Erasable and Programmable ROM.

Electrically Erasable and Programmable ROM
Similar to an EPROM, but it stores the charge on a floating gate. Newer EEPROMs can erase individual data bytes.

EPLD
Erasable Programmable Logic Device, such as an EPROM.

EPROM
See Erasable Programmable ROM.

Erasable Programmable ROM
Usually refers to the UV erasable, 2764 device type. Generally, EPROMs are erased by shining ultraviolet light on the chip through a quartz window on the package.

Expand

A transparent software routine that performs one of two functions depending on your design. Converts Boolean equations to sum-of-product-terms form,.and state-machine constructs to Boolean Exclusive-OR.

Field

An area in a form where you enter information by typing or selecting from a list of options. Also, an area in a PALASM language construct where you enter specific information.

Field Programmable Logic

Standard products that the user can configure to a specific application, such as PAL and FPLA devices.

Fit

A software process that places pins and nodes after compilation. The placement is automatic if pins and nodes are left floating. This process generates the JEDEC fuse map.

FITR

The FITR software automatically manages the internal arrangement of resources. The software automatically distributes product terms to the macrocells and adjusts the distribution as required by the design.

I/O cell

The I/O cell provides a three-state output buffer. The three-state buffer can be left permanently enabled for use only as an output; permanently disabled for use as an input; or it can be controlled by one of two product terms, for bidirectional signals and bus connections. The two product terms provided are common to a bank of eight I/O cells.

Float

A feature that allows pin and node placement to occur automatically during the fitting process. To leave a pin or node floating, you use a question mark, ?, in the location field of a pin or node statement.

Form

Forms provide specifications to the software. Each form provides one or more text, status, or option fields.

Functional Primitives

Design building blocks, such as adders, shifters, decoders, and memory. A functional primitive differs from a gate-level primitive only in the individual element's degree of functional complexity.

Gate Splitting

An automatic software process that routes feedback through the switch matrix, AND array, and logic allocator to the assigned pin. You can use product terms from nonadjacent macrocells in a MACH 110 or 210 device, including other blocks. Gate splitting uses product terms in multiples of four. However, unlike product-term steering, gate splitting adds one propagation delay for each pass through the AND logic array.

Initial Value

The preset value for an option when the software is invoked. After the software starts executing, the option's value can be changed. See Default Value.

Initialize

The process of establishing an initial condition or starting state. For example, setting logic elements in a digital circuit, or the contents of a storage location, to known states so that subsequent application of digital test patterns drive the logic elements to another known state. Initialization sets counters, switches, and addresses to zero or other starting values at the beginning of, or at prescribed points in, a computer routine.

Keyword

A word used in a language to indicate a specific construct or structure.

Library

See Macro Library.

Log File

A file containing all processes and messages generated during a software processing session.

Logic Allocator

The logic allocator is a block within which different product terms are allocated to the appropriate macrocells.

MACH Device

The MACH is a new 15 ns CMOS EE PLD available in 44-, 68-, and 84-pin packages. The MACH device provides programmable logic capabilities from approximately 900 to 3600 gates. The MACH 1 and 2 families consist of PAL blocks interconnected by a programmable switch matrix. Each family member is differentiated by the number of pins, the number of macrocells, and the amount of interconnect. The MACH 1 family has only output macrocells. The MACH 2 family has output and buried macrocells. Otherwise the families are the same.

MACH 110

The MACH 110 consists of two PAL blocks interconnected by a programmable switch matrix. The MACH 110 has 32 macrocells, 44 pins and 38 inputs.

MACH 120

The MACH 120 consists of four PAL blocks interconnected by a programmable switch matrix. It has 48 macrocells, 68 pins and 58 inputs.

MACH 130

The MACH 130 consists of four PAL blocks interconnected by a programmable switch matrix. It has 64 macrocells, 84 pins and 70 inputs.

MACH 210

The MACH 210 consists of four PAL blocks interconnected by a programmable switch matrix. It has 64 macrocells, 44 pins and 38 inputs. The MACH 210 also has dedicated buried macrocells.

MACH 220

The MACH 220 consists of six PAL blocks interconnected by a programmable switch matrix. It has 96 macrocells, 68 pins and 48 inputs. The MACH 220 also has dedicated buried macrocells.

MACH 230

The MACH 230 consists of eight PAL blocks interconnected by a programmable switch matrix. It has 128 macrocells, 84 pins and 70 inputs. The MACH 230 also has dedicated buried macrocells.

Macro

The elements in a library that you can retrieve to create a schematic.

Macrocells

There are two fundamental types of macrocells: an output macrocell and a buried macrocell. A buried macrocell is found only in the MACH 2 family of devices. The use of buried macrocells effectively doubles the number of macrocells available without increasing the pin count. Both macrocell types can generate registered or combinatorial output. If used, the register can be as a T- or D-type flip-flop. Programmable polarity (for output macrocells) and the T-type flip-flop give the software a way to minimize the number of product terms needed. All macrocells have internal feedback, allowing a pin to be used as an input if the macrocell signal is not needed externally. See also Output Macrocell.

Macro Library

A collection of macros, organized or classified by function, that contain the macros required for schematic capture.

Merge

Combine two or more text files into one description or PDS file. A synonym for Combine.

Minimize

A transparent software routine that reduces a set of Boolean equations to a simpler sum-of-products form, usually involving fewer product terms or literals.

Netlist

A list of circuit elements and their interconnections.

Node

An identifiable point in a design. A node can be associated with a specific device, geographic location, or signal.

Nominal Delay

The mean time signals take to propagate through a logic element or a wire. The effect of an input change to an element on the output does not occur until after the nominal delay.

Observability

In addition to the control offered by preload, testing requires observability of the internal state of the device following a sequence of vectors. The MACH device offers an observability feature that allows the user to send hidden buried register values to observable output pins. For macrocells that are configured as combinatorial, the observability function suppresses the selection of the combinatorial output by forcing the macrocell output multiplexer into registered output mode. The observability function allows observation of the associated registers by overriding the output enable control and enabling the output buffer.

Option List

A list that appears when you press [F2] in an option field of a software form. You select an option from the list or change the specification in the selected field.

Output Macrocell

The output macrocell sends its output back to the switch matrix, via internal feedback, and to the I/O cell. The feedback is always available regardless of the configuration of the I/O cell. This allows for buried combinatorial or registered functions, freeing the I/O pins for use as inputs if not needed as outputs.

PAL

See Programmable Array Logic.

PAL Blocks

The PAL blocks can be viewed as independent PAL devices on the chip. Each PAL block contains a product-term array, a logic allocator, macrocells, and I/O cells. PAL blocks communicate with each other only through the switch matrix. Additionally, each PAL block contains an asynchronous reset product term and an asynchronous preset product term. This allows the flip-flops within a single PAL block to be initialized as a bank.

PALASM

The PALASM software development system runs on PC/AT compatible and 386-based systems. The package provides low-cost CAD capabilities for the following design phases: design entry, implementation, verification, programming and testing. The software operates with an easy-to-use pull-down menu interface that allows most operations to be performed with a single keystroke. Designs can be entered using mixed schematic, state machine, and Boolean input formats.

Programmable Array Logic

A programmable logic architecture having two levels of logic with a programmable AND array.

Parse

A transparent software routine that checks the syntax of the design file and creates the intermediate design.TRE and design.pin files.

PDS File

A textual representation of a design file using PALASM language constructs.

PLA

See Programmable Logic Array.

PLD Device

A general category of programmable logic devices that contains the two subcategories of PALs and PLAs.

Power-up Reset

All flip-flops power-up to a logic LOW for predictable system initialization. The actual values of the outputs of the MACH devices depend on the configuration of the macrocell. The VCC rise must be a monotonic and the reset delay time is 1000 ns maximum.

Product-Term Array

A product-term array consists of of a number terms that form the basis of the logic being implemented. The product terms drive the logic allocator, which allocates the product terms to the appropriate macrocells. The number of product terms allocated to each array is not fixed and the full sum of products is not realized in the array. The inputs to the AND gates come from the switch matrix, and are provided in both true and complement forms for efficient logic implementation. There are several three-state product terms that provide three-state control to the I/O cells.

Product-Term Steering

An automatic software process that borrows supplemental terms from adjacent macrocells, in addition to the four product terms available to each macrocell. This may occur when the maximum gate width is set to 12 and you include more than four product terms in an equation. In MACH 110 designs, up to eight product terms can be borrowed from adjacent macrocells for a total of 12; four product terms can be borrowed from the adjacent macrocell above and below. In MACH 210 designs, up to 12 product terms can be borrowed from adjacent macrocells for a total of 16.

Programmable Logic Array

A rectangular array of AND and OR gates used to generate logic functions in sum-of-products form.

Programmable Read-Only Memory

A ROM that can be programmed by the customer.

PROM

See Programmable Read-Only Memory.

Reference Designator

The reference designator is used to create unique net and block names. See Macro.

Register Preload

All registers on the MACH devices can be preloaded from the I/O pins to facilitate functional testing of complex state machine designs. This feature allows direct loading of arbitrary states, making it unnecessary to cycle through long test vector sequences to reach a desired state. In addition, transitions from illegal states can be verified by loading illegal states and observing proper recovery.

Reserved Word

See Keyword.

Schematic Diagram

A circuit diagram in which components are represented by standard, simple, easily drawn symbols.

Security Bit

A security bit is provided on the MACH device as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, this bit defeats readback of the programmed pattern by a device programmer.

Signal Contention

Conflicts between signal assignments that arise when you merge one PDS file with another. For instance, input signals may have the same location; this may or may not be appropriate for your design.

Spike

The output condition where the inputs are being manipulated faster than the element's propagation delay.

Switch Matrix

The switch matrix provides communication between PAL blocks and routes inputs to these blocks. The switch matrix takes all dedicated inputs, I/O feedback signals, and buried feedback signals and routes them as needed to the various PAL blocks. The switch matrix makes the MACH device more than just multiple PAL devices on a single chip.

Unit-Delay

A simulation technique used to verify functionality. In this technique all the delays of the elements are set to one time unit.

Unit-Delay Simulation

A simulation that assumes equal propagation delays.

Zero-Delay Simulator

A simulator used for functional validation. The signals have no delay.

Zero-/Unit-Delay Simulator

Combination of the zero-delay and unit-delay simulation selements and with unit-delay circuits inserted into the feedback lines and memory elements.

INDEX

NOTES

NOTES

ADVANCED
MICRO
DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale
California 94088-3453
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL-FREE
(800) 538-8450

APPLICATIONS HOTLINE &
LITERATURE ORDERING
(800) 222-9323
(408) 749-5703

SUPPORT PRODUCTS
ENGINEERING HOTLINE
JAPAN 0-031-11-1129
UK 0-800-89-1131
USA (800) 222-9323

RECYCLED &
RECYCLABLE

Printed In USA
CPS-3M-6/92-0
12379C

PALASM® 4 Getting Started and MACH™ Workbook

1992

Advanced
Micro
Devices

PALASM® 4 USER'S MANUAL

VOLUME 1 - PALASM 4 GETTING STARTED AND MACH™ WORKBOOK

901 Thompson Place
P.O. Box 3453
Sunnyvale, CA 94088

TEL: 408-732-2400
TWX: 910339-9280
TELEX: 34-6306
TOLL FREE: 800-538-8450

APPLICATIONS HOTLINE: 800-222-9323

Advanced Micro Devices reserves the right to make changes in specifications at any time and without notice. The information furnished by Advanced Micro Devices is believed to be accurate and reliable. However, no responsibility is assumed by Advanced Micro Devices for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of Advanced Micro Devices.

PALASM 4 USER'S MANUAL

VOLUME 1 - PALASM 4 Getting Started and MACH Workbook

MACH Design Workbook

VOLUME 2 - PALASM 4 Reference Guide

Section IV: Software Reference

Section V: Appendices

Section VI: Glossary / Index

TABLE OF CONTENTS

VOLUME 1 - Getting Started and MACH Workbook

SECTION I: GETTING STARTED

CHAPTER 1: INSTALLATION GUIDE

CHAPTER 2: DESIGN ENTRY

CHAPTER 3: SCHEMATIC-BASED MACH DESIGN DEMONSTRATION

SECTION II: DESIGNER'S GUIDE

CHAPTER 4: ENTRY

CHAPTER 5: COMPILATION / FITTING

CHAPTER 6: SIMULATION

MACH Design Workbook

VOLUME 2 - Reference Guide

SECTION III: LIBRARY REFERENCE

CHAPTER 7: INTRODUCTION

CHAPTER 8: MACRO AND SCHEMATIC DATASHEETS

SECTION IV: SOFTWARE REFERENCE

CHAPTER 9: MENUS AND COMMANDS

CHAPTER 10: LANGUAGE REFERENCE

CHAPTER 11: DEVICE PROGRAMMING REFERENCE

SECTION V: APPENDICES

APPENDIX A: PLD TEXT EDITOR

SECTION VI: GLOSSARY / INDEX

PREFACE

This manual is organized into two volumes and six sections. Sections I-III are contained in volume 1, titled "PALASM 4 User's Manual - Getting Started and MACH Workbook", and Sections IV-VI are contained in volume 2, titled "PALASM 4 User's Manual - Reference Guide".

- Section I provides hands-on tutorials that guide you through the installation and provide a quick tour of the design process, software, and special considerations for both text-based and schematic-based MACH designs.

- Section II describes schematic, Boolean, and state-machine entry methods; identifies MACH design strategies and the tradeoffs between various considerations and solutions; and provides an introduction into the PALASM simulator.

- Section III introduces the AMD-supplied library for MACH designs produced using OrCAD/SDT™ III, and includes both macrocell and schematic datasheets.

- Section IV provides the Software Reference, which describes software operations and defines all commands and options; the Language Reference, which is organized alphabetically and describes the PALASM language syntax; and the Device Programming Reference, which provides information on programming PLD and MACH devices.

- Section V provides appendices for information outside the scope of the chapters. Initially, this section includes only one appendix, which describes how to use the PLD Text Editor.

- Section VI provides a glossary of terms and an index for the manual.

The manual places each title and major topic at the top of a new page. Each chapter introduction lists the major topics therein. Each major topic introduction identifies the second-level topics to watch for, and so on.

Note: Abbreviations in this manual that are not explicitly defined are those deemed standard by the IEEE.

The MACH Workbook appears at the back of the first volume. It is intended as a tutorial-based guide to designing with AMD MACH devices within PALASM 4 software.

AUDIENCE

The reader of this manual is assumed to have a working knowledge of the design, testing, and reliability of programmable logic devices. Additionally, the reader

- Must have a BSEE degree or equivalent and be experienced in general system-level logic design
- May be first-time PLD design developers or may have a high level of expertise
- May be new to PALASM software or may be familiar with earlier versions of PALASM software

PREREQUISITES

The following prerequisites are recommended.

- Section I, Getting Started, is intended for new users; **no** prerequisites are needed.

- Section II, Designer's Guide, is intended for advanced users; familiarity with MACH- and PLD-device datasheets is required.
- Section III, Library Reference, is intended for MACH users; no prerequisites are needed.
- Section IV, Software Reference, is intended for intermediate and advanced users; familiarity with Section I, Getting Started, is needed for Chapters 10 and 11.
- Section V, Appendices, is intended for all users; no prerequisites are needed.
- Section VI, Glossary / Index, is intended for all users; no prerequisites are needed.

ACKNOWLEDGEMENTS

For their great help with the **publication** of this manual, Advanced Micro Devices would like to thank the following major contributors and others too numerous to mention.

John Davis
Sharon Garner
Jana McNulty
George Roark
Bob Steggles
Joe Walcek
Eric Fleischman
Arthur Khu
Stan Potts
Nick Schmitz
Gail Tiberi

Important: For answers to questions about any information in this workbook, contact a **customer support** representative at the AMD Applications Hotline: **800-222-9323**. The people acknowledged above are **not** part of the customer support group and are **not** available to answer questions.

SECTION I
GETTING STARTED

Chapter 1

Installation Guide

CONTENTS

1 INSTALLATION GUIDE

This guide provides steps you complete to install the PALASM®4 version 1.1 software. This software supports both PLD- and MACH™-device designs. The installation process takes about 15 minutes.

- The requirements discussion identifies the hardware and software you need to complete the installation successfully for the kinds of designs you'll produce.

- The steps discussion guides you through the installation process, whether you use the defaults or change them.

Important: The **left column** of this guide identifies which key you must press, what you type, or what you must select on the screen to respond correctly.

Sometimes, a prompt appears in the left column before your response to help you track the process

The **right column** of this guide provides numbered steps that explain what to select or do. Following most steps is a description of what happens on the screen.

Also: **To expedite** the installation process, just read the **left column,** then type the text, select the boxed command, or press the named key.

1.1 REQUIREMENTS

Discussions here identify hardware and software requirements.

1.1.1 HARDWARE

PALASM 4 software is **not** supported across networks. The following hardware is required.

A. One IBM® PC XT™, PC AT™, PS/2™, or 100%-compatible computer

B. A minimum of 512 kilobytes (kB) of system RAM available after all device drivers and TSRs have been installed

C. 10 megabytes (MB) of hard disk space available

> **Note**: You can use the following DOS command to determine the amount of disk space and memory available,
>
> - Type CHKDSK and press [Enter].
>
> Available disk space is identified as bytes available on disk; available RAM is listed as bytes free.

D. One floppy-diskette drive

E. One 1 MB **extended** memory system board, required for large designs or designs with a large number of simulation test vectors

F. One mouse for OrCAD/SDT®III only

G. Graphics capability for OrCAD/SDT III only

Recommendation: If you are using a monochrome screen with a graphics board, use the following commands to set up your system after installing the software.

- `CD \PALASM\DAT [Enter]`
- `DEL PALASM.PCX [Enter]`
- `SET MODE=MONO [Enter]`

Include the last command in your AUTOEXEC.BAT file also.

H. One of the following optional printers

- IBM ProPrinter
- Epson FX 80 series
- HP LaserJet and HP LaserJet Series II

  The HP LaserJet printers must include a font cartridge with line-drawing capability, such as cartridge #HP3659A.

1.1.2 SOFTWARE

The minimum software requirement, which must be met before installing and running PALASM 4 version 1.1, is listed below.

A. MS-DOS™ version 3.1 or later
B. Proper software driver files for peripheral devices

The ***PALASM 4 Release Notes*** that accompany this software contain important, recently acquired information on PALASM's compatability with new versions of MS-DOS (such as MS-DOS 5.0), networks, early BIOS ROMs, extended and expanded memory.

1.2 STEPS

You cannot run the PALASM 4 software from a floppy disk. Also, you must **install** the PALASM 4 software; simply copying the files to your hard disk does not work. The steps below guide you through the installation and configuration process.

1.2.1 INSTALLATION

Before you begin, you may want to check the path to the following files so you can complete the installation form appropriately.

- COMMAND.COM
- DRAFT.EXE

> **Important**: If this is the second or subsequent installation of the PALASM 4 software on the system, it is a good idea to delete all earlier PALASM directory files and subdirectories before the new installation. This ensures that all extraneous files are cleared from the \PALASM hierarchy.
>
> **Also**: To facilitate removal of an old PALASM directory and subdirectories, **do not** store design files or personal files in the \PALASM hierarchy.

A:INSTALL [Enter]

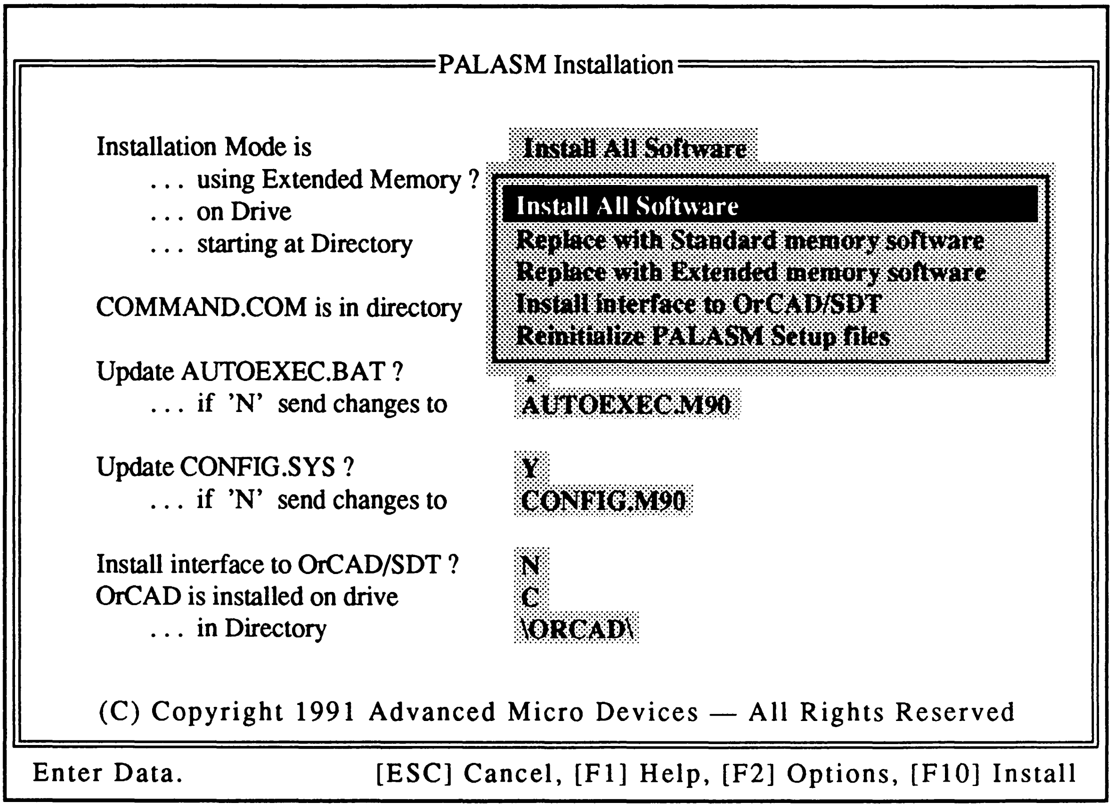
1. Place installation disk 1 into drive A, type the command shown in the left column, then press [Enter].

   The AMD copyright screen appears.

[Enter]

2. Press any key to dismiss the copyright notice and proceed with the installation.

   The installation menu and form appears; the menu covers part of the form.



```
                        PALASM Installation

   Installation Mode is               Install All Software
      . . . using Extended Memory ?   Install All Software
      . . . on Drive                  Replace with Standard memory software
      . . . starting at Directory     Replace with Extended memory software
                                      Install interface to OrCAD/SDT
   COMMAND.COM is in directory        Reinitialize PALASM Setup files

   Update AUTOEXEC.BAT ?
      . . . if 'N' send changes to    AUTOEXEC.M90

   Update CONFIG.SYS ?                Y
      . . . if 'N' send changes to    CONFIG.M90

   Install interface to OrCAD/SDT ?   N
   OrCAD is installed on drive        C
      . . . in Directory              \ORCAD\

   (C) Copyright 1991 Advanced Micro Devices — All Rights Reserved

 Enter Data.          [ESC] Cancel, [F1] Help, [F2] Options, [F10] Install
```

Commands on the menu allow you to selectively install the software, as follows.

- **Install All Software** installs all software and related files as you specify on the form beneath the menu.

  You must use this command initially. Then use others as needed since they affect only portions of the installed software.

- **Replace with Standard memory software** overwrites the existing executable files with those required for standard memory.

- **Replace with Extended memory software** overwrites current executable files with those required for extended memory.

- **Install interface to OrCAD/SDT** installs the library and other files required to use the schematic interface.

Important: OrCAD/SDT III is not installed using this command. A copy of the OrCADSDT.OVL file, with the correct library prefix and files, must reside in each schematic-based design directory to support schematic capture. The correct prefix and file entries are shown next.

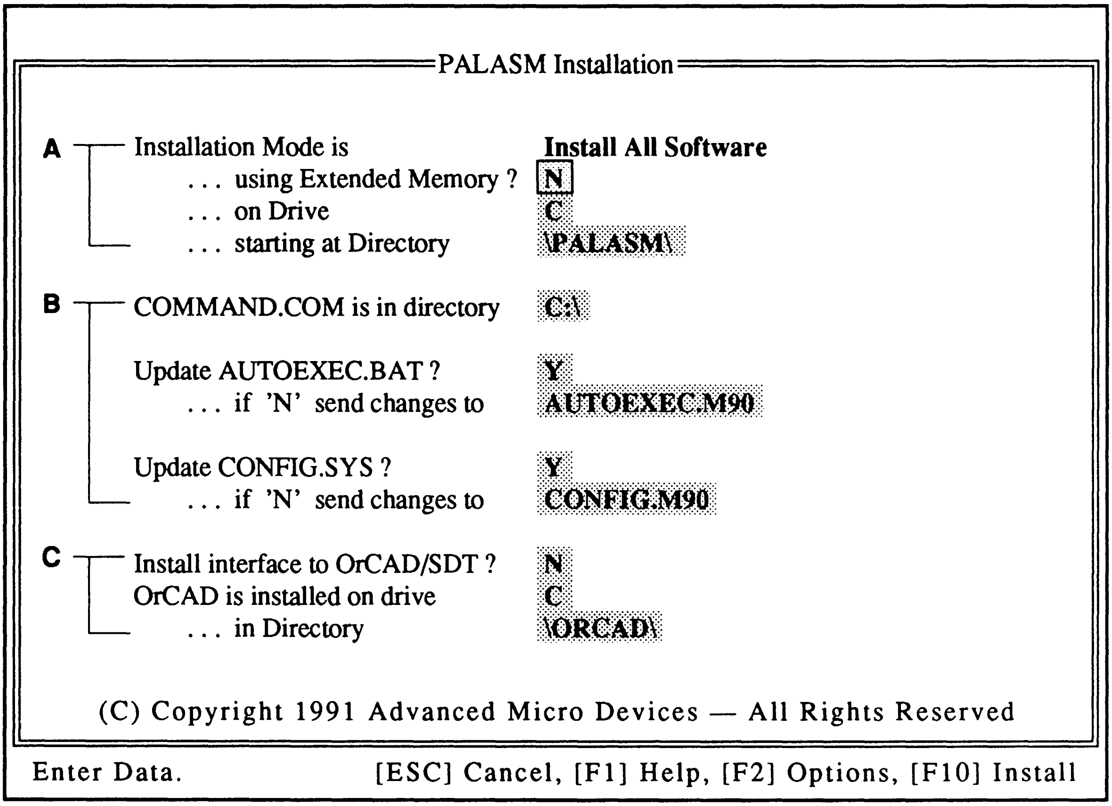
```
LP - Library Prefix      <drive> : \PALASM\LIB\
LF - Library Files
        MACH.LIB
        LOGIC.LIB
        74XX.LIB
```

- **Reinitialize PALASM Setup files** reinstates the standard SETUP.PAL, CONFIG.SYS, and AUTOEXEC.BAT files, when you want to reset all files to their original state and start over.

[Enter]

3. Select Install All Software and press [Enter].

   The menu is dismissed and the form is now completely visible, as shown next.



```
=================PALASM Installation=================

Installation Mode is                 Install All Software
      ... using Extended Memory ?    N
      ... on Drive                   C
      ... starting at Directory      \PALASM\

COMMAND.COM is in directory          C:\

Update AUTOEXEC.BAT ?                Y
      ... if 'N' send changes to     AUTOEXEC.M90

Update CONFIG.SYS ?                  Y
      ... if 'N' send changes to     CONFIG.M90

Install interface to OrCAD/SDT ?     N
OrCAD is installed on drive          C
      ... in Directory               \ORCAD\

(C) Copyright 1991 Advanced Micro Devices — All Rights Reserved

Enter Data.       [ESC] Cancel, [F1] Help, [F2] Options, [F10] Install
```

The information on this form is divided into three groups of text fields.

A. Installation mode data

   These fields identify the extended memory option and specify the destination drive and starting directory for the installation.

Note: When you select the Replace with Extended memory command from the menu, enter the letter Y beside using Extended memory.

```
... using Extended memory?          Y
```

Similarly: When you select the Install interface to OrCAD/SDT command on the menu, you must enter the letter Y in the corresponding field in this group.

```
Install interface to OrCAD/SDT?     Y
```

B. DOS file data

Fields in this group specify the location of the COMMAND.COM file and indicate whether the AUTOEXEC.BAT and CONFIG.SYS files should be updated automatically. Discussions 1.3.1 and 1.3.2 define the information that's added during the automatic update.

Important: If you do not update files automatically, new data is added **either** to the default file **or** to the file you specify. In this case, you may need to manually update information in the .BAT or .SYS files before you can use the PALASM 4 software.

C. PALASM interface to OrCAD/SDT

In these fields, you specify options to install the optional PALASM 4 interface to OrCAD/SDT III and identify the drive and directory where the schematic-capture software is installed.

Note: When you select the Install interface to OrCAD/SDT command on the menu, you must enter the letter Y in the corresponding field.

```
Install interface to OrCAD/SDT?     Y
```

field ...
response

4. Change any options in the form: press an arrow key to highlight (activate) the field, type the appropriate information, and press [Enter].

   The next field on the form is automatically activated.

After entering appropriate data into all fields,

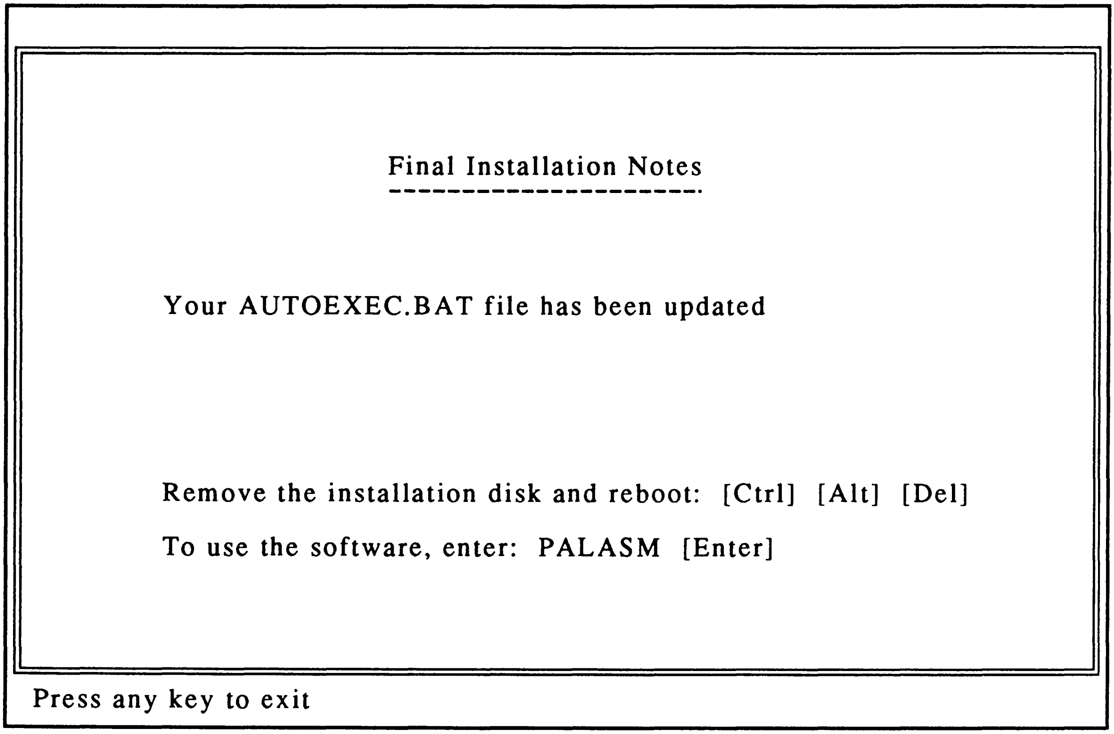
[F10]

5. Confirm your specifications and initiate the installation by pressing [F10].

   A window opens in the lower half of the screen and the process begins. Messages keep you informed and you are prompted to insert disks as required.

> **Important**: Should a box appear informing you about disk space, you must delete old files to create more space before you can install the software.
>
> If you are installing the PALASM 4 software on this workstation for the first time, just delete files you no longer need. However, if you are installing the PALASM 4 software for a second or subsequent time, you can continue with the installation and files will be overwritten automatically.

You are informed when the installation is complete. Messages depend on the data you specified on the installation form.



Final Installation Notes

Your AUTOEXEC.BAT file has been updated

Remove the installation disk and reboot: [Ctrl] [Alt] [Del]

To use the software, enter: PALASM [Enter]

Press any key to exit

To use the software, you need to restart the computer. However, you may want to check the .BAT and .SYS files and the configuration as explained under 1.2.2 before doing so.

[Ctrl] [Alt] [Del]

6. Remove the diskette and restart the computer. Hold down the [Ctrl] and [Alt] keys while you press the [Del] key.

1.2.2 CONFIGURATION

Following the installation you may need to configure your system as follows.

1. Verify or add data to the .BAT and .SYS files as discussed under 1.3.1 and 1.3.2.

If you are using schematic capture,

2. Verify that an OrCADSDT.OVL file, with the correct prefix and file entries, is located in each MACH schematic-based design directory.[1]

If you are using a monochrome screen, the following steps are recommended.

```
CD \PALASM\DAT [Enter]
```

3. Change the directory to \PALASM\DAT.

```
DEL PALASM.PCX [Enter]
```

4. Delete PALASM.PCX.

```
SET MODE=MONO [Enter]
```

5. Set mode to mono. Also include this command in your AUTOEXEC.BAT file.

The next discussion shows the information that should appear in the AUTOEXEC.BAT and CONFIG.SYS files.

[1] Refer to discussion 1.2.1 for the correct library prefix and files.

1.3 FILE UPDATES

You are asked about updating the .BAT and .SYS files on the installation form. The following discussions indicate the information that is added during the automatic update. If you did not specify an automatic update, you must ensure that this information appears in the appropriate file.

1.3.1 AUTOEXEC

The following information should appear in the AUTOEXEC.BAT file before you can use the PALASM 4 software.

```
PATH C:\PALASM\EXE;C:\DOS;
```

- A new PATH declaration includes the specified COMMAND.COM directory, the path to the PALASM executable files, and any path information previously defined.

```
SET PALASM=C:\PALASM\
```

- A new variable indicates the starting directory for the software installation.

```
SET ORCAD=C:\ORCAD\
```

- A new variable indicates where the OrCAD/SDT III software is installed.

If you did not update your .BAT file, check to ensure the appropriate entries are included.

1.3.2 CONFIG.SYS

The following line should appear in the CONFIG.SYS file if the current files variable is set to less than 20.

```
files=20
```

Important: If you entered the letter N in the update CONFIG.SYS field, you must manually edit the file so that the variable is set to 20 or greater.

CHAPTER 2

DESIGN ENTRY DEMONSTRATION

CONTENTS

2 DESIGN ENTRY DEMONSTRATION

Steps in this guide demonstrate various design entry methods.

- You can create a text-based design following the steps under discussion 2.1.
- You can begin a schematic-based MACH-device design using steps provided in discussion 2.2.
- You can combine multiple PDS files to produce a single MACH-device design using steps in discussion 2.3.

Important: Each demonstration assumes you have installed the PALASM 4 software and configured your workstation as described in Chapter 1.

Activities here assume you are familiar with the following; therefore, information from the sources below is not repeated.

- OrCAD/SDT III software and manuals
- PALASM language syntax
- device datasheets

Important: The **left column** of this guide shows prompts and **identifies** which key you press, what you type, or what you must select on the screen to respond correctly. Sometimes, a prompt appears in the left column before the required response to help you track the process.

The **right column** of this guide provides information and numbered steps that **explain** what to do or select in more detail and describes what happens on the screen.

Tip: **For a quick tour**, you can type the text, select the boxed item, or press the key named in the left column; refer to the right column for details or clarification.

2.1 CREATING A TEXT-BASED DESIGN

Here, you'll use the PALASM 4 software to start a text-based design for any PAL, PLS, or MACH device you choose.

To begin from the DOS operating system,

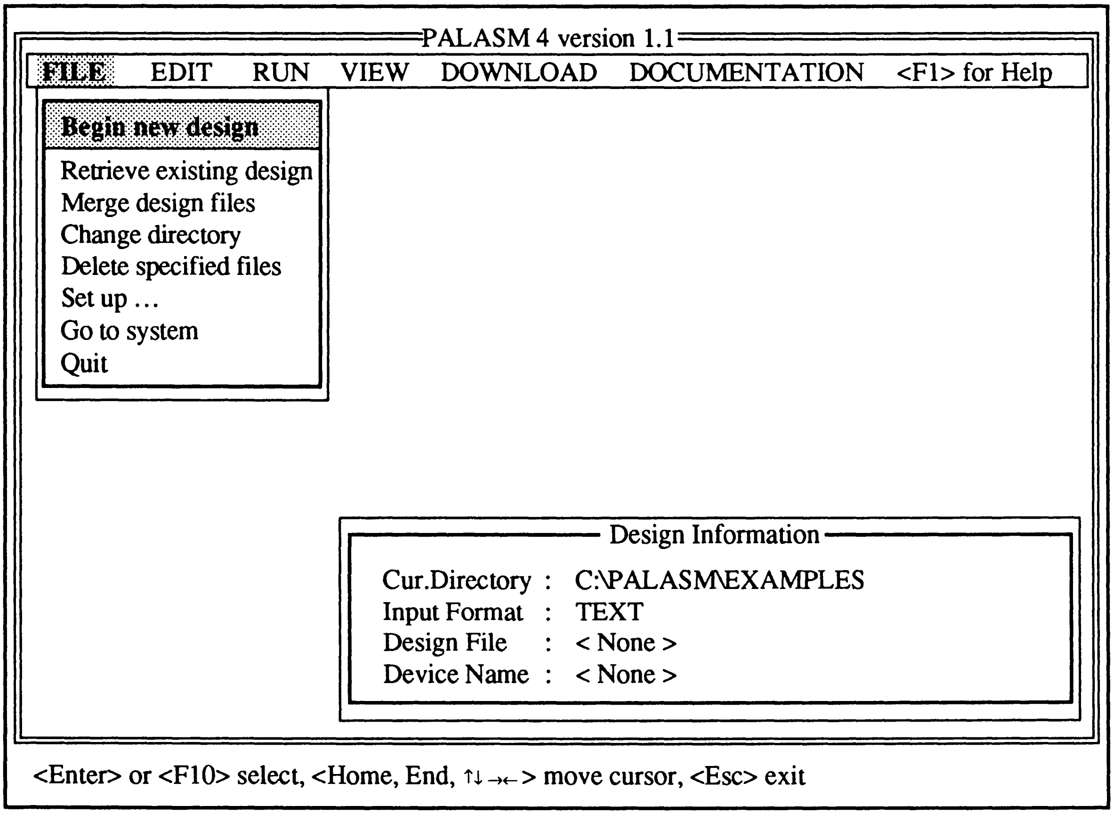
```
C:\>
  PALASM [Enter]
```

1. Execute the PALASM 4 software: type PALASM and press [Enter].

   The AMD logo and copyright notice appear. A message at the bottom of the screen prompts you.

```
Press any key ...
  [Space bar]
```

2. Dismiss the copyright notice and continue: press the [Space bar] or any other key.



Cur.Directory : C:\PALASM\EXAMPLES
Input Format : TEXT
Design File : < None >
Device Name : < None >

<Enter> or <F10> select, <Home, End, ↑↓→←> move cursor, <Esc> exit

Menu names appear across the top of the screen. The File menu is open and the first item is highlighted, as shown in the previous figure. Depending on the working environment you set up, information for the current design appears in the lower-right corner.

PALASM 4 is easy to work with. The following features are standard.

- Menus list all available commands; those followed by ellipses, such as Set up, display a submenu of additional commands.

- Forms are provided where you can fill in additional information needed to complete a command.

- A list appears with one or more choices, if an option field is active in a form when you press [F2].

The sequence of tasks to create a new text-based design follows.

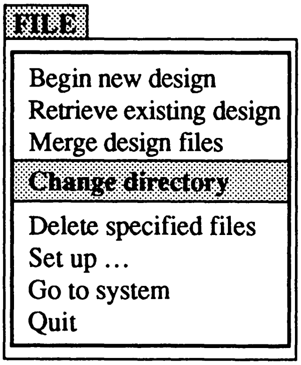
- Begin a new design using the PDS declaration-segment form.

- Complete the design using a text editor.

2.1.1 BEGIN THE DESIGN

All designs you create are stored in the current working directory. Before you create the new design, it's a good idea to verify the current working directory.

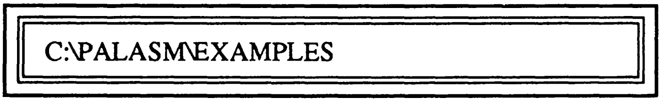
To verify or change the working directory,



1. Select the Change directory command from the File menu: type the first letter of the command, which is a capital letter, when the menu is open.

   Alternatively, you can press the down arrow to highlight the command, then press [Enter] to select it.

   A form like the one below appears showing the current working directory path. The path on your screen may differ.



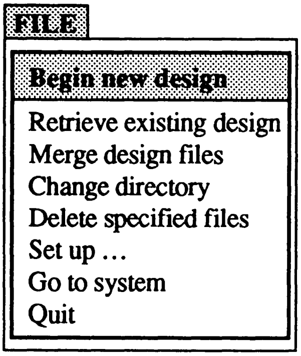
C:\PALASM\EXAMPLES

If the directory path on your screen does not match the one shown, complete step 2. Otherwise, just press [F10] to accept the directory path shown.

C:\PALASM\EXAMPLES
[F10]

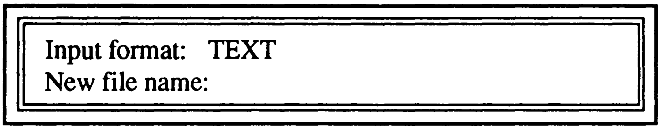
2. Type a path name as shown beside this paragraph and press [F10] to confirm it **or** press [F10] if you're satisfied with the existing path.

To begin the new design,



1. Select the Begin new design command from the File menu: type the letter B.

   A form like the one below appears where you specify the file type and name. Text is the default format.



Input format: TEXT
New file name:

[Enter]

2. Activate the New file name field: press [Enter].

   The active field is highlighted.

TEXT1.PDS

3. Type the name of the file using standard DOS naming conventions and include a .PDS extension.

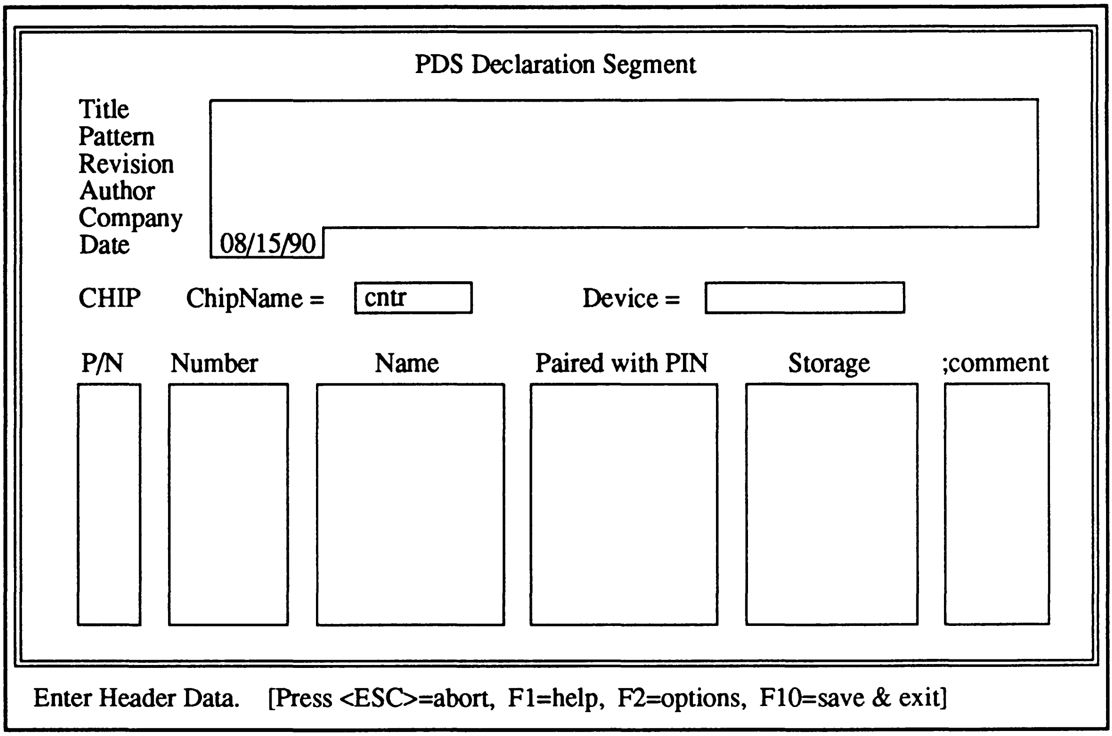
[F10]

4. Confirm the specifications in the form: press [F10].

   The form is dismissed and the process is initiated.

Important: If the name you specify matches an existing name, you're asked if you want to overwrite the existing file. The file is overwritten if you respond by typing the letter Y. However, if you type the letter N, you can change the name in the form and proceed.

A PDS declaration-segment form appears, as shown below. The fields on this form assist you in completing the declaration segment of a PDS file. The Title field is active when the form appears.



PDS Declaration Segment

Title
Pattern
Revision
Author
Company
Date 08/15/90

CHIP ChipName = cntr Device =

Enter Header Data. [Press <ESC>=abort, F1=help, F2=options, F10=save & exit]

The text and option fields in this form are described as you fill in data using the steps below.

2.1.2 FILL IN THE PDS DECLARATION-SEGMENT FORM

Text fields are provided for Title through Company data. Each can contain up to 59 characters. You can use any combination of letters, numbers, or symbols in these fields.

To complete the Title through Company fields,

any title [Enter]

1. Type a title for the design and press [Enter] to move to the Pattern field.

Note: If you make a mistake you want to **correct**, **either** press the backspace key to erase characters, then retype **or** use the arrow keys to move the cursor back and type over existing text.

your own pattern [Enter]

2. Enter a pattern and activate the Revision field: type, then press [Enter].

a version number [Enter]

3. Enter a revision number and activate the Author field.

your name [Enter]

4. Enter your name in the field and activate the Company field.

company name [Enter]

5. Enter a company name, then activate the Date field.

Note: You can return to a previous field by pressing the up-arrow key to activate the field; then make changes.

The Date field provides today's date automatically, as specified by the operating system. You can change the date using only a ##/##/## format.

[Enter]

6. Skip the Date field: press [Enter] to confirm its content.

ChipName is a text field that automatically displays the design-file name preceded by an underscore and without the .PDS extension. However, you can specify any name of up to eight alphanumeric characters, including the underscore character.

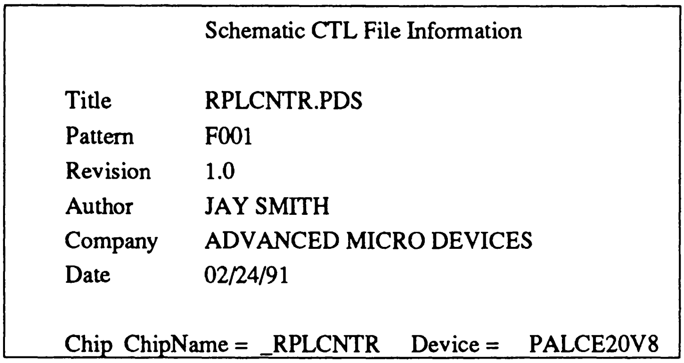
chipname [Enter]

7. Enter a chip name field and press [Enter].

The Device = field is an option field. A list appears automatically when this field is active. The list includes all PLD and MACH device types. You must specify a device type before saving information and leaving the form.

↓ [Enter] 8. Select a device type: press the down arrow to highlight your choice, then press [Enter].

The top part of the form on your screen should look something like the one below.



Schematic CTL File Information

Title	RPLCNTR.PDS
Pattern	F001
Revision	1.0
Author	JAY SMITH
Company	ADVANCED MICRO DEVICES
Date	02/24/91

Chip ChipName = _RPLCNTR Device = PALCE20V8

The lower-half of the form includes both text and option fields, which will be identified as you fill in each one.

The P/N field is an option field that identifies the statement as either a pin or node statement. Pin is the default.

To enter a pin statement,

[Enter] 1. Activate the P/N option field: press [Enter].

The field is highlighted and Pin appears automatically.

Since you are entering a pin statement, you can proceed to the Number field.

[Enter] 2. Activate the Number field.

The Number field is a text field where you enter a pin or node location. You can enter **either** a whole number that fixes the location on the device **or** a question mark, ?, which defines a floating location.[1]

? [Enter]

3. Enter a floating pin location and activate the Name field: type a question mark and press [Enter].

Name is a text field where you enter a pin or node name.[2]

Q3_WD [Enter]

4. Enter a pin name and activate the next field.

Paired with pin is a text field where you can enter an optional node attribute. This attribute applies only to node statements.[3] For the moment, you'll skip this field.

[Enter]

5. Skip the Paired with pin field and activate the Storage field: press [Enter].

Storage refers to an optional storage type. Combinatorial is the default that is used when this field is left blank. You can display a list of the choices by pressing [F2] when the field is active.

[F2]

6. Display a list of storage options: press [F2].

   A list appears with several choices: a blank to specify the default is first, followed by Combinatorial, Latched, and Registered.

R [Enter]

7. Select Registered for this design, then activate the next field: type the letter R and press [Enter].

[1] Refer to Section IV, Chapter 10, for details about using the question mark to float pin and node locations.

[2] Refer to Section IV, Chapter 10, for details about naming syntax in pin and node statements.

[3] Refer to Section IV, Chapter 10, for details about pairing a node with a pin.

You can add an optional signal-type comment to the statement using the Comment field. You can leave this field blank and add optional comments of another nature when you edit the PDS file in the text editor.

Each comment is preceded by a semicolon in the PDS file but not on the form. You can display a list of options by pressing [F2].

[F2]

8. Display a list of signal-comment options.

   The list appears and includes a blank that you can select to add the semicolon to the PDS file to prepare for other comments you may include later. In addition the following are listed: Input, Output, IO, Clock, and Enable.

I [Enter]

9. Select the Input option, then activate the P/N field: type the letter I and press [Enter].

   The comment is added to the field and the P/N field is activated. Pin appears automatically as the default.

Next, you'll enter a node statement.

[F2]

1. Display a list of options for the P/N field: press [F2].

   Again, a list appears with appropriate choices.

N [Enter]

2. Select the Node option and activate the Number field: type the letter N, then press [Enter].

? [Enter]

3. Enter a floating node location and activate the Name field: type a question mark and press [Enter].

Q2_WD [Enter]

4. Enter a node name, then activate the next field.

You may recall that the optional **Pair** attribute applies only to node statements. Next, you'll pair this node with the previous pin by specifying the pin name in the Paired with pin field.

Paired with pin
Q3_WD [Enter]

5. Type the name of the previous pin to pair the node with it, then activate the Storage field.

   The storage field becomes active and is filled with the storage type specified for the pin with which the node is paired.

When you pair a node with a pin, they must be of the same storage type.

[Enter]

6. Accept the existing type, Registered: press [Enter].

   Registered is accepted and the comment field is activated.

Again, the comment reflects the same signal type for the node as the input pin with which the node is paired.

[Enter]

7. Accept the existing comment, Input: press [Enter].

P/N

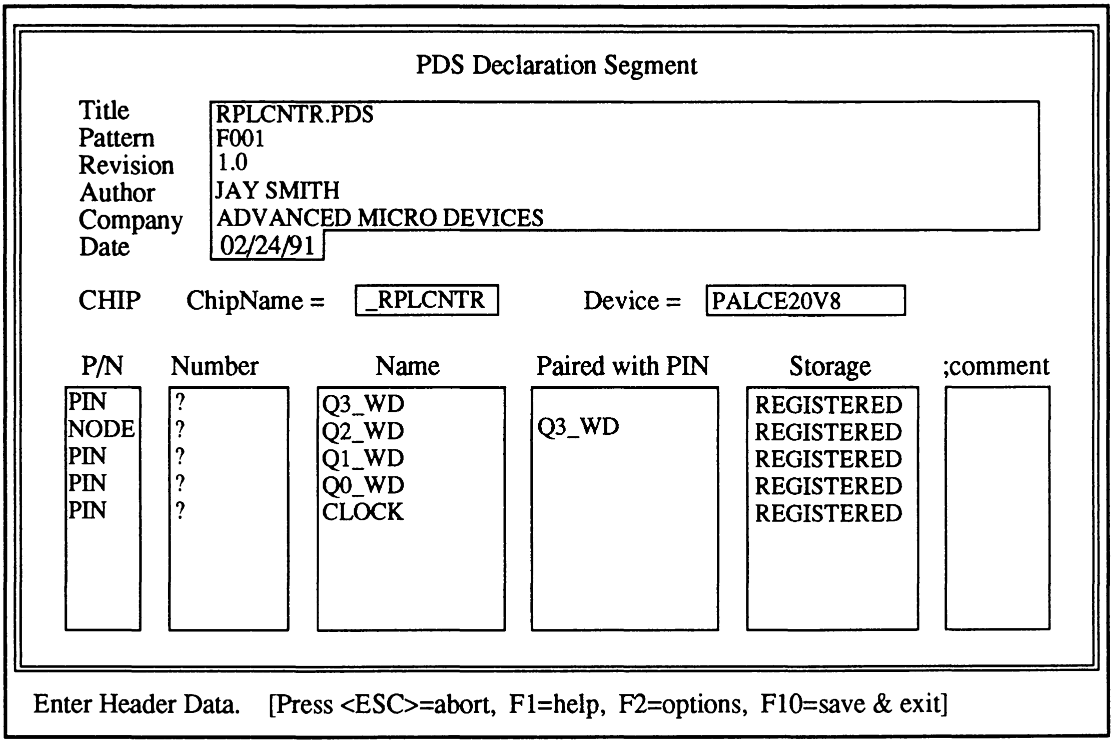
8. Continue adding pin and node statements using the steps above until all are defined for your design.

Once you leave the declaration-segment form, you cannot return to it. From that point on, you'll have to use the text editor to complete specifications for the PDS file.

[F10]

9. Confirm all specifications when you finish adding pin and node statements: press [F10].

Your form should look something like the one shown next.



PDS Declaration Segment

Title	RPLCNTR.PDS
Pattern	F001
Revision	1.0
Author	JAY SMITH
Company	ADVANCED MICRO DEVICES
Date	02/24/91

CHIP ChipName = _RPLCNTR Device = PALCE20V8

P/N	Number	Name	Paired with PIN	Storage	;comment
PIN	?	Q3_WD		REGISTERED	
NODE	?	Q2_WD	Q3_WD	REGISTERED	
PIN	?	Q1_WD		REGISTERED	
PIN	?	Q0_WD		REGISTERED	
PIN	?	CLOCK		REGISTERED	

Enter Header Data. [Press <ESC>=abort, F1=help, F2=options, F10=save & exit]

You're ready to complete the design.

2.1.3 COMPLETE THE PDS FILE USING A TEXT EDITOR

After you confirm specifications in the declaration segment form, you're transferred to the text editor where the PDS file is displayed. The information you supplied in the declaration-segment form is present, and looks something like the one shown below.

```
;PALASM Design Description

;------------------------------Declaration Segment------------------------------

TITLE              RPLCNTR.PDS
Pattern            F001
Revision           1.0
Author             JAY SMITH
Company            ADVANCED MICRO DEVICES.
Date               02/24/91

Chip  _RPLCNTR        PALCE20V8

;------------------------------Pin Declarations---------------------------------

 PIN      ?       Q3_WD                REGISTERED          ;INPUT
 PIN      ?       Q2_WD                REGISTERED          ;INPUT
 PIN      ?       Q1_WD                REGISTERED
 PIN      ?       Q0_WD                REGISTERED

;---------------------------Boolean Equation Segment----------------------------
EQUATIONS

;-----------------------------Simulation Segment--------------------------------
SIMULATION

;-------------------------------------------------------------------------------
```

The Equations and Simulation segments are blank. You can add or edit any information in the PDS file using the text editor. For example, you can add more pin and node statements and the equations and simulation commands needed to complete the design.[4]

The design you produce can be based **either** upon Boolean equations **or** state-machine language using standard PALASM language syntax.

Some sample Boolean equations are shown below.

```
...
;-------------------------Boolean Equation Segment-------------------------------
EQUATIONS

Q3_WD = ((Q3_WD :+: /(Q2_WD + Q1_WD + Q0_WD + /ENABLE)) * ((/TC * ENABLE) *
    /LDWCNTR)) + (Q3_WD * ((/TC * ENABLE) * /LDWCNTR)) + (D3 * LDWCNTR)
Q3_WD.clkf = CLOCK

Q3_WD.setf = GND
Q3_WD.rstf = GND
Q2_WD = ((Q2_WD :+: /(Q1_WD + Q0_WD + /ENABLE)) * ((/TC * ENABLE) *
      /LDWCNTR)) + (Q2_WD * (/((/TC * ENABLE) * /LDWCNTR)) + (D2 * LDWCNTR)
Q2_WD.clkf = CLOCK
...
```

[4] Refer to Section IV, Chapter 10, for details on the language syntax and simulation commands.

You can enter simulation commands in the PDS file, as shown in the sample below.

```
;-------------------------------Simulation Segment------------------------------------
   SIMULATION

   TRACE_ON CLOCK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 TC LDACNTR ENABLE
   SETF /CLOCK /D0 /D1 /D2 /D3 /D4 /D5 /D6 /D7 /LDACNTR /ENABLE
   SETF /LDWCNTR
   CLOCKF CLOCK
   CHECK /Q0 /Q1 /Q2 /Q3 /Q4 /Q5 /Q6 /Q7 /TC
   ...
   FOR I:=0 TO 127 DO
             BEGIN
                 CLOCKF CLOCK
             END
   CHECK Q0 Q1 Q2 Q3 Q4 Q5 Q6 /Q7
   TRACE_OFF
```

Once you complete the design, you can define logic synthesis, compilation, and MACH-fitting options; then compile, simulate, and download the design as usual.[5]

Note: Because there is no way of knowing which text editor you are using, specific commands to finish the design are not provided here.[6]

[5] Refer to discussions in Chapter 3 of this section for a demonstration of setting up and running compilation, fitting, and simulation processes using a schematic-based design. Processes are the same regardless of the entry method. Also, refer to Section IV, Chapter 9, for details about each of the options available under the Set up command on the File menu and the commands on the Run menu.

[6] If you are using the AMD-supplied text editor, refer to Section V, Appendix A, for details about the PLD text editor. Also, refer to Section IV, Chapter 10, for details about PALASM language syntax for equations and pin and node statements.

2.2 SCHEMATIC-BASED DESIGN ENTRY

Here, you can use the PALASM 4 software to start any schematic-based design for a MACH device.

To begin from the DOS operating system,

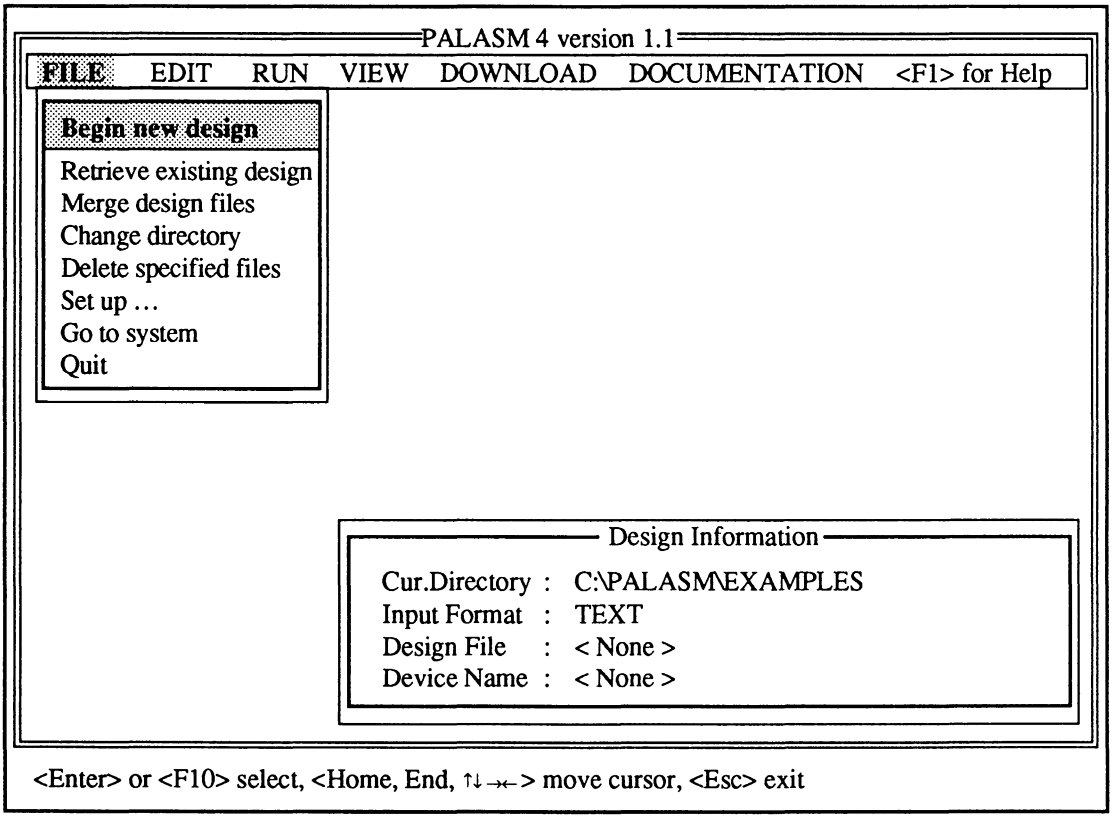
C:\>
PALASM [Enter]

1. Execute the PALASM 4 software: type PALASM and press [Enter].

   The AMD logo and copyright notice appear. A message at the bottom of the screen prompts you.

Press any key ...
[Space bar]

2. Dismiss the copyright notice and continue: press the [Space bar] or any other key.



```
PALASM 4 version 1.1
FILE   EDIT   RUN   VIEW   DOWNLOAD   DOCUMENTATION   <F1> for Help

Begin new design
Retrieve existing design
Merge design files
Change directory
Delete specified files
Set up ...
Go to system
Quit

Design Information
Cur.Directory :  C:\PALASM\EXAMPLES
Input Format  :  TEXT
Design File   :  < None >
Device Name   :  < None >

<Enter> or <F10> select, <Home, End, ↑↓→←> move cursor, <Esc> exit
```

Menu names appear across the top of the screen. The File menu is open and the first item is highlighted, as shown in the figure above. Depending on the working environment you set up, information for the current design appears in the lower-right corner.

PALASM 4 is easy to work with. The following features are standard.

- Menus list all available commands; those followed by ellipses, such as Set up, display a submenu of additional commands.

- Forms are provided where you can fill in additional information needed to complete a command.

- A list appears with one or more choices if an option field is active in a form when you press [F2].

The sequence of tasks to create a new schematic-based design follows.

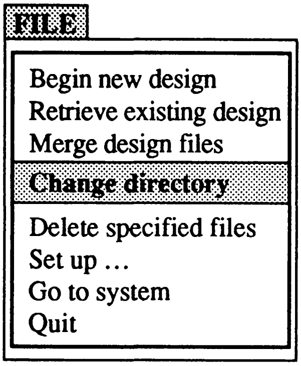
- Begin a new design using the schematic control-file form.

- Complete the design using the OrCAD/SDT III schematic editor.

2.2.1 BEGIN THE DESIGN

All designs you create are stored in the current working directory. Before you create the new design, it's a good idea to verify the current working directory.

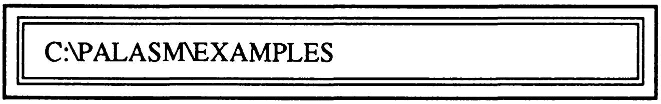
To verify or change the working directory,



1. Select the Change directory command from the File menu: type the first letter of the command, which is a capital letter, when the menu is open.

   Alternatively, you can press the down arrow to highlight the command, then press [Enter] to select it.

   A form like the one below appears showing the current working directory path. The path on your screen may differ.

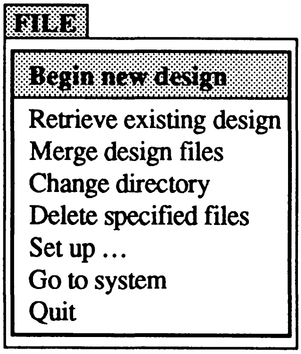

C:\PALASM\EXAMPLES

If the directory path or your screen does not match the one shown, complete step 2. Otherwise, just press [F10] to accept the directory path shown.

C:\PALASM\EXAMPLES
[F10]

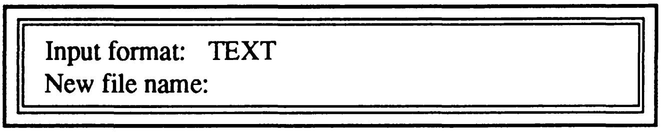
2. Type a path and press [F10] to confirm to confirm it.

To begin the new design,



1. Select the Begin new design command from the File menu: type the letter B.

   A form like the one shown next appears where you specify the file type and name: Text is the default format.



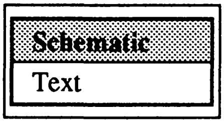
Input format: TEXT
New file name:

For this design, you must change the input format to schematic.

Input format: TEXT
[F2]

2. Display the input-format options: press the [F2] key.

   The first option is highlighted when the list appears.



Schematic
Text

3. Select Schematic: type the letter S.

   You're returned to the form and Schematic is highlighted.

[Enter]

4. Activate the next field: press [Enter].

   The field is highlighted to show that it is active.

This text field is where you enter the name of the design you will create.

File name:
sample1.SCH [F10]

5. Type the name shown at left, then confirm the specifications: type the name then press [F10].

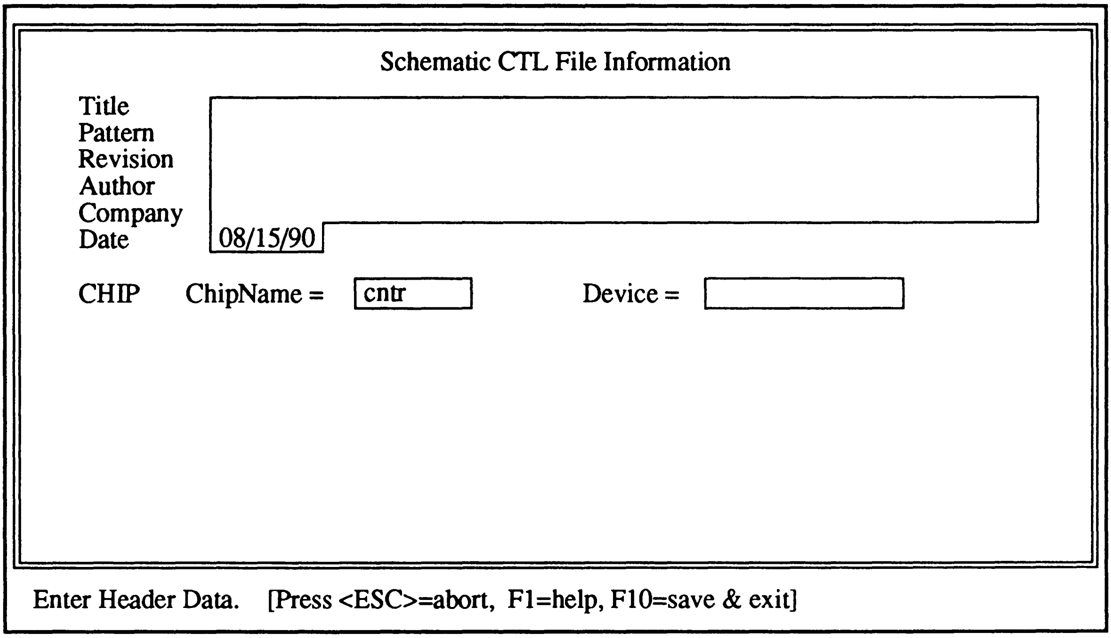
Important: If the name you specify matches an existing name, you're asked if you want to overwrite the existing file. The file is overwritten if you respond by typing the letter Y. However, if you type the letter N, you can change the name in the form and proceed.

After you confirm a schematic-based format and design name, two files are automatically created.

- An empty schematic worksheet file is created using the name you specified.

- An empty control file is created using the design name with a .CTL extension, design.CTL.

You're immediately transferred to the control-file form shown below.



Schematic CTL File Information

Title
Pattern
Revision
Author
Company
Date 08/15/90

CHIP ChipName = cntr Device =

Enter Header Data. [Press <ESC>=abort, F1=help, F10=save & exit]

When schematic data is converted to a PDS format, the information in this form provides the declaration segment of the PDS file.

Important: When you create a schematic-based design for a MACH device, you must specify the device type in the schematic control-file form. Any device type you specify in the schematic is ignored.

Also: Simulation commands for schematic-based designs are best placed in an auxiliary simulation file that you create using a text editor. No simulation commands appear in the PDS file generated from converted schematic data.

You may recognize that this form is similar to the one you used to enter declaration-segment data for a text-based design. However, this form does not include any pin or node statements as these are derived from schematic data during compilation.

Tip: If you know how to enter data in this form, you can skip some of the following discussions. However, **do complete the activities** below so you can enter the schematic editor automatically upon confirmation, as described next.

2.2.2 FILL IN THE SCHEMATIC CONTROL-FILE FORM

The text and option fields in this form are described as you fill in data using the steps below. If you know how to enter data here, you may want to complete the keystrokes in the left column without reading the right column.

Text fields are provided for Title through Company data. Each can contain up to 59 characters. You can use any combination of letters, numbers, or symbols in these fields.

To complete the Title through Company fields,

any title [Enter]

1. Enter a design title: type, then press [Enter] to activate the Pattern field.

   The next field is highlighted to show it is active.

Note: If you make a mistake you want to **correct**, **either** press the backspace key to erase characters then retype **or** use the arrow keys to move the cursor back and type over existing text.

your own pattern [Enter]

2. Enter a pattern and activate the Revision field: type, then press [Enter].

a version number [Enter]

3. Enter a revision number and activate the Author field.

your name [Enter]

4. Enter your name in the field and activate the Company field.

company name [Enter]

5. Enter a company name, then activate the Date field.

> **Note**: You can return to a previous field by pressing the up-arrow key to activate the field; then make changes.

The Date field provides today's date automatically, as specified by the operating system. You can change the date using only a ##/##/## format.

[Enter]

6. Skip the Date field: press [Enter].

ChipName is a text field that automatically displays the design-file name preceded by an underscore and without the .PDS extension. However, you can specify any name of up to eight alphanumeric characters and the underscore character.

chipname [Enter]

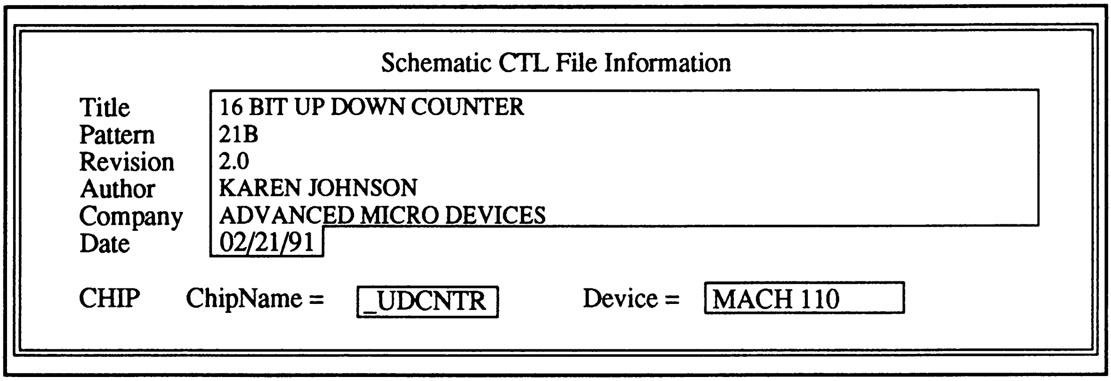
7. Enter a ChipName and activate the Device field.

The Device = field is an option field. A list appears automatically when this field is active. The list includes all PLD and MACH device types. You must specify a device type before saving information and leaving the form.

↓ [Enter]

8. Select a device type: press the down arrow to highlight your choice, then press [Enter].

The form on your screen should look something like the one below.



Schematic CTL File Information

Title	16 BIT UP DOWN COUNTER
Pattern	21B
Revision	2.0
Author	KAREN JOHNSON
Company	ADVANCED MICRO DEVICES
Date	02/21/91

CHIP ChipName = _UDCNTR Device = MACH 110

[F10]

9. Confirm specifications in the form when you are satisfied: press [F10].

After you create and confirm specifications, you're returned to the PALASM environment for a few seconds before you're automatically transferred to the OrCAD editor.[7]

> **Note**: You can return to the control-file form to edit information later, using the Control file for schematic command on the PALASM Edit menu.

At this point, you are ready to create the schematic.

2.2.3 CREATE THE SCHEMATIC

When you exit the control-file form and enter the OrCAD environment, a blank worksheet becomes available with the name you specified earlier. The AMD-supplied MACH library is also available.

[7] Refer to the *OrCAD/SDT III Schematic Design Tools* manual for details about using the schematic editor.

Press any key...
[Space bar]

1. Dismiss the logo and continue: press [Space bar] or any other key.

   The copyright screen appears with a prompt at the top of the screen.

[Space bar]

2. Dismiss the copyright screen and continue.

   The OrCAD editor becomes available and a new worksheet appears.

OrCAD/SDT III lists available commands on menus and submenus. You access all commands through the main menu as follows.

[Enter]

3. Display the main menu: press [Enter].

   All primary commands are listed on the main menu.

There are several ways to select a command in OrCAD; this guide uses the first method listed below.

- Type the capital letter at the beginning of the command, as you do with the PALASM 4 software.
- Press arrow keys to highlight a command and press [Enter] to select it, as you can with the PALASM 4 software.
- Use the mouse to highlight the command and click the left button to select it.

To place elements from the AMD-supplied MACH library,

G INV [Enter]

1. Get a library element to place: type the letter G, the name of the element, then press [Enter].

   The specified element appears and a command line runs across the top of the screen.

—> P

2. Place the first instance: move the cursor to the location and type the letter P.

You can repeat step 2 to place another instance of the same element or end the sequences as shown next.

[Esc]

3. End the placement sequence: press [Esc].

G 74162 [Enter]
—> P [Esc]

4. Repeat the sequence to get and place other components.

To wire components,

P W

1. Initiate the Place Wire command: type the letters P and W.

—> B —> E

2. Begin and end the wire: move the cursor and type the letter B to begin the wire; move the cursor and type the letter E to end the wire.

To save the worksheet and return to the PALASM environment,

Q

1. Select the Quit command: type the letter Q.

   A submenu appears listing additional commands to choose from.

U

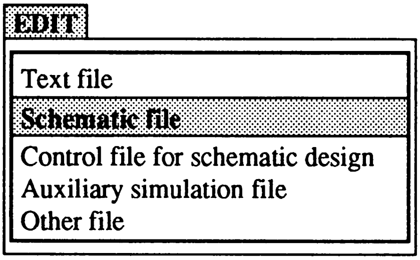
2. Select the Update file command: type the letter U.

Q A

3. Leave the OrCAD environment when you're finished with the schematic: select the Quit and Abandon edits commands.

   You're returned to the PALASM environment.

You can return to the schematic to add or edit data any time. To do so,

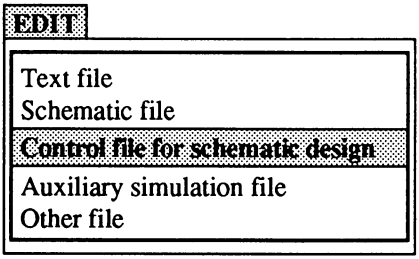


1. Select the Schematic file command from the Edit menu: press the right-arrow key to open the Edit menu and type the letter S.

...

2. Complete OrCAD commands as usual.

You can also edit the schematic control-file form anytime to change device type or other header information. To do so,



1. Select the Schematic file command from the PALASM Edit menu: press the right-arrow key to open the Edit menu and type the letter S.

...

2. Activate and edit fields as usual.

Once you complete the schematic, you can define logic synthesis, compilation, and MACH-fitting options and compile the design.[8] During compilation, schematic data is automatically converted to pin and node statements and Boolean equations. This data is combined with the schematic control-file you created when you began the design.

Subsequent editing and recompiling the schematic results in a new PDS file, so any editing you do to the

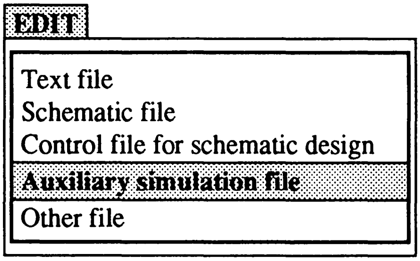
[8] Refer to Chapter 3 in this section for a detailed hands-on demonstration of these and related schematic-design-process activities.

PDS file is lost. Therefore, it's a good idea to create an auxiliary file of simulation commands.

The auxiliary simulation file contains the same information as the simulation segment of a PDS file, shown in the partial sample below.

```
SIMULATION

TRACE_ON CLOCK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 TC LDACNTR  ENABLE
SETF /CLOCK /D0 /D1 /D2 /D3 /D4 /D5 /D6 /D7 /LDACNTR /ENABLE
SETF /LDWCNTR
CLOCKF CLOCK
FOR I:=1 TO 300 DO
                  BEGIN
                         CLOCKF CLOCK
                  END
TRACE_OFF
```

The only difference between these two files is the auxiliary file is named after the schematic using a .SIM extension. To begin an auxiliary simulation file,



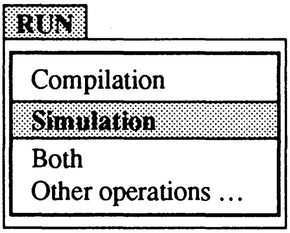
1. Select Auxiliary simulation file from the Edit menu: press the right-arrow key to open the Edit menu, then type the letter A.

   The text editor becomes available.

...

2. Enter simulation commands as usual.[9]

> **Note**: Because there is no way of knowing which text editor you are using, specific commands are not provided here.[10]



3. Simulate as usual.[11]

[9] Refer to Section IV, Chapter 10, for details the about simulation command syntax.

[10] If you are using the AMD-supplied text editor refer to Section V, Appendix A. Also refer to Section IV, Chapter 10, for details about PALASM language syntax.

[11] Refer to Chapter 3 in this section for a detailed hands-on demonstration of simulating a schematic-based design.

2.3 MERGING MULTIPLE PDS FILES

After you evaluate designs[12] to determine whether or not it is feasible to merge them into a single MACH-device design, you can begin the merge process.

- **Set Up**
  Initiate the process
  Specify setup options

- **Retrieve Files**
  First file
  Subsequent files

- **Resolve Conflicts**
  Review the detectable conflicts table
  Rename signals in the input buffer
  Bind signals together

- **Merge Files**
  Edit combined data
  Save combined in a file

- **Re-Engineer the Combined Design**

Following discussions identify each task and the order of activities as pertinent facts are described.[13]

Important: Unlike other tutorials in this manual, information you select or type is **not** shown in the left column. Everything you need is on the **right**.

Also: To complete activities below, use any PDS files you choose. Steps here assume that you know how to select commands and fill in and confirm specifications on forms.

[12] Refer to Section II, Chapter 4, for details about evaluating designs prior to merging them.

[13] Refer to Section IV, Chapter 9, for details about specific commands and forms.

2.3.1 SET UP

To merge, you initiate the process and set up the environment.

Initiate the Process

It's a good idea to verify the current working directory where the combined file will be stored.

1. Change the current working directory.

2. Select the Merge design files command on the File menu to initiate the process.

If the name you supply matches an existing name, you're asked if you want to use the data in the existing file.[14] You can type the letter Y to use the existing data or the letter N to overwrite it.

When the merge screen appears, four menus provide all commands for the process.

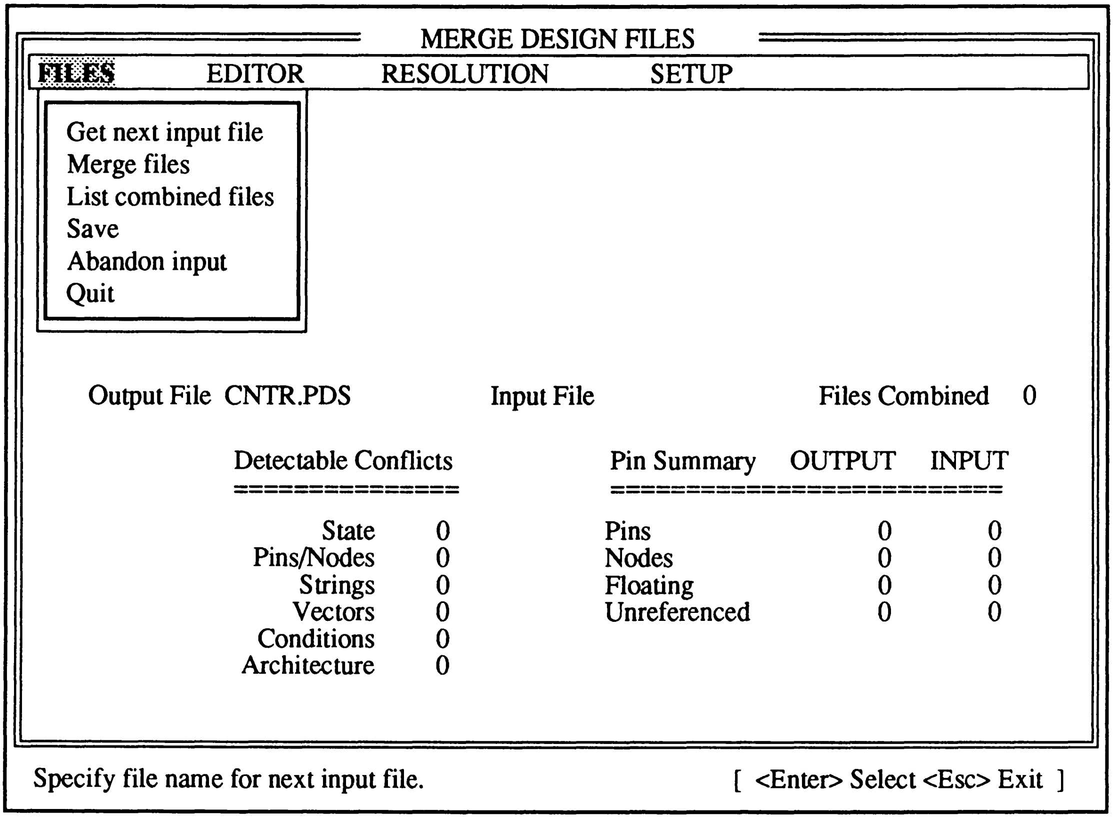
Files
Editor
Resolution
Setup

Status fields across the center of the screen identify the output-file name, current input-file name, and the number of files combined during this session.

Initially, the output-file name is specified. However, the input file name is not listed until you retrieve the first input file.

The detectable-conflicts and pin-summary tables reflect the status of a comparison that's made after you retrieve an input file or resolve conflicts. Messages and prompts appear at the bottom of the screen as usual.

[14] You can begin with existing data and merge other designs with the existing file.



```
Output File  CNTR.PDS          Input File              Files Combined    0

              Detectable Conflicts           Pin Summary    OUTPUT   INPUT
              ====================           ============================

                     State    0              Pins               0        0
                Pins/Nodes    0              Nodes              0        0
                   Strings    0              Floating           0        0
                   Vectors    0              Unreferenced       0        0
                Conditions    0
              Architecture    0
```

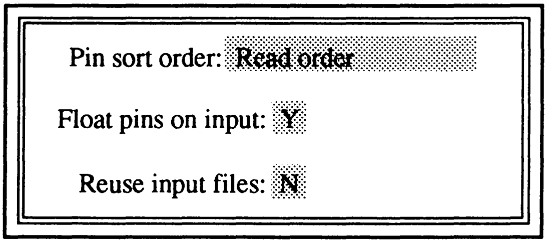
Specify file name for next input file.

Specify Setup Options

To specify the setup options,

1. Select the Options command from the Setup menu.

A form appears where you can specify the pin sort order, floating pins on input, and reusing input files, as shown next.



Pin sort order: Read order

Float pins on input: Y

Reuse input files: N

2. Display options for the pin sort order and select one.

It's a good idea to float pins on input, which assigns a question mark, ?, in the location field of all pin and node statements in the output file. This resolves pin-number conflicts and allows locations to be determined automatically during compilation and fitting.

3. Confirm options in the form: press [F10].

Next, you can specify a substitute-naming algorithm to use when a signal in the input buffer conflicts with a signal in the output buffer. The default, $_#, adds an underscore and a three-digit number to the signal name in the input buffer: _00n, for example.

Once you've set up the environment, you can begin the process.

2.3.2 RETRIEVE FILES

The merge procedure temporarily stores input and output data in separate memory buffers. Initially the input buffer is clear. The output buffer contains only a PDS-file shell, which contains empty declaration and equation segments.[15]

Note: The declaration segment contains only a few keywords. Header information for the combined design is taken from the first input file.

The output file name appears on the screen under the Files menu; the input file name area is blank.

[15] If you started the merge process using data from an existing file, the output buffer contains design information from that file. In this case, skip to the discussion on subsequent Files.

First File

To get the first file,

1. Select the Get next input file command on the Files menu to retrieve a design.

A form appears where you can **either** type a file name **or** display a list of files and select a name from the list.

- If the form contains *.pds, press [F2] or [Enter] to display a list of all PDS files.
- If the form contains *.*, press [F2] or [Enter] to display a list of all files; however, select only a PDS file.

When the file is retrieved, it is automatically parsed, expanded, and minimized. If errors are found in the input file, you must quit and correct them before you can merge the file. However, if no errors are found, the file is placed in the input buffer.

2. Review the pin-summary data on the screen.

The pin summary table provides the information shown in Table 1, next.

Table 1: Pin Summary

Fields		Definitions
Pins	48	Number of pins defined in the input and output files
Nodes	12	Number of nodes defined in the input and output files
Floating	60	Number of floating pins and nodes in the input and output files
Unreferenced	15	Number of named pins not used in equations

You can use the Abandon input file command on the Files menu, to clear the input buffer if the file is inappropriate, then get a different input file. If the file is appropriate,

3. Select the Merge files command on the Files menu to combine the input file with the output-file shell.

   The input file is moved into the output buffer; the input buffer is now cleared. The status line in the center of the screen notes that one file has been merged into the output buffer.

You can view the information in the output buffer at any time using the View output buffer command on the Editor menu. If you do this, you'll notice that the header, pin and node statements, and equations are derived from the input file. The simulation data was omitted. It's best to write simulation commands for the combined design in an auxiliary simulation file at a later time. To return to the merge screen, just press [Esc].

Recommendation: It's a good idea to save the output before getting the next input file.

4. Select the Save command on the Files menu to write the information in the output buffer to a file.

Subsequent Files

This is a repeat of earlier steps.

1. Get the next input file.

This file is placed in the input buffer and compared with existing data in the output buffer. Status tables change appropriately.

2. Review the pin summary table to determine if the design fits in the specified device.

   - If the design is not suitable, you can abandon the input file to clear the buffer and get a different input file.

- If the design is suitable, begin resolving conflicts, as discussed next.

2.3.3 RESOLVE CONFLICTS

Conflicts fall into two categories: detectable and undetectable. Detectable conflicts occur when signals in the input and output buffer have the same name or pin number. These can be resolved using the conflict-resolution form.

- If the intent is to use separate signals, you must rename one.
- If the intent is to use the same signal, you must bind them together.

The default action is to rename the signal in the input buffer.

When you specify no floating pins on input as a setup option **and** two pins are assigned to the same pin location on the device, the word Wildcard appears in the Action list. In this case, a question mark is automatically assigned to the pin location in the input buffer.[16] To restore the pin location specified in the input buffer, you must edit the pin/node list in the output buffer after combining the files.

Undetectable, resolvable conflicts occur when two signals with different names refer to the same signal. You can bind these signals together to resolve the conflict. Conflicts with other named elements, such as state, group, and string names do not occur because the input file was expanded and minimized so these potential conflicts are no longer present.

[16] Refer to Section II, Chapter 5 and Section IV, Chapter 10, for additional details about floating pin locations.

Review Detectable Conflicts Table

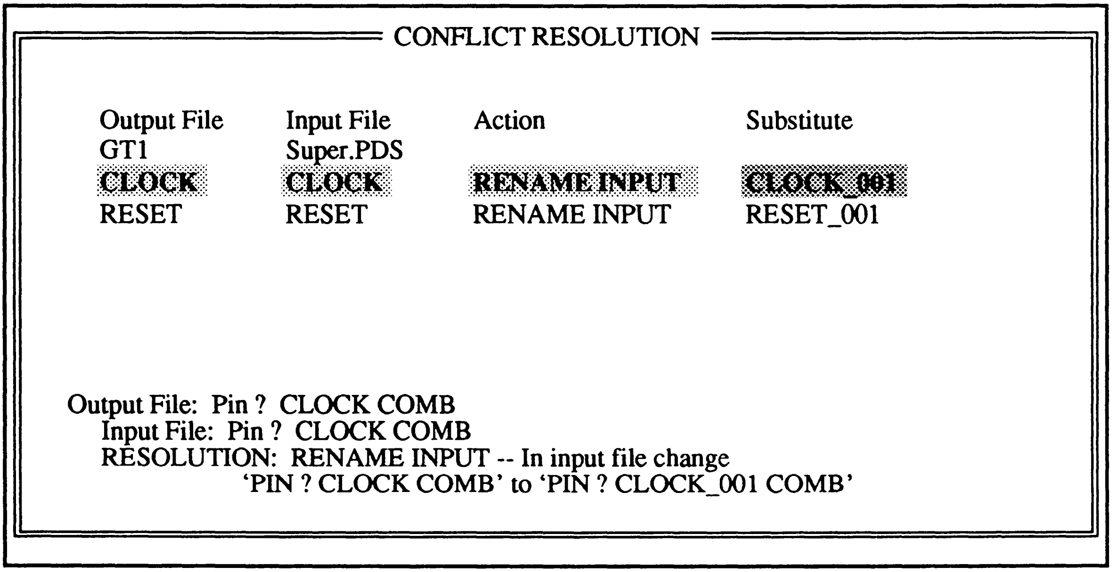
To determine which detectable conflicts exist,

1. Review the detectable-conflicts table, which identifies the number of pin/node conflicts detected when data in the input and output buffers were compared.

When no conflicts are reported, you can merge the files and save the data and then get the next input file. However, it is best to resolve conflicts in the input buffer, if they exist, before merging data.

2. Select the Resolve detectable conflicts command on the Resolution menu.

A conflict-resolution form appears listing all signals with detectable conflicts.



```
                    CONFLICT RESOLUTION

   Output File     Input File     Action           Substitute
   GT1             Super.PDS
   CLOCK           CLOCK          RENAME INPUT     CLOCK_001
   RESET           RESET          RENAME INPUT     RESET_001



Output File:  Pin ?  CLOCK COMB
   Input File:  Pin ?  CLOCK COMB
   RESOLUTION:  RENAME INPUT -- In input file change
                  'PIN ? CLOCK COMB' to 'PIN ? CLOCK_001 COMB'
```

This form includes two columns with option fields and two columns with status fields.

- Status fields: Output File and Input File

- Option fields: Action and Substitute

Each row identifies a single conflict and includes the information in Table 2, below.

Table 2: Conflict Resolution

FIELDS	DEFINITIONS
Output File	Signal name in the output file
Input File	Signal name in the input file
Action	The action to resolve the conflict: Rename input, Bind, or Wildcard
Substitute	The name that's assigned automatically to the signal in the input buffer

At this point, you take action as follows. Discussions below provide guidelines for each task. So skip to the discussion that suits your design.

- Rename signals in the input buffer.
- Bind signals together.
- Combine files when all conflicts are resolved.
- Edit the pin/node list in the output buffer.
- Save the output file.
- Edit the combined design to re-engineer it for the MACH device.

Important: The input file is not changed, only data in the input buffer is altered. Changes do not become part of the output file until you combine files and save the data in the output buffer to a file.

Rename Signals in the Input Buffer

This discussion covers two cases, so skip to the case that is required for your design.

It's best to handle Wildcard actions first. For example, if you have two designs with the following pin statements and you specify no floating pins on input as a merge-setup option, a pin number conflict is detected and Wildcard appears in the Action column of the conflict resolution form.

```
PIN     6     CLOCK       ;design 1
PIN     6     CLK         ;design 2
```

To recover from a Wildcard action, you complete one of two procedures.

1. Resolve other conflicts on the resolution form, press [F10] to accept the changes and return to the status screen, merge and save the design, then edit the pin/node list in the output buffer as described later.

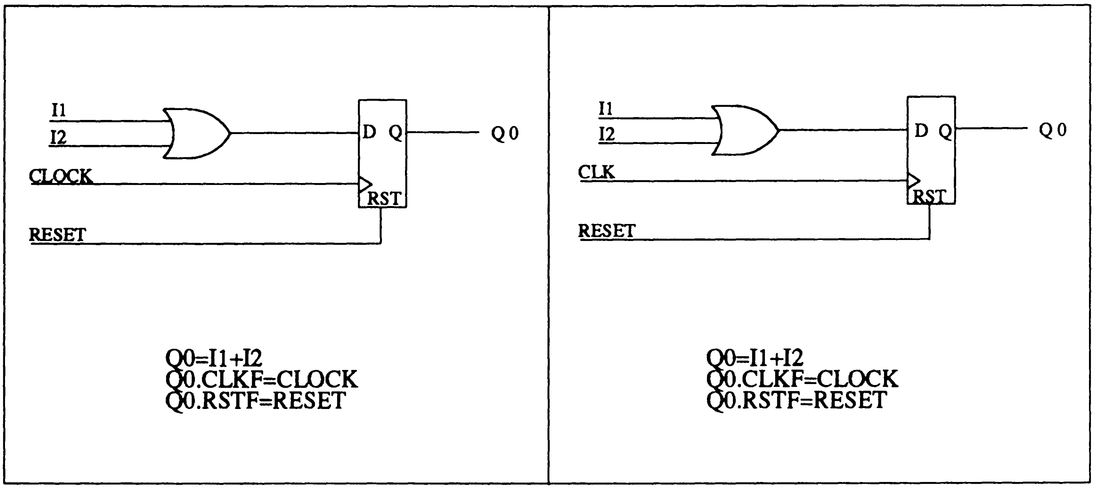
or

2. Leave the conflict resolution form, abandon the input file, specify floating pins on input as a setup option, and get the file again.

After resolving wildcard actions, you may find the same name is used in both the input and output design though it references different signals. When this conflict is detected, one of the signals must be renamed.

When a **signal** is **renamed**, a substitute name is provided automatically to replace every occurrence of the original in the input buffer. Default substitute names include the original signal name and an extension that's defined using the Set renaming strategy command on the merge-process Setup menu. You can either accept or change the substitute name in the conflict-resolution form.

For example, suppose you have two designs that each contain a single equation with registered output, as shown next.



A conflict is detected for each set of signals with the same name: I1, I2, Q0, and RESET. In this case, I1, I2, and Q0 represent different signals in each design. The desired resolution is to rename the signals in the input buffer as I3, I4, and Q1; the substitute name that's assigned automatically is not appropriate.

To change the substitute that's used to rename a signal,

1. Select the Substitute field: press the [Tab] key.

2. Type a new name to distinguish it from the signal in the output buffer and press [Enter].

   The substitute name must be unique; it cannot match an existing name without causing a new conflict.

Changes aren't reflected in the status data at the bottom of the screen until you leave the field and return to it. Each occurrence of the name is replaced in the input buffer.

When all signals are renamed appropriately, you can **either** bind signals as described in the next discussion

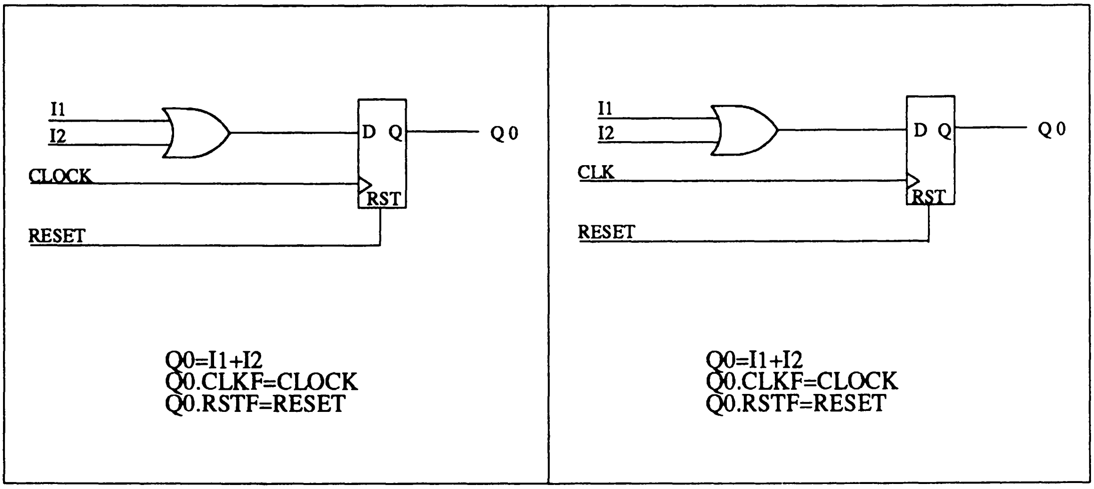
or complete step 3 to return to the detectable conflicts table.

3. Confirm the changes and return to the tables: press [F10].

Bind Signals Together

When one name is used in both the input and output designs to reference the same signal, a conflict is detected. In this case, the problem signals are listed on the conflict-resolution form. However, when two signals of different names refer to the same signal, no conflict is detected and you must bind these signals manually using the Bind command to display the appropriate form.

For example, suppose you have two designs each containing a single equation a with registered output, as shown next. You intend to use the same control signals to clock and reset both equations.



In this case, respective signals must be bound using a common name. Reset appears in both designs so it is listed on the conflict-resolution screen and a substitute name is provided automatically. The appropriate action is to bind rather than rename the signal in the input buffer.

However, the clock signals in each design are currently named differently. Since no conflict is detected, you use the Bind command on the Resolution menu to display the bind form where you can select from a list of signals in each design.

To bind signals with identical names, you use the conflict-resolution form, as follows.

1. Select the Resolve detectable conflicts command from the Resolution menu.

2. Activate the Action field on the form that corresponds with the appropriate signal and display the list of options: press [Tab] then press [F2].
3. Select Bind from the list and press [Enter].

   Changes aren't reflected in messages at the bottom of the screen until you leave the field and return to it.

When all appropriate signals are bound,

4. Save changes to the conflict-resolution form: press [F10].

   A message indicates the number of signals that will be moved to the bind form.

When you leave the conflict-resolution form, the bound signals are moved to the bind form.

To undo the bind operation after leaving the conflict-resolution form,

1. Select Bind pins/nodes from the Resolution menu: type the letter B.

2. Select the Action field for the appropriate signal and display the options: press [Tab] then [F2].

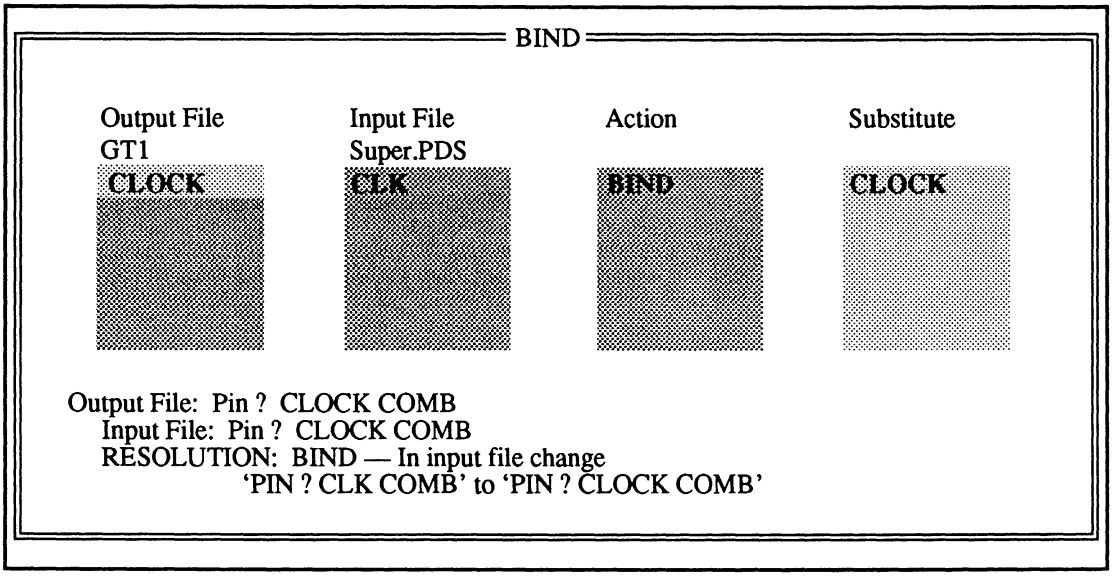
3. Select No action from the list: type the letter N.

4. Confirm changes and leave the form: press [F10].

To bind signals with different names, you must use the Bind form, as follows.

1. Select Bind pins/nodes from the Resolution menu: type the letter B.

   The bind form appears listing all signals that were bound from the conflict-resolution form, as shown next.



BIND

Output File	Input File	Action	Substitute
GT1	Super.PDS		
CLOCK	CLK	BIND	CLOCK

Output File: Pin ? CLOCK COMB
Input File: Pin ? CLOCK COMB
RESOLUTION: BIND — In input file change
'PIN ? CLK COMB' to 'PIN ? CLOCK COMB'

This form is similar to the conflict resolution form; however, the field types differ as follows.

- Option fields: Output File and Input File
- Status fields: Action and Substitute

To bind signals here,

2. Select an empty Output File field and display the options: press [Tab] then [F2].

   A list of all signals in the design is displayed.

3. Choose the signal in the output file, CLK, for example: press the down-arrow key, then [Enter].

4. Activate the Input File field, display the options, and select the name, for example, CLOCK.

   The two signals are bound together and the name from the output buffer is used. Each occurrence of the name is replaced in the input buffer.

5. Confirm changes on the bind form, then check the conflict form for new conflicts.

After all conflicts are resolved, you can merge the files as discussed next.

2.3.4 MERGE FILES

You merge data in the input buffer with the data in the output buffer after all conflicts have been resolved. Then you can edit the pin/node list if needed.

1. Review the detectable-conflicts table to verify that all have been resolved.

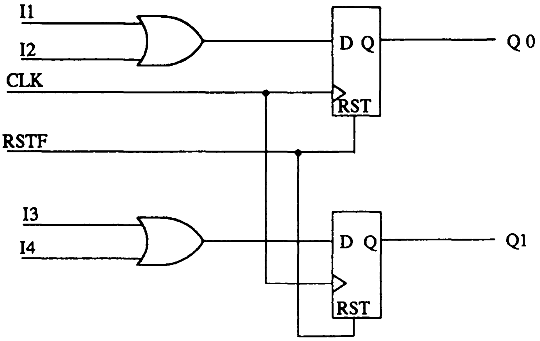
> **Important**: Once begun, the merge process cannot be stopped. To reverse this operation, you must quit and restart as follows.
>
> - Select Quit from the Files menu; do not save the current output file.
>
>   You lose all changes to the input buffer.
>
> - Select the Merge design files command from the File menu to initiate the operation again, then specify a new output file name.
>
> - Get and combine the previous session's output file into the new output-file shell and continue.

2. Select the Merge files command from the Files menu, then save the output.

The resolved, merged design that was discussed earlier is shown next in both text equations and schematic form.

```
Q0=I1+I2
Q0.CLKF=CLOCK
Q0.RSTF=RESET
```

```
Q0=I3+I4
Q0.CLKF=CLOCK
Q0.RSTF=RESET
```



Edit Combined Data

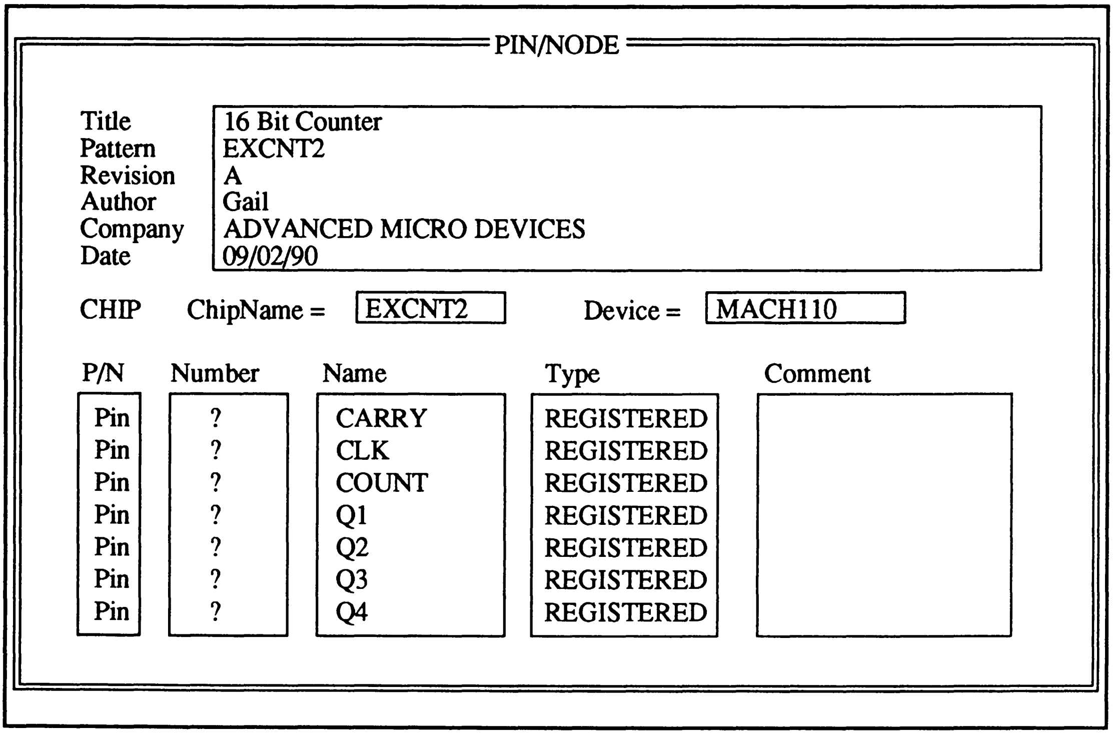
After merging files, you can edit the combined data to change header information, the device type, or change the name or location of a signal.

To edit header information,

1. Select the Edit pin/node list command on the Resolution menu.

   Data in the output buffer becomes available and includes the header and pin/node list from all designs combined thus far.

Data appears in a form, like the declaration-segment form for new PDS files. The top of the form includes the usual header fields; the information here is the same as from the first input file, shown below.



PIN/NODE

Title	16 Bit Counter
Pattern	EXCNT2
Revision	A
Author	Gail
Company	ADVANCED MICRO DEVICES
Date	09/02/90

CHIP ChipName = EXCNT2 Device = MACH110

P/N	Number	Name	Type	Comment
Pin	?	CARRY	REGISTERED	
Pin	?	CLK	REGISTERED	
Pin	?	COUNT	REGISTERED	
Pin	?	Q1	REGISTERED	
Pin	?	Q2	REGISTERED	
Pin	?	Q3	REGISTERED	
Pin	?	Q4	REGISTERED	

2. Activate the appropriate field and edit text as usual.

To change the device type,

1. Activate the Device field and display a list of options.

2. Select the MACH device you want to use.

The pin node list at the bottom of the screen can be scrolled and includes all combined data in the output buffer.

To change a pin name or number,

1. Select the appropriate pin or node statement.

2. Enter information using appropriate syntax for each field as you would in the declaration segment form.

3. Save data in the buffer when all editing is complete: press [F10].

You can re-enter the buffer to create additional pin/node fields. At least twenty empty fields become available at the end of the pin/node list each time you enter this buffer.

Save Combined Data

When the files are merged,

1. Save the output to disk.

2. Get the next input file and repeat the entire process until all files are combined.

3. Quit when you're finished.

4. Edit the combined design using a text editor to re-engineer it for a MACH device, as discussed next.

You can now re-engineer the combined design.

2.3.5 RECOMPILE THE COMBINED DESIGN

After you combine all files for a single MACH-device design, you need to recompile the resulting PDS file. The new .PDS file may be processed as any other .PDS file, using the full array of optimization options.

CHAPTER 3

SCHEMATIC-BASED MACH-DESIGN DEMONSTRATION

CONTENTS

3 SCHEMATIC-BASED MACH-DESIGN DEMONSTRATION

Steps in this guide demonstrate the schematic-based design flow using a sample design provided by AMD. As you complete activities in this guide at the workstation, you'll explore features of the MACH 110 device and the PALASM 4 software.

- You'll view and compile and fit a design that implements the heart of a DMA address generator for a single MACH 110 device.

  In the process, OrCAD/SDT III schematic files are automatically converted into a single PDS file.

- You'll view reports produced during the compilation and fitting process.
- You'll simulate the design and view the results.

Activities here assume you are familiar with the following; therefore, information from the sources below is not repeated.

- OrCAD/SDT III software and manuals
- MACH Devices Data Book

3.1 DESIGN EXAMPLE

This design implements the fundamental features of a DMA address generator, which is used to generate addresses for data that's transferred sequentially to or from a memory. The number of transfers is controlled by a word counter.

The DMA address generator consists of two major portions: an address counter and a word counter. The address counter is implemented with two cascaded 4-bit up counters and the word counter is implemented with two cascaded 4-bit down counters.

The DMA address generator setup involves the following steps.

- Load the address counter with the starting address of the memory transfer.
- Load the word counter with the number of memory transfers.
- Assert the enable signal.

Once the enable signal is activated, the following events occur at each rising edge of the clock signal.

A. The word counter is decremented by one.

B. The address counter is incremented by one.

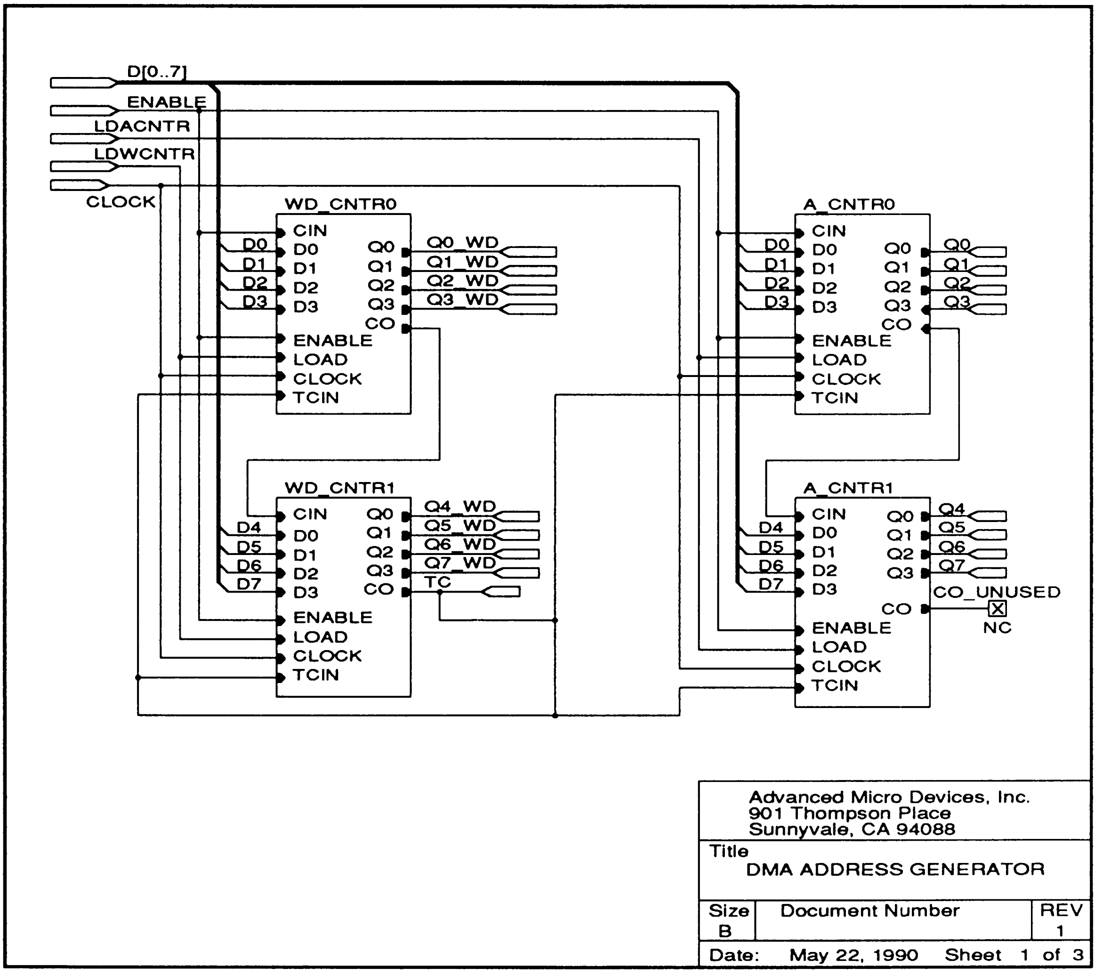
When the word counter reaches a count of zero, the terminal-count signal, TC, is asserted, which inhibits further counting.

The top-level logic diagram is shown next. It's composed of four hierarchical blocks connected by a common bus. Each block is linked by module ports to a lower-level schematic that defines one of the two counters.

- The two blocks on the left are connected to word-counter logic.

- The two blocks on the right are connected to address-counter logic.

To allow cascading, each counter has count-in and count-out ports and uniquely numbered output signals.



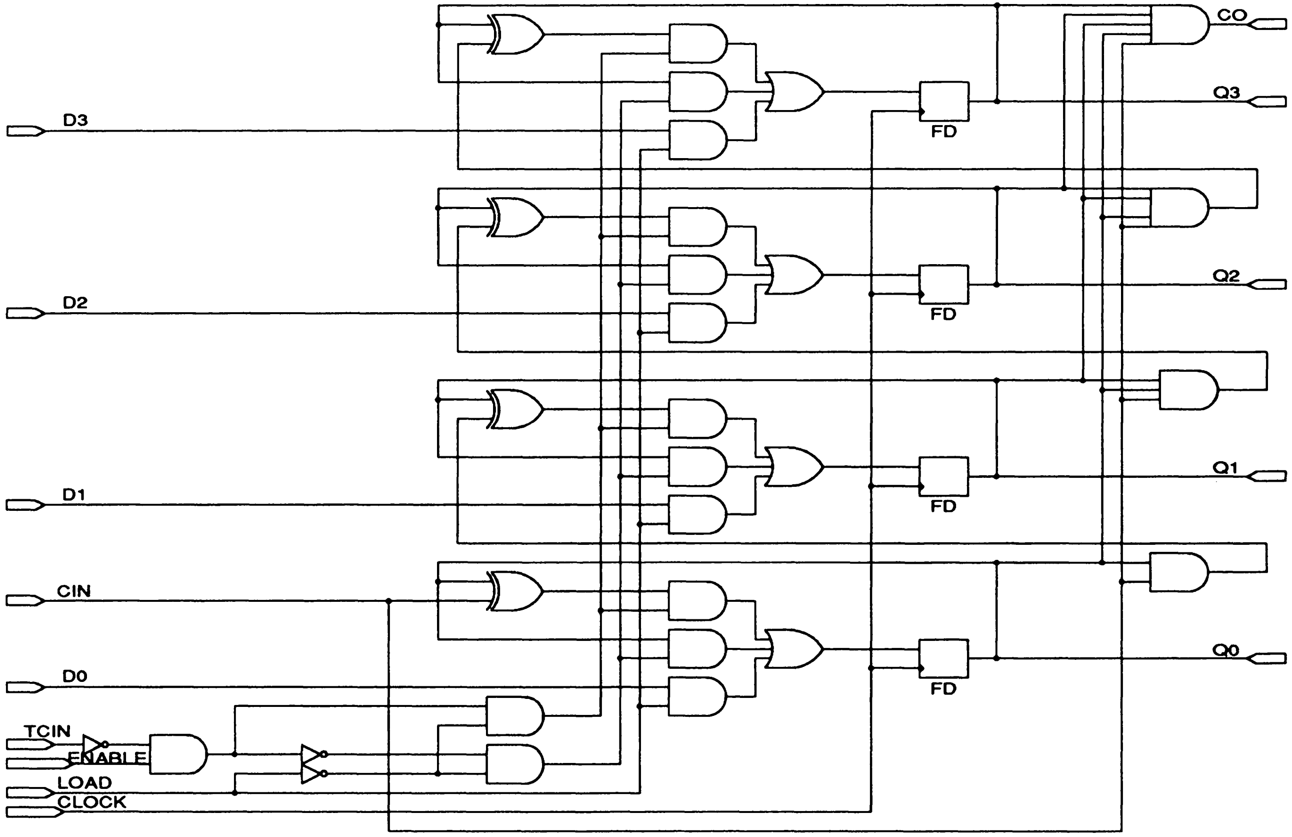
3.1.1 ADDRESS COUNTER

The address counter, an 8-bit **up** counter, provides the current memory address. This design is composed of two identical 4-bit up counters, each of which has carry-in, CIN, and carry-out, CO, ports to allow cascading.

The counter is loaded by asserting the LDACNTR signal for one clock period while presenting the starting address at the data inputs.

The counter increments on each clock edge when CIN and ENABLE are both high and TC is low.

Note: The address counters A_CNTR0 and A_CNTR1 are two instances of the same macro. Therefore, only one underlying schematic sheet, labeled sheet 2 of 3, is needed.



Advanced Micro Devices, Inc.
901 Thompson Place
Sunnyvale, CA 94088

Title
ADDRESS COUNTER

Size	Document Number	REV
B		1

Date: May 22, 1990 Sheet 2 of 3

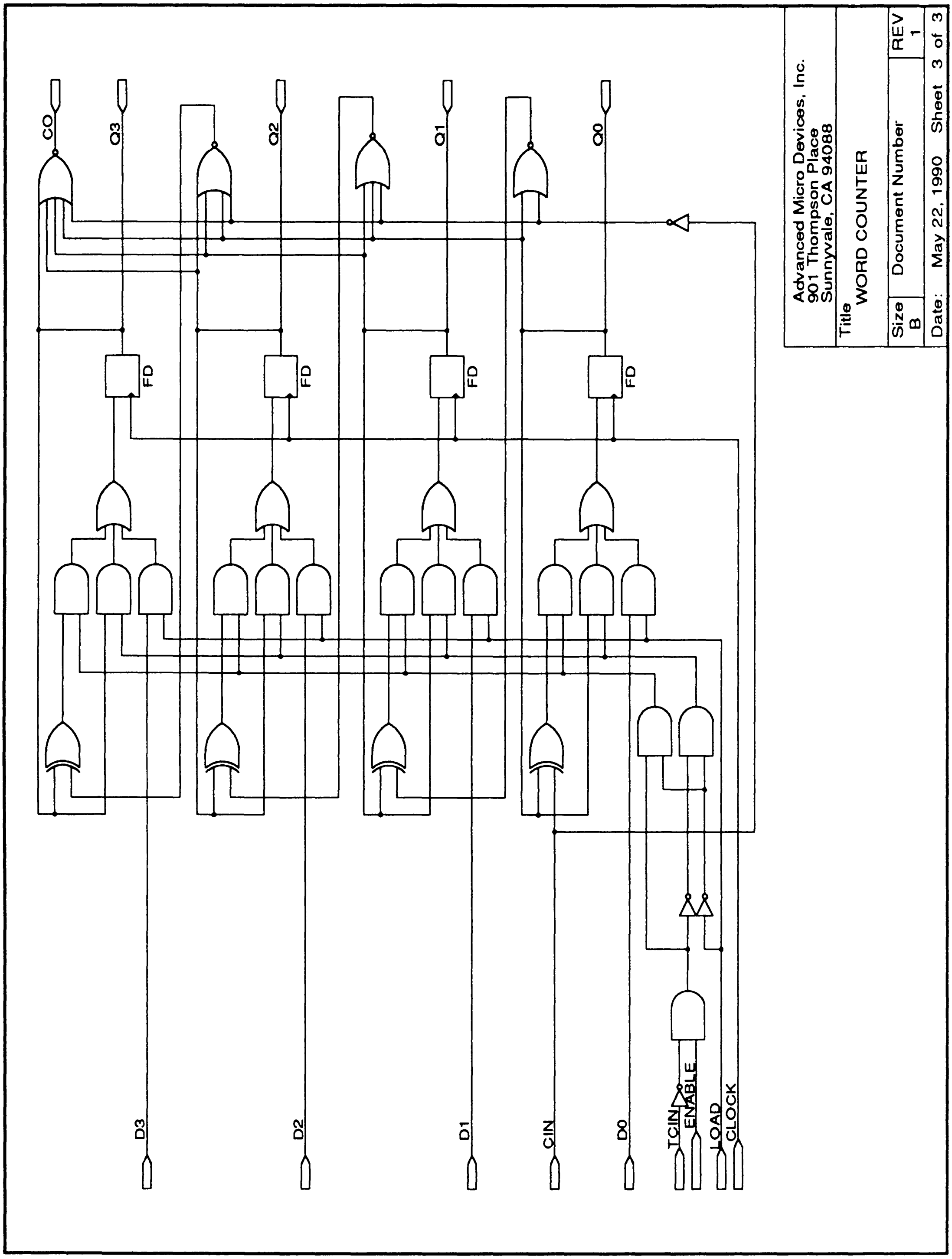
3.1.2 WORD COUNTER

The word counter, an 8-bit **down** counter, provides the count of the remaining memory transfers. This design is composed of two identical 4-bit down counters each of which has CIN and CO ports to allow cascading.

The counter is loaded by asserting the LDWCNTR signal for one clock period while presenting the initial word count at the data inputs.

The counter decrements on each clock edge when CIN and ENABLE are both high and TC is low.

Note: The word counters WD_CNTR0 and WD_CNTR1 are two instances of the same macro. Therefore, only one underlying schematic sheet, labeled sheet 3 of 3, is needed.


CO
Q3
Q2
Q1
Q0
FD
FD
FD
FD
D3
D2
D1
CIN
D0
TCIN
ENABLE
LOAD
CLOCK
Advanced Micro Devices, Inc.
901 Thompson Place
Sunnyvale, CA 94088
Title
WORD COUNTER
Size
B
Document Number
REV
1
Date: May 22, 1990
Sheet 3 of 3

3.2 CONFIRM THE SETUP

The steps that guide you through various activities in the schematic-based design cycle are presented next.

Important: The **left column** of this guide shows prompts and **identifies** which key you press, what you type, or what you must select on the screen to respond correctly. Sometimes, a prompt appears in the left column before the required response to help you track the process.

The **right column** of this guide provides information and numbered steps that **explain** what to do or select in more detail and describe what happens on the screen.

Tip: **For a quick tour**, you can type the text, select the boxed item, or press the key named in the left column; refer to the right column for details or clarification.

To begin from the operating system,

C:\>

PALASM [Enter]

1. Type PALASM and press [Enter] to execute the software.

   The AMD logo and copyright notice appear. A message at the bottom of the screen prompts you.

Press any key ...

[Spacebar]

2. Dismiss the copyright notice and continue: press the [Space bar] or any other key.

   Menu names appear across the top of the screen. The File menu is open, the first item is highlighted.

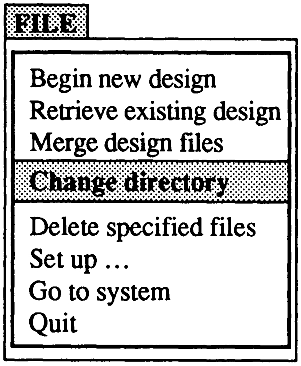
The sequence of tasks is listed below.

- Retrieve the design
- Verify the setup

3.2.1 RETRIEVE THE DESIGN

Before you can perform any design task, you must identify the name of the design you'll work on. To ensure that you'll be able to retrieve the AMD-supplied design example, first change the working directory.

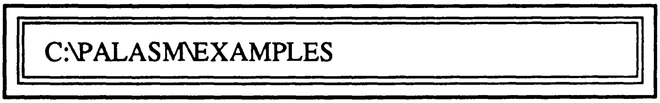
To change the working directory,



1. Select the Change directory command from the File menu: type the first letter of the command, which appears in the menu as a capital letter, when the menu is open.

   Alternatively, you can press the down arrow to highlight the command, then press [Enter] to select it.

   A form like the one below appears showing the current working directory path. The path on your screen may differ.



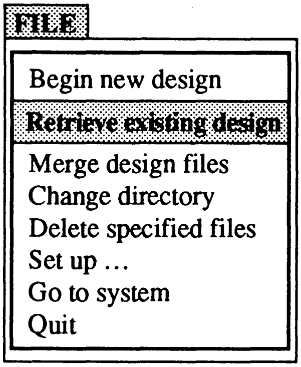
C:\PALASM\EXAMPLES

If the directory path on your screen does not match the one shown above, complete step 2. Otherwise, just press [F10] to accept the directory path shown.

C:\PALASM\EXAMPLES
[F10]

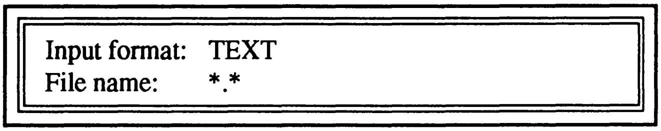
2. Type the path name shown beside this paragraph and press [F10] to confirm it.

To retrieve the design,



1. Select Retrieve existing design from the File menu: type the letter R.

   A form appears, as shown below. The default format, TEXT, is specified; space is provided for the file name. The highlighted field is active.



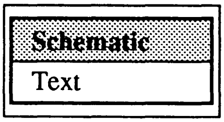
Input format: TEXT
File name: *.*

If the input format on your screen is TEXT, complete steps 2 and 3. Otherwise, skip to step 4.

Input format: TEXT
[F2]

2. Display the format options: press the [F2] key.

   The first option is highlighted when the list appears.



3. Select Schematic: type the letter S.

   You're returned to the form and Schematic is highlighted.

↓

4. Activate the next field: press the down-arrow key.

   The field is highlighted to show that it is active. *.* indicates you can display a list of all file names to choose from, or type a name.

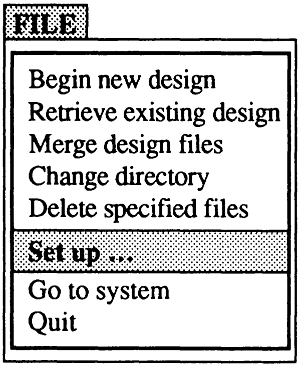
File name:
ORCADDMA.SCH [F10]

5. Type the name shown at left, ORCADDMA.SCH, then press [F10] to confirm your specifications.

   The form is dismissed. Design information may appear in the lower-right corner of the screen.

3.2.2 VERIFY SETUP

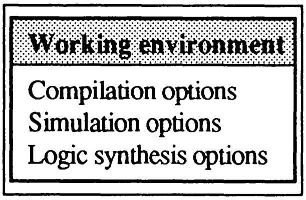
You're ready to verify that the defined setup is appropriate for this design. There are four kinds of setup options you can access through the File menu.



1. Select Set up from the File menu: type the letter S.

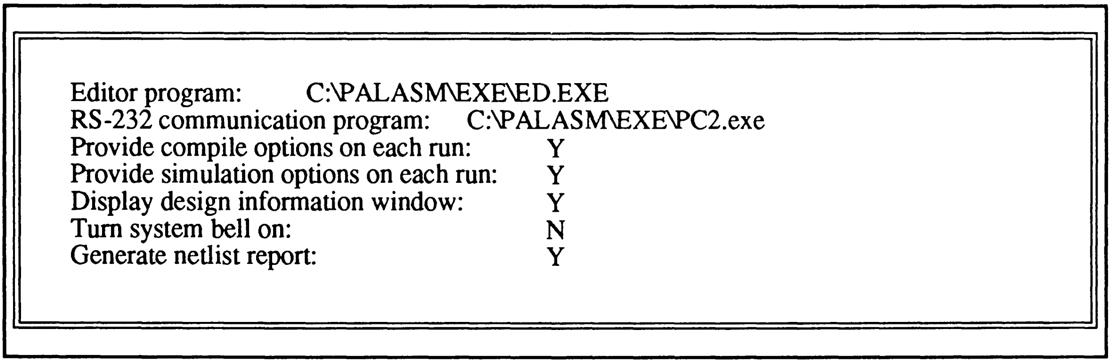
   A submenu opens offering four options: Working environment, Compilation options, Simulation options, and Logic synthesis options.

Any command that includes an ellipses, ..., displays a submenu of associated commands when selected.



2. Select Working environment from the submenu: type the letter W.

   A form appears that identifies specifications like those shown below.



```
Editor program:          C:\PALASM\EXE\ED.EXE
RS-232 communication program:    C:\PALASM\EXE\PC2.exe
Provide compile options on each run:       Y
Provide simulation options on each run:    Y
Display design information window:         Y
Turn system bell on:                       N
Generate netlist report:                   Y
```

The first field identifies the path name to the text editor you use to edit a PDS file. The second field identifies the path to the communications software used to download a design.

The third and fourth lines define whether or not a setup form appears when you compile or simulate the design using the corresponding command on the Run menu.

Important: **For this example**, the specification for both forms should be the letter Y so they appear when you process the design.

```
Provide compile options on each run          Y
Provide simulation options on each run       Y
```

Note: When working on your own design, you can display the corresponding setup form at any time using the steps below.

- Select the Set up command from the File menu.

- Select **either** Compilation options **or** Simulation options from the submenu.

The option below defines whether or not current design information appears in the lower-right corner of the screen.

```
...
Display design information window      Y
...
```

When the system bell is turned on, a tone sounds to warn you about errors. Also, a netlist report can be generated; however, it is not critical for this design.

If the options on your screen differ from those above, complete step 3. Otherwise, skip to the step 4.

Y

3. Activate the appropriate field, then type the letter Y: just press the down-arrow key, then type.

   After you type, the next field is automatically activated (highlighted).

When you finish,

[F10]

4. Confirm the specifications and dismiss the form: press the [F10] key.

   You're returned to the submenu; the last command selected remains highlighted.

[Esc]

5. Dismiss the submenu: press [Esc].

   You're returned to the File menu; the Set up command remains highlighted.

Next, you view the design file.

3.3 VIEW DESIGN FILES

The schematic-based design you use for activities in this chapter is already complete.

Note: When you create a schematic design using the Begin new design file command on the File menu, two files are created simultaneously.

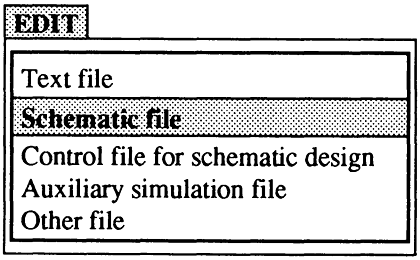
- An empty schematic file is produced using the specified design name.
- A control file is created using the specified design name with a .CTL extension, design CTL. A form appears so you can enter the declaration segment for the PDS file that is produced when schematic data is converted to text.

You use commands on the Edit menu to review both files for this design.

3.3.1 VIEW THE SCHEMATIC

In the following steps, you review the schematic design you identified using the Retrieve design file command.

Tip The following steps are provided to familiarize you with the AMD-supplied design example. If you already know OrCAD/SDT III commands and don't want to review the schematic on the screen, you can skip this discussion and review the control file for the schematic as discussed under 3.3.2.



1. Select Schematic file from the Edit menu: press the right-arrow key to open the menu, then select the command as usual.

   After a few seconds, you leave the PALASM environment temporarily and OrCAD/SDT III becomes available. The OrCAD/SDT III logo appears with a prompt at the top of the screen.

Press any key...
[Spacebar]

2. Dismiss the logo and continue: press [Space bar] or any other key.

   The copyright screen appears with a prompt at the top of the screen.

[Spacebar]

3. Dismiss the copyright screen and continue.

   The OrCAD editor becomes available and the top-level schematic appears; the entire sheet is reduced to fit in the upper-left corner of the screen.

OrCAD/SDT III lists available commands on menus and submenus. You access all commands through the main menu as follows.

[Enter]

4. Display the main menu: press [Enter].

   All primary commands are listed on the main menu.

There are several ways to select a command in OrCAD; this guide uses the first method listed below.

- Type the capital letter at the beginning of the command, as you do with PALASM 4 .
- Press arrow keys to highlight a command and press [Enter] to select it, as you can with PALASM 4.
- Use the mouse to highlight the command and click the left button to select it.

Grid parameters assist you in locating specific areas of the schematic.

To display grid references,

S

1. Select the Set command: type the letter S.

   A submenu of set commands appears.

G G

2. Select the Grid parameters command, then select the Grid references command: type the letter G twice.

Y

3. Confirm the display of grid references: type the letter Y.

   References appear across the top of the worksheet and along the left border.

To view the design in greater detail,

—>

1. Move the cursor to the area you want to display, for example, a block on the sheet or the title box in the lower-right corner.

Z

2. Select the Zoom command.

   A submenu appears offering several choices.

I

3. Select the In command.

   The screen shows the schematic in greater detail and the submenu is dismissed.

Unlike PALASM 4, where the menu must be open to select a command, the main menu need not be visible to select primary OrCAD/SDT commands. However, a submenu must be open to select secondary commands.

Z I

4. Select Zoom and In again so you can read the signal names on the worksheet.

You can use either the arrow keys or the mouse to drive the cursor and to move the design around the screen. In addition, you can use the Find command to move the cursor to a specific named component or block.

To find a hierarchical block,

F

1. Select the Find command.

   The command line changes to a prompt.

The Find command does not locate generic component names nor signal names. You must specify the assigned name or assigned reference designator.

Find?
WD_CNTR1 [Enter]

2. Type the name of the word-counter block in the lower-left corner, WD_CNTR1, and press [Enter].

   The cursor moves immediately to the upper-left corner of the block you identified, which now appears centered on the screen.

To view the sheet below the selected block,

Q

1. Select the Quit command.

   A submenu appears listing options.

E E

2. Select the Enter sheet command, then select the Enter command: type the letter E **twice**.

   The lower-level sheet is displayed. This sheet is also reduced.

You are now in a viewing mode. Currently available commands are listed across the top of the screen.

To zoom in and compare this schematic against the appropriate one at the beginning of this chapter,

—>
Z I
Z I
Z I

1. Move the cursor into the area you want to view and select the Zoom In command **three** times.

You can use the Jump command to immediately display a specific area of the design based on grid and screen coordinates.

J

2. Select the Jump command.

   A menu of Jump options appears.

Tag options are listed in the Jump menu but there are no tags in this design. Reference, X, and Y options refer to grid parameters and screen coordinates.

R

3. Select the Reference command.

   A menu displays letters that correspond to grid references along the left border of the worksheet.

B 4

4. Type the letter B and the number 4 to specify an area.

   The specified area is displayed in the center of the screen.

At this point, you could exit the viewing mode to edit the lower-level sheet. However, since no changes are needed for this example don't escape from the viewing mode.

To return to the top-level sheet and leave OrCAD/SDT III,

L [Esc]

1. Select the Leave command, then press [Esc] to end the sequence.

   You're returned to the top-level sheet, which is restored to the original reduced size.

Important: If you exited the viewing mode to edit the worksheet, you must select the Quit command before you can leave the lower-level sheet.

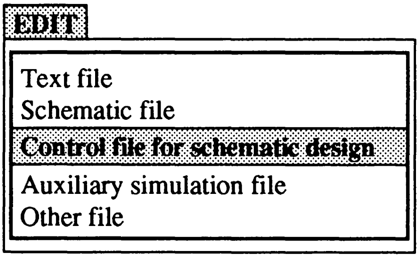
Also: **Do not** update the file; to do so may adversely affect the results. If you made changes, you're asked to abandon changes upon leaving. In this case, your response should be Yes.

Q A

2. Select the Quit command, then select the Abandon Edits command.

   You're returned to the PALASM environment.

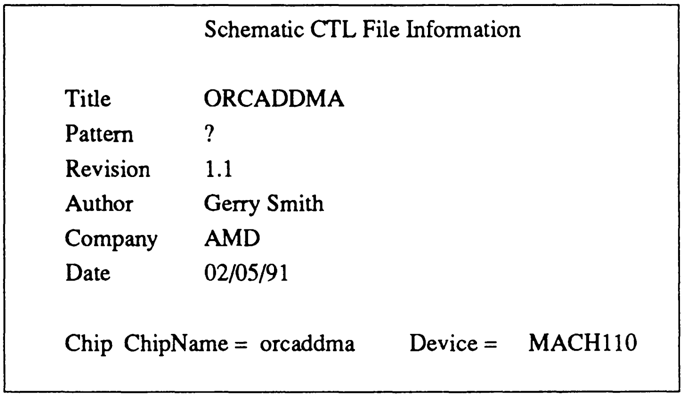
3.3.2 VIEW THE CONTROL FILE



Next, you' view the control file for this design, which will be used as the declaration segment of the PDS file that's produced later from converted schematic data.

1. Select the Control file for schematic design command from the Edit menu: press the right-arrow key to open the menu, then type the letter C.

   The form below appears containing all standard PDS-file header information. The form contains text fields and one option field. The Title field is active. The device type must be specified here, not in the schematic.



Schematic CTL File Information

Title	ORCADDMA
Pattern	?
Revision	1.1
Author	Gerry Smith
Company	AMD
Date	02/05/91

Chip ChipName = orcaddma Device = MACH110

This form was completed by the author before the schematic was started. The date is today's date. You can change the information in this file anytime.

Important: Don't change any information during this exercise.

[Esc]

2. Dismiss the form and cancel any changes you may have made by pressing [Esc].

   A confirmation form asks if you are sure.

Y

3. Type the letter Y to indicate you are sure.

   You're returned to the PALASM environment.

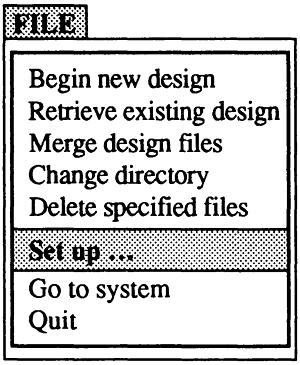
Next, you compile and fit the design.

3.4 COMPILE THE DESIGN

Before you compile a design it's a good idea to review logic synthesis, compilation, and MACH fitting options as follows.

1. Select Set up from the File menu: press the left-arrow key to open the menu, then type the letter S.

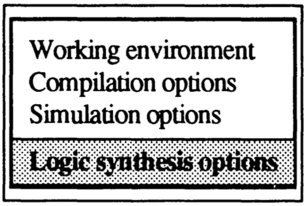
   A submenu appears with four options.

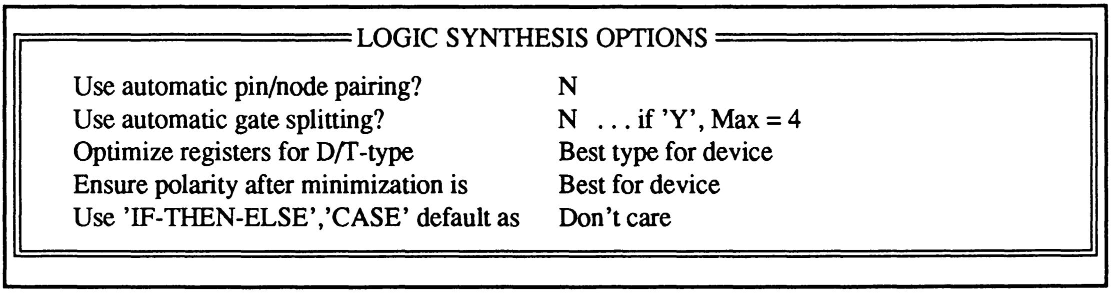


The compilation and simulation-options forms are set to appear automatically when you complete those operations using commands on the Run menu. For now, you review only the logic-synthesis options.

2. Select Logic synthesis options from the submenu.

   A form appears that identifies specifications for pairing, gate splitting, register optimization, polarity, and unspecified-default condition settings, as shown next.




LOGIC SYNTHESIS OPTIONS

Use automatic pin/node pairing?	N
Use automatic gate splitting?	N ... if 'Y', Max = 4
Optimize registers for D/T-type	Best type for device
Ensure polarity after minimization is	Best for device
Use 'IF-THEN-ELSE','CASE' default as	Don't care

Important: For this design, the following options must be specified.

Use automatic pin/node pairing?	N
Use automatic gate splitting?	N
Optimize registers for D/T-type	Best type for device
Ensure polarity after minimization is	Best for device
Use 'IF-THEN-ELSE', 'CASE' default as	Don't care

If the specifications on your screen differ, complete steps 3 or 4. Otherwise, skip to step 5.

field name ...
N

3. Activate the appropriate text field and type the letter N to change the specification in a text field.

field name...
[F2]
B

4. Activate the appropriate option field, press [F2] to display options, type the letter B to select either of the Best for device options.

When all specifications match those in the figure above,

[F10]

5. Accept all specifications: press [F10].

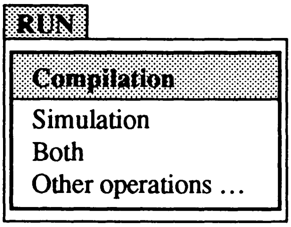
   The form is dismissed, any changes are recorded, and the Set up submenu is again available.

[Esc]

6. Dismiss the submenu: press [Esc].

   The File menu is again available.

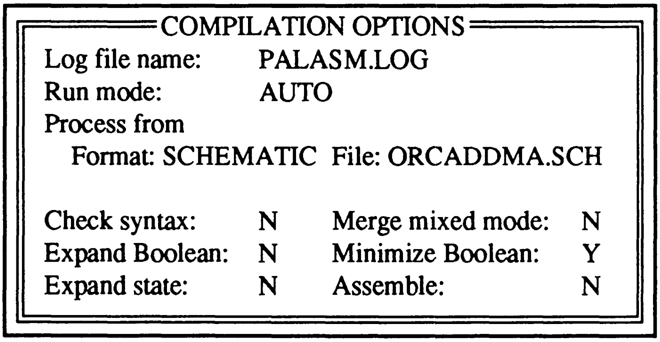
To begin compilation,



1. Select Compilation from the Run menu.

   The form below appears because of the following specification on the Working environment form.

   ...



```
COMPILATION OPTIONS
Log file name:     PALASM.LOG
Run mode:          AUTO
Process from
  Format: SCHEMATIC  File: ORCADDMA.SCH

Check syntax:      N    Merge mixed mode:   N
Expand Boolean:    N    Minimize Boolean:   Y
Expand state:      N    Assemble:           N
```

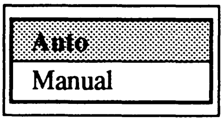
The first field is a text field that contains the default name for the execution log file: PALASM.LOG. The second field is an option field that designates the run mode; AUTO completes all appropriate processes based on the specified device type. AUTO is appropriate for this example.

The Format and File fields are status fields that reflect a schematic format and the name of this design, ORCADDMA.SCH. All other specifications apply only when you specify a manual run mode.

If the form on your screen specifies a run mode other than AUTO, complete steps 2 and 3. Otherwise, skip to step 4.

Run mode: MANUAL
[F2]

2. Activate the Run mode field and display options: press the down-arrow key, then press [F2].



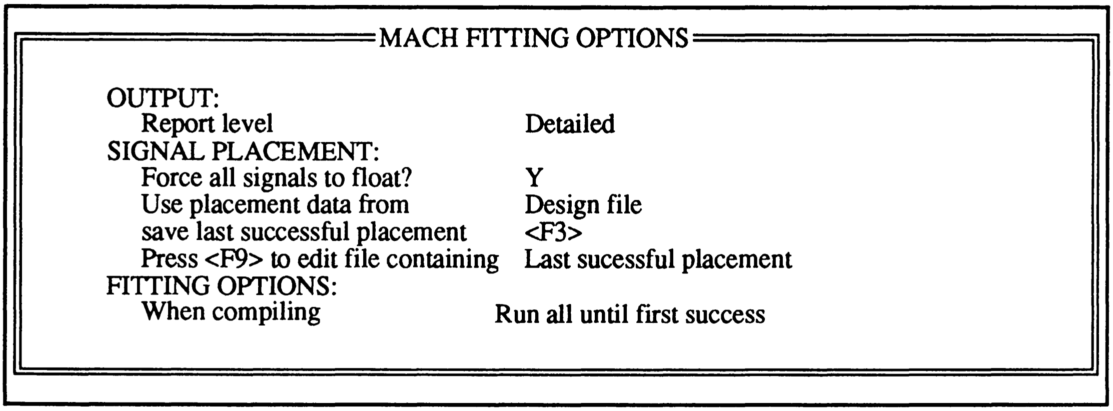
3. Select Auto from the list to change the setting.

   The list is dismissed and the specification in the form changes.

[F10]

4. Confirm all specifications: press [F10].

   The form below appears offering fitting options **only** when a MACH device is specified.



```
                    MACH FITTING OPTIONS

OUTPUT:
   Report level                          Detailed
SIGNAL PLACEMENT:
   Force all signals to float?           Y
   Use placement data from               Design file
   save last successful placement        <F3>
   Press <F9> to edit file containing    Last sucessful placement
FITTING OPTIONS:
   When compiling                     Run all until first success
```

Important: For this design, the following options must be specified.

```
OUTPUT:
  Report level                   Detailed
SIGNAL PLACEMENT:
  Force all signals to float?    Y
...
FITTING OPTIONS:
  When compiling                 Run  all until first success
```

If the specifications on your screen differ, complete steps 5 and 6. Otherwise, skip to step 7.

To float signals,

Force all signals to float
Y

5. Activate the appropriate field and type the letter Y to force all signals to float.

To change an option field,

field ...
[F2]
R

6. Activate the field, display the options, then select the appropriate one.

   The list is dismissed and the specification in the form changes.

When all specifications match those in the figure above,

[F10]

7. Confirm all specifications: press [F10].

   The form is dismissed and a window opens in the lower-half of the screen; the process begins.

Messages scroll by to keep you informed; errors and warnings are identified as they are discovered. The entire process takes approximately two minutes for this design and includes the following activities. Designs having more than 14 unique signals in an equation can cause runtime to increase exponentially. This design has no more than eight unique signals in a single equation.

A. Certain OrCAD utilities are run to check schematics for electrical design-rule violations; verified schematic data is converted into PDS format and a PDS file is created for further processing.

Note: You can manually convert schematic data to PDS format without additional processing, which can be useful when debugging a design.

- Select the Other operations command from the Run menu.

- Select Convert schematic to text from the submenu.

B. A syntax check is performed on the PDS file.

C. Equations are expanded, and minimized, then compilation is completed.

D. Fitting is completed and the device map and JEDEC files are produced.

Note: There should be no process errors; this design should fit the first time.

When the process finishes, you can scroll through messages on the screen or review certain reports using commands on the View menu. The following files deserve a look.

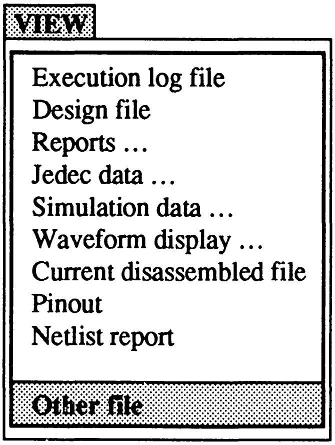
- The PDS file produced from schematic data
- The device pin-out report

To view the PDS file created from schematic data,

[Esc]

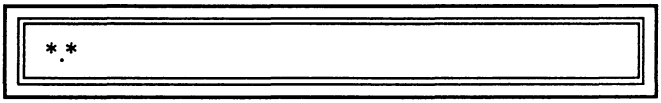
1. Dismiss the process window: press [Esc].

   The Run menu remains open with the last command highlighted.



2. Select Other file from the View menu.

   A form appears so you can enter the name.



.

The PDS file is named after the schematic design with a .PDS extension.

ORCADDMA.PDS [F10]

3. Type the file name, ORCADDMA.PDS, into the form and confirm with [F10].

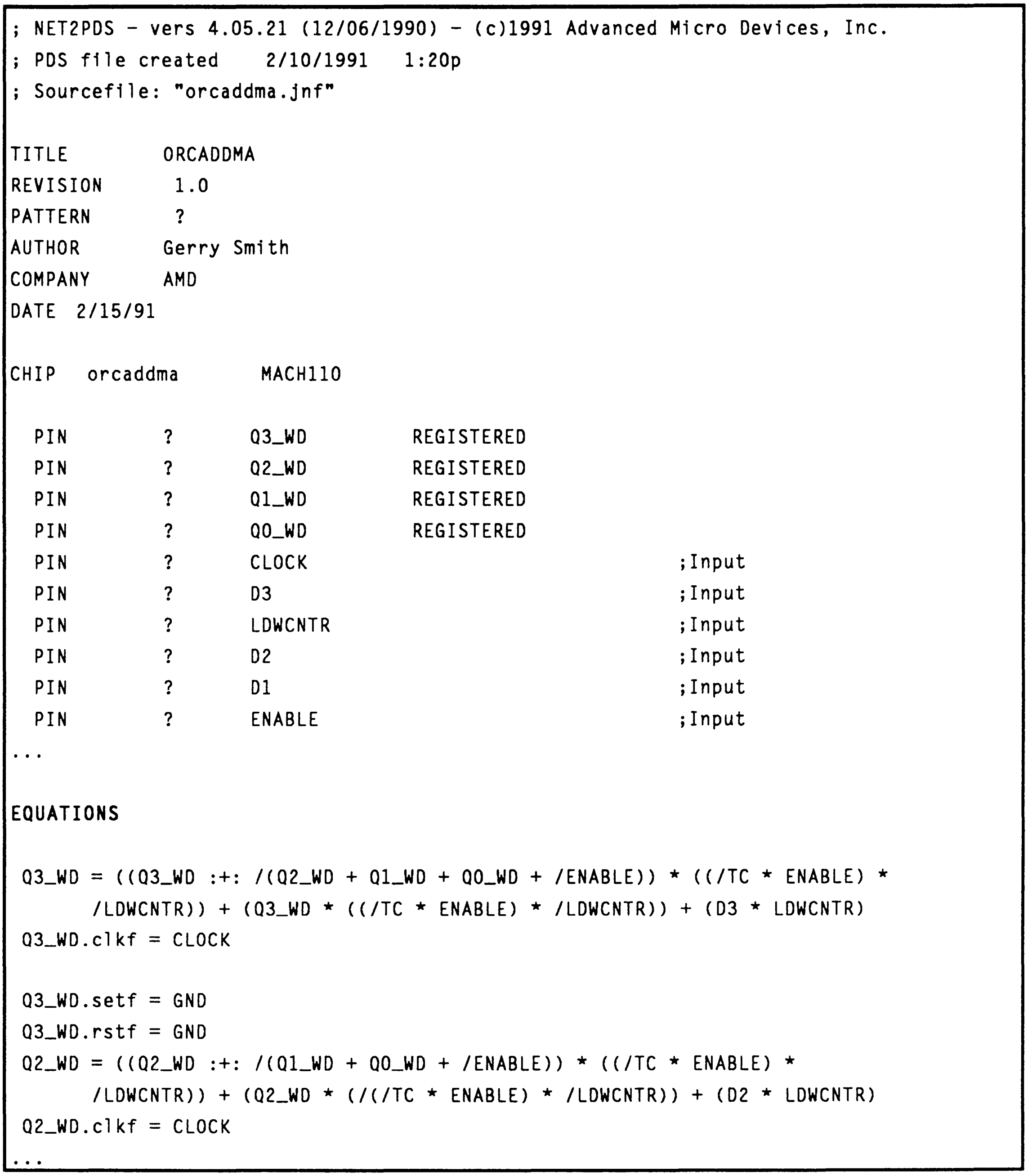
   A window opens so you can view the file. However, you cannot edit the file in this mode.

This PDS file represents the text version of converted schematic-based design data. Portions of the file are shown below, including the following.

- Declaration segment you viewed as a control file
- Partial equations segment

The question marks, ?, in pin statements indicate floating pin locations on the device.

Note: The simulation segment must be defined in a separate auxiliary file.



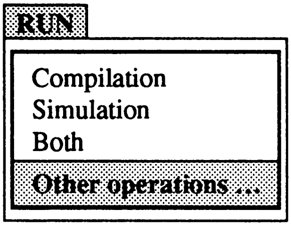
```
; NET2PDS - vers 4.05.21 (12/06/1990) - (c)1991 Advanced Micro Devices, Inc.
; PDS file created    2/10/1991   1:20p
; Sourcefile: "orcaddma.jnf"

TITLE        ORCADDMA
REVISION      1.0
PATTERN       ?
AUTHOR       Gerry Smith
COMPANY      AMD
DATE  2/15/91

CHIP   orcaddma       MACH110

  PIN        ?       Q3_WD        REGISTERED
  PIN        ?       Q2_WD        REGISTERED
  PIN        ?       Q1_WD        REGISTERED
  PIN        ?       Q0_WD        REGISTERED
  PIN        ?       CLOCK                              ;Input
  PIN        ?       D3                                 ;Input
  PIN        ?       LDWCNTR                            ;Input
  PIN        ?       D2                                 ;Input
  PIN        ?       D1                                 ;Input
  PIN        ?       ENABLE                             ;Input
...

EQUATIONS

 Q3_WD = ((Q3_WD :+: /(Q2_WD + Q1_WD + Q0_WD + /ENABLE)) * ((/TC * ENABLE) *
       /LDWCNTR)) + (Q3_WD * ((/TC * ENABLE) * /LDWCNTR)) + (D3 * LDWCNTR)
 Q3_WD.clkf = CLOCK

 Q3_WD.setf = GND
 Q3_WD.rstf = GND
 Q2_WD = ((Q2_WD :+: /(Q1_WD + Q0_WD + /ENABLE)) * ((/TC * ENABLE) *
       /LDWCNTR)) + (Q2_WD * (/(/TC * ENABLE) * /LDWCNTR)) + (D2 * LDWCNTR)
 Q2_WD.clkf = CLOCK
...
```

[Esc]

4. Dismiss the view window when you are finished with the PDS file: press [Esc].

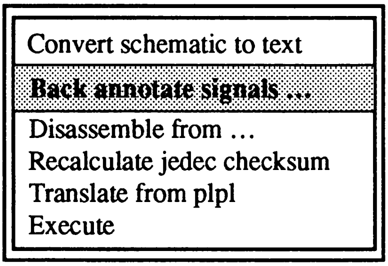
You can back annotate the PDS file to include the signal placements assigned automatically during the fitting process.

To back annotate the PDS file,



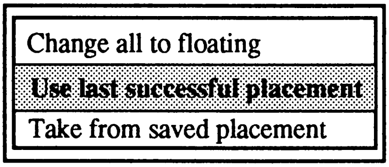
1. Select Other operations from the Run menu.

   A submenu opens offering six options, as shown next.


Convert schematic to text
Back annotate signals ...
Disassemble from ...
Recalculate jedec checksum
Translate from plpl
Execute

2. Select Back annotate signals from the submenu.

   A list of three options appears.



3. Select Use last successful placement from the list.

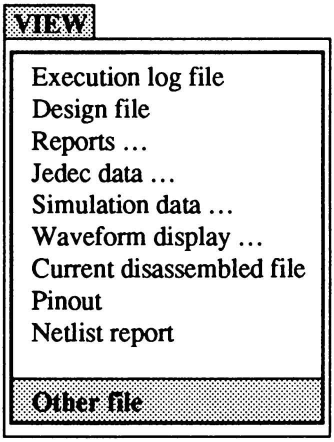
   A window opens and messages keep you informed about the process. Errors and warnings are identified as they are discovered. There should be no errors in this file.

[Esc] [Esc] [Esc]

4. Dismiss the window, list, and submenu: press [Esc] three times.

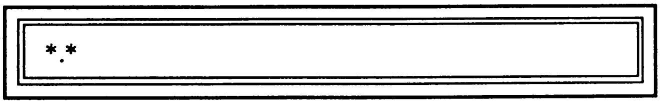
   You are returned to the Run menu.

To view the back-annotated PDS file,



1. Select Other file from the View menu.

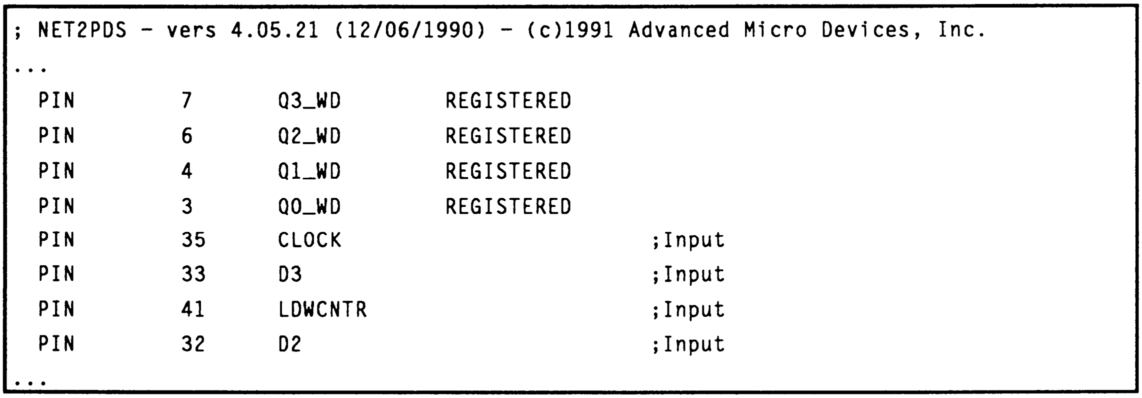
   Again, the form appears so you can enter a name.



ORCADDMA.PDS [F10]

2. Type the file name, ORCADDMA.PDS, into the form and confirm.

Pin statements in the back-annotated PDS file are shown in part below. The question marks, which previously designated floating pin locations, have been replaced with actual locations assigned during fitting.

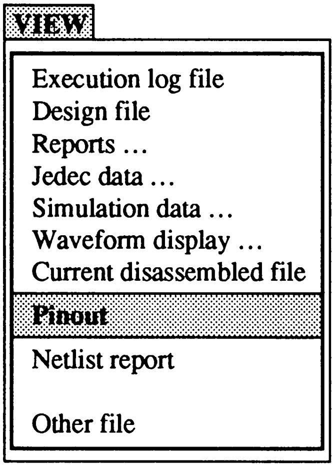


```
; NET2PDS - vers 4.05.21 (12/06/1990) - (c)1991 Advanced Micro Devices, Inc.
...
  PIN       7     Q3_WD       REGISTERED
  PIN       6     Q2_WD       REGISTERED
  PIN       4     Q1_WD       REGISTERED
  PIN       3     Q0_WD       REGISTERED
  PIN       35    CLOCK                         ;Input
  PIN       33    D3                            ;Input
  PIN       41    LDWCNTR                       ;Input
  PIN       32    D2                            ;Input
...
```

[Esc]

3. Dismiss the viewing window when you finish.

Next, you'll view the device pin-out report.



1. Select Pinout from the View menu.

   The process takes a moment to initiate. A window opens and the file is displayed.

The pin-out report is produced during the fitting process. It provides device pin-out data in graphic form and includes pin and node assignments for the device. Prior to back annotating signals, this report would not show a signal name associated with a pin. However, after back annotating signals, the appropriate name appears beside each pin on the device.

The cursor appears as a bar across the first line of the file. You can use the arrow keys to move up and down or right and left.

->

2. Move the cursor down and to the right to display information currently hidden offscreen.

[Esc]

3. Dismiss the window when you're finished.

You can simulate the design next.

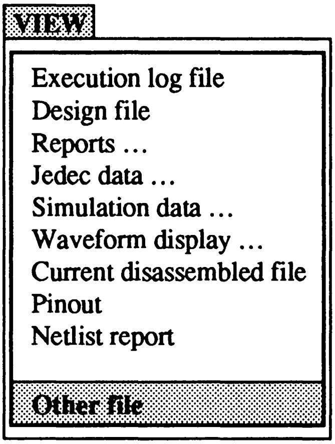
```
TITLE:      ORCADDMA
REVISION:   1.1
PATTERN:    ?
AUTHOR:     GERRY SMITH
COMPANY:    AMD
DATE:       8/31/90
MACRO:      ORCADDMA

                  GND   «----------------------+
              COM TC    «-------------------+  |  +-------------------»  VCC
              REG Q0_WD «----------------+  |  |  |  +----------------»  Q7
              REG Q1_WD «-------------+  |  |  |  |  |  +-------------»  LDACNTR
                  NC    «----------+  |  |  |  |  |  |  |  +----------»  LDWCNTR
              REG Q2_WD «-------+  |  |  |  |  |  |  |  |  |  +-------»  D7
                                +  +  +  +  +  +  +  +  +  +  +
                             +-----------------------------------+
                             |  6  5  4  3  2  1 44 43 42 41 40  |
              REG Q3_WD   +-|  7                            39  |-+     D6
              REG Q4_WD   +-|  8                            38  |-+     D5
              REG Q5_WD   +-|  9                            37  |-+     D4
                  ENABLE  +-|  10                           36  |-+     NC
                  D0      +-|  11                           35  |-+     CLOCK
                  GND     +-|  12         MACH 110          34  |-+     GND
                  D1      +-|  13                           33  |-+     D3
              REG Q6_WD   +-|  14                           32  |-+     D2
                  NC      +-|  15                           31  |-+     Q6
                  NC      +-|  16                           30  |-+     Q5
                  NC      +-|  17                           29  |-+     Q4
                             | 18 19 20 21 22 23 24 25 26 27 28  |
                             +-----------------------------------+
                                +  +  +  +  +  +  +  +  +  +  +
                  NC    «-------+  |  |  |  |  |  |  |  |  |  +-------»  Q3
                  NC    «----------+  |  |  |  |  |  |  |  +----------»  NC
                  NC    «-------------+  |  |  |  |  |  +-------------»  Q2
              REG Q7_WD «----------------+  |  |  |  +----------------»  Q1
                  VCC   «-------------------+  |  +-------------------»  Q0
                  GND   «----------------------+
```

3.5 SIMULATE THE DESIGN

Simulation can occur before or after compilation, or you can use the Both command to run compilation and simulation sequentially.

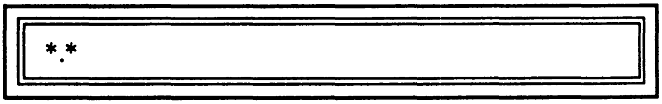
Before you simulate this design, you can examine the simulation file. You can use **either** the Auxiliary simulation file command on the Edit menu to display the simulation file in the text editor **or** you can view the file as you have viewed other files.

To view the simulation file,



1. Select Other file from the View menu.

   Again, the form opens so you can enter the name.



.

ORCADDMA.SIM [F10]

2. Type the name of the simulation file, ORCADDMA.SIM, and press {F10]..

   The file appears, as shown next.

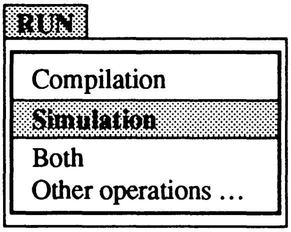
```
SIMULATION


TRACE_ON CLOCK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 TC LDACNTR ENABLE
SETF /CLOCK /D0 /D1 /D2 /D3 /D4 /D5 /D6 /D7 /LDACNTR /ENABLE
SETF /LDWCNTR
CLOCKF CLOCK
CHECK /Q0 /Q1 /Q2 /Q3 /Q4 /Q5 /Q6 /Q7 /TC
SETF D1 D3 D5 D7 LDWCNTR LDACNTR
CLOCKF CLOCK
CHECK /Q0 Q1 /Q2 Q3 /Q4 Q5 /Q6 Q7
SETF D0 /D1 D2 /D3 D4 /D5 D6 /D7 /ENABLE
CLOCKF CLOCK
CHECK Q0 /Q1 Q2 /Q3 Q4 /Q5 Q6 /Q7
SETF D1 D3 D5 D7
CLOCKF CLOCK
CHECK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7
SETF ENABLE /LDACNTR /LDWCNTR
FOR I:=0 TO 127 DO
      BEGIN
          CLOCKF CLOCK
      END
CHECK Q0 Q1 Q2 Q3 Q4 Q5 Q6 /Q7
TRACE_OFF
```

Notice the CLOCKF and SETF commands in the file above. You'll see the results of these and the TRACE commands in the simulation report. The CHECK statements verify the validity of the signals. If a discrepancy occurs, a question mark appears in the simulation output file. Again, you cannot edit the file in view mode.

[Esc]

3. Dismiss the window when you're finished viewing the file.

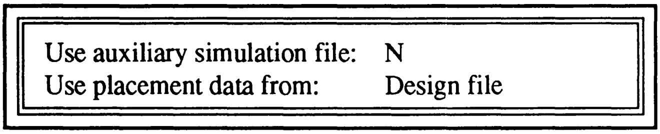
To simulate this design,



1. Select Simulation from the Run menu.

   The form below appears because of the specification on the Working environment form.

   Provide simulation options on each run Y



Use auxiliary simulation file: N
Use placement data from: Design file

No is the default specification for the Use auxiliary simulation file option; however, Y is appropriate for this example.

Y

2. Type the letter Y to indicate simulation information is in a separate file.

The Use placement data from option field allows you to specify the source of signal placement data needed to generate test vectors during simulation. Design file is the default. It is appropriate for this example, since you performed back annotation of signal placement data to the design file.

[F10]

3. Confirm the specification and begin the simulation.: press [F10]

   A window opens and the process begins. Messages scroll by to keep you informed; the process for this example takes about six minutes.

Note: Errors and warnings are identified as they are discovered. There should be no errors in this design. However, numerous warnings are generated that do not affect the accuracy or completeness of the design.

- Warning M2005: No Logic Equations found is generated whenever you use an auxiliary simulation file because it contains only simulation commands, not logic equations.

- Most warnings are repetitions of D2130, which is generated because the flip-flops are not explicitly set or reset. You can explicitly set or reset flip-flops using the AINIT macro with three-terminal flip-flops. However, that is not the case with this design.

When the simulation finishes, you can scroll through messages on the screen or view the reports.

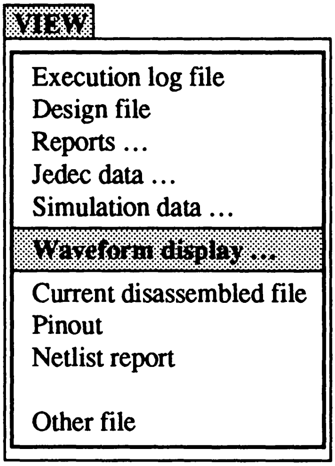
[Esc]

4. Dismiss the simulation-process window and return to the menu.

   The Run menu remains open.

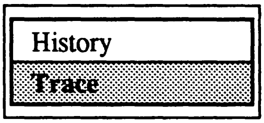
The **trace waveform** file contains data specified using the TRACE command.

To view the trace waveform file,



1. Select Waveform display from the View menu.

   A submenu appears with two options, History and Trace.

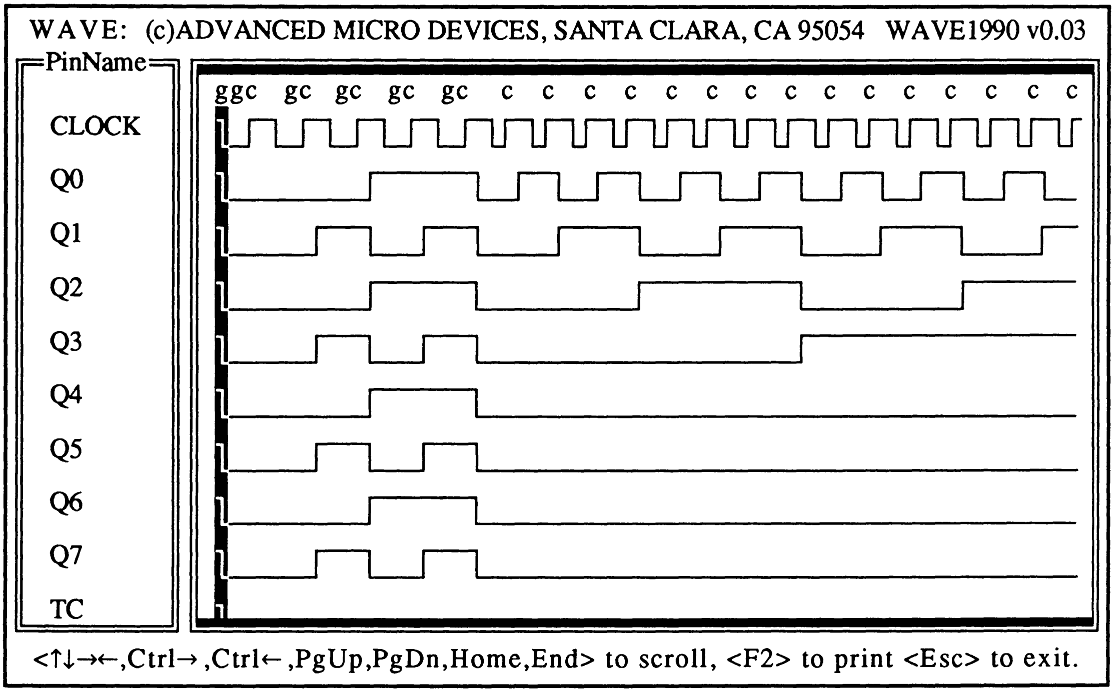


2. Select Trace from the submenu.

   A window opens and the file appears, as shown below.

   - The left column provides pin names.

   - The right column records high and low signals graphically.



```
WAVE: (c)ADVANCED MICRO DEVICES, SANTA CLARA, CA 95054   WAVE1990 v0.03
```

```
<↑↓→←,Ctrl→,Ctrl←,PgUp,PgDn,Home,End> to scroll, <F2> to print <Esc> to exit.
```

This file contains only the behavior of signals specified using the TRACE command in the simulation file.

- Each instance of g represents the SETF command in the simulation file.

- Each instance of c represents a complete clock cycle, which is defined by the CLOCKF command in the simulation file.

You can track values on the screen using the cursor, which is displayed as a thick vertical bar.

—>

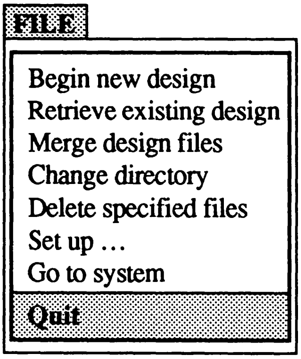
3. Press the right arrow key to move the cursor toward the right.

[Esc]

4. Press [Esc] to dismiss the file and return to the menu.

This concludes the tutorial.

To exit,



1. Select Quit from the File menu.

   A confirmation form appears.

Y

2. Type the letter Y to confirm.

SECTION II
DESIGNER'S GUIDE

CHAPTER 4

ENTRY

CONTENTS

4 ENTRY

This chapter discusses strategies for MACH-device design entry. Information here is divided into four major topics.

- The overview, 4.1, describes the methodologies supported for MACH-design entry.
- The design flow, discussed under 4.2, describes both schematic-based and text-based design flows in detail.
- The schematic verses text entry discussion, 4.3, provides considerations to help you decide on an entry method and describes design parameters you can control.
- The combining schematic and text descriptions discussion, 4.4, explains how to combine schematic and/or text files.
- The merging multiple PDS files discussion, 4.5, describes how to combine multiple text files into one MACH-device design.

4.1 OVERVIEW

How you design a PLD depends on the kind of device.

A. Enter the entire design as a single text file.

   Describing logic using Boolean equations or state-machine language is the traditional method.

B. Enter the MACH-device design in two or more text files and combine them into a single file.

   This method allows two or more people to work on the same MACH-device design. You can incorporate two or more existing PLD designs into a single MACH-device design.

C. Enter the entire MACH-device design as an OrCAD/SDT III schematic.

   This new method produces a graphic representation of the logic, which can include multiple sheets in a hierarchical structure.

D. Begin the MACH-device design as a schematic, convert schematic data into a PDS file, then edit the PDS file as needed to complete the design.

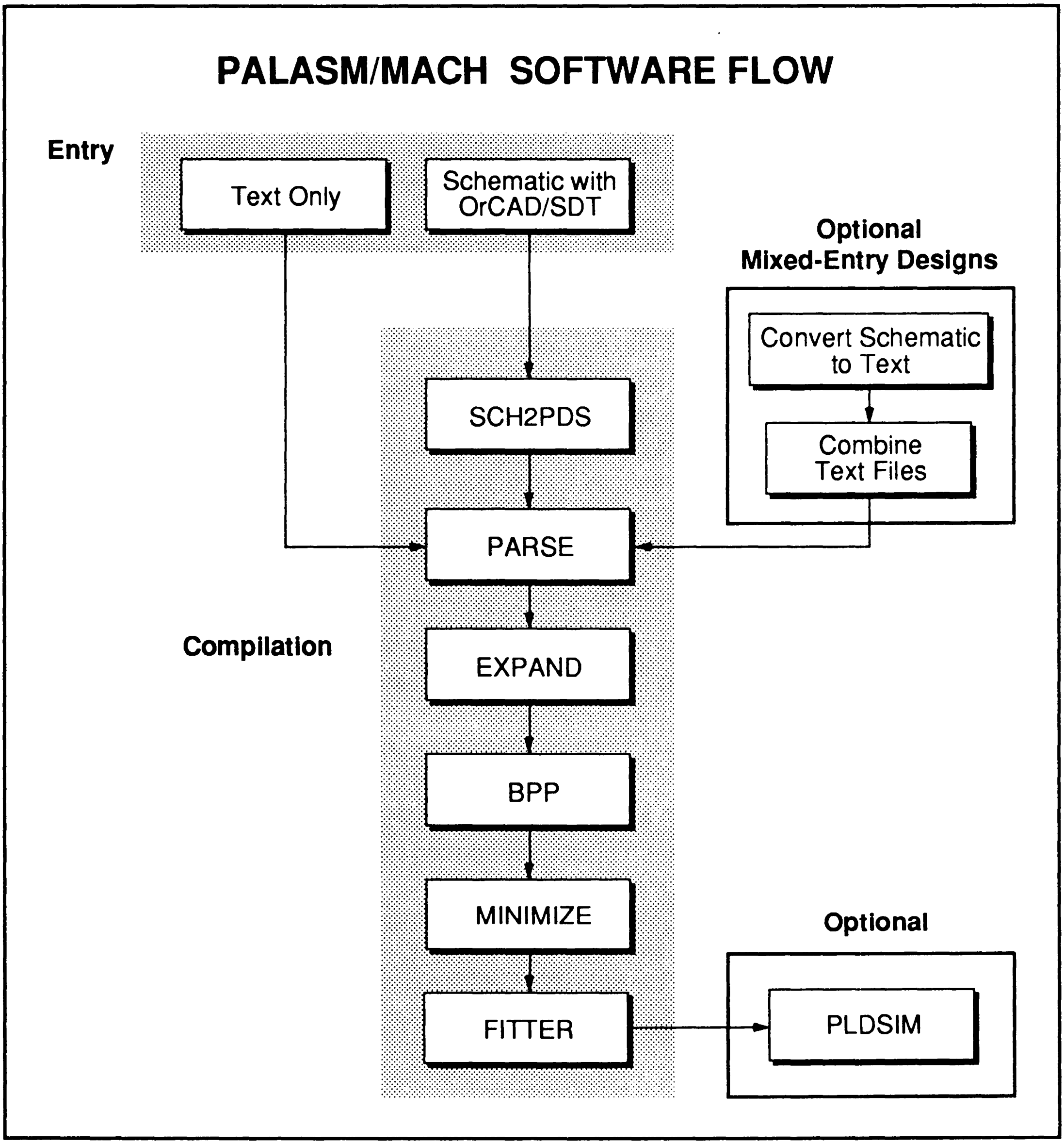
   This new method can be used when the AMD-supplied MACH-library elements do not provide the exact functionality you need.

E. Enter part of the MACH-device design as a schematic and part as a separate PDS file, convert schematic data to a PDS file, then combine the schematic-based PDS file with the text-based PDS file to produce a single new design file.

   This method provides maximum flexibility for MACH-device designs when part of the design is easier to enter as a schematic and the rest is best described through equations or state-machine language.

4.2 DESIGN FLOW

The diagram below shows the tasks associated with both schematic and text entry methods. The design process itself is iterative. For example, after compilation or simulation, you may want to return to the schematic or PDS file and change the design.



4.2.1 TEXT ENTRY

When you create a new design and specify text as the input format, an empty PDS file is created under the specified design name. A declaration-segment form appears on the screen to expedite entry of header statements, chip name and device type, and pin/node statements.[1]

When you leave the declaration-segment form, the text editor becomes available. The PDS file is displayed, which includes the declaration segment you just entered. You can add equations or state-machine constructs to complete the design.

When you leave the text editor, you're returned to the PALASM environment. You can compile, simulate, and download the design to a device programmer as usual. You can also combine this PDS file with others, including those produced from schematic-based designs.

4.2.2 SCHEMATIC ENTRY

When you begin a new MACH-device design and specify a schematic input format, two files are created.

- A blank schematic worksheet is created and named as you specified.
- A control file is created using the specified design name with a .CTL extension.

  The control file provides data for the declaration segment of the PDS file that's produced during compilation. The PDS file will include Boolean equations derived from converted schematic data.

The control-file form appears on the screen to streamline entry of the header-statement, chip name, and device-type.[2] When you leave the form, the

[1] Refer to Section IV, Chapter 9, for details about commands, input-file formats, and the text-based declaration-segment form.

[2] Refer to Section IV, Chapter 9, for details about commands and the schematic control-file form.

OrCAD/SDT III editor becomes available with the AMD-supplied MACH library. A named, blank worksheet appears so you can create a schematic and update the design file. When you leave OrCAD/SDT, you're returned to the PALASM software environment.

When you compile a schematic-based design, the following occurs.

- Data on each worksheet is converted into a single netlist.

- Individual netlists for the entire design are combined into one netlist.

- Data from the combined netlist is translated into Boolean equations and combined with control-file statements to produce the PDS file.

The resulting PDS file is then compiled. You can simulate,[3] modify and recompile, and download this file to a device programmer. In addition, you can combine this PDS file with others to create a single design for a MACH device. For example, you can combine up and down counter files together to produce a DMA address generator.

Important: Be sure to use only alphanumeric characters for signal names in a schematic. You cannot use a slash to indicate active-low polarity in a signal name in a schematic.

Also: You can connect a NODE macro only to a wire, not to a bus, in a schematic.

[3] To simulate a schematic-based design, you must create a separate simulation file or add a simulation section to the translated PDS file.

4.2.3 COMBINED ENTRY METHODS

There may be times when you want to use more than one entry method to produce a single design. Mixed entry methods are supported.

- You can enter part of a MACH-device design as a schematic and specify certain parameters using text.[4]

- You can **either** combine multiple text files for a new MACH-device design when several people work on different parts **or** merge an existing PLD design with new or existing PDS files to create a single MACH-device design.[5]

[4] Refer to discussion 4.6, in this chapter, for details about combining schematic and text descriptions.

[5] Refer to discussion 4.7, in this chapter, for details about merging multiple PDS files.

4.3 BOOLEAN DESIGN STRATEGIES

Discussions below provide strategies for Boolean-equation designs.

- 4.3.1, Output Polarity
- 4.3.2, Controlling Output Buffers Using .TRST
- 4.3.3, Controlling Clocks With .CLKF
- 4.3.4, Controlling SET/RESET Using .SETF/.RSTF
- 4.3.5, Using High-Level Constructs
- 4.3.6, Controlling Logic Reduction

4.3.1 OUTPUT POLARITY

An output pin that goes high when the corresponding equation is true is called active high. An output pin that goes low when the corresponding equation is true is called active low.

The PAL16L8 and PAL16R8 devices have outputs that are always inverted. Thus, these devices are commonly referred to as active-low devices. The PAL10H20EG8 device has outputs that are never inverted, and is commonly referred to as an active-high device.

Devices such as the PAL22V10 and PAL32VX10 have outputs that can be configured as either active-high or active-low output. This valuable feature is known as programmable polarity.

4.3.1.1 The Two Components of Polarity

In a PALASM design file, the active high/low nature of each pin is a function of its polarity definition. Polarity is defined in both the pin statement and the output equation.

The polarity rule for PALASM design files is defined below.

- If the equation and pin statement have the same polarity, the output is active high.

- If the equation and pin statement have opposite polarity, the output is active low.

Active-low polarity in a PALASM design file is indicated with a slash, /.

4.3.1.2 Controlling Polarity from the Equation

If you prefer to control output polarity from an equation, always use active-high (non-complemented) pin names in the pin statements.

```
PIN 14 O1      COMBINATORIAL      ;output
PIN 15 O2      REGISTERED         ;output
```

With active-high pin statements, the polarity of the output is the same as the polarity of the equation. You create an active-low equation by placing a slash before the pin name on the right-hand side of the equation.

```
O1 = I1 * I2              ;active-high output
/O2 = I1 * I2             ;active-low output
```

The recommended way to control polarity is to control it from an equation. This method of controlling polarity offers the following advantages and disadvantages.

- **Advantage**: You can determine the logic and polarity of any equation from the equation itself.
- **Advantage**: In state equations, you can specify the desired output at the pin without worrying about the pin's polarity. The PALASM software automatically adjusts the equations to deliver the specified signal to the output pin.
- **Disadvantage**: In simulation, you must use the TRACE_ON command to correct for active-low polarity.[6]

[6] Refer to Section IV, Chapter 10, for details on using the TRACE_ON command.

4.3.1.3 Controlling Polarity from the Pin or Node Statement

If you prefer to control output polarity from the pin or node statement, always use the non-complemented pin name on the left side of output equations.

```
O1 = I1 * I2
O2 = I1 * I2
```

If you choose this method, the polarity of the output is the same as the polarity of the pin or node statement.

```
PIN  14  O1  COMBINATORIAL        ;active-high output
PIN  15  /O2  REGISTERED          ;active-low output
```

This method of controlling polarity offers the following advantages and disadvantages.

- **Advantage**: The default simulation output shows the correct output polarity for each pin. The PALASM software inverts the name of each active-low pin automatically.
- **Disadvantage**: You must look at both the pin statement and the equation to determine the logic and polarity of an equation.
- **Disadvantage**: State-machine design is considerably more complicated with pins and nodes defined as active low.

Warning: If you choose to control polarity from the pin or node statements, you must know how the software handles polarity in the equations, state, and simulation segments.

4.3.1.4 Creating Equivalent Logic

In general, it is most efficient to implement active-high logic on active-high devices and active-low logic on active-low devices.

A device that has programmable polarity works equally well with active-high and active-low logic. The software automatically configures the device to the optimal polarity for each equation when the minimize routine is run. The optimal polarity is the one that results in the lowest number of product terms for each equation.

Sometimes, however, you must implement one or more equations in a device having non-optimal polarity. For example, you might wish to implement the following simple design.

```
/O1 = I1 *  I2 * I3
/O2 = I1 * /I2 * I3
 O3 = I3 * /I4
```

This design can fit easily on a PAL16L8, which is an active-low device. However, the active-high equation must be converted to an an equivalent active-low equation, using DeMorgan's theorem.

The final active-low equation is logically equivalent to the original active-high equation, except as follows.

- In the active-low form, the design can be implemented on an active-low device such as the PAL16L8.
- In this example, the active-low form requires more product terms than the active-high form of the same equation.

To determine how many product terms are required when you invert an equation, you must convert the equation using DeMorgan's theorem. However, in many cases, the number of product terms is not important as long as the design fits on the specified device. In this case, simply write the equation using whichever polarity is easiest and the equation is converted for you automatically.

Important: To have the software convert equations to the optimal polarity, you must set the compilation mode to automatic or select the Minimize Boolean option from the compile options form.[7] In addition, you must set the polarity minimization option in the Logic Synthesis form to Best for Device.

The minimization routine automatically converts the equations to match the polarity of the device. If you forget to run minimization and the equation's polarity does not match that of the device, an error is reported in the execution log.

If the device has programmable polarity, minimization chooses the polarity for each equation that results in the minimum number of product terms.

4.3.2 CONTROLLING OUTPUT BUFFERS USING .TRST

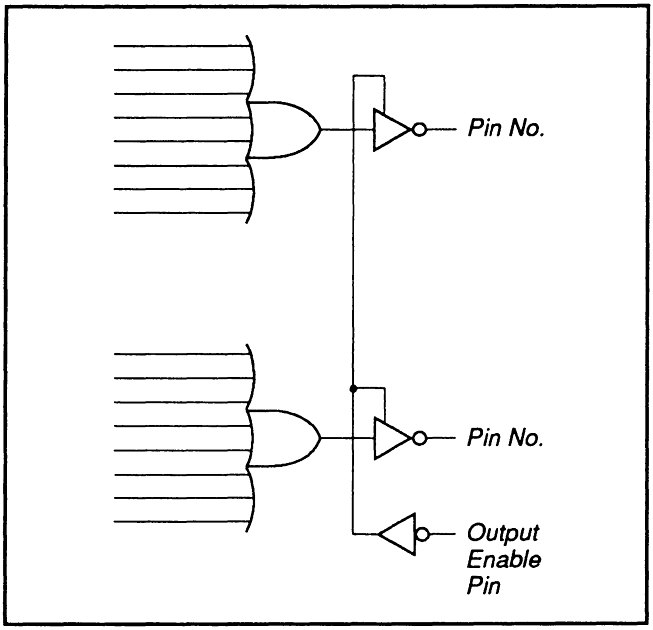
There are three basic configurations for three-state output buffers.

- Bank output enable
- Individual output enable
- Grouped output enable

4.3.2.1 Bank Output Enable

For devices with bank output enable, you use a special output-enable pin to control a group of outputs called a bank. The PAL16R8 is an example of this type of device.

[7] Refer to Section IV, Chapter 9, for information on compilation options.

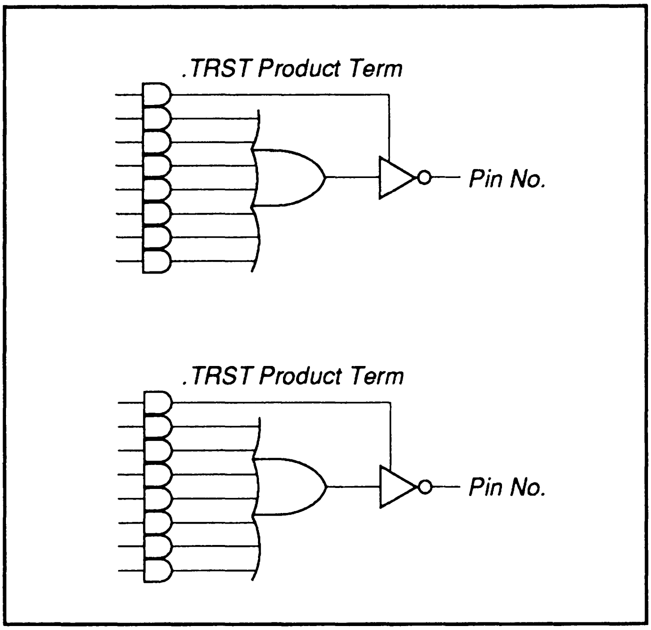


Bank Output Enable

For these types of devices, the outputs in the bank are always enabled unless you assert the output-enable pin high.

4.3.2.2 Individual Output Enable

For devices with individual output enable, you can control each output independently using an output-enable product term. The PAL16L8 is an example of this type of device.



Individual Output Enable

These types of devices are automatically programmed for the product terms to be unconditionally true unless you explicitly write an equation to control the three-state buffers.

To explicitly control the three-state buffer, you use a .TRST functional equation with the following syntax.

Pin name.TRST = Product term

You have three options when defining a .TRST equation.

- Enable the output buffer at all times.
- Disable the output buffer at all times.
- Enable the output buffer under certain conditions.

To enable the output buffer at all times, set the .TRST equation equal to Vcc. To disable the output buffer at all times, set the .TRST equation equal to GND. The following example unconditionally enables pin A and disables pin B.

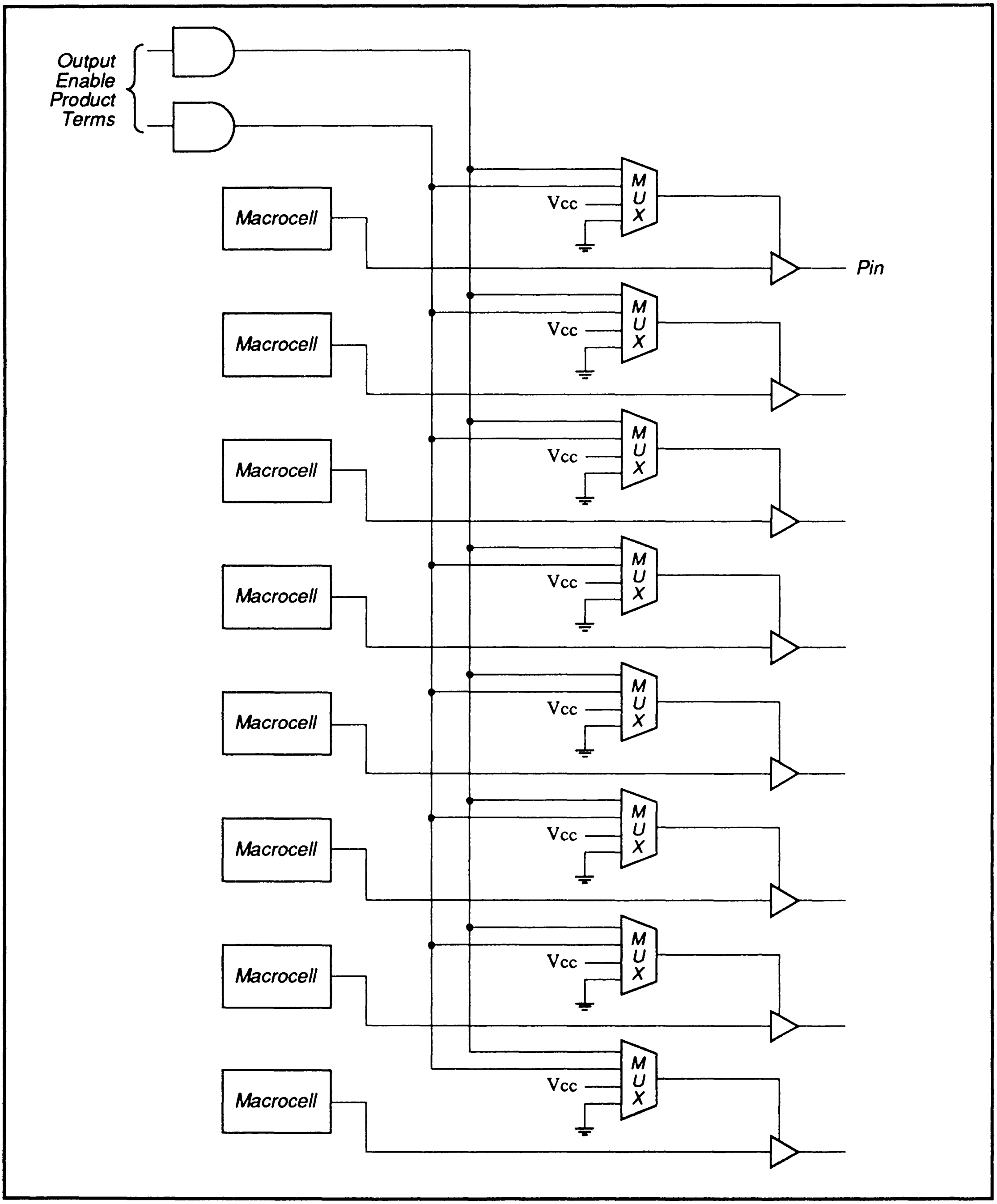
```
A.TRST = VCC  ;enables output A unconditionally
B.TRST = GND  ;disables output B unconditionally
```

To enable the output buffer under certain conditions, set the .TRST equation equal to a Boolean expression. The following example enables pin B when the signal GO is high and STOP is low. GO and STOP must be defined as pins or nodes in the declaration segment of the PDS file.

```
A.TRST = GO * /STOP
```

4.3.2.3 Grouped Output Enable

For devices with grouped output enable, one or more product terms are available to control a group of outputs. The MACH 110 is an example of this type of device.



MACH 110 Output Enable Resources For One Group

In the MACH 110, two product terms are provided to control each group of eight I/O cells. Within each group, each I/O cell can be permanently enabled, permanently disabled, or controlled by either of the two product terms.

You use the statements described under individual output enable to control the outputs. If you use more than the available number of product terms, the software will report an error during compilation.

4.3.3 CONTROLLING CLOCKS WITH .CLKF

You use a .CLKF functional equation to control the clock signal to flip-flops in a PAL device.

Some devices have a single dedicated clock input to all flip-flops. Others allow you to specify multiple clock signals or a different clock signal for each flip-flop.[8]

To control the clock of a flip-flop, you define the clock signal with a pin statement in the declaration segment of the PDS file. Then you use this signal in a .CLKF functional equation.

The following example shows how to define and use a clock signal for a PAL16R8.

```
;-------------------------------Declaration Segment------------------------------
PIN       1        CLK
PIN       2         A        ;INPUT
PIN       3         B        ;INPUT
PIN       12        AREG     ;OUTPUT

;---------------------------------Equations Segment------------------------------
AREG  = A + B
AREG.CLKF = CLK
```

Pin 1 is the clock pin on this device. The example above assigns the name CLK to this pin using a pin

[8] Refer to Section IV, Chapter 11, for information on the clocking capabilities of a specific device.

statement. You place the statement in the declaration segment of the PDS file. You can assign any valid name to the clock pin although it helps to use an easily understood name to represent the clock signal.

Then you use the clock signal in a .CLKF functional equation in the equations segment of the PDS file to control the clock of the register associated with output pin AREG.

4.3.4 CONTROLLING SET/RESET USING .SETF AND .RSTF

You use .SETF and .RSTF functional equations to control the set and reset functions of flip-flops in a PAL device.

The general forms for .SETF and .RSTF functional equations are shown below.

Pin name.SETF = <Pin or product term>
Pin name.RSTF = <Pin or product term[9]>

The following example defines the signals SET and RST and uses them in a functional equation to control the flip-flop associated with pin AREG.

```
;--------------------------------Declaration Segment--------------------------------
PIN      2       SET          ;INPUT
PIN      3       RST          ;INPUT
PIN      12      AREG         ;OUTPUT

;----------------------------------Equation Segment---------------------------------
AREG  = A + B
AREG.SETF = SET * /RST
AREG.RSTF = RST * /SET
```

4.3.4.1 Banking Set and Reset in MACH Devices

In the MACH device, all outputs within a bank share common set and reset signals. As a result, if you group signals in a common block, they do not require separate set or reset controls.

[9] Refer to Section IV , Chapter 10 for a description of set and reset resources available in each device.

However, if you do not write a .RSTF functional equation for one of the outputs in the block, it is not affected by assertion of the reset signal. The same holds true for set signals.

For example, the following statements cause A[0] through A[3] to be reset whenever the signal RST is high. A[4] through A[7] are not affected when RST is asserted.

```
;------------------------------- PIN Declarations --------------------------------
PIN          ?        RST                                     ;INPUT
PIN          ?        IN                                      ;INPUT
PIN          ?        A[7..0]       REGISTERED                ;OUTPUT
PIN          ?        CLK                                     ;CLOCK
GROUP MACH_SEG_A A[7..0]

;---------------------------- Boolean Equation Segment -----------------------------
EQUATIONS

A[3..0].RSTF = RST
A[7..0].CLKF = CLK
```

4.3.5 USING HIGH-LEVEL CONSTRUCTS

The following discussions provide information about vector notation, radix notation, IF-THEN-ELSE and Case constructs.

4.3.5.1 Vector Notation

A vector is a set of inputs, outputs, or internal nodes that have the same root name. Members of the vector are differentiated by subscript. For example, in the vector NAME[1..5], the members are shown next.

NAME[1]
NAME[2]
NAME[3]
NAME[4]
NAME[5]

Vectors are declared in pin and node statements and referenced in other statements. You must observe the following rules for vector notation.

- Declare all pins or nodes in the vector in one pin or node statement.

- Reference individual pins or nodes in a vector using subscripted pin or node names. Use the format NAME[1] rather than NAME 1.
- Reference groups of pins or nodes in a vector using the range operator. Use the format NAME[1..3].

You can include input and output pins in the same vector if your application calls for it, but you cannot include pins and nodes in the same vector. You define ranges of contiguous pins by separating the first and last members with two periods. For example, you specify the range of input pins 3 through 6 as follows.

```
3..6
```

To include non-contiguous pin numbers, you must separate them using commas as shown next.

```
1..4, 8..11
```

In the pin name field, enter the vector name followed by a range that indicates the desired subscripts. Enclose the range in square brackets as follows.

```
NAME[1..4]
```

Enter the pin numbers using range notation as indicated in the next example.

```
PIN  3..6  NAME[1..4]  COMBINATORIAL
```

In a MACH design, you can use a single question mark, ?, to float the location of all the pins in the vector as shown in the next example.

```
PIN     ?  NAME[1..4]  COMBINATORIAL
```

Enter the polarity and storage type attributes for the vector as you would for a single pin.

4.3.5.2 Radix Notation

A radix is a construct used on the right side of an equation, or in a case statement, to represent a number in binary, octal, decimal, or hexadecimal format. The radix is converted automatically to a binary bit pattern, which is then compared with a vector of pin or node values on the left side of the equation.

The decimal radix (base 10) is the default for case statements. You can also use binary, octal, or hexadecimal radices (base 2, 8, and 16, respectively).

To use a radix other than the default, you must precede the test condition with the appropriate radix operator. The table below shows the radix operators for all four radices supported by the software.

OPERATORS	DEFINITIONS
#b or #B	Specifies the binary radix, base 2
#o or #O	Specifies the octal radix, base 8
#d or #D	Specifies the decimal radix, base 10
#h or #H	Specifies the hexadecimal radix, base 16

If you omit the radix operator altogether, the default, which is decimal form, is assigned.

Examples of each are shown below.

#b1011 or #B1011	;represents 11_{10} (radix 2)
#o13 or #O13	;represents 11_{10} (radix 8)
11 or #d11 or #D11	;represents 11_{10} (radix 10)
#hB or #HB	;represents 11_{10} (radix 16)

When you use radix notation in a statement, it is automatically expanded to its binary equivalent and compared to the vector specified on the left side of the equation. If the binary equivalent does not have enough digits, leading zeros are added during processing as required.

The first number in the vector is the most significant bit. For example, in the vector ADDRESS[3..0], ADDRESS[3] is the most significant bit.

When the example below is used in a case construct, the radix on the right hand side of the equation is compared to the vector on the left.

ADDRESS[3..0] = 5

The radix 5 is expanded to its binary equivalent, 101.

Since the vector on the right side of the equation contains four signals, one leading zero is added to the binary equivalent, 0101.

The four signals in the vector are compared to their corresponding bits in the expanded radix as indicated below.

ADDRESS[3] compared to 0
ADDRESS[2] compared to 1
ADDRESS[1] compared to 0
ADDRESS[0] compared to 1

If all four conditions evaluate true, the equation is true.

Important: When comparing a vector to a radix, be careful to specify the order of the least and most significant bits correctly. For example, the first line gives different results than the second.

ADDRESS[3..0] = 5
ADDRESS[0..3] = 5.

4.3.5.3 IF-THEN-ELSE Statement

The IF-THEN-ELSE construct is a flow-of-control construct that expresses logical operations in natural language. You can use this construct as an alternative to writing Boolean equations.

The syntax for the IF-THEN-ELSE statement is as shown next.

```
  IF Test condition THEN
     BEGIN
           Action(s)           ;performed if test condition = true
     END
  ELSE
     BEGIN
           Action(s)           ;performed if test condition = false
     END
```

If you do not specify the else condition, it is treated as a don't care when the logic is generated.[10]

The following example shows testing the high order bit on an 8-bit address line. If it is equal to 1, the signal named HIBIT is set high and LO_BANK_ENA is set low. If it is equal to 0, HIBIT is set low and LO_BANK_ENA is set high.

```
;---------------------------------- PIN Declarations -----------------------------------
    PIN     ?     ADDRESS[7..0]                     ;INPUT
    PIN     ?     HIBIT          REGISTERED         ;OUTPUT
    PIN     ?     LO_BANK_ENA    REGISTERED         ;OUTPUT
    PIN     ?     CLK                               ;CLOCK

----------------------------- Boolean Equation Segment ---------------------------------
    EQUATIONS
    IF ADDRESS[7] = 1 THEN
         BEGIN
              HIBIT = 1
              LO_BANK_ENA = 0
         END
    ELSE
         BEGIN
              HIBIT = 0
              LO_BANK_ENA = 1
         END

    HIBIT.CLKF = CLK
    LO_BANK_ENA.CLKF = CLK
```

4.3.5.4 CASE Statement

The CASE statement is a flow-of-control construct, which is ideal when you need to test for a number of different conditions.

[10] Refer to Section IV, Chapter 10, for additional information on the IF-THEN-ELSE construct.

The syntax for the CASE statement is as follows.

```
CASE (Condition_signals)
    BEGIN
         Value: BEGIN
                        Action
                  END

         Value: BEGIN
                        Action
                  END

    OTHERWISE:  BEGIN
                       Action
                END
    END
```

The following example asserts enable lines for four peripheral devices named UNIT1 through UNIT4 by checking for their hexadecimal address on an 8-bit address line. The declarations are shown first.

```
;---------------------------------- PIN Declarations ----------------------------------
     PIN     ?     ADD[7..0]                      ;INPUT
     PIN     ?     UNIT1      REGISTERED          ;OUTPUT
     PIN     ?     UNIT2      REGISTERED          ;OUTPUT
     PIN     ?     UNIT3      REGISTERED          ;OUTPUT
     PIN     ?     UNIT4      REGISTERED          ;OUTPUT
     PIN     ?     CLK                            ;CLOCK
```

A special test condition, indicated by a value of 0, 1, 2 or 3 on the address bus, is checked. If this condition is detected, all four enable lines are asserted. The range notation is used to test for this condition, which results in a more compact notation.

```
;--------------------------------- Boolean Equation Segment -----------------------
EQUATIONS
CASE (ADD[7..0])
     BEGIN
          #h0F:
               BEGIN
                    UNIT1 = 1
                    UNIT2 = 0
                    UNIT3 = 0
                    UNIT4 = 0
               END
          #h2F:
               BEGIN
                    UNIT1 = 0
                    UNIT2 = 1
                    UNIT3 = 0
                    UNIT4 = 0
               END
          #h5F:
               BEGIN
                    UNIT1 = 0
                    UNIT2 = 0
                    UNIT3 = 1
                    UNIT4 = 0
               END
          #hFF:
               BEGIN
                    UNIT1 = 0
                    UNIT2 = 0
                    UNIT3 = 0
                    UNIT4 = 1
               END
          0..3:
               BEGIN
                    UNIT1 = 1
                    UNIT2 = 1
                    UNIT3 = 1
                    UNIT4 = 1
               END
     OTHERWISE:
          BEGIN
               UNIT1 = 0
               UNIT2 = 0
               UNIT3 = 0
               UNIT4 = 0
          END
     END

UNIT1.CLKF = CLK
UNIT2.CLKF = CLK
UNIT3.CLKF = CLK
UNIT4.CLKF = CLK
```

4.3.6 CONTROLLING LOGIC REDUCTION

The two ways to control logic reduction are, globally, from the compilation options form and locally, using statements in the PDS file

To disable logic reduction for the entire design file, use the options below on the Compilation Options form to set the run mode to manual and disable minimization.

Run mode:	Manual
...	
Minimize Boolean:	N

To selectively disable logic reduction, use a pair of MINIMIZE_OFF and MINIMIZE_ON commands.

Important: Turning Minimize on and off only affects certain aspects of minimization.[11]

[11] Refer to Section IV, Chapter 10, for additional information on the effects of MINIMIZE_ON and MINIMIZE_OFF.

4.4 STATE-MACHINE DESIGN STRATEGIES

Building a PALASM design file for a state machine is similar to building a PALASM design file for a Boolean design.

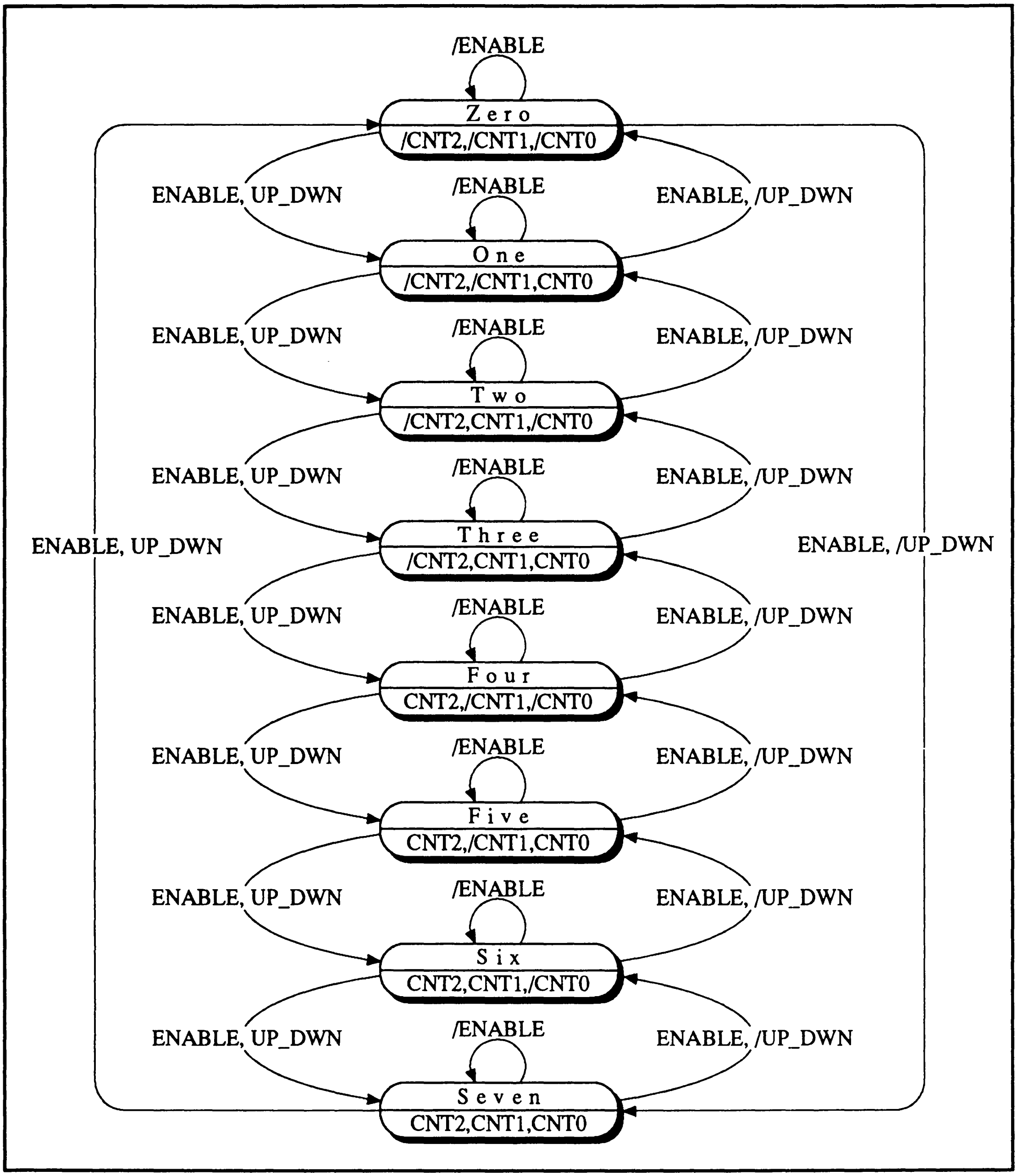
Before entering your state-machine description, it is useful to draw a state diagram. This diagram helps you determine the transitions from one state to another and the conditions that cause these transitions.

A state diagram for a 3-bit up/down counter is illustrated in the diagram on the next page.

This design is implemented as a Moore machine and uses the following inputs and outputs.

SIGNAL	DESCRIPTION
ENABLE	High value enables the counter. Low value disables the counter.
UP_DWN	High value indicates up count. Low value indicates down count.
CNT2	Most significant output bit of counter.
CNT1	Next significant output bit of counter.
CNT0	Least significant output bit of counter.

In the next diagram, the entry in the top half of a bubble indicates the name of the state. The entry in the bottom half of a bubble indicates the value of the outputs for that state. An arrow indicates a possible transition from one state to another. The values of the inputs next to the arrow indicate the input condition that causes that transition.



State-Machine Diagram

4.4.1 STATE SEGMENT OVERVIEW

The state-machine design file must include a program segment identified with the keyword STATE. This is called the state segment.

Note: The state segment typically replaces the equations segment. It is possible to modify state equations with Boolean equations by including both equation and state segments, in any order. If your design contains a state segment and a Boolean segment, you must select the Merge Mixed Mode option from the Compile Setup menu.

The state segment consists of the following syntax elements.

SYNTAX	DEFINITION
State	This identifies the state machine segment of the PDS file.
Machine-type	This identifies the state-machine type as either Moore or Mealy.
Start Up	This defines the state of the machine at power-up.
Global Defaults	This defines the default transitions if none of the specified conditions for a state are satisfied.
Transition Equations	This section defines the transitions from one state to the next.
Output Equations	This section defines the outputs for each possible state.
State Assignments	This optional section defines each state as a unique pattern of state bits.
Condition Equations	This section defines the set of inputs that represents each condition.

4.4.2 DEFINING MOORE AND MEALY MACHINES

State-machine designs are divided into two basic types: Moore and Mealy.

- Outputs in a Moore machine are dependent only on the present state.

- Outputs in a Mealy machine are dependent on the present state and the present inputs.

You begin the state segment with the keyword STATE on a new line. Then you define the state-machine type using one of the state-machine-type keywords.

```
MOORE_MACHINE
or
MEALY_MACHINE
```

The default is Mealy.

A state-machine design must be either all Moore or all Mealy, since the PALASM 4 software does not allow you to mix types in the same state machine. If even one state uses outputs that are input-dependent, you must convert the entire design to a Mealy machine.

> Note: You can add Mealy features to a Moore Machine by writing a Boolean equation segment that further decodes the state machine's inputs and outputs. To compile a design that contains state machine and Boolean equation segments, you must specify Y to Merge mixed mode on the Compilations Options form.

Another reason to convert a Moore design to Mealy is to reduce the total number of states in a design. If you are running short of flip-flops in which to store state bits, you may be able to reduce the number of states, and thus the number of state bit flip-flops required, by implementing the design in Mealy form. To reduce the number of states, the application must include cases in which multiple states can be collapsed down to a single state that produces different outputs depending on the inputs.

Do not convert a Mealy design to the Moore model unless Mealy-specific features are deleted. If the Mealy design includes multiple transitions to the same state, each having different outputs, the equivalent Moore design will require additional states. In some cases, a Moore design will not fit on a given device, while the same design implemented in Mealy form will fit.

4.4.3 CREATING STATE-MACHINE EQUATIONS

There are four types of state-machine equations. They have the following functions.

- Transition equations (required)

  For each state, the equations specify what the next state will be under various conditions. See Condition Equations below.

- Output Equations (optional)

  These equations specify the outputs of the state machine. No output cases are required when the state bits themselves are the outputs.

- Condition Equations (normally required)

  These equations specify a condition name for each set of input values used to determine a transition. You can use input names directly only if a single input controls the transition; otherwise, you must use condition names.

- State-Assignment Equations (optional)

  These equations specify the bit code to be assigned to each state name used in the design. If these equations are omitted, the software will assign the bit codes automatically.

4.4.3.1 Condition Equations

You must replace each set of inputs that controls a transition with a logical name, called a condition.

The condition equations, preceded by the keyword CONDITIONS, must appear either before the keyword STATE or after all state-segment statements. CONDITIONS are written as simple Boolean equations.

```
CONDITIONS
Condition 1 = Boolean Expression
Condition 2 = Boolean Expression
...
Condition n = Boolean Expression
```

- If a condition consists of a single input, you can use the input name instead of a condition equation.
- If two conditions evaluate true at the same time, the software issues an overlapping condition error message.

```
>> ERROR Overlapping state transition conditions
```

To remove the overlapping conditions, you must write the equations so that no more than one equation can be evaluated as true at any time.

4.4.3.2 Transition Equations

You must write one transition equation for each state. Within each state's transition equation, you must also write one expression to define each possible transition to a next state.

Use default branches to define the next state if the inputs fail to match any of the transition conditions defined for the present state. Global defaults specify the default procedure for the entire state-machine design. Local defaults specify the default procedure for one state only.[12]

[12] Refer to discussion 4.4.4, in this chapter, for additional information on default branches.

```
Present state := Condition name -> Next state
              + Condition name -> Next state
              ...
              + ->State name   ;default  branch
```

4.4.3.3 Output Equations

To specify outputs for a Moore machine, you need to specify only the present state and the desired outputs, since the outputs are not affected by input conditions. The syntax for a Moore machine output equation follows.

```
Statename.OUTF = Output expression
```

To specify outputs for a Mealy machine you must specify the input condition along with the present state. The syntax for Mealy machine output equations is as follows.

```
Statename.OUTF = Condition 1 -> Output 1
               + Condition 2 -> Output 2
              ...
               + Condition n -> Output n
```

The software allows you to specify the desired output pin values for each state or transition, without regard to the polarity of the device. The output equations are adjusted automatically to produce the requested behavior.

If you define the output pins as active low by using complemented pin names in the pin statements, the output pin will have the opposite value of the equation.

4.4.3.4 State-Machine Example

The following example shows a 3-bit up/down counter described in state-machine language. The declaration segment is shown next.

```
;------------------------------- Declaration Segment ------------------------------
     TITLE    COUNTER STATE MACHINE
     ..
     CHIP  _CTR  MACH110

;----------------------------------- PIN Declarations -----------------------------
     PIN     35    CLOCK                     ;CLOCK
     PIN     ?     ENABLE                    ;ENABLE
     PIN     ?     UP_DWN                    ;INPUT
     PIN     ?     CNT0         COMB         ;OUTPUT
     PIN     ?     CNT1         COMB         ;OUTPUT
     PIN     ?     CNT2         COMB         ;OUTPUT

;------------------------------------- State Segment ------------------------------
     STATE
     MOORE_MACHINE

     ZERO       := UP   -> ONE
                + DOWN -> SEVEN
                + STOP -> ZERO

     ONE        := UP   -> TWO
                + DOWN -> ZERO
                + STOP -> ONE

     TWO        := UP   -> THREE
                + DOWN -> ONE
                + STOP -> TWO

     THREE      := UP   -> FOUR
                + DOWN -> TWO
                + STOP -> THREE

     FOUR       := UP   -> FIVE
                + DOWN -> THREE
                + STOP -> FOUR

     FIVE       := UP   -> SIX
                + DOWN -> FOUR
                + STOP -> FIVE

     SIX        := UP   -> SEVEN
                + DOWN -> FIVE
                + STOP -> SIX
```

Continued on next page

Continued from prior page

```
SEVEN     := UP   -> ZERO
          + DOWN -> SIX
          + STOP -> SEVEN

ZERO.OUTF     = /CNT2*/CNT1*/CNT0
ONE.OUTF      =  CNT2*/CNT1* CNT0
TWO.OUTF      = /CNT2*CNT1* CNT0
THREE.OUTF    = /CNT2* CNT1* CNT0
FOUR.OUTF     =  CNT2*/CNT1*/CNT0
FIVE.OUTF     =  CNT2*/CNT1* CNT0
SIX.OUTF      =  CNT2* CNT1*/CNT0
SEVEN.OUTF    =  CNT2* CNT1* CNT0

CONDITIONS
UP = ENABLE * UP_DWN
DOWN = ENABLE * /UP_DWN
STOP  = /ENABLE
```

4.4.4 DEFAULT BRANCHES

You use default branches to define the next state should the inputs fail to match any of the transition conditions defined for the present state.

The software supports two types of defaults.

- Global defaults specify the default branch for all states except those for which local defaults are defined.
- Local defaults specify the default branch for one state only.

You can include both local and global defaults in your design. Local defaults will override global defaults.

4.4.4.1 Global Defaults

Global defaults are defined after the machine-type definition. The global default statement can specify the default branch in one of three ways. The statement below causes the state machine to remain in the same state if the inputs do not match any of the defined transition conditions for that state.

```
DEFAULT_BRANCH  HOLD_STATE
```

The following statement causes the state machine to branch to the specified state if the inputs do not match any of the defined transition conditions for that state.

```
DEFAULT_BRANCH  State name
```

The next statement causes the state machine to branch to the next state if the inputs do not match any of the defined transition conditions for that state. The next state is defined as the state whose transition equation follows the transition equation for the present state in the PDS file. There is no next-state branch possible from the state whose transition equations appear last.

```
DEFAULT_BRANCH  NEXT_STATE
```

4.4.4.2 Local Defaults

Unlike global defaults, local defaults always specify a branch to a specific state. Local defaults can be used alone or in combination with global defaults.

- In combination with global defaults, local defaults provide a mechanism for defining default branches that differ from the norm.
- Used alone, local defaults offer a way to specify each default branch explicitly. Local defaults allow you to see all possible branches from a given state at one glance.

Local defaults appear as the last line in a transition equation, using the special symbol +->, which is formed by typing the characters +, -, and >.

```
Present state := Condition name -> Next state
          + Condition name -> Next state
          ...
          + ->State name   ;default  branch
```

4.4.4.3 Example With Default Branches

The following example shows the declaration segment of the 3-bit counter that will be modified to include a global default branch. The declaration segment is summarized below.

```
;-----------------------------------Declaration Segment ------------------------------
     TITLE  COUNTER STATE MACHINE WITH DEFAULT BRANCH
     ...
     CHIP  _CTR  MACH110

;------------------------------------ PIN Declarations --------------------------------
     PIN     35        CLOCK                 ;CLOCK
     PIN     ?         ENABLE                ;ENABLE
     PIN     ?         UP_DWN                ;INPUT
     PIN     ?         CNT0      COMB        ;OUTPUT
     PIN     ?         CNT1      COMB        ;OUTPUT
     PIN     ?         CNT2      COMB        ;OUTPUT

;-------------------------------------- State Segment ---------------------------------
     STATE
     MOORE_MACHINE
```

By specifying the hold state as the global default as shown next, you can remove the Stop condition from each of the transition equations. If the conditions UP or DOWN are not satisfied, the current state will be held. This results in the same behavior as the previous example.

```
DEFAULT_BRANCH HOLD_STATE

ZERO       := UP   -> ONE
           + DOWN -> SEVEN

ONE        := UP   -> TWO
           + DOWN -> ZERO

TWO        := UP   -> THREE
           + DOWN -> ONE

THREE      := UP   -> FOUR
           + DOWN -> TWO

FOUR       := UP   -> FIVE
           + DOWN -> THREE

FIVE       := UP   -> SIX
           + DOWN -> FOUR

SIX        := UP   -> SEVEN
           + DOWN -> FIVE

SEVEN      := UP   -> ZERO
           + DOWN -> SIX

ZERO.OUTF     =  /CNT2*/CNT1*/CNT0
ONE.OUTF      =   CNT2*/CNT1* CNT0
TWO.OUTF      =  /CNT2* CNT1*/CNT0
THREE.OUTF    =  /CNT2* CNT1* CNT0
FOUR.OUTF     =   CNT2*/CNT1*/CNT0
FIVE.OUTF     =   CNT2*/CNT1* CNT0
SIX.OUTF      =   CNT2* CNT1*/CNT0
SEVEN.OUTF    =   CNT2* CNT1* CNT0

CONDITIONS
UP = ENABLE * UP_DWN
DOWN = ENABLE * /UP_DWN
```

4.4.5 ASSIGNING STATE BITS

In some applications, you must control the assignment of the state-bit code. However, most of the time, the state-bit code is not important as long as it allows the device to differentiate between states.

4.4.5.1 Automatic State-Bit Assignment

You can allow the software to assign state bit-codes to state registers automatically. To do this, simply omit the state assignment equations. When the file is compiled, the software displays the following type of message to the screen and writes it to the log file.

```
|> WARNING E1351  Automatically assigning state
     bit _ST0 to ? NODE.
|> WARNING E1351  Automatically assigning state
     bit _ST1 to ? NODE.
|> WARNING E1351  Automatically assigning state
     bit _ST2 to ? NODE.

STATE REGISTERS USED

PIN NUMBER:     PIN NAME:
  ? NODE_           _ST0
  ? NODE            _ST1
  ? NODE            _ST2

STATE BIT ASSIGNMENT USED

  STATE NAME:         STATE REGISTERS VALUES:
          _ST2        _ST1      _ST0
ZERO       0           0         0
ONE        0           0         1
TWO        0           1         0
THREE      0           1         1
FOUR       1           0         0
FIVE       1           0         1
SIX        1           1         0
SEVEN      1           1         1
```

The warning message lists the pins to which state bits were assigned and the state-bit code for each state. In the 3-bit counter example, three-state registers are used to allow for 8 possible states. These are defined as nodes and named _ST0, _ST1 and _ST2.

State ZERO is assigned the bit code 0,0,0 which means all the state registers are low. State ONE is assigned bit code 0,0,1. A bit code for each state is listed with the message.

The first state defined in the transition equations is the first to be assigned a state code. If there is no start-up statement, the software assigns the first state all zeros when the device specifies power-up reset, and all ones when the device specifies power-up preset.

4.4.5.2 Manual State-Bit Assignment

You can control state-bit assignment manually using state assignment equations. To do this, you must define a pin or node for each of the state bits. You do this in the declaration segment of the PDS file just as you would define any pin or node. Then you write an equation for each state specifying the value of the state bits in Boolean format.

State name = Boolean Equation

If you don't need to use the state bits as outputs and the device you are using contains buried flip-flops, you can assign state bits to them. This will save output pins that can be used for other purposes.

4.4.5.3 Choosing State-Bit Assignments

The state-bit assignments you choose have a large impact on the number of product terms that will be required to implement your design. If you choose assignments so that the state-register bits change by only one bit at a time, as the state machine goes from state to state, the number of product terms will often be reduced.

For example, consider a design consisting of four states, A, B, C and D, where the transition between states is alphabetical. One possible assignment is to use a simple binary count as follows.

STATE	BIT ASSIGNMENT
A	00
B	01
C	10
D	11

Notice that this assignment causes two bits to change as the machine moves from state B to state C. The following is a better assignment for product-term reduction.

STATE	BIT ASSIGNMENT
A	00
B	01
C	11
D	10

Notice that this assignment causes only one bit to change as the machine moves from B to C.

If you need to use the state bits as outputs to identify when the machine is in a particular state, you can minimize the number of required outputs by choosing state bits appropriately.

For example, consider a design that has six states, A through F, where you need to identify states C, D and E. The following assignment allows you to identify these states using only one output pin.

STATE	BIT2	BIT1	BIT0
A	0	0	0
B	0	0	1
C	1	0	1
D	1	1	1
E	1	1	0
F	0	1	0

This assignment lets you use BIT2 as an output to identify when the machine is in any of the three states of interest. BIT2 can be assigned to an output pin and BIT1 and BIT0 can be assigned to buried nodes, freeing output pin resources.

4.4.5.4 Example Using Manual State-Bit Assignment

The following example uses state-assignment equations to manually assign the state bits to nodes named BIT0, BIT1 and BIT2.

```
;------------------------------- Declaration Segment ------------------------------
     TITLE  COUNTER STATE MACHINE WITH STATE BIT ASSIGNMENT
     ...
     CHIP  _CTR  MACH110

;------------------------------------- PIN Declaration------------------------------
     PIN     35    CLOCK                   ;CLOCK
     PIN     ?     ENABLE                  ;ENABLE
     PIN     ?     UP_DWN                  ;INPUT
     PIN     ?     CNT0     COMB           ;OUTPUT
     PIN     ?     CNT1     COMB           ;OUTPUT
     PIN     ?     CNT2     COMB           ;OUTPUT
     NODE    ?     BIT0     REGISTERED     ;OUTPUT
     NODE    ?     BIT1     REGISTERED     ;OUTPUT
     NODE    ?     BIT2     REGISTERED     ;OUTPUT

;------------------------------------ State Segment ----------------------------------
     STATE
     MOORE_MACHINE

     DEFAULT_BRANCH HOLD_STATE

     ZERO      := UP   -> ONE
               + DOWN -> SEVEN

     ONE       := UP   -> TWO
               + DOWN -> ZERO

     TWO       := UP   -> THREE
               + DOWN -> ONE

     THREE     := UP   -> FOUR
               + DOWN -> TWO

     FOUR      := UP   -> FIVE
               + DOWN -> THREE

     FIVE      := UP   -> SIX
               + DOWN -> FOUR

     SIX       := UP   -> SEVEN
               + DOWN -> FIVE
```

Continued on next page

Continued from prior page

```
SEVEN     := UP   -> ZERO
          + DOWN -> SIX

ZERO      =  /BIT2 * /BIT1 */BIT0
ONE       =  /BIT2 * /BIT1 * BIT0
TWO       =  /BIT2 *  BIT1 */BIT0
THREE     =  /BIT2 *  BIT1 * BIT0
FOUR      =   BIT2 * /BIT1 */BIT0
FIVE      =   BIT2 * /BIT1 * BIT0
SIX       =   BIT2 *  BIT1 */BIT0
SEVEN     =   BIT2 *  BIT1 * BIT0

ZERO.OUTF    =  /CNT2*/CNT1*/CNT0
ONE.OUTF     =  /CNT2*/CNT1* CNT0
TWO.OUTF     =  /CNT2* CNT1*/CNT0
THREE.OUTF   =  /CNT2* CNT1* CNT0
FOUR.OUTF    =   CNT2*/CNT1*/CNT0
FIVE.OUTF    =   CNT2*/CNT1* CNT0
SIX.OUTF     =   CNT2* CNT1*/CNT0
SEVEN.OUTF   =   CNT2* CNT1* CNT0

CONDITIONS
UP = ENABLE * UP_DWN
DOWN = ENABLE * /UP_DWN
```

4.4.6 USING STATE BITS AS OUTPUTS

Combining the state and output functions allows you to use less resources than if you use separate state bits and output bits. This can sometimes allow you to implement a design in a device that could not otherwise accommodate it.

Due to practical considerations, you can occasionally create a state-machine design where all of the outputs are also used as state bits. To do this, your design must meet three conditions.

- All state bits must be stored in flip-flops that are associated with output or I/O pins.
- The desired output in each state must be different from the desired output in every other state.
- The outputs in the design that combine state bits and outputs cannot be combinatorial, since the state bits must be registered.

To use state bits as outputs, you write state-assignment equations.[13] Make sure the state bits are assigned to registered pins in the declaration segment of the PDS file. Then you simply omit the output equations from the design.

4.4.7 INITIALIZING A STATE MACHINE

You use initialization routines to ensure the state machine powers up in a known state or branches to a known state whenever the initialization condition occurs.

The START_UP command [14] allows you to specify the starting state for devices that always power up with all bits high or all bits low, or that can be programmed to power up in any configuration.

The following is the syntax for Moore machines.

```
START_UP := POWER_UP -> State name
          + Condition1 -> State Name
```

The following is the syntax for Mealy machines.

```
START_UP.OUTF := POWER_UP -> Outputs
               + Condition1 -> Outputs
```

The power-up parameter has the following effects.

- In devices that initialize with all flip-flops high or all flip-flops low, the START_UP command assigns the appropriate all-high or all-low state-bit code to the specified state.

- In devices with programmable power up, the START_UP command programs the device to power up in the specified state. If you specify a

[13] Refer to discussion 4.4.6, in this chapter, for additional information about using state bits as outputs.

[14] Refer to Section IV, Chapter 10, for more information on the START_UP command.

particular state-bit code using the manual state-bit assignment syntax, the software programs the flip-flops to initialize with the specified values.

If you do not include a start-up statement, the device will power up in the state that appears in the first transition equation in the PDS file.

This condition lets you specify an asynchronous branch to a specific state whenever the specified condition occurs. For example, you can specify a transition to state zero in the event of the condition INIT. To do this, you must define INIT as a condition in the design file.

4.4.8 CLOCKING A STATE MACHINE

The clock input to the state registers is normally connected to the default clock. For devices with multiple clock sources or clocks formed by product terms, there are two ways to use a clock other than the default.

- The clock source equation is placed in the state segment of a PDS file and is used to specify a clock signal for all flip-flops in the state machine.

  The following is syntax for clock source equations.

  CLKF = Clock Signal

- The .CLKF function equation is placed in the equation segment of the PDS file. To use this method, you must declare the state registers, manually assign the state bits, and write a .CLKF equation for each register in the state machine.

4.4.8.1 Example Using State Bits as Outputs, Power-Up and Clock Equations

The following example modifies the 3-bit counter design to add a power-up routine, use the state bits as outputs, and specify a clock signal other than the default.

Notice that the state bits have been defined as pins instead of nodes and the output equations have been removed.

```
;------------------------------ Declaration Segment ------------------------------------
     TITLE COUNTER STATE MACHINE USING STATE BITS AS OUTPUTS, INITIALIZATION AND NON-
     DEFAULT CLOCKING
     ...
     CHIP  _CTR  MACH110

;--------------------------------- PIN Declarations ------------------------------------
     PIN     13    CLOCK                       ;CLOCK
     PIN     ?     ENABLE                      ;ENABLE
     PIN     ?     UP_DWN                      ;INPUT
     PIN     ?     BIT0     REGISTERED         ;OUTPUT
     PIN     ?     BIT1     REGISTERED         ;OUTPUT
     PIN     ?     BIT2     REGISTERED         ;OUTPUT
;----------------------------------- State Segment -------------------------------------
     STATE
     MOORE_MACHINE
     START_UP := POWER_UP -> ZERO

     CLKF = CLOCK

     DEFAULT_BRANCH HOLD_STATE

     ZERO      := UP   -> ONE
               + DOWN -> SEVEN

     ONE       := UP   -> TWO
               + DOWN -> ZERO

     TWO       := UP   -> THREE
               + DOWN -> ONE

     THREE     := UP   -> FOUR
               + DOWN -> TWO

     FOUR      := UP   -> FIVE
               + DOWN -> THREE
```

Continued on next page

Continued from prior page

```
FIVE      := UP   -> SIX
           + DOWN -> FOUR

SIX       := UP   -> SEVEN
           + DOWN -> FIVE

SEVEN     := UP   -> ZERO
           + DOWN -> SIX

ZERO     =  /BIT2 * /BIT1 * /BIT0
ONE      =  /BIT2 * /BIT1 *  BIT0
TWO      =  /BIT2 *  BIT1 * /BIT0
THREE    =  /BIT2 *  BIT1 *  BIT0
FOUR     =   BIT2 * /BIT1 * /BIT0
FIVE     =   BIT2 * /BIT1 *  BIT0
SIX      =   BIT2 *  BIT1 * /BIT0
SEVEN    =   BIT2 *  BIT1 *  BIT0

CONDITIONS
UP = ENABLE * UP_DWN
DOWN = ENABLE * /UP-DWN
```

4.5 SCHEMATIC VERSUS TEXT ENTRY

There are several factors that can impact your decision to use one entry method over another.

- Your own experience and preference
- Control over design parameters
- Design-documentation requirements
- Device type
- Content of the AMD-supplied MACH library

For example, **text entry** is an excellent method when you produce a design for any PLD. In this case, you can choose between Boolean equation or state-machine descriptions or produce a design using both.

However, **schematic entry** may be best when you are producing a MACH-device design and

- you either prefer graphic entry or require a schematic for design documentation, and

- the MACH library provides the logic you need.

The amount of **control** you need over design parameters, such as pin and node locations and logic-bank assignment is very important. In general, it's easier to specify parameters for large portions of a design as text because a single description line can be used. To specify these parameters in a schematic, you must edit the part-fields of each symbol separately.

Discussions below provide additional considerations.

- 4.3.1, Library Analysis
- 4.3.2, Schematic Parameters
- 4.3.3, Design Documentation Issues
- 4.3.4, Converting Existing Schematics to MACH-device designs

4.5.1 LIBRARY ANALYSIS

An important consideration when choosing an entry method is the availability of elements in the AMD-supplied library for MACH-device designs and the nature of the design's logic.

- **Schematic** entry may be easier than text-based descriptions when library elements provide the exact functionality you need, or when logic modification simply means removing one or more signals.

  For example, by wiring unused outputs to the NC macro and unused inputs to a PUP or PDWN macro, you can direct the removal of unused logic during the compilation process. In this case, any signal without a load is deleted along with any gate that is disabled by tying its inputs high or low. You use the appropriate macro for an efficient implementation even if all macro functions are not used.

- **Text** entry may be easier than schematic entry when library elements do not exist for the function you need **and** design logic is well structured, as in a counter.

  If you must add control signals to a macro, text entry may be better choice.

Suppose your task is to enter the logic for a **4-bit decade counter**. The MACH library includes a 74162 macro, a 4-bit decade counter with preset and clear capabilities, and a ripple carry output.[15] In this case, **schematic** entry is easy: you just retrieve the library symbol, place it on the worksheet, and wire it to the rest of the circuit.

Text entry may be easier when you want the logic to behave like the decade counter, but you want it to **skip the fourth state**: that is, count 0,1,2,4,5,6,7,8,9. In

[15] Refer to Section III, Chapter 8, for the 74162 macro datasheet.

this case, you cannot easily use the 74162 macro. If you choose schematic entry, you must design the gate-level logic and enter it using gate and flip-flop elements. It is easier to enter this type of structured logic using appropriate state-machine language syntax.

If you want to enter logic for a skip counter, but the **state** you want to skip is subject to **change**, text entry is also the better choice. A small change to the text that describes the counter will suffice. If you use schematic entry, you must redesign the logic and make extensive changes to the schematic.

4.5.2 SCHEMATIC PARAMETERS

Certain schematic macros provide control over the following parameters; you just edit the corresponding part field on the macro.[16]

- Pin or node location, part-field 1
- Logic block assignment, part-field 2
- Minimization control, part-field 3

Other macros are provided to help you specify asynchronous set-/reset-control signals and the removal of unused logic. Discussions that follow identify strategies for the activities listed below.

- 4.5.2.1, Fixing Pin Locations
- 4.5.2.2, Fixing Node Locations
- 4.5.2.3, Assigning Logic to a Block
- 4.5.2.4, Controlling Minimization
- 4.5.2.5, Controlling Set/Reset
- 4.5.2.6, Deleting Unused Logic

[16] Refer to Section III, Chapter 7 for details about macros that support part field assignments.

4.5.2.1 Fixing Pin Locations

When you do not specify a fixed pin location, it is left floating and the software determines its location on the device. In a schematic, you can specify device pin locations as follows.

1. Place an unresolved module port[17] on the sheet.
2. Attach a NODE macro to the module port net.
3. Edit part field 1 of the NODE macro.[18]

After schematic data is converted to equations, specified locations appear in the location-number field of the pin statement, in the declaration segment of the PDS file.

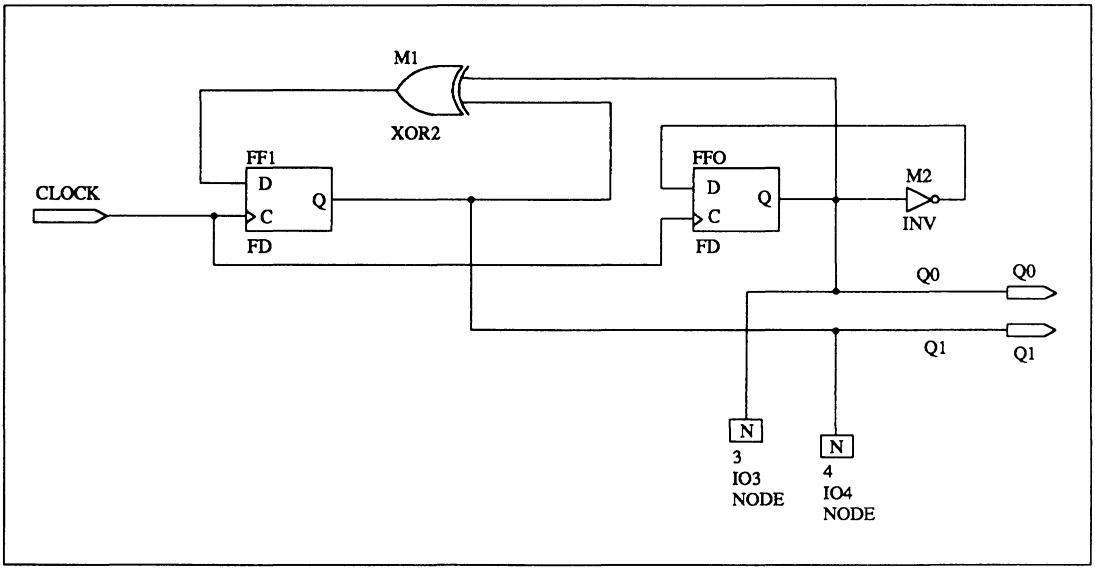
Recommendation: Judicious assignment steers the fitting process; random assignment obstructs the fitting process. For best fitting results, allow pins and nodes to float.[19]

For example, pin locations 3 and 4 are specified on NODE macros in the schematic shown next.

[17] An unresolved module port is one with no complementary connection.

[18] Refer to Section III, Chapter 7, for details about forming pins and fixing locations.

[19] Refer to Chapter 5, in this section, for details about fitting strategies and fixing pin locations.



The pin statements shown below appear in the PDS file after schematic data is converted.

```
  PIN   3   Q0          REGISTERED
  PIN   4   Q1          REGISTERED
  PIN   ?   CLOCK                            ;Input
```

4.5.2.2 Fixing Node Locations

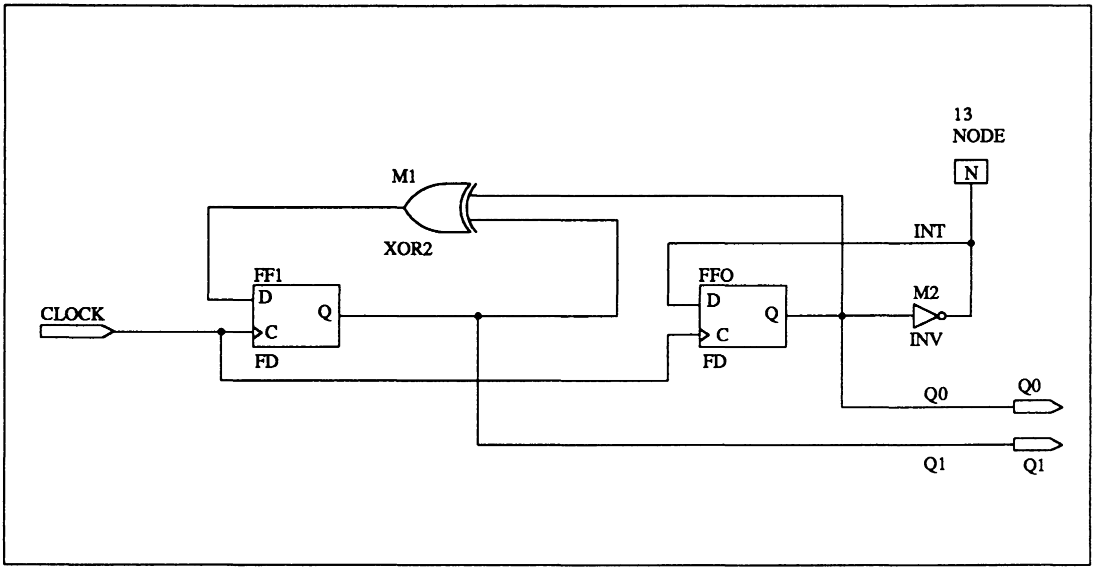
When you do not specify a fixed node location, it is left floating and the software determines its location on the device. In a schematic, you can specify device node locations as follows.

- Attach a NODE macro to a net.
- Edit part field 1 of the NODE macro.

After converting schematic data to equations, specified locations appear in the location-number field of the pin statement in the declaration segment of the PDS file.

Recommendation: Judicious assignment steers the fitting process; random assignment obstructs the fitting process. For best fitting results, allow pins and nodes to float.[20]

For example, node location 13 is specified on the NODE macro in the schematic below.



The pin and node statements below appear in the PDS file after schematic data is converted.

```
PIN    ?   Q0        REGISTERED
PIN    ?   Q1        REGISTERED
NODE   13  INT       COMBINATORIAL
PIN    ?   CLOCK                         ;Input
```

[20] Refer to Chapter 5, in this section, for details about fitting strategies and fixing pin locations.

4.5.2.3 Assigning Logic to a Block

To assign logic in a schematic to a specific block in a MACH device, you can edit part field 2 of the following macro symbols.

- NODE macro
- Flip-flops

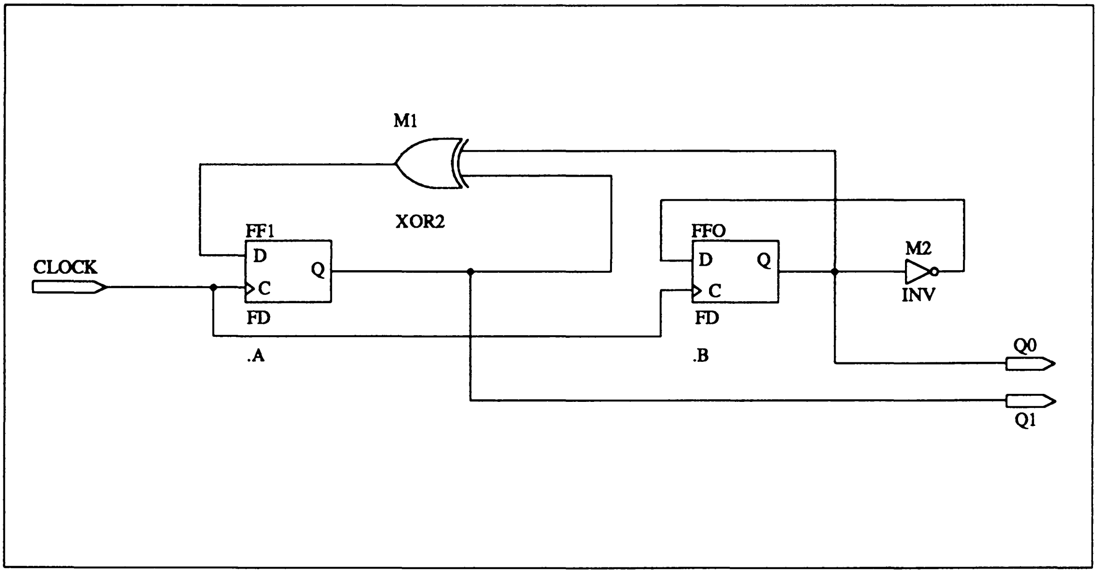
Grouping logic with common inputs and feedback assists during the fitting process.[21]

When schematic data that includes a specification in part field 2 is converted, a group statement is produced in the resulting PDS file. The statement includes the reserved word MACH_SEG_*block*[22] as the group name.

For example, the following schematic shows two flip-flops assigned to two different blocks in the MACH device, .A and .B, through specifications in the second part field.

[21] Refer to Chapter 5, in this section, for details about fitting strategies and logic-block assignment. Refer to Section III, Chapter 7, for details about specifying block locations in a schematic.

[22] Refer to Section IV, Chapter 10, for details about MACH_SEG_*block*.



The following group statements appear in the PDS file after schematic data is converted.

```
GROUP MACH_SEG_A Q1 CLOCK
GROUP MACH_SEG_B Q0 CLOCK
```

4.5.2.4 Controlling Minimization

Design logic is automatically minimized during the compilation process. However, you can suppress the minimization process in one of two ways.

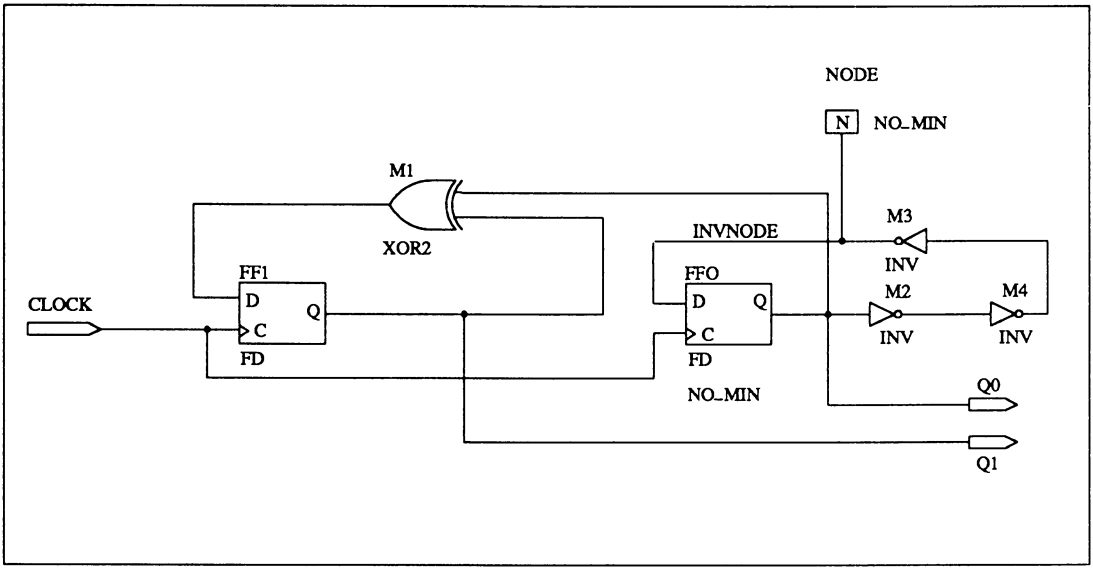
- Cancel minimization for the entire design: specify a manual run mode and no minimization on the Compile options form.[23]
- Cancel minimization discriminately at the node level in a schematic by specifying No_Min in part field 3 of either a NODE or storage-device macro.[24]

[23] Refer to Section IV, Chapter 9, for details about the Setup command on the File menu and the Compilation options command on the submenu.

[24] Refer to Section III , Chapter 7, for details.

The information in part field 3 is translated into equations in the resulting PDS file. No specification in part field 3 means minimization is on.

For example, the following schematic shows No_Min specified in part field 3 of a NODE macro and a flip-flop.



The statements shown next appear in the equations segment of the PDS file after the schematic data shown above is converted.

```
    MINIMIZE_OFF
/INVNODE = Q0
    MINIMIZE_ON
Q1 = Q0 :+: Q1
Q1.clkf = CLOCK

Q1.setf = GND
Q1.rstf = GND
    MINIMIZE_OFF
Q0 = INVNODE
Q0.clkf = CLOCK

Q0.setf = GND
Q0.rstf = GND
    MINIMIZE_ON
```

4.5.2.5 Controlling Set/Reset

MACH devices feature independent set/reset-control signals for each block in the device. Storage devices within each block share common set/reset lines. As a result, each time you specify a new set/reset signal in a schematic, the logic connected to that signal is placed in a new block.

The MACH library provides an **AINIT macro** that allows you to easily specify the first pair of set/reset signals. All three-terminal storage macros are controlled implicitly by the AINIT macro; they do not contain explicit set and reset pins.[25] You use a single AINIT macro to specify common set/reset-control signals for all three-terminal storage macros in the design.

To specify **additional set/reset signals**, you must use **five-terminal storage-device macros** that contain explicit set and reset pins. To specify a common set or reset line for these macros, you must connect the set or reset pins to a common source using a wire.

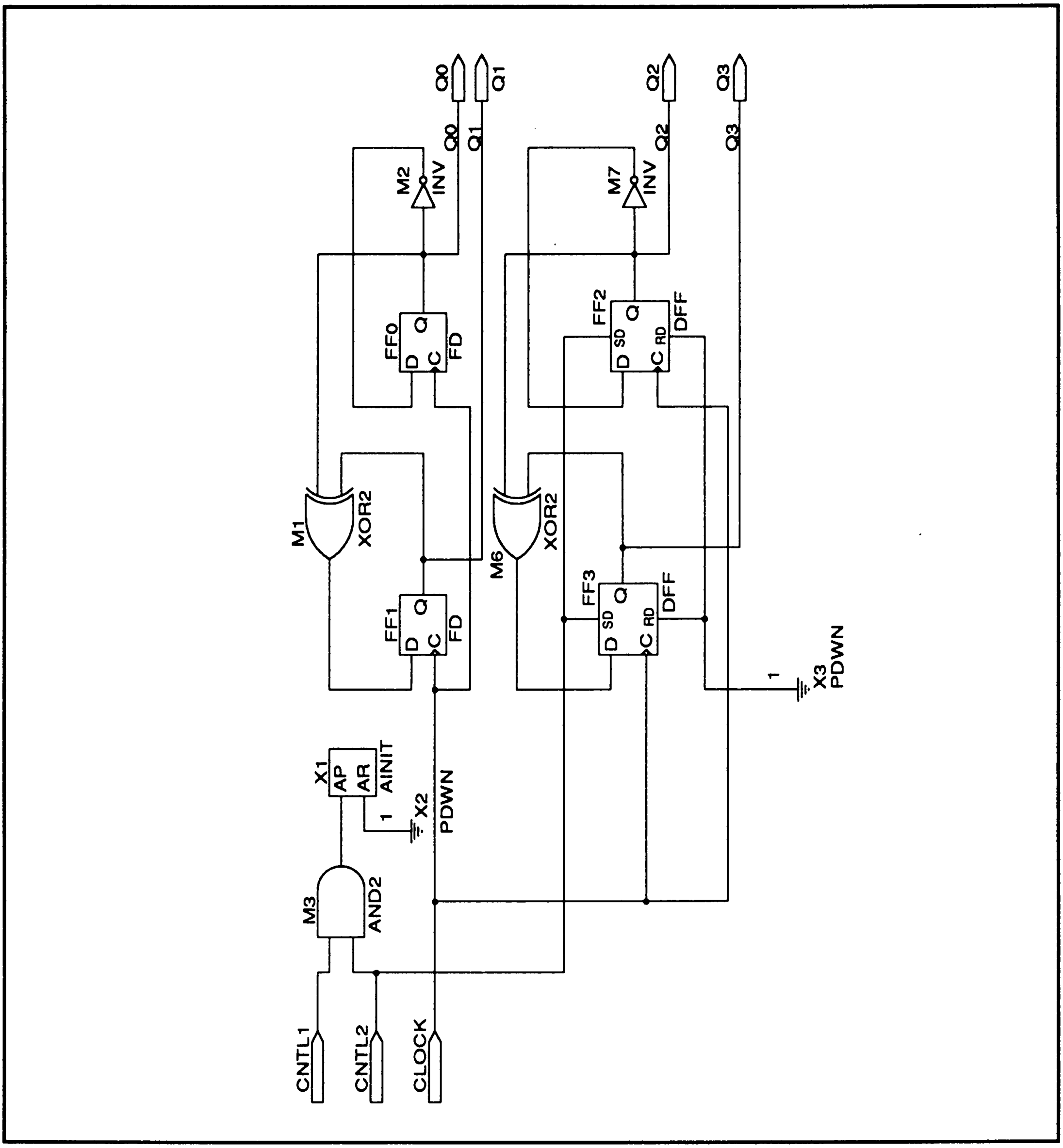
[25] Refer to Section III , Chapter 7, for details about using the AINIT macro.

Tip: It's much easier to place and wire the AINIT macro than to explicitly draw and connect set/reset lines for each storage device. For this reason, you should use the three-terminal storage macros, along with the AINIT macro, for the largest group of storage devices that share set/reset signals.

An example schematic is shown on the next page. The group of two FD macros are set by the product of control the signals CNTL1 and CNTL2 using the AINIT macro. The group of two DFF macros is set by the control signal CNTL2. When the design is compiled, the two groups are placed in separate blocks in the MACH device because they have different set signals.

Caution: Be sure not to specify more set or reset signals than the device supports.[26]

[26] Refer to Chapter 5, in this section, for details about set/reset signals.



The following statements appear in the equations segment of the PDS file after the schematic data is converted.

```
Q1 = Q0 :+: Q1
Q1.clkf = CLOCK
Q1.setf = CNTL1 * CNTL2
Q1.rstf = GND

Q0 = Q0
Q01.clkf = CLOCK
Q0.setf  = CNTL1 * CNTL2
Q0.rstf  = GND

Q3 = Q2 :+: Q3
Q3.clkf = CLOCK
Q3.setf = CNTL2
Q3.rstf = GND

Q2 = Q2
Q2.clkf = CLOCK
Q2.setf = CNTL2
Q2.rstf = GND
```

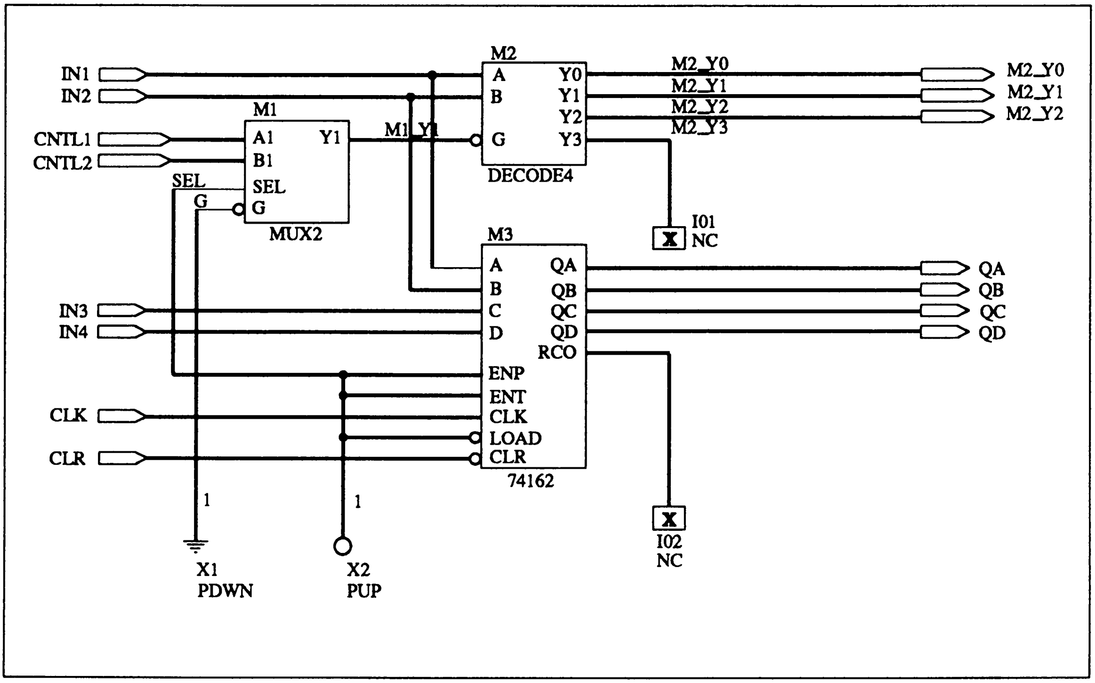
4.5.2.6 Deleting Unused Logic

You can identify unused portions of a macro for removal. For example, you may decide not to use the preload and ripple carry-out signals of the 74162 macro. In this case, you must identify the pins and logic associated with these functions so they can be deleted automatically during compilation.

To delete unused logic, follow the rules below.

- Connect unused **output pins** to an NC macro.
- Connect unused **input pins** to either a pull-up, PUP, or pull-down, PDWN, macro.

The next schematic shows a 74162 macro, DECODE4, and MUX2 with unused logic flagged for deletion.



4.5.3 DESIGN DOCUMENTATION

The software produces a device pinout drawing in the MACH report for each compiled design. You can use this drawing as part of the design's documentation. The information in the file is in ASCII format, which you can print in two ways.

- Use the Other files command on the Edit menu, enter the name of the file, design.rpt, and use the print command in the text editor.

or

- Print the design.rpt file from the operating system using the standard DOS print command.

Also, as stated earlier, a schematic graphically presents information about the logical content of MACH-device designs. There are two methods to print a schematic.

- Use the Hardcopy command in OrCAD/SDT III. Make sure your printer is specified correctly in the OrCAD setup file before printing.[27]

or

- Convert the schematic to a standard interchange format using OrCAD's Plotall command, which produces one file for each sheet in a hierarchical design.[28]

  OrCAD supplies drivers to convert a schematic to Postscript, DXF, and many other formats. You can send the resulting files to a plotter or transfer them to other documentation systems. In this case, you complete the steps below.

  1. Ensure your plotter driver is correctly specified in the OrCAD DRAFT/C program.

  2. Enter the operating system and set the current working directory to the one that contains the schematic files you'll convert.

     The top-level schematic name corresponds to the file name assigned when the schematic was created.

  3. Type the command below from the operating system to start the conversion.

     ```
     PLOTALL name.SCH  outfile  /S  .x
     ```

     Name.SCH is the name of the top-level schematic; outfile specifies the name of the converted file. If you do not specify an output

[27] Refer to Section IV, Chapter 9, for details about entering OrCAD/SDT III using the Schematic file command on the Edit menu, and using the Execute command from the Run menu, to change OrCAD configuration data.

[28] Refer to the OrCAD manual for details about printing and plotting.

file name, the information is sent directly to the plotter.

The /S .x parameter is optional and specifies a scale factor for the plot. For example, /S .7 scales the plot by 70%.

4.5.4 CONVERSION, EXISTING SCHEMATICS TO MACH-DEVICE DESIGNS

When you have an existing schematic-based design created in OrCAD/SDT, using a generic TTL or logic library, you may be able to convert it to a MACH-device schematic without re-entering the design. However, each symbol in the existing design must correspond to a symbol with the same name in the AMD-supplied MACH library.

To convert an existing OrCAD schematic to a MACH-device design, you use the procedure below.

1. Enter OrCAD DRAFT with the existing design, as usual.

2. Delete any symbols that don't have exact equivalents in the MACH library and save the design.

   Deleted symbols can be replaced with logic from the MACH library after the design is converted.

3. Run the PALASM 4 software.

4. Change the directory, if necessary, and retrieve the existing schematic design.

   > **Important:** Be sure to specify a schematic design and use the name of the root-level schematic.

A schematic control file form appears automatically. After you complete the form and enter OrCAD/SDT from the PALASM environment, the existing schematic is read. Each symbol in the design is automatically replaced with the corresponding symbol from the MACH library.

5. Edit the schematic to add logic from the AMD-supplied MACH library.

> **Important**: Symbols with identical names do not always have the exact same functionality.
>
> The converted design should be carefully simulated to ensure the desired functionality was preserved.

4.6. COMBINING SCHEMATIC AND TEXT DESCRIPTIONS

If you choose this method, you begin the MACH-device design by entering logic in a schematic, then complete the design by specifying parameters, such as block assignment, using text entry. The procedure is identified below.[29]

1. Begin a new schematic-based design from within the PALASM environment, as usual.

   To use the AMD-supplied MACH library for OrCAD/SDT III, you must begin the schematic from within the PALASM environment.

2. Generate a PDS file from the schematic data using one of the two methods below.

   - Compile the schematic to produce the PDS file.

   or

   - Select Other operations from the Run menu, then select the Convert schematic to text command from the submenu.

3. Retrieve the text-based version of the design.

   > **Important**: If you do not change the input format to text, the PDS file is overwritten the next time you compile.

4. Edit the PDS version to specify additional parameters.

Changes you make to the PDS file do not appear in the schematic. This means the schematic may not accurately document the completed design. If you edit and recompile the schematic, any changes you made to the PDS file are overwritten with recompiled schematic data.

[29] Refer to Section IV, Chapter 9, for details about specific commands and forms.

Important: Once you begin editing the PDS file, do not change the schematic and recompile. **Do not** switch from text entry back to schematic entry.

5. Compile the text-based PDS file as usual, then download the design to a device programmer.

Important: Specify text as the input format when you retrieve the PDS file.

4.7 MERGING MULTIPLE PDS FILES

You can combine multiple PDS files into a single MACH-device design. You can use any valid PDS file. Topics here address requirements and guidelines.

- 4.7.1, Input Files
- 4.7.2, Design Evaluation
- 4.7.3, Guidelines

4.7.1 INPUT FILES

You can merge information from any of the following PDS files to produce a single MACH-device design.

- Existing PDS files for any PAL device
- New PDS files for any PLD design
- Converted schematic descriptions for a MACH-device design

Important: Designs described in other languages must be converted to PALASM syntax before merging.

Also: Only Boolean descriptions are accepted. State-machine descriptions must be expanded and minimized to Boolean form before files can be combined. The software automatically converts state-machine language to Boolean equations during compilation. However, it may be difficult to relate the names of resulting Boolean equations to the original state-machine description.

The input files are not changed. Merged information is stored in a single new output file. Simulation segments are removed during the process; these commands can be added to the combined file later or stored in a separate simulation file.

Many factors affect whether or not the combined design is suitable for a MACH device. These are discussed next.

4.7.2 DESIGN EVALUATION

Before you choose existing PLD files for a combined design, it's a good idea to consider the following.

- Compatibility
- Inputs, Clock Signals, and Set/Reset Control

4.7.2.1 Compatibility

To determine which designs to integrate into a single MACH-device design, you must evaluate the following.

- Characteristics of each design
- Interaction between designs
- Device architecture
- Logic complexity

Speed is an important **characteristic**. Speed grades for some simple PLDs are faster than speed grades for MACH devices. If integration allows you to eliminate critical delays getting on and off the chip, it's possible to meet your speed requirements using a MACH device.

Integration of several devices makes the most sense if the combined design requires fewer pins than the gross of both designs. Two situations can lead to this result and are outlined below.

- The designs you plan to merge have many input signals in common.

- The internal logic variables of the designs and the fanout to other devices will be integrated.

Even designs with few common inputs and feedback may result in substantial board-space savings and lend themselves well to a MACH-device design.

Existing PLD designs that are **well suited** to MACH-device designs include those for the following **architectures**.

- 16R/4, 16R/6, and 16R/8
- 22V10
- 16V8

XOR gates, wide OR structures, and independent flip-flop controls are not easy to implement in the MACH device. For this reason, the MACH architecture is **not** particularly well **suited** to designs for the following architectures.

- XPAL
- PSL
- RA-PAL

Complexity is an important consideration. It's important to analyze potential designs to determine if the combined design can fit in the chosen MACH device. You do this by determining the number of pins, logic equations, macrocells, and internal logic variables in the combined design. Then compare the data against available resources for the chosen MACH device.[30]

An easy way to determine the resources required by the combined design is to combine them, compile and fit the combined design, and review the MACH report. The report provides statistics on all design resources.

[30] Refer to Chapter 5, in this section, and Section III, Chapter 8, for additional information.

4.7.2.2 Inputs, Clock Signals, and Set/Reset Control

Other considerations for design suitability should also be evaluated.

Each block in the MACH device has 22 **inputs**. Designs taken from larger devices, such as the 26V12, a PLS device, or a 29M16, may have equations that use more than 22 inputs. In the MACH-device design, these additional inputs must be rewritten as several smaller equations and implemented using multiple passes through the array. Additional passes through the array may create undesirable timing delays.

The MACH architecture also allows two different **clock sources**. The combined design must not require more than two clocks. The MACH device does not allow product-term developed clocks or inversion along the clock path.

Independent set and reset controls are provided for each bank of flip-flops in a MACH device. Equations with different set or reset functions are placed in different banks. If the number of different set or reset functions exceeds the number of banks, the design will not fit in the MACH device.

4.7.3 GUIDELINES

After you evaluate the designs to determine whether or not it is feasible to merge them into a single MACH device, you begin the merge process. The merge process is divided into two stages: one for the first file and one for subsequent files. Each stage involves several major tasks and minor activities, which are outlined below.

- **Set Up**
  Initiate the process
  Specify setup options

- **Retrieve Files**
  First File
  Subsequent Files

- **Resolve Conflicts**
  Review the detectable conflicts table
  Rename signals in the input buffer
  Bind signals together

- **Merge Files**
  Edit combined data
  Save combined in a file

- **Re-engineer the Combined Design**

The following discussions identify the tasks and the order of events, and provide some facts about each activity.[31]

4.7.3.1 Set up

To begin the merge process, you initiate the process and set up the environment.

Initiate the Process

To initiate the process,

1. Change the current working directory within the PALASM environment, as usual.

   The directory you choose designates where the combined output file will be stored.

2. Use the Merge design files command on the File menu to initiate the process, then identify the name of the output file that will include all combined data.

If the output file name you supply matches an existing name, you're asked if you want to use the data in the existing file.[32] You can type the letter Y to use the existing data or the letter N to destroy the existing data.

[31] Refer to Section IV, Chapter 9, for details about specific commands and forms.

[32] If you use existing data, it is moved into the output buffer automatically so you can continue the merge process and add other designs to the existing file.

Specify Setup Options

After specifying the file name, several new menus and a status screen appear.

1. Use the Options command on the Setup menu to display a form where you can specify pin sort order, floating pins on input, and reuse of input files.

> **Recommendatlon**: Float pins on input to assign a question mark, ?, in the location field of all pin and node statements in the output file. This resolves pin-number conflicts and allows locations to be determined automatically during compilation and fitting.

2. Use the Set renaming strategy command on the Setup menu to specify a substitute naming algorithm to use when a signal in the input buffer conflicts with a signal in the output buffer.

   The default, $_#, adds an underscore and a three-digit number to the signal name in the input buffer: _00n, for example.

Once you've set up the environment, you can begin combining files.

4.7.3.2 Retrieve Files

The merge procedure temporarily stores input and output data in separate memory buffers. Initially the input buffer is clear. The output buffer contains only a PDS-file shell, which contains empty declaration and equation segments.[33]

Note: The declaration segment contains only a few keywords. Header information for the combined design is taken from the first input file.

The output file name appears on the screen under the Files menu; the input file name area is blank.

First File

To get the first file,

1. Use the Get next input file command on the Files menu to retrieve a design.

   The file is automatically parsed, expanded, and minimized.

   - If errors are found in the input file, you must quit and correct them before you can merge the file.

   - If no errors are found, the file is placed in the input buffer.

2. Review the pin summary data on the screen.

The pin summary table provides the information shown below.

[33] If you started the merge process using data from an existing file, the output buffer contains design information from that file. In this case, skip to the discussion on subsequent files.

Pin Summary Table

Fields		Definitions
Pins	8	Number of pins defined in the input and output files
Nodes	12	Number of nodes defined in the input and output files
Floating	60	Number of floating pins and nodes in the input and output files
Unreferenced	15	Number of named pins not used in equations

You can use the Abandon input file command on the Files menu to clear the input buffer if the file is inappropriate, then get a different input file.

3. Use the Merge files command on the Files menu to combine the input file with the output-file shell.

   The input file is moved into the output buffer; the input buffer is now empty. The status line in the center of the screen notes that one file has been merged into the output buffer.

You can view the information in the output buffer at any time using the View output buffer command on the Editor menu. To return to the merge screen, just press [Esc].

4. Use the Save command on the Files menu to write the information in the output buffer to a file, then continue merging subsequent files.

   > **Recommendation**: It's a good idea to save the output file before getting the next input file.

Subsequent Files

This is a repeat of earlier steps.

1. Get the next input file.

   This file is placed in the clear input buffer and compared with existing data in the output buffer. Status tables change appropriately.

2. Review the pin summary table to determine if the design fits in the specified device.

   - If the design is suitable, begin resolving conflicts.

   - If the design is not suitable, you can **either** abandon the input file to clear the buffer **or** edit the pin/node list in the output file to change device the type.

4.7.3.3 Resolve Conflicts

Conflicts fall into two categories: detectable and undetectable. Detectable conflicts occur when signals in the input and output buffer have the same name or pin number. These can be resolved from the conflict resolution form.

- If the intent is to use separate signals, you must rename one.

- If the intent is to use the same signal, you must bind them together.

The default action is to rename the signal in the input buffer.

When you specify no floating pins on input as a setup option **and** two pins are assigned to the same pin location on the device, the word Wildcard appears in the Action list. In this case, a question mark is automatically assigned to the pin location in the input buffer.[34] To restore the pin location specified in the input buffer, you must edit the pin/node list in the output buffer after combining the files.

Undetectable, resolvable conflicts occur when two signals with different names refer to the same signal. You can bind these signals together to resolve the conflict. Conflicts with other named elements, such as state, group, and string names do not occur because the input file was expanded and minimized. Hence, these potential conflicts are no longer present.

Review Detectable Conflicts Table

To determine which detectable conflicts exist,

1. Review the detectable-conflicts table.

   The table identifies the number of pin/node conflicts detected when data in the input and output buffers were compared.

2. Use the Resolve detectable conflicts command on the Resolution menu to specify changes to the input buffer.

A conflict resolution form appears listing all signals with detectable conflicts. Each row identifies a single conflict and includes the information shown next

[34] Refer to Chapter 5, in this section, and Section IV, Chapter 10, for additional details about floating pin locations.

Conflict Resolution Table

FIELDS	DEFINITIONS
Output File	Signal name in the output file
Input File	Signal name in the input file
Action	The action to resolve the conflict: Rename input, Bind, or Wildcard
Substitute	The name that's assigned automatically to the signal in the input buffer.

At this point, you take action as follows; discussions below provide guidelines for each task.

- Rename signals in the input buffer
- Bind signals together
- Combine files when all conflicts are resolved
- Edit the pin/node list in the output buffer
- Save the output file
- Edit the combined design to re-engineer it for the MACH device

Important: The input file is not changed, only data in the input buffer is altered. Changes do not become part of the output file until you combine files and save the data in the output buffer to a file.

Rename Signals in the Input Buffer

It's best to handle wildcard actions first. For example, you have two designs with the following pin statements and you specify no to floating pins on input in the setup-options form.

```
PIN 6   CLOCK          ;design 1
PIN 6   CLK            ;design 2
```

A pin number conflict is detected and Wildcard appears in the Action column of the conflict resolution form.

To recover from a wildcard action, you complete one of two procedures.

1. Resolve other conflicts on the resolution form, press [F10] to accept the changes and return to the status screen, merge and save the design, then edit the pin/node list in the output buffer as described later.

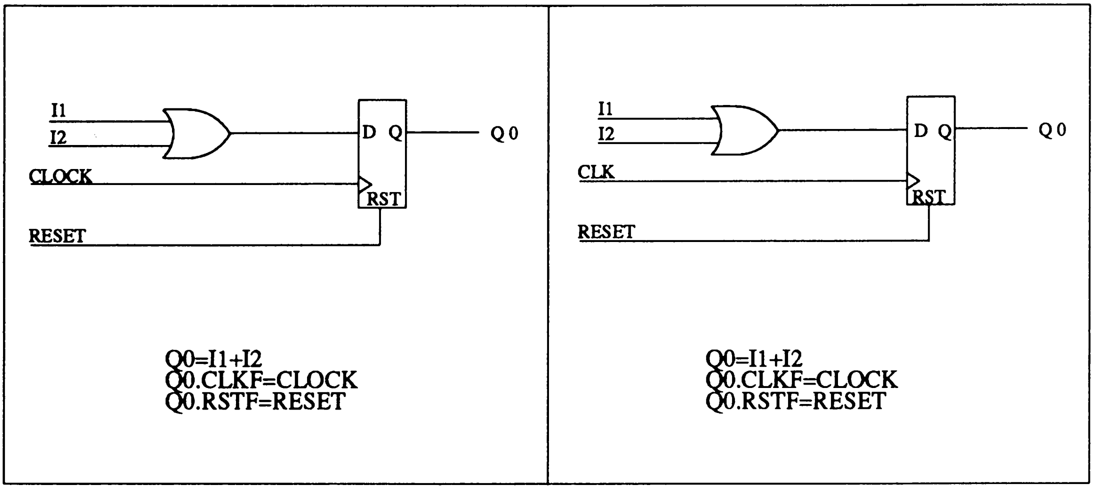
or

2. Leave the conflict resolution form, abandon the input file, specify floating pins on input as a setup option, and retrieve the file again.

After resolving wildcard actions, you may find the same name is used in both the input and output designs, though it references different signals. When this conflict is detected, one of the signals must be renamed.

When a signal is renamed, a substitute is provided automatically to replace every occurrence of the original in the input buffer. Default substitute names include the original signal name and an extension that's defined using the Set renaming strategy command on the Setup menu. You can either accept or change the substitute name in the conflict resolution form.

For example, suppose you have two designs that each contain a single equation with registered output, as illustrated below.



A conflict is detected for each set of signals with the same name: I1, I2, Q0, and RESET. In this case, I1, I2, and Q0 represent different signals in each design. The desired resolution is to rename the signals in the input buffer as I3, I4, and Q1; the substitute name that's assigned automatically is not appropriate.

To change the substitute that's used to rename a signal,

1. Select the Substitute field and press the [Tab] key.
2. Type a new name to distinguish it from the signal in the output buffer and press [Enter].

   The name must be unique; it cannot match an existing name without causing a new conflict.

Changes aren't reflected in the status at the bottom of the screen until you leave the field and return to it. Each occurrence of the name is replaced in the input buffer.

When all signals are renamed appropriately, you can **either** bind signals as described in the next discussion,

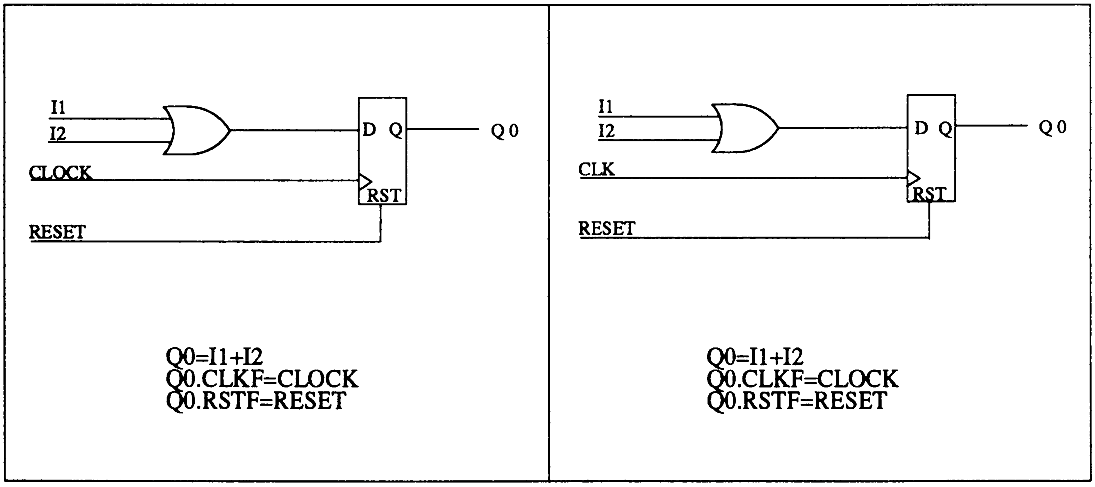
or complete step 3 to return to the detectable conflicts table.

3. Confirm the changes and return to the tables: press [F10].

Bind Signals Together

When one name is used in both the input and output designs to reference the same signal, a conflict is detected. In this case, the problem signals are listed on the conflict-resolution form. However, when two signals of different names refer to the same signal, no conflict is detected and you must bind these signals manually using the Bind command to display the appropriate form.

For example, suppose you have two designs that each contain a single equation with registered output as shown next. You want to use the same control signals to clock and reset both equations.



In this case, respective signals must be bound using a common name. Reset appears in both designs so it is listed on the conflict-resolution screen and a substitute name is provided automatically. The appropriate action is to bind rather than rename the signal in the input buffer.

However, the clock signals in each design are currently named differently. In this case, no conflict is detected so you use the Bind command on the Resolution menu to display the bind form. Then you can select from a list of signals in each design.

To bind signals with the same name you must use the conflict-resolution form, as follows.

1. Use the Resolve detectable conflicts command on the Resolution menu to specify changes to the input buffer.

2. Activate the Action field on the conflict-resolution form that corresponds with the appropriate signal and display the list of options: press [Tab] then press [F2].

3. Select Bind from the list and press [Enter].

   Changes are reflected in messages at the bottom of the screen only after you leave the field and return to it.

When all appropriate signals are bound,

4. Save changes to the conflict-resolution form: press [F10].

   A message indicates the number of signals that will be moved to the bind form.

When you leave the conflict-resolution form, bound signals are moved to the bind form.

To undo the bind operation after leaving the conflict-resolution form,

1. Select Bind pins/nodes from the Resolution menu.

2. Select the Action field for the appropriate signal and display the options.

3. Select No action from the list.

4. Press [F10] to leave the bind form when you have finished.

To bind signals with different names, you must use the Bind form as described below.

1. Select Bind pins/nodes from the Resolution menu.

   The bind form appears listing all signals that were bound from the conflict-resolution form.

2. Select an empty Output File field and display the options.

   A list of all signals in the design is displayed.

3. Choose the signal in the output file, CLK, for example.

4. Activate the Input File field, display the options, and select the name, CLOCK, for example.

   The two signals are bound together and the name from the output buffer is used. Each occurrence of the name is replaced in the input buffer.

5. Save changes to the bind form and check the conflict form for new conflicts: press [F10].

After all conflicts are resolved, you can merge the files as discussed next.

4.7.3.4 Merge Files

You merge the files only after all conflicts are resolved. Then you can edit the pin/node list if needed.

1. Verify all conflicts have been resolved appropriately.

Important: Once begun, the merge process cannot be stopped. If you must reverse this operation, you must quit and restart as follows.

- Select Quit from the Files menu; do not save the current output file.

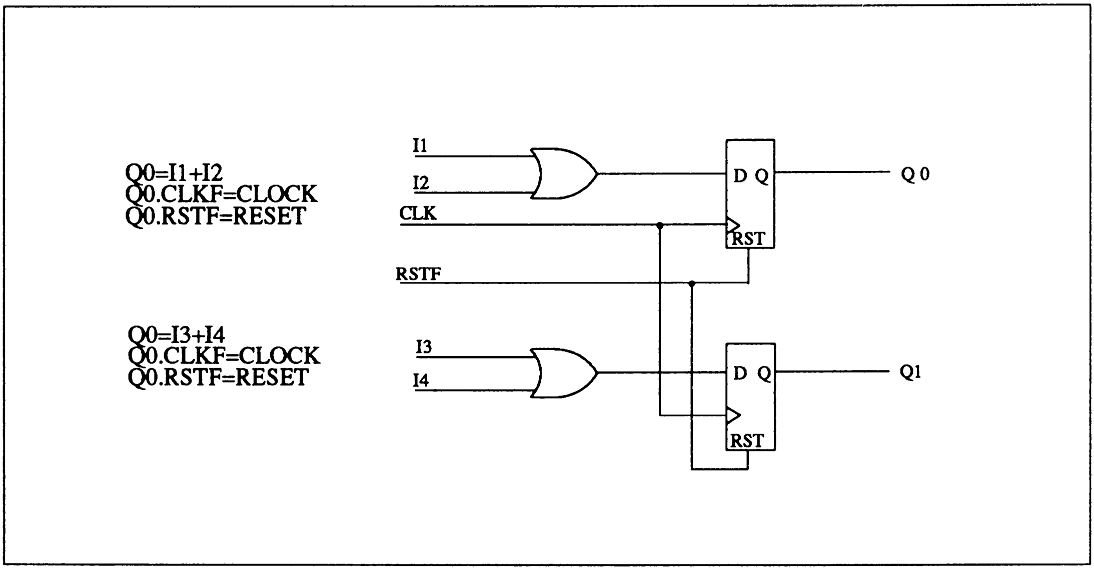
  You lose all changes to the input buffer.

- Select the Merge design files command from the File menu to initiate the operation again, then specify a new output file name.

- Get and combine the previous session's output file into the new output-file shell and continue.

2. Merge the files and save the output.

You can edit the pin/node list. The resolved, merged design discussed earlier is shown next.



```
Q0=I1+I2
Q0.CLKF=CLOCK
Q0.RSTF=RESET
```

```
Q0=I3+I4
Q0.CLKF=CLOCK
Q0.RSTF=RESET
```

Edit Combined Data

After merging files, you can edit the combined data to change header information, the device type, or the name or location of a signal.

To edit header information,

1. Use the Edit pin/node list command on the Resolution menu.

   Data in the output buffer becomes available and includes the header and pin/node list from all designs combined thus far.

Data appears in a form, like the declaration segment form for new PDS files. The top of the form includes seven header fields. The information here comes from the first input file.

2. Select the appropriate field and type text as usual, then press [Enter] to select the next field.

If you are finished editing, complete step 3. Otherwise, continue with the next procedure.

3. Save data in the buffer when all editing is complete: press [F10].

To change the device type,

1. Activate the Device field and display a list of options.

2. Select the MACH device you want to use.

If you are finished editing, complete step 3. Otherwise, continue with the next procedure.

3. Save data in the buffer when all editing is complete: press [F10].

The pin node list at the bottom of the screen can be scrolled and includes all combined data in the output buffer.

To change a pin name or number,

1. Select the appropriate pin or node statement.

2. Enter information using appropriate syntax for each field as you would in the declaration segment form.

3. Save data in the buffer when all editing is complete: press [F10].

You can re-enter the buffer to create additional pin/node fields. At least twenty empty fields become available at the end of the pin/node list each time you enter this buffer.

4. Save data in the buffer when all editing is complete: press [F10].

Save Combined Data

When the files are merged,

1. Save the output to disk.

2. Get the next input file and repeat the entire process until all files are combined.

3. Quit when you're finished.

4. Edit the combined design using a text editor to re-engineer it for a MACH device as discussed next.

4.7.3.5 Re-engineer the Combined Design

After you combine all files for a single MACH-device design, you need to edit the resulting file to add the following information.

- Shared resource information
- Group statements with MACH block names
- Simulation commands

Shared Resources

Shared resources in MACH-device designs include clock signals, reset functions, and three-state enables. Some PAL devices do not have shared resources.

To take advantage of shared resources for a MACH-device design, you need to add appropriate statements to the combined PDS file.[35]

- A CLKF statement is required for registered outputs in a MACH-device design.
- SETF, RSTF, and TRST statements are optional.
- Global statements can be used to control the entire device
- Group statements control multiple signals within a single MACH block

Group Statements with MACH Block Names

Several **group names** are reserved for MACH-device designs so you can assign signals to specific blocks in the device. You just add the appropriate group statements to the merged PDS file.[36] For example,

```
GROUP  MACH_SEG_A   R[0]  R[1]  R[2]
GROUP  MACH_SEG_B   O[1]  O[2]  O[3]
```

Simulation Commands

You must create **simulation** commands for the combined design. These can be stored in the combined PDS file or in a separate simulation file.

[35] Refer to Chapter 5, in this section, and to Section IV, Chapter 10, for more information about MACH-device designs and for details about generic PALASM language constructs and syntax.

[36] Refer to Section IV, Chapter 10, for more information about MACH_SEG group names and for details about the language constructs and syntax of simulation commands.

CHAPTER 5

COMPILATION / FITTING

CONTENTS

5 COMPILATION / FITTING

This reference-style chapter is divided into five major topics.

- The overview, 5.1, introduces the fitting process for MACH-device designs, which replaces the assembly process performed on other PLD designs.
- The process discussion, 5.2, identifies the three stages of fitting a design and explains what occurs in each phase.
- The discussion on designing to fit, 5.3, explores how to design so there is good prospect of fitting the design during the first compilation.
- The discussion on designs that don't fit, 5.4, describes how to interpret error messages (and recover from an error), interpret the MACH report, and explores strategies to change a design that doesn't fit initially.
- The discussion on changes after a successful fit, 5.5, describes alterations to logic and pin out that do not affect fitting results.

5.1 OVERVIEW

The last phase of the compilation process for MACH-device designs is the fitting process. During fitting, the design is mapped to the physical resources of the specified MACH device.

The goal of the fitting process is to determine pin/node placements and routing that satisfy design requirements.

Although the fitting process is automatic, there are many things you can do that affect the likelihood of a successful fit.

- Assign or allow floating pin and node locations.
- Fix pin and node locations judiciously.
- Assign logic to specific blocks in a MACH device.
- Change the architecture of logic in your design.
- Change the amount of logic in your design.
- Change compilation, logic synthesis, and MACH fitting options.

5.2. THE FITTING PROCESS

The fitting process consists of three stages. An understanding of each phase can help you choose the best corrective action if the design does not fit.

- Initialization
- Block partitioning
- Resource assignment

5.2.1 INITIALIZATION

Two files are read by the MACH Fitter during the initialization phase.

- design.TRE is produced during compilation; it contains the target device type, signal information from pin and node statements, and the design description encoded in Boolean sum-of-products form.

- design.PLC contains data generated during the last successful fitting process, including pin and node placement information that reflects the compilation and MACH fitting options you've specified. This file is only used when you select the last successful placement option.

After reading the files, information about the internal architecture of the specified device is loaded and resource checks are performed on the design. Errors are reported if the design exceeds the available product term, macrocell, pin, or clock resources.

5.2.2 BLOCK PARTITIONING

After initialization, the design is segmented to be fit into individual blocks of the specified MACH device. Segmentation is accomplished by assigning affinity measures to the logic equations. Equations that share common inputs have strong affinities and are grouped together in blocks.

Proper block partitioning is critical for a successful fit. You can manually control this process by preplacing portions of the logic in specific blocks using the reserved word, MACH_SEG_*block*, as a name in a Group statement.[1]

5.2.3 RESOURCE ASSIGNMENT

In the final phase of the fitting process, individual equations are assigned to physical resources. This is done sequentially for each block in the device, as follows.

- Logic equations associated with each pin are assigned first.
- Buried logic functions are placed in the remaining unused macrocells.
- Inputs are assigned to any available pads last.

  These pads may be **either** dedicated inputs **or** from macrocells that are unused or that were used for buried logic functions.

Important: At each point during resource assignment, connection resources are marked as used, which affects later resource assignments. For this reason, the order in which design equations are processed can affect the outcome.

Recommendation: Signals with specified pin and node locations are processed first, which can block a connection path needed for another signal. It's a good idea to float all pins and nodes so the software automatically determines processing order and locations.

[1] Refer to discussion 5.3.4, in this chapter, for more information on block placement.

5.3 DESIGNING TO FIT

Decisions you make when entering the design **and** the logic synthesis, compilation, and fitting options you specify greatly impact the amount of logic that can fit in the device. Some of your decisions also affect design performance.

A clear understanding of the fitting process and the resources available in the MACH device can help you make sound decisions to achieve the density and performance you need.

The recommended methodology is to float all signals initially. With all signals floating, the software determines placements and has the greatest chance of achieving a successful fit. See the discussion on the AMD MACH Fitter in the ***PALASM 4 Release Notes*** that accompany your software for a complete discussion of these techniques. The ***MACH Technical Brief*** titled "MACH Design Planning Guide" that accompanies this software provides an excellent introduction to planning MACH designs. (The "MACH Design Planning Guide" is also published in the ***MACH Databook***. Both publications are available from AMD Literature.)

After finding a successful fit you can try modifying the placements to achieve a more desirable pin out. .

5.3.1 METHODOLOGY

See the The "MACH Design Planning Guide" found in the ***MACH Technical Briefs*** (also reprinted in the ***MACH Databook)*** for a complete discussion.

5.3.2 ANALYZE DEVICE RESOURCES

5.3.2.1 Clock Signals

MACH devices support multiple clock signals. For example, the MACH 110 and 210 devices support a maximum of two clock signals. Both clocks are available to every macrocell in the device. If you specify more than two clock signals in a design, the fitting process will fail and report the following error message.

```
CLK.CNT -1 Too many clocks in user design!
```

The device-resource check area of the fitting process report shows the number of clock signals used in the design. For example, a typical report may show the following.

```
            Available      Used      Remaining
Clocks:         2            1           1
```

In the example above, only one clock signal is used, though two were available. One remains in this case.

5.3.2.2 Set/Reset Signals

In the MACH device, all macrocells within a block share common set and reset signals, except for the MACH215. For example:

- The MACH 110 provides two blocks and supports two unique set or reset signals.
- The MACH 210 contains four blocks and supports up to four unique set or reset signals.

The specification of set and reset signals has an important effect on block partitioning during the fitting process. Logic with different set or reset signals must be placed in different blocks.

Tip: Whenever possible, specify common set and reset signals for logic that shares primary inputs or feedback. This allows the logic to be placed in a single block and increases the likelihood of a successful fit.

The use of **multiple** set or reset signals also impacts the availability of macrocell resources. For example, consider a design with 20 flip-flops where two share one reset signal and 18 share another. When this design is partitioned, two flip-flops are placed in one block, which leaves 18 flip-flops that must not be placed in the same block as the first two. Since the MACH 110 has only one other block with 16 macrocells, this design will not fit even though 30 macrocells remain in the device. To fit this design, you must **either** change the reset control **or** switch to a larger MACH device.

5.3.2.3 Macrocells and I/O Pins

MACH devices have a varying number of I/O pins with some configured as inputs, and some as output pins. Exceeding these limits results in an error.

For example, the MACH **110** contains six dedicated inputs and 32 macrocells. Each macrocell can be used as either an input or an output. The MACH **210** contains six dedicated inputs and 64 macrocells: 32 macrocells are output macrocells and 32 are buried macrocells. The 32 output macrocells can be used to bring signals on and off the device. The 32 buried macrocells are used to create internal signals.

Guideline: Do not use more than 28 macrocells in a MACH 110 design or more than 57 macrocells in a MACH 210 design. Exceeding these limits increases the difficulty of fitting. These guidelines also apply to the total number of flip-flops and latches.

5.3.2.4 Product Terms

MACH devices have a varying number of product terms. For example, the MACH 110 can support up to 128 product terms; the MACH 210 can support up to 256 product terms.

Guideline: Do not use more than 115 product terms in a MACH 110 design or more than 230 in a MACH 210 design.

Four product terms are available to each macrocell in the MACH device. Equations with product terms in mul-

5.3.2.1 Clock Signals

The MACH 110 and 210 devices support a maximum of two clock signals. Both clocks are available to every macrocell in the device. If you specify more than two clock signals in a design, the fitting process will fail and report the following error message.

```
CLK.CNT -1 Too many clocks in user design!
```

The device-resource check area of the fitting process report shows the number of clock signals used in the design. For example, a typical report may show the following.

```
                Available      Used     Remaining
Clocks:             2            1          1
```

In the example above, only one clock signal is used, though two were available. One remains in this case.

5.3.2.2 Set/Reset Signals

In the MACH device, all macrocells within a block share common set and reset signals.

- The MACH 110 provides two blocks and supports two unique set or reset signals.
- The MACH 210 contains four blocks and supports up to four unique set or reset signals.

The specification of set and reset signals has an important effect on block partitioning during the fitting process. Logic with different set or reset signals must be placed in different blocks.

> **Tip**: Whenever possible, specify common set and reset signals for logic that shares primary inputs or feedback. This allows the logic to be placed in a single block and increases the likelihood of a successful fit.

The use of **multiple** set or reset signals also impacts the availability of macrocell resources. For example, consider a design with 20 flip-flops where two share one reset signal and 18 share another. When this design is partitioned, two flip-flops are placed in one block, which leaves 18 flip-flops that must not be placed in the same block as the first two. Since the MACH 110 has only one other block with 16 macrocells, this design will not fit even though 30 macrocells remain in the device. To fit this design, you must **either** change the reset control **or** switch to a larger MACH device.

5.3.2.3 Macrocells and I/O Pins

Your design can have a maximum of 38 I/O signals with up to 32 configured as outputs and up to 38 configured as inputs. Exceeding these limits results in an error.

Guideline: Do not use more than 34 I/O pins for either a MACH 110 or a MACH 210 design.

The MACH **110** contains six dedicated inputs and 32 macrocells. Each macrocell can be used as either an input or an output. The MACH **210** contains six dedicated inputs and 64 macrocells: 32 macrocells are output macrocells and 32 are buried macrocells. The 32 output macrocells can be used to bring signals on and off the device. The 32 buried macrocells are used to create internal signals.

Guideline: Do not use more than 28 macrocells in a MACH 110 design or more than 57 macrocells in a MACH 210 design. Exceeding these limits increases the difficulty of fitting. These guidelines also apply to the total number of flip-flops and latches.

5.3.2.4 Product Terms

The MACH 110 can generate 128 product terms; the MACH 210 can generate 256 product terms.

Guideline: Do not use more than 115 product terms in a MACH 110 design or more than 230 in a MACH 210 design.

Four product terms are available to each macrocell in the MACH device. Equations with product terms in multiples of four make the most efficient use of device resources.

An equation with four product terms can be realized using one macrocell and one pass though the array, which results in the minimum propagation delay. Equations with more than four product terms are realized using product-term steering or gate splitting.[2]

Product-term steering uses resources from more than one macrocell but requires only one pass through the array. Equations with up to 12 product terms in the MACH 110 and 16 product terms in the MACH 210 can be implemented using this method.

If you enable the automatic gate-splitting option, equations containing more than the maximum number of product terms are implemented using gate splitting. This requires multiple passes through the array and results in increased propagation delay.

5.3.2.5 Interconnection Resources

Utilization of interconnection resources is one factor in the fitting process. This measure is calculated by the software and used in the MACH fitting process.

This internal measure of chip connectivity does not appear in the resource utilization table, but you can affect it indirectly using techniques described under discussions 5.3.3, 5.3.4, and 5.3.5. These techniques will improve the efficiency of fitting your design in a MACH device.

[2] Refer to Section IV, Chapter 11, for detailed information on steering product terms and splitting gates.

5.3.3 ASSIGNING PIN AND NODE LOCATIONS

Arbitrary preplacement of pins and nodes is likely to result in a no-fit situation and is not recommended. There are techniques you can use to improve the pin out once a successful fit is accomplished with all pins and nodes floating. These techniques are described under discussion 5.5.1.

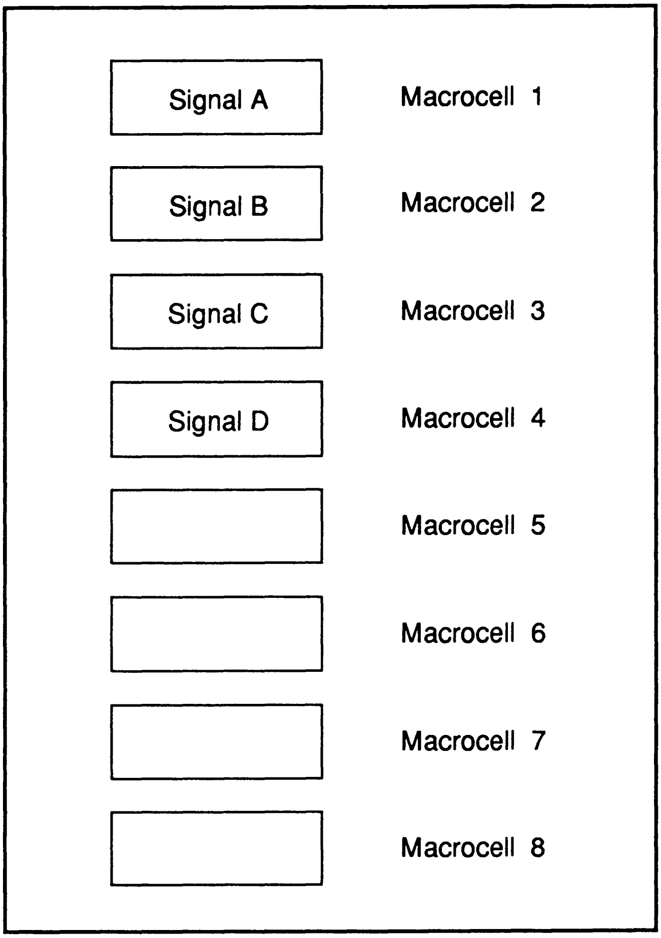
If you must fix locations to satisfy PCB routing or other requirements you should use the following guidelines.

Build up the preplacement list gradually starting with only one or two signals; leave the rest floating. Place large logic functions first then smaller ones; place inputs last.

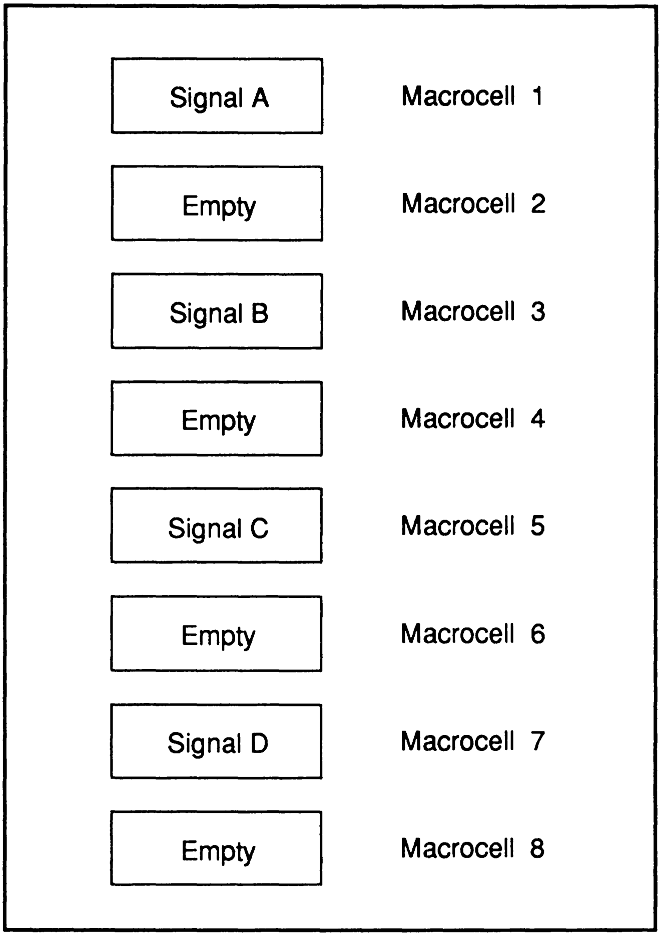
- If fitting is successful, place one or two more.
- If fitting is not successful, try different placement locations.

When placing signals in macrocell locations, you can either use adjacent macrocells or leave them empty. Leaving a macrocell empty releases its associated switch-matrix resources.

The following figure shows placement using adjacent macrocells.



The next figure shows placement leaving adjacent macrocells empty.



Determining the correct placement technique depends on the type of logic in the design, as discussed next.

5.3.3.1 Large Logic Functions

Spread out large logic functions. Logic functions with more than four product terms need the resources of more than one macrocell. As a result, placing large functions in adjacent macrocells limits the use of product-term steering and may lead to problems.

In the MACH 110, a macrocell can borrow four product terms from two adjacent macrocells for a total of 12.

Borrowing can only occur if the adjacent macrocell is not already used.

- If an equation has five to eight product terms, leave at least one adjacent macrocell empty.
- If it has more than eight product terms, leave both adjacent macrocells empty.

In the MACH **210**, a macrocell can borrow four product terms from each of three macrocells; one directly above and two directly below.

- If an equation has five to eight product terms leave at least one adjacent macrocell empty.
- If an equation has nine to 12, leave at least two of the three empty.
- If an equation has more than 12, leave all three empty.

5.3.3.2 Large Functions at the End of a Block

The macrocells at the end of a block have only one adjacent cell that can be used for product-term steering. MACH 210 block end-cell numbers are 0 and 15; MACH 110 block-end cell numbers are 0, 7, 8 and 15. Do not use these locations for logic functions with more than eight product terms.

5.3.3.3 Adjacent Macrocell Use

Use adjacent macrocells for small logic functions that require a moderate percentage of a block's resources and have significant intra-block communications.

Leave adjacent macrocells empty when placing small logic functions that use a high percentage of a block's resources with few intra-block communications.

Leaving adjacent macrocells empty also works if you must place logic functions with many common inputs in different blocks. Logic functions with many common inputs, such as a Barrel Shifter, should be staggered and spaced when placed in successive blocks.

Logic functions associated with pin or node statements are placed in blocks in the order you list them in a Group statement. In this sense, you can control logic placement inside a MACH block. You can use statements such as the one shown next to stagger Qx logic placement.

```
...
GROUP MACH_SEG_A    Q0 Q2 Q1 A3
...
```

Leave adjacent macrocells empty in a MACH 210 design when placing functions using double feedback and input registers. Additional interconnection resources are needed for functions that use feedback from the output macrocell and the buried macrocell. This is also true for functions that use input registers. Leave adjacent macrocells empty when placing these functions. When placing functions in other blocks that use these feedback signals or registered inputs, leave adjacent macrocells empty.

5.3.4 GROUPING LOGIC

Block partitioning is one of the most important phases of the fitting process. In this phase, the software segments the design into groups to be fit into blocks in the MACH device.

Fitting success is heavily affected by the number of connections between different blocks. Logic grouping allows you to place a subset of the logic into a particular block without placing any other restrictions on specific cell placement.

Whenever possible, you should place logic with common inputs and feedback in the same block. This minimizes the number of wires crossing between blocks, which results in a lower demand for interconnection resources and an increased likelihood of a successful fit.

For a group of logic to fit into one block it must share common set and reset signals and follow the output-enable product-term rules. In addition, it must not exceed the block's product-term, macrocell, and storage-device resources.

To specify logic grouping during design entry, use the group MACH_SEG_*block* statement in your PDS file or edit part field 2 of the NODE or storage-device macros in the schematic.[3]

5.3.5 SETTING COMPILATION AND FITTING OPTIONS

You can affect the fitting process by changing the setting of various compilation and fitting options. A common design methodology is to use the default settings in the Compilation, MACH Fitting Options, and Logic Synthesis Options forms, then review the results of the compilation process.

If the design does not fit you can analyze the MACH report to determine the best options or use the Run until first success option on the MACH Fitting Options form.[4]

5.3.5.1 Gate Splitting

The gate splitting option on the Logic Synthesis Options form[5] controls the splitting of equations into smaller ones with fewer product terms.

- If the option is set to N, your equations will not be changed.

[3] Refer to Section III, Chapter 7, and Section IV, Chapter 11, for more details on the syntax of these commands.

[4] Refer to Section IV, Chapter 9, for details about the MACH Fitting Options form and other compilation options. Also, refer to Section IV, Chapter 11, for device-specific details.

[5] Refer to Section IV, Chapter 9, for details about each of the options available for this specification.

- If the option is set to Y, every equation that contains more than the maximum number of product terms will be split into smaller equations.

For example, assume your design contains an equation with 12 product terms and the option is set to Y with a maximum of four. The equation will be split into three equations of four product terms. It will take three passes through the array to implement the new equations.[6]

```
Logic Synthesis Options
Use automatic gate splitting?  Y   ... if 'Y', Max = 4
```

The default setting for automatic gate splitting is N.

Using product-term steering for the MACH 210 device can implement equations with up to 16 product terms. The MACH 110 can implement equations with up to 12 product terms using product-term steering.

- If you have a MACH 110 design that contains equations with more than 12 product terms, you must set the gate-splitting option to Y and set the maximum gates to 12 or less.

- If you have a MACH 210 design with equations that contain more than 16 product terms, you must set the gate-splitting option to Y and the maximum number to 16 or less.

For maximum speed, you should set this option to N and let the fitter implement larger equations using product-term steering. Equations implemented using this method require only one pass through the array. Product-term steering also decreases the total demand for signal routing resources because no feedback signals are required.

[6] Refer to Section IV, Chapter 11, for detailed information on splitting gates and steering product terms.

Use the following options on the MACH Fitting Options form to enable skipping adjacent macrocells option.

```
...
When compiling          Select one combination
Expand all PT spacing?              Y
...
```

See the discussion on the AMD MACH Fitter in the ***PALASM 4 Release Notes*** that accompany your software for a complete discussion of these techniques.

5.4 STRATEGIES, DESIGNS THAT DON'T FIT

Designs with high-utilization factors or preplaced pins and nodes may not fit on the first attempt. If the design does not fit, the following error message is reported on the last line of the MACH report.

```
File Processing Terminated
```

If this occurs you can use the fitting process output reports and error messages to locate problem areas. Corrective actions depend on the type of problem and include the following.

- Set compile options.
- Manually assign logic to blocks in the device.
- Float the locations of all pins and nodes.
- Fix the locations of pins and nodes judiciously.
- Change the architecture of the logic.
- Change the amount of logic in the design.
- Use a different MACH device.

If the design does not fit **and** you have preplaced any pins or nodes, the first thing you should do is force all signals to float using the MACH Fitting options form, then recompile the design.

- If this results in a successful fit you can try to modify the resulting pin cut using techniques in discussion 5.5.1.

- If the design does not fit with all pins and nodes floating, you should carefully examine the MACH report, described next, to identify problem areas and determine corrective actions.

5.4.1 - 5.4.13 MACH REPORT

The MACH report file contains error messages, statistics about the design, and detailed information on the result of the fitting process. For a complete discussion of the report file, see the ***MACH Technical***

"Interpretation and Use of the .RPT File", and the ***PALASM 4 Release Notes*** that accompanies your software.

5.4.1.14 Connection Status

The connection status follows the pin map. The presence of the pin map indicates a connection status of 100%.

5.4.1.15 Output Files, Errors and Warnings

The output files are indicated here along with error and warning counts and a fitting status message.

```
   The Design Doc is  stored in ===> C16_CARY.Rpt
   The Jedec Data is  stored in ===> C16_CARY.Jed
   The Placements are stored in ===> C16_CARY.Plc

   %% FITR %% Error Count: 0, Warning Count: 2
   %% FITR %% File Processed Successfully. - File: C16 CARY
```

Design doc refers to the name of the MACH report produced for this design. The names of the JEDEC and placement files are also listed. The design name appears on the last line following the status of the fitting process.

5.4.2 INTERPRETING ERROR MESSAGES

Three types of messages are produced by the fitting process: status, simple error, and complex error messages.

Status messages provide information about the progress of the fitting process. They can indicate potential problems in a design but do not necessarily indicate a no fit situation.

Simple error messages provide information on an error condition with an easily identifiable cause and solution. These errors are usually due to a design-rule violation. An example follows. Recovery for these types of messages are provided in the online help.

Too many clocks in design.

Complex error messages indicate a difficult error condition. In addition, there is one status message that often accompanies these errors. To interpret these messages, you must analyze the MACH report and your design logic. Recovery from these conditions can be an iterative process.

The following discussions identify some of the more common status message and complex error messages, possible causes, related parts of the MACH report, and recovery procedures. For a a more complete listing, see the ***PALASM 4 Release Notes*** that accompanies your software.

You can access the report by selecting Reports from the View menu, then choosing MACH report from the submenu.

The following information is provided in the MACH report after the header. The header includes the release number, copyright notice, and details about the design and file being processed.

5.4.1.1 Flags Used

A summary of the selected MACH fitting-process options follows.

```
   Flags Used:                        Unplace=False                Max Packing=True
   Flags Used:                    Expand Small=False                Expand All=True
```

Entering the letter Y beside an option on the MACH Fitting options form sets a flag to true, as identified in the table shown next.

MACH Fitting Options		Flag
OUTPUT:		
Report level		Detailed
Signal Placement		
Force all signals to float	Y	Unplace
Use placement data from		Design file
Fitting Options		
When compiling		Select one combination...
Save last successful placement<F3>		
Press <F9> to edit file containing		Last successful placement
Maximize packing of logic blocks?	Y	Max Packing
Expand small PT spacing?	Y	Expand Small
Expand all PT spacing?	Y	Expand All

5.4.1.2 Pair Analysis

A report on input and output pairs. Errors are flagged if you include illegal pair declarations in pin/node statements. Nothing is reported if no pairs are declared or produced automatically. The example that was used to generate this report did not include pairs.

```
        PAIR Analysis...
```

5.4.1.3 Pre-Placement & Equation Usage Checks

Illegal and conflicting preplacement errors are flagged here. For example, an error is reported when the placement of a signal defined using a Group statement, with the reserved word MACH_SEG_*block* as the group name, conflicts with the pin assignment for that signal in a pin or node statement.

```
     Pre-Placement & Equation Usage Checks...
```

5.4.1.4 Timing Analysis for Signals

Data includes the timing parameters and the maximum and minimum delays associated with signals.

- Pin-to-pin combinatorial propagation delay (Tpd)
- Flip-flop setup time (Tsu)
- Clock to output delay (Tco)
- Clock to register delay (Tcr)

The signal list includes only those with maximum delays; signals with less than maximum delays are not listed. Timing information is presented in terms of the number of passes through the array. The actual delay times for one pass through the array is provided in the device datasheet.

To calculate the actual delay time in nanoseconds, you multiply the propagation delay shown in the datasheet by the number of passes through the array.

```
  *** Timing Analysis for Signals

   Parameter    Min    Max           Signal List (Those having Max delay.)
        Tsu      1      1                 Q0              Q1              Q2
                                          Q3              Q4              Q5
                                          Q6              Q15
        Tco      0      0                 Q0              Q1              Q2
                                          Q3              Q4              Q5
                                          Q6              Q15
        Tcr      1      2                 Q8              Q9              Q10
                                          Q11             Q12             Q13
                                          Q14             Q15

   Key:
    Tpd - Combinatorial propagation delay, input to output
    Tsu - Combinatorial setup delay before clock
    Tco - Register clock to combinatorial output
    Tcr - Register thru combinatorial logic to setup
    All delay values quoted in terms of array passes
```

5.4.1.5 Device Resource Checks

This table shows the resources available in the selected device and those required by the design. If the resources required by a design are greater than those supported by the device, an error is reported and the process is terminated.

Important: Product terms are allocated in clusters of four. Therefore, the number of remaining product-term clusters may not correspond exactly to the number of product terms available minus the number used.

I/O macros include only those connected to I/O pads in the device. The total macro line includes buried macrocells.

Review the information contained in this section and compare it to utilization guidelines provided in discussion 5.3.2.

```
*** Device Resource Checks

                Available        Used         Remaining
        Clocks:     2              1              1
          Pins:    38             36              2      ->     94%
     I/O Macro:    32             16             16
   Total Macro:    32             18             14
 Product Terms:   128             65             56      ->     50%
MACH-PLD Resource Checks OK!
```

5.4.1.6 Block Partitioning

Statistical information, such as number of array inputs, I/O and buried macros used, product terms used, fanout, etc., are reported for each block. In addition, the signals assigned to each block are listed.

Results of the block partitioning phase of the fitting process are shown, including the distribution of logic among blocks and statistics about the utilization of each block.

This part of the report is important when locating problems related to block partitioning. You can correct these types of problems using logic grouping.

```
Partitioning Design into Blocks...

*** Last Equations Placed in Blocks

Weakly -

*** Block Partitioning Results
              Array    Macros     # I/O    Buried   Product    Signal
             Inputs    Remain     Macro     Logic    Terms     Fanout
   Block-> A   19        6          8         2        40        11
   Block-> B   22        8          8         0        32         8


*** Block Signal List
Block-> A        CARY_DN        CARY_UP         Q7
                      Q6             Q5         Q4            Q3
                      Q2             Q1         Q0

Block-> B            Q15            Q14        Q13
                     Q12            Q11        Q10            Q9
                      Q8
```

Information in this section of the report is described in detail below. The sequential list, if any, that follows the word Weakly contains affinity information on the last signals to be placed in blocks. The design used to generate this report had nothing to report.

The columns from Block through Buried Logic tell you how heavily a block is used and if you can pack more logic into it. The Product Terms column indicates connectivity. If the fanout number in the Signal Fanout column is excessive, it burdens the interconnection resources and causes congestion.

A. **Weakly**

   The first line contains signals with a weak affinity to other equations in the block, which means they share few common inputs. The second line contains signals with no affinity, which are arbitrarily placed into blocks with sufficient resources. If you need to remove logic from your design to achieve a fit, this information can help you determine which signals to remove.

B. **Block name** identifies the name of the block to which the following information applies.

C. **Array Inputs** shows the number of inputs to the array block, the maximum for each block is 22.

D. **Macros remaining** identifies the number of unused macrocells.

E. **I/O macros** indicates the number of macros used for I/Os connected to pins.

F. **Buried macros** defines the number of buried macrocells used to generate logic, not connected to pins.

G. **Product terms** identifies the number of product terms used; each block has 64.

H. **Signal fanout** lists the number of signals fanning out to other blocks.

The number of connections fanning out to other blocks is a measure of the number of wires crossing between blocks. A high number is an indication of a block partitioning problem. This problem can be corrected by manually partitioning signals among blocks using logic grouping commands.

The goal of logic grouping is to minimize the number of connections between blocks. You do this by grouping signals that share common inputs in the same block.[8]

I. **Block signal list** identifies the list of signals placed within the block.

5.4.1.7 Utilization

The overall utilization for the device is identified in this section of the MACH report. High utilization, 75-80%, indicates routing congestion caused by too many interconnects or excessive fanout. Very high device-utilization numbers, greater than 90%, can indicate difficulties in fitting the circuit; they may also suggest that the selected device is too small for the circuit.[9]

```
   Device Utilization....... *: 66%
```

[8] Refer to discussion 5.3.4, in this chapter, for more information on logic grouping strategies. To specify logic grouping, use the MACH_SEG_*block* statement in your PDS file, as described in Section IV, Chapter 10, or edit Part field 2 of the NODE storage-device macro in the schematic, as described in Section III, Chapter 7.

[9] Refer to discussions 5.3.3 through 5.3.5, in this chapter, for the utilization guidelines.

5.4.1.8 Assigning Resources

This area of the report provides statistics on the progress of the fitting process as it attempts to place and route signals. Errors are reported if signals cannot be routed. To report these statistics, select detailed reports on the MACH Fitting options form.[10]

```
...
Report level        Detailed
...
```

```
Assigning Resources...

*** Macro Block A
 Buried Logic>         CARY_DN         CARY_UP
      Targets>   0( 2)   2( 4)  4( 6)   6( 8)   8(14)  10(16)  12(18)  14(20)
        CARY_DN (A  0) -> (B  0)
        CARY_UP (A  2) -> (B  2)

   I/O Macros>              Q0             Q1             Q2             Q3
                            Q4             Q5             Q6             Q7
      Targets>   1( 3)   3( 5)  5( 7)   7( 9)   9(15)  11(17)  13(19)  15(21)

             Q0 (A  1) -> (A  1) (B  1)
             Q1 (A  3) -> (A  3)
             Q2 (A  5) -> (A  5)
             Q3 (A  7) -> (A  7)
          ...

*** Macro Block Inputs
       Inputs>            LOAD              UP          ENABLE            I5
                                   I6
      Targets>   0(10)   1(11)   2(13)   3(32)   4(33)

           LOAD (I  0) -> (A 16) (B 16)
             UP (I  1) -> (A 17) (B 17)
         ENABLE (I  2) -> (A 19) (B 19)
             I5 (I  3) -> (A 20)
             I6 (I  4) -> (A 21)
*** Macro Block B
       Inputs>            LOAD              UP          ENABLE            I5
```

[10] Refer to Section IV, Chapter 9, for detailed information on MACH fitting options.

5.4.1.9 Signals, Tabular

The next table lists signal names and corresponding pin numbers, block and cell locations in the device, the types and number of product terms for outputs, etc.

The information in this table is useful in identifying and resolving problems if the design does not fit.

You can use this information when minimizing inter-block connections as discussed under 5.4.1.10, Signals, Equations.

```
*** Signals - Tabular Information

          Signal  #   P/N #   (Loc)      Type     Logic  # PT  -Blocks-
            CLK   1   35       I  5    clock pin      .
         ENABLE   2   13       I  2      input        .            AB
           LOAD   3   10       I  0      input        .            AB
             UP   4   11       I  1      input        .            AB
             Q0   5    3       A  1     i/o pin   t-ff     3       AB
             Q1   6    5       A  3     i/o pin   t-ff     4       A
             Q2   7    7       A  5     i/o pin   t-ff     4       A
          ...
            Q15  20   42       B 14     i/o pin   t-ff     4       B
             I0  21   25       B  1      input        .            A
             I1  22   27       B  3      input        .            A
             I2  23   29       B  5      input        .            A
          ...
           I15  36   20       A 14      input        .            B
        CARY_UP  37    4       A  2      buried    cmb     1       B
        CARY_DN  38    2       A  0      buried    cmb     1       B
.
 Key:
  P/N #  - Pin/Node Number
    ?    - Signal Unplaced
  ...
  (Loc)  - Macrocell Location (Block & Cell)
  # PT   - Number of used product terms in logic
 -Blocks - Device blocks driven by signal
    comb - Combinatorial logic function
    d-ff - D-Type Flip-flop
    t-ff - T-Type Flip-Flop
```

Information in this section includes the items listed below. Information in the (Loc) and -Blocks- areas can be used to minimize interblock connections. In this case, you can try to confine signals to as few blocks as possible by placing related logic in the same block.

A. **Signal name** identifies the signal to which the following information applies.

B. **#** indicates the order of the pin/node statements in the declaration segment of the PDS file.

C. **P/N #** shows the pin number on the device to which the signal is assigned.

D. **(Loc)** indicates where the signal is located: Input, I, or block A, B, C, or D.

E. **Type** identifies the kind of signal, that is CLK, input, I/O, or internal (buried macros), and shows how device resources are used.

F. **Logic** identifies the flip-flop type: DFF, TFF, combinatorial.

G. **# PT** identifies the number of product terms used by the the signal.

H. **-Blocks-** indicates which blocks are fed by this signal.

5.4.1.10 Signals, Equations

This table is valuable if you need to minimize the use of signal-routing resources through the judicious grouping of signals.

```
*** Signals - Equations Where Used

   Signal Source              Fanout List
          CLK
       ENABLE:          Q0          Q1          Q2          Q3
            :           Q4          Q5          Q6          Q7
            :           Q8          Q9          Q10         Q11
            :           Q12         Q13         Q14         Q15
            {.... .... ..BB BBAA}

         LOAD:          Q0          Q1          Q2          Q3
            :           Q4          Q5          Q6          Q7
            :           Q8          Q9          Q10         Q11
            :           Q12         Q13         Q14         Q15
            {.... .... ..BB BBAA}

           UP:          Q1          Q2          Q3          Q4
            :           Q5          Q6          Q7          Q8
            :           Q9          Q10         Q11         Q12
            :           Q13         Q14         Q15
            {.... .... .BBB BAA}

           Q0:          Q0          Q1          Q2          Q3
```

Signals that drive other blocks require additional interconnection resources. If you can group signals so they only drive equations located in their own block, the likelihood of a successful fit increases. You can do this by building a list of signals that share common inputs as indicated in this table. Then place signals that share common inputs in the same block.

You can use information in tabular data to insure the design does not exceed block, macrocell, or product-term resources. Also be sure not to violate the set/reset rules described in 5.3.2.2.

You probably need not assign a group location to every signal to achieve a fit. When you do, start with the signals that result in the biggest reduction in the number of interblock connections.

To specify logic grouping you can **either** use the Group statement, with the MACH_SEG_*block* reserved word as a group name, in the PDS file **or** edit part field 2 of the NODE or storage-device macro in the schematic.[11]

5.4.1.11 Feedback Map

This map shows how each input and feedback signal is routed and complements information in the logic map. Feedback and interconnection blocks feed the PAL arrays, which in turn feed macros through the logic allocator. The map provides an overview of output signals being fed back to drive other outputs. Since the map shows which signals are available in a logic block, it can be useful when making decisions to modify existing logic without disturbing the fitting process beyond recovery. It also provides a visual measure of connectivity requirements.

In the example below, you can see that input signal I2 is assigned to A2; I10 is assigned to B5. I/O signals used by other outputs and all internally generated nodes are fed back through the interconnection resources.

```
*** Feedback Map - 16 BIT UP / DOWN COUNTER

Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
        | 0|          | 0|     |21| I6          CARY_DN : 0|     |21| Q9
        | 1|       Q0 : 1|     |20| I5               Q0 : 1|     |20| I11
        | 2|       I2 : 2|     |19| ENABLE      CARY_UP : 2|     |19| ENABLE
        | 3|       Q1 : 3|     |18| I7               I8 : 3|     |18| Q8
        | 4|       I1 : 4|     |17| UP              Q10 : 4|     |17| UP
        | 5|       Q2 : 5|     |16| LOAD            I10 : 5|     |16| LOAD
        '--'       I0 : 6|     |15: Q7              Q11 : 6|     |15: I14
                   Q3 : 7|     |14: I3               I9 : 7|     |14: Q15
                      | 8|     |13: Q6              Q12 : 8|     |13: I15
                   Q4 : 9|     |12: I4              I13 : 9|     |12: Q14
                      |10|     |11: Q5              Q13 :10|     |11: I12
                      '--+-u--u+--'                     '--+-u--u+--'
```

[11] Refer to Section III, Chapter 7, and Section IV, Chapter 10, for details on the syntax of these commands.

The example above also shows that Q0 is assigned to cells A1 and B1; in the logic map under 5.4.1.12, you can see that Q0 is assigned to logic block A, macro 1. Using the two maps with equations in the PDS file, you can determine that Q0 is generated as an output signal in logic block A. It is also fed back to blocks A and B to generate output signals in logic blocks A and B.

5.4.1.12 Logic Map

The logic map, shown next, graphically summarizes the assignment of output signals and internal nodes for each block in the MACH device. It also shows the signals assigned to global input pins. This visual summary helps you gauge device utilization, distribution of signals among logic blocks, and assignment of signals to macros.

```
*** Logic Map - 16 BIT UP / DOWN COUNTER

Gbl Inp .--.     I/O  .--+--A--+--.  I/O                     I/O  .--+--B--+--.  I/O
    LOAD| 0|  CARY_DN | 0| 1   |21|                            Q8 | 0| 4   |21|
      UP| 1|       Q0 | 1| 3   |20|                               | 1| .   |20|
  ENABLE| 2|  CARY_UP | 2| 1   |19|                            Q9 | 2| 4   |19|
      I5| 3|       Q1 | 3| 4   |18|                               | 3| .   |18|
      I6| 4|          | 4| .   |17|                           Q10 | 4| 4   |17|
     CLK| 5|       Q2 | 5| 4   |16|                               | 5| .   |16|
        '--'          | 6| .  4|15|  Q7                       Q11 | 6| 4  .|15|
                   Q3 | 7| 4  .|14|                               | 7| .  4|14|  Q15
                      | 8| .  4|13|  Q6                       Q12 | 8| 4  .|13|
                   Q4 | 9| 4  .|12|                               | 9| .  4|12|  Q14
                      |10| .  4|11|  Q5                       Q13 |10| 4  .|11|
                      '--+-u--u+--'                               '--+-u--u+--'
```

The small block on the extreme left shows the signals assigned to global input pins. In addition, each block in the MACH device is illustrated.

The signal name that appears is the one declared in the pin/node statements of the PDS file. The logic map shows the output signals, nodes, or I/Os assigned to different macros in each block.

The example above shows 22 possible signal locations for each logic block. However, macros are associated with only 16; the rest are inputs. The internal-node signal CARY_DN is assigned to block A, macro 0; output signal Q15 is assigned to block B, macro 14.

Beside each macro number, near the center of the block, is the number of product terms used for the signal. For example, Q0 has three product terms and Q8 has four.

- A **blank** that appears in place of a product term indicates a dedicated input.
- A **dot** in place of a product term indicates a macro that is not used.
- An **asterisk** in place of a number of product terms indicates product-term steering.

  In this case, the PT available to that macro was borrowed by the adjacent macro to generate logic requiring more than four product terms.

5.4.1.13 Pin Map

The pin map shows the pin assignment for each input and output signal in the design.

```
*** Pin Map
                                 I8
                                  |
                             Q0   |           .
                              |   |           |
                         I7   |   |           |   Q15
                          |   |   |           |   |
                     Q1   |   |   |           |   |   .
                      |   |   |   |           |   |   |
                 I9   |   |   |   |           |   |   |   Q14
                  |   |   |   |   |           |   |   |   |
           .------'---'---'---'---'---o-------'---'---'---'---.
           |                              4   4   4   4   4   |
           |      6   5   4   3   2   1   4   3   2   1   0   |
        Q2| 7                                              39|I4
       I10| 8                        G   V                 38|Q13
        Q3| 9                        n   c                 37|I3
      LOAD|10                        d   c                 36|Q12
        UP|11                                              35|CLK
       Gnd |12                                             34| Gnd
    ENABLE|13                                              33|I6
       I12|14                  V   G                       32|I5
        Q4|15                  c   n                       31|I11
       I13|16                  c   d                       30|Q11
        Q5|17                                              29|I2
           |    1   1   2   2   2   2   2   2   2   2   2     |
           |    8   9   0   1   2   3   4   5   6   7   8     |
           '----.---.---.---.-----------.---.---.---.---.-----'
                |   |   |   |           |   |   |   |   |
              I14   |   |   |           |   |   |   |   Q10
                    |   |   |           |   |   |   |
                   Q6   |   |           |   |   |   I1
                        |   |           |   |   |
                      I15   |           |   |   Q9
                            |           |   |
                           Q7           |   I0
                                        |
                                        Q8
```

5.4.1.14 Connection Status

The connection status follows the pin map. The presence of the pin map indicates a connection status of 100%.

5.4.1.15 Output Files, Errors and Warnings

The output files are indicated here along with error and warning counts and a fitting status message.

```
The Design Doc is  stored in ===> C16_CARY.Rpt
The Jedec Data is  stored in ===> C16_CARY.Jed
The Placements are stored in ===> C16_CARY.Plc

%% FITR %% Error Count: 0, Warning Count: 2
%% FITR %% File Processed Successfully. - File: C16_CARY
```

Design doc refers to the name of the MACH report produced for this design. The names of the JEDEC and placement files are also listed. The design name appears on the last line following the status of the fitting process.

5.4.2 INTERPRETING ERROR MESSAGES

Three types of messages are produced by the fitting process: status, simple error, and complex error messages.

Status messages provide information about the progress of the fitting process. They can indicate potential problems in a design but do not necessarily indicate a no fit situation.

Simple error messages provide information on an error condition with an easily identifiable cause and solution. These errors are usually due to a design-rule violation. An example follows. Recovery for these types of messages are provided in the online help.

Too many clocks in design.

Complex error messages indicate a difficult error condition. In addition, there is one status message that often accompanies these errors. To interpret these messages, you must analyze the MACH report and your design logic. Recovery from these conditions can be an iterative process.

The following discussions identify the status message and each of the complex error messages, possible causes, related parts of the MACH report, and recovery procedures.

- 5.4.2.1, Marginal block partitioning measure, Warning F120

- 5.4.2.2, Partitioning could not place all signals into blocks, Error F580

- 5.4.2.3, Product term distribution, Error 610

- 5.4.2.4, Not all input signals connected, Error F600

- 5.4.2.5, Connection problem (wiring congested), Error F590

- 5.4.2.6, Mapping difficulty - no feasible solution, Error F620

Since two error conditions often occur together, and one message is a status message, design examples are not provided for all error conditions. Each design example shows a typical sequence of actions you can take to recover from problems.

5.4.2.1 Marginal Block Partitioning Measure, Warning F120

This message is only a warning; the fitting process may still be successful. However, poor partitioning may cause problems later in the fitting process that result in unconnected signals and product-term distribution errors.

In the partitioning phase of the fitting process, equations are assigned to blocks in the MACH device. This assignment is accomplished by analyzing the number of common inputs shared between equations. Equations are assigned similarity measures and are placed to reduce connections between blocks.

```
|> WARNING F120 - Marginal Block Partitioning Measure:  (too high)  18
-----------------------------------------------
Explanation:
 The process of dividing up user logic signals between MACH device blocks
 was not optimal for this design. You should consider using manual block
 partitioning commands to improve partitions and make more room for logic.
```

However, this process fails when there are **either** too few **or** too many shared signals. In this case, signals with common inputs are scattered, which results in higher utilization of interconnection resources. The number following this error message is the similarity count, which is an indirect measure of the number of common inputs.

- A low number, such as four, indicates too few.
- A high number, such as 15, indicates too many.

Interpreting the Report

Several parts of the fitting-process report contain information that can help you determine how to manually group signals.

- **Signals, Tabular**: For each signal in the design, this table shows the blocks each signal drives.
- **Signals, Equations**: This table shows the fanout list for each signal in your design.
- **Block Partitioning Results**: The first table shows statistics on each block in the device; the second table provides a list of signals placed in each block.

Recovery

Consider using Group statements with the reserved word MACH_SEG_*block* as a group name to place signals in logical groups. Group signals with common inputs in the same block to minimize the number of interconnections between blocks. You must answer no to force all signals to float on the MACH fitting options form when your use a Group statement. Otherwise, you preplacements will be ignored.

```
...
Force all signals to float?            N
...
```

For **state machine designs**, you can implement all state bits within a single block when product terms and input resources are sufficient. In this case, you place decoded outputs in other blocks based on the state bits or inputs they use. However, if product term and input requirements prevent these groupings, consider introducing extra state bits to decouple transitions and outputs from each other. Then place the outputs and state bits together in the same block.

Designs with minimum shared inputs require other considerations. In data path and shift register applications, one output feeds only a few equations in a deep chain. Often there are only a few inputs that drive all stages of the design. For this type of application, you can usually reduce intra-block communication by manually grouping signals in a vertical bit-slice or nibble fashion.

5.4.2.2 Partitioning Could Not Place All Signals into Blocks, Error F580

During the partitioning phase of the fitting process, equations are assigned to blocks in the MACH device. The partitioner places equations in a block until it becomes full or until all related equations are placed. The partitioner then moves on to another block or another group of related equations.

This error is issued when there are no remaining blocks to place equations. Following the error message are the signals that could not be placed in blocks.

```
|> ERROR F580 - Partitioning could not place all signals into blocks!
-------------------------------------------------
Explanation:
 The process of dividing up user logic signals between the MACH device
 blocks failed. Please rework design and/or remove equations to correct.
 (See online documentation for additional explanation & recovery actions.)

-------------------------------------------------
       Signals:   Q8
```

The following considerations are evaluated during the fitting process when determining partitions between blocks.

- The similarity between equations

- Shared resources such as set and reset signals
- The number of product terms
- The number of block inputs
- The number of macrocells

Some resources may be unused or reserved if the maximize packing of logic blocks option is set to N in the MACH Fitting options form.

Interpreting the Report

This error is usually followed by a diagnostic message which provides more information on the cause of the failure. For example, the message shown next may be issued.

```
|> WARNING F110 - Blk A full! (all available Inputs used)
|> WARNING F110 - Blk B full! (all available Inputs used)

Try Using Max Packing Density Option
```

The partitioning-results table shows statistics for each block in the device. This provides a list of signals placed in each block.

Recovery

There are several possible recovery procedures.

If the root cause of the problem is exhaustion of product terms or input or I/O resources, you can **maximize the packing of logic blocks** using appropriate specifications in the MACH Fitting Options form. This allows the fitting process to use all the resources in each block and may produce a fit.

```
...
While compiling                  Select one combination
Maximize packing of logic blocks?              Y
```

Improve the use of shared resources. In the MACH device, all equations in a block share set, reset, and output enable control signals. Whenever, a new control signal is specified, the equations using the signal must be placed in a new block.

Using too many independent control signals in your design prevents the fitting process from placing all of your equations into blocks. In this case, you can remove all SETF, RSTF and TRST equations and recompile. If this results in a fit, incrementally add the equations back into the design to isolate the problem.

Once the problem is traced to a particular resource, you must usually rework your logic to fit within the MACH architecture.

Rework the logic. If the procedures above fail to produce a fit, you must rework the logic. You can looking at the diagnostic messages to determine which resources are depleted. For example, if the message indicates all available inputs are used, reduce the number of inputs until the design fits.[12]

5.4.2.3 Product Term Distribution, Error 610

During the fitting process, equations are divided into product terms, the number of available product terms is verified, and placement is attempted.

Each macrocell can implement four product terms. The MACH **110** can support equations with 12 product terms. The MACH **210** can implement equations with 16 product terms. If the equation requires more than four product terms, the macrocell can borrow additional terms from adjacent neighbors, which is called **product-term steering**.

[12] Refer to discussion 5.4.2.7, in this chapter, for details about reducing the logic in a design.

It is also helpful to remove a signal which is the sole user of a device input. This eliminates the need to bring that input into the device, freeing up internal routing resources.

5.4.3 USING DIFFERENT FITTING OPTIONS

You may be able to remedy a no-fit situation by using different MACH fiting options during compilation. See the discussion on the MACH Fitter in the ***PALASM 4 Release Notes*** that accompanies your software.

The **device-resource checks** table shows the total number of product terms available in the device, the total number of product terms in the design, and the number of four-product-term clusters remaining.

Product terms are allocated in clusters of four. As a result there may be zero product terms available even if the total number in your design is less than the total number in the device. For example, an equation with six product terms requires resources from two macrocells. The first macrocell is used to implement four product terms and the second implements the remaining two. This leaves two product terms in the second cell that can not be used by another equation.

The **tabular signals** table shows the signal type and number of product terms for each signal in the design. In this case you must do the following for each signal. Divide the number of product terms by four to determine the number of adjacent macrocells needed to implement the equation using product-term steering.

Important: Buried nodes are processed last in the fitting process and are most likely to be affected by fragmented product-term resources.

Recovery

There are several possible recovery procedures.

Change preplacement locations. Check to see if you specified locations for any of the signals reported with this message. In this case, try the strategies below.

- Change specific locations in the design and spread out large equations leaving adjacent macrocells empty. Do not place large equations in end-cell locations.[13]

[13] Refer to 5.3.3, in this chapter, for details about assigning pin and node locations.

5.5 CHANGES AFTER SUCCESSFUL FITTING

It may be desirable to make a change to a design following a successfu! fit. For example, you may want to change the pin out to accommodate a new design requirement. Making changes to the design after a successful fit may result in a no fit situation. However, there are techniques you can use to make changes to the design without causing these problems.

See the ***MACH Technical Brief*** titled "Designing for Change with MACH Devices" for suggestions on how to modify a design file so that circuit revisions do not impact fitting or alter device pinouts.

Simplify the logic to reduce the number of product terms. This may involve logic optimization, changing the architecture of your design, or removing some of the logic.[14]

Design Example

A design with a large number of product terms produces this error. After compilation, the MACH report indicates a product-term distribution error and lists the equations that could not be placed for this reason.

```
|> ERROR F610 - Product Term distribution - No feasible solution!
-----------------------------------------------
Explanation:
 The particular distribution of user product terms is incompatible
 with the PT resources and connection limits of the product term allocator.
 Try to redistribute logic between blocks and/or simplify equation terms.
 (See online documentation for additional explanation & recovery actions.)

-----------------------------------------------
Try Using Expand Product Term Option
```

The following statements in the MACH report indicate that the 77% overall device utilization is above the 70% guideline but might still fit if there are no other parameters near their limits.

```
|> INFORMATION F050 - Device Utilization....... *: 77 %
```

The utilization section of the report shows high product-term utilization at 100 percent. In addition, the 0 under remaining product terms indicates that all four-product-term clusters are used.

[14] Refer to discussion 5.4.2.7, in this chapter, for details about reducing the logic in a design.

```
*** Device Resource Checks

               Available        Used        Remaining
       Clocks:    2              0             2
         Pins:    38             30            8      ->    78%
    I/O Macro:    32             11            21
  Total Macro:    64             24            40
Product Terms:    256            224           0      ->   100%

MACH-PLD Resource Checks OK!
```

The tabular signal information, below, indicates each signal reported within the product-term distribution error is a buried node. These signals all contain more than four product terms so they require resources from more than one macrocell.

```
*** Signals - Tabular Information

         Signal   #   P/N #   (Loc)     Type    Logic  # PT    Blocks
           S01   31    0       .?.     buried    comb   15        C
           S02   32    0       .?.     buried    comb    6        C
           S03   33    0       .?.     buried    comb   15     @   D
           S11   34    0       .?.     buried    comb   15         D
           S12   35    0       .?.     buried    comb    6         D
           S13   36    0       .?.     buried    comb   15     @A
```

There are several possible solutions to produce a fit for this design. First, you can check for improper preplacement of signals. In this case, however, all signals are designated as floating in the PDS file.

Next, you can re-run the fitting process with different sets of options. Although, in this case, running all combinations until the first fit still results in a no fit. Changing block assignment using the Group statement with the MACH_SEG_*block* reserved word as a name also fails to produce a fit.

Further analysis of the PDS file, shown next, reveals that many equations share groups of product terms. For example, S11 and C11 share the same product terms described in the String C10 statement.

```
    STRING C10 '((A1 * B0 * K1) + (A1 * B0 * S01) + (K1 * S01))'

    S11 = (A1 * B1) * C10 * S02
        + (A1 * B1) * /C10 * /S02
        + /(A1 * B1) * /C10 * S02
        + /(A1 * B1) * C10 * /S02

    C11 = (A1 * B1) * C10
        + (A1 * B1) * S02
        + C10 * S02
```

If you create a node that implements the shared product terms and use it in both equations you can reduce the number of product terms in the design. This reduction causes additional propagation delay for the equations using the node output. You can create eight nodes to implement shared groups of product terms and rerun the fitting process with successful results.

5.4.2.4 Not All Input Signals Were Connected, Error F600

In the last phase of the resource allocation process, after all equations are placed, the input signals are assigned to specific locations. These signals can be placed in any of the following locations.

- Dedicated input pads
- Cells associated with buried logic nodes
- Unused macrocells
- Macrocells whose product terms are allocated to other logic equations

If the fitting process is unable to find locations and connection resources for all inputs, the above error is output. The numbers following this message indicate the number of unplaced signals and unrouted connections.

Interpreting the Report

Two sections of the MACH report contain information you can use to determine manual block placement strategies to minimize the number of blocks driven by each input.

- The **tabular signals** table shows the blocks driven by each input.

- The **signal equations** table shows the fanout list for each input in your design.

Recovery

There are two possible recovery procedures.

Use **manual block assignment** to minimize the number of different blocks the input signals need to drive. This reduces the number of connections needed and may result in a successful fit.

Removing logic equations provides additional locations for inputs to be placed and frees switch-matrix resources.

Design Example

An example of a design that fails this way is a 16-bit preloadable counter followed by a 16-bit multiplexer in a Mach 110 device. In this design, a SELECT signal controls the output of the MUX. When the SELECT signal is low, the output of the MUX contains the output of the counter. When the SELECT signal is high, the output of the MUX contains the preload input.

The design I/O consists of the following. The location of all pins and nodes are left floating.

- 16 preload inputs
- Four control inputs
- One clock input
- 16 counter outputs assigned to buried nodes
- 16 multiplexer outputs assigned to pins.

When you run this design through the fitter, the output report contains an error message indicating it was not able to place and connect all the inputs successfully. The list of signals shows that it failed to place all the preload inputs except one. Each of the 15 unplaced signals has a fanout of two, which results in 30 no connects as indicated in the error message.

```
.......********.********...
|> ERROR F600 - Not all input signals were connected! (signals=15/nc=30)
-------------------------------------------------
Explanation:
 All possible pads that could host inputs have been tried for these signals
 with no success in making necessary connections through the Switch Matrix.
 You need to reduce the amount of logic contained within the design or
 redistribute functions between logic blocks to alter the connections needed.
 (See online documentation for additional explanation & recovery actions.)

-------------------------------------------------
Try Using Expand Product Term Option
```

The utilization section of the MACH report shows that the pin and macrocell utilizations are high while the product-term utilizations are acceptable.

```
                    Available        Used         Remaining
            Clocks:    2              1              1
              Pins:    38             37             1      ->      97%
         I/O Macro:    32             16             16
       Total Macro:    32             32             0
     Product Terms:   128             94             0      ->      72%
```

This design uses all 32 available macrocells. 16 are used for I/O and 16 for buried nodes.

The fitting process can place input signals in any empty macrocell or in macrocells used for buried nodes. In addition, the device contains dedicated locations for input signals. In this design there are no empty macrocells; 16 are used for buried nodes. These 16, together with the six dedicated inputs, result in a total of 22 possible locations for input signals.

The design uses 21 inputs, which makes it difficult for the fitter to place them all using default fitting options.

Analysis of the block assignments, shown below, indicates the fitter has done a good job of partitioning by placing all logic associated with the low order bits in block A and high order bits in block B. In this report, Q0 through Q15 are the counter outputs and O0 through O15 are the multiplexer outputs.

```
*** Block Signal List

Block-> A           O7          O6          O5          O4
                    O3          O2          O1          O0
                    Q0          Q1          Q2          Q3
                    Q4          Q5          Q7          Q6

Block-> B          O17         O16         O15         O14
                   O13         O12         O11         O10
                   Q10         Q11         Q12         Q13
                   Q14         Q15         Q17         Q16
```

Analysis of the logic map, shown next, reveals the six inputs that were placed used the dedicated input locations. In addition, the multiplexer output equations were placed in adjacent locations. The locations that remain for inputs apparently do not contain sufficient interconnect resources to implement required logic.

```
*** Logic Map - COUNTER FOLLOWED BY MULTIPLEXER

 Gbl Inp .--.    I/O  .--+--A--+--.  I/O          I/O  .--+--B--+--.  I/O
    LOAD| 0|      O0  | 0| 2    |21|               O10 | 0| 2    |21|
  SELECT| 1|      O1  | 1| 2    |20|               O11 | 1| 2    |20|
      UP| 2|      O2  | 2| 2    |19|               O12 | 2| 2    |19|
   COUNT| 3|      O3  | 3| 2    |18|               O13 | 3| 2    |18|
      I5| 4|      O4  | 4| 2    |17|               O14 | 4| 2    |17|
    CLK1| 5|      O5  | 5| 2    |16|               O15 | 5| 2    |16|
        '--'      O6  | 6| 2   4|15|  Q7           O16 | 6| 2   4|15|  Q17
                  O7  | 7| 2   4|14|  Q6           O17 | 7| 2   4|14|  Q16
                  Q0  | 8| 3   4|13|  Q5           Q10 | 8| 3   4|13|  Q15
                  Q1  | 9| 4   4|12|  Q4           Q11 | 9| 4   4|12|  Q14
                  Q2  |10| 4   4|11|  Q3           Q12 |10| 4   4|11|  Q13
                      '--+-u--u+--'                    '--+-u--u+--'
```

To recover from this error, you can try using different fitting options: Run until first success, for example. Some of these options force the fitter to leave empty locations between macrocells when it places multiplexer outputs. This provides a greater range of choices when placing the inputs and potentially leads to a successful fit.

The MACH report indicates the fitter found a successful placement using the Expand small PT spacing and Maximize packing of logic block options.

5.4.2.5 Connection Problem (Wiring Congested), Error F590

The routing resources to make connections for feedback signals are finite. This message indicates one of the required connections is blocked by previously routed signals.

Interpreting the Report

The information in the utilization segment of the MACH report does not show interconnection resource utilization. You deduce an indication of heavy congestion in the wiring channels by observing a large number of product terms and a large demand for pins.

Recovery

Float all signals. If you preplaced any pins or nodes rerun the fitting process with all signals floating.

Try different fitting process options. Use the **Run to first fit option** to try all combinations.

Reducing the complexity of the logic in your design reduces the number of connections needed to implement your design.[15]

Design Example

The following MACH 110 device 13-bit up/down preloadable counter is an example of a design that fails this way. The design I/O consists of the following elements.

- 13 preload inputs
- Three control inputs
- One clock input

[15] Refer to discussion 5.4.2.7, in this chapter, for details about reducing the logic in a design.

- 13 counter outputs

  The counter outputs are assigned to pins; the location of each pin is left floating.

In a standard 13-bit binary counter, the output of each flip-flop drives the inputs to all of the higher-order bits. As a result, the product terms that feed the high-order bits have a large number of inputs.

When the counter is partitioned for the Mach 110 device, some of the bits are placed in block A and some are placed in block B. This results in a large number of inputs to both blocks and a large number of interconnections between blocks that causes fitting to fail with the error message shown next.

```
 |> ERROR F590 - Connection problem (Wiring Congested) - Q10
-----------------------------------------------
Explanation:
 The interconnect switch matrix is unable to provide a needed
 connection at this point. (It was allocated to some other variable.)
 You need to reduce the amount of logic contained within the design.
 (See online documentation for additional explanation & recovery actions.)

-----------------------------------------------
*..*****..***.....
```

In addition, the fitter cannot connect all the inputs as indicated by error message 600.

```
.......********.********...
|> ERROR F600 - Not all input signals were connected! (signals=15/nc=30)
-----------------------------------------------
Explanation:
 All possible pads that could host inputs have been tried for these signals
 with no success in making necessary connections through the Switch Matrix.
 You need to reduce the amount of logic contained within the design or
 redistribute functions between logic blocks to alter the connections needed.
 (See online documentation for additional explanation & recovery actions.)

-----------------------------------------------
Try Using Expand Product Term Option
```

These messages often occur together since the fitter places and connects inputs after all other signals. If logic feedback uses a high percentage of internal routing resources there is little left for input signals.

Analysis of the utilization section of the MACH report, shown next, indicates macrocell and product-term utilization are acceptable. This confirms the root of the problem is high demand for internal routing.

```
*** Device Resource Checks

                 Available        Used         Remaining
        Clocks:     2              1              1
          Pins:    38             36              2       ->     94%
     I/O Macro:    32             16             16
   Total Macro:    32             18             14
 Product Terms:   128             65             56       ->     56%
```

There are several possible strategies to fit this design. First, first look at signal preplacement in the PDS file. In this design, all signals are floating. Had any signals been preplaced, you could re-fit using the Force all signals to float option on the MACH Fitting Options form.

...
Force all signals to float?   Y
...

Next, try re-fitting using different options.

...
When compiling   Run all until first success
...

In this case, however, the Run all until first success option fails to produce a fit. When this occurs, the next step is to try a manual partitioning solution. Interblock connections can be improved by manually grouping the low-order bits in block A and the high-order bits in block B. Although, for this particular design, fitting with this grouping still results in a no fit situation.

To get this circuit to fit, you must redesign the logic to reduce the number of interconnections between the blocks. This can be accomplished by dividing the counter into two smaller counters that are cascaded with a carry signal. You manually place one of the counters in block A and the other in block B using Group statements with the MACH_SEG_*block* reserved word as a name.[16]

After updating the design logic, you only have to route one carry signal instead of routing all the low-order counter outputs to block B. This significantly reduces the number of interconnections between blocks. Then compile the updated design using the Run all until first success option on the MACH Fitting Options form.

The MACH report indicates fitting was successful using the Maximum packing of logic blocks and Expand small PT spacing options.

Important: This modification does affect circuit performance by adding an additional pass through the array. Analysis of the logic shows the low-order bit is the only signal to change on every clock cycle. You can recover the lost performance by removing the low-order bit from the carry logic and routing it to block B.

This results in one additional interblock connection but does not cause the fitting process to fail.

5.4.2.6 Mapping Difficulty – No Feasible Solution, Error F620

This message indicates the resource-allocation process has failed to find a feasible mapping solution for a signal. There are several possible recovery procedures.

Float all signals. If you preplaced any pins or nodes rerun the fitting process with all signals floating.

[16] Refer to Section IV, Chapter 10, for details about using MACH_SEG_*block*.

```
...
Force all signals to float?      Y
...
```

Try different fitting process options. Use the following option on the MACH Fitting Options form to try all combinations.

```
...
When compiling                Run all until first success
...
```

Reducing the complexity of the logic in your design reduces the number of internal connections the fitting software must use to implement the design.

5.4.2.7 Procedures For Reducing Logic Complexity

The final recovery action for fitting process errors is to reduce the complexity of the logic in your design. This discussion provides guidelines on how to proceed if this becomes necessary.

Enable logic minimization by turning minimization on to reduce the number of product terms and input signals for each function.

Selecting a **flip-flop optimization** option using the Logic Synthesis Options form can also reduce the complexity of your logic, especially for counter functions.

Remove Logic Equations: The weakly segment within the block-partitioning section of the MACH report contains information on the last signals to be placed in blocks. This is a sequential list.

The first line contains signals with a weak affinity to other equations in the block. This means they share few common inputs. The second line contains signals with no affinity. These signals are placed arbitrarily into blocks with sufficient resources. If you need to remove logic from the design to achieve a fit, this information can help you determine which signals to remove. Start at the **end** of the list and remove signals until the design fits successfully.

It is also helpful to remove a signal which is the sole user of a device input. This eliminates the need to bring that input into the device, freeing up internal routing resources.

5.4.3 USING DIFFERENT FITTING OPTIONS

You may be able to correct a no-fit situation by using different MACH fitting options during compilation. Review the information discussed under 5.3.5 to see how the logic in the design relates to certain options, then use these options accordingly: Use automatic gate splitting, Maximize packing of logic blocks, Expand small PT spacing, or Expand all PT spacing.

- If you are not sure about the logic or if the design contains a mix of different logic types, try the option below on the MACH Fitting Options form.

```
...
While compiling                 Run all until first success
...
```

- If the MACH report contains the error message about unplaced elements, try the option below on the MACH Fitting Options form, as described under 5.3.5.2.

```
...
While compiling                 Select one combination
Expand all PT spacing   Y
...
```

5.4.4 CHOOSING A LARGER MACH DEVICE

When the device utilization section of the MACH report contains errors due to lack of available resources or if you cannot get the design to fit using the techniques described above, you may have to switch to a larger MACH device. Review the utilization guidelines under 5.3.2 to determine which MACH device is appropriate for your application.

To switch to a larger MACH device in a schematic-based deign, you must change the Device = field in the schematic control file. In a text-based design, you must change the Device = field in the PDS file.

5.5 CHANGES AFTER SUCCESSFUL FITTING

It may be desirable to make a change to a design following a successful fit. For example, you may want to change the pin out to accommodate a new design requirement. Making changes to the design after a successful fit may result in a no fit situation. However, there are techniques you can use to make changes to the design without causing these problems.

5.5.1 CHANGING THE PIN OUT

After achieving a fit with all pins floating you may want to make changes to the pin out to get a more desirable signal order. You can make the following changes to the pin order of a successful fit without causing fitting problems. To implement these changes, you can back annotate the PDS file to include signal placements from the last successful fitting process, then edit the updated PDS file.[17]

You can **exchange individual inputs** that drive the same block. In this case, you can swap the locations of any two inputs that drive the same block or set of blocks without affecting the fitting success.

You can **exchange any two outputs** placed **within the same block** as long as they have the same product-term requirements. To determine if two outputs in the same block can be swapped, you must calculate the number of four product-term clusters used by the equation. To do this, you locate the two signals in the tabular signals section of the MACH report and divide the number of product terms by four. Any fractional result should be increased to the next highest number.

For example, an equation with eight product terms uses two clusters. An equation with nine product terms uses three clusters.

[17] Refer to Section IV, Chapter 9, for details about back annotating signals.

If two outputs in the same block use the same number of product-term clusters their locations can be swapped without affecting fitting results. Swapping inputs or outputs that do not follow this guideline may result in a no fit situation.

5.5.2 CHANGING LOGIC

After achieving a successful fit you can make the following changes to the logic without affecting fitting results.

You can add a product term to an equation as long as it meets the following criteria.

- It does not cause the equation to cross a four cluster boundary.
- It uses inputs that are already used elsewhere in the block.

For example consider the following equations that have been placed in block A, where B through H are primary inputs to the device.

X = B+C+D+E+F+G

Y = G+H

The following change to equation X will not change the pin out or risk a no fit.

X = B+C+D+E+F+G+H

The original equation for X contains six product terms. As a result, it has been allocated two clusters or eight product terms. This leaves two unused product terms that are available for future changes. Adding product-term H does not cross the two cluster boundary and therefore does not require additional product-term resources.

The equation for Y already contains the input signal H. Since equation Y is in block A, input H has been successfully routed and is available to all the macrocells in the block. Adding this input to X does not require any additional routing resources.

- If the change to X had increased the number of product terms from six to nine, additional resources would have been required.
- If the change to X had added an input that was not already used in the block, additional routing resources would be required.

In either case the pin out may have changed or a not fit situation may have resulted.

- If the changes to the logic meet the above guidelines, you can preserve the pin out generated from a successful placement by using the back annotate signals command.[18]

[18] Refer to Section IV, Chapter 9, for more information on placement back annotation.

CHAPTER 6

SIMULATION

CONTENTS

6 SIMULATION

This chapter describes the features of the PALASM 4 simulator and provides a Boolean equation and state-machine design example to illustrate simulation concepts. The chapter is divided into five major discussions.

- Discussion 6.1 presents an overview of the PALASM 4 simulation process.
- Discussion 6.2 presents a summary of the simulation keywords and considerations for simulating a design.[1]
- Discussion 6.3 provides general information about viewing simulation results.
- Discussion 6.4 provides general information about using the FOR, WHILE, and IF-THEN-ELSE simulation constructs.
- Discussion 6.5 presents a Boolean equation and a state-machine design example to illustrate the simulator's features.

Note: The simulation files for the design examples provide comments to explain each line.

[1] Refer to Section IV, Chapter 10, for details on the command syntax and device support.

6.1 OVERVIEW

The PALASM 4 simulator allows you to perform functional verification of all PLD designs, including those for a MACH-device. You define simulation commands in **either** the simulation segment of the PDS file **or** in an auxiliary simulation file.

After entering the simulation commands, you simulate the design and view the results in either a graphical waveform or text format.

Note: Because the PALASM 4 simulator performs functional verification, additional delays induced by looping back through an array are not reflected in the simulation results.

6.2 CREATING A SIMULATION FILE

You create a simulation file to specify a sequence of simulator input commands. A command is comprised of a keyword and a list of parameters. This discussion provides a summary of the simulation keywords, information about the simulation segment and auxiliary file, and considerations for simulating a design.

6.2.1 SIMULATION COMMAND SUMMARY

The following keywords are provided with the PALASM 4 simulator. Included here is a brief summary of each command.[2] In the following examples, O1, O2, O3, and O4 are pin names;Q0 and Q1 are register output names; and PLAYING is a state name.

Note: The Boolean equation and state-machine designs provided in discussion 6.5 exemplify the use of many of these commands.

CHECK

Use this keyword to verify that values at the pin are equal to expected values. For example:

```
CHECK O1 /O2 ^O3 %O4 PLAYING
```

CHECKQ

Use this keyword to verify that values at the register outputs are equal to expected values. For example:

```
CHECKQ Q0 /Q1 PLAYING
```

FOR LOOP

Use this construct to perform a task a specified number of times.[3]

[2] Refer to Section IV, Chapter 10, for a thorough explanation of each keyword,.

[3] Refer to discussion 6.4.1 for an example of a FOR loop.

IF-THEN-ELSE

Use this construct to test a condition and then perform one of two tasks, depending on the test results.[4]

PRELOAD

Use this keyword to load specified values into the register outputs. For example:

```
PRELOAD Q0 /Q1 PLAYING
```

PRLDF

This keyword is only used with a few devices.[5] PRLDF loads specified values at the pins. For example:

```
PRLDF O1 \O2
```

SETF

Use this keyword to assign specific values to inputs during simulation. For example:

```
SETF IN1 /OE
```

SIMULATION

Use this keyword at the beginning of each simulation segment or auxiliary simulation file.

TEST

This keyword is only applicable for MACH device designs. TEST checks values at the register outputs. This keyword also creates a "T" vector in the JEDEC file, per standard 3B, and flags discrepancies between specified values and expected results. For example:

```
TEST Q0 /Q1 PLAY
```

TRACE_OFF

Use this keyword to end a simulation section being traced by the TRACE_ON keyword. For example:

[4] Refer to discussion 6.4.3 for an example of an IF-THEN-ELSE construct.

[5] Refer to Section IV, Chapter 10, for details on simulation command syntax and device support.

```
TRACE_OFF
```

TRACE_ON

Use this keyword to define which signals to record in the trace file during simulation. For example:

```
TRACE_ON IN1 IN2 O1 CLOCK
```

WHILE LOOP

Use this construct when you cannot predetermine how many times to perform a task. The task will be performed while a condition is true.[6]

6.2.2 SIMULATION SEGMENT VS. AUXILIARY FILE

You define the simulator input commands in **either** the simulation segment of the PDS file **or** in a auxiliary simulation file.

Note: In this chapter, the term simulation file is used to refer to either a simulation segment or an auxiliary simulation file.

The simulation segment looks the same as the auxiliary file, except it is part of the design file, as shown next.

[6] Refer to discussion 6.4.2 for an example of a WHILE loop.

```
;PALASM Design Description
;--------------------------- Declaration Segment ---------------------------------
                                         :
;---------------------------- PIN Declarations -----------------------------------
                                         :
;------------------------- Equations or State Segment ----------------------------
                                         :
;--------------------------- Simulation Segment ----------------------------------
SIMULATION

TRACE_ON    INPUT  CLOCK  OUTPUT  ;Specify signals for the trace output file

SETF        /CLOCK  INPUT         ;Initialize INPUT to logical 1, CLOCK to logical 0
CLOCKF      CLOCK                 ;Apply a full clock cycle to CLOCK.
CHECK       OUTPUT                ;Verify that the output pin is at logical 1
CHECKQ      /Q0                   ;Verify that the Q0 register is at logical 0

TRACE_OFF                         ;Turn tracing off
```

The auxiliary simulation file is a stand-alone file that must be in the same directory as the design; the file name should match the name of the design file and include a **.SIM** extension. An example of an auxiliary simulation file is shown below.

```
SIMULATION

TRACE_ON    INPUT  CLOCK  OUTPUT  ;Specify signals for the trace output file

SETF        /CLOCK  INPUT         ;Initialize INPUT to logical 1, CLOCK to logical 0
CLOCKF      CLOCK                 ;Apply a full clock cycle to CLOCK.
CHECK       OUTPUT                ;Verify that the output pin is at logical 1
CHECKQ      /Q0                   ;Verify that the Q0 register is at logical 0

TRACE_OFF                         ;Turn tracing off
```

Depending on the working environment you've set up, a message asks if you are using an auxiliary simulation file, **either** on demand **or** automatically when you simulate.[7]

An auxiliary simulation file is the best choice when you enter a schematic-based design.[8] In this case, a PDS file is created from schematic data; however, it does not include simulation data. Creating an auxiliary simulation file is useful because it is not erased each time a new PDS file is generated, and therefore provides a permanent medium for the simulator input data.

It is also important to use an auxiliary simulation file when you merge PDS files. In this case, the resulting design file contains a blank simulation segment. You can create the auxiliary file before or after the merge process because the auxiliary file is not over written.

6.2.3 CONSIDERATIONS

The following discussions provide general considerations for simulating a design.

- 6.2.3.1, Flip-Flops
- 6.2.3.2, Internal Nodes
- 6.2.3.3, Latches
- 6.2.3.4, Output Enable
- 6.2.3.5, Preloaded Registers
- 6.2.3.6, Verified Signal Values

[7] Refer to Section IV, Chapter 9, for details about using the Setup command on the file menu to define working-environment preferences.

[8] Schematic entry is available for MACH-device designs only.

6.2.3.1 Flip-Flops

Flip-flops are automatically reset at the start of a simulation. You can set the output state of any flip-flop during simulation by using the PRELOAD or PRLDF commands.[9]

Flip-flops can be clocked with the CLOCKF command or with a series of SETF commands.

Note: If you are not using the default clock, use SETF /<clock_name> to initialize the clock at the beginning of the simulation. Otherwise, the simulator reports a warning.

Also: Do not change the value of the data at the same time you apply the rising edge of a flip-flop clock. The simulator will not clock the data, and the programmer-based JDC file will be affected.

Finally: Do not apply a set and reset signal to a flip-flop simultaneously. This is simulated as an illegal state and results in an error message.

6.2.3.2 Internal Nodes

Internal nodes are treated as any other signal during simulation, but they are not included in the JEDEC output file. The logic states of internal nodes declared in the pin declarations segment are automatically displayed in the history file. The trace file displays only the signals specified with the TRACE_ON command.

[9] Refer to discussion 6.2.3.5, in this chapter, and the corresponding command entries in Section IV, Chapter 10, for more information.

6.2.3.3 Latches

The following illegal latch states will result in simulator error messages for an active-low latch.

LATCH ENABLE	ASYNC. RESET	ASYNC. PRESET
1	1	1
0	0	1
0	1	0
0	1	1

The following illegal latch states will result in simulator error messages for an active-high latch.

LATCH ENABLE	ASYNC. RESET	ASYNC. PRESET
0	1	1
1	0	1
1	1	0
1	1	1

6.2.3.4 Output Enable

If you did not write a pin statement for the output-enable pin, the simulator presumes the output pins to be enabled.

If you did write a pin statement for the output-enable pin, the simulator presumes the output pins to be disabled unless you explicitly set the output-enable pin active in the simulation file.

6.2.3.5 Preloaded Registers

You can preload a value into any register during a simulation session. In state-machine designs, this allows you to set the state bits as required to access any state directly.[10]

The PRELOAD command sets the Q output of the flip-flop to the specified value. The PRLDF command sets

[10] Refer to Section IV, Chapter 10, for details on the command syntax and device support.

the Q output of the flip-flop to whatever value produces the specified value at the pin.

Note: Even if a device does not physically support the preload function, you can simulate the design as if it did. However, no test vectors are generated in this mode.

6.2.3.6 Verified Signal Values

There are three simulator commands that verify the logic states of signals: CHECK, CHECKQ, and TEST.[11]

The CHECK command verifies that the simulation result(s) at the pin correspond to your predictions of the design's behavior. If a discrepancy is detected, a question mark is inserted in the simulation history and trace files at the corresponding signal and vector, and a warning is issued in the execution-log file.

The CHECKQ command verifies the value of a specified signal at the output of the register. This command is useful in state-machine and other designs where an inverter exists between the register and pin.

The TEST command generates messages about conflicts just like CHECKQ. However, in addition to verifying the signal at the output of the register, it changes the simulation results to match the specified values and generates a vector in JEDEC using the new format defined in the JEDEC 3B standard.

Note: TEST is only valid for MACH device designs. If this keyword is used for other types of designs, it is converted to a CHECKQ keyword during simulation.

[11] Refer to Section IV, Chapter 10, for details on the command syntax and device support.

6.3 VIEWING SIMULATION RESULTS

Once a simulation completes successfully, the results are stored in a history, and optionally, a trace file. You can view the results in a text or graphical format.

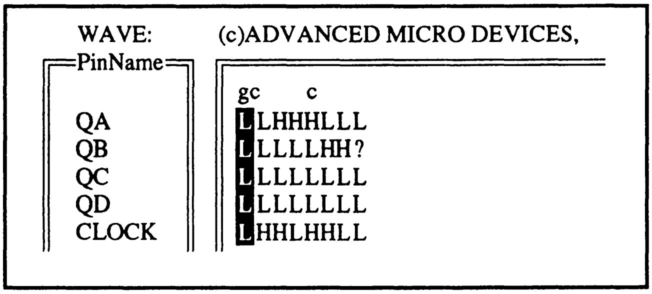
6.3.1 HISTORY FILE

The history file shows the results for every pin and node defined in the pin list of the design. The polarity of each pin and node is displayed according to the definition in the pin list. You can view this file in a graphical or text format.

Use the sequence View/Simulation Data.../History to view the text version of the history file. This display shows the status of all signals defined in the pin list using letters to represent various states.

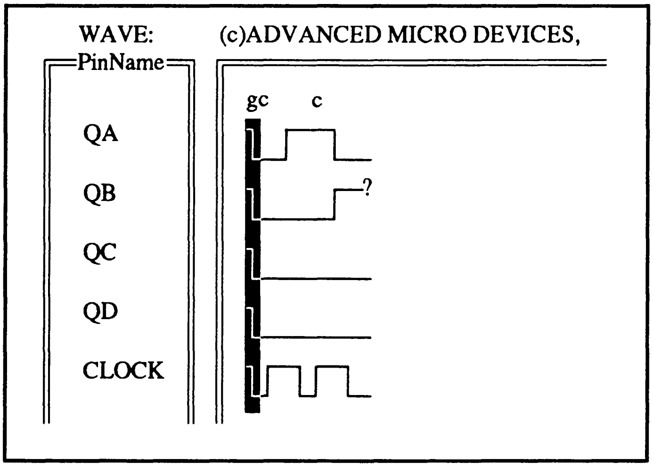
- H = high
- L = low
- X = undefined
- Z = high impedance
- ? = discrepancy

An example of a history text display is shown below.



```
 WAVE:        (c)ADVANCED MICRO DEVICES,
 PinName
               gc     c
 QA            LLHHHLLL
 QB            LLLLLHH?
 QC            LLLLLLLL
 QD            LLLLLLLL
 CLOCK         LHHLHHLL
```

Use the sequence View/Waveform Display.../History to view the waveform version of the history file. This sequence converts the ASCII characters into a graphical display similar to a timing diagram. An example of a history waveform display is shown next.



Note: If the simulation file includes CHECK, CHECKQ, or TEST commands, discrepancies between a specified value and the simulated value of a signal are flagged with a question mark, **?**, at the location of the discrepancy.

6.3.2 TRACE FILE

The trace file is only generated if the TRACE_ON command is included in the simulation file. This file shows results for the signals specified as parameters in the command. The polarity of each pin and node is displayed according to the definition in the TRACE_ON command. The trace file is useful for the following three situations.

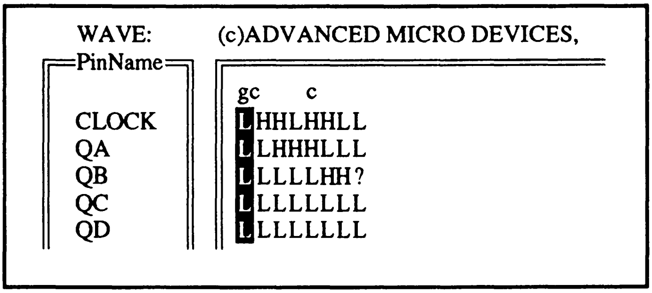
- If you do not want to display certain pins or nodes that are not relevant to a simulation session
- If you want to group signals by function, so they can be viewed on the same page of the display
- If you want to reverse the polarity of output signals that are defined as active high, but have active-low equations

You can view the trace file in a graphical or text format.

Use the sequence View/Simulation Data.../Trace to view the text version of the trace file. This sequence shows the status of all signals defined in the TRACE_ON command using letters to represent various states:

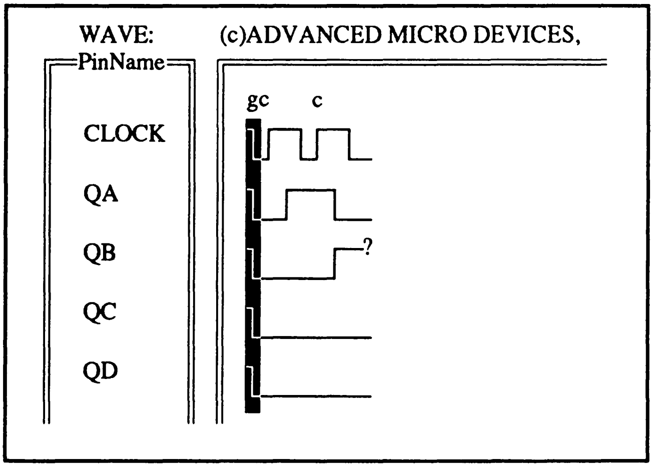
- H = high
- L = low
- X = undefined
- Z = high impedance
- ? = discrepancy

An example of a trace text display is shown next.



```
   WAVE:        (c)ADVANCED MICRO DEVICES,
  PinName
                  gc   c
   CLOCK          LHHLHHLL
   QA             LLHHHLLL
   QB             LLLLLHH?
   QC             LLLLLLLL
   QD             LLLLLLLL
```

Use the sequence View/Waveform Display.../Trace to view the waveform version of the trace file. This sequence converts the ASCII characters into a graphical display similar to a timing diagram. An example of a trace waveform display is shown below.



Note: If a forward slash, /, is used as part of a signal name in the TRACE_ON command line, the displayed value for that signal is inverted in the trace file.

Also: If the simulation file includes CHECK, CHECKQ, or TEST commands, discrepancies between the specified value and the simulated value of a signal are flagged with a question mark, **?**, at the location of the discrepancy.

6.4 USING SIMULATION CONSTRUCTS

The PALASM 4 simulator provides the following constructs for developing loops and making decisions during a simulation session.[12]

6.4.1 FOR LOOP

The FOR loop is the most basic flow-of-control construct. It is ideal for applications in which you can predetermine how many times you must repeat a set of instructions, as illustrated below.

```
SIMULATION

SETF /OE /CLOCK COUNT
FOR X:= 1 TO 9 DO
         BEGIN
            CLOCKF CLOCK
         END
```

6.4.2 WHILE LOOP

When you cannot predetermine how many times to perform a task, you can use the WHILE loop to perform the task while some condition remains true, as illustrated below.

```
SIMULATION

SETF /OE /CLOCK COUNT
WHILE ( /(BIT3 * /BIT2 * BIT1 * /BIT0) ) DO
         BEGIN
            CLOCKF CLOCK
         END
```

[12] Refer to Section IV, Chapter 10, for details on the construct syntax.

6.4.3 IF-THEN-ELSE

The IF-THEN-ELSE construct is for testing a condition and then performing one of two tasks, depending on the test results. The following example nests two IF-THEN-ELSE loops in a FOR loop.

```
SIMULATION

FOR I := 1 TO 16 DO
     BEGIN
         IF ( I <= 9 ) THEN            ;If I is less than or equal to 9, enable count.
             BEGIN
                 SETF CNT /RST
                 CLOCKF CLOCK
             END
         ELSE
             IF ( I < 16 ) THEN        ;If I is greater than 9 but less than 16,
                 BEGIN                 ;continue with no count.
                     SETF /CNT /RST
                     CLOCKF CLOCK
                 END
             ELSE                      ;If I is equal to 16, reset the state machine.
                 BEGIN
                     SETF RST
                     CLOCKF CLOCK
                 END
     END
```

6.5 DESIGN EXAMPLES

The following discussions illustrate how to simulate a Boolean 4-bit counter and a simple state-machine telephone answering device. The simulation commands are contained in auxiliary simulation files.

6.5.1 BOOLEAN EQUATION DESIGN

This discussion is based on simulating the basic 4-bit counter design shown below.

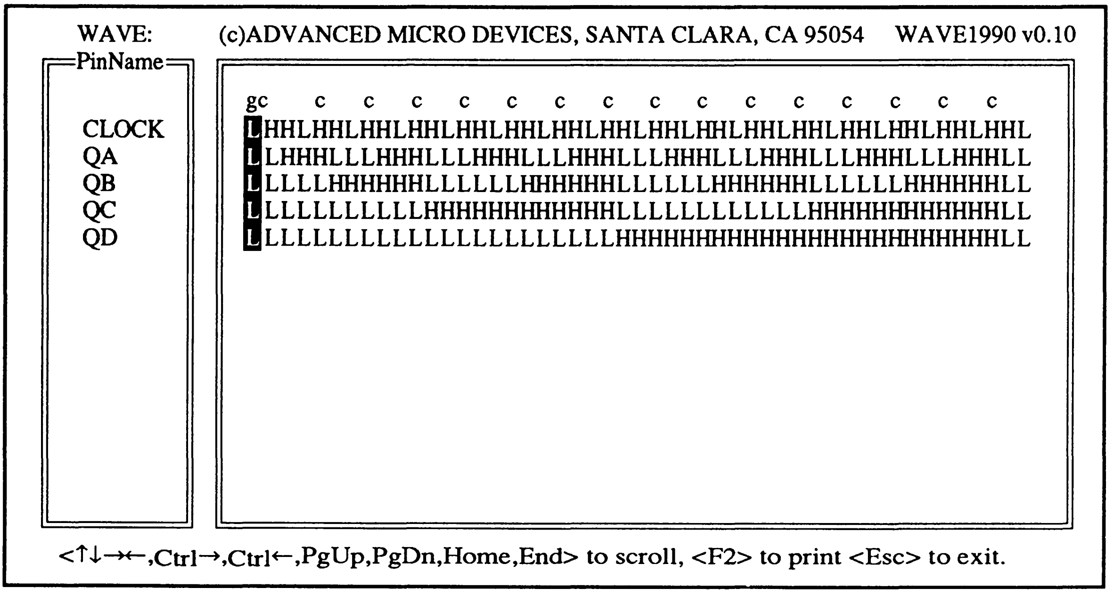
```
CHIP     _bcntr    MACH110

  PIN 2 QA                REGISTERED
  PIN 35 CLOCK
  PIN 3 QB                REGISTERED
  PIN 4 QC                REGISTERED
  PIN 5 QD                REGISTERED

EQUATIONS

 QA.T = VCC
 QA.clkf = CLOCK

 QB.T = QA
 QB.clkf = CLOCK

 QC.T = QA * QB
 QC.clkf = CLOCK

 QD.T = QC * QB * QA
 QD.clkf = CLOCK
```

To simulate this design, you can set up a FOR loop that clocks the counter 16 times, as illustrated in the auxiliary simulation file shown next.

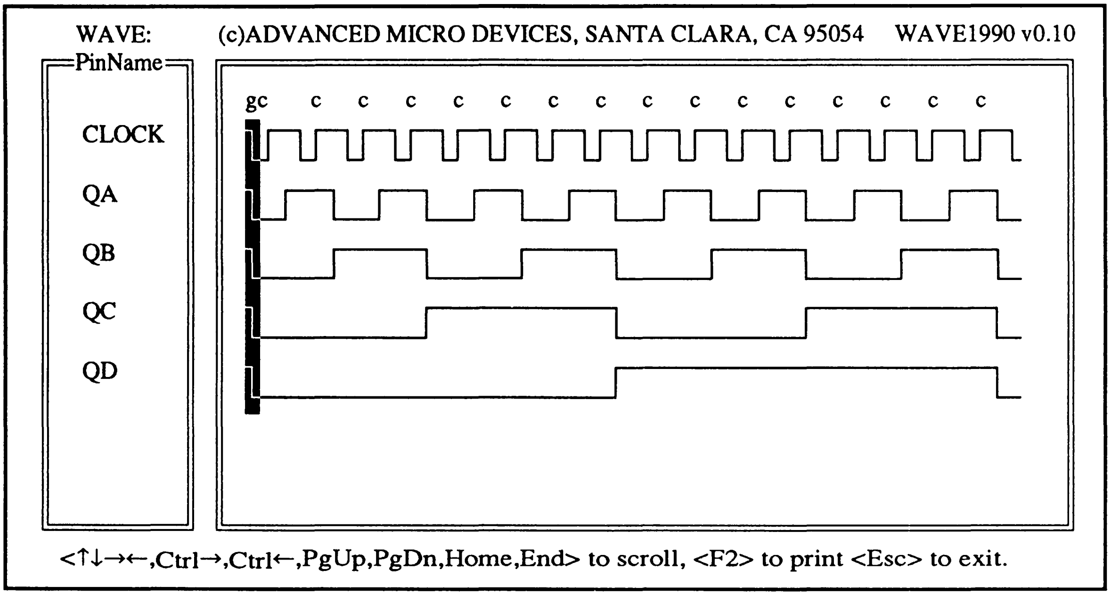
```
sIMULATION

TRACE_ON CLOCK QA QB QC QD        ;Generate a trace file with the specified signals.
SETF /CLOCK                       ;Initialize the clock signal to a logical 0.

FOR I:= 1 TO 16  DO               ;Clock the counter 16 times.  This FOR loop
     BEGIN                        ;takes the place of 16 individual Clockf commands.
       CLOCKF CLOCK
     END

TRACE_OFF                         ;Turn tracing off.
```

The simulation results are recorded in a history and a trace file. You can view either of these files in a text or a graphical mode. The trace text and waveform results are shown in the following figures.

Note: In the waveform display, the letters g and c indicate the occurrence of SETF and CLOCKF commands, respectively.



```
WAVE:     (c)ADVANCED MICRO DEVICES, SANTA CLARA, CA 95054     WAVE1990 v0.10
PinName

           gc  c  c  c  c  c  c  c  c  c  c  c  c  c  c  c
CLOCK      LHHLHHLHHLHHLHHLHHLHHLHHLHHLHHLHHLHHLHHLHHLHHLHHL
QA         LLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLL
QB         LLLLLHHHHHHLLLLLLHHHHHHLLLLLLHHHHHHLLLLLLHHHHHHLL
QC         LLLLLLLLLLLHHHHHHHHHHHHLLLLLLLLLLLLHHHHHHHHHHHHLL
QD         LLLLLLLLLLLLLLLLLLLLLLLHHHHHHHHHHHHHHHHHHHHHHHHLL

<↑↓→←,Ctrl→,Ctrl←,PgUp,PgDn,Home,End> to scroll, <F2> to print <Esc> to exit.
```



WAVE: (c)ADVANCED MICRO DEVICES, SANTA CLARA, CA 95054 WAVE1990 v0.10

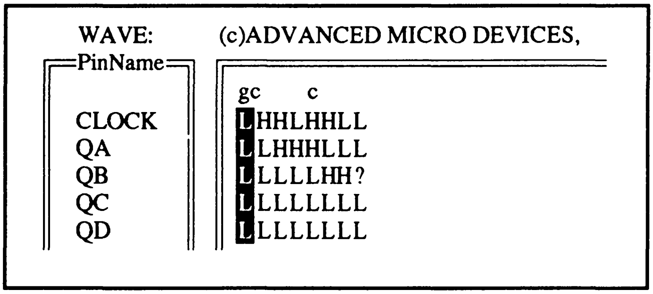
<↑↓→←,Ctrl→,Ctrl←,PgUp,PgDn,Home,End> to scroll, <F2> to print <Esc> to exit.

If you know what the simulation results should be during any portion of the simulation session, you can use the CHECK, CHECKQ, or TEST commands to automatically flag discrepancies. The following simulation purposely checks for a wrong value.

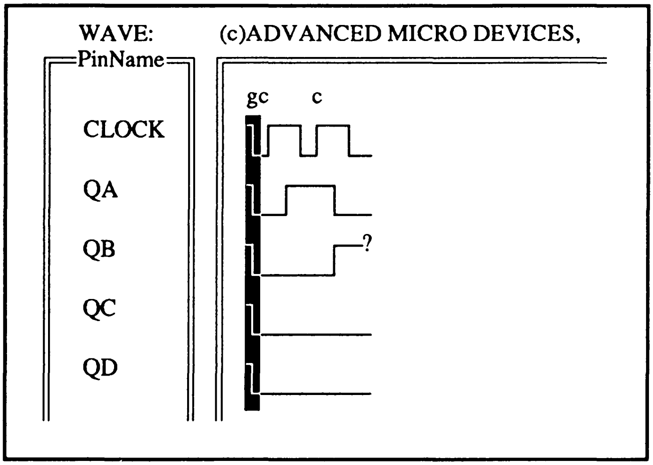
```
SIMULATION
TRACE_ON CLOCK QA QB QC QD      ;Generate a trace file with the specified signals.
SETF /CLOCK                     ;Initialize the clock signal to logical 0.
CLOCKF CLOCK                    ;Clock the counter to 0001.
CLOCKF CLOCK                    ;Clock the counter to 0010.
CHECK /QA /QB /QC /QD           ;Check for 0000, this flags a discrepancy.

TRACE_OFF                       ;Turn tracing off.
```

The simulation results mark the location of the discrepancy with a question mark, as shown in the trace text and waveform figures on the next page.

Note: In the waveform display, the letters g and c indicate the occurrence of SETF and CLOCKF commands, respectively.



```
    WAVE:       (c)ADVANCED MICRO DEVICES,
  =PinName=
                 gc     c
   CLOCK         LHHLHHLL
   QA            LLHHHLLL
   QB            LLLLLHH?
   QC            LLLLLLLL
   QD            LLLLLLLL
```



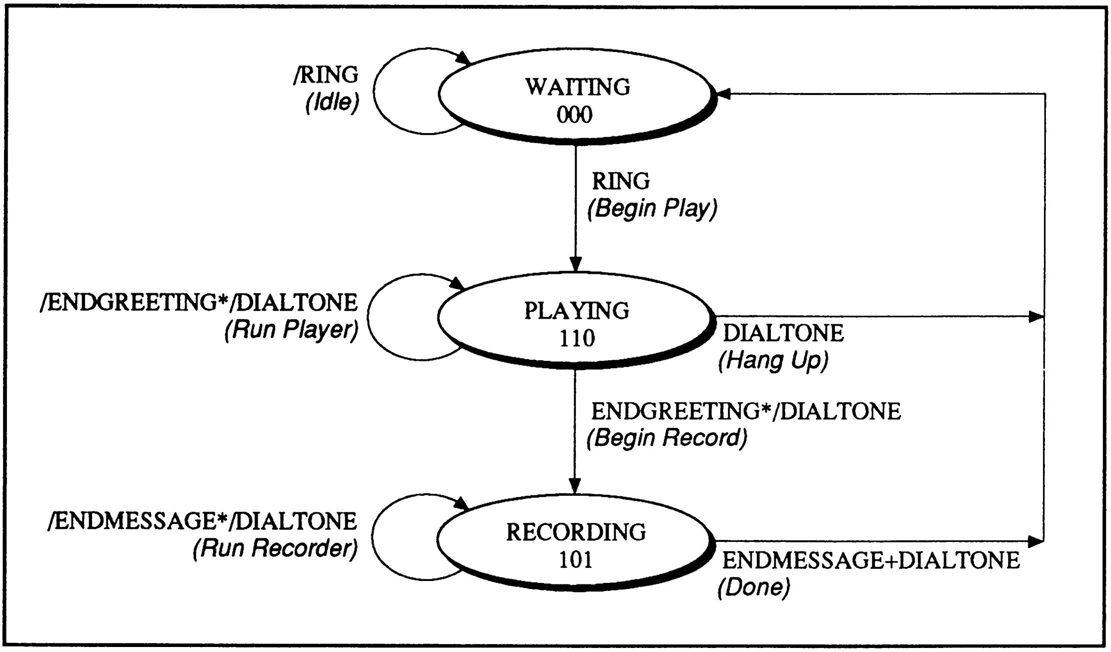
6.5.2 STATE-MACHINE DESIGN

The following state-machine design represents a simple telephone answering device. This is a Moore machine with three states: WAITING, PLAYING, and RECORDING.

```
CHIP   _ANSWER   MACH110

PIN 35 CLOCK              COMBINATORIAL          ; INPUT
PIN 32 DIALTONE           COMBINATORIAL          ; INPUT
PIN 11 RING               COMBINATORIAL          ; INPUT
PIN 13 ENDGREETING        COMBINATORIAL          ; INPUT
PIN 10 ENDMESSAGE         COMBINATORIAL          ; INPUT
PIN 4  ANSWER             REGISTERED             ; OUTPUTS
PIN 6  PLAY               REGISTERED             ; OUTPUTS
PIN 2  RECORD             REGISTERED             ; OUTPUTS

STATE
CLKF = CLOCK
MOORE_MACHINE
START_UP := POWER_UP -> WAITING

WAITING         := BEGINPLAY   -> PLAYING
                +  IDLE        -> WAITING
PLAYING         := BEGINRECORD -> RECORDING
                +  RUNPLAYER   -> PLAYING
                +  HANGUP      -> WAITING
RECORDING       := DONE        -> WAITING
                +  RUNRECORDER -> RECORDING

WAITING         = /ANSWER * /PLAY * /RECORD
PLAYING         =  ANSWER *  PLAY * /RECORD
RECORDING       =  ANSWER * /PLAY *  RECORD

CONDITIONS
IDLE                 = /RING
HANGUP             =  DIALTONE
BEGINPLAY        =  RING
RUNPLAYER       = /ENDGREETING  * /DIALTONE
BEGINRECORD    =  ENDGREETING  * /DIALTONE
RUNRECORDER   =  /ENDMESSAGE  * /DIALTONE
DONE                 =  ENDMESSAGE   +  DIALTONE
```

The state machine can be represented by the following transition diagram on the next page.

Note: Conditions are shown in parentheses.



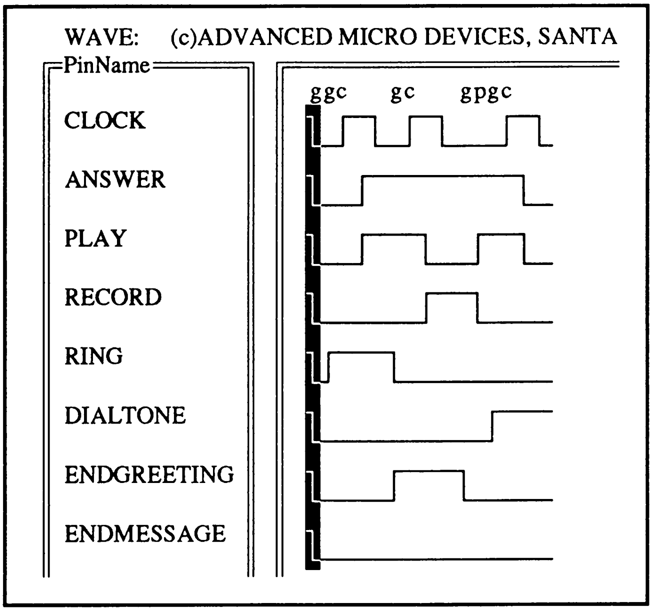
The following simulation file transitions the state machine from WAITING to PLAYING to RECORDING and then back to PLAYING.

Note: The Preload command is used to transition from RECORDING to PLAYING.

```
SIMULATION

TRACE_ON CLOCK ANSWER PLAY RECORD RING DIALTONE ENDGREETING ENDMESSAGE
SETF /CLOCK /RING /ENDGREETING /ENDMESSAGE /DIALTONE
CHECK WAITING                     ;Verify WAITING state.

SETF RING
CLOCKF CLOCK                      ;Transition to PLAYING.
CHECK PLAYING                     ;Verify PLAYING state.

SETF ENDGREETING /RING
CLOCKF CLOCK                      ;Transition to RECORDING.
CHECK RECORDING                   ;Verify RECORDING state.

SETF /ENDGREETING
PRELOAD PLAYING                   ;Preload the PLAYING state.
CHECK PLAYING                     ;Verify PLAYING state.

SETF DIALTONE
CLOCKF CLOCK                      ;Transition to WAITING.
CHECK WAITING                     ;Verify WAITING state.

TRACE_OFF                         ;Turn tracing off.
```

The simulation results are shown below.

Note: In the graphical- and text-format displays, the letters g, c, and p are used to indicate the occurrence of SETF, CLOCKF, and PRELOAD commands, respectively.


WAVE: (c)ADVANCED MICRO DEVICES, SANTA
PinName
ggc gc gpgc
CLOCK
ANSWER
PLAY
RECORD
RING
DIALTONE
ENDGREETING
ENDMESSAGE

INDEX

MACH™ Design Workbook

© 1991 Advanced Micro Devices, Inc.
901 Thompson Place
P.O. Box 3453
Sunnyvale, CA 94088

TEL: 408-732-2400
TWX: 910339-9280
TELEX: 34-6306
TOLL FREE: 800-538-8450

APPLICATIONS HOTLINE: 800-222-9323

Advanced Micro Devices reserves the right to make changes in specifications at any time and without notice. The information furnished by Advanced Micro Devices is believed to be accurate and reliable. However, no responsibility is assumed by Advanced Micro Devices for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of Advanced Micro Devices.

TABLE OF CONTENTS

PREFACE

This workbook presents three MACH™ design applications to help you gain an understanding of how to successfully compile MACH device designs.

Each design is presented in a separate chapter, as follows.

- Chapter 1 uses an 8-bit barrel shift-register design to illustrate potential problems associated with a circuit that has a large number of interconnects.
- Chapter 2 uses a 16-bit counter with a MUX to demonstrate considerations and tradeoffs associated with circuits that make heavy use of device resources.
- Chapter 3 uses a Universal Asynchronous Receiver/Transmitter (UART) design to show how grouping signals that share common resources can lead to a successful fit.

Note: The designs in this workbook have been implemented using the PALASM® 4 software; they have not been implemented in hardware.

Each chapter is organized as follows.

- A brief introduction is followed by a description of the specific MACH-device design and its device-resource requirements.

- The steps to set up your working environment and compile and fit the design are provided so you can complete them at the workstation.

> **Note**: These steps are the same in each scenario; only path and file names differ.

- An interpretation of the initial MACH fitting report[1] explores problem areas and discusses strategies and tradeoffs to improve results.

- The steps to specify the design are provided for you to complete at the workstation.

> **Note**: These steps differ in each scenario due to differences in designs and fitting problems.

- A discussion of the subsequent MACH fitting report follows the second process.

> **Note**: Additional strategies may add steps and discussions.

[1] Refer to the *PALASM 4 User's Manual*, Section II, Chapter 5, for additional details about the MACH report.

Numbered discussions within each scenario provide **tutorial steps** you can complete at the workstation. The **right column** explains what to do. Pertinent descriptions about what happens on the screen and discussions about specific options are provided.

The **left column** beside each step identifies which key you must press, what you must type, or what you must select on the screen to take appropriate action. A prompt may appear with your response to help you track the process.

AUDIENCE

The reader of this workbook is assumed to have a working knowledge of the following concepts, which are not included in this manual.

- The design of programmable logic devices
- The PALASM 4 software language syntax and design-entry procedures[2]
- MACH device architecture and performance characteristics[3]

[2] Refer to the *PALASM 4 User's Manual*, Section I, Chapter 2, which provides both text-based and schematic-based design-entry tutorials, and Section IV, Chapter 10, which provides language syntax.

[3] Refer to the *High Density EE CMOS Programmable Logic MACH 1 and MACH 2 Families Data Book* for details.

Note: Designs that require up to 70% of MACH-device resources can be achieved with very little effort. Certain designs in this workbook show that MACH-device utilizations of greater than 70% can be achieved using various combinations of language syntax and software fitting options. The degree of fit varies from design to design.

Most abbreviations are those defined as standard by the IEEE. Abbreviations unique to the PALASM 4 software and specific designs are defined at first use.

ACKNOWLEDGEMENTS

For their great help with the publication of this workbook, Advanced Micro Devices (AMD) would like to thank the following major contributors. AMD also extends thanks to the many other people, too numerous to mention here, for their help with, and commitment to the quality of, this publication.

Sharon Garner
Jana McNulty
Joe Walcek
Martin Hamilton
George Roark

Important: For answers to questions about any information in this casebook, contact a customer support representative at the AMD Applications Hot-line: 800-222-9323. The people acknowledged above are not part of the customer support group and are not available to answer questions.

8-BIT BARREL SHIFT REGISTER

A DESIGN WITH NUMEROUS INTERCONNECTS

CONTENTS

1

8-Bit Barrel Shift Register: A Design with Numerous Interconnects

The design in this example, an 8-bit barrel shift register, illustrates management of interconnects between subfunctions. Logic elements in this design share a large number of common inputs.

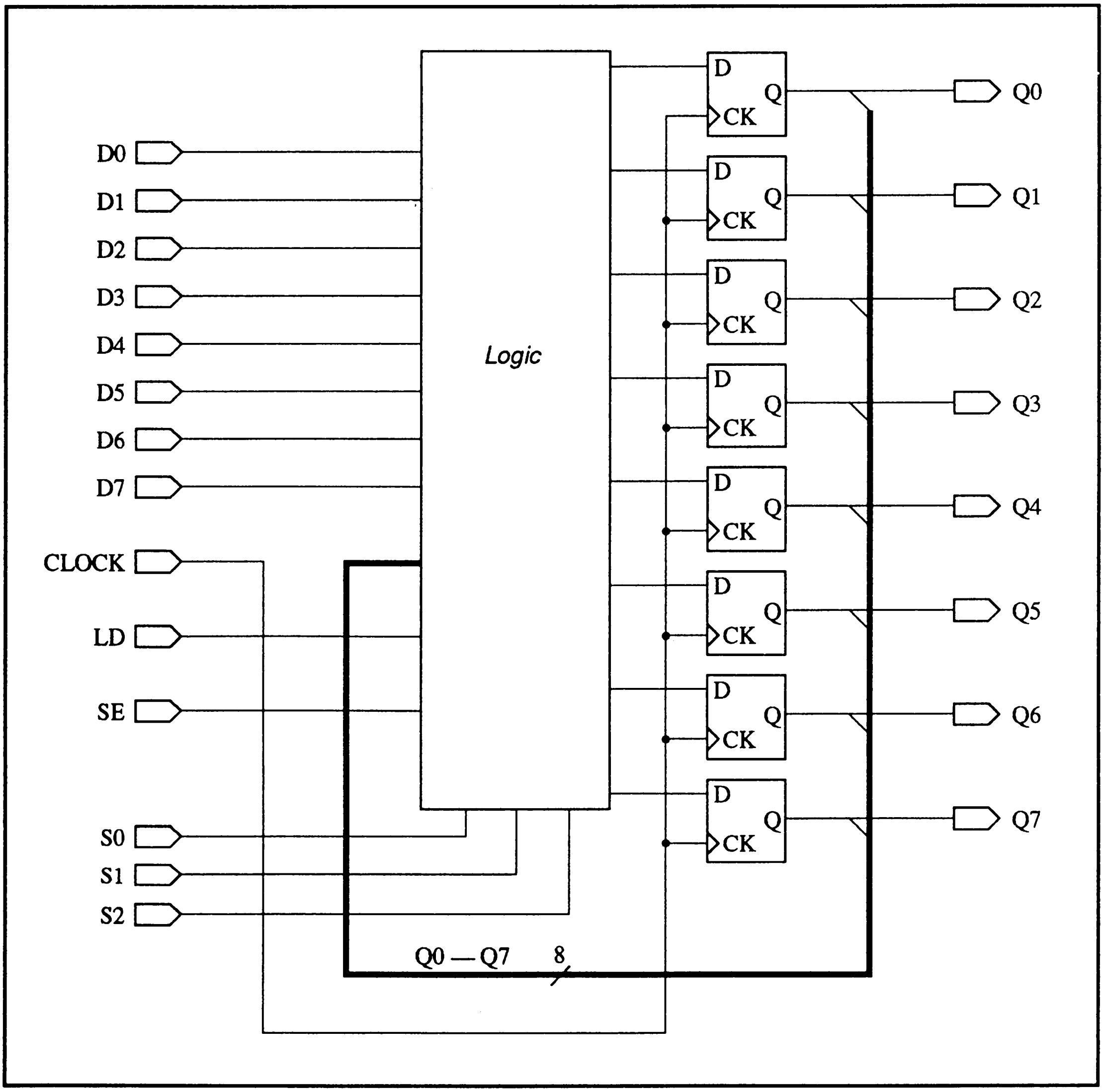
The fitting process for MACH-device designs uses specific criteria to partition logic between blocks. One such criterion is affinity, which can be defined as the measure of common inputs shared by logic elements. Yet if all logic elements show a similar or equal affinity, that criteria cannot play a significant role in partitioning.

The MACH report generated during the initial fitting process for this design provides insight to the fitting problem vis-a-vis circuit characteristics. Following report interpretation, you re-engineer the design and compile the updated version to produce a successful fit.

1.1 DESIGN DESCRIPTION

All files related to this AMD-supplied design are stored in the directory noted below. A logic diagram follows.

\PALASM\EXAMPLES\WB\BSR



When the load signal, LD, is asserted and the shift-enable signal, SE, is low the register is loaded with input data from D0 through D7. Signals S0 through S2 define the amount by which outputs Q0 through Q7 are shifted: 0 results in no shift, 7 results in full rotation.

The following table illustrates the resource requirements of the 8-bit barrel shift register and the resources available in a MACH 110 device.

RESOURCE	DESIGN REQUIREMENTS	MACH 110
Clocks	1	2
Flip-flops	8	32
Product terms	80	128
I/Os	22	44

Resource requirements can be a factor during the fitting process. However, the requirements for this design suggest it can easily fit in a MACH 110 device.

This design was implemented using Boolean equations. Part of the PDS file is shown next. Each string statement at the end of the pin declarations segment defines a single name you can use repeatedly in later equations. For example, you can enter RL1 in equations rather than entering the information RL1 represents.

Equations in this design define, among other things, signal feedback. The equation for Q0 indicates this signal uses feedback from all eight outputs: Q0 through Q7. Each of these signals, and the design inputs, must be routed through the switch matrix.

```
:
 CHIP  BRL MACH110
;-------------------------------- PIN Declarations --------------
PIN ?  CLOCK
PIN ?  D0
PIN ?  D1
PIN ?  D2
PIN ?  D3
PIN ?  D4
PIN ?  D5
PIN ?  D6
PIN ?  D7
PIN ?  Q0 REG
PIN ?  Q1 REG
PIN ?  Q2 REG
PIN ?  Q3 REG
PIN ?  Q4 REG
PIN ?  Q5 REG
PIN ?  Q6 REG
PIN ?  Q7 REG

PIN ?  S0
PIN ?  S1
PIN ?  S2
PIN ?  SE
PIN ?  LD

STRING RL1 '/S2 * /S1 * /S0 * SE'
STRING RL2 '/S2 * /S1 *  S0 * SE'
STRING RL3 '/S2 *  S1 * /S0 * SE'
STRING RL4 '/S2 *  S1 *  S0 * SE'
STRING RL5 ' S2 * /S1 * /S0 * SE'
STRING RL6 ' S2 * /S1 *  S0 * SE'
STRING RL7 ' S2 *  S1 * /S0 * SE'
STRING RL8 ' S2 *  S1 *  S0 * SE'
;-------------------------------- Boolean Equation Segment ------
EQUATIONS

Q0 = D0 * LD
   + Q0 * /SE
   + Q7 * RL1
   + Q6 * RL2
   + Q5 * RL3
   + Q4 * RL4
   + Q3 * RL5
   + Q2 * RL6
   + Q1 * RL7
   + Q0 * RL8

:
```

continued ...

```
    :

Q7 = D7 * LD
   + Q7 * /SE
   + Q6 * RL1
   + Q5 * RL2
   + Q4 * RL3
   + Q3 * RL4
   + Q2 * RL5
   + Q1 * RL6
   + Q0 * RL7
   + Q7 * RL8


Q0.CLKF = CLOCK
Q1.CLKF = CLOCK
Q2.CLKF = CLOCK
Q3.CLKF = CLOCK
Q4.CLKF = CLOCK
Q5.CLKF = CLOCK
Q6.CLKF = CLOCK
Q7.CLKF = CLOCK

Q0.SETF = GND
Q1.SETF = GND
Q2.SETF = GND
Q3.SETF = GND
Q4.SETF = GND
Q5.SETF = GND
Q6.SETF = GND
Q7.SETF = GND

Q0.RSTF = GND
Q1.RSTF = GND
Q2.RSTF = GND
Q3.RSTF = GND
Q4.RSTF = GND
Q5.RSTF = GND
Q6.RSTF = GND
Q7.RSTF = GND

;--------------------------------- Simulation Segment -----------
SIMULATION

TRACE_ON LD SE S2 S1 S0 CLOCK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7
SETF /CLOCK /SE /S0 /S1 /S2 /D0 /D1 /D2 /D3 /D4 /D5 /D6 /D7 /LD
CLOCKF CLOCK
SETF D0 LD
CLOCKF CLOCK
SETF /LD
CLOCKF CLOCK
:
```

1.2 COMPILE THE DESIGN

This discussion is divided into two procedures.

- Setup
- Compilation

Note: If you have a good understanding of the PALASM 4 software, you can use the prompts and figures on the left side of this discussion to quickly advance to discussion 1.3.

1.2.1 SETUP

The following steps guide you as you retrieve the PDS file and verify setup options to ensure they are appropriate to initially compile and fit this design.

To begin from DOS,

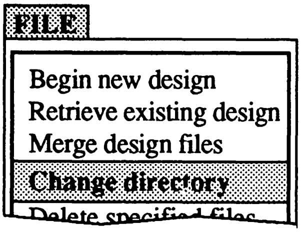
C:\
PALASM [Enter]

1. Type PALASM from the DOS prompt, then press [Enter] to run the software.

Press any key ...
[Space bar]

2. Dismiss the copyright notice and continue.

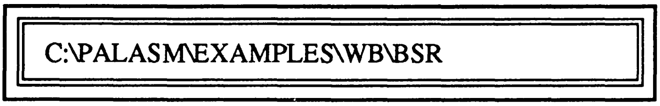
To retrieve the design,

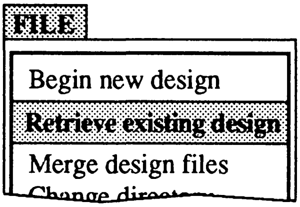


1. Select the Change directory command from the File menu.

C:\PALASM\EXAMPLES\WB\BSR [F10]

2. Type the path name shown below and confirm it.


C:\PALASM\EXAMPLES\WB\BSR

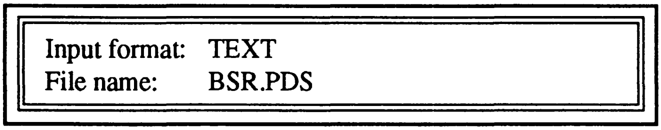


3. Select Retrieve existing design from the File menu.

[F2]
T [Enter]
BSR.PDS [F10]

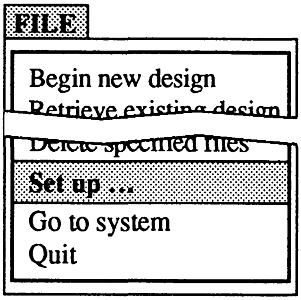
4. Display the submenu, type the letter T to select Text, activate the file name field, type the name at left and confirm.

The form on your screen should match the one below.



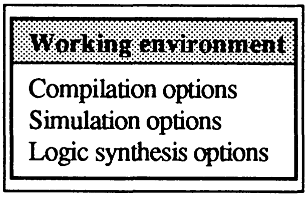
Input format: TEXT
File name: BSR.PDS

To verify the setup,



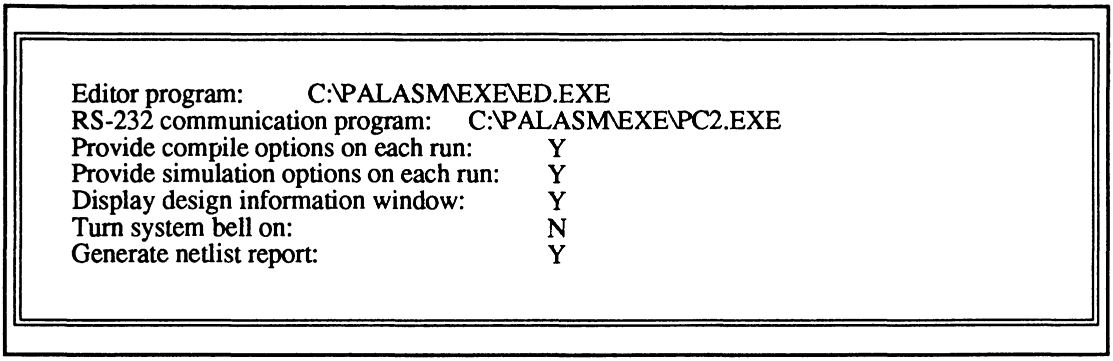
1. Select Set up from the File menu.

   A submenu opens offering four setup options.



2. Select Working environment from the submenu.

   A form appears that identifies the following.



```
Editor program:        C:\PALASM\EXE\ED.EXE
RS-232 communication program:    C:\PALASM\EXE\PC2.EXE
Provide compile options on each run:      Y
Provide simulation options on each run:   Y
Display design information window:        Y
Turn system bell on:                      N
Generate netlist report:                  Y
```

The third specification allows you to confirm compilation options immediately before the process begins.

> **Important**: If the third specification on your screen differs, change it as indicated in step 3. In any case, complete step 4.

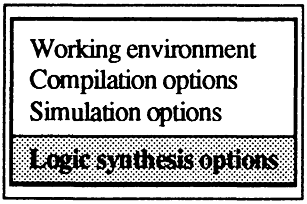
↓
Provide compile options ...
Y

3. Type the letter Y to enable compile options each time you compile.

[F10]

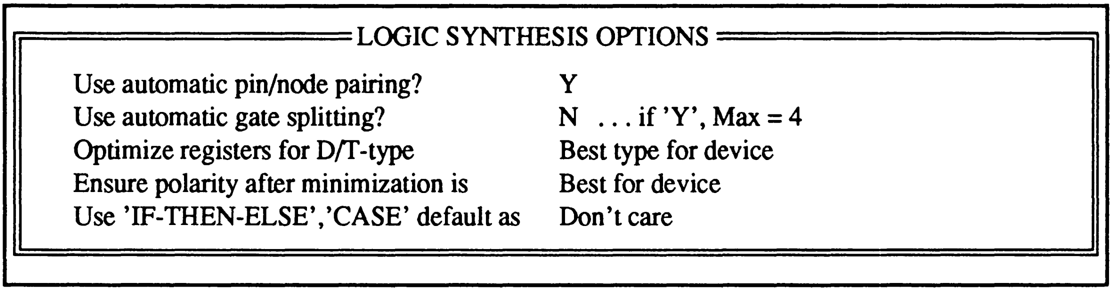
4. Confirm all specifications and dismiss the form by pressing [F10].

To verify the logic-synthesis setup,



1. Select Logic synthesis options from the submenu.

   The following form appears.



```
═══════════════ LOGIC SYNTHESIS OPTIONS ═══════════════
  Use automatic pin/node pairing?            Y
  Use automatic gate splitting?              N  . . . if 'Y', Max = 4
  Optimize registers for D/T-type            Best type for device
  Ensure polarity after minimization is      Best for device
  Use 'IF-THEN-ELSE','CASE' default as       Don't care
```

The figure above shows the options that must be specified for the first compilation of this design.

Important: If options on your screen differ, complete steps 2 through 4. In any case, complete steps 5 and 6.

Use auto ... pairing?
Y
Use auto ... splitting?
N

2. Type the correct letter to enable automatic pairing and disable gate splitting.

Optimize registers ...
[F2]
B [Enter]
Ensure polarity ...
[F2]
B [Enter]

3. Display the list of options and type the letter B to select the Best ... options for register optimization and polarity specifications, respectively.

Use 'IF-THEN-ELSE', 'CASE'...
[F2]
D

4. Display the list of options and type the letter D to select the Don't Care option for the language specification.

[F10]

5. Confirm all specifications in the form.

   The form is dismissed, any changes are recorded, and the Set up submenu is again available.

[Esc]

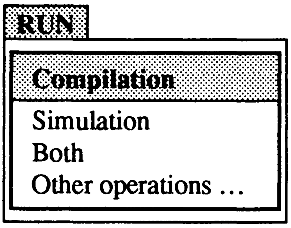
6. Dismiss the submenu.

   You're returned to the File menu; the Set up command remains highlighted.

1.2.2 COMPILATION

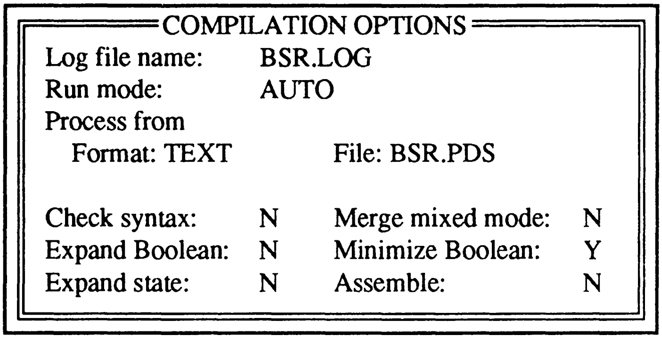
Now that you've verified basic setup options, you compile the design and interpret the report.

To begin,



1. Select Compilation from the Run menu.

   The Compilation Options form appears listing various specifications.



```
                    COMPILATION OPTIONS
Log file name:      BSR.LOG
Run mode:           AUTO
Process from
  Format: TEXT                File: BSR.PDS

Check syntax:       N         Merge mixed mode:   N
Expand Boolean:     N         Minimize Boolean:   Y
Expand state:       N         Assemble:           N
```

The specifications in the lower half of the form are used only when a manual run mode is specified. For this design, automatic run mode is used.

Important: Be sure to complete the steps below to set the execution-log name and run mode.

BSR.LOG [Enter]

2. Enter the execution-log file name shown at left.

   The Run mode field becomes active.

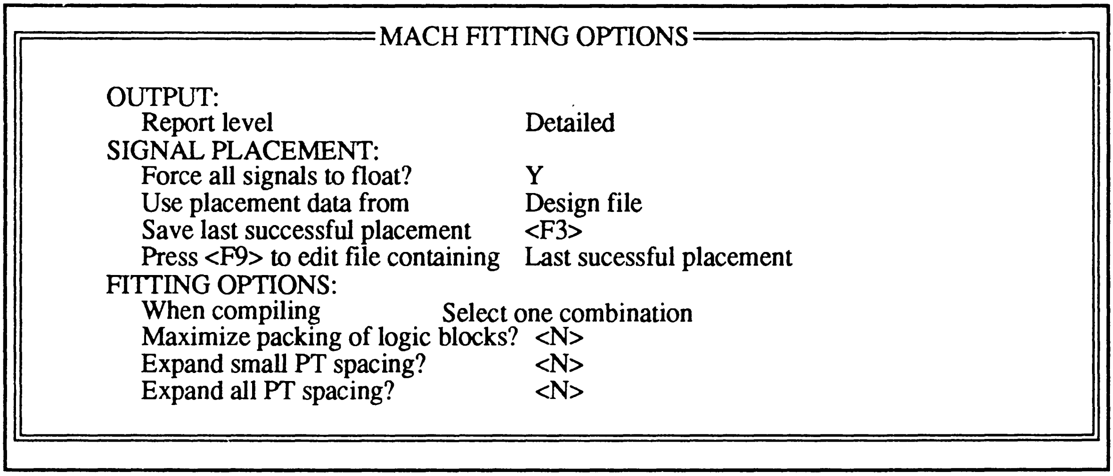
[F2]
A

3. Display the options list and type the letter A to select Automatic.

[F10]

4. Confirm specifications in the form.

   A new form appears showing the MACH fitting options to be used.



```
                    MACH FITTING OPTIONS

    OUTPUT:
       Report level                      Detailed
    SIGNAL PLACEMENT:
       Force all signals to float?       Y
       Use placement data from           Design file
       Save last successful placement    <F3>
       Press <F9> to edit file containing  Last sucessful placement
    FITTING OPTIONS:
       When compiling               Select one combination
       Maximize packing of logic blocks?  <N>
       Expand small PT spacing?           <N>
       Expand all PT spacing?             <N>
```

Important: The specifications on your screen should match the previous figure. If they differ, complete steps 5 through 8. In any event, complete steps 9 and 10.

Report level
[F2]
D [Enter]

5. Display the list of options and type the letter D to select Detailed for the report option and confirm.

Force all signals ...
Y

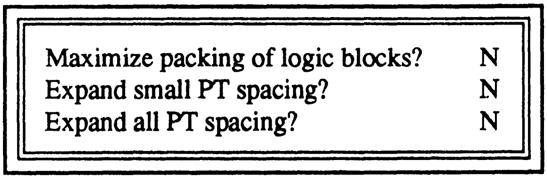
6. Type the letter Y to force all signals to float during compilation and confirm.

Use placement ...
[F2]
D [Enter] [Enter]

7. Display the list of options and type the letter D to select Design file as the placement-data option.

When compiling
[F2]
S
N
N
N [F10]

8. Display fitting options, type the letter S to select one combination, type the letter N beside each suboption, and confirm.



```
Maximize packing of logic blocks?   N
Expand small PT spacing?            N
Expand all PT spacing?              N
```

[F10]

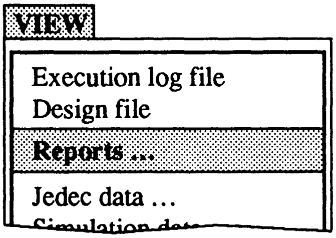
9. Confirm all specifications.

A window opens as the process begins. Upon completion, you see the fitting process was not successful.

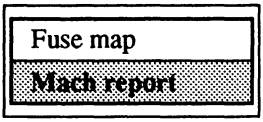
[Esc]

10. Dismiss the process window.

To review the MACH report,



1. Select the Reports command from the View menu.



2. Select the Mach report command from the submenu, then scroll through the report.

Tip: To print the report, you select the Other file command from the Edit menu, enter the name below, and print from the text editor as usual.

BSR.RPT

1.3 INTERPRET INITIAL REPORT

This discussion is divided into three parts.

- Interpretation
- Problem Summary
- Improvement Strategies

1.3.1 INTERPRETATION

Discussions here show various segments of the MACH report to explain what each reveals.

1.3.1.1 Flags Used

The flags-used portion of the report indicates which settings you specified on the MACH Fitting Options form.

```
   Flags Used:              Unplace=True              Max Packing=False
   Flags Used:        Expand Small=False               Expand All=False
```

You forced all signals to float during compilation, and this set the flag: Unplace=True. Similarly, you disabled the logic-block packing and product-term expansion options. Those flags are set to False.

Ordinarily, the pair analysis segment identifies errors caused by illegal pair declarations in pin and node statements. The preplacement segment identifies illegal and conflicting preplacement data. Neither the pair analysis nor the preplacement and equation-use segments are shown because no errors were reported for this circuit.

1.3.1.2 Timing Analysis for Signals

The timing-analysis table, shown next, indicates various kinds of delays. Listed signals have the maximum delay, which specifies the number of passes through the array. No problems are revealed here.

```
   *** Timing Analysis for Signals


Parameter    Min    Max            Signal List (Those having Max delay.)
    Tsu       1      1                   Q0               Q1                Q2
                                         Q3               Q4                Q5
                                         Q6               Q7
    Tco       0      0                   Q0               Q1                Q2
                                         Q3               Q4                Q5
                                         Q6               Q7
    Tcr       1      1                   Q1               Q2                Q3
                                         Q4               Q5                Q6
                                         Q7               Q0
   Key:
   Tpd - Combinatorial propagation delay, input to output
   Tsu - Combinatorial setup delay before clock
   Tco - Register clock to combinatorial output
   Tcr - Register thru combinatorial logic to setup
   All delay values are expressed in terms of array passes
```

1.3.1.3 Device-Resource Checks

This table shows the resources available in the selected device, those required by the design, and the percentage of resources used. You can see 57% of the pins and 74% of the product terms are used. This design's requirements are within the capacity of the MACH 110 device. Again, no problems are revealed.

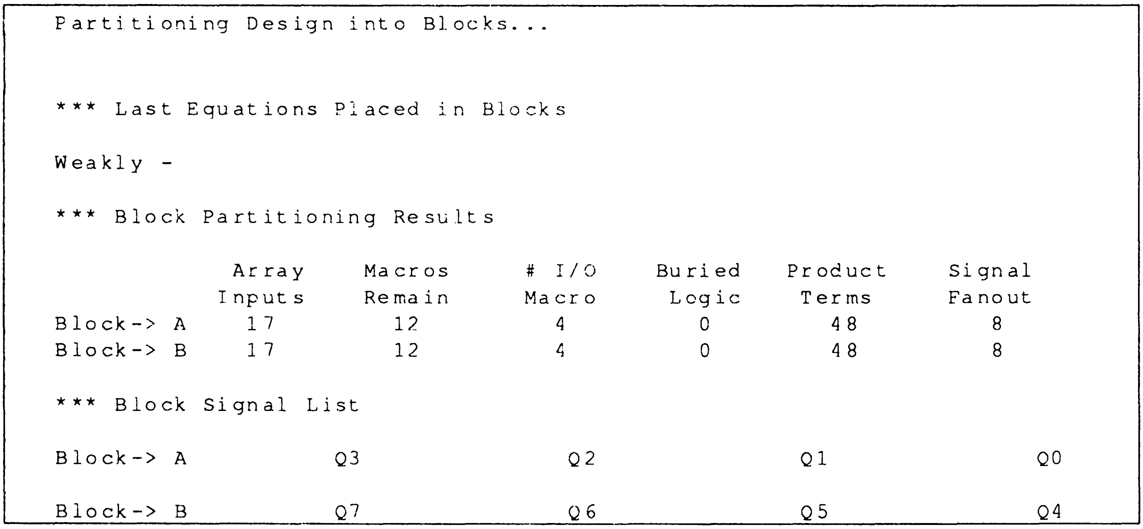
```
     *** Device Resource Checks

                     Available        Used         Remaining
           Clocks:       2              1              1
             Pins:      38             22             16       ->     57%
        I/O Macro:      32              8             24
      Total Macro:      32              8             24
    Product Terms:     128             80             32       ->     74%
      MACH-PLD Resource Checks OK!
```

1.3.1.4 Block Partitioning

This segment provides statistical information for each block, such as the number of array inputs, I/O and buried macros used, the number of product terms used, and signal fanouts.

Signals with a weak affinity to other equations in the block, meaning the signals share few common inputs, are identified under the heading Weakly. However, such is **not** the case with this design, the heading stands alone with no accompanying information.

As you can see in the block-partitioning results, the logic has been partitioned between the two MACH blocks in equal proportions. The block-signal list shows signals Q0 through Q3 were placed in block A and signals Q4 through Q7 were placed block B.



```
Partitioning Design into Blocks...


*** Last Equations Placed in Blocks

Weakly -

*** Block Partitioning Results

             Array     Macros     # I/O     Buried    Product     Signal
            Inputs     Remain     Macro      Logic     Terms      Fanout
Block-> A     17         12         4          0         48          8
Block-> B     17         12         4          0         48          8

*** Block Signal List

Block-> A           Q3              Q2              Q1              Q0

Block-> B           Q7              Q6              Q5              Q4
```

All logic elements have equal affinity. This can cause the marginal partitioning warning shown in a later segment. Further information is provided under discussions on resource assignments and the feedback map.

1.3.1.5 Device Utilization

This design requires an overall device-utilization of 68%. The MACH 110 device clearly has enough resources to accommodate this design.

```
   |> INFORMATION F050 - Device Utilization....... *: 68 %
```

1.3.1.6 Assigning Resources

Errors in the resource-assignment segment, shown next, indicate a connection problem based on wiring congestion.

```
   *** Macro Block A


   I/O Macros>             Q0              Q1              Q2              Q3
      Targets>   1( 3)   4( 6)   9(15)   12(18)

            Q0 (A  1) -> (A  1) (B  1)
            Q1 (A  4) -> (A  4) (B  4)
            Q2 (A  9) -> (A  9) (B  9)
            Q3 (A 12) -> (A 12) (B 12)

   *** Macro Block B

   I/O Macros>             Q4              Q5              Q6              Q7
      Targets>   1(25)   4(28)   9(37)   12(40)

            Q4 (B  1)?            Q5 (B  4)?            Q6 (B  9)?
            Q7 (B 12)?
            Q4 (B  1) ->   Blocked -> Reshuffling SwMtrx
            Q0 -> (A 18)    Moved.
            Q4 -> (A  1) (B 18)
            Q5 (B  4) ->   Blocked -> No Reshuffle Possible

   |> ERROR F590 - Connection problem (Wiring Congested) - Q5
            Q6 (B  9) ->   Blocked -> Reshuffling SwMtrx
            Q2 -> (A 20)    Moved.
            Q6 -> (A  9) (B 20)
            Q7 (B 12) ->   Blocked -> No Reshuffle Possible

   |> ERROR F590 - Connection problem (Wiring Congested) - Q7
```

Equal affinity increases the difficulty of dividing equations into device blocks during the fitting process. Question marks, ?, and messages in this segment indicate several partitioning attempts. When you interpret this information with circuit characteristics and the warning at the end of the report, you see that marginal partitioning, while not inherently bad, is the probable cause.

The inputs are routed when partitioning is finished. In this design, output signals were assigned to macros in close proximity to each other during partitioning. A number of input and feedback signals, required to generate the outputs, were not routed due to wiring congestion. Error F590 was issued to alert you to the fact there are no connection points available.

1.3.1.7 Signals, Tabular

The information in this table can be useful when you want to minimize inter-block connections. The following information is provided.

- Signal names and corresponding pin numbers
- Pin and node numbers
- Macrocell block and cell location
- Pin type
- Logic type: D-type, T-type, Comb, etc.
- Number of product terms used for outputs
- The block(s) driven by each signal

In this design, ten product terms are required for each Qx signal and all Qx signals must be routed through both blocks of the device. The question marks, ?, in the location field indicate that two signals, Q5 and Q7, are not placed.

```
*** Signals - Tabular Information
         Signal    #    P/N #   (Loc)        Type      Logic  # PT    Blocks
         CLOCK     1     35     I  5      clock pin      .
            D0     2     26     B  2         input       .              A
            D1     3     28     B  4         input       .              A
            D2     4     29     B  5         input       .              A
            D3     5     31     B  7         input       .              A
            D4     6     36     B  8         input       .               B
            D5     7     38     B 10         input       .               B
            D6     8     41     B 13         input       .               B
            D7     9     43     B 15         input       .               B
            Q0    10      3     A  1       i/o pin     d-ff     10      AB
            Q1    11      6     A  4       i/o pin     d-ff     10      AB
            Q2    12     15     A  9       i/o pin     d-ff     10      AB
            Q3    13     18     A 12       i/o pin     d-ff     10      AB
            Q4    14     25     B  1       i/o pin     d-ff     10      AB
            Q5    15      0      .?.       i/o pin     d-ff     10      AB
            Q6    16     37     B  9       i/o pin     d-ff     10      AB
            Q7    17      0      .?.       i/o pin     d-ff     10      AB
            S0    18     10     I  0         input       .              AB
            S1    19     11     I  1         input       .              AB
            S2    20     13     I  2         input       .              AB
            SE    21     33     I  4         input       .              AB
            LD    22     24     B  0         input       .              AB
```

1.3.1.8 Signals, Equations

The equations segment, shown next, provides a comprehensive list of source signals. The signals driven by each source appear in the fanout list. Block information is enclosed in braces.

In the fanout list you can see each output, Q0 through Q7, drives all other outputs and itself.

Though not the case with this design, output signals that are not fed back through the array would be listed as outputs with no feedback at the end of this segment.

```
Signal Source                   Fanout List

       CLOCK
          D0:             Q0
            {A}
          D1:             Q1
            {A}
          D2:             Q2
            {A}
          D3:             Q3
            {A}
          D4:             Q4
            {B}
          D5:             Q5
            {.}
          D6:             Q6
            {B}
          D7:             Q7
            {.}
          Q0:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.}

          Q1:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.}

          Q2:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.}
          Q3:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.}

          Q4:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.}

          Q5:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.B}

          Q6:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.}

          Q7:           Q0            Q1            Q2            Q3
            :           Q4            Q5            Q6            Q7
            {AAAA B.B.}
:
```

1.3.1.9 Feedback Map

The feedback map, shown next, graphically identifies how input and output signals are routed through the switch matrix to the array to drive other outputs. Macrocells are numbered within each block.

```
*** Feedback Map - BARREL SHIFT REGISTER


Gbl Inp .--.      I/O  .--+--A--+--.  I/O               I/O   .--+--B--+--.  I/O
        | 0|       D3 : 0|      |21| SE                      | 0|      |21| SE
        | 1|       Q4 : 1|      |20| Q2               Q0 : 1|      |20| Q6
        | 2|       D2 : 2|      |19| S2                      | 2|      |19| S2
        | 3|       D1 : 3|      |18| Q0                      | 3|      |18| Q4
        | 4|       Q1 : 4|      |17| S1               Q1 : 4|      |17| S1
        | 5|       D0 : 5|      |16| S0                      | 5|      |16| S0
        '--'            | 6|      |15|                       | 6|      |15: D4
                   LD : 7|      |14|                  LD : 7|      |14|
                        | 8|      |13|                  D7 : 8|      |13: D5
                   Q6 : 9|      |12: Q3               Q2 : 9|      |12: Q3
                        |10|      |11|                  D6 :10|      |11|
                        '--+-u--u+--'                        '--+-u--u+--'
```

Numbers on the feedback map correspond to macrocells in switch-matrix blocks.

- The switch-matrix feeds the PAL arrays.
- The PAL arrays feed macros through the logic allocator.

The feedback map indicates that, as expected, outputs Q0 through Q4 and Q6, are feedback to drive signals in both blocks A and B. Feedback from outputs Q5 and Q7 could not be routed. You can review equations in the PDS file to determine which output signals are used as feedback.

1.3.1.10 Logic Map

The logic map, shown next, indicates placement of the output signals in MACH blocks. All outputs shown require ten product terms; asterisks indicate product-term steering. Q5 and Q7 could not be placed due to routing congestion.

Product terms (PTs) are used in clusters of four. Three clusters of four product terms each, or a total of 12 PTs, are required for a signal that requires ten product terms. In such cases, automatic product-term steering allocates terms from adjacent macrocells within the same MACH block.

```
*** Logic Map - BARREL SHIFT REGISTER


Gbl Inp .--.      I/O  .--+--A--+--.  I/O          I/O  .--+--B--+--.  I/O
     S0| 0|            | 0| *    |21|                   | 0| *    |21|
     S1| 1|       Q0   | 1|10    |20|              Q4   | 1|10    |20|
     S2| 2|            | 2| *    |19|                   | 2| *    |19|
       | 3|            | 3| *    |18|                   | 3| *    |18|
     SE| 4|       Q1   | 4|10    |17|                   | 4| *    |17|
  CLOCK| 5|            | 5| *    |16|                   | 5| *    |16|
       '--'            | 6| .   .|15|                   | 6| .   .|15|
                       | 7| .   .|14|                   | 7| .   .|14|
                       | 8| *   *|13|                   | 8| *   *|13|
                  Q2   | 9|10 10|12|  Q3          Q6   | 9|10   *|12|
                       |10| *   *|11|                   |10| *   *|11|
                       '--+-u--u+--'                   '--+-u--u+--'
```

1.3.1.11 Design Report, Errors and Warnings

The final messages in the report indicate the name of this report and the number of errors and warnings. In addition, a separate warning about marginal partitioning is mentioned. No pin-out map could be generated.

```
   The Design Doc is  stored in ===> BSR.Rpt
   |> WARNING F120 - Marginal Block Partitioning Measure:  (Utilizations)
   %% FITR %% Error Count: 2, Warning Count: 1
   %% FITR %% File Processing Terminated. - File: BSR (10 nc)
```

1.3.2 PROBLEM SUMMARY

Two problems were revealed during report interpretation.

- Wiring congestion, evident in the resource-assignment segment of the MACH report, was indicated for a number of Qx signals.
- Too many inter-block connections, noted in the tabular signals segment of the MACH report, were indicated for several Qx signals.

Ten product terms are required for each equation and each equation must be routed to both blocks.

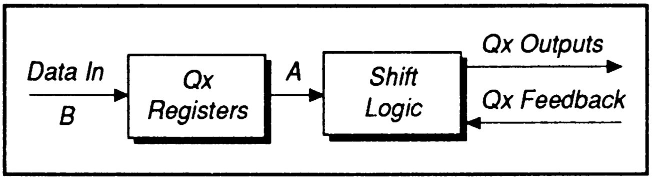
The logic in this circuit has many common inputs. Equal affinity results and negates one of the partitioning criteria used during the fitting process. In this case, marginal partitioning resulted in blocked wiring paths and several feedback signals could not be routed.

1.3.3 STRATEGIES TO REDUCE CONGESTION AND INTER-BLOCK CONNECTIONS

Not every equation can have a large number of product terms and a large number of wiring channels from block to block. The challenge here is to reduce the number of equations or find a way to reduce the amount of inter-block wiring. In fact, an approach that provides insight to other designs can be found if you associate the equations with the actual MACH architecture.

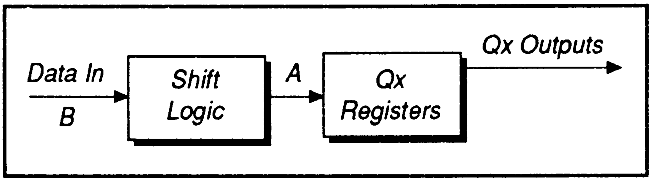
In this design, Qx outputs must be routed to both blocks, as seen in the tabular signals segment of the MACH fitting report. The logic array for each block feeds the Qx registers in that block, which practically eliminates the possibility of wiring congestion. Consequently, the congestion that causes the fitting failure must be inter-block congestion resulting from Qx outputs in one block going to a different block.

A good solution to reduce the wiring congestion is to reduce requirements for Qx output routing. To gain an understanding of the requirements for Qx output routing, you can look at the equations that define the Qx signals. For example, shift-function equations are expressed in terms of Qx outputs: Q0=Q7*RL1 (Rotate left 1). If you convert this equation to its architectural equivalent, shown next, you get a picture of a bank of flip-flops feeding a shift network that feeds back to the registers.



Area A in the figure above represents an area already congested by fuses and various combinations of output and feedback polarity options. Area B has only input signals. None of the register options are used at the input macrocells.

If you reverse the architecture, as shown next, a less congested wiring-channel is achieved. In the new architecture, there is no feedback at all. Input data is shifted and loaded to the correct register initially.



1.4 RE-ENGINEER THE DESIGN AND COMPILE

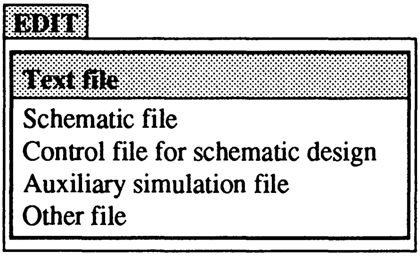
Follow the steps below to re-engineer the design and compile it using the same options you used initially.

Note: A copy of the re-engineered design file is supplied on the PALASM 4 installation diskettes under the name BSR_INPT.PDS. If you don't want to make the design edits, simply skip to discussion 1.4.1 and compile the BSR_INPT.PDS file.

1.4.1 RE-ENGINEER THE DESIGN

The following changes move the shift operation from the output side of the function to the input side.

To edit the PDS file,



1. Select the Text file command from the Edit menu.

   The PDS file appears on the screen.

To change product terms associated with shifting from Q-based to D-based terms in equations, you make the following kinds of changes to equations in the PDS file.

```
Q0 = D0 * LD * /SE
   + Q0 * /SE * /LD
   :
   + D0 * RL0
   ...
```

2. Edit the third and following lines of each equation, as shown at left, then compare your changes with the **partial** equations segment shown next.

```
[Esc]
   F
   C
New filename: <Esc = abort>
   new_name.PDS [Enter]
```

3. Change the name of the new file so you can retain the original design file.

```
[Esc]
   Q
   X
Save Changes (Y/N-<CR> for Yes)?
   [Enter]
```

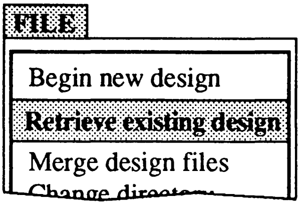
4. Save the file and quit from the editor to PALASM 4.

```
;---------------------------------- Boolean Equation Segment ------
EQUATIONS

Q0 = D0 * LD
   + Q0 * /SE
   + D7 * RL1
   + D6 * RL2
   + D5 * RL3
   + D4 * RL4
   + D3 * RL5
   + D2 * RL6
   + D1 * RL7
   + D0 * RL8

Q1 = D1 * LD
   + Q1 * /SE
   + D0 * RL1
   + D7 * RL2
   + D6 * RL3
   + D5 * RL4
   + D4 * RL5
   + D3 * RL6
   + D2 * RL7
   + D1 * RL8

Q2 = D2 * LD
   + Q2 * /SE
   + D1 * RL1
   + D0 * RL2
   + D7 * RL3
   + D6 * RL4
   + D5 * RL5
   + D4 * RL6
   + D3 * RL7
   + D2 * RL8

:

Q7 = D7 * LD
   + Q7 * /SE
   + D6 * RL1
   + D5 * RL2
   + D4 * RL3
   + D3 * RL4
   + D2 * RL5
   + D1 * RL6
   + D0 * RL7
   + D7 * RL8
:
```

1.4.1 COMPILE THE UPDATED DESIGN

Next, you retrieve and compile the updated design using the same logic-synthesis and fitting options as you did before. The log file name will differ.

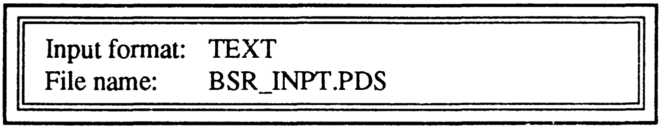
Note: A re-engineered version of the 8-bit barrel shift register design is stored as BSR_INPT.PDS, and is referenced in the following discussion.



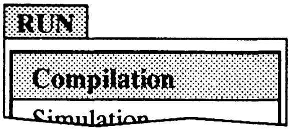
1. Select Retrieve existing design from the File menu.

↓
BSR_INPT.PDS [F10]

2. Activate the file-name field, type the name at left, and confirm.



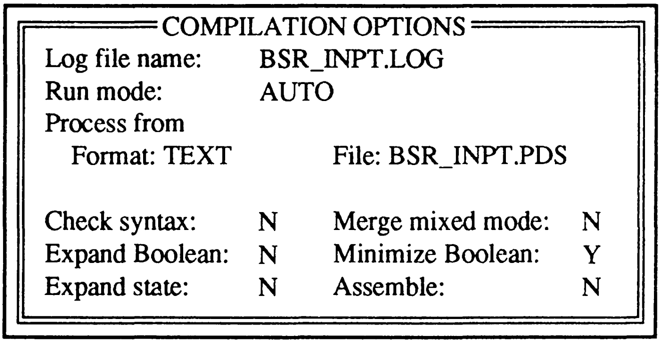
```
Input format:  TEXT
File name:     BSR_INPT.PDS
```



3. Select Compilation from the Run menu.

BSR_INPT.LOG [F10]

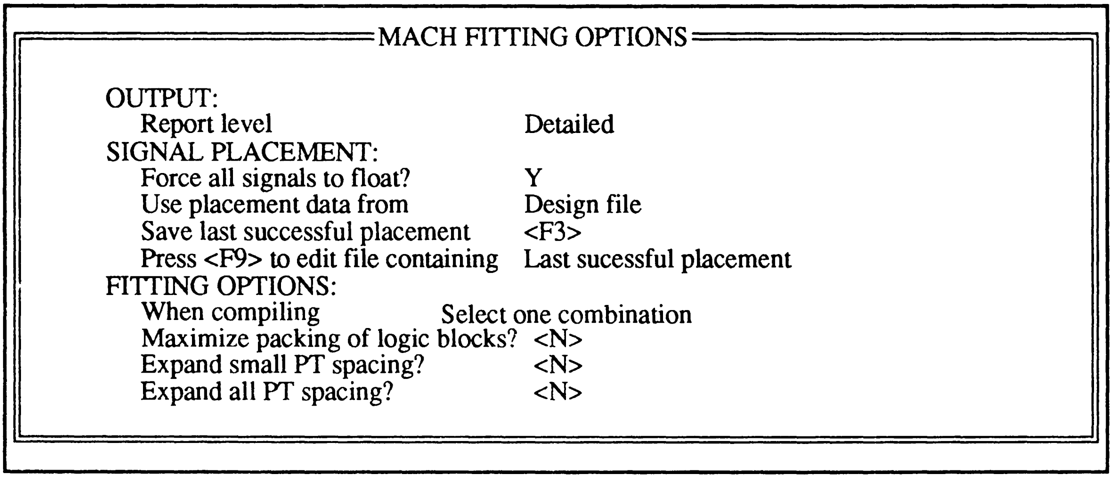
4. Change the execution-log file name as shown.



```
═══════════ COMPILATION OPTIONS ═══════════
Log file name:     BSR_INPT.LOG
Run mode:          AUTO
Process from
  Format: TEXT             File: BSR_INPT.PDS

Check syntax:      N    Merge mixed mode:   N
Expand Boolean:    N    Minimize Boolean:   Y
Expand state:      N    Assemble:           N
```

[F10]

5. Confirm MACH Fitting Options, shown next.



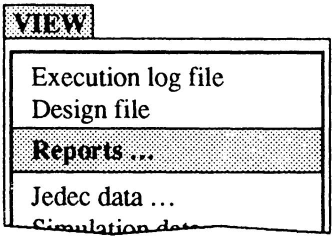
```
MACH FITTING OPTIONS

OUTPUT:
    Report level                        Detailed
SIGNAL PLACEMENT:
    Force all signals to float?         Y
    Use placement data from             Design file
    Save last successful placement      <F3>
    Press <F9> to edit file containing  Last sucessful placement
FITTING OPTIONS:
    When compiling               Select one combination
    Maximize packing of logic blocks?  <N>
    Expand small PT spacing?           <N>
    Expand all PT spacing?             <N>
```

This time the process is successful.

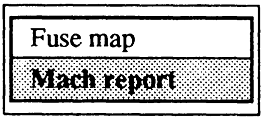
[Esc]

6. Dismiss the process window.

To review the report,



1. Select the Reports command from the View menu.



2. Select Mach report from the submenu.

Tip: Again, you can select the Other file command from the Edit menu, enter the name below, and print the report from the text editor as usual.

BSR_INPT.RPT

1.5 INTERPRET FINAL REPORT

Discussions here provide an interpretation of relevant segments of the MACH report and a conclusion.

1.5.1 INTERPRETATION

As you might expect, most areas of the report show no significant difference when compared with the previous one. However, some sections merit a look.

1.5.1.1 Timing Analysis for Signals

In the timing-analysis table shown next, the Tcr delay has been eliminated. In the updated design, data enters from bit 0 and moves directly to Qx and out of the register rather than looping back through.

```
   *** Timing Analysis for Signals


 Parameter    Min    Max            Signal List (Those having Max delay.)
      Tsu      1      1                  Q0              Q1              Q2
                                         Q3              Q4              Q5
                                         Q6              Q7
      Tco      0      0                  Q0              Q1              Q2
                                         Q3              Q4              Q5
                                         Q6              Q7
   Key:
   :
```

1.5.1.2 Block Partitioning

The only change in block partitioning results is the signal fanout for each block, which has been reduced from eight per block to four per block.

```
   Partitioning Design into Blocks...



   :
   *** Block Partitioning Results

                Array     Macros      # I/O     Buried    Product     Signal
               Inputs     Remain      Macro      Logic     Terms      Fanout
   Block-> A     17         12          4          0         48          4
   Block-> B     17         12          4          0         48          4
   :
```

1.5.1.3 Assigning Resources

All signals are placed in both blocks. Wiring congestion, seen in this segment for the previous design iteration, has been eliminated. Overall device utilization remains at 68%.

```
*** Macro Block Inputs

      Inputs>             S0           S1            S2           LD           D0
     Targets>   0(10)   1(11)   2(13)   3(32)   4(33)

           S0 (I  0) -> (A 16) (B 16)
           S1 (I  1) -> (A 17) (B 17)
           S2 (I  2) -> (A 19) (B 19)
           LD (I  3) -> (A 20) (B 20)
           D0 (I  4) -> (A 21) (B 21)

*** Macro Block A
 I/O Macros>              Q0           Q1            Q2           Q3
     Targets>   1( 3)   4( 6)   9(15)  12(18)

           Q0 (A  1) -> (A  1)
           Q1 (A  4) -> (A  4)
           Q2 (A  9) -> (A  9)
           Q3 (A 12) -> (A 12)

*** Macro Block B
 I/O Macros>              Q4           Q5            Q6           Q7
     Targets>   1(25)   4(28)   9(37)  12(40)

           Q4 (B  1) -> (B  1)
           Q5 (B  4) -> (B  4)
           Q6 (B  9) -> (B  9)
           Q7 (B 12) -> (B 12)
      Inputs>             D1           D2            D3           D4
                          SE           D5            D6           D7
     Targets>   0(24)   2(26)   3(27)   5(29)   6(30)   7(31)   8(36)
               10(38)  11(39)  13(41)  14(42)  15(43)
```

1.5.1.4 Signals, Tabular

You can see a redistribution of signals driving blocks in the segment shown next. All Dx signals now drive both blocks A and B. The Qx signals now drive either block A or block B and all signals are placed. Product-term use for Qx signals has not changed.

```
*** Signals - Tabular Information
        Signal    #    P/N #    (Loc)       Type      Logic   # PT     Blocks
        CLOCK     1     35       I  5    clock pin      .
           D0     2     33       I  4        input      .               AB
           D1     3     24       B  0        input      .               AB
           D2     4     26       B  2        input      .               AB
           D3     5     29       B  5        input      .               AB
           D4     6     31       B  7        input      .               AB
           D5     7     38       B 10        input      .               AB
           D6     8     41       B 13        input      .               AB
           D7     9     43       B 15        input      .               AB
           Q0    10      3       A  1      i/o pin    d-ff     10       A
           Q1    11      6       A  4      i/o pin    d-ff     10       A
           Q2    12     15       A  9      i/o pin    d-ff     10       A
           Q3    13     18       A 12      i/o pin    d-ff     10       A
           Q4    14     25       B  1      i/o pin    d-ff     10        B
           Q5    15     28       B  4      i/o pin    d-ff     10        B
           Q6    16     37       B  9      i/o pin    d-ff     10        B
           Q7    17     40       B 12      i/o pin    d-ff     10        B
           S0    18     10       I  0        input      .               AB
           S1    19     11       I  1        input      .               AB
           S2    20     13       I  2        input      .               AB
           SE    21     36       B  8        input      .               AB
           LD    22     32       I  3        input      .               AB
 .
 :
```

1.5.1.5 Signals, Equations

This segment shows a redistribution of signals driven by each source. The fanout list for Dx and Qx signals has changed considerably. In fact, it's reversed from the previous distribution because of the design changes.

```
  Signal Source                   Fanout List


        CLOCK
        D0:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        D1:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        D2:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        D3:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        D4:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        D5:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        D6:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        D7:          Q0           Q1           Q2           Q3
          :          Q4           Q5           Q6           Q7
          {AAAA BBBB}
        Q0:          Q0
          {A}
        Q1:          Q1
          {A}
        Q2:          Q2
          {A}
        Q3:          Q3
          {A}
        Q4:          Q4
          {B}
        Q5:          Q5
          {B}
        Q6:          Q6
          {B}
        Q7:          Q7
          {B}
  :
```

1.5.1.6 Feedback Map

In the feedback map, shown next, you can see how design changes have impacted the way in which input and output signals are routed through the switch matrix to the array to drive other outputs.

```
*** Feedback Map - BARREL SHIFT REGISTER


Gbl Inp .--.      I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
        | 0|       D4 : 0|      |21| D0              D4 : 0|      |21| D0
        | 1|       Q0 : 1|      |20| LD              Q4 : 1|      |20| LD
        | 2|       D3 : 2|      |19| S2              D3 : 2|      |19| S2
        | 3|          | 3|      |18|                    | 3|      |18|
        | 4|       Q1 : 4|      |17| S1              Q5 : 4|      |17| S1
        | 5|       D2 : 5|      |16| S0              D2 : 5|      |16| S0
        '--'          | 6|      |15: SE                 | 6|      |15: SE
                   D1 : 7|      |14|                 D1 : 7|      |14|
                   D7 : 8|      |13: D5              D7 : 8|      |13: D5
                   Q2 : 9|      |12: Q3              Q6 : 9|      |12: Q7
                   D6 :10|      |11|                 D6 :10|      |11|
                       '--+-u--u+--'                     '--+-u--u+--'
```

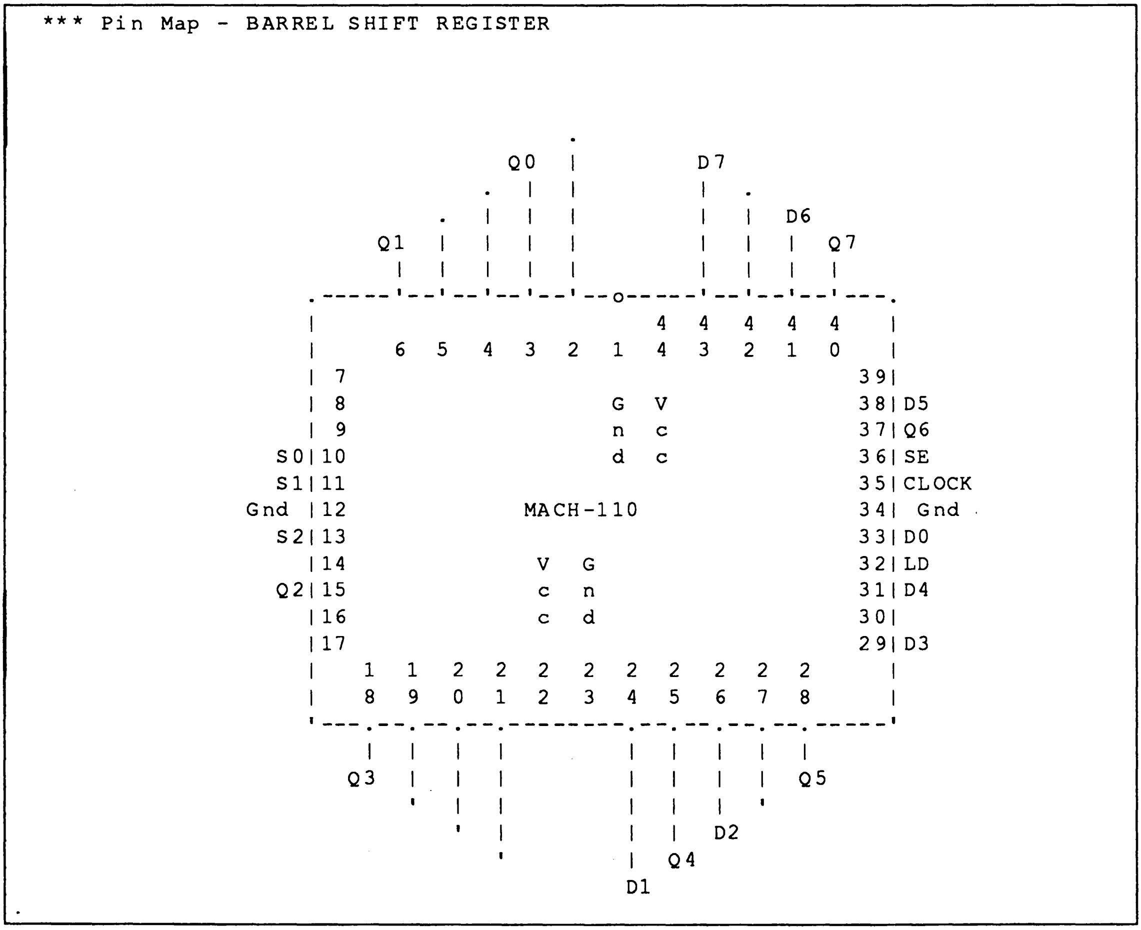
1.5.1.7 Logic Map

The logic map includes Q5 and Q7. Each output signal still requires ten product terms, which are automatically allocated from adjacent macrocells as needed.

```
*** Logic Map - BARREL SHIFT REGISTER


Gbl Inp .--.      I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
      S0| 0|          | 0| *    |21|                    | 0| *    |21|
      S1| 1|       Q0 | 1|10    |20|                 Q4 | 1|10    |20|
      S2| 2|          | 2| *    |19|                    | 2| *    |19|
      LD| 3|          | 3| *    |18|                    | 3| *    |18|
      D0| 4|       Q1 | 4|10    |17|                 Q5 | 4|10    |17|
   CLOCK| 5|          | 5| *    |16|                    | 5| *    |16|
        '--'          | 6| .   .|15|                    | 6| .   .|15|
                      | 7| .   .|14|                    | 7| .   .|14|
                      | 8| *   *|13|                    | 8| *   *|13|
                   Q2 | 9|10  10|12| Q3              Q6 | 9|10  10|12| Q7
                      |10| *   *|11|                    |10| *   *|11|
                      '--+-u--u+--'                     '--+-u--u+--'
```

1.5.1.8 Device Pin-Out Map

The device pin-out map is generated only when a successful fit occurs. The following pin-out segment is shown near the end of the MACH fitting report.



```
*** Pin Map - BARREL SHIFT REGISTER



                                    .
                                Q0  |        D7
                              .  |  |        |  .
                           .  |  |  |        |  |  D6
                       Q1  |  |  |  |        |  |  |  Q7
                        |  |  |  |  |        |  |  |  |
                  .-----'--'--'--'--'--o-----'--'--'--'---.
                  |                       4  4  4  4  4   |
                  |     6  5  4  3  2  1  4  3  2  1  0   |
                  | 7                                   39|
                  | 8                  G  V             38|D5
                  | 9                  n  c             37|Q6
                S0|10                  d  c             36|SE
                S1|11                                   35|CLOCK
              Gnd |12           MACH-110                34| Gnd
                S2|13                                   33|D0
                  |14             V  G                  32|LD
                Q2|15             c  n                  31|D4
                  |16             c  d                  30|
                  |17                                   29|D3
                  |   1  1  2  2  2  2  2  2  2  2  2     |
                  |   8  9  0  1  2  3  4  5  6  7  8     |
                  '---.--.--.--.--------.--.--.--.--.-----'
                      |  |  |  |        |  |  |  |  |
                     Q3  |  |  |        |  |  |  |  Q5
                         '  |  |        |  |  |  '
                            '  |        |  |  D2
                               '        |  Q4
                                        D1
```

1.5.2 CONCLUSION

The logic in this design includes many common inputs. The initial MACH report indicated wiring congestion and routing failure.

You surmised that reversing the architecture to eliminate feedback could achieve less congested wiring channels. Input data would be shifted and loaded to the correct register initially.

To accomplish this you moved the shift operation from the output side of the function to the input side by changing all product terms associated with shifting from Q-based to D-based terms in equations.

No change in MACH fitting options was required to achieve a successful fit after re-engineering the design.

This concludes this example.

Two 8-Bit Counters with a MUX

A Design with High Device-Resource Requirements

CONTENTS

2

Two 8-Bit Counters with a MUX: A Design with High Device-Resource Requirements

This chapter uses two 8-bit counters with a MUX to illustrate how a design with high resource requirements, such as pins, I/O macros, and product terms, is fit into a MACH device.

The design in this example has an overall device-utilization of 95%.[1] You initially compile the design without specific fitting options. During MACH-report interpretation, you gain insight into the partitioning and subsequent routing problems caused by tightly packed blocks and, in this particular design, outputs placed adjacent to each other. You then enable specific fitting options one at a time to alleviate these obstacles and achieve a fit.

[1] Designs that require up to 70% of MACH-device resources can be achieved with very little effort. This example shows that MACH-device utilizations of greater than 70% can be achieved using various combinations of language syntax and software fitting options. The degree of fit varies from design to design.

2.1 DESIGN DESCRIPTION

Files related to this AMD-supplied design are stored in the directory noted below.

\PALASM\EXAMPLES\WB\CNTMUX

The logic diagram for this design, shown opposite, includes two individual 8-bit counters followed by a MUX. Thirty two inputs feed the MUX as follows.

- 16 external inputs feed the MUX through two buses, I0 through I7 and I8 through I15.

- 16 counter outputs feed the MUX through two busses, Q0 through Q7 and Q8 through Q15.

Outputs from the MUX, O0 through O15, connect the circuit to the outside world. Signal SEL is the control signal to the MUX.

- When SEL is high, the inputs I0 - I15 are routed to the multiplexer outputs.

- When SEL is low, the counter outputs Q0 - Q15 are routed to the multiplexer outputs.

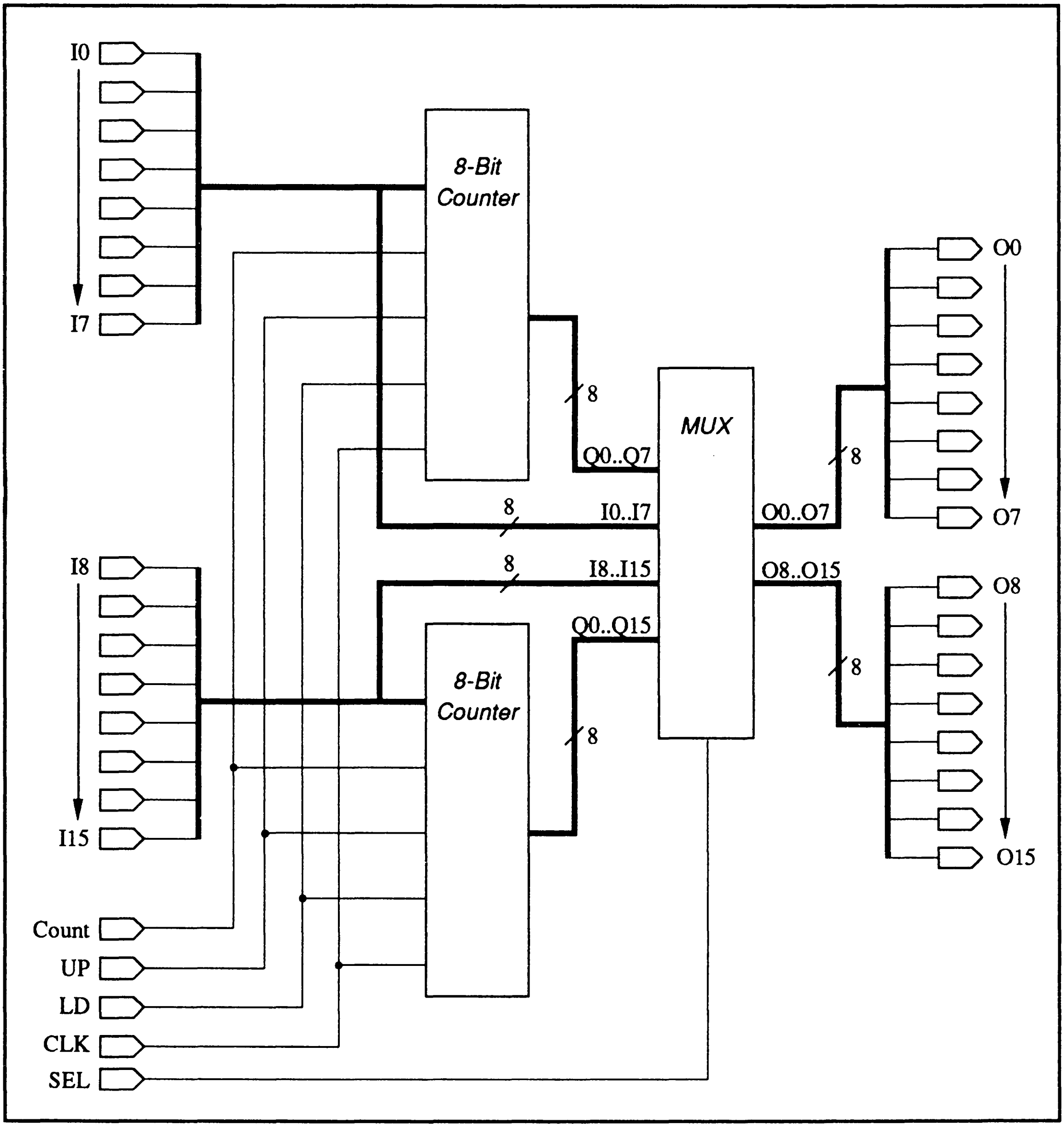
When the load-enable signal, LD, is asserted, the counters are loaded with data from I0 through I7 and I8 through I15. Counters are enabled when the Count control signal is asserted. The counter counts up when control-signal UP is high and down when UP is low.

2.1.1 PINS

A MACH 110 device supports 38 signal pins. This design uses 37 of the pins as follows.

- 21 input pins are required: I0 through I15, CLK, SEL, COUNT, UP, and DOWN.

- 16 output pins are required: O0 through 15.



Two 8-bit Counters with a MUX

2.1.2 MACROCELLS

The MACH 110 provides 32 output macrocells, which can be used either as I/O macrocells or as buried macrocells for internal signals. This design uses all 32 macrocells as follows.

- The output signals from the MUX, O0 through O15, require 16 I/O macrocells.
- Each counter requires eight buried macrocells, for a total of 16, for signals Q0 through Q7 and Q8 through Q15, respectively.

2.1.3 PRODUCT TERMS

The MACH 110 device provides 128 product terms. As currently implemented, each of the 16 equations for counter outputs Q0 through Q15 requires four product terms for a total of 64. The 16 equations for MUX outputs O0 through O15 require two product terms each, for a total of 32.

You can see in the PDS file, shown in part next, node statements specify registered signals that feed back into the circuit. These are not connected to external I/Os. Certain equations include the .T notation to specify the use of a T-type flip-flop.

```
:
CHIP CNTMUX MACH110
pin 1 gnd
pin 44 vcc
pin 35 CLK
;------------------------------- PIN Declarations --------------
PIN  ?        COUNT              ;
PIN  ?        UP                           ;
PIN  ?        LOAD               ;
PIN  ?        SELECT             ;
PIN  ?        I0                 ;
:
PIN  ?        I15                          ;
PIN  ?        O0              COMBINATORIAL ;
:
PIN  ?        O15             COMBINATORIAL ;
NODE ?        Q0               REGISTERED ;
:
NODE ?        Q15             REGISTERED ;
;-------------------------------- Boolean Equation Segment ------
EQUATIONS
Q0.T := COUNT
    + LOAD * I0 * /Q0
    + LOAD * /I0 * Q0
Q0.CLKF=CLK
Q0.RSTF=GND
Q0.SETF=GND

Q1.T := COUNT * UP * Q0
    + COUNT * /UP * /Q0
    + LOAD * I1 * /Q1
    + LOAD * /I1 * Q1
Q1.CLKF=CLK
Q1.RSTF=GND
Q1.SETF=GND


:
Q15.T := COUNT * UP * Q8 * Q9 * Q10 * Q11 * Q12 * Q13 *Q14
    + COUNT * /UP * /Q8 * /Q9 * /Q10 * /Q11 * /Q12 * /Q13 * /Q14
    + LOAD * I15 * /Q15
    + LOAD * /I15 * Q15
Q15.CLKF=CLK
Q15.RSTF=GND
Q15.SETF=GND

O0 = SELECT * I0
  + /SELECT * Q0
:
O15 = SELECT * I15
  + /SELECT * Q15
;-------------------------------- Simulation Segment -----------
:
```

2.2 COMPILE THE DESIGN

This discussion is divided into two procedures.

- Setup
- Compilation

Note: If you have a good understanding of the PALASM 4 software, you can use the prompts and figures on the left side of this discussion to quickly advance to discussion 2.3.

2.2.1 SETUP

The following steps guide you as you retrieve the PDS file and verify setup options to ensure they are appropriate to initially compile and fit this design.

To begin from DOS,

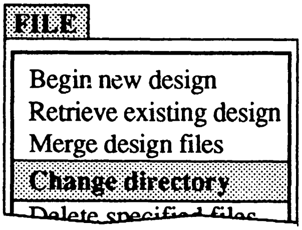
C:\
PALASM [Enter]

1. Type PALASM at the DOS prompt, then press [Enter] to run the software.

Press any key ...
[Space bar]

2. Dismiss the copyright notice and continue.

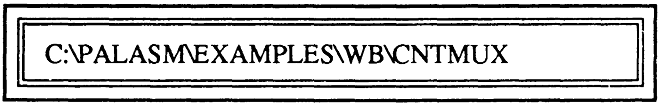
To retrieve the design,



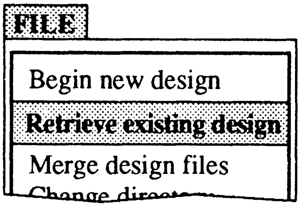
1. Select the Change directory command from the File menu.

C:\PALASM\EXAMPLES\WB\CNTMUX [F10]

2. Type the path name shown below and confirm it.



C:\PALASM\EXAMPLES\WB\CNTMUX

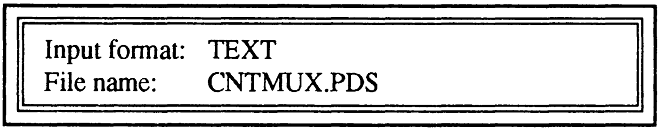


3. Select Retrieve existing design from the File menu.

[F2]
T [Enter]
CNTMUX.PDS [F10]

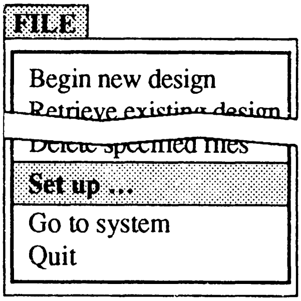
4. Display the submenu and select Text, activate the file-name field, type the name at left and confirm.

The form on your screen should match the one below.



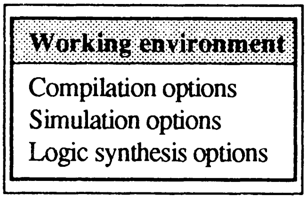
Input format: TEXT
File name: CNTMUX.PDS

To verify the setup,



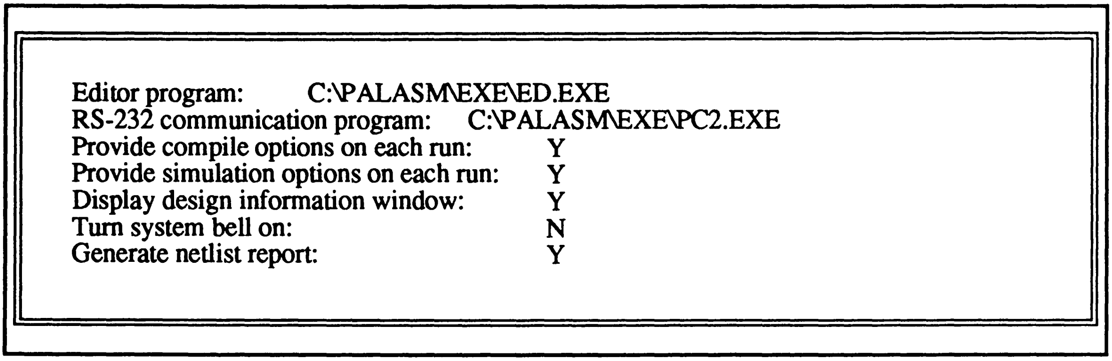
1. Select Set up from the File menu.

   A submenu opens offering four setup options.



2. Select Working environment from the submenu.

   A form appears that identifies the following.



```
Editor program:          C:\PALASM\EXE\ED.EXE
RS-232 communication program:    C:\PALASM\EXE\PC2.EXE
Provide compile options on each run:      Y
Provide simulation options on each run:   Y
Display design information window:        Y
Turn system bell on:                      N
Generate netlist report:                  Y
```

The third specification allows you to confirm compilation options immediately before the process begins.

> **Important**: If the third specification on your screen differs, change it as indicated in step 3. In any case, complete step 4.

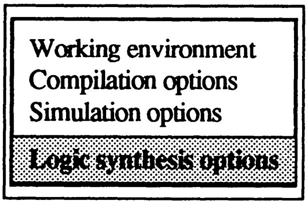
↓
Provide compile options ...
Y

3. Change the compile options specification if needed.

[F10]

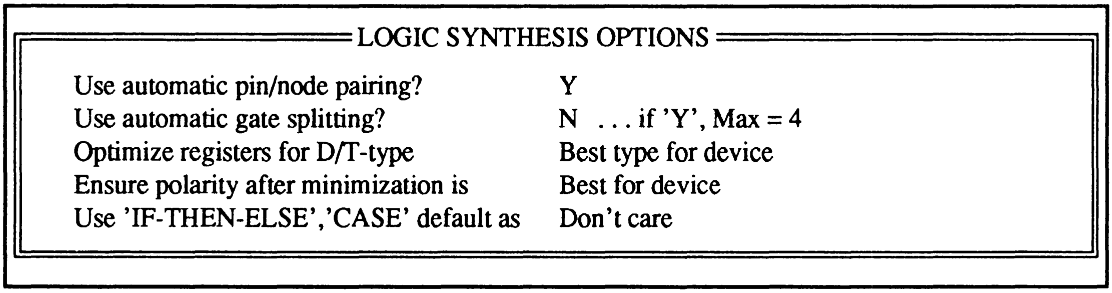
4. Confirm all specifications and dismiss the form by pressing [F10].

To verify the logic-synthesis setup,



1. Select Logic synthesis options from the submenu.

   The following form appears.



```
LOGIC SYNTHESIS OPTIONS
Use automatic pin/node pairing?          Y
Use automatic gate splitting?            N  . . . if 'Y', Max = 4
Optimize registers for D/T-type          Best type for device
Ensure polarity after minimization is    Best for device
Use 'IF-THEN-ELSE','CASE' default as     Don't care
```

The figure above shows the options that must be specified for the first compilation of this design.

Important: If options on your screen differ, complete steps 2 through 4. In any case, complete steps 5 and 6.

Use auto ... pairing?
Y
Use auto ... splitting?
N

2. Type the correct letter to enable automatic pairing and disable gate splitting.

Optimize registers ...
[F2]
B [Enter]
Ensure polarity ...
[F2]
B [Enter]

3. Display the list of options and type the letter B to select the Best ... options for register optimization and polarity specifications, respectively.

Use 'IF-THEN-ELSE', 'CASE'...
[F2]
D

4. Display the list of options and type the letter D to select the Don't Care option for the language specification.

[F10]

5. Confirm all specifications in the form.

   The form is dismissed, any changes are recorded, and the Set up submenu is again available.

[Esc]

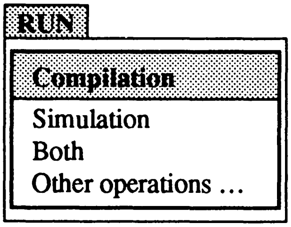
6. Dismiss the submenu.

   You're returned to the File menu; the Set up command remains highlighted.

2.2.2 COMPILATION

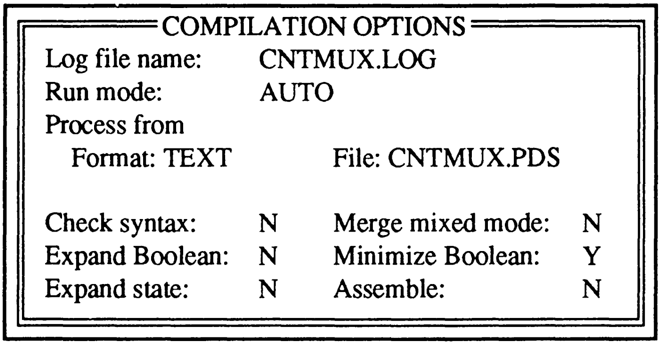
Now that you've verified basic setup options, you compile the design and interpret the report.

To begin,



1. Select Compilation from the Run menu.

   The Compilation Options form appears listing various specifications.



```
COMPILATION OPTIONS
Log file name:     CNTMUX.LOG
Run mode:          AUTO
Process from
  Format: TEXT            File: CNTMUX.PDS

Check syntax:      N      Merge mixed mode:    N
Expand Boolean:    N      Minimize Boolean:    Y
Expand state:      N      Assemble:            N
```

The specifications in the lower half of the form are used only when a manual run mode is specified. For this design, automatic run mode is used.

Important: Be sure to complete the steps below to set the execution-log name and run mode.

CNTMUX.LOG [Enter]

2. Enter the execution-log file name shown at left.

   The Run mode field becomes active.

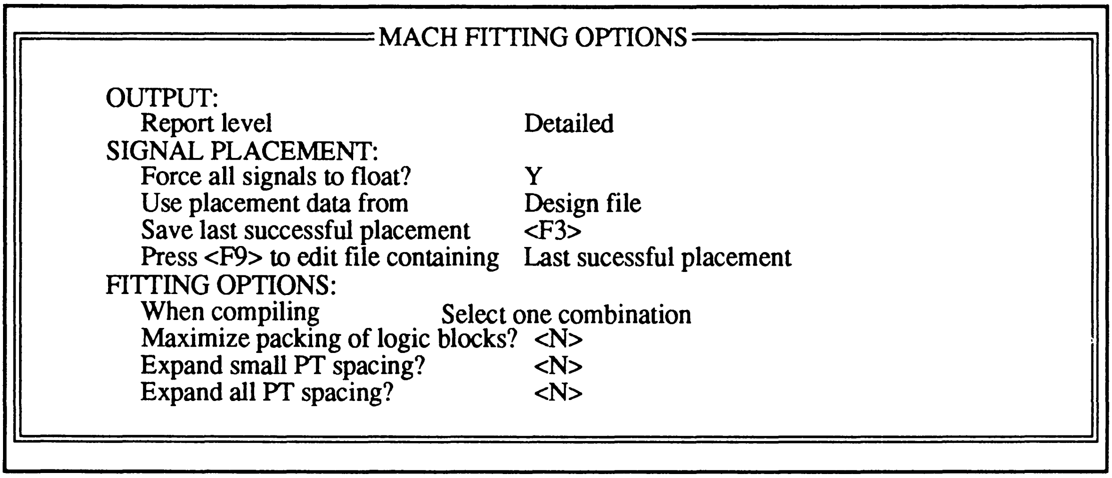
[F2]
A

3. Display the options list and select Automatic.

[F10]

4. Confirm specifications in the form.

   A new form appears showing the MACH fitting options to be used.



```
MACH FITTING OPTIONS

OUTPUT:
   Report level                        Detailed
SIGNAL PLACEMENT:
   Force all signals to float?         Y
   Use placement data from             Design file
   Save last successful placement      <F3>
   Press <F9> to edit file containing  Last sucessful placement
FITTING OPTIONS:
   When compiling                 Select one combination
   Maximize packing of logic blocks?  <N>
   Expand small PT spacing?            <N>
   Expand all PT spacing?              <N>
```

Important: The specifications on your screen should match the previous figure. If they differ, complete steps 5 through 8. In any event, complete steps 9 and 10.

Report level
[F2]
D [Enter]

5. Display the list of options and type the !etter D to select Detailed for the report option.

Force all signals ...
Y

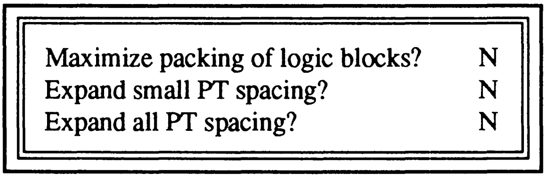
6. Type the letter Y to force all signals to float during compilation.

Use placement ...
[F2]
D [Enter] [Enter]

7. Display the list of options and type the letter D to select Design file as the placement data option.

When compiling
[F2]
S
N
N
N [F10]

8. Display fitting options, type the letter S to select one combination, type the letter N beside each suboption, and confirm.



```
Maximize packing of logic blocks?    N
Expand small PT spacing?             N
Expand all PT spacing?               N
```

[F10]

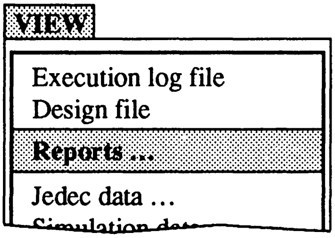
9. Confirm all specifications.

A window opens as the process begins. Upon completion, you see the fitting process was not successful.

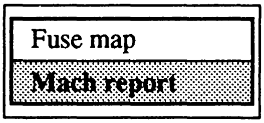
[Esc]

10. Dismiss the process window.

To review the MACH report,



1. Select the Reports command from the View menu.



2. Select the Mach report command from the submenu, then scroll through the report.

Tip: To print the report, you select the Other file command from the Edit menu, enter the name below, and print from the text editor as usual.

CNTMUX.RPT

2.3 INTERPRET INITIAL REPORT

This discussion is divided into three parts.

- Interpretation
- Problem Summary
- Improvement Strategies

2.3.1 INTERPRETATION

Discussions below explain and show what various segments of the MACH report reveal about this design and the unsuccessful fitting process.

2.3.1.1 Flags Used

The flags-used segment indicates which settings you specified on the MACH Fitting Options form.

```
   Flags Used:              Unplace=True                    Max Packing=False
   Flags Used:        Expand Small=False                     Expand All=False
```

You forced all signals to float during compilation , and this set the flag: Unplace=True. Similarly, you disabled the logic-block packing and product-term expansion options. Those flags are set to False.

Ordinarily, the pair analysis segment identifies errors caused by illegal pair declarations in pin and node statements. The preplacement segment identifies illegal and conflicting preplacement data. Neither segment is shown because no errors were reported for this circuit.

2.3.1.2 Timing Analysis for Signals

The timing-analysis table, shown next, indicates various kinds of delays. All signals listed have the maximum delay, which specifies the number of passes through the array. Signals in this design do not exceed one array-pass delay. This segment offers no clue about the unsuccessful fit.

```
 *** Timing Analysis for Signals


Parameter   Min   Max          Signal List (Those having Max delay.)
     Tpd     1     1                O0             O1              O2
                                    O3             O4              O5
                                    O6             O15
     Tsu     1     1                Q0             Q1              Q2
                                    Q3             Q4              Q5
                                    Q6             Q15
     Tco     1     1                O0             O1              O2
                                    O3             O4              O5
                                    O6             O15
     Tcr     1     1                Q1             Q2              Q3
                                    Q4             Q5              Q6
                                    Q7             Q15
  .
  Key:
   Tpd - Combinatorial propagation delay, input to output
   Tsu - Combinatorial setup delay before clock
   Tco - Register clock to combinatorial output
   Tcr - Register thru combinatorial logic to setup
   All delay values are expressed in terms of array passes
```

2.3.1.3 Device-Resource Checks

The device-resource table indicates available resources in the selected device, those required by the design, and the percentage of utilization. As expected, this segment of the report, shown next, indicates an extremely high use of device resources.

- Pin utilization 97%
- Product-term utilization 100%

Strictly speaking, the number of product terms required for this design is 96. Product terms (PTs) are automatically allocated in clusters of four. Two clusters of four PTs each, or a total of eight PTs, are allocated for a signal that requires six product terms. As a result, this design requires less than the maximum number of PTs yet none remain due to automatic product-term steering.

```
   *** Device Resource Checks


              Available        Used        Remaining
        Clocks:    2              1             1
          Pins:   38             37             1      ->     97%
     I/O Macro:   32             16            16
   Total Macro:   32             32             0
 Product Terms:  128             94             0      ->    100%

   MACH-PLD Resource Checks OK!
```

2.3.1.4 Block Partitioning

The partitioning segment of the report is shown next. It provides several groups of information.

Last Equations Placed In Blocks is divided into two groups, Weakly and Assign, that identify signal affinity, if any. Though not relevant to this design, information here can help you determine which signals to remove if you need to re-engineer the design.

Block Partitioning Results provides statistics about the placement of logic in MACH blocks, such as the number of array inputs, I/O and buried macros used, the number of product terms used, and signal fanouts. As you can see, most logic is partitioned between the two MACH blocks in equal proportions. However, some logic has not been placed at all, as confirmed by error messages generated at the end of this process.

Block Signal List identifies which signals were placed into the specified MACH blocks.

Errors and warnings appear if any occur during partitioning. This design needs all 32 macrocells. Though all available inputs and product terms are used, signals O6, O7, O14, O15, were not placed. Certain macros may be unused as a result of the disabled logic-packing option. As suggested, enabling the packing option may improve results.

```
Partitioning Design into Blocks...

*** Last Equations Placed in Blocks
Weakly -           O0          O1          O2          O3
Assign -           O4          O5          O6          O7

*** Block Partitioning Results
            Array    Macros    # I/O    Buried   Product   Signal
            Inputs   Remain    Macro    Logic    Terms     Fanout
  Unplaced->  9       12        4        0        16        9
Block-> A    20        2        6        8        56        8
Block-> B    20        2        6        8        56        8

*** Block Signal List
Block-> A          O5           O4           O3           O2
                   O1           O0           Q0           Q1
                   Q2           Q3           Q4           Q5
                   Q7           Q6

Block-> B          O13          O12          O11          O10
                    O9           O8           Q8           Q9
                   Q10          Q11          Q12          Q13
                   Q15          Q14
|> ERROR F580 - Partitioning could not place all signals into blocks!
    Signals:  O15  O14   O7   O6
|> WARNING F110 - Blk A full! (all available Inputs used)
!> WARNING F110 - Blk A full! (all available PTs used)
|> WARNING F110 - Blk B full! (all available Inputs used)
|> WARNING F110 - Blk B full! (all available PTs used)
Try Using Max Packing Density Option
```

2.3.1.5 Device Utilization

This segment indicates an overall device utilization of 95%. The resource assignment area is **not shown**. Though it provides information related to internal processing, it does not reveal any fitting problems.

```
|> INFORMATION F580 - Device Utilization....... *: 95 %
```

2.3.1.6 Signals, Tabular

The information in this table can be useful when you want to minimize inter-block connections. The following information is provided.

- Signal names and corresponding pin numbers
- Pin and node numbers
- Macrocell block and cell location
- Pin type
- Logic type: D-type, T-type, Comb, etc.
- Number of product terms used for outputs
- The block(s) driven by each signal

Each question mark, ?, in the pin-location field indicates an unplaced signal. Each at-sign, @, in the block field indicates part of an equation was not placed in the block, which can be helpful for manual placement and redesign strategies.

You can see in the segment shown next, pin placement data is not available because the process terminated before completion.

```
*** Signals - Tabular Information

        Signal    #   P/N #   (Loc)      Type      Logic  # PT   Blocks
           CLK    1    35      I  5   clock pin      .
         COUNT    2     0      .?.       input       .             AB
            UP    3     0      .?.       input       .             AB
          LOAD    4     0      .?.       input       .             AB
        SELECT    5     0      .?.       input       .            @AB
            I0    6     0      .?.       input       .             A
             :
            I7   13     0      .?.       input       .            @A
            I8   14     0      .?.       input       .              B
             :
           I15   21     0      .?.       input       .            @ B
            O0   22     0      .?.     i/o pin     comb      2
             :
            O7   29     0      .?.     i/o pin     comb      2
            O8   30     0      .?.     i/o pin     comb      2
             :
           O15   37     0      .?.     i/o pin     comb      2
            Q0   38     0      .?.      buried     d-ff      3     A
             :
            Q7   45     0      .?.      buried     t-ff      4    @A
            Q8   46     0      .?.      buried     d-ff      3      B
             :
           Q15   53     0      .?.      buried     t-ff      4    @ B
```

2.3.1.7 Signals, Equations

This segment of the report provides a comprehensive list of source signals; the signals driven by each source appears in the fanout list. Output signals not fed back through the array are listed as outputs with no feedback at the end of this segment.

Part of this segment is shown next. As expected, control signals and counter outputs Q0 through Q15 drive a large number of output signals, O0 through O15, are also identified as having no feedback.

No feedback, logic, or pin-out maps were generated because signal placement failed. The final messages in the report indicate the name of this report and the number of errors and warnings.

```
*** Signals - Equations Where Used
Signal Source                  Fanout List
         CLK
       COUNT:        Q0            Q1            Q2            Q3
         :           Q4            Q5            Q6            Q7
         :           Q8            Q9            Q10           Q11
         :           Q12           Q13           Q14           Q15
         {AAAA AAAA BBBB BBBB}
       UP:           Q1            Q2            Q3            Q4
         :           Q5            Q6            Q7            Q9
         :           Q10           Q11           Q12           Q13
         :           Q14           Q15
         {AAAA AAAB BBBB BB}
        LOAD:        Q0            Q1            Q2            Q3
         :           Q4            Q5            Q6            Q7
         :           Q8            Q9            Q10           Q11
         :           Q12           Q13           Q14           Q15
         {AAAA AAAA BBBB BBBB}
      SELECT:        O0            O1            O2            O3
         :           O4            O5            O6            O7
         :           O8            O9            O10           O11
         :           O12           O13           O14           O15
         {AAAA AA.. BBBB BB..}
       I0:           O0            Q0
         {AA}
       I1:           O1            Q1
         {AA}
       :
       I8:           O8            Q8
         {BB}
       :
       I15:          O15           Q15
         {.B}
       Q0:           O0            Q0            Q1           Q2
         :           Q3            Q4            Q5           Q6
         :           Q7
         {AAAA AAAA A}
    :
       Q7:           O7            Q7
         {.A}
       Q8:           O8            Q8            Q9           Q10
         :           Q11           Q12           Q13          Q14
         :             Q15
         {BBBB BBBB B}
    :
       Q15:          O15           Q15
         {.B}
*** Outputs with no feedback
         O0            O1            O2            O3            O4
         O5            O6            O7            O8            O9
         O10           O11           O12           O13           O14        O15
```

2.3.2 PROBLEM SUMMARY

Two obstacles were revealed during report interpretation.

- Logic was not distributed optimally within each block and not all signals could be placed, as noted in the block-partitioning segment of the report.

- The design requires almost 100% of all device resources, as noted in the device-resource checks.

Important: When working on your own designs, you may want to address both issues at one time. However, to show the impact of each strategy, this example explores only one problem at a time.

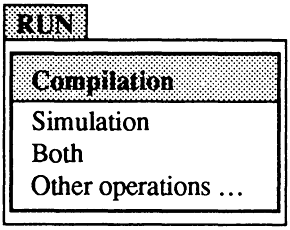
2.3.3 STRATEGIES TO MAXIMIZE LOGIC USE

As suggested in the MACH report, you can enable the following fitting option to maximize the logic in each MACH block.

```
When compiling
Maximize packing of logic blocks?          Y
...
```

2.4 ENABLE LOGIC PACKING

You can use steps below to initiate a new compilation process and enable the appropriate fitting option.



1. Select Compilation from the Run menu.

> **Note**: If you have a good understanding of the PALASM 4 software, you can use the prompts and figures on the left side of this discussion to quickly advance to discussion 2.5.

[F10]

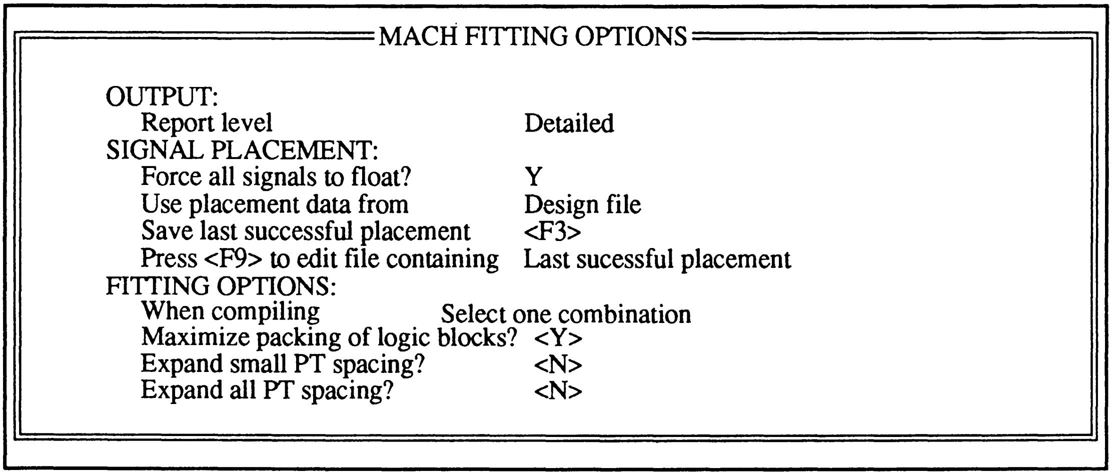
2. Confirm specifications in the Compile Options form.

The MACH Fitting Options form that appears shows specifications for the last compilation, which must be changed.

↓
When compiling
[F2]
S
Y [F10]

3. Activate the field, display the options, type the letter S to select one combination, type the letter Y to maximize packing ..., and confirm.

   The form on your screen should match the one below.



```
                    MACH FITTING OPTIONS

     OUTPUT:
        Report level                      Detailed
     SIGNAL PLACEMENT:
        Force all signals to float?       Y
        Use placement data from           Design file
        Save last successful placement    <F3>
        Press <F9> to edit file containing  Last sucessful placement
     FITTING OPTIONS:
        When compiling               Select one combination
        Maximize packing of logic blocks?  <Y>
        Expand small PT spacing?           <N>
        Expand all PT spacing?             <N>
```

[F10]

4. Confirm all specifications.

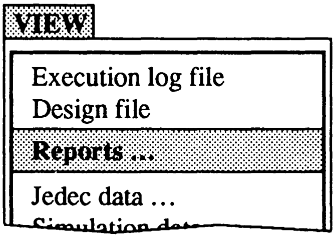
When the process finishes you see that despite the enabled logic-packing option, the fitting process was not completely successful.

[Esc]

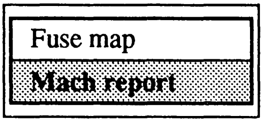
5. Dismiss the process window.

To review the MACH report,

1. Select the Reports command from the View menu.



2. Select Mach report from the submenu.



Tip: Again, to print the report you select the Other file command from the Edit menu, enter the name below, and print from the text editor as usual.

CNTMUX.RPT

2.5 INTERPRET SECOND REPORT

Here too, discussions are divided into report interpretation, problem summary, and additional improvement strategies.

2.5.1 INTERPRETATION

As you might expect, most areas of the report show no significant difference when compared with the initial one. However, certain segments are revealing.

2.5.1.1 Flags Used

This segment indicates the logic-packing option was enabled for this process: Max Packing=True.

```
Flags Used:           Unplace=True          Max Packing=True
Flags Used:      Expand Small=False          Expand All=False
```

The fitting option you enabled had no affect on the following segments, which are not shown.

- Pair Analysis
- Pre-Placement and Equation Usage
- Timing Analysis
- Device-Resource Checks

2.5.1.2 Block Partitioning

The first part of this segment, with last equations placed and affinity information, is unchanged. However, partitioning results confirm all node and output signals were placed within MACH 110 logic blocks.

- Output signals O0 through O7 and node signals Q0 through Q7 were placed in block A.

- Output signals O8 through O15 and node signals Q8 through Q15 were placed in block B.

Signal distribution in blocks A and B differ from those generated initially, as shown next.

```
Partitioning Design into Blocks...

*** Last Equations Placed in Blocks
:
*** Block Partitioning Results
             Array     Macros      # I/O      Buried    Product     Signal
            Inputs     Remain      Macro       Logic     Terms      Fanout
Block-> A     20          0          8           8         64          8
Block-> B     20          0          8           8         64          8
*** Block Signal List
Block-> A              O7              O6              O5              O4
                       O3              O2              O1              O0
                       Q0              Q1              Q2              Q3
                       Q4              Q5              Q7              Q6
Block-> B             O15             O14             O13             O12
                      O11             O10              O9              O8
                       Q8              Q9             Q10             Q11
                      Q12             Q13             Q15             Q14
```

Additional signal placements are discussed later, as you review the logic map. Device utilization is unchanged and is not repeated.

2.5.1.3 Assigning Resources

Error F600, at the end of this segment, shows a number of input signals were not placed and suggests a new solution.

```
Assigning Resources...
:
*** Macro Block B
         I12 (B 14)?              I13 (B 15)?
          I6 (B  8) ->   Blocked -> No Reshuffle Possible
          :
|> ERROR F600 - Not all input signals were connected! (signals=15/nc=30)
Try Using Expand Product Term Option
          I9 Unplaced
         I10 Unplaced
         I11 Unplaced
         :
          I7 Unplaced
          I8 Unplaced
```

2.5.1.4 Signals, Tabular

All control signals, node signals Q0 through Q15, and output signals O0 through O15, were placed. However, signals I0 through I15 are not placed.

```
*** Signals - Tabular Information
         Signal    #    P/N #    (Loc)      Type      Logic   # PT    Blocks
            CLK    1     35      I  5    clock pin      .
          COUNT    2     32      I  3        input      .               AB
             UP    3     13      I  2        input      .               AB
           LOAD    4     10      I  0        input      .               AB
         SELECT    5     11      I  1        input      .               AB
             I0    6      0      .?.         input      .               A
              :
             I7   13      0      .?.         input      .               A
             I8   14      0      .?.         input      .                B
              :
            I15   21      0      .?.         input      .                B
             O0   22      2      A  0      i/o pin    comb      2
              :
             O7   29      9      A  7      i/o pin    comb      2
             O8   30     24      B  0      i/o pin    comb      2
              :
            O15   37     31      B  7      i/o pin    comb      2
             Q0   38     10      A  8       buried    d-ff      3       A
              :
             Q7   45     17      A 15       buried    t-ff      4       A
              :
            Q15   53     33      B 15       buried    t-ff      4        B
```

The signals equations segment of the report remains essentially the same and is not shown. Again, the fanout list indicates that the control signals and counter outputs drive a large number of signals.

2.5.1.5 Feedback Map

Though the second fitting process was not completely successful, the enabled logic-packing option aided in feedback-map generation.

The feedback map provides an overview of output signals being fed back to drive other outputs. It also provides a visual measure of connectivity requirements. Input signals that could not be routed are not shown. Information here compliments information in the logic map.

This map shows how each input and feedback signal is routed through the switch matrix to the array. The numbers on this map correspond to switch-matrix blocks, not to logic-block locations. Switch-matrix blocks feed the PAL arrays, which in turn feed macros through the logic allocator.

If you refer to the PDS file, you can see that counter outputs Q0 through Q15 are designated as node signals, which are fed back through the array to generate MUX outputs O0 through O15. The feedback map shows node signals Q0 through Q15 as feedback inputs.

```
*** Feedback Map - COUNTER FOLLOWED BY MULTIPLEXER

Gbl Inp .--.       I/O  .--+--A--+--.  I/O        I/O  .--+--B--+--.  I/O
        | 0|            | 0|      |21| I5              | 0|      |21|
        | 1|            | 1|      |20| COUNT           | 1|      |20| COUNT
        | 2|            | 2|      |19| UP              | 2|      |19| UP
        | 3|            | 3|      |18|                 | 3|      |18|
        | 4|            | 4|      |17| SELECT          | 4|      |17| SELECT
        | 5|            | 5|      |16| LOAD            | 5|      |16| LOAD
        '--'            | 6|      |15: Q7              | 6|      |15: Q15
                        | 7|      |14: Q6              | 7|      |14: Q14
                   Q0 : 8|      |13: Q5         Q8 : 8|      |13: Q13
                   Q1 : 9|      |12: Q4         Q9 : 9|      |12: Q12
                   Q2 :10|      |11: Q3        Q10 :10|      |11: Q11
                        '--+-u--u+--'              '--+-u--u+--'
```

2.5.1.6 Logic Map

The logic map graphically summarizes the assignment of output signals and internal nodes for each block in the MACH device. It also shows the signals assigned to global input pins. This can help you gauge device utilization, distribution of signals among logic blocks, and assignment of signals to macros.

The logic map generated during this fitting process appears next. It shows that outputs O0 through O7 and O8 through O15 were placed adjacent to each other in blocks A and B, while nodes Q0 through Q7 and Q8 through Q15 were grouped together in blocks A and B. During the fitting process, output signals are placed before input signals are routed. However, in this design, logic is packed within the blocks so tightly that it is not possible to route appropriate inputs to generate the outputs.

```
*** Logic Map - COUNTER FOLLOWED BY MULTIPLEXER

Gbl Inp .--.      I/O  .--+--A--+--.  I/O        I/O  .--+--B--+--.  I/O
   LOAD| 0|        O0  | 0| 2    |21|             O8  | 0| 2    |21|
 SELECT| 1|        O1  | 1| 2    |20|             O9  | 1| 2    |20|
     UP| 2|        O2  | 2| 2    |19|            O10  | 2| 2    |19|
  COUNT| 3|        O3  | 3| 2    |18|            O11  | 3| 2    |18|
     I5| 4|        O4  | 4| 2    |17|            O12  | 4| 2    |17|
    CLK| 5|        O5  | 5| 2    |16|            O13  | 5| 2    |16|
        '--'       O6  | 6| 2   4|15|  Q7        O14  | 6| 2   4|15|  Q15
                   O7  | 7| 2   4|14|  Q6        O15  | 7| 2   4|14|  Q14
                   Q0  | 8| 3   4|13|  Q5         Q8  | 8| 3   4|13|  Q13
                   Q1  | 9| 4   4|12|  Q4         Q9  | 9| 4   4|12|  Q12
                   Q2  |10| 4   4|11|  Q3        Q10  |10| 4   4|11|  Q11
                       '--+-u--u+--'                  '--+-u--u+--'
```

2.5.2 SUMMARY

The feedback and logic maps illustrate that the outputs and nodes were grouped and placed next to each other. In this situation, flexibility when routing inputs is obstructed.

For example, as the logic map shows, inputs cannot be brought in using pins connected to macrocells 0 through 7 in blocks A and B, because these macrocells are used for I/O signals.

Interspersing signals O0 through O15 with Q0 through Q15 would result in increased flexibility during input routing. In that case, pins connected to Q0 through Q15 could be used to bring in the input signals.

2.5.3 STRATEGIES TO IMPROVE SIGNAL SPACING

The report suggested using a PT-spacing option to allocate empty space between macrocells.

The option below would allocate an empty space between each macrocell. Based on device-utilization and product-term requirements, that would be too much space for this design.

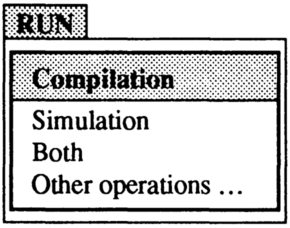
```
Expand all PT spacing
```

The following option would better distribute space between certain macrocells for the unrouted signals.

```
When compiling            Select one combination
:
Expand small PT spacing   Y
```

2.6 ENABLE SMALL PT SPACING

You can use the steps below to specify new options and initiate a new compilation process.



1. Select Compilation from the Run menu.

> **Note**: If you have a good understanding of the PALASM 4 software, you can use the prompts and figures on the left side of this discussion to quickly advance to discussion 2.7.

[F10]

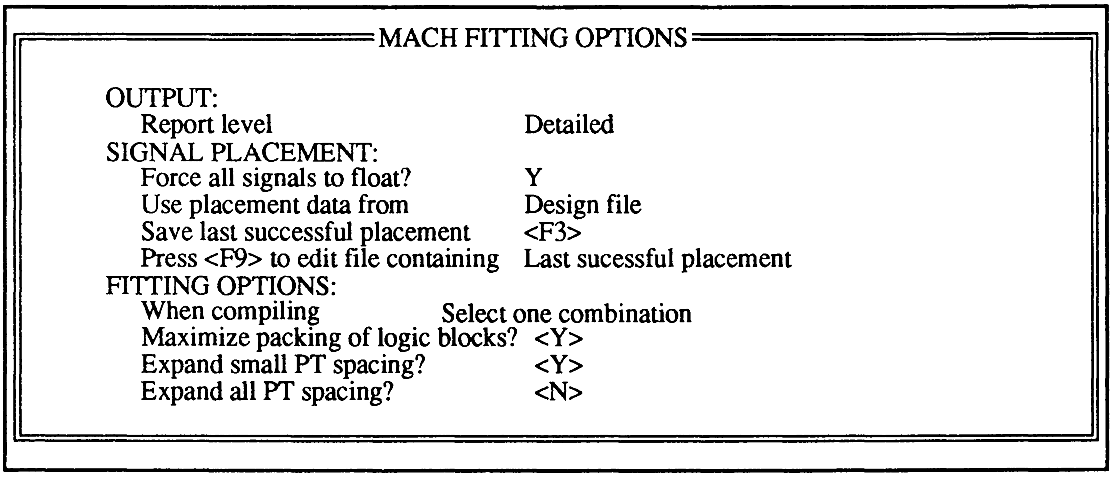
2. Confirm specifications in the first form.

The MACH Fitting Options form that appears shows current specifications; you must change options.

↓
When compiling
[F2]
S
↓
Expand small PT spacing
Y [F10]

3. Activate the field, display the options list, choose Select one combination, type the letter Y beside the small PT-spacing option, and confirm.

   The form should match the one below.



```
═══════════════ MACH FITTING OPTIONS ═══════════════

OUTPUT:
   Report level                          Detailed
SIGNAL PLACEMENT:
   Force all signals to float?           Y
   Use placement data from               Design file
   Save last successful placement        <F3>
   Press <F9> to edit file containing    Last sucessful placement
FITTING OPTIONS:
   When compiling                   Select one combination
   Maximize packing of logic blocks?   <Y>
   Expand small PT spacing?            <Y>
   Expand all PT spacing?              <N>
```

When specifications on your screen match the previous figure,

[F10]

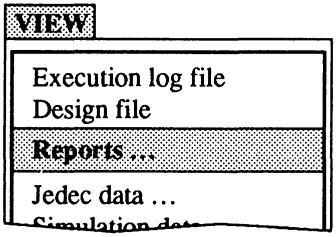
4. Confirm all fitting specifications.

When the process finishes you can see it was successful.

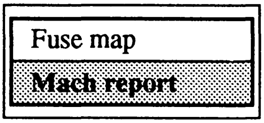
[Esc]

5. Dismiss the process window.

To review the MACH report,



1. Select the Reports command from the View menu.



2. Select Mach report from the submenu.

Tip: Again, to print the report you select the Other file command from the Edit menu, enter the name below, and print from the text editor as usual.

CNTMUX.RPT

2.7 INTERPRET FINAL REPORT

The two discussions here are divided according to report interpretation and the conclusion.

2.7.1 INTERPRETATION

As expected, the flags-used segment indicates you enabled the floating signals, logic-packing, and small-PT spacing options.

```
   Flags Used:              Unplace=True              Max Packing=True
   Flags Used:        Expand Small=True               Expand All=False
```

The fitting option you enabled did not affect the following report segments, which are not shown.

- Pair Analysis
- Pre-Placement and Equation Usage
- Timing Analysis
- Device-Resource Checks
- Block Partitioning
- Device Utilization

The only significant changes to the resource-assignment segment is that no errors or unplaced signals are reported. No significant changes appear in the signal-equations table. These segments are not shown.

The tabular-signals table and the feedback and logic maps have changed, as shown next. Also, a device pin-out map was produced.

2.7.1.1 Signals, Tabular

As you can see in the partial tabular-signals list below, the input signals have now been placed in blocks.

```
  *** Signals - Tabular Information

        Signal    #    P/N #    (Loc)        Type      Logic   # PT    Blocks
           :
           I0     6      3       A  1         input        .              A
           I1     7      5       A  3         input        .              A
           I2     8     15       A  9         input        .              A
           I3     9     17       A 11         input        .              A
           :
          I12    18     41       B 13         input        .               B
          I13    19     43       B 15         input        .               B
          I14    20      7       A  5         input        .               B
          I15    21      9       A  7         input        .               B
           :
```

2.7.1.2 Feedback Map

The feedback map reflects the new distribution and placement of signals within blocks. All input signals, both external and feedback signals, have been successfully roufed. The figure below shows the new feedback map. Ix and Cx signals are now interspersed.

```
*** Feedback Map - COUNTER FOLLOWED BY MULTIPLEXER

Gbl Inp .--.      I/O  .--+--A--+--.  I/O          I/O  .--+--B--+--.  I/O
        | 0|       I1 : 0|      |21| I5             I9 : 0|      |21|
        | 1|       Q0 : 1|      |20| COUNT          Q8 : 1|      |20| COUNT
        | 2|       I0 : 2|      |19| UP             I8 : 2|      |19| UP
        | 3|       Q1 : 3|      |18|                Q9 : 3|      |18|
        | 4|       I7 : 4|      |17| SELECT        I15 : 4|      |17| SELECT
        | 5|       Q2 : 5|      |16| LOAD          Q10 : 5|      |16| LOAD
        '--'       I6 : 6|      |15: Q7           I14 : 6|      |15: Q15
                   Q3 : 7|      |14: I4           Q11 : 7|      |14: I10
                   I3 : 8|      |13: Q6           I13 : 8|      |13: Q14
                   Q4 : 9|      |12|              Q12 : 9|      |12: I11
                   I2 :10|      |11: Q5           I12 :10|      |11: Q13
                       '--+-u--u+--'                   '--+-u--u+--'
```

2.7.1.3 Logic Map

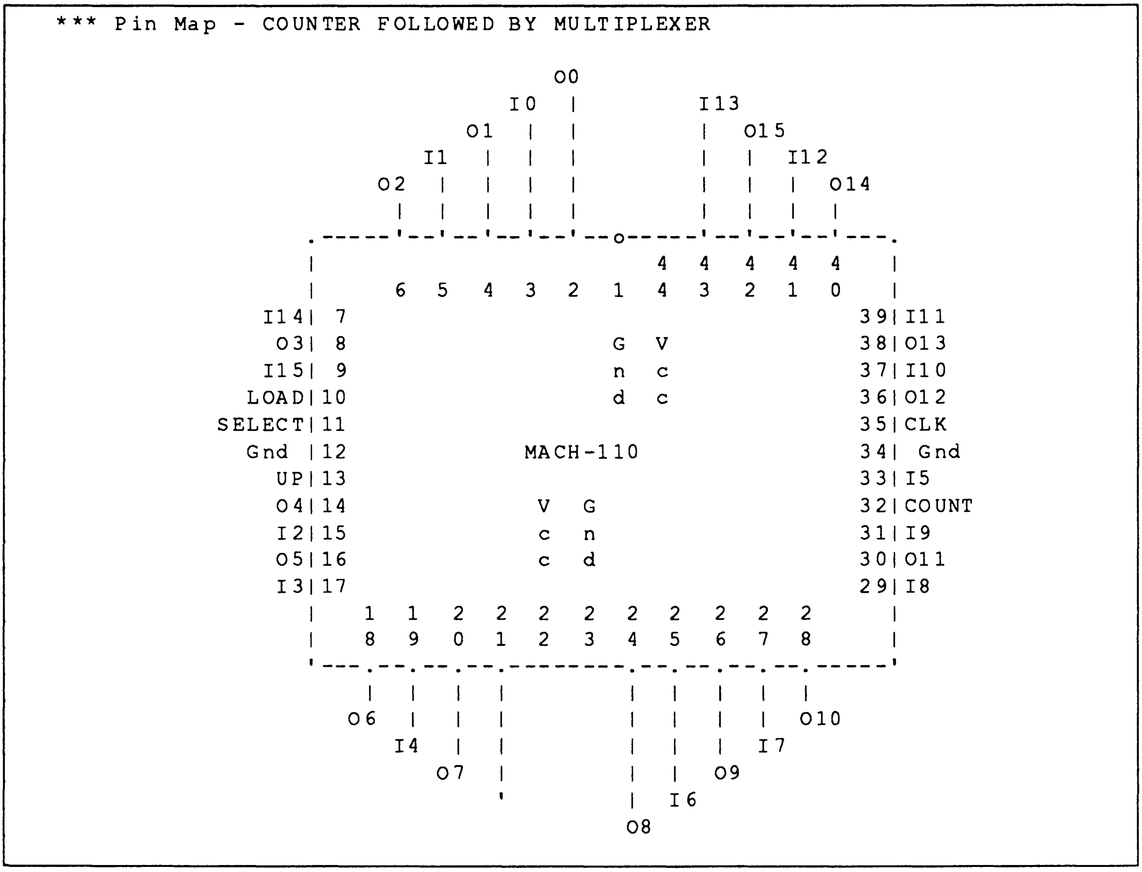
The logic map summarizes the new placement. Logic blocks are well packed. Output signals O0 through O15 are interspersed with node signals Q0 through Q15.

The small product-term expansion option provided space for input signals so pins associated with Q0 through Q15 can be used as inputs. The result is the successful routing of input signals.

```
*** Logic Map - COUNTER FOLLOWED BY MULTIPLEXER

Gbl Inp .--.        I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
   LOAD| 0|          O0 | 0| 2    |21|                 O8 | 0| 2    |21|
 SELECT| 1|          Q0 | 1| 3    |20|                 Q8 | 1| 3    |20|
     UP| 2|          O1 | 2| 2    |19|                 O9 | 2| 2    |19|
  COUNT| 3|          Q1 | 3| 4    |18|                 Q9 | 3| 4    |18|
     I5| 4|          O2 | 4| 2    |17|                O10 | 4| 2    |17|
    CLK| 5|          Q2 | 5| 4    |16|                Q10 | 5| 4    |16|
       '--'          O3 | 6| 2   4|15| Q7             O11 | 6| 2   4|15| Q15
                     Q3 | 7| 4   2|14| O7             Q11 | 7| 4   2|14| O15
                     O4 | 8| 2   4|13| Q6             O12 | 8| 2   4|13| Q14
                     Q4 | 9| 4   2|12| O6             Q12 | 9| 4   2|12| O14
                     O5 |10| 2   4|11| Q5             O13 |10| 2   4|11| Q13
                        '--+-u--u+--'                     '--+-u--u+--'
```

2.7.1.4 Device Pin-Out Map

The following device pin-out map was generated during this successful placement.



```
   *** Pin Map - COUNTER FOLLOWED BY MULTIPLEXER

                                     O0
                                  I0  |        I13
                               O1  |  |        |  O15
                            I1  |  |  |        |  |  I12
                         O2  |  |  |  |        |  |  |  O14
                          |  |  |  |  |        |  |  |  |
                    .-----'--'--'--'--'--o-----'--'--'--'---.
                    |                       4  4  4  4  4   |
                    |     6  5  4  3  2  1  4  3  2  1  0   |
                 I14| 7                                   39|I11
                  O3| 8                  G  V             38|O13
                 I15| 9                  n  c             37|I10
                LOAD|10                  d  c             36|O12
              SELECT|11                                   35|CLK
                Gnd |12            MACH-110               34| Gnd
                  UP|13                                   33|I5
                  O4|14             V  G                  32|COUNT
                  I2|15             c  n                  31|I9
                  O5|16             c  d                  30|O11
                  I3|17                                   29|I8
                    |   1  1  2  2  2  2  2  2  2  2  2     |
                    |   8  9  0  1  2  3  4  5  6  7  8     |
                    '---.--.--.--.--------.--.--.--.--.-----'
                        |  |  |  |        |  |  |  |  |
                       O6  |  |  |        |  |  |  |  O10
                          I4  |  |        |  |  |  I7
                             O7  |        |  |  O9
                                 '        |  I6
                                          O8
```

2.7.2 CONCLUSION

Partitioning problems arose initially when logic-block packing was not optimized for this design with high device-resource requirements.

When logic packing was optimized, routing problems occurred because the outputs were placed adjacent to each other, which obstructed the routing of inputs.

By enabling the following two fitting options together, both the partitioning and routing problems were improved and a successful fit resulted.

```
When compiling                     Select one combination
Maximize packing of logic blocks?  Y
Expand small PT spacing            Y
...
```

This concludes the example.

Universal Asynchronous Receiver / Transmitter

A Design with Signal Grouping Requirements

CONTENTS

3

Universal Asynchronous Receiver / Transmitter: A Design with Signal Grouping Requirements

This chapter uses a Universal Asynchronous Receiver/Transmitter (UART) design to illustrate how grouping signals that share common resources can lead to a successful fit in a MACH 230 device. The Signals – Equations Where Used segment of the MACH report contains information to help you make grouping decisions.

The design in this example has an overall device-utilization of approximately 50%. You initially compile the design without grouping any signals. During MACH-report interpretation, you gain insight into the partitioning problems caused by product term distribution, wiring congestion, and block utilization. You then group specific signals so logic with common connections is placed in the same block. This opens up interconnect resources between blocks and reduces wiring congestion. Recompiling the design results in a successful fit in a MACH 230 device.

3.1 DESIGN DESCRIPTION

Files related to this AMD-supplied design are stored in the directory noted below.

```
\PALASM\EXAMPLES\WB\UART
```

A block diagram of the UART design is shown on the next page. The TRANSMITTER block contains the input register, shift register and the parity and control logic. The RECEIVER block contains the receiver registers and clocking logic.[1]

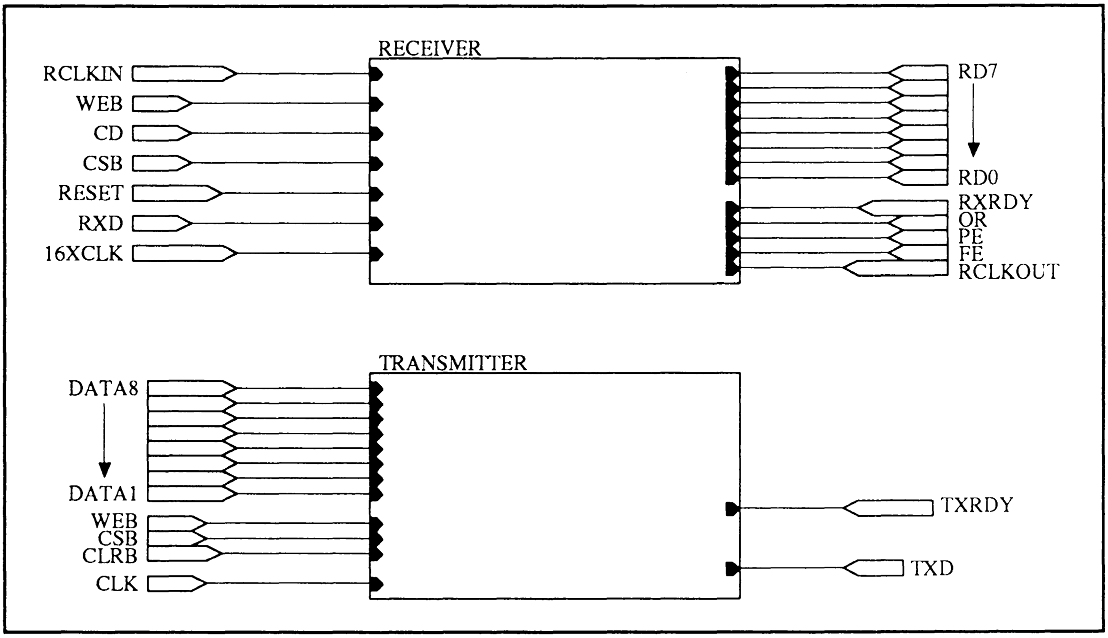
This UART uses eight-bit data words with even parity, making it especially useful as a peripheral to a microprocessor data bus. During transmission, data is clocked into the input registers and then loaded into the shift registers. Parity and framing bits are added to the serial output stream. When input data is available, the UART supplies the clock for the serial input data stream. Start and stop bits synchronize the data so it can be shifted into a set of registers. If an error is detected, a corresponding flag is set to indicate the data is corrupt. If good data is received, it is loaded into the parallel registers that drive the receiver data bus.

3.1.1 PINS

A MACH 230 device supports 70 signal pins. This design uses 32 of the pins as follows.

- 17 input pins for DATA1 – DATA8, WEB, CSB, CLRB, CLK, CD, RXD, RESET, 16XCLK, and RCLKIN.
- 15 output pins for TXRDY, TXD, OR, PE, FE, RXRDY, RCLKOUT, RD0 – RD7.
- 2 output pins are unusable due to the internal nodes _55_Idle and _55_Idlectl.

[1] Refer to Discussion 3.6, Schematics, for a complete set of UART schematic figures.



UART Block Diagram

3.1.2 MACROCELLS

The MACH 230 provides 64 output macrocells and 64 buried macrocells. This design uses

- 24 I/O macrocells for the signals DATA1 – DATA8, TXRDY, TXD, _55_IDLE, RXRDY, RCLKOUT, RD0 – RD7, OR, PE, FE.
- 47 buried macrocells for other internal signals.

3.1.3 PRODUCT TERMS

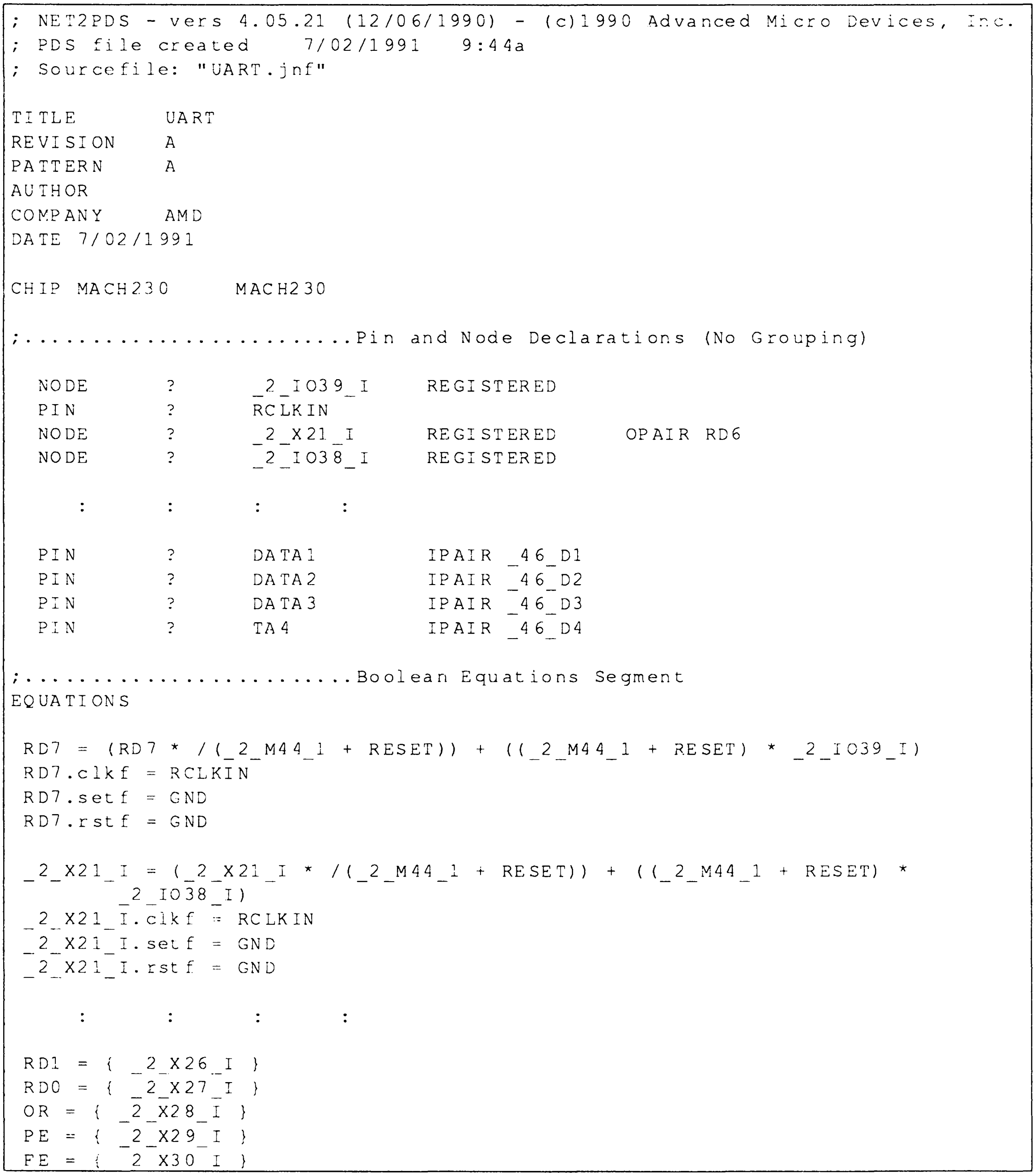
The MACH 230 device provides 512 product terms (PTs). As currently implemented, 37 equations require one product term, 14 equations require two product terms, and 12 equations require three product terms.

However, because product terms are allocated in groups of four, the total number of product terms actually used is 252, as shown next.

IMPLEMENTATION	ACTUAL PTS	ALLOCATED PTS
37 Eqns. X 1 PT	37	148
14 Eqns. X 2 PTs	28	56
12 Eqns. X 3 PTs	36	48
TOTAL	101	252

A portion of the PDS file for the UART design is shown on the next page. This file was generated automatically during compilation from the schematic-based design. Prefixes, such as _46_, and suffixes, such as _2, result from the schematic hierarchy.[2]

[2] Refer to the *PALASM 4 User's Manual*, Section III, Chapter 7, for information on schematic signal names.



```
; NET2PDS - vers 4.05.21 (12/06/1990) - (c)1990 Advanced Micro Devices, Inc.
; PDS file created    7/02/1991   9:44a
; Sourcefile: "UART.jnf"

TITLE       UART
REVISION    A
PATTERN     A
AUTHOR
COMPANY     AMD
DATE 7/02/1991

CHIP MACH230     MACH230

;..........................Pin and Node Declarations (No Grouping)

  NODE      ?      _2_IO39_I     REGISTERED
  PIN       ?      RCLKIN
  NODE      ?      _2_X21_I      REGISTERED      OPAIR RD6
  NODE      ?      _2_IO38_I     REGISTERED

     :      :      :      :

  PIN       ?      DATA1         IPAIR _46_D1
  PIN       ?      DATA2         IPAIR _46_D2
  PIN       ?      DATA3         IPAIR _46_D3
  PIN       ?      TA4           IPAIR _46_D4

;..........................Boolean Equations Segment
EQUATIONS

 RD7 = (RD7 * /(_2_M44_1 + RESET)) + ((_2_M44_1 + RESET) * _2_IO39_I)
 RD7.clkf = RCLKIN
 RD7.setf = GND
 RD7.rstf = GND

 _2_X21_I = (_2_X21_I * /(_2_M44_1 + RESET)) + ((_2_M44_1 + RESET) *
        _2_IO38_I)
 _2_X21_I.clkf = RCLKIN
 _2_X21_I.setf = GND
 _2_X21_I.rstf = GND

     :      :      :      :

 RD1 = { _2_X26_I }
 RD0 = { _2_X27_I }
 OR = { _2_X28_I }
 PE = { _2_X29_I }
 FE = { _2_X30_I }
```

UART.PDS Design File

3.2 COMPILE THE DESIGN

This discussion provides the setup and compilation steps to compile the text-based design, UART.PDS. However, you can also compile the top-level schematic design, UART.SCH, which is provided with the installation diskettes.

Note: If you have a good understanding of the PALASM 4 software, you can use only the prompts and figures on the left side of this discussion to quickly advance to discussion 3.3.

3.2.1 SETUP

The following steps guide you as you retrieve the PDS file and verify setup options to ensure they are appropriate to initially compile and fit this design.

To begin from DOS,

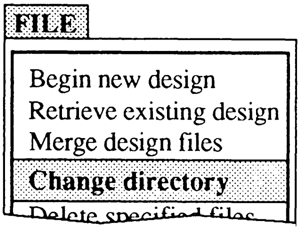
C:\
PALASM [Enter]

1. Type PALASM at the DOS prompt, then press [Enter] to run the software.

Press any key ...
[Space bar]

2. Dismiss the copyright notice and continue.

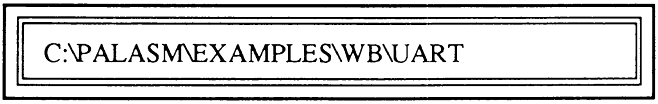
To retrieve the design,



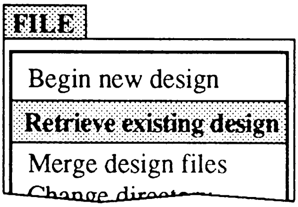
1. Select the Change directory command from the File menu.

**C:\PALASM\EXAMPLES\
WB\UART [F10]**

2. Type the path name shown on the left and confirm.



C:\PALASM\EXAMPLES\WB\UART

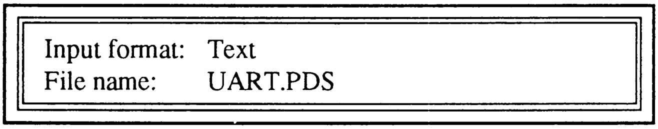


3. Select Retrieve existing design from the File menu.

**[F2]
T [Enter]
UART.PDS [F10]**

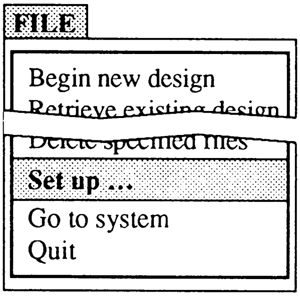
4. Display the submenu and select Text, activate the file-name field, type the file name shown on the left and confirm.

The form on your screen should match the one below.



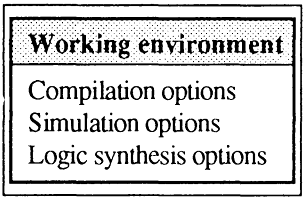
Input format: Text
File name: UART.PDS

To verify the setup,



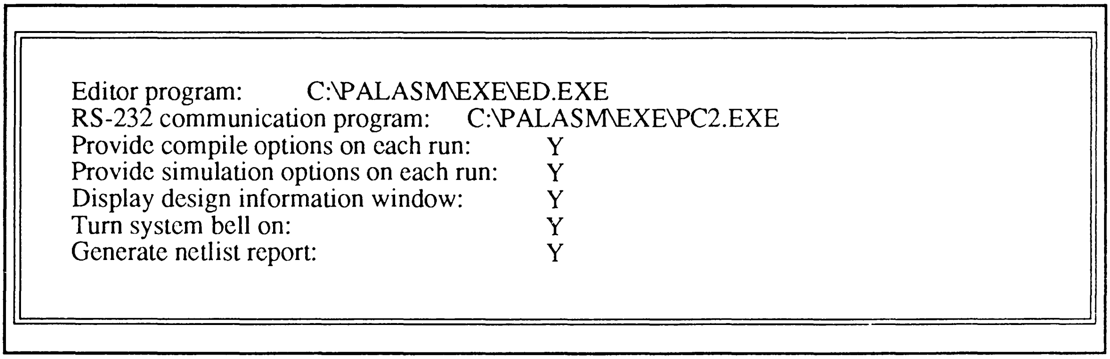
1. Select Set up ... from the File menu.

   A submenu opens offering four setup options.



2. Select Working environment from the submenu.

   A form appears that identifies the following.



```
Editor program:         C:\PALASM\EXE\ED.EXE
RS-232 communication program:    C:\PALASM\EXE\PC2.EXE
Provide compile options on each run:        Y
Provide simulation options on each run:     Y
Display design information window:          Y
Turn system bell on:                        Y
Generate netlist report:                    Y
```

The third specification allows you to confirm compilation options immediately before the process begins.

Important: If the third specification on your screen differs, change it as indicated in step 3. In any case, complete step 4.

↓
Provide compile options ...
Y

3. Change the compile options specification if needed.

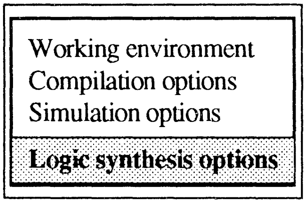
[F10]

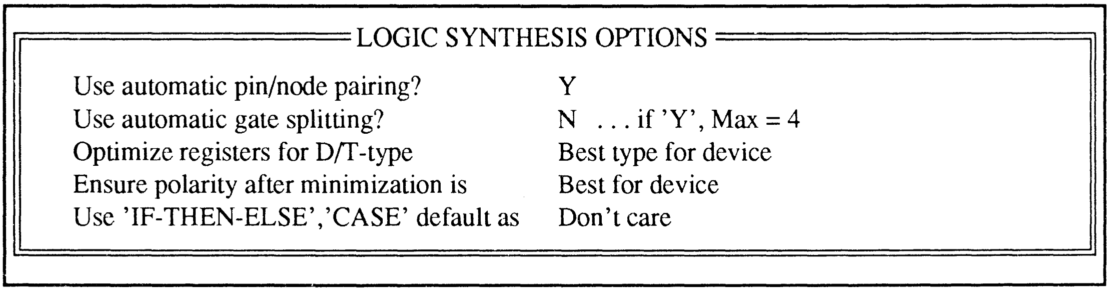
4. Confirm all specifications and dismiss the form by pressing [F10].

To verify the logic-synthesis setup,

1. Select Logic synthesis options from the submenu.

   The following form appears.





LOGIC SYNTHESIS OPTIONS

Use automatic pin/node pairing?	Y
Use automatic gate splitting?	N ... if 'Y', Max = 4
Optimize registers for D/T-type	Best type for device
Ensure polarity after minimization is	Best for device
Use 'IF-THEN-ELSE','CASE' default as	Don't care

The figure above shows options that must be specified for the first compilation of this design.

Important: If options on your screen differ, complete steps 2 through 4. In any case, complete steps 5 and 6.

Use auto ... pairing?
Y
Use auto ... splitting?
N

2. Type the correct letter to enable automatic pairing and disable gate splitting.

Optimize registers ...
[F2]
B [Enter]
Ensure polarity ...
[F2]
B [Enter]

3. Display the list of options and type the letter B to select the Best ... options for register optimization and polarity specifications, respectively.

Use 'IF-THEN-ELSE', 'CASE'...
[F2]
D

4. Display the list of options and type the letter D to select the Don't Care option for the language specification.

[F10]

5. Confirm all specifications in the form by pressing [F10].

   The form is dismissed, any changes are recorded, and the Set up submenu is again displayed.

[Esc]

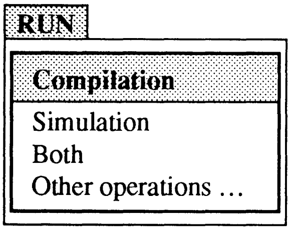
6. Dismiss the Set up ... submenu.

   You're returned to the File menu; the Set up command remains highlighted.

3.2.2 COMPILATION

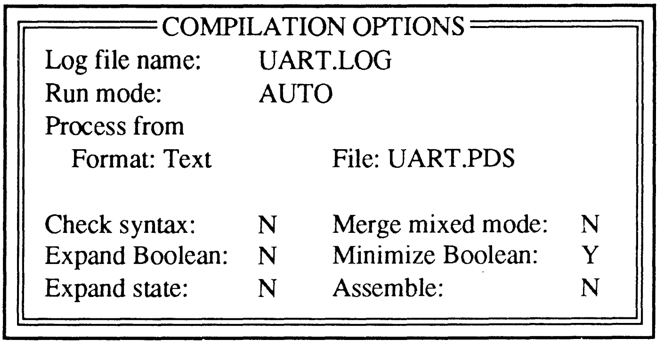
Now that you've verified basic setup options, you compile the design and interpret the report.

To begin,



1. Select Compilation from the Run menu.

   The Compilation Options form appears listing various specifications.



```
COMPILATION OPTIONS
Log file name:    UART.LOG
Run mode:         AUTO
Process from
  Format: Text            File: UART.PDS

Check syntax:     N       Merge mixed mode:   N
Expand Boolean:   N       Minimize Boolean:   Y
Expand state:     N       Assemble:           N
```

For this design, automatic run mode is used. In this mode, a Y after an option in the lower part of the form guarantees the option will execute. The software determines if any other options need to be run.

> **Important**: Be sure to complete the steps below to set the execution-log name and run mode.

UART.LOG [Enter]

2. Enter the execution-log file name shown on the left.

   The Run mode field becomes active.

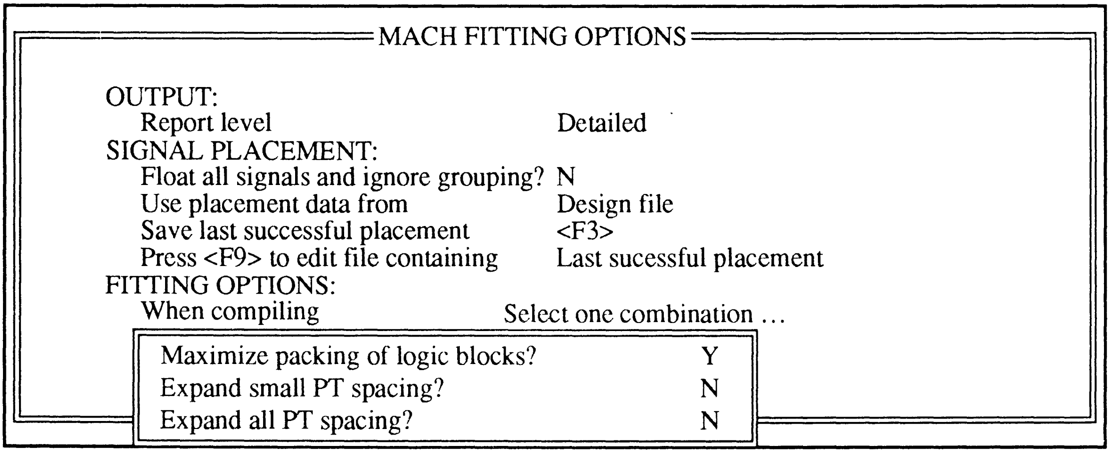
[F2]
A

3. Display the options list and select Automatic.

[F10]

4. Confirm specifications in the form by pressing [F10].

   A new form appears showing the MACH fitting options to be used.



```
═══════════════════ MACH FITTING OPTIONS ═══════════════════

    OUTPUT:
       Report level                          Detailed
    SIGNAL PLACEMENT:
       Float all signals and ignore grouping? N
       Use placement data from               Design file
       Save last successful placement        <F3>
       Press <F9> to edit file containing    Last sucessful placement
    FITTING OPTIONS:
       When compiling                     Select one combination ...
       ┌──────────────────────────────────────────────┐
       │ Maximize packing of logic blocks?         Y  │
       │ Expand small PT spacing?                  N  │
       │ Expand all PT spacing?                    N  │
       └──────────────────────────────────────────────┘
```

> **Important**: The specifications on your screen should match the above figure. If they differ, complete steps 5 through 8. In any event, complete steps 9 and 10.

Report level
[F2]
D [Enter]

5. Display the list of options and type the letter D to select Detailed for the report option.

Force all signals ...
N

6. Type the letter N to use pin and grouping assignments.

Use placement ...
[F2]
D [Enter] [Enter]

7. Display the list of options and type the letter D to select Design file as the placement data option.

When compiling
[F2]
S Y N N [F10]

8. Display fitting options, type the letter S for Select One Combination, then the letters Y N N to set the options.

[F10]

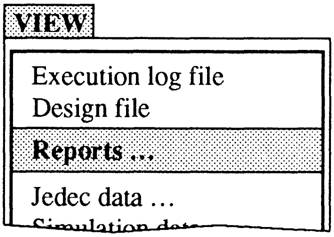
9. Confirm all specifications.

A window opens as the process begins. Error messages and explanations will scroll by as the compilation executes; this takes several minutes. Upon completion, you also see the fitting process terminated with errors and warnings.

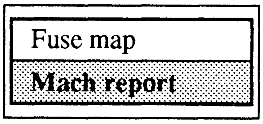
[Esc]

10. Dismiss the process window.

To review the MACH report,



1. Select the Reports ... command from the View menu.



2. Select the Mach report command from the submenu, then scroll through the report.

Tip: To print the report, you select the Other file command from the Edit menu, enter the name below, and print from the text editor as usual.

UART.RPT

3.3 INTERPRET INITIAL REPORT

This discussion is divided into three parts, as follows.

- Interpretation
- Problem Summary
- Improvement Strategies

> **Important**: This discussion illustrates portions of the MACH report. However, many of the report segments are too lengthy to show in their entirety. Missing text is indicated with vertical ellipses.

3.3.1 INTERPRETATION

Discussions below explain and show what various segments of the MACH report reveal about this design and the unsuccessful fitting process.

3.3.1.1 Flags Used

The flags-used segment indicates the flags used for a specific fitting attempt, as shown next.

```
Flags Used:                 Unplace=True                   Max Packing=True
Flags Used:                 Expand Small=False             Expand All=False
```

3.3.1.2 Pair Analysis/Preplacement

Ordinarily, the pair analysis segment identifies errors caused by illegal pair declarations in pin and node statements. The preplacement segment identifies illegal and conflicting preplacement data. Neither segment is shown because no errors were reported for this circuit.

3.3.1.3 Timing Analysis for Signals

The timing-analysis table, shown on the next page, indicates various kinds of delays.

All signals listed have the maximum delay, which specifies the number of passes through the array. Signals in this design do not exceed two array-pass delays. This segment offers no clue about the unsuccessful fit.

```
*** Timing Analysis for Signals

 Parameter   Min   Max    Signal List (Those having Max delay.)
      Tpd     1     1          RCLKOUT         _2_IO2_I         _2_M48_O
                                 TXRDY            TXD
      Tsu     1     2            RXRDY             PE                 FE
      Tco     0     2       _59_M15_2
      Tcr     1     2          _46_TAG        _56_M19_1   _55_PARITY_001
                            _59_M11_1              OR                 PE
                                    FE


 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - clock to output (register output to combinatorial output)
 Tcr - Clock to register setup delay
.......(Register output thru combinatorial logic to reg input)
 All delay values are expressed in terms of array passes
```

3.3.1.4 Device-Resource Checks

The device-resource table indicates available resources in the selected device, those required by the design, and the percentage of utilization.

The segment of the report, shown next, indicates the following use of device resources.

- Pin utilization, 48%
- Product-term utilization, 48%

Strictly speaking, the number of product terms required for this design is 101. However, product terms are automatically allocated in clusters of four, so this design actually uses 252, or 48% of the total product-term resources.

```
*** Device Resource Checks

                Available       Used        Remaining
       Clocks:      4            3             1
         Pins:     70           34            36      ->     48%
    I/O Macro:     64           16            48
  Total Macro:    128           63            65
Product Terms:    512          101           260      ->     48%

MACH-PLD Resource Checks OK!
```

3.3.1.5 Block Partitioning

The partitioning segment of the report is shown below. It provides several groups of information.

Last Equations Placed in Blocks is divided into two groups, Weakly and Assign, that identify signal affinity, if any.

Though not relevant to this design, information here can help you determine which signals to remove if you need to re-engineer the design.

```
Partitioning Design into Blocks...

*** Last Equations Placed in Blocks

Weakly -          RXRDY          TXRDY          TXD       _2_IO38_I
Assign -        _2_M44_1       _2_M69_1       _2_M48_O      _2_M49_I
```

Block Partitioning Results provides statistics about the placement of logic in MACH blocks: remaining macros, the number of array inputs, I/O and buried macros used, the number of product terms used, and signal fanouts.

As you can see, the logic is partitioned between eight MACH blocks, A – H, in equal proportions. Each block is limited to one unique reset and preset definition.

```
*** Block Partitioning Results

                Array    Macros     # I/O    Buried   Product    Signal
               Inputs    Remain     Macro     Logic    Terms     Fanout
   Block-> A     19         8         2         6        32         11
   Block-> B      7         3         2        11        36         11
   Block-> C     15         9         2         5        28         11
   Block-> D     15        10         2         4        24         11
   Block-> E     10         7         2         7        32         12
   Block-> F     10         6         1         9        36         16
   Block-> G     11         7         3         6        32          6
   Block-> H     12         7         2         7        32         19
```

Block Signal List identifies which signals were placed into the specified MACH blocks.

```
*** Block Signal List

Block-> A        _2_M48_O        _2_RDR1_D             RD2       _59_M11_1
                _59_M15_2   _55_PARITY_001       _59_TSC1_Q        _55_IDLE

Block-> B           DATA8            _46_D8           DATA6          _46_D6
                    DATA3            _46_D3           DATA1          _46_D1
                 _2_M71_I         _2_I047_I             RD1         _2_I02_I
                      RD7

Block-> C        _46_M6_I            _46_Q4         _2_M54_3        _2_M70_2
                _2_I038_I               RD0              PE

Block-> D          _46_Q6            _46_Q2         _2_M54_2              OR
                 _2_M72_I           RCLKOUT

Block-> E           DATA7            _46_D7           _46_Q3         _46_TAG
                 _2_M57_I          _2_M69_1        _2_I037_I              FE
                      RD6

Block-> F          _46_Q7             DATA5           _46_D5          _46_Q1
                 _2_M60_2          _2_M49_I        _2_I033_I        _2_M69_2
                _2_I039_I               RD5

Block-> G          _46_Q8             DATA4           _46_D4             TXD
                    TXRDY      _55_TLOADCTL        _56_M19_1       _2_I032_I
                      RD4

Block-> H       _59_M18_2            _46_Q5           DATA2          _46_D2
                 _2_M56_2          _2_M44_1           RXRDY       _2_I031_I
                      RD3
```

Device Utilization identifies an overall device utilization of 47%. Though it provides information related to internal processing, it does not reveal any fitting problems.

```
|> INFORMATION F050 - Device Utilization....... *: 47 %
```

Errors and warnings appear if any occur during partitioning. In this design, errors are reported concerning connection problems and congested wiring. These problems can be solved by redistributing signals into strategic groups. This frees interconnection resources for large equations like _59_TSC1_Q.

```
Assigning Resources...



*** Macro Block H


   I/O Macros>            RD3           RXRDY
      Targets>   0(82)   2(81)

            RD3 (H  0) -> (H  8)
          RXRDY (H  2) -> (D  1) (H  3)

 Buried Logic>          _46_D2
      Targets>   4(80)  12(76)

         _46_D2 (H  5) -> (A  1)   Blocked -> No Reshuffle Possible


|> ERROR F590 - Connection problem (Wiring Congested) - _46_D2

 Buried Logic>        _2_M56_2          _46_Q5        _59_M18_2         _2_M44_1
_2_IO31_I
      Targets>   1(na)   3(na)   4(80)   5(na)   6(79)   7(na)   8(78)   9(na)
                10(77)  11(na)  12(76)  13(na)  14(75)  15(na)

     * Retry Mapping
     * Retry Mapping
        _2_M56_2 (H  4) -> (D  2) (E  2)
       _59_M18_2 (H  5) -> (A 19)
          _46_Q5 (H  6) -> (A 11) (D  3)
        _2_M44_1 (H 14) -> (A 17) (B  0) (C  8) (D  8) (E  8) (F  8) (G  0) (H  0)
       _2_IO31_I (H 15) -> (A 15)

      :                :                :              :
```

3.3.1.6 Signals, Tabular

The information in this table can be useful when you want to minimize inter-block connections. The following information is provided.

- Signal names and corresponding pin numbers
- Pin and node numbers
- Macrocell block and cell location

- Pin type
- Logic type: D-type, T-type, Comb, etc.
- Number of product terms used for outputs
- The block(s) driven by each signal

Each question mark, ?, in the (Loc) field indicates an unplaced signal. You can see in the segment shown next, some pins are unplaced because the process terminated before completion.

```
*** Signals - Tabular Information

          Signal     #    P/N #    (Loc)      Type      Logic   # PT    Blocks
          RCLKIN     1     65       I  4   clock pin      .       .    .........
           RXRDY     2     81       H  2     i/o pin    d-ff      2    ....D...H
          16XCLK     3     62       I  3   clock pin      .       .    ....D....
             RXD     4     57       F  8       input      .       .    ......F..
           RESET     5     41       I  2       input      .       .    .ABCDEFGH
             RD7     6     19       B  0     i/o pin    d-ff      3    ..B......
         _2_X21_I    7     68       E  2    out pair    d-ff      3    .....E...
         _2_X22_I    8     82       F  0    out pair    d-ff      3    ......F..

   :                  :                 :                :

         _46_M6_I   85     46       C 12      buried    comb      1    .A...E.G.
     _55_TLOADCTL   86    112       G 14      buried    d-ff      1    ...C.....
   _55_PARITY_001   87     16       A 14      buried    d-ff      3    .......G.
        _59_M15_2   88      9       A  7      buried    comb      1    .A.......
        _59_M11_1   89      7       A  5      buried    d-ff      3    .A.......
        _59_M18_2   90    119       H  5      buried    d-ff      1    .A.......
       _59_TSC1_Q   91      3       A  1      buried    d-ff      1    ........H
```

3.3.1.7 Signals, Equations

This segment of the report provides a comprehensive list of source signals; the signals driven by each source appear in the fanout list. Output signals not fed back through the array are listed as outputs with no feedback at the end of this segment. Part of this segment is shown next.

This segment is very useful in determining which signals need to be placed in different blocks to reduce wiring congestion. For each signal source, a fanout list is provided, followed by the block location of each fanout signal. For example, in this segment for the UART design, the signal _46_Q1 fans out to only one signal, _46_Q2, which is located in block A.

Note that the blocks associated with the signals are enclosed in brackets. Ideally, all the letters indicating blocks in each set of brackets would be the same for minimal wiring congestion. As expected, the Q shift-register outputs in the transmitter block drive a large number of output signals.

```
*** Signals - Equations Where Used

   Signal Source                   Fanout List
         RCLKIN
          RXRDY:          RXRDY              OR              OR
               {HDD}
         16XCLK:        RCLKOUT
               {D}
            RXD:       _2_M49_I        _2_M60_2
               {FF}
          RESET:            RD7             RD6             RD5             RD4
               :            RD3             RD2             RD1             RD0
               :             OR         RCLKOUT       _2_IO39_I             RD6
               :      _2_IO38_I             RD5       _2_IO37_I             RD4
               :      _2_IO33_I             RD3       _2_IO32_I             RD2
               :      _2_IO31_I             RD1       _2_RDR1_D             RD0
               :      _2_IO47_I        _2_M44_1        _2_IO2_I        _2_M69_1
               :             OR        _2_M48_O        _2_M69_2
               {BEFG HABC DDFE CFEG FHGA HBAC BHBE DAF}

      :                :               :              :

         _46_Q7:       _55_IDLE          _46_Q8  _55_PARITY_001       _59_M15_2
               :      _59_M11_1      _59_TSC1_Q
               {AGAA AA}

         _46_Q8:       _55_IDLE  _55_PARITY_001       _59_M15_2       _59_M11_1
               :     _59_TSC1_Q
               {AAAA A}

      :                :               :              :

     _59_TSC1_Q:      _59_M18_2
               {H}


*** Outputs with no feedback

          RCLKOUT           TXRDY             TXD
```

Although a feedback and a logic map were generated, they provide no further information for determining the grouping structure for the UART design. The final messages in the report indicate the name of this report and the number of errors and warnings.

3.3.2 PROBLEM SUMMARY

The main obstacle revealed in the MACH report is wiring congestion.

3.3.3 GROUPING SIGNALS TO REDUCE WIRING CONGESTION

You can use the information in the Signals – Equations Where Used segment of the MACH report to group signals. Ideally, all letters indicating blocks in each set of brackets should be the same for minimal wiring congestion.

For example, refer to the fanout list for signals _46_D2 and _2_M48_0, shown next.

```
     _46_D2:        _55_IDLE          _46_Q2  _55_PARITY_001        _59_M15_2
           :       _59_M11_1      _59_TSC1_Q
           {ADAA AA}

   _2_M48_O:              OR              PE              FE               OR
           :              PE              FE
           {DCED CE}
```

For signal _46_D2, the bracketed letters, {ADAA AA}, refer to the fanout list as follows.

- _55_IDLE is in block A
- _46_Q2 is in block D
- _55_PARITY_001 is in block A
- _59_M15_2 is in block A
- _59_M11_1 is in block A
- _59_TSC1_Q is in block A

If signal _46_Q2 is grouped into block A, instead of D, wiring congestion is reduced because _46_D2 no longer fans out to block D. The new block fanout can be summarized as {AAAA AA}.

The same principle can be applied to all the signals in the _2_M48_0 fanout list: they can all be placed in block F to reduce wiring congestion.

You can add group statements in the PDS file, immediately preceding the equations segment. The UART_G.PDS file, supplied with the installation software, contains the following group statements.[3]

```
group MACH_SEG_G _2_IO2_I OR PE FE

group MACH_SEG_F RD6 RD5 RD1 RD0 RESET
      _2_IO39_I _2_IO38_I _2_IO37_I _2_IO47_I
      _2_M48_O

group MACH_SEG_E RD4 RD3 RD2 _2_IO33_I
      _2_IO32_I _2_IO31_I

group MACH_SEG_A _55_IDLE _59_TSC1_Q DATA2
      _46_D2 _46_Q2

group MACH_SEG_B DATA5 DATA6 _46_D5 _46_D6
      _46_Q5 _46_Q6

group MACH_SEG_D DATA7 DATA8 _46_D7 _46_D8
      _46_Q7 _46_Q8

group MACH_SEG_C DATA1 DATA3 DATA4 _46_D1
      _46_D3 _46_D4 _46_Q1 _46_Q3 _46_Q4
```

[3] The UART files are located in the directory \PALASM\EXAMPLES\WB\UART.

Note: When you compile the UART_G.SCH design, the resulting PDS file will automatically contain the correct group statements.

You can also group signals at the schematic level by adding node macros and editing part field 2.[4] A schematic reflecting the grouped signals is shown in discussion 3.6, with the name UART_G.SCH.

Important You cannot group signals with different preset or reset product terms in the same block. Each block in the MACH 230 device contains only one unique preset and reset product term.

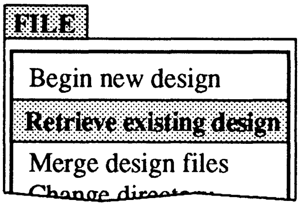
The next discussion describes how to compile the UART_G.PDS design.

[4] Refer to the *PALASM 4 User's Manual*, Section III, Chapter 7, for information on the node macro.

3.4 RE-COMPILING

You use steps below to retrieve the UART_G design and initiate a new compilation process.

To retrieve the design,

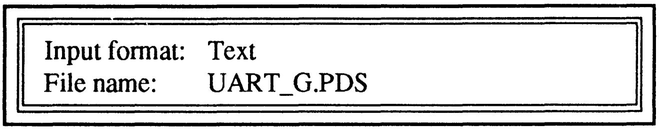


1. Select Retrieve existing design from the File menu.

```
[F2]
  T [Enter]
  UART_G.PDS [F10]
```

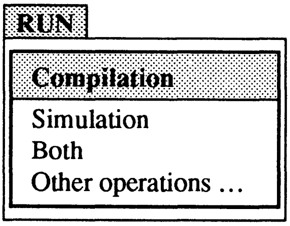
2. Display the submenu and select Text, activate the file-name field, type the name at left and confirm.

The form on your screen should match the one below.



Input format: Text
File name: UART_G.PDS

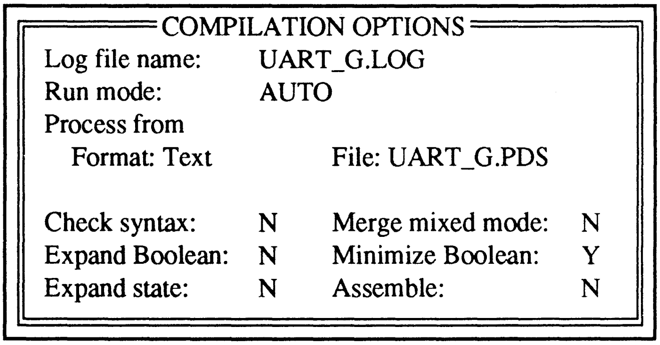
To compile the design,



1. Select Compilation from the Run menu.

Note: If you have a good understanding of the PALASM 4 software, you can use the prompts and figures on the left side of this discussion to quickly advance to discussion 3.5.

The Compilation Options form appears listing various specifications.



```
COMPILATION OPTIONS
Log file name:    UART_G.LOG
Run mode:         AUTO
Process from
  Format: Text            File: UART_G.PDS

Check syntax:     N       Merge mixed mode:   N
Expand Boolean:   N       Minimize Boolean:   Y
Expand state:     N       Assemble:           N
```

For this design, automatic run mode is used. In this mode, a Y after an option in the lower part of the form guarantees the option will execute. The software determines if any o:her options need to be run.

Important: Be sure to complete the steps below to set the execution-log name and run mode.

UART_G.LOG [Enter]

2. Enter the execution-log file name shown at left.

   The Run mode field becomes active.

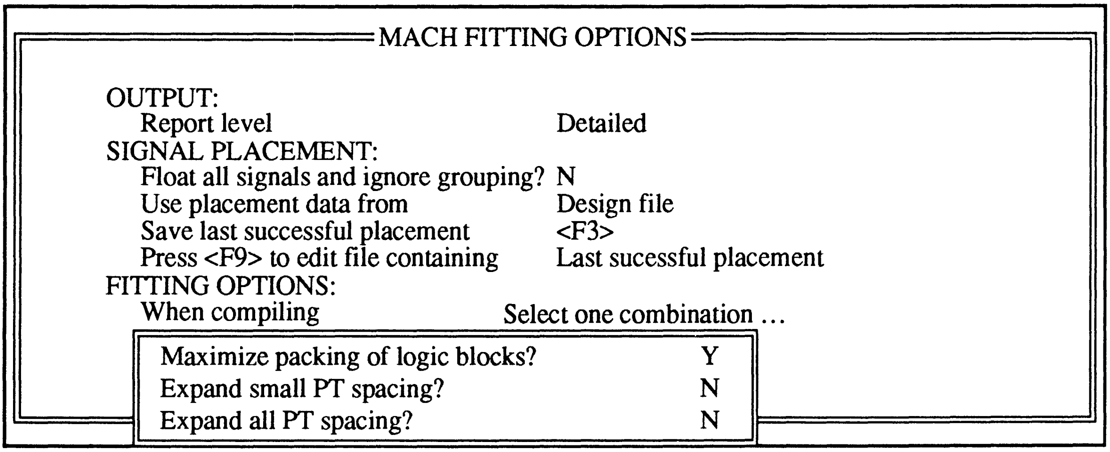
[F2]
A

3. Display the options list and select Automatic.

[F10]

4. Confirm specifications in the form.

   A new form appears showing the MACH fitting options to be used.



```
MACH FITTING OPTIONS
  OUTPUT:
     Report level                           Detailed
  SIGNAL PLACEMENT:
     Float all signals and ignore grouping? N
     Use placement data from                Design file
     Save last successful placement         <F3>
     Press <F9> to edit file containing     Last sucessful placement
  FITTING OPTIONS:
     When compiling                      Select one combination ...
       Maximize packing of logic blocks?             Y
       Expand small PT spacing?                      N
       Expand all PT spacing?                        N
```

[F10]

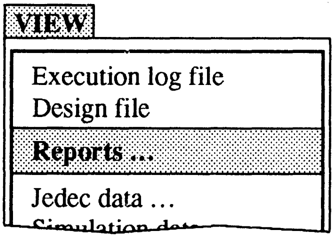
3. Confirm all specifications.

When the process finishes you see that the compilation was successful.

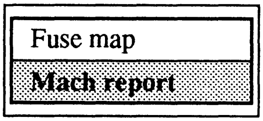
[Esc]

4. Dismiss the process window.

To review the MACH report,



1. Select the Reports ... command from the View menu.



2. Select Mach report from the submenu.

Tip: Again, to print the report you select the Other file command from the Edit menu, enter the name below, and print from the text editor as usual.

UART_G.RPT

3.5 INTERPRET FINAL REPORT

The two discussions here are divided according to report interpretation and the conclusion.

3.5.1 INTERPRETATION

As you might expect, most areas of the MACH report show no significant difference when compared with the initial report. However, certain segments are revealing.

3.5.1.1 Block Partitioning

The first part of this segment, with last equations placed and affinity information, is unchanged. However, partitioning results confirm all node and output signals were placed within MACH 230 logic blocks.

- Signals _2_IO2_I OR PE FE are in block G.
- Signals RD6 RD5 RD1 RD0 RESET _2_IO39_I _2_IO38_I _2_IO37_I _2_IO47_I _2_M48_O are in block F.
- Signals RD4 RD3 RD2 _2_IO33_I _2_IO32_I _2_IO31_I are in block E.
- Signals _55_IDLE _59_TSC1_Q DATA2 _46_D2 _46_Q2 are in block A.
- Signals DATA5 DATA6 _46_D5 _46_D6 _46_Q5 _46_Q6 are in block B.
- Signals DATA7 DATA8 _46_D7 _46_D8 _46_Q7 _46_Q8 are in block D.
- Signals DATA1 DATA3 DATA4 _46_D1 _46_D3 _46_D4 _46_Q1 _46_Q3 _46_Q4 are in block C.

> **Tip**: If you want an even more optimal fit, you can group other signals to free resources for future design revisions.

```
*** Block Partitioning Results

               Array     Macros      # I/O     Buried   Product     Signal
              Inputs     Remain      Macro     Logic     Terms      Fanout
   Block-> A    22          7          2         7         36         13
   Block-> B    10          8          2         6         32          8
   Block-> C     9          9          0         7         28         11
   Block-> D    10          9          0         7         28         13
   Block-> E     9          7          3         6         36          9
   Block-> F    10          6          4         6         40         14
   Block-> G    11         10          4         2         24          7
   Block-> H    11          9          1         6         28         18



*** Block Signal List

Block-> A         _2_M72_I          RCLKOUT        _59_M11_1        _59_M15_2
           _55_PARITY_001       _59_TSC1_Q            _46_Q2            DATA2
                   _46_D2         _55_IDLE

Block-> B             TXD            TXRDY     _55_TLOADCTL        _56_M19_1
                   _46_Q6           _46_Q5            DATA6           _46_D6
                    DATA5           _46_D5

Block-> C        _2_M54_2           _46_Q4           _46_Q3           _46_Q1
                    DATA4           _46_D4            DATA3           _46_D3
                    DATA1           _46_D1

Block-> D          _46_TAG         _2_M56_2          _2_M49_I           _46_Q8
                   _46_Q7            DATA8           _46_D8            DATA7
                   _46_D7

Block-> E        _59_M18_2         _2_M60_2         _2_RDR1_D        _2_IO31_I
                      RD2        _2_IO32_I              RD3        _2_IO33_I
                      RD4

Block-> F         _2_M69_2         _2_M48_O         _2_IO47_I              RD0
                      RD1        _2_IO37_I              RD5        _2_IO38_I
                      RD6        _2_IO39_I

Block-> G         _2_M71_I              RD7               FE               PE
                       OR          _2_IO2_I

Block-> H         _46_M6_I         _2_M54_3          _2_M57_I         _2_M70_2
                  2 M69 1          2 M44 1            RXRDY
```

Additional signal placements are discussed later, as you review the logic map.

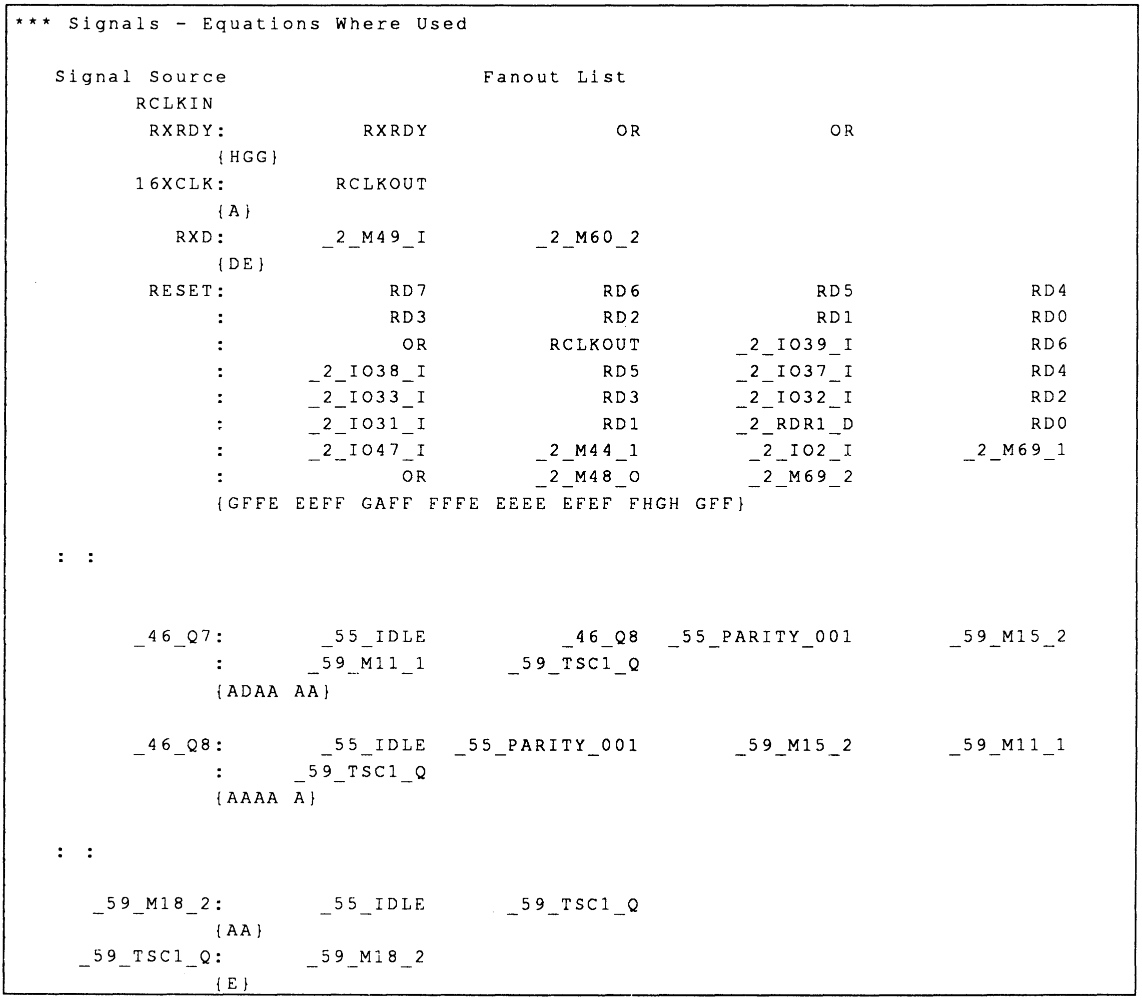
3.5.1.2 Signals, Tabular

The Signals – Tabular Information segment indicates every signal in the UART design has been placed.

```
*** Signals - Tabular Information

          Signal    #    P/N #    (Loc)      Type      Logic   # PT     Blocks
         RCLKIN     1     65       I  4    clock pin     .       .     .........
          RXRDY     2     82       H  0      i/o pin   d-ff      2     .......GH
         16XCLK     3     62       I  3    clock pin     .       .     .A.......
            RXD     4     37       D  6        input     .       .     ....DE...
          RESET     5     56       F 10        input     .       .     .A...EFGH
            RD7     6     66       G  0      i/o pin   d-ff      3     .......G.
        _2_X21_I    7     88       F  6     out pair   d-ff      3     ......F..
        _2_X22_I    8     86       F  4     out pair   d-ff      3     ......F..
        _2_X23_I    9     70       E  4     out pair   d-ff      3     .....E...
        _2_X24_I   10     68       E  2     out pair   d-ff      3     .....E...

      :                  :                    :                :

          _46_Q6   81     32       B 14       buried   d-ff      1     .A..D....
          _46_Q7   82     54       D  4       buried   d-ff      1     .A..D....
          _46_Q8   83     56       D  6       buried   d-ff      1     .A.......
       _56_M19_1   84     31       B 13       buried   d-ff      2     ..B......
         _46_M6_I  85    124       H 10       buried   comb      1     .AB.D....
     _55_TLOADCTL  86     30       B 12       buried   d-ff      1     ........H
   _55_PARITY_001  87      8       A  6       buried   d-ff      3     ..B......
        _59_M15_2  88     17       A 15       buried   comb      1     .A.......
        _59_M11_1  89      9       A  7       buried   d-ff      3     .A.......
        _59_M18_2  90     80       E 14       buried   d-ff      1     .A.......
       _59_TSC1_Q  91      6       A  4       buried   d-ff      1     .....E...
```

3.5.1.3 Signals, Equations

As a result of the grouping, several signals fan out to fewer blocks than before, as illustrated in the table shown next.



```
*** Signals - Equations Where Used

   Signal Source                 Fanout List
        RCLKIN
         RXRDY:          RXRDY              OR              OR
              {HGG}
        16XCLK:        RCLKOUT
              {A}
           RXD:        _2_M49_I        _2_M60_2
              {DE}
         RESET:            RD7             RD6             RD5             RD4
              :            RD3             RD2             RD1             RD0
              :             OR         RCLKOUT       _2_IO39_I             RD6
              :      _2_IO38_I             RD5       _2_IO37_I             RD4
              :      _2_IO33_I             RD3       _2_IO32_I             RD2
              :      _2_IO31_I             RD1       _2_RDR1_D             RD0
              :      _2_IO47_I        _2_M44_1        _2_IO2_I        _2_M69_1
              :             OR        _2_M48_O        _2_M69_2
              {GFFE EEFF GAFF FFFE EEEE EFEF FHGH GFF}

   : :


        _46_Q7:        _55_IDLE          _46_Q8 _55_PARITY_001        _59_M15_2
              :       _59_M11_1      _59_TSC1_Q
              {ADAA AA}

        _46_Q8:        _55_IDLE  _55_PARITY_001       _59_M15_2       _59_M11_1
              :      _59_TSC1_Q
              {AAAA A}

   : :

     _59_M18_2:        _55_IDLE      _59_TSC1_Q
              {AA}
    _59_TSC1_Q:       _59_M18_2
              {E}
```

3.5.1.4 Feedback Map

The feedback map provides an overview of output signals feeding back to drive other outputs. It also provides a visual measure of connectivity requirements. Input signals that could not be routed are not shown. Information here complements information in the logic map.

This map shows how each input and feedback signal is routed through the switch matrix to the array. The numbers on this map correspond to switch-matrix blocks, not to logic-block locations. Switch-matrix blocks feed the PAL arrays, which in turn feed macros through the logic allocator.

If a signal name appears in the feedback map, it means the signal is fed back through the array to drive another equation. Note that most of the signals in the design are fed back to the internal logic.

```
*** Feedback Map - UART_G

 Gbl Inp .--.       I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
         | 0|            | 0|      |25| _46_Q5         CSB : 0|      |25|
         | 1|    _46_D2 : 1|      |24| _46_Q6    _55_PARI  1|      |24|
         | 2|   _59_M11_  2|      |23| _46_Q3      _46_D6 : 2|      |23|
         | 3|    _46_Q4 : 3|      |22| _46_Q1      _46_Q4 : 3|      |22|
         | 4|   _46_M6_I  4|      |21| _46_Q2    _46_M6_I  4|      |21|
         | 5|    _46_Q8 : 5|      |20| _2_M54_        WEB : 5|      |20|
         '--'   _59_M15_  6|      |19| _59_M18              | 6|      |19| _46_Q5
                  RESET : 7|      |18| _46_Q7               | 7|      |18| CLRB
                _2_M56_2  8|      |17| _46_TAG              | 8|      |17|
                 16XCLK : 9|      |16| _55_IDL              | 9|      |16|
                _2_M72_I 10|      |15: _2_M44_     _46_D5 :10|      |15|
                _2_M57_I 11|      |14: _2_M54_             |11|      |14: _56_M19
                         |12|      |13|                     |12|      |13|
                         '--+--C--+--'                     '--+--D--+--'


                      :                    :                       :


                    RD4 : 0|      |25|                 RD1 : 0|      |25| RESET
               _59_TSC1  1|      |24|           _2_M69_1  1|      |24|
               _2_IO32_  2|      |23|                RD6 : 2|      |23|
               _2_IO37_  3|      |22|           _2_IO37_  3|      |22| CSB
               _55_IDLE  4|      |21|                RD0 : 4|      |21|
                        | 5|      |20|                    | 5|      |20|
               _2_IO33_  6|      |19| RD2       _2_M70_2  6|      |19|
                    CSB : 7|      |18|           _2_IO38_  7|      |18| _2_IO39
               _2_M44_1  8|      |17|           _2_M44_1  8|      |17| _2_M71_
                    RXD : 9|      |16| RESET     _2_M69_2  9|      |16| _2_IO47
               _2_IO31_ 10|      |15|                    |10|      |15: _2_RDR1
                      CD :11|      |14|                 CD :11|      |14: RD5
                    RD3 :12|      |13: WEB                 |12|      |13: WEB
                         '--+--G--+--'                     '--+--H--+--'
               _2_M44_1  0|      |25|           _2_M44_1  0|      |25|
                    CSB : 1|      |24|                    | 1|      |24| _2_M56_
                      CD : 2|      |23|           _2_M54_2  2|      |23| _55_TLO
                      PE : 3|      |22|                    | 3|      |22|
                        | 4|      |21| CD        _55_IDLE  4|      |21|
                    WEB : 5|      |20| OR        _2_IO2_I  5|      |20|
                        | 6|      |19| _2_M69_             | 6|      |19|
                    RD7 : 7|      |18| RXRDY        RESET : 7|      |18| _2_M72_
                      FE : 8|      |17| RESET        RXRDY : 8|      |17| _2_IO47
               _2_M69_2  9|      |16|                    | 9|      |16|
               _2_IO39_ 10|      |15: _2_M49_   _2_M54_3 10|      |15: _2_M49_
               _2_M48_O 11|      |14|                    |11|      |14|
                         |12|      |13|                     |12|      |13: _2_M71_
                         '--+-u--u+--'                     '--+-u--u+--'
```

3.5.1.5 Logic Map

The logic map graphically summarizes the assignment of output signals and internal nodes for each block in the MACH device. It also shows the signals assigned to global input pins. This can help you gauge device utilization, distribution of signals among logic blocks, and assignment of signals to macros. During the fitting process, output signals are placed before input signals are routed.

```
*** Logic Map - UART_G

 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
         | 0|  RCLKOUT | 0| 2   |25|               TXRDY | 0| 1   |25|
      CLK| 1|          | 1| .   |24|                     | 1| .   |24|
      CSB| 2|  _55_IDLE  2| 2   |23|                 TXD | 2| 1   |23|
   16XCLK| 3|          | 3| .   |22|                     | 3| .   |22|
   RCLKIN| 4|  _59_TSC1  4| 1   |21|                     | 4| .   |21|
      WEB| 5|   _46_D2 | 5| 1   |20|              _46_D5 | 5| 1   |20|
         '--'  _55_PARI  6| 3   |19|              _46_Q5 | 6| 1   |19|
               _59_M11_  7| 3   |18|              _46_D6 | 7| 1   |18|
                 _46_Q2 | 8| 1   |17|                     | 8| .   |17|
                       | 9| .   |16|                     | 9| .   |16|
                       |10| .  1|15| _59_M15             |10| .  .|15|
                       |11| .  2|14| _2_M72_             |11| .  1|14| _46_Q6
                       |12| .  .|13|             _55_TLOA 12| 1  2|13| _56_M19
                       '--+--C--+--'                     '--+--D--+--'

   :  :

                   RD2 | 0| 3   |25|                 RD0 | 0| 3   |25|
               _2_M60_2  1| 1   |24|             _2_IO38_  1| 1   |24|
                   RD3 | 2| 3   |23|                 RD1 | 2| 3   |23|
                       | 3| .   |22|                     | 3| .   |22|
                   RD4 | 4| 3   |21|                 RD5 | 4| 3   |21|
               _2_IO31_  5| 1   |20|                     | 5| .   |20|
                       | 6| .   |19|                 RD6 | 6| 3   |19|
               _2_IO32_  7| 1   |18|                     | 7| .   |18|
               _2_RDR1_  8| 1   |17|             _2_IO47_  8| 1   |17|
                       | 9| .   |16|             _2_IO37_  9| 1   |16|
                       |10| .  1|15| _2_IO33     _2_M69_2 10| 2  .|15|
                       |11| .  1|14| _59_M18             |11| .  2|14| _2_M48_
                       |12| .  .|13|             _2_IO39_ 12| 1  .|13|
                       '--+--G--+--'                     '--+--H--+--'
                   RD7 | 0| 3   |25|               RXRDY | 0| 2   |25|
                       | 1| .   |24|                     | 1| .   |24|
                    FE | 2| 2   |23|             _2_M69_1  2| 1   |23|
                       | 3| .   |22|                     | 3| .   |22|
                    PE | 4| 3   |21|             _2_M54_3  4| 2   |21|
                       | 5| .   |20|                     | 5| .   |20|
                    OR | 6| 2   |19|             _2_M57_I  6| 2   |19|
                       | 7| .   |18|                     | 7| .   |18|
                       | 8| .   |17|                     | 8| .   |17|
                       | 9| .   |16|                     | 9| .   |16|
                       |10| .  .|15|             _46_M6_I 10| 1  .|15|
                       |11| .  2|14| _2_IO2_             |11| .  1|14| _2_M44_
               _2_M71_I 12| 1  .|13|             _2_M70_2 12| 1  .|13|
                       '--+-u--u+--'                     '--+-u--u+--'
```

3.5.1.6 Device Pin-Out Map

The following device pin-out map was generated during this successful placement.

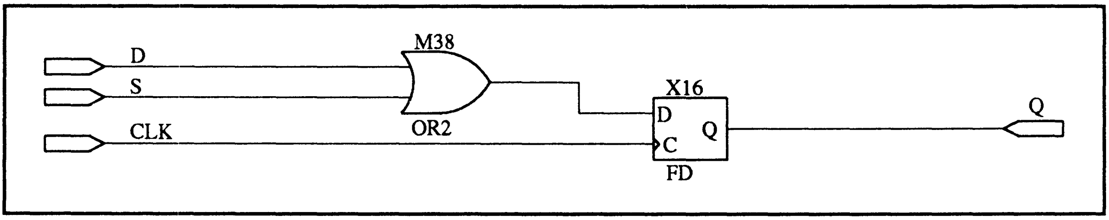
```
*** Pin Map - UART_G
                                             WEB
                               RCLKOUT       | RXRDY
                            _55_IDLE |       | | .
                             DATA2 | |       | | | .
                               . | | |       | | | | .
                             . | | | |       | | | | | .
                           . | | | | |       | | | | | | .
                         . | | | | | |       | | | | | | | .
                       . | | | | | | |       | | | | | | | | .
                       | | | | | | | |       | | | | | | | | |
                .------'-'-'-'-'-'-'-'---o---'-'-'-'-'-'-'-'-'----.
                |    1 1                   8 8 8 8 8 7 7 7 7 7    |
                |    1 0 9 8 7 6 5 4 3 2 1 4 3 2 1 0 9 8 7 6 5    |
                |12                                         Gnd 74|
                |13  G                 V G V                    73|
                |14  n                 c n c                    72|
                |15  d                 c d c                    71|
           DATA6|16                                             70|
           DATA5|17                                             69|OR
             TXD|18                                             68|PE
           TXRDY|19                                             67|FE
                |20                                             66|RD7
                |21 Vcc               MACH-230                  65|RCLKIN
                |22 Gnd                                     Gnd 64|
             CLK|23                                         Vcc 63|
           DATA1|24                                             62|16XCLK
           DATA3|25                                             61|RD0
                |26                                             60|RD1
                |27                                             59|RD5
                |28                                             58|RD6
           DATA4|29                    V G V                 G  57|
                |30                    c n c                 n  56|RESET
                |31                    c d c                 d  55|
                |32 Gnd                                         54|
                |    3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5    |
                |    3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3    |
                '----.-.-.-.-.-.-.-.-.-------.-.-.-.-.-.-.-.------'
                     | | | | | | | | |       | | | | | | | |
                     ' | | | | | | | |       | | | | | | | |
                       ' | | | | | | |       | | | | | | | '
                        CD | | | | | |       | | | | | | '
                        CLRB | | | | |       | | | | | '
                           RXD | | | |       | | | | '
                     _55_IDLECTL | | |       | | | '
                             DATA8 | |       | | RD4
                               DATA7 |       | RD3
                                   CSB       RD2
```

3.5.2 CONCLUSION

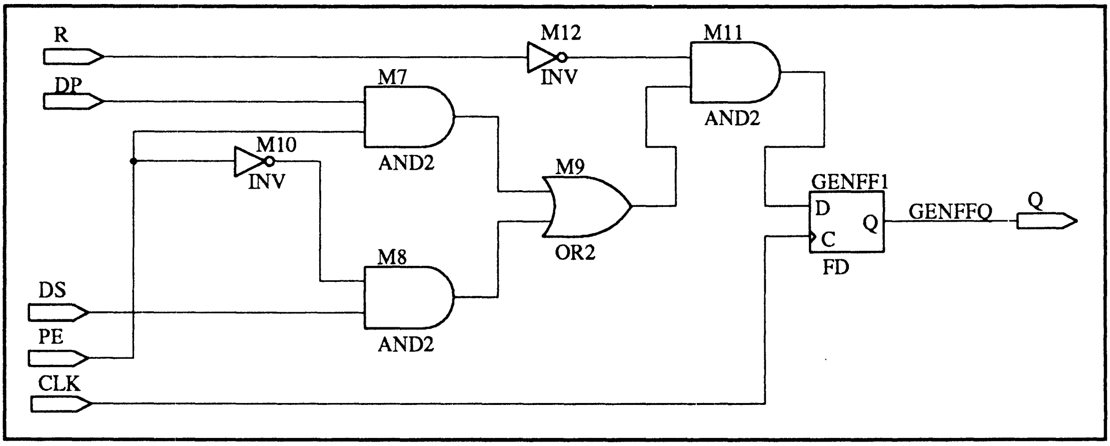
We see that grouping signals eases the wiring congestion in the UART design, enabling the design to fit into a MACH 230 device.

3.6 SCHEMATICS

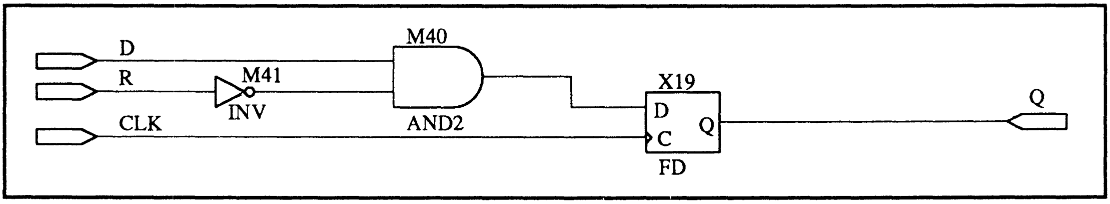
Figures of each schematic in the UART design are included next. The top-level schematic is shown twice: first without grouping at the beginning of this section, and then with grouping in place at the end of this section.



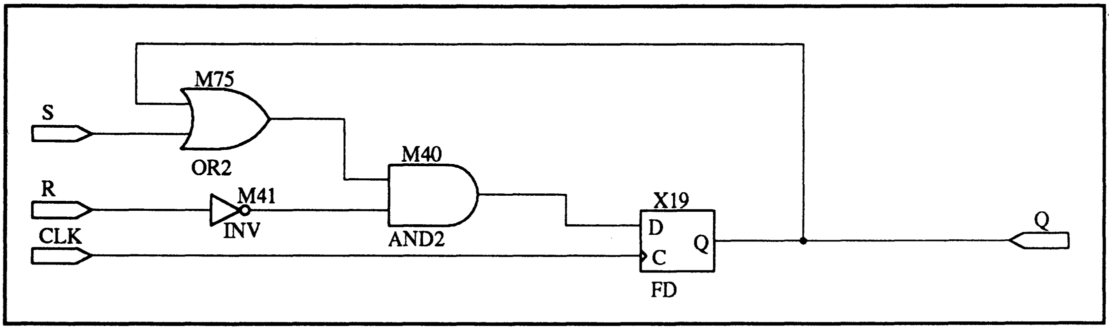
FFDS



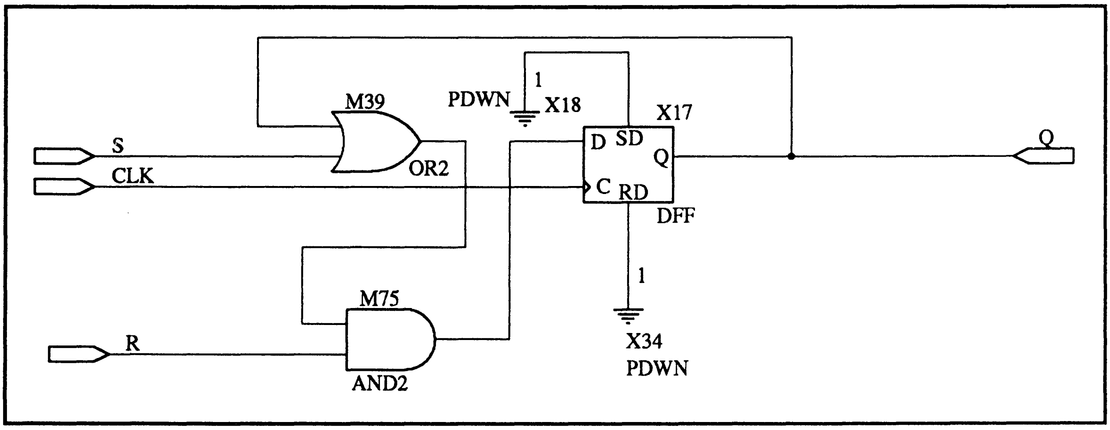
FFLDR



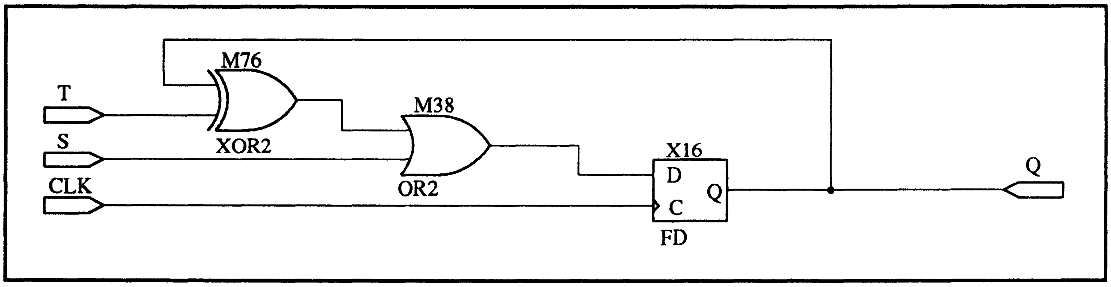
FFR



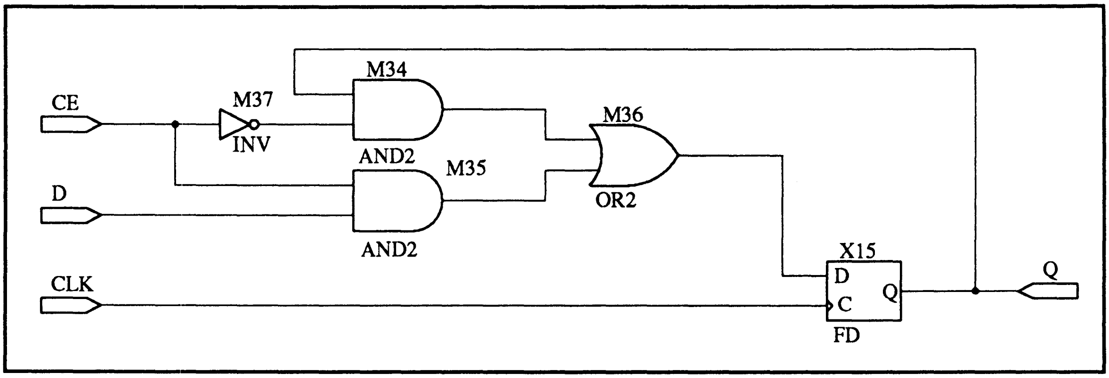
FFSR



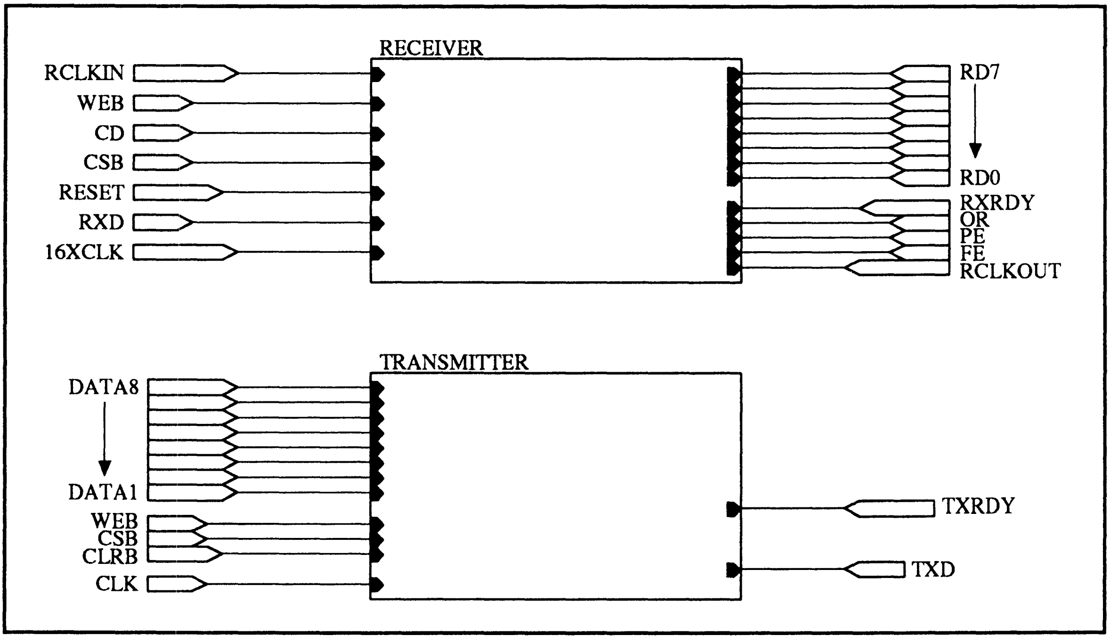
FFSRD



FFT

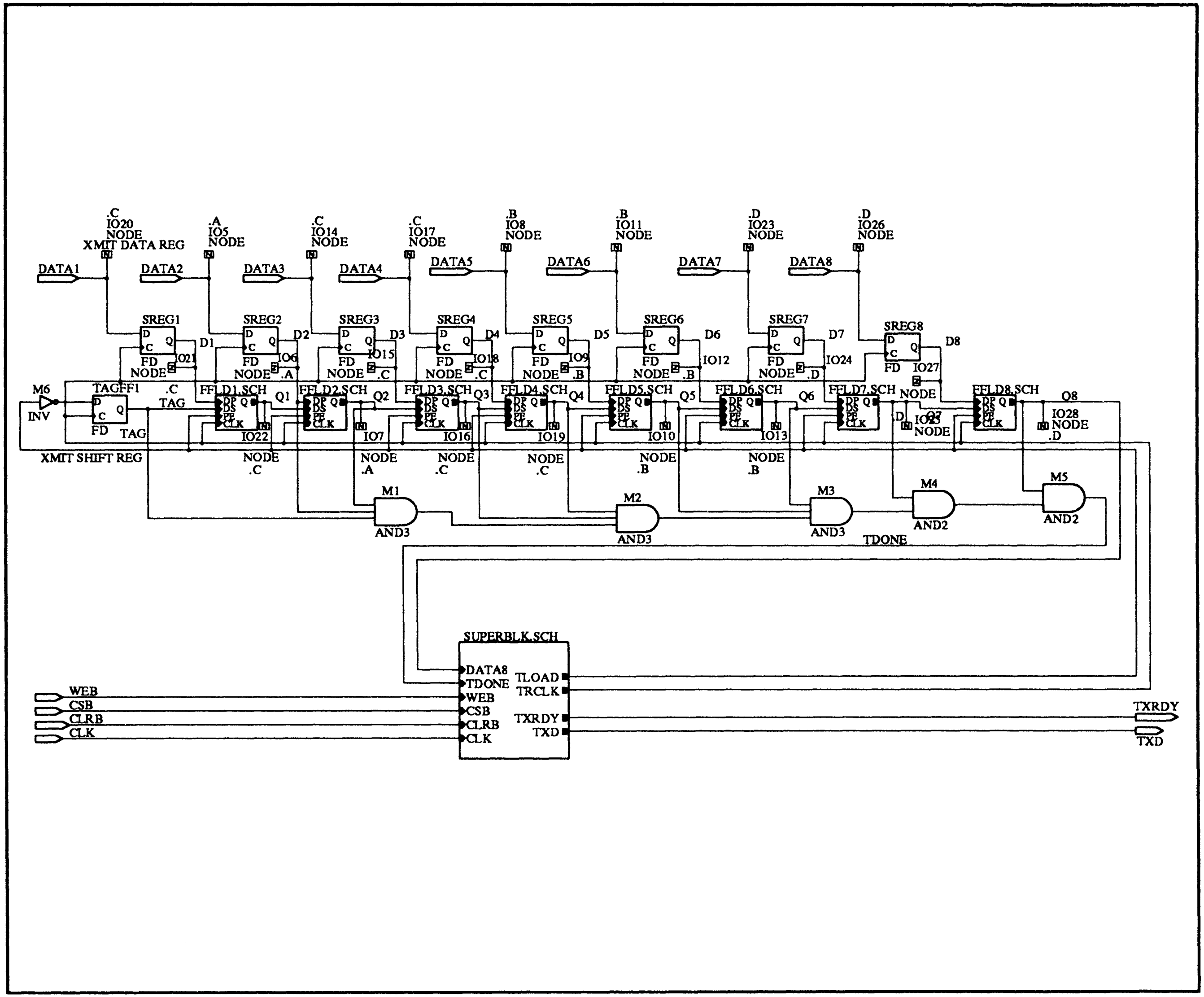

CE
D
CLK
M37
INV
M34
AND2
M35
AND2
M36
OR2
X15
D
C
Q
FD
Q

RDR

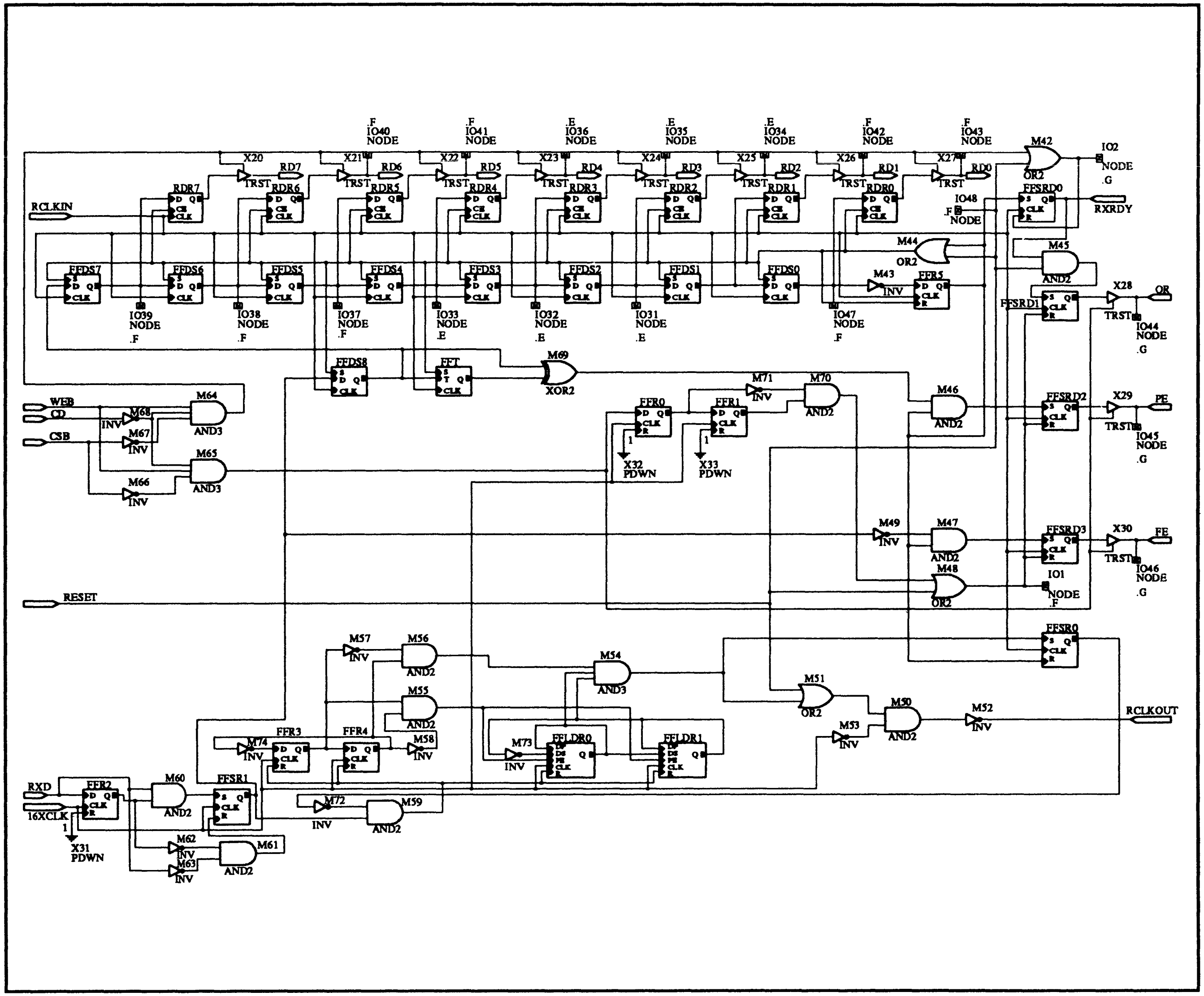

RECEIVER
RCLKIN
WEB
CD
CSB
RESET
RXD
16XCLK
RD7
RD0
RXRDY
OR
PE
FE
RCLKOUT
TRANSMITTER
DATA8
DATA1
WEB
CSB
CLRB
CLK
TXRDY
TXD

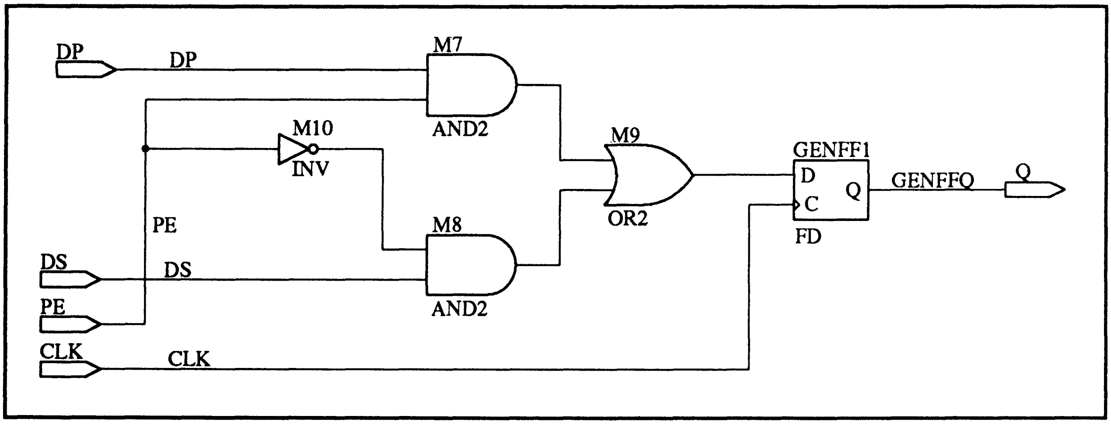
UAT_G

UART_TOP

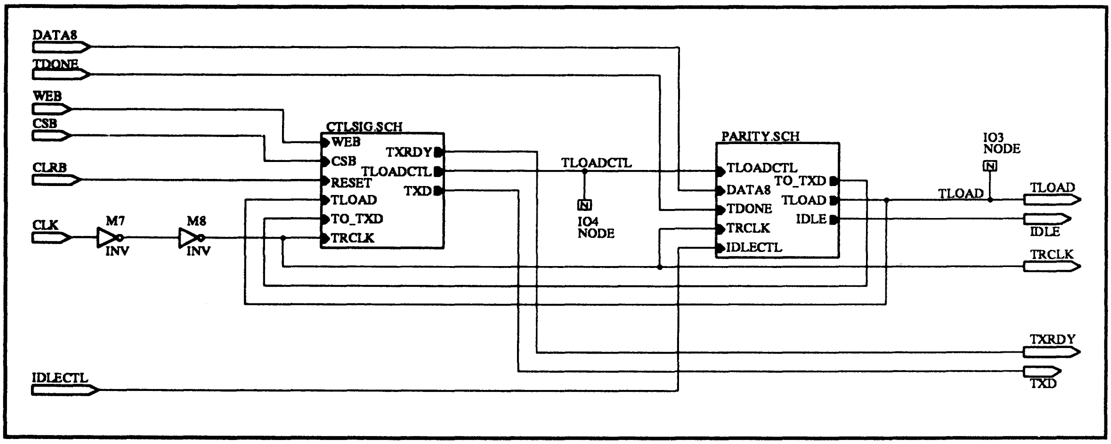

XMIT DATA REG
.C IO20 NODE
.A IO5 NODE
.C IO14 NODE
.C IO17 NODE
.B IO8 NODE
.B IO11 NODE
.D IO23 NODE
.D IO26 NODE
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
SREG1
SREG2
SREG3
SREG4
SREG5
SREG6
SREG7
SREG8
FD
D1
D2
D3
D4
D5
D6
D7
D8
IO21
IO6
IO15
IO18
IO9
IO12
IO24
IO27
NODE .A
NODE .C
NODE .B
NODE .D
M6
INV
TAGFF1
TAG
FFLD1.SCH
FFLD2.SCH
FFLD3.SCH
FFLD4.SCH
FFLD5.SCH
FFLD6.SCH
FFLD7.SCH
FFLD8.SCH
DP
DS
PE
CLK
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
IO22
IO7
IO16
IO19
IO10
IO13
IO25
IO28
XMIT SHIFT REG
M1
M2
M3
M4
M5
AND3
AND2
TDONE
SUPERBLK.SCH
DATA8
TDONE
WEB
CSB
CLRB
CLK
TLOAD
TRCLK
TXRDY
TXD
WEB
CSB
CLRB
CLK
TXRDY
TXD

REC


RCLKIN
RDR7
RDR6
RDR5
RDR4
RDR3
RDR2
RDR1
RDR0
X20
X21
X22
X23
X24
X25
X26
X27
TRST
RD7
RD6
RD5
RD4
RD3
RD2
RD1
RD0
.F IO40 NODE
.F IO41 NODE
.E IO36 NODE
.E IO35 NODE
.E IO34 NODE
.F IO42 NODE
.F IO43 NODE
M42
OR2
IO2 NODE .G
FFSRD0
RXRDY
IO48 .F NODE
M44
OR2
M45
AND2
FFDS7
FFDS6
FFDS5
FFDS4
FFDS3
FFDS2
FFDS1
FFDS0
M43
INV
FFR5
IO39 NODE .F
IO38 NODE .F
IO37 NODE .F
IO33 NODE .E
IO32 NODE .E
IO31 NODE .E
IO47 NODE .F
FFSRD1
X28
OR
TRST
IO44 NODE .G
FFDS8
FFT
M69
XOR2
M71
INV
M70
AND2
WEB
CD
CSB
M64
AND3
M68
INV
M67
INV
M65
AND3
M66
INV
FFR0
FFR1
X32 PDWN
X33 PDWN
M46
AND2
FFSRD2
X29
PE
IO45 NODE .G
M49
INV
M47
AND2
FFSRD3
X30
FE
IO46 NODE .G
M48
OR2
IO1 NODE .F
RESET
FFSR0
M57
INV
M56
AND2
M54
AND3
M55
AND2
M51
OR2
M50
AND2
M52
INV
RCLKOUT
M53
INV
FFR3
FFR4
M74
INV
M58
INV
M73
INV
FFLDR0
FFLDR1
RXD
16XCLK
FFR2
X31 PDWN
M60
AND2
FFSR1
M72
INV
M59
AND2
M62
INV
M63
INV
M61
AND2

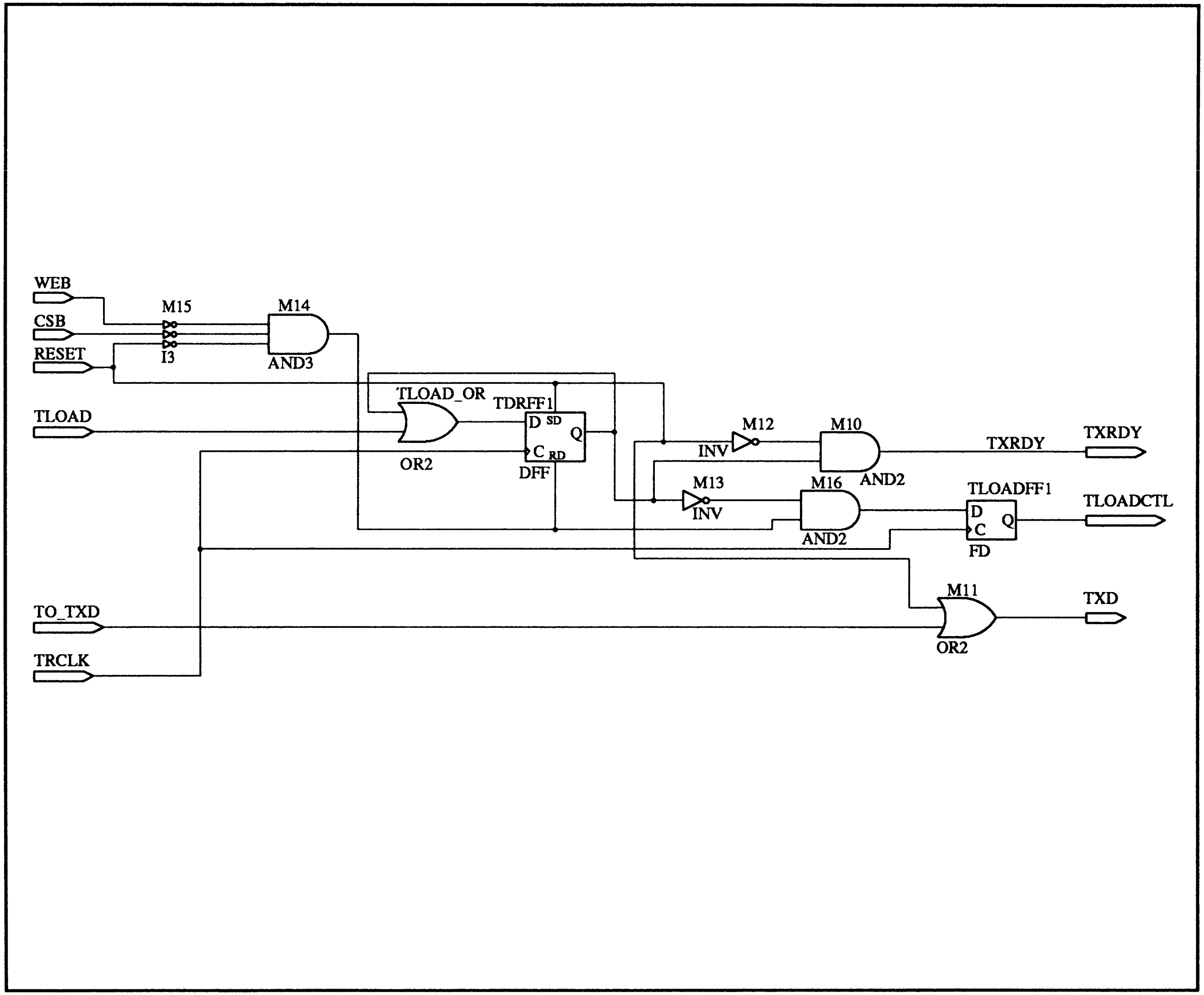


FFLD

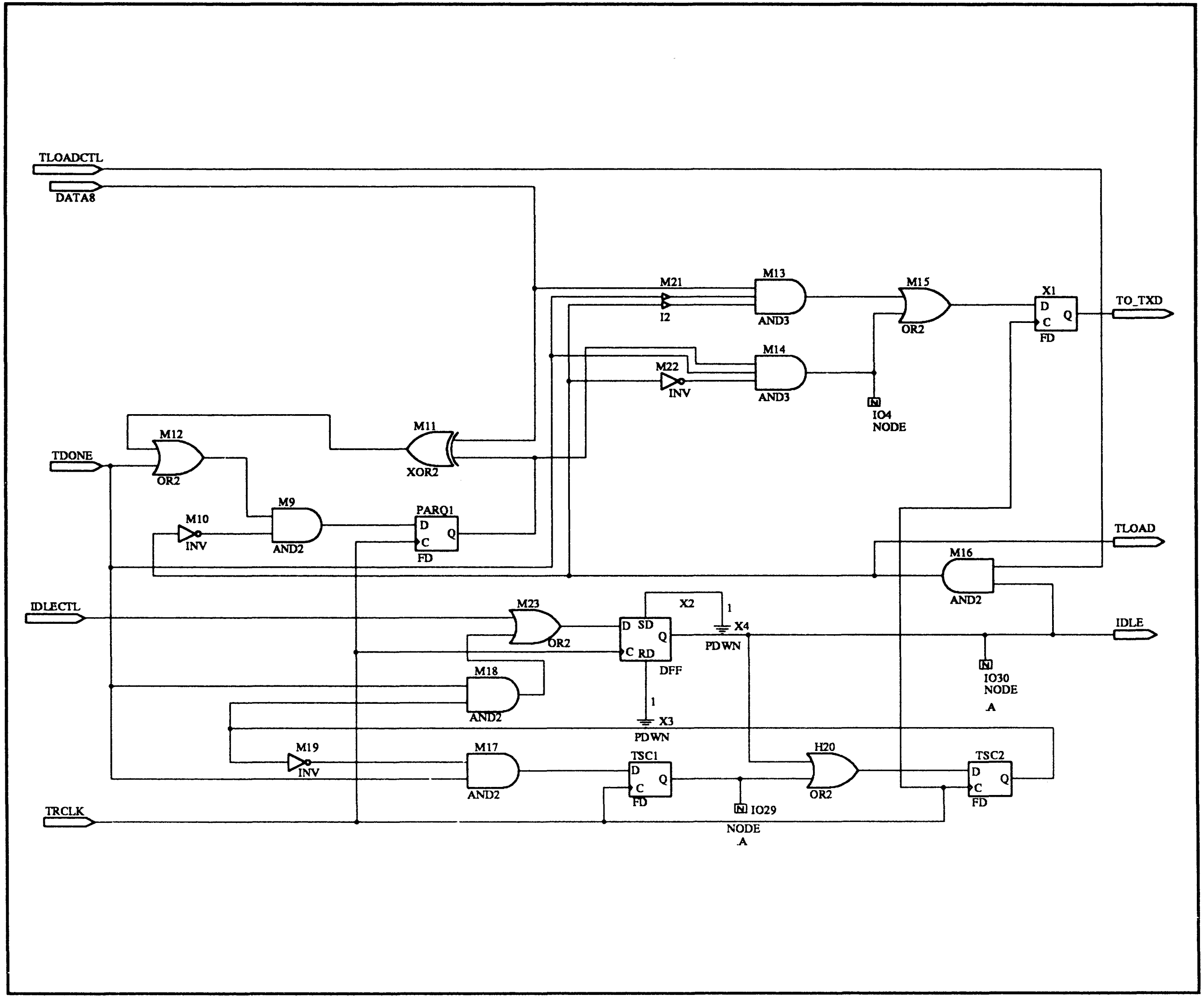


SUPERBLK

CTLSIG


WEB
CSB
RESET
M15
I3
M14
AND3
TLOAD
TLOAD_OR
OR2
TDRFF1
D SD
Q
C RD
DFF
M12
INV
M10
AND2
TXRDY
TXRDY
M13
INV
M16
AND2
TLOADFF1
D
C
Q
FD
TLOADCTL
M11
OR2
TXD
TO_TXD
TRCLK

PARITY


TLOADCTL
DATA8
M21
I2
M13
AND3
M15
OR2
X1
D
Q
C
FD
TO_TXD
M14
AND3
M22
INV
IO4
NODE
M11
XOR2
M12
OR2
TDONE
M9
AND2
M10
INV
PARQ1
D
C
Q
FD
TLOAD
M16
AND2
IDLECTL
M23
OR2
X2
D SD
Q
C RD
DFF
1
X4
PDWN
IDLE
IO30
NODE
.A
M18
AND2
1
X3
PDWN
M19
INV
M17
AND2
TSC1
D
C
Q
FD
H20
OR2
TSC2
D
C
Q
FD
IO29
NODE
.A
TRCLK

MACH Design Workbook Index

NOTES

NOTES

NOTES

NOTES

ADVANCED
MICRO
DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale
California 94088-3453
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL-FREE
(800) 538-8450

APPLICATIONS HOTLINE &
LITERATURE ORDERING
(800) 222-9323
(408) 749-5703

SUPPORT PRODUCTS
ENGINEERING HOTLINE
JAPAN 0-031-11-1129
UK 0-800-89-1131
USA (800) 222-9323

RECYCLED &
RECYCLABLE

Printed in USA
CPS-3M-6/92-0
17044A