ST9 FAMILY 8/16 BIT MCU

TECHNICAL MANUAL

1st EDITION

SGS-THOMSON MICROELECTRONICS

ST9 FAMILY TECHNICAL MANUAL
SGS-THOMSON MICROELECTRONICS

4660

ST9 FAMILY
8/16 BIT MCU

TECHNICAL MANUAL

1st EDITION

NOVEMBER 1991

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED

SGS-THOMSON PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF SGS-THOMSON Microelectronics. As used herein:

1. Life support devices or systems are those which (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonably expected to result in significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can reasonably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

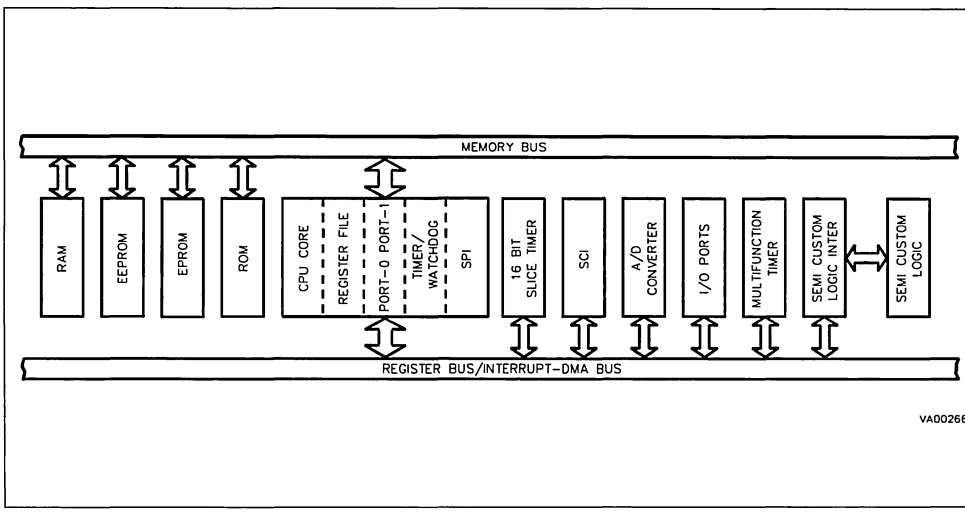
1.1 THE ST9 AND MACROCELLS

The ST9 is based on the twin ideas of simplicity and flexibility. To achieve these ends, an approach was adopted that uses an advanced core with standard cell expansion. This approach, by simplifying internal communication and architecture, allows on-chip customization. Among the benefits are reduced operation times, reduced production costs, and low power consumption, the typical advantages of a good design, and adaptation to customer requirements. ST9 devices are produced using an advanced HCMOS process which ensures high speed, high density and good reliability.

The standard cells (or Megacells) are seen as the basic building blocks of the system and include such units as the Core, A/D Converter, Serial Communications Interface (SCI) 16 bit Multi-function Timer with input capture/output compare capability, memory cells (ROM, EPROM, RAM, EEPROM) and others coming. All these functions are envisaged as a library of dedicated functions from which customers may choose their own preferred selection,according to their application needs.

The block diagram (figure 1.1) shows the simplicity of the standard cell approach, with the two internal buses and the physical design layout of the ST9. This gives the system the ability to easily add on extra cells of any type from the library (as dedicated demands or future developments).

Figure 1-1. ST9 Architecture



1.2 ST9 BASIC FEATURES

The ST9 Core is based on a CPU which offers the advantages commonly associated with a register based micro-computer architecture. The main features are :

- 87 types of Instruction
- bit, byte and word operations
- 14 addressing modes
- 224 byte register file
- 64 possible pages (of 16 registers each)
- a fully expandable Interrupt controller (up to 126 different vectors)
- On-chip DMA channels
- 24 MHz external clock
- 128k (64k Program + 64k Data) of Memory Addressing Space

Two basic macrocells are also included in the ST9 Core :

- 16 bit Timer/Watchdog (with an 8 bit prescaler)
- A Serial Peripheral Interface (SPI)

Peripherals, memories and I/O ports can be included in ST9 products:

- Expandable number of I/O ports (minimum 1),
- 0 (ROMless) to 32k on-chip ROM/EPROM
- Up to 2k byte RAM
- Up to 1k byte of EEPROM
- 16 bit Multi-Function Timer (8 bit prescaler) with 2 input capture and 2 output compare capability
- 8 bit /8 channels A/D Converter with fast conversion time (11µs)
- Fully programmable asynchronous/synchronous Serial Communication Interface.

Moreover, specific and customized macrocells can be designed on a ST9 product upon request.

Development tools are provided to support application development phase as :

- TOTEM Emulator *
- Software Simulator(MS-DOS, VMS, UNIX operating systems)
- Macro assembler (MS-DOS, VMS, UNIX operating systems)
- ANSI C-Compiler (on MS-DOS, VMS, UNIX operating system)
- ST9 products are available in 40/48 Plastic and Ceramic DIL ; 44/68/84 pin PLCC and CLCC, 80QFP.

* TOTEM : The Total Emulator can emulate all the available on chip peripherals with an add-in board and can address the full addressing space.

The wide range of instructions facilitates full use of the register file and address spaces, hence reducing operation times, while the register pointer mechanism allows an unmatched code efficiency and ultrafast context switching. A particularly notable feature is the comprehensive "Any Bit, Any Register" (ABAR) addressing capability of the Boolean instructions.

1.3 ST9 CORE

The ST9 Core includes the CPU, that is the microcode sequencer, the data path, the interrupt/DMA controller and the Register File. Two peripherals are also included: an SPI and a Timer/Watchdog which are present in all ST9 products.

1.4 ST9 MACROCELLS

The modular approach of the ST9 allows easy implementation of application oriented microcontrollers simply by integrating on-chip peripherals (macrocells) and dedicated functions from the existing library, thus creating in a single package systems that previously required a microcontroller plus additional peripheral circuits. Here following is the list of the existing macrocells. Other standard and dedicated functions can be added to the already existing library.

1.4.1 I/O Ports

All the 8 bit on-chip I/O ports can be bit programmed. This allows configuration of individual bits as input, bidirectional, output or alternate function.
Input level and output configuration can also be selected bit by bit by choosing within TTL/CMOS level (input) and Push-Pull, open drain, weak push-Pull driving capabilities. Alternate function selection (for output only) allows the specific pin to be used to connect an on-chip peripheral output.
Port 0 and 1 are used when required, to interface with external memories. Port 0 is present in all ST9 products.

1.4.2 STANDARD PERIPHERALS

The following peripherals are available :

- 16 bit Multi-Function Timer
- A/D converter
- SCI interface
- Memory Bank Switch

1.4.2.1 16 BIT MULTI-FUNCTION TIMER

The ST9 can have up to eight 16-bit multi-function timer units each with its own 8-bit prescaler. Each counter/timer has two load/capture and two compare registers and can operate in a wide variety of modes using internal or external clocks and trigger events which may be set to be level or edge sensitive. Each multi-function timer is controlled by two dedicated control registers and is provided with 2 input and 2 output pins.

1.4.2.2 ANALOG TO DIGITAL CONVERTER

The ST9 peripherals include an 8-bit A/D converter with a conversion time of 11µs, 8 multiplexed analog inputs, and separate analog Vss and Vcc pins. Additional features such as the Programmable Auto Scanning mode and the window Detector are supported to give high performance.

1.4.2.3 SERIAL COMMUNICATION INTERFACE (SCI)

The SCI allows handling of the great variety of asynchronous serial communication formats available, and provides means for a single, high speed, synchronous link. To achieve this the ST9 SCI allows full duplex character-oriented synchronous and asynchronous operation and offers fully programmable serial interface characteristics (programmable baud rate and extensive internal and external error detection and location capabilities). Other notable features include a programmable address indication bit which provides efficient use of other microcontroller/processors in network arrangements, double buffering for transmission and reception, and an off-chip clock capability using programmable I/O ports.

1.4.2.4 MEMORY BANK SWITCH

The Memory Bank Switch (BS) is an address expander allowing ST9 device to extend its addressing range from 64k program plus 64k data memory up to 8M program plus 8M data memory organized in 32k byte segments mapped from the address 8000h up to FFFFh, and a 32k byte common segment (segment0) in the address range 0000h-7FFFh.

1.4.3 Memory Options

Two memory spaces can be addressed by ST9 : program memory and data memory. Four different kinds of memory can be available on ST9 :

- ROM : up to 32k byte (with 4k byte block step)
- EPROM : as for ROM. Used to support the development phase of the application
- RAM : up to 2k byte (by 128 byte step). RAM blocks can be added (as peripherals) up to 2k bytes.
- EEPROM : available in 256 byte steps up to 1k byte.

ST9 SELECTOR GUIDE

DEVICE	ROM	EPROM	RAM	EEPROM	TWD	SPI	MFT	SCI	A/D	BS	MAX I/O	HSHK	PACKAGE
ST9026	16K		256		1	1	1	1			40	1	PDIP48
ST9027	16K		256		1	1	1	1			32	1	PDIP40
ST9028	16K		256		1	1	1	1			36	1	PLCC44
ST90E26		16K	256		1	1	1	1			40	1	CDIP48-W
ST90E27		16K	256		1	1	1	1			32	1	CDIP40-W
ST90E28		16K	256		1	1	1	1			36	1	CLCC44-W
ST90T26		16K	256		1	1	1	1			40	1	PDIP48
ST90T27		16K	256		1	1	1	1			32	1	PDIP40
ST90T28		16K	256		1	1	1	1			36	1	PLCC44
ST90R26	-	-	256		1	1	1	1			32	1	PDIP48
ST9030	8K				1	1	2	1	1		56	1	PLCC68
ST9031	8K				1	1	2	1	1		38	1	PDIP48
ST90E30		8K			1	1	2	1	1		56	1	CLCC68-W
ST90E31		8K			1	1	2	1	1		38	1	CDIP48-W
ST90T30		8K			1	1	2	1	1		56	1	PLCC68
ST90T31		8K			1	1	2	1	1		38	1	PDIP48
ST90R30	-	-			1	1	2	1	1		40	1	PLCC68
ST9036	16K		256		1	1	2	1	1		40	1	PLCC68/QFP80
ST9040	16K		256	512	1	1	2	1	1		56	1	PLCC68
ST90E40		16K	256	512	1	1	2	1	1		56	1	CLCC68-W
ST90T40		16K	256	512	1	1	2	1	1		56	1	PLCC68
ST90R40	-	-	256	512	1	1	2	1	1		40	1	PLCC68
ST90R50	-	-			1	1	3	2	1	1	56	2	PLCC84
ST9054	32K		1280		1	1	3	2	1	1	72	2	PLCC84
ST90E54		32K	1280		1	1	3	2	1	1	72	2	CLCC84-W
ST90T54		32K	1280		1	1	3	2	1	1	72	2	PLCC84

Note: ALL devices have 256 byte Register File with 224 General Purpose Registers (Accumulators/RAM).

Key :

TWD	Timer/Watchdog	SCI	Serial Communications Interface
SPI	Serial Peripheral Interface	A/D	8 bit 8 channel A/D Converter
MFT	Multi-Function Timer	BS	Bankswitch logic 16M byte address range
I/O	In : TTL/CMOS, Out: OD/PP	HSHK	# Ports with Handshake capability
	Alternate Functional Peripheral		

2.1 ADDRESS SPACES

The ST9 has three separate address spaces:

- Register File: 240 8-bit registers plus up to 64 pages of 16 bytes each, located in the on-chip peripherals.
- Data memory with up to 64k (65536) bytes
- Program memory with up to 64k (65536) bytes

2.1.1 CPU Register File

The Register File consists of 240 registers divided into 15 groups of 16 registers plus paging facilities based on the top group (group F, R240-R255).

The groups of 16 registers can be referred to by their hexadecimal group address, so that registers R0-R15 form group 0, registers R16-R31 form group 1, R160-R175 form group Ah and so on. Group Eh (registers R224-R239) is the system register group.

Figure 2-1. Address Spaces

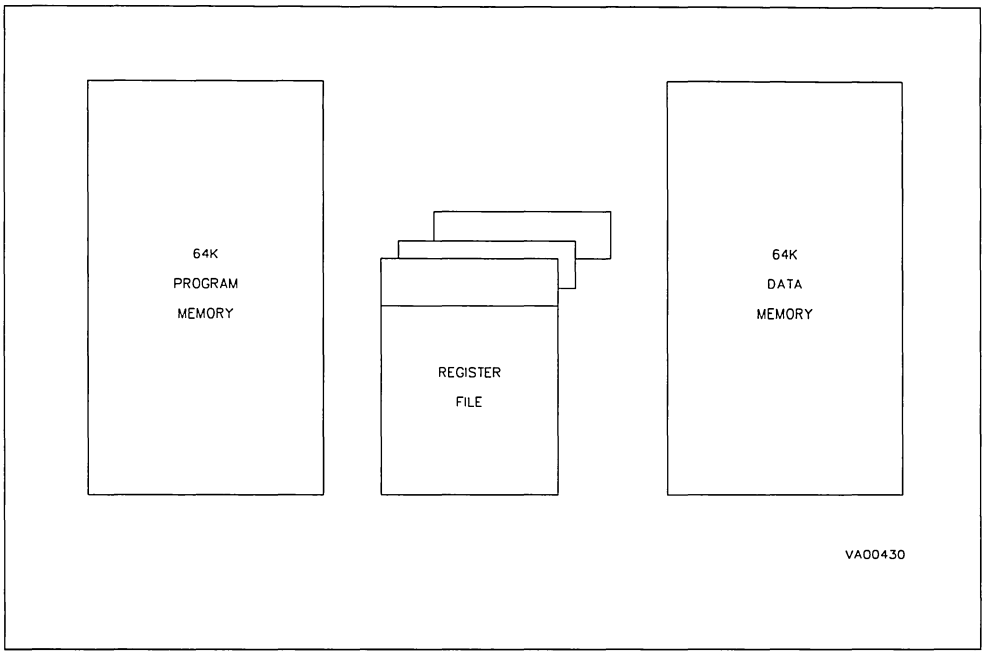


Figure 2-2. Addressing The Register File

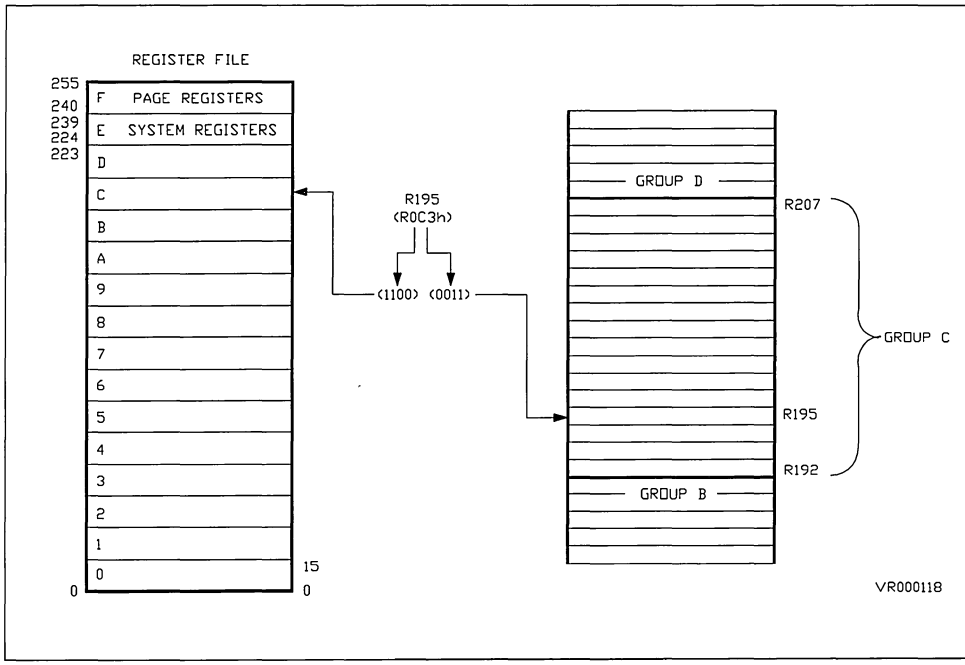
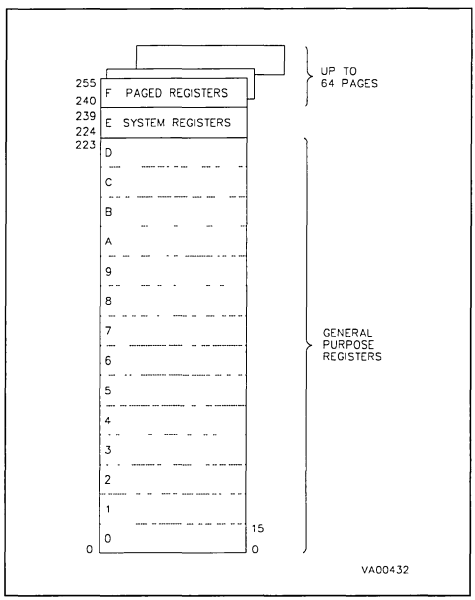


Figure 2-3. Register Grouping



All registers in the Register File and pages can be specified using a decimal, hex or binary address, e.g. R40 or R28h or R00101000b

Working Register Addresses

The 8-bit Register address is formed by 2 nibbles, for example, for register C3h or R195 or R11000011, 1100 specifies the 13th group (i.e. group C) and 0011 specifies the 3rd register in that group.

Working registers are addressed by supplying the least significant nibble in the instruction and adding it to the most significant nibble found in the Register Pointer. Working register addressing is shown in figures 2.3.

2.1.2 System Registers

The 16 system registers form group E (i.e. R224-R239) and are shown in the figure 2.4.

System registers are addressable using any of the 4 register addressing modes (paragraph 2.2) and the most significant nibble will, in all cases, be 14 (0Eh,1110 bin).

Figure 2-4. Single Group of 16 Working Registers

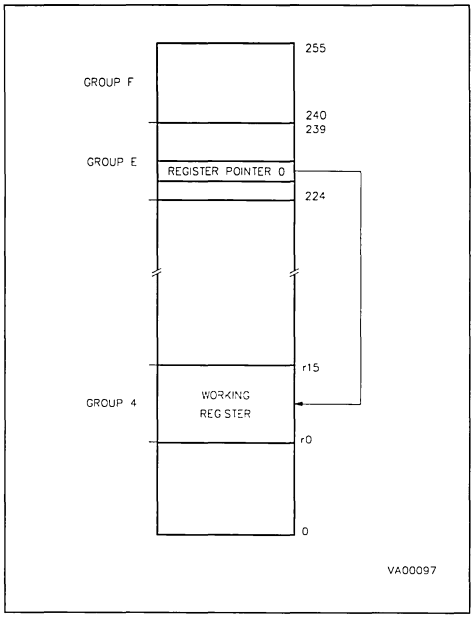
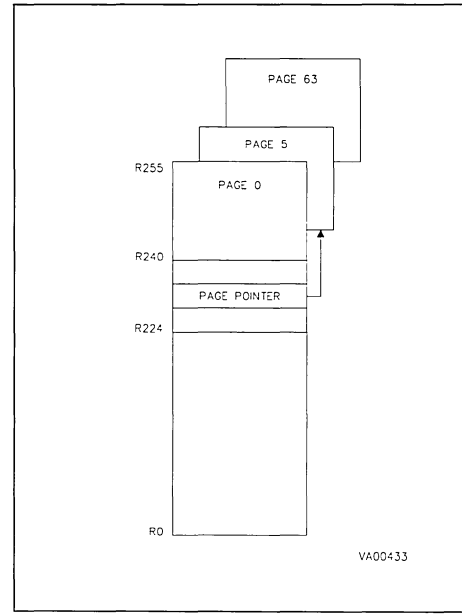


Figure 2-5. Page Pointer Configuration



2.1.3 Paged Registers

There are a maximum of 64 pages each containing 16 registers. These are addressed using the register addressing modes with the addition of the Page Pointer register, R234 (0EAh). This register selects the page to be addressed in group F and once set, does not need to be changed if two or more registers on the same page are to be addressed in succession.

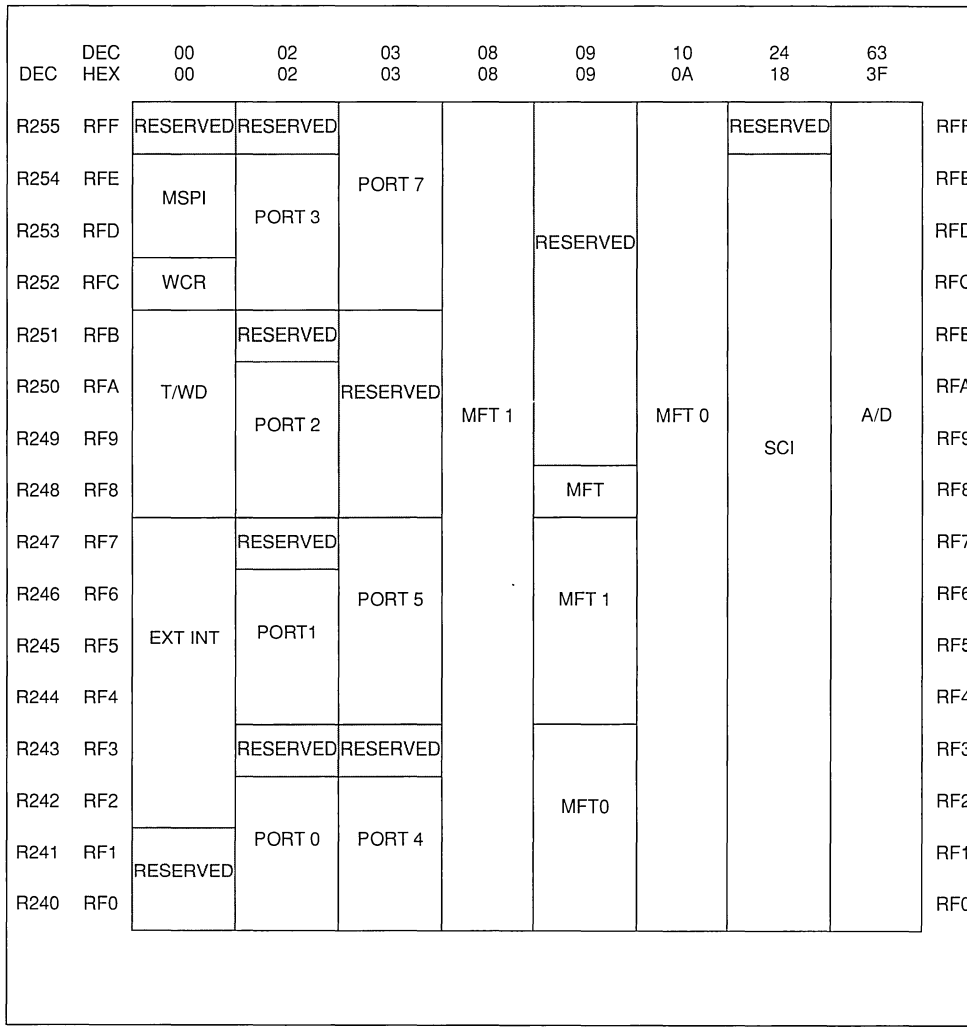
Therefore if the Page Pointer, R234 (0EAh), is set to 5, the instruction `ld R242, r4` will load the contents of working register r4 into the third register (R242, R0F2h) of page 5.

These paged registers hold data and control registers related to the on-chip peripherals, and thus the configuration depends upon the peripheral organisation of each ST9 family member. i.e. pages only exist if the silicon of the peripheral exists on the ST9.

Figure 2-6. System Registers

R239 (ROEFh)	SYS STACK POINTER LOW
R238	SYS. STACK POINTER HIGH
R237	USER STACK POINTER LOW
R236	USER STACK POINTER HIGH
R235	MODE REGISTER
R234	PAGE POINTER
R233	REGISTER POINTER 1
R232	REGISTER POINTER 0
R231	FLAGS
R230	CENTRAL INT. CNTL REG
R229	PORT5
R228	PORT4
R227	PORT3
R226	PORT2
R225	PORT1
R224 (ROEOh)	PORT0

Figure 2-7. Example, ST9030 Group F Peripheral Organisation



2.1.4 Memory

There is a total of 128k bytes of addressable memory space, Program and Data memory, 64 Kbytes each.

The 16 bit address may be supplied directly using the absolute memory location address or indirectly using a pair of registers. In addition the address can be given by an indexed mode when a short (byte) or long (word) offset is added to an indirect base word address.

Before either program or data area is addressed, one of the two instructions `sdm` and `spm` (Set Data Memory and Set Program Memory) should be used. It is not necessary to use either `sdm` or `spm`:

- when operating external stacks where the data memory is automatically used

- when using the memory-indirect to memory-indirect post increment addressing mode when the memory types are specified in the instruction (i.e. `ldpd`, load from data memory to program memory. see chapter 3).

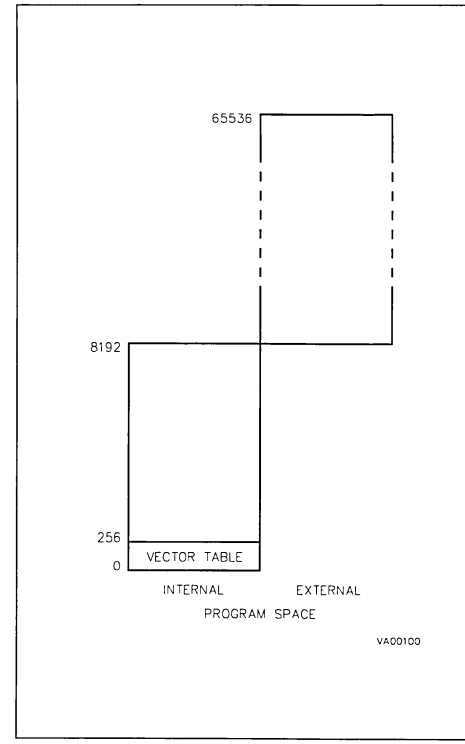
Either the Data Memory or the Program memory can be addressed using memory addressing modes as described in Table 2.1.

2.1.4.1 PROGRAM MEMORY

The program memory size can be up to 64k bytes. Access to the external program memory is allowed ONLY for addresses greater than the existing on-chip program memory. For example, when having 8192 bytes of on-chip program memory, external memory fetches are performed at addresses above location 8192 (see figure 2.4a).

Within the program memory address space, the first 256 locations (0-255) can be used for the interrupt vector table. (Locations 00h, 01h for the Reset Vector; 02h and 03h for the Divide by zero Vector and 04h, 05h for the Top level interrupt vector). Apart from this case no other part of the Program memory has a predetermined function.

Figure 2-8. External Program Memory



2.1.4.2 DATA MEMORY

The data memory maximum size is 64k bytes and has exactly the same addresses and addressing modes as the program memory, the spaces being distinguished by the use of memory setting commands (`sdm`, Set Data Memory).

2.2 ADDRESSING MODES

The ST9 offers a wide variety of established and new addressing modes and combinations to facilitate full and rapid access to the address spaces while reducing program length. The available addressing modes are shown in Table 2-1:

Single operand arithmetic, logic and shift byte instructions have direct register and indirect register addressing modes. For a full list of the possible combinations for each instruction type, please refer to chapter 3 (Instruction Set) or to the ST9 Programming Manual.

Table 2-1. ST9 Addressing Modes

Operand is in	Addressing Mode	Destination Location	Notation
Instruction	Immediate	Byte Word	#N #NN
Register File	Direct	Byte Word	r rr
	Indirect	Byte/Word	(r)
	Indexed	Byte/Word	N(r)
	Indirect Post-Increment	Byte	(r)+
Program or Data Memory	Direct	Byte/Word	NN
	Indirect	Byte/Word	(rr)
	Indirect Post-Increment	Byte/Word	(rr)+
	Indirect Pre-Decrement	Byte/Word	-(rr)
	Short Indexed	Byte/Word	N(rr)
	Long Indexed	Byte/Word	NN(rr)
	Register Indexed	Byte/Word	rr(rr)
Any bit of any working register	Direct	Bit	r.b
Any bit in program or data memory	Indirect	Bit	(rr).b

Two Operands Arithmetic and Logic Instructions	
Destination	**Source**
Register Direct	Register Direct
Register Direct	Register Indirect
Register Direct	Memory Indirect
Register Direct	Memory Indexed
Register Direct	Memory Indirect with Post-Increment
Register Direct	Memory Indirect with Pre-Decrement
Register Direct	Memory Direct
Register Indirect	Register Direct
Memory Indirect	Register Direct
Memory Indexed	Register Direct
Memory Indirect with Post-Increment	Register Direct
Memory Indirect with Pre-Decrement	Register Direct
Memory Direct	Register Direct
Register Direct	Immediate
Memory Direct	Immediate
Memory Indirect	Immediate

Two Operands Arithmetic, Logic and Load Instructions	
Destination	**Source**
Memory Indirect	Memory Direct

Two Operands Load Instructions	
Destination	**Source**
Register Direct	Register Direct
Register Direct	Register Indirect
Register Direct	Register Indexed
Register Direct	Memory Indirect
Register Direct	Memory Indexed
Register Direct	Memory Indirect with Post-Increment
Register Direct	Memory Indirect with Pre-Decrement
Register Direct	Memory Direct
Register Indirect	Register Direct
Register Indexed	Register Direct
Memory Indirect	Register Direct
Memory Indexed	Register Direct
Memory Indirect with Post-Increment	Register Direct
Memory Indirect with Pre-Decrement	Register Direct
Memory Direct	Register Direct
Register Direct	Immediate
Memory Direct	Immediate
Memory Indirect	Immediate
Long Indexed Memory (1)	Immediate

Two Operands Load Instructions (2)	
Destination	**Source**
Register Indirect with Post-Increment	Memory Indirect with Post-Increment
Memory Indirect with Post-Increment	Register Indirect with Post-Increment
Memory Indirect with Post-Increment	Memory Indirect with Post-Increment

Notes:
1 Word Instructions Only
2 Load Byte Only

2.2.1 Register Addressing Modes

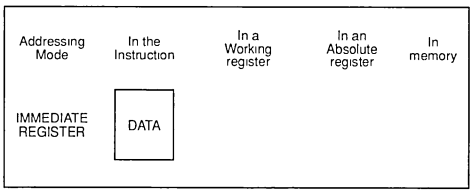
2.2.1.1 IMMEDIATE ADDRESSING MODE

In the Immediate addressing mode, the data is found in the instruction. When using immediate data, a hash-mark (#) is used to distinguish it from an absolute address in memory.

Example: `ldw RR42,#65536`
loads the immediate value 65536 into the register pair R42 & R43. While the example shows decimal data, hexadecimal and binary values may also be used.

Example: `ldw RR42,#0FFFFh.`

Figure 2-9. Immediate Register

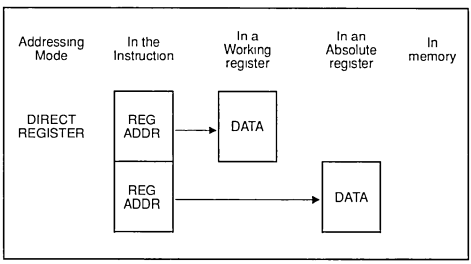


2.2.1.2 DIRECT ADDRESSING MODE

In the direct addressing mode, a register can be addressed by using its absolute address in the Register File (in decimal, hexadecimal or binary form). Alternatively a register can be addressed directly as a working register;

Example: `xch R0A2h,r4`
exchanges the values in the register 0A2h and working register number 4.

Figure 2-10. Direct Register



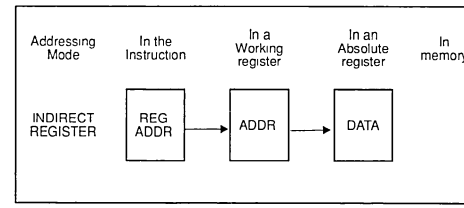
2.2.1.3 INDIRECT ADDRESSING MODE

In the Indirect Register Addressing mode, the address of the data does not appear in the instruction but is located in a working register. The address of this register is given in the instruction. The indirect addressing mode is indicated by the use of parentheses.

Example:
If register 200 contains 178 and working register 11 contains 86 then the instruction `ld (r11),R200` loads the value 178 into register 86.

Note: the indirect address can only be contained in a working register

Figure 2-11. Indirect Register



2.2.1.4 INDEXED ADDRESSING MODE

To address a register using the Indexed mode, an offset value is used to add to an index value (which acts as a base or starting value). The offset value is the Immediate value given in the instruction while the index value is given by the contents of the working register.

Example: if working register 10 contains 55 then the instruction

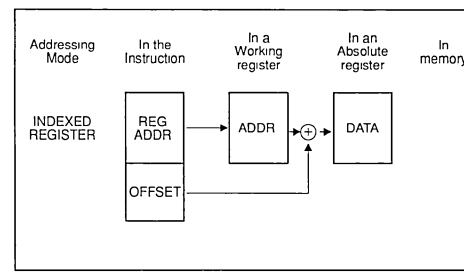
`ld 40(r10),r18`

loads register 95 (i.e.55+40) with the content of working register 18.

The Register File never needs an absolute value requiring more than one byte and therefore only requires a short offset and a single register to contain the index.

Note: The index value can only be contained in a working register

Figure 2-12. Indexed Register



2.2.1.5 INDIRECT REGISTER POST-INCREMENT ADDRESSING MODE

In this addressing mode, both destination and source addresses are given by the contents of working registers which are then post-incremented. The address of the memory location is contained in a working register pair, and the address of the register is contained into a single working register. Only working registers may be used to contain the addresses, this mode being indicated by both source and destination using parentheses followed by plus sign.

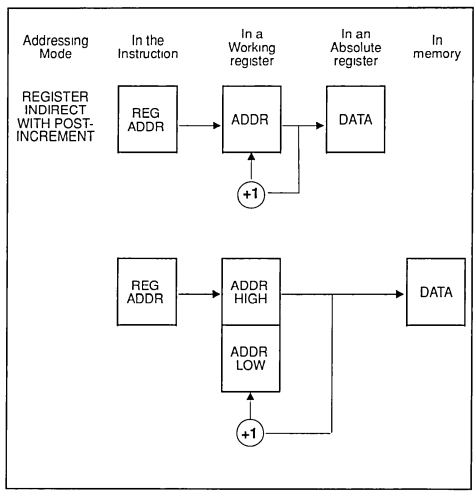
Example: if working register 8 contains the value 44, working register pair rr2 contains the value 2000, and register 44 contains the value 56, then by using the instruction

```
ld (rr2)+,(r8)+
```

the memory location 2000 will be loaded with the value 56. Immediately following this, the contents of r8 is incremented to 45 and the contents of rr2 is incremented to 2001.

This addressing mode is useful for moving blocks of data either from Register File to Memory or from Memory to Register File.

Figure 2-13. Register Indirect Post-Increment



2.2.1.6 DIRECT BIT ADDRESSING MODE

In the direct bit addressing mode, any bit in any working register can be addressed

Examples: `bset r7.3`

This instruction sets the bit 3 of the working register 7.

```
bld r7.3, r12.6
```

This instruction loads the bit 6 of the working register 12 in bit 3 of working register 7

2.2.2 Memory Addressing Modes

The memory addressing modes described in this section are available to data and program memory. Thus before addressing the memory, it is necessary to indicate by use of the Set Program/Data Memory instructions, `spm` and `sdm`, in which memory the instructions are working. Since each memory space is 64k byte long, a word address is necessary to specify memory locations.

2.2.2.1 DIRECT ADDRESSING MODE

The Memory Direct addressing mode requires the specific location within the memory. This only needs the absolute offset value which can be given in decimal, hex or binary form.

Thus the instruction

```
ld 12345,r9
```

loads working register 9 data into memory location 12345

In the memory direct mode, it is possible to use an immediate addressing mode for the source operand.

Examples:

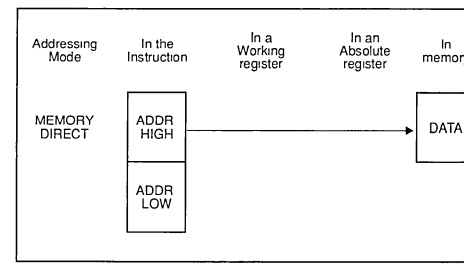
```
ld 12354,#34
```

will load the value 34 into the memory location 12354.

```
ldw 12354,#3457
```

will load the location pair 12354 and 12355 with the value 3457.

Figure 2-14. Memory Direct



2.2.2.2 INDIRECT ADDRESSING MODE

When using the indirect addressing mode to access memory, the address is contained in a pair of working registers.

Example: if the working register pair r8 and r9 contains the value 2000 then the instruction

```
ld (rr8),#34
```

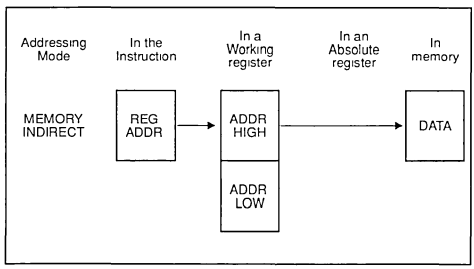
loads the value 34 into memory location 2000.

If the data to be stored is a word then the instruction ldw will automatically interpret the address as a pair of memory locations. So if rr8 contains 2000 then the instruction

```
ldw (rr8),#3467
```

loads the memory locations 2000 and 2001 with the value 3467.

Figure 2-15. Memory Indirect



2.2.2.3 INDIRECT WITH POST-INCREMENT ADRESSING MODE

The indirect with post-increment addressing mode is similar to the memory indirect addressing mode but, in addition, after accessing the data in the currently pointed address, the value in the pointing working register pair is incremented. This mode is indicated by a plus sign following a working register pair in parentheses, e.g. (rr4)+.

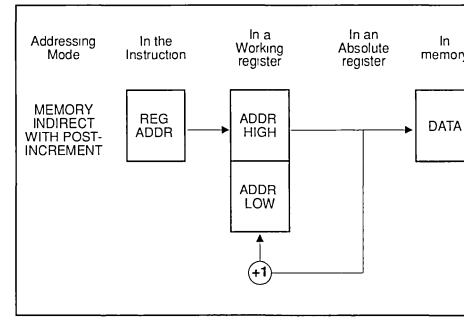
Example:
If the working register pair rr4 (working registers r4 and r5) contains the value 3000 and memory location 3000 contains the value 88, then the instruction

```
ld R50,(rr4)+
```

loads register 50 with the value 88 and then the value in rr4 to be incremented to 3001.

This mode uses only working registers to contain the address. Thus the Indirect with Post-Increment addressing mode is most useful in repeated situations when a number of adjacent items of data are required in succession. The use of this addressing mode saves both time and program memory space since it cuts the usual increment instruction.

Figure 2-16. Memory Indirect Post-Increment



2.2.2.4 INDIRECT WITH PRE-DECREMENT ADDRESSING MODE

This indirect memory addressing mode has an automatic pre-decrement. The address can only be contained in working registers and the mode is indicated by a minus sign in front of the working registers which are in parentheses, e.g. -(rr6).

Thus if the working register pair rr6 contains the value 1111 and location 1110 contains the value 40 then the instruction

```
ld R56,-(rr6)
```

decrements the value in rr6 to 1110 and then loads the value 40 into register 56.

This addressing mode allows the ST9 to deal in the reverse order with data previously managed using the indirect post-increment mode without resetting the pointing registers (of the last post-increment).

The pre-decrement mode has the same benefits of time and program memory saving as the post-increment mode.

Figure 2-17. Memory Indirect Pre-Decrement

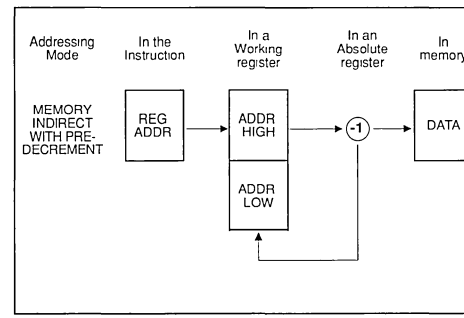
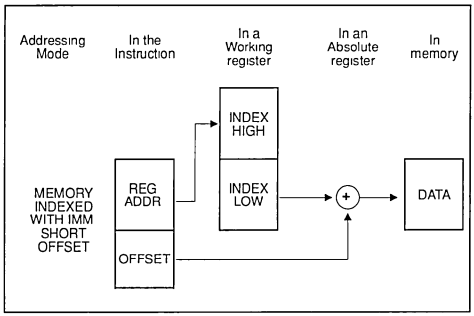


Figure 2-18. Memory Indexed with Immediate Short Offset



2.2.2.5 INDEXED ADDRESSING MODES

There are three indexed addressing modes, each using an indirect address plus offset format. The index address is given as an indirect address contained in a working register pair, while the offset can be long or short (a word or a byte). The address of the data required is given by the value of the working register pair indicated, (the index), plus the value of the given offset. The specification of this offset which differentiates the three modes, is as follows:

– Indexed with an Immediate Short and Long Offset

In these indexed modes the offset is a fixed and Immediate value included in the instruction. It may be either a short or long index as required, this immediate value being added to the address given by the working register pair.

Example: if the working register pair, rr6, contains the value 8000 and memory location 8034 contains the value 254 then the instruction

```
ld R55,34(rr6)
```

loads the value 254 into register 55.

Or, as another example, if the working register pair rr2 contains the value 2000 and register 78 contains the value 34 then the instruction.

```
ld 322(rr2),r78
```

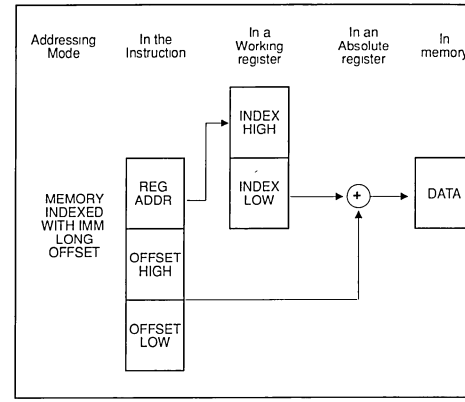
loaded the value 34 into memory location 2322.

– Indexed with a Register Offset

In this addressing mode, the index is supplied by one pair of working registers and the offset is supplied by a second pair of working registers. The format is rrx(rry), x and y being in the range 0,2,4...12,14.

Example
If working register pair rr0 contains the value 2222 and working register pair rr4 contains 3333 while

Figure 2-19. Memory Indexed with Immediate Long Offset

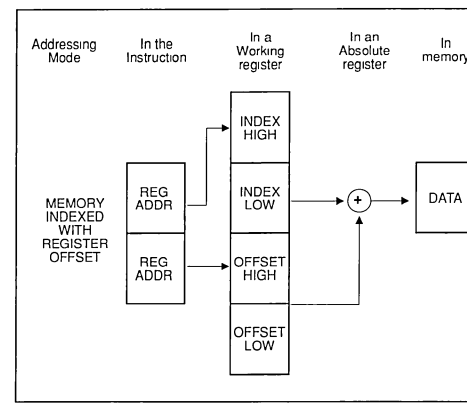


register 45 contains the value 78 then the instruction

```
ld rr4(rr0),R45
```

loads the value 78 into memory location 5555.

Figure 2-20. Memory Indexed with Register Offset



2.2.2.6 INDIRECT MEMORY BIT ADDRESSING MODE

In the indirect memory bit addressing mode, any bit of Program/Data memory location can be addressed with the `btset` (Bit Test and SET) instruction.

Example

```
btset (rr8).3
```

This instruction sets the bit 3 of the memory location addressed by the working registers r8, r9 content.

2.3 THE REGISTER FILE

The Register File consists of:

- 224 general purpose registers (00h to 0DFh)
- 16 system registers (0E0h to 0EFh)
- up to 64 I/O pages (0F0h to 0FFh), each containing up to 16 registers

This means that in the maximum expansion, the ST9 can have up to 1264 registers.

The Port registers are located in two areas:

Port registers 0-5 occupy the first 6 registers of the system register group and have the absolute addresses R224 to R229 (0E0h to 0E5h).

The Port registers for ports 6 to 7 are located in page 3, in registers 251 (0FBh) and 255 (0FFh) respectively.

2.3.1 System Registers

The sixteen system registers, including Port registers 0-5 (R224-R229), are in locations 224 to 239 (0E0h-0EFh see system registers map, figure 2.4.). In the following paragraphs an explanation is given for each system register and for its specific function.

2.3.2 Register Pointing Techniques

Two registers, R232 and R233, within the system register group, are available for register pointing. R232 and R233 may be used together as a single pointer for a 16 register working space or separately for two 8 register spaces, in which case R232 becomes Register Pointer 0 (RP0) and R233 becomes Register Pointer 1 (RP1).

Figure 2-21. Register File

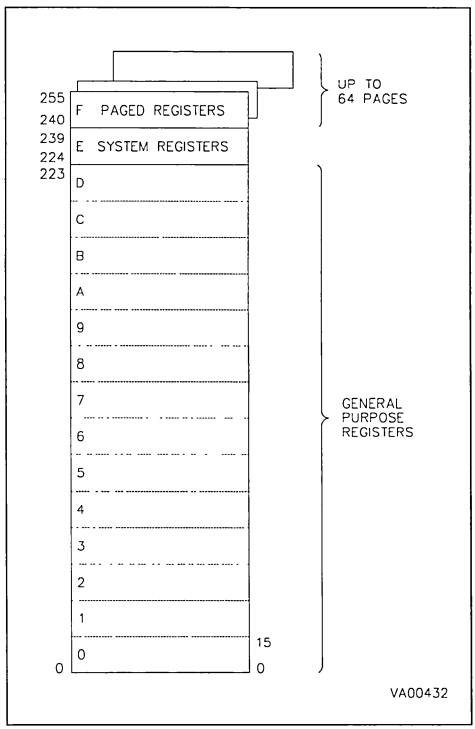


Table 2-2. Register File Organization

Register File Group	Hex. Address	Decimal Address	General Function
Group F	0F0-0FF	240-255	Page Registers
Group E	0E0-0EF	224-239	System Registers
Group D	0D0-0DF	208-223	G.P. Registers
Group C	0C0-0CF	192-207	G.P. Registers
Group B	0B0-0BF	176-191	G.P. Registers
Group A	0A0-0AF	160-175	G.P. Registers
Group 9	090-09F	144-159	G.P. Registers
Group 8	080-08F	128-143	G.P. Registers
Group 7	070-07F	112-127	G.P. Registers
Group 6	060-06F	96-111	G.P. Registers
Group 5	050-05F	80-95	G.P. Registers
Group 4	040-04F	64-79	G.P. Registers
Group 3	030-03F	48-63	G.P. Registers
Group 2	020-02F	32-47	G.P. Registers
Group 1	010-01F	16-31	G.P. Registers
Group 0	000-00F	00-15	G.P. Registers

The instructions SRP, SRP0 and SRP1 (the Set Register Pointer instructions) automatically inform the ST9 whether the Register File is to operate with a single 16-register group or two 8-register groups. The SRP0 and SRP1 instructions automatically set the twin 8-register group mode while the SRP instruction sets the single 16-register group mode. There is no limitation on the order or positions of these chosen register groups other than they must be on 8 or 16 register boundaries.

The addressing of working registers involves use of the Register Pointer value plus an offset value given by the number of the addressed working register.

When addressing a register, the most significant nibble (bits 4-7) gives the group address and the least significant nibble (bits 0-3) gives the register within that group.

2.3.2.1 REGISTER POINTER 0

RP0-R232 (OE8h) System Read/Write
Register Pointer 0
Reset Value : undefined

7							0
RG7	RG6	RG5	RG4	RG3	RPS	D1	D0

b7-b3 = **RG7-RG3:** *Register Group number.* These bits contain the number (from 0 to 31) of the group of working registers indicated in the instructions srp0 or srp. When using a 16-register group, a number between 0 and 31 must be used in the srp instruction indicating one of the two adjacent 8-register group of working registers used. RG7 is the MSB.

b2 = **RPS:** *Register Pointer Selector.* This bit is set by the instructions srp0 and srp1 to indicate that a double register pointing mode is used. Otherwise, the instruction srp resets the RPS bit to zero to indicate that a single register pointing mode is used.

b1,b0 = **D1,D0:** These bits are fixed by hardware to zero and are not affected by any writing instruction trying to modify their value.

2.3.2.2 REGISTER POINTER 1

RP1-R233 (OE9h) System Read/Write
Register Pointer 1
Reset Value : undefined

7							0
RG7	RG6	RG5	RG4	RG3	RPS	D1	D0

This register is used only with double register pointing mode; otherwise, using single register pointing mode, the RP1R register has to be considered as reserved and not usable as a general purpose register.

b7-b3 = **RG7-RG3:** *Register Group number.* These bits contain the number (from 0 to 31) of the group of 8 working registers indicated in the instructions srp1. Bit 7 is the MSB.

b2 = **RPS:** *Register Pointer Selector.* This bit is automatically set by the instructions srp0 and srp1 to indicate that a double register pointing mode is used. Otherwise the instruction srp reset the RPS bit to zero to indicate that a single register pointing mode is used.

b1,b0 = **D1,D0:** These bits are hardware fixed to zero and are not affected by any writing instruction trying to modify their value.

2.3.2.3 EXAMPLES

Using the Single 16 Register Group

When the system is operating in the single 16-register group mode, the registers are referred to as r0-r15. In this mode, the offset value (i.e. the number of the working register referred to) is supplied in the address (preceded by a small r, e.g. r5) and is added to the Register Pointer 0 value to give the absolute address.

For example, if the Register Pointer contains the value 70h, then working register r7 would have the absolute address, R77h.

In this mode, the single 16-registers group will always start from the lowest even number equal or lower to the number given in the instruction.

Example: srp #3 is equivalent to srp #2.

Using the Twin 8-Register Group

When working in the twin working group mode, the registers pointed by Register Pointer 0 (RP0R), are referred as r0-r7 and those pointed by Register Pointer 1 (RP1R), are referred to as r8-r15, regardless of their absolute addresses. In this mode, when operating with the first 8 working registers (i.e. r0 - r7) the working register number acts as an offset which is added to the value in Register Pointer 0.

So if Register Pointer 0 contains the value 96, then working register 0 has the absolute address 96, working register 5 has the absolute address 101, and so on. The second group of working registers, r8-r15, has the offset values 0 to 7 repectively (i.e. r8 has the offset value 0, r9 has the offset value 1, and so on), this offset value being added to the value in Register Pointer 1.

Figure 2-22a. Single 16 Register Pointing Mode

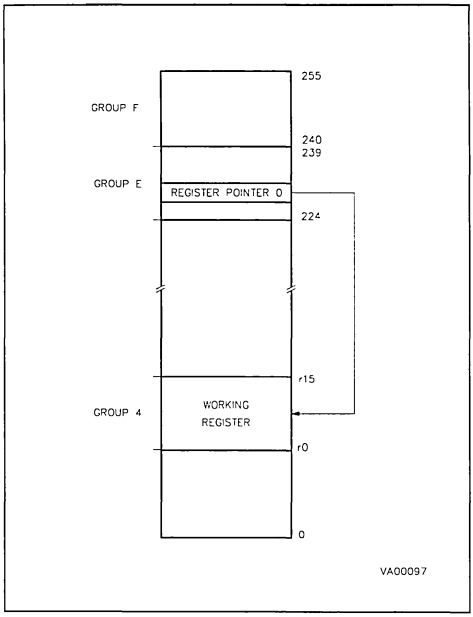
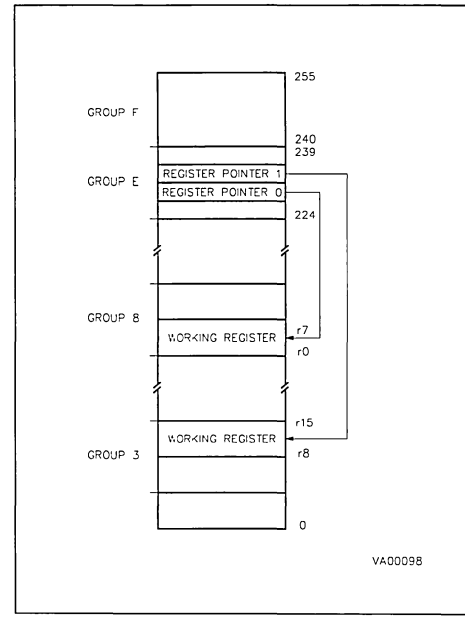


Figure 2-22b. Double Register Pointing Mode



For example, given that the value in Register Pointer 1 is 32, then working register 12 supplies an offset value of 4 (given by 12 minus 8) to the value in Register Pointer 1 to give an absolute address of 36.

Note: If working in twin 8-register group mode but only using SRP0 (i.e. only using one 8-register group) the unused register (R233) is to be considered as reserved and not usable as a general purpose register.

The group of registers immediately below the system registers (i.e. group R208-R223) can only be accessed via the Register Pointers. To address group D then, it is necessary to set the Register Pointer to group D and then use the addressing procedure for working registers. The programmer is required to remember that the D-group (0D0h-0DFh, R208-R223) should be used as a stacking area. This point is also covered in the Stack Pointers paragraph (2.3.4.3).

2.3.3 Page Configuration

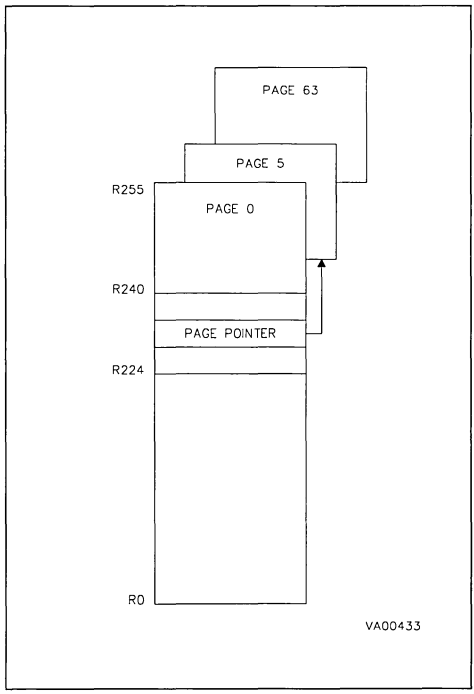
The pages are available to be used for the storage of control information (such as interrupt vector pointers) relevant to particular peripherals. There are up to 64 pages (each with 16 registers) based on registers 240-255. These paged registers are addressable via the page pointer register (PPR), which is system register R234.

To address a paged register the page pointer register (R234) must be loaded with the relevant page number using the `spp` instruction (Set Page Pointer) and subsequently any address from the top (F) group (R240-R255) will be referred to that page.

For example if register 23 contains the value 44, the following sequence loads the third register R242 on page 5 with the value 44.

```
spp 5
ld R242, R23
```

Figure 2-23. Page Pointer Configuration



2.3.3.1 PAGE POINTER REGISTER

PPR-R234 (OEAh) System Read/Write
Page Pointer Register
Reset value : undefined

7							0
PP7	PP6	PP5	PP4	PP3	PP2	PP1	PP0

b7-b2 = **PP7-PP2:** *Page Pointer.* These bits contain the number (between 0 to 63) of the page chosen by the instruction ssp (Set Page Pointer). PP7 is the MSB of the page address. Once the page pointer has been set, there is no need to refresh it unless a different page is required.

b1-b0 = **D1,D0:** These bits are fixed by hardware to zero and are not affected by any writing instruction trying to modify their value.

2.3.3.2 PAGE 0 CONFIGURATION

This page contains the control registers of:

– the external interrupt
– the watchdog timer
– the wait logic states
– the serial peripheral interface (SPI)
– the EPROM (when present, otherwise this register is reserved and cannot be used (as well as register 255.)
– the EEPROM (when present)

Please refer to the Interrupt, the EPROM and EEPROM, the Timer/Watchdog, the SPI, and the Clock chapters for more detailed information concerning these registers.

Table 2-3. Page 0 Configuration

Register Number		Page 0 Register
0	Reserved	
1	EEPROM	EEPROM control register
2	EITR	External Interrupt Trigger event register
3	EIPR	External Interrupt Pending Register
4	EIMR	External Interrupt bit Mask Register
5	EIPLR	External Interrupt Priority Level Register
6	EIVR	External Interrupt Vector Register
7	NICR	Nested Interrupt Control Register
8	WDTHR	Watchdog/Timer-High Register
9	WDTLR	Watchdog/Timer-Low Register
A	WDTPR	Watchdog/Timer Prescaler Register
B	WDTCR	Watchdog/Timer Control Register
C	WCR	Wait Control Register
D	SPIDR	SPI Data Register
E	SPICR	SPI Control Register
F	Reserved	

2.3.4 Stack Pointers

There are two separate, double register stack pointers available (named System Stack Pointer and User Stack Pointer), both of which can address registers or memory.

The stack pointers point to the bottom of the stacks which are filled using the `push` commands and emptied using the `pop` commands. The stack pointer is automatically pre-decremented when data is "pushed in" and post-incremented when data is "popped out".

For example, the register address space is selected for a stack and the corresponding stack pointer register contains 220. When a byte of data is "pushed" into the stack, the stack pointer register contents is decremented to 219, then the data byte is "loaded" into register 219. Conversely, if a stack pointer register contains 189 and a byte of data is "popped" out, the byte of data is then extracted from the stack and then the stack pointer register is incremented to 190.

The `push` and `pop` commands used to manage the system stack area are made applicable to the user stack by adding the suffix U, while to use a stack instruction for a word a W is added.

For example `push` inserts data into the system stack, but an added U indicates the user stack and W means a word, so the instruction `pushuw` loads a word into the bottom of the user stack.

If the User Stack Pointer register contains 223 (working in register space) the instruction `pushuw` will decrement User Stack Pointer register to 222 and then load a word into register R222 and R221.

When bytes (or words) are "popped out" the values in those registers are left unchanged until fresh data is loaded into those locations. Thus when data is "popped" out from a stack area, the stack content remains unchanged.

NOTE: Stacks should not be located in the pages or the system register area, because of the risk of losing valuable data

Figure 2-24a. System And/Or User Stack in Register Stack Mode

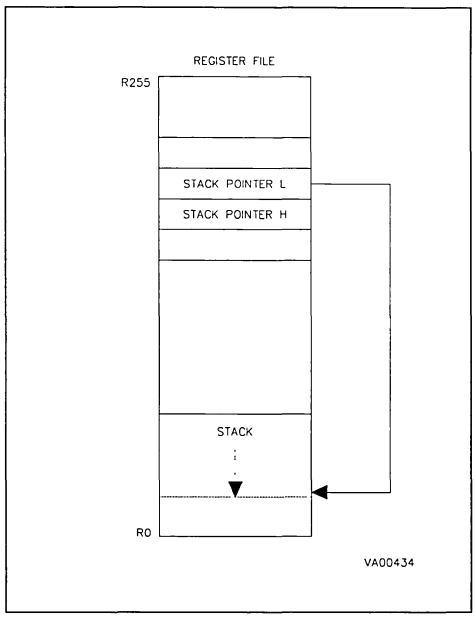
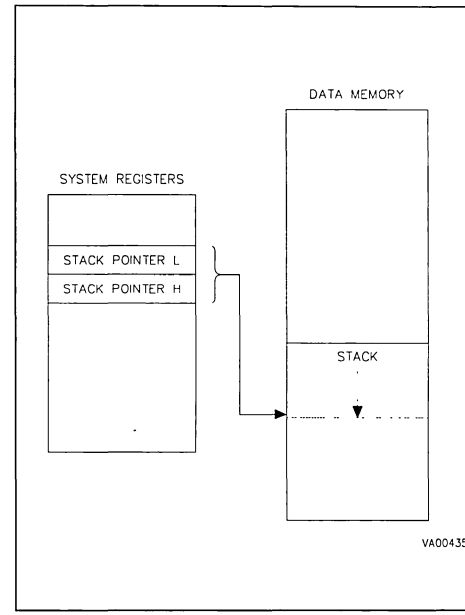


Figure 2-24b. System And/Or User Stack in Memory Stack Mode



2.3.4.1 THE SYSTEM STACK AREA AND THE SYSTEM STACK POINTER

The System Stack area is used for the storage of temporarily suspended system and/or control registers, i.e. the Flag register and the Program counter, while interrupts are being serviced. For subroutine execution only the Program Counter needs to be saved in the System stack area.

There are two situations when this occurs automatically, one being when an interrupt occurs and the other when the instruction call subroutine is used. When the system stack area is in the Register File, the stack pointer, which points to the bottom of the stack, only needs one byte for addressing, in which case the System Stack Pointer Low Register (R239) is sufficient for addressing purposes. As a result the System Stack Pointer High Register (R238) becomes redundant BUT must be considered as reserved. Clearly when the stack is external a full word address is necessary and so both registers are used to point, the even register providing the MSB and the odd register providing the LSB.

2.3.4.2 THE USER STACK AREA AND USER STACK POINTER

The User Stack area is completely free from all interference from automatic operations and so it provides a totally user controlled stacking area, that area being in any part of the memory which is of a RAM nature, or the first 14 groups of the general Register File i.e. not in the system register or page groups.

The User Stack Pointer consists of two registers, R236 and R237, which are both used for addressing an external stack, while, when stacking in the Register File, the User Stack Pointer High Register, R236, becomes redundant but must be considered as reserved.

2.3.4.3 STACK LOCATION

Care is necessary when managing stacks as there is no limit to stack sizes apart from the bottom of any address space in which the stack is placed. Consequently programmers are advised to use a stack pointer value as high as possible, particularly when using the Register File as a stacking area. This will also benefit programmers who may locate the stacks in group D using, for example the instruction `ld R237, #223` which loads the value 223 into the User Stack Pointer Low Register. The Programmer will not need to remember to set the Register Pointer to 208 to gain access to registers in the D-group, a problem outlined in paragraph 2.3.2.3

Stacks may be located anywhere in the first 14 groups of the Register File (internal stacks) or the data memory (external stacks). It is not necessary to set the data memory using the instruction `sdm` as external stack instructions automatically use the data memory.

2.3.5 Mode Register

This register MODER is located in the System Register Group at the address 235. Using this register it is possible:

- to select either internal or external System and User Stack area,
- to manage the clock frequency
- to enable the Bus request and Wait signals when interfacing external memory.

MODER-R235 (OE3h) Sys. Reg. Read/Write
Mode Register
Reset value : 1110 0000

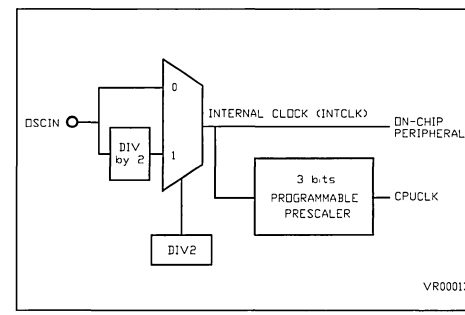
7							0
SSP	USP	DIV2	PRS2	PRS1	PRS0	BRQEN	HIMP

b7 = **SSP:** *System Stack Pointer.* This bit selects internal (in the Register File) or external (in the external Data Memory) System Stack area, logical "1" for internal, and logical "0" for external. After Reset the value of this bit is "1".

b6 = **USP:** *User Stack Pointer.* Same as bit 7 for the User Stack Pointer;

b5 = **DIV2:** *OSCIN Clock Divided by 2.* This bit controls the divide by 2 circuit which operates on the OSCIN Clock. A logical "1" value means that the OSCIN clock is internally divided by 2, and a logical "0" value means that no division of the OSCIN Clock occurs.

Figure 2-25 . Clocks Generation



b4-b2 = **PRS2-PRS0:** *ST9 CPUCLK Prescaler.* These bits load the prescaling module of the internal clock (INTCLK). The prescaling value selects the frequency of the ST9 clock, which can be divided by 1 to 8. See Clock chapter for more information.

b1 = **BRQEN:** *Bus Request Enable.* This bit is a software enable of an External Bus Request. When set to "1", it enables a Bus Request on the BUSREQ pin.

b0 = **HIMP:** *High Impedance Enable.* When Port 0 and/or Port 1 are programmed as multiplexed address and Data lines to interface external Program and/or Data Memory, these lines can be forced into the High Impedance state by setting to 1 the HIMP bit. When this bit is reset, it has no effect on P0 and P1 lines.

If Port 1 is declared as an address AND as an I/O port (example: P10 ... P14 = Address, and P15 ... P17 = I/O), HIMP has no effect on the I/O lines (in the previous example: P15 ... P17).

2.3.6 Flag Register

The Flag Register, R231 (0E7h), contains 8 flags indicating the status of the ST9. During an interrupt the flag register is automatically stored in the system stack area and recalled at the end of the interrupt service routine so that the ST9 is returned to the original status. This occurs for all interrupts and, when operating in the nested mode, up to seven versions of the flag register may be stored.

FLAGR-R231 (OE7h) Sys. Reg. Read/Write
Flag Register
Reset value : undefined

7							0
C	Z	S	V	DA	H	UF	DP

b7 = **C:** *Carry Flag.* The carry flag C is affected by the following instructions: Addition (`add, addw, adc, adcw`), Subtraction (`sub, subw, sbc, sbcw`), Compare (`cp, cpw`), Shift Right Arithmetic (`sra, sraw`), Rotate (`rrc, rrcw, rlc, rlcw, ror, rol`), Decimal Adjust (`da`), and Multiply and Divide (`mul, div, divws`) instructions. When set, it generally indicates a carry out of the most significant bit position of the register being used as an accumulator (bit 7 for byte and bit 15 for word operations).

The carry flag can be set by the Set Carry Flag (scf) instruction, cleared by the Reset Carry Flag (rcf) instruction, and complemented (changed to "0" if "1", and vice versa) by the Complement Carry Flag (ccf) instruction.

b6 = **Z:** *Zero Flag.* The Zero flag is affected by the same instructions as the Carry flag, plus the Logical (`and, andw, or, orw, xor, xorw, cpl`), Increment and Decrement (`inc, incw, dec, decw`), Test (`tm, tmw, tcm, tcmw, btset`). In most cases, the Zero flag is set when the register being used as an accumulator register, following one of the above operations, is zero.

b5 = **S:** *Sign Flag.* The Sign flag is affected by the same instructions as the Zero flag. The Sign flag is set when bit 7 (for byte operation) or bit 15 (for word operation) of the register used as an accumulattor is one.

b4 = **V:** *Overflow Flag.* The Overflow flag is affected by the same instructions as the Zero and Sign flags. When set, the Overflow flag indicates that a two's-complement number, in a result register, is in error, since it has exceeded the largest (or is less than the smallest), number that can be represented in twos-complement notation.

b3 = **DA:** *Decimal Adjust Flag.* The Decimal Adjust flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag is used to specify which type of instruction was executed last, so that the subsequent Decimal Adjust (`da`) operation can perform its function correctly. The Decimal Adjust flag cannot normally be used as a test condition by the programmer.

b2 = **H:** *Half Carry Flag.* The Half Carry flag indicates a carry out of (or a borrow into) bit 3, as the result of adding or subtracting two 8-bit bytes, each representing two BCD digits. The Half Carry flag is used by the Decimal Adjust (`da`) instruction to convert the binary result of a previous addition or subtraction into the correct BCD result. Like the Decimal Adjust flag, this flag is not normally accessed by the user.

b1 = **UF:** *User Flag.* Bit 1 in the flag register (UF) is available to the user, but it must be set or cleared by an instruction.

b0 = **DP:** *Data/Program Memory Flag.* This bit in the flag register indicates which memory area is addressed. Its value is affected by the Set Data Memory (`sdm`) and Set Program Memory (`spm`) instructions.

If the bit is set, the ST9 addresses the Data Memory Area; when the bit is cleared, the ST9 addresses the Program Memory Area. By reading this bit, the user can verify in which memory area the processor is working. The user writes this bit with the `sdm` or `spm` instructions.

2.3.7 Central Interrupt Control Register

This Register CICR is located in the system Register Group at the address R230 (0E6h). Please refer to "INTERRUPT and DMA" chapter in order to get the background of the ST9 interrupt philosophy.

CICR-R230 (OE6h) Sys. Reg. Read/Write
Central Interrupt Control Register
Reset Value : 1000 0111

b7 = **GCEN:** *Global Counter Enable.* This bit is the Global Counter Enable of the 2 x 16 bit Timers of the Timer cell. The GCEN bit is ANDed with the CE (Counter Enable) bit of the Timer Control Register (explained in the Timer chapter) in order to enable the Timers when both bits are set. This bit is set

7							0
GCEN	TLIP	TLI	IEN	IAM	CPL2	CPL1	CPL0

after the Reset cycle.

b6 = **TLIP:** *Top Level Interrupt Pending.* This bit is automatically set when a Top Level Interrupt Request is recognized. This bit can also be set by Software in order to simulate a Top Level Interrupt Request.

b5 = **TLI:** *Top Level Interrrupt bit.* When this bit is set, a Top Level interrupt request is acknowledged depending on the IEN bit and the TLNM bit (in Nested Interrupt Control Register). If the TLM bit is reset the top level interrupt acknowledgement depends on the TLNM alone.

b4 = **IEN:** *Enable Interrupt.* This bit, (when set), allows interrupts to be accepted. When reset no interrupts other than the NMI can be acknowledged. It is cleared by interrupt acknowledgement for concurrent mode and set by interrupt return (IRET). It can be managed by hardware and software (`ei` and `di` instruction).

b3 = **IAM:** *Interrupt Arbitration Mode.* This bit covers the selection of the two arbitration modes, the Concurrent Mode being indicated by the value "0" and the Fully Automatic Nested Mode by the value "1". This bit is under software control.

b2-b0 = **CPL2-CPL0:** *Current Priority Level.* These three bits record the priority level of the interrupt presently under service (i.e. the Current Priority Level, CPL). For these priority levels 000 is the highest priority and 111 is the lowest priority. The CPL bits can be set by hardware or software and give the reference by which following interrupts are either left pending or able to interrupt the current interrupt. When the present interrupt is replaced by one of a greater priority, the current priority value is automatically stored until required.

2.3.8 Input/Output Ports

The input/output ports are located in two areas. The port registers for ports 0-5 are located at the bottom of the system register group in locations R224 to R229 (0E0h - 0E5h), while port 6 and 7 are located in page three, in registers 251 (0FBh) and 255 (0FFh) respectively.

Each Port has three associated Control registers, which determine the individual pin modes (I/O, Open-Drain etc). These registers are located in pages 2 and 3 (see the I/O Ports chapter 9 for detailed information).

CHAPTER 3

INSTRUCTION SET

3.1 THE INSTRUCTION SET

The ST9 instruction set consists of 87 instruction types which can be divided into eight groups:

- Load (two operands)
- Arithmetic & logic (two operands)
- Arithmetic Logic and Shift (one operand)
- Stack (one operand)
- Multiply & Divide (two operands)
- Boolean (one or two operands)
- Program Control (zero to three operands)
- Miscellaneous (zero to two operands)

The wide range of instructions eases use of the register file and address spaces, reducing operation times, while the register pointers mechanism allows an unmatched code efficiency and ultrafast context switching. A particularly notable feature is the comprehensive "Any Bit, Any Register" (ABAR) addressing capability of the Boolean instructions.

The ST9 can operate with a wide range of data lengths from single bits, 4-bit nibbles which can be in the form of Binary Coded Decimal (BCD) digits, 8-bit bytes, and 16-bit words.

The following summary shows the instructions belonging to each group and the number of operands required for each instruction. The source operand is "src", "dst" is the destination operand, and "cc" is a condition code.

LOAD INSTRUCTIONS (two operands)

Mnemonic	Operands	Instruction
LD LDW	dst,src dst,src	Load Load Word
LDPP LDPD LDDP LDDD	dst,src dst,src dst,src dst,src	Load Program Memory -> Program Memory Load Data Memory -> Program Memory Load Program Memory -> Data Memory Load Data Memory -> Data Memory

ARITHMETIC & LOGIC (two operands)

Mnemonic	Operands	Instruction
ADD ADDW	dst,src dst,src	Add Add Word
ADC ADCW	dst,src dst,src	Add With carry Add Word With Carry
SUB SUBW	dst,src dst,src	Substract Substract Word
SBC SBCW	dst,src dst,src	Substract With Carry Substract Word With Carry
AND ANDW	dst,src dst,src	Logical AND Logical Word AND
OR ORW	dst,src dst,src	Logical OR Logical Word OR
XOR XORW	dst,src dst,src	Logical Exclusive OR Logical Word Exclusive OR
CP CPW	dst,src dst,src	Compare Compare Word
TM TMW	dst,src dst,src	Test Under Mask Test Word Under Mask
TCM TCMW	dst,src dst,src	Test Complement Under Mask Test Word Complement Under Mask

ARITHMETIC LOGIC & SHIFT (one operand)

Mnemonic	Operands	Instruction
INC INCW	dst dst	Increment Increment Word
DEC DECW	dst dst	Decrement Decrement Word
SLA SLAW	dst dst	Shift Left Arithmetic Shift Word Left Arithmetic
SRA SRAW	dst dst	Shift Right Arithmetic Shift Word Right Arithmetic
RRC RRCW	dst dst	Rotate Right Through Carry Rotate Word Right Through Carry
RLC RLCW	dst dst	Rotate Left Through Carry Rotate Word Left Through Carry
ROR	dst	Rotate Right
ROL	dst	Rotate Left
CLR	dst	Clear
CPL	dst	Complement
SWAP	dst	Swap Nibbles
DA	dst	Decimal Adjust

STACK INSTRUCTIONS (one operand)

Mnemonic	Operands	Instruction
PUSH PUSHW PEA	src src src	Push on System Stack Push Word on System Stack Push Effective Address on System Stack
POP POPW	dst dst	Pop From System Stack Pop Word from System Stack
PUSHU PUSHUW PEAU	src src src	Push on User Stack Push Word on User Stack Push Effective Address on User Stack
POPU POPUW	dst dst	Pop From User Stack Pop Word From User Stack

MULTIPLY AND DIVIDE INSTRUCTIONS (two operands)

Mnemonic	Operands	Instruction
MUL	dst,src	Multiply 8x8
DIV DIVWS	dst,src dst,src	Divide 16/8 Divide Word Stepped 32/16

BOOLEAN INSTRUCTIONS (one and two operands)

Mnemonic	Operands	Instruction
BSET	dst	Bit Set
BRES	dst	Bit Reset
BCPL	dst	Bit Complement
BTSET	dst	Bit Test and Set
BLD	dst,src	Bit Load
BAND	dst,src	Bit AND
BOR	dst,srrc	Bit OR
BXOR	dst,src	Bit XOR

PROGRAM CONTROL INSTRUCTIONS (one, two or three operands)

Mnemonic	Operands	Instruction
RET		Return from Subroutine
IRET		Return from Interrupt
WFI		Stop Program Execution and Wait for the next Enabled Interrupt. If a DMA request is present, the CPU executes the DMA service routine and then automatically returns to the WFI
HALT		Stop Program Execution Until Next System Reset
JR	cc,dst	Jump Relative If Condition is Met
JP	cc,dst	Jump if Condition is Met
JP	dst	Unconditional Jump
CALL	dst	Unconditional Call
BTJF	dst,N	Bit Test and Jump if false
BTJT	dst,N	Bit Test and Jump if True
DJNZ	dst,N	Decrement a Working Register and Jump if Non Zero
DWJNZ	dst,N	Decrement a Register Pair and Jump if Non Zero
CPJFI	dst,N	Compare and Jump on False. Otherwise Post Increment
CPJTI	dst,N	Compare and Jump on True. Otherwise Post Increment

MISCELLANEOUS (none, one or two operands)

Mnemonic	Operands	Instruction
XCH	dst,src	Exchange Registers
SRP	src	Set Register Pointer Long (16 working registers)
SRP0	src	Set Register Pointer 0 (8 LSB working register)
SRP1	src	Set Register Pointer 1 (8 MSB working register)
SPP	src	Set Page Pointer
EXT	dst	Sign Extend
EI		Enable Interrupts
DI		Disable Interrupts
SCF		Set Carry Flag
RCF		Reset Carry Flag
CCF		Complement Carry Flag
SPM		Select Program Memory
SDM		Select Data Memory
NOP		No Operation

3.2 ST9 PROCESSOR FLAGS

An important feature of a single chip microcomputer is the ability to test data and make the appropriate action based on the results. In order to provide this facility, FLAGR (register 231) in the register file is used as a flag register. Six bits of this register are used as the following flags:

C - Carry

Z - Zero

S - Sign

V - Overflow

D - Decimal Adjust

H - Half Carry

Bit 1 is available to the user. Bit 0 is the Program/Data Memory selector bit.

The flags and their positions in the FLAGR are shown below.

FLAGR - R231 (0E7h) Sys. Reg. ;Read/Write

Flag Register

Reset value: Undefined

7							0
C	Z	S	V	D	H	UF	DP

Note : When making a logical and arithmetic operation on the flag register, the result is undefined For example, performing a CLR instruction on the flag register will return an undefined value on the Z (status bit 6) flag In fact, the clear operation would force to 0 all the register bits, while flag definition would require bit 6 to be set to 1.

b7 = **C:** *Carry Flag.* The carry flag C is affected by the following instructions: Addition (`add, addw, adc, adcw`), Subtraction (`sub, subw, sbc, sbcw`), Compare (`cp, cpw`), Shift Right Arithmetic (`sra, sraw`), Rotate (`rrc, rrcw, rlc, rlcw, ror, rol`), Decimal Adjust (`da`), and Multiply and Divide (`mul, div, divws`) instructions. When set, it generally indicates a carry out of the most significant bit position of the register being used as an accumulator (bit 7 for byte and bit 15 for word operations).

The carry flag can be set by the Set Carry Flag (scf) instruction, cleared by the Reset Carry Flag (rcf) instruction, and complemented (changed to "0" if "1", and vice versa) by the Complement Carry Flag (ccf) instruction.

b6 = **Z:** *Zero Flag.* The Zero flag is affected by the same instructions as the Carry flag, plus the Logical (`and, andw, or, orw, xor, xorw, cpl`), Increment and Decrement (`inc, incw, dec, decw`), Test (`tm, tmw, tcm, tcmw, btset`). In most cases, the Zero flag is set when the register being used as an accumulator register, following one of the above operations, is zero.

b5 = **S:** *Sign Flag.* The Sign flag is affected by the same instructions as the Zero flag. The Sign flag is set when bit 7 (for byte operation) or bit 15 (for word operation) of the register used as an accumulattor is one.

b4 = **V:** *Overflow Flag.* The Overflow flag is affected by the same instructions as the Zero and Sign flags. When set, the Overflow flag indicates that a two's-complement number, in a result register, is in error, since it has exceeded the largest (or is less than the smallest), number that can be represented in twos-complement notation.

b3 = **DA:** *Decimal Adjust Flag.* The Decimal Adjust flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag is used to specify which type of instruction was executed last, so that the subsequent Decimal Adjust (`da`) operation can perform its function correctly. The Decimal Adjust flag cannot normally be used as a test condition by the programmer.

b2 = **H:** *Half Carry Flag.* The Half Carry flag indicates a carry out of (or a borrow into) bit 3, as the result of adding or subtracting two 8-bit bytes, each representing two BCD digits. The Half Carry flag is used by the Decimal Adjust (`da`) instruction to convert the binary result of a previous addition or subtraction into the correct BCD result. Like the Decimal Adjust flag, this flag is not normally accessed by the user.

b1 = **UF:** *User Flag.* Bit 1 in the flag register (UF) is available to the user, but it must be set or cleared by an instruction.

b0 = **DP:** *Data/Program Memory Flag.* This bit in the flag register indicates which memory area is addressed. Its value is affected by the Set Data Memory (`sdm`) and Set Program Memory (`spm`) instructions.

If the bit is set, the ST9 addresses the Data Memory Area; when the bit is cleared, the ST9 addresses the Program Memory Area. By reading this bit, the user can verify in which memory area the processor is working. The user writes this bit with the `sdm` or `spm` instructions.

3.3 CONDITION CODES

Flags C, Z, S, and OV control the operation of the "conditional" Jump instructions. The next table shows the condition codes and the flag settings.

Note : Some of the Status flags are used to indicate more than one condition e.g . Zero and Equal. In such cases the condition code is the same for both conditions.

Table 1. Condition Codes Table

MNEMONIC CODE	MEANING	FLAG SETTING	HEX. VALUE	BINARY VALUE
F	Always False	____	0	0000
T	Always true	____	8	1000
C	Carry	C=1	7	0111
NC	No carry	C=0	F	1111
Z	Zero	Z=1	6	0011
NZ	No Zero	Z=0	E	1110
PL	Plus	S=0	D	1101
MI	Minus	S=1	5	0101
OV	Overflow	V=1	4	0100
NOV	No Overflow	V=0	C	1100
EQ	Equal	Z=1	6	0110
NE	Not Equal	Z=0	E	1110
GE	Greater Than or Equal	(S xor V)=0	9	1001
LT	Less Than	(S xor V)=1	1	0001
GT	Greater Than	(Z or(S xor V))=0	A	1010
LE	Less Than or Equal	(Z or(S xor V))=1	2	0010
UG	Unsigned Greater Than or Equal	C=0	F	1111
UL	Unsigned Less Than	C=1	7	0111
UGT	Unsigned Greater Than	(C=0 and Z=0)=1	B	1011
ULE	Unsigned Less Than or Equal	(C or Z)=1	3	0011

3.4 NOTATION

Operands and status flags are represented by a notational shorthand in the detailed instruction description (see programming manual). The notation for operands (condition codes and address modes) and the actual operands they represent are as follows:

Table 2. Notations (Part 1)

Notation	Significance	Actual Operand/Range
cc	Condition Code	See previous table 1
#N #NN	Immediate Byte Immediate Word	# data where data is a byte expression # data where data is a word expression
r	Direct Working Register	rn, where n=0-15
R	Direct Register	Rn, where n=0-255
rr	Direct Working Register Pair	rrn, where n is an even number in the range 0-15. (n=0,2,4,6....14)
RR	Direct Register Pair	RRn, where n is an even number in the range 0-254. (n=0,2,4,6....254)
(r)	Indirect Working Register	(rn), where n=0-15
(R)	Indirect register	(Rn), where n=0-255
(r)+	Indirect working register post increment	(rn)+, where n=0-15
N(rx)	Indexed register	N(rx), where x=0-15; N=0-255 (one byte)
N	Memory relative Short Address	Program label or expression in the range +127/-128 starting from the address of the next instruction
NN	Direct Memory Long Address	Program label or expression in the range 0-65535 in memory area

Table 2. Notations (Part 2)

Notation	Significance	Actual Operand/Range	
(rr)	Indirect Pair of Working Register Pointers	(rrn)	Where n is an even number in the range 0-15.(n=0,2,4,6....14)
(rr)+	Indirect Pair of Working Register Pointers with Post Increment	(rrn)+	where n is an even number in the range 0-15.(n=0,2,4,6....14)
-(rr)	Indirect Pair of Working Register Pointers with Pre Decrement	-(rrn)	where n is an even number in the range 0-15.(n=0,2,4,6....14)
N(rrx)	Indexed Pair of Working Register Pointers with Short Offset	N(rrx)	where x is an even number in the range 0-15.(n=0,2,4,6....14) and N is a signed one byte expression between +127/-128
NN(rrx)	Indexed Pair of Working Register Pointers with Long Offset	NN(rrx)	where x is an even number in the range 0-15.(n=0,2,4,6....14) and NN is word expression in the range between 0 and 65535
N(RRx)	Indexed Pair of Register Pointers with Short Offset	N(RRx)	where x is an even number in the range 0-255.(n=0,2,4,6...254) and N is a one byte signed expression in the range +127/-128
NN(RRx)	Indexed Pair of Register Pointers with Long Offset	NN(RRx)	where x is an even number in the range 0-255.(n=0,2,4,6....14) and NN is word expression in the range between 0 and 65535
rr(rrx)	Indexed Pair of Working Registers with a Pair of Working Registers used as Offset	rrn(rrx)	where n and x are two even numbers in the range 0-15. (n,x=0,2,4,6....14)
r.b	Bit pointer in a direct working register	rn.b	n=0.15 and b is a number between 0-7;0 LSB 7 MSB
(rr).b	Bit pointer in a Memory Location using a Pair of Indirect Working Registers as Address Pointer	(rrn).b	where n is an even number in the range 0-15.(n=0,2,4,6....14) and be is a number between 0-7 0 LSB 7 MSB
(RR)	Indirect pair of Register Pointer	(RRn)	where n is an even number in the range 0-255.(n=0,2,4,6....254)

3.5 INSTRUCTION SUMMARY

The following tables summarize the operation for each of the instructions which are listed with their corresponding mnemonic codes, addressing modes, byte counts, timing information, and affected flags.

GENERAL NOTES FOR THE ABOVE-MENTIONED TABLES :

- dst: destination operand
- src: source operand
- SSP: system stack pointer
- USP: user stack pointer
- PC: program counter
- cc: condition code
- C: carry flag
- Z: zero flag
- S: sign flag
- V: overflow flag
- D: decimal adjust flag
- CIC: central interrupt control register
- DP : data/program memory flag

FLAGS STATUS:

- ^ : affected
- - : not affected
- 0 : reset to zero
- 1 : set to one
- ? : undefined

Note: for detailed information on the instruction set refer to the ST9 programming manual.

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADC : Addition of 2 bytes with carry						
ADC	r	r	2	6	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	R	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	R	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	r	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	(r)	2	6	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	(r)	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	(rr)	3	12	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	(rr)	3	12	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	NN	4	18	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	N(rrx)	4	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	N(rrx)	4	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	NN(rrx)	5	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	NN(rrx)	5	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	rr(rrx)	3	22	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	(rr)+	3	16	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	R	(rr)+	3	16	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	r	-(rr)	3	16	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	-(rr)	3	16	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(r)	r	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(r)	R	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	r	3	18	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	R	3	18	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)+	r	3	22	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	(rr)+	R	3	22	dst<-dst+src+C rr<-rr+1	^ ^ ^ ^ 0 ^
ADC	NN	r	4	20	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	N(rrx)	r	4	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	N(rrx)	R	4	26	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	NN(rrx)	r	5	28	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	NN(rrx)	R	5	28	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	rr(rrx)	r	3	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	-(rr)	r	3	24	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	-(rr)	R	3	22	rr<-rr-1 dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	r	#N	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	R	#N	3	10	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	#N	3	16	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	NN	#N	5	24	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(rr)	(rr)	3	20	dst<-dst+src+C	^ ^ ^ ^ 0 ^
ADC	(RR)	(rr)	3	20	dst<-dst+src+C	^ ^ ^ ^ 0 ^

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADCW : Add word with carry						
ADCW	rr	rr	2	10	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	RR	3	12	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	RR	3	12	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	rr	3	12	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	(r)	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	(r)	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	(rr)	2	16	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	(rr)	3	18	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	NN	4	22	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	N(rrx)	4	28	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	N(rrx)	4	28	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	NN(rrx)	5	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	NN(rrx)	5	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	rr(rrx)	3	26	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	(rr)+	3	22	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	RR	(rr)+	3	22	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(r)	rr	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(r)	RR	3	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	rr	2	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	RR	3	30	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)+	rr	3	32	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	(rr)+	RR	3	32	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
ADCW	NN	rr	4	32	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	N(rrx)	rr	4	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	N(rrx)	RR	4	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN(rrx)	rr	5	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN(rrx)	RR	5	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr(rrx)	rr	3	34	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	-(rr)	rr	3	34	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	rr	#NN	4	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	RR	#NN	4	14	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	#NN	4	32	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN	#NN	6	36	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	N(rrx)	#NN	5	36	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	NN(rrx)	#NN	6	38	dst<-dst+src+C	^ ^ ^ ^ ? ?
ADCW	(rr)	(rr)	2	32	dst<-dst+src+C	^ ^ ^ ^ ? ?

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADD : Addition of 2 bytes without carry						
ADD	r	r	2	6	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	R	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	R	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	r	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	(r)	2	6	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	(r)	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	(rr)	3	12	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	(rr)	3	12	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	NN	4	18	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	N(rrx)	4	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	N(rrx)	4	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	NN(rrx)	5	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	NN(rrx)	5	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	rr(rrx)	3	22	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	(rr)+	3	16	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	R	(rr)+	3	16	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	r	-(rr)	3	16	rr<-rr-1 dst<-dst+src	♣ ^ ^ 0 ^
ADD	R	-(rr)	3	16	rr<-rr-1 dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(r)	r	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(r)	R	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	r	3	18	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	R	3	18	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)+	r	3	22	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	(rr)+	R	3	22	dst<-dst+src rr<-rr+1	^ ^ ^ ^ 0 ^
ADD	NN	r	4	20	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	N(rrx)	r	4	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	N(rrx)	R	4	26	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	NN(rrx)	r	5	28	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	NN(rrx)	R	5	28	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	rr(rrx)	r	3	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	-(rr)	r	3	22	rr<-rr-1 dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	-(rr)	R	3	22	rr<-rr-1 dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	r	#N	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	R	#N	3	10	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	#N	3	16	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	NN	#N	5	24	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(rr)	(rr)	3	20	dst<-dst+src	^ ^ ^ ^ 0 ^
ADD	(RR)	(rr)	3	20	dst<-dst+src	^ ^ ^ ^ 0 ^

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ADDW : Add word without carry						
ADDW	rr	rr	2	10	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	RR	3	12	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	RR	3	12	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	rr	3	12	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	(r)	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	(r)	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	(rr)	2	16	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	(rr)	3	18	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	NN	4	22	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	N(rrx)	4	28	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	N(rrx)	4	28	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	NN(rrx)	5	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	NN(rrx)	5	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	rr(rrx)	3	26	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	(rr)+	3	22	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	RR	(rr)+	3	22	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(r)	rr	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(r)	RR	3	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	rr	2	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	RR	3	30	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)+	rr	3	32	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	(rr)+	RR	3	32	dst<-dst+src rr<-rr+2	^ ^ ^ ^ ? ?
ADDW	NN	rr	4	32	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	N(rrx)	rr	4	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	N(rrx)	RR	4	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN(rrx)	rr	5	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN(rrx)	RR	5	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr(rrx)	rr	3	34	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	rr	#NN	4	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	RR	#NN	4	14	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	#NN	4	32	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN	#NN	6	36	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	N(rrx)	#NN	5	36	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	NN(rrx)	#NN	6	38	dst<-dst+src	^ ^ ^ ^ ? ?
ADDW	(rr)	(rr)	2	32	dst<-dst+src	^ ^ ^ ^ ? ?

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
AND : Logical AND between 2 bytes						
AND	r	r	2	6	dst<-dst AND src	- ^ ^ 0 - -
AND	R	R	3	10	dst<-dst AND src	- ^ ^ 0 - -
AND	r	R	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	R	r	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	r	(r)	2	6	dst<-ds AND src	- ^ ^ 0 - -
AND	R	(r)	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	r	(rr)	3	12	dst<-ds AND src	- ^ ^ 0 - -
AND	R	(rr)	3	12	dst<-ds AND src	- ^ ^ 0 - -
AND	r	NN	4	18	dst<-ds AND src	- ^ ^ 0 - -
AND	r	N(rrx)	4	24	dst<-ds AND src	- ^ ^ 0 - -
AND	R	N(rrx)	4	24	dst<-ds AND src	- ^ ^ 0 - -
AND	r	NN(rrx)	5	26	dst<-ds AND src	- ^ ^ 0 - -
AND	R	NN(rrx)	5	26	dst<-ds AND src	- ^ ^ 0 - -
AND	r	rr(rrx)	3	22	dst<-ds AND src	- ^ ^ 0 - -
AND	r	(rr)+	3	16	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	R	(rr)+	3	16	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	r	-(rr)	3	16	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	R	-(rr)	3	16	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	(r)	r	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	(r)	R	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	r	3	18	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	R	3	18	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)+	r	3	22	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	(rr)+	R	3	22	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
AND	NN	r	4	20	dst<-ds AND src	- ^ ^ 0 - -
AND	N(rrx)	r	4	26	dst<-ds AND src	- ^ ^ 0 - -
AND	N(rrx)	R	4	26	dst<-ds AND src	- ^ ^ 0 - -
AND	NN(rrx)	r	5	28	dst<-ds AND src	- ^ ^ 0 - -
AND	NN(rrx)	R	5	28	dst<-ds AND src	- ^ ^ 0 - -
AND	rr(rrx)	r	3	24	dst<-ds AND src	- ^ ^ 0 - -
AND	-(rr)	r	3	22	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	-(rr)	R	3	22	rr<-rr-1 dst<-ds AND src	- ^ ^ 0 - -
AND	r	#N	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	R	#N	3	10	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	#N	3	16	dst<-ds AND src	- ^ ^ 0 - -
AND	NN	#N	5	24	dst<-ds AND src	- ^ ^ 0 - -
AND	(rr)	(rr)	3	20	dst<-ds AND src	- ^ ^ 0 - -
AND	(RR)	(rr)	3	20	dst<-ds AND src	- ^ ^ 0 - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ANDW : Logical AND between two words						
ANDW	rr	rr	2	10	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	RR	3	12	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	RR	3	12	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	rr	3	12	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	(r)	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	(r)	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	(rr)	2	16	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	(rr)	3	18	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	NN	4	22	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	N(rrx)	4	28	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	N(rrx)	4	28	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	NN(rrx)	5	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	NN(rrx)	5	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	rr(rrx)	3	26	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	(rr)+	3	22	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	RR	(rr)+	3	22	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	(r)	rr	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(r)	RR	3	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	rr	2	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	RR	3	30	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)+	rr	3	32	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	(rr)+	RR	3	32	dst<-dst AND src rr<-rr+2	- ^ ^ 0 - -
ANDW	NN	rr	4	32	dst<-dst AND src	- ^ ^ 0 - -
ANDW	N(rrx)	rr	4	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	N(rrx)	RR	4	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN(rrx)	rr	5	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN(rrx)	RR	5	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr(rrx)	rr	3	34	dst<-dst AND src	- ^ ^ 0 - -
ANDW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst AND src	- ^ ^ 0 - -
ANDW	rr	#NN	4	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	RR	#NN	4	14	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	#NN	4	32	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN	#NN	6	36	dst<-dst AND src	- ^ ^ 0 - -
ANDW	N(rrx)	#NN	5	36	dst<-dst AND src	- ^ ^ 0 - -
ANDW	NN(rrx)	#NN	6	38	dst<-dst AND src	- ^ ^ 0 - -
ANDW	(rr)	(rr)	2	32	dst<-dst AND src	- ^ ^ 0 - -

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
BAND : Bit AND						
BAND BAND	r.b r.b	r.b r.!b	3 3	14 14	dst bit<-dst bit AND src bit dst bit<-dst bit AND complemented src bit	- - - - - - - - - - - -
BCPL : Bit Complement						
BCPL	r.b		2	6	dst bit<-dst bit complemented	- - - - - -
BLD : Bit Load						
BLD BLD	r.b r.b	r.b r.!b	3 3	14 14	dst bit<-src bit dst bit<-src bit complemented	- - - - - - - - - - - -
BOR : Bit OR						
BOR BOR	r.b r.b	r.b r.!b	3 3	14 14	dst bit<-dst bit OR src bit dst bit<-dst bit OR complemented src bit	- - - - - - - - - - - -
BRES : Bit Reset						
BRES	r.b		2	6	dst bit<- 0	- - - - - -
BSET : Bit Set						
BSET	r.b		2	6	dst bit<- 1	- - - - - -
BTJF, BTJT : Bit test and jump						
BTJF	r.b	N	3	14/16	If test bit is 0, PC<-PC+N	- - - - - -
BTJT	r.b	N	3	14/16	If test bit is 1, PC<-PC+N	- - - - - -
BXOR : Bit Exclusive OR						
BXOR BXOR	r.b r.b	r.b r.!b	3 3	14 14	dst bit<-dst bit XOR src bit dst bit<-dst bit XOR complemented src bit	- - - - - - - - - - - -
BTSET : Bit Test and Set						
BTSET BTSET	r.b (rr).b		2 2	8 20	If test bit = 0, test bit <-1,Z<-1 If test bit = 0, test bit <-1,Z<-1	- ^ ^ 0 - - - ^ ^ 0 - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
CALL : Call a subroutine						
CALL	NN		3	18	SSP<-SSP-2,(SP)< PC, PC<-dst	- - - - - -
CALL	(rr)		2	16	" "	- - - - - -
CALL	(RR)		2	16	" "	- - - - - -
CCF : Complement Carry Flag						
CCF			1	6	C <- C complemented	- - - - - -
CLR : Clear register						
CLR	r		2	6	dst<-0	- - - - - -
CLR	R		2	6	dst<-0	- - - - - -
CLR	(r)		2	6	dst<-0	- - - - - -
CLR	(R)		2	6	dst<-0	- - - - - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
CP : Compare bytes						
CP	r	r	2	6	dst-src	^ ^ ^ ^ - -
CP	R	R	3	10	dst-src	^ ^ ^ ^ - -
CP	r	R	3	10	dst-src	^ ^ ^ ^ - -
CP	R	r	3	10	dst-src	^ ^ ^ ^ - -
CP	r	(r)	2	6	dst-src	^ ^ ^ ^ - -
CP	R	(r)	3	10	dst-src	^ ^ ^ ^ - -
CP	r	(rr)	3	12	dst-src	^ ^ ^ ^ - -
CP	R	(rr)	3	12	dst-src	^ ^ ^ ^ - -
CP	r	NN	4	18	dst-src	^ ^ ^ ^ - -
CP	r	N(rrx)	4	24	dst-src	^ ^ ^ ^ - -
CP	R	N(rrx)	4	24	dst-src	^ ^ ^ ^ - -
CP	r	NN(rrx)	5	26	dst-src	^ ^ ^ ^ - -
CP	R	NN(rrx)	5	26	dst-src	^ ^ ^ ^ - -
CP	r	rr(rrx)	3	22	dst-src	^ ^ ^ ^ - -
CP	r	(rr)+	3	16	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	R	(rr)+	3	16	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	r	-(rr)	3	16	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	R	-(rr)	3	16	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	(r)	r	3	10	dst-src	^ ^ ^ ^ - -
CP	(r)	R	3	10	dst-src	^ ^ ^ ^ - -
CP	(rr)	r	3	18	dst-src	^ ^ ^ ^ - -
CP	(rr)	R	3	18	dst-src	^ ^ ^ ^ - -
CP	(rr)+	r	3	22	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	(rr)+	R	3	22	dst-src,rr<-rr+1	^ ^ ^ ^ - -
CP	NN	r	4	20	dst-src	^ ^ ^ ^ - -
CP	N(rrx)	r	4	26	dst-src	^ ^ ^ ^ - -
CP	N(rrx)	R	4	26	dst-src	^ ^ ^ ^ - -
CP	NN(rrx)	r	5	28	dst-src	^ ^ ^ ^ - -
CP	NN(rrx)	R	5	28	dst-src	^ ^ ^ ^ - -
CP	rr(rrx)	r	3	24	dst-src	^ ^ ^ ^ - -
CP	-(rr)	r	3	22	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	-(rr)	R	3	22	rr<-rr-1,dst-src	^ ^ ^ ^ - -
CP	r	#N	3	10	dst-src	^ ^ ^ ^ - -
CP	R	#N	3	10	dst-src	^ ^ ^ ^ - -
CP	(rr)	#N	3	16	dst-src	^ ^ ^ ^ - -
CP	NN	#N	5	22	dst-src	^ ^ ^ ^ - -
CP	(rr)	(rr)	3	18	dst-src	^ ^ ^ ^ - -
CP	(RR)	(rr)	3	18	dst-src	^ ^ ^ ^ - -
CPL : Complement register						
CPL	r		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPL	R		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPL	(r)		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPL	(R)		2	6	dst<- NOT dst	- ^ ^ 0 - -
CPJFI, CPJTI : Compare with post-increment						
CPJFI	(rr)	r,N	3	22/24	If compare not verified jump otherwise post-increment	- - - - - -
CPJTI	(rr)	r,N	3	22/24	If compare verified jump otherwise post-increment	- - - - - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
CPW : Compare word						
CPW	rr	rr	2	10	dst-src	^ ^ ^ ^ - -
CPW	RR	RR	3	12	dst-src	^ ^ ^ ^ - -
CPW	rr	RR	3	12	dst-src	^ ^ ^ ^ - -
CPW	RR	rr	3	12	dst-src	^ ^ ^ ^ - -
CPW	rr	(r)	3	14	dst-src	^ ^ ^ ^ - -
CPW	RR	(r)	3	14	dst-src	^ ^ ^ ^ - -
CPW	rr	(rr)	2	16	dst-src	^ ^ ^ ^ - -
CPW	RR	(rr)	3	18	dst-src	^ ^ ^ ^ - -
CPW	rr	NN	4	22	dst-src	^ ^ ^ ^ - -
CPW	rr	N(rrx)	4	28	dst-src	^ ^ ^ ^ - -
CPW	RR	N(rrx)	4	28	dst-src	^ ^ ^ ^ - -
CPW	rr	NN(rrx)	5	30	dst-src	^ ^ ^ ^ - -
CPW	RR	NN(rrx)	5	30	dst-src	^ ^ ^ ^ - -
CPW	rr	rr(rrx)	3	26	dst-src	^ ^ ^ ^ - -
CPW	rr	(rr)+	3	22	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	RR	(rr)+	3	22	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	rr	-(rr)	3	24	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	RR	-(rr)	3	24	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	(r)	rr	3	14	dst-src	^ ^ ^ ^ - -
CPW	(r)	RR	3	14	dst-src	^ ^ ^ ^ - -
CPW	(rr)	rr	2	26	dst-src	^ ^ ^ ^ - -
CPW	(rr)	RR	3	28	dst-src	^ ^ ^ ^ - -
CPW	(rr)+	rr	3	30	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	(rr)+	RR	3	30	dst-src rr<-rr+2	^ ^ ^ ^ - -
CPW	NN	rr	4	30	dst-src	^ ^ ^ ^ - -
CPW	N(rrx)	rr	4	36	dst-src	^ ^ ^ ^ - -
CPW	N(rrx)	RR	4	36	dst-src	^ ^ ^ ^ - -
CPW	NN(rrx)	rr	5	36	dst-src	^ ^ ^ ^ - -
CPW	NN(rrx)	RR	5	36	dst-src	^ ^ ^ ^ - -
CPW	rr(rrx)	rr	3	32	dst-src	^ ^ ^ ^ - -
CPW	-(rr)	rr	3	30	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	-(rr)	RR	3	30	rr<-rr-2 dst-src	^ ^ ^ ^ - -
CPW	rr	#NN	4	14	dst-src	^ ^ ^ ^ - -
CPW	RR	#NN	4	14	dst-src	^ ^ ^ ^ - -
CPW	(rr)	#NN	4	30	dst-src	^ ^ ^ ^ - -
CPW	NN	#NN	6	34	dst-src	^ ^ ^ ^ - -
CPW	N(rrx)	#NN	5	34	dst-src	^ ^ ^ ^ - -
CPW	NN(rrx)	#NN	6	36	dst-src	^ ^ ^ ^ - -
CPW	(rr)	(rr)	2	32	dst-src	^ ^ ^ ^ - -

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
DA : Decimal adjust						
DA	r		2	6	dst<- DA dst	^ ^ ^ ? - -
DA	R		2	6	dst<- DA dst	^ ^ ^ ? - -
DA	(r)		2	6	dst<- DA dst	^ ^ ^ ? - -
DA	(R)		2	6	dst<- DA dst	^ ^ ^ ? - -
DEC : Decrement						
DEC	r		2	6	dst<- dst-1	- ^ ^ ^ - -
DEC	R		2	6	dst<- dst-1	- ^ ^ ^ - -
DEC	(r)		2	6	dst<- dst-1	- ^ ^ ^ - -
DEC	(R)		2	6	dst<- dst-1	- ^ ^ ^ - -
DECW : Decrement Word						
DECW	rr		2	8	dst<-dst-1	- ^ ^ ^ - -
DECW	RR		2	8	dst<-dst-1	- ^ ^ ^ - -
DI : Disable Interrupts						
DI			1	6	Bit 4 of the CIC Register is set to 0	- - - - - -
DIV : Divide 16 by 8						
DIV	rr	r	2	28/20	dst / src <- dst high=remainder 16/8 <- dst low=result	note 1
DIVWS : Divide Word Stepped 32 by 16						
DIVWS	rrhigh rrlow	rr	3	28	32/16	note 1
DJNZ : Decrement a working register and Jump if Non Zero						
DJNZ	r	N	2	10/12	r <- r-1, If r=0 then PC<-PC+N	note 2
DWJNZ : Decrement a register pair and Jump if Non Zero						
DWJNZ	rr	N	3	12/16	rr<-rr-1, If rr=0 then PC<-PC+N	note 2
DWJNZ	RR	N	3	12/16	RR<-RR-1,If RR=0 then PC<-PC+N	

Notes :
1 Refer to the ST9 Programming Manual for detailed information
2 Working registers in groups D, E and F are not allowed

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
EI : Enable Interrupts						
EI			1	6	Bit 4 of the CICR register is set to 1	- - - - - -
EXT : Sign extend						
EXT EXT	rr RR		2 2	10 10	r(7) --> r(n) n=8-15 R(7) --> R(n) n=8-15	- - - - - - - - - - - -
HALT						
HALT			2	6	Stops all internal clocks until next system reset	- - - - - -
INC : Increment						
INC INC INC INC	r R (r) (R)		2 2 2 2	6 6 6 6	dst<- dst+1 dst<- dst+1 dst<- dst+1 dst<- dst+1	- ^ ^ ^ - - - ^ ^ ^ - - - ^ ^ ^ - - - ^ ^ ^ - -
INCW : Increment Word						
INCW INCW	rr RR		2 2	8 8	dst<-dst+1 dst<-dst+1	- ^ ^ ^ - - - ^ ^ ^ - -
IRET : Return from Interrupt Routine						
IRET			1	16	FLAGS<-(SSP),SSP<-SSP+1, PC<-(SSP), SSP<-SPP+2, CIC(4)<-1	note 1
JP : Jump to a Routine						
JP	NN		3	10	PC<-dst	- - - - - -
JP	(rr)		2	8	PC<-dst	- - - - - -
JP	(RR)		2	8	PC<-dst	- - - - - -
JPcc	NN		3	10	IF cc(condition code) is true, PC<-dst	- - - - - -
JRcc : Conditional Relative Jump to a Routine						
JRcc	N		2	10/12	IF cc(condition code)is true, PC<-PC+dst	- - - - - -

Note 1 : All flags are restored to original setting (before interrupt occured)

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
LD : Load byte instructions						
LD	r	r	2	6	dst<-src	- - - - - -
LD	R	R	3	10	dst<-src	- - - - - -
LD	r	R	2	6	dst<-src	- - - - - -
LD	R	r	2	6	dst<-src	- - - - - -
LD	r	(r)	2	6	dst<-src	- - - - - -
LD	R	(r)	3	10	dst<-src	- - - - - -
LD	r	(rr)	2	10	dst<-src	- - - - - -
LD	R	(rr)	3	12	dst<-src	- - - - - -
LD	r	NN	4	18	dst<-src	- - - - - -
LD	r	N(rx)	3	10	dst<-src	- - - - - -
LD	r	N(rrx)	4	24	dst<-src	- - - - - -
LD	R	N(rrx)	4	24	dst<-src	- - - - - -
LD	r	NN(rrx)	5	26	dst<-src	- - - - - -
LD	R	NN(rrx)	5	26	dst<-src	- - - - - -
LD	r	rr(rrx)	3	22	dst<-src	- - - - - -
LD	r	(rr)+	3	16	dst<-src,rr<-rr+1	- - - - - -
LD	R	(rr)+	3	16	dst<-src,rr<-rr+1	- - - - - -
LD	r	-(rr)	3	16	rr<-rr-1,dst<-src	- - - - - -
LD	R	-(rr)	3	16	rr<-rr-1,dst<-src	- - - - - -
LD	(r)	r	2	6	dst<-src	- - - - - -
LD	(r)	R	3	10	dst<-src	- - - - - -
LD	(rr)	r	2	10	dst<-src	- - - - - -
LD	(rr)	R	3	14	dst<-src	- - - - - -
LD	(rr)+	r	3	18	dst<-src,rr<-rr+1	- - - - - -
LD	(rr)+	R	3	18	dst<-src,rr<-rr+1	- - - - - -
LD	NN	r	4	18	dst-src	- - - - - -
LD	N(rx)	r	3	10	dst-src	- - - - - -
LD	N(rrx)	r	4	24	dst-src	- - - - - -
LD	N(rrx)	R	4	24	dst-src	- - - - - -
LD	NN(rrx)	r	5	26	dst-src	- - - - - -
LD	NN(rrx)	R	5	26	dst-src	- - - - - -
LD	rr(rrx)	r	3	22	dst-src	- - - - - -
LD	-(rr)	r	3	18	rr<-rr-1,dst<-src	- - - - - -
LD	-(rr)	R	3	18	rr<-rr-1,dst<-src	- - - - - -
LD	r	#N	2	6	dst<-src	- - - - - -
LD	R	#N	3	10	dst<-src	- - - - - -
LD	(rr)	#N	3	12	dst<-src	- - - - - -
LD	NN	#N	5	20	dst<-src	- - - - - -
LD	(rr)	(rr)	3	16	dst<-src	- - - - - -
LD	(RR)	(rr)	3	16	dst<-src,	- - - - - -
LD	(r)+	(rr)+	2	14	rr<-rr+1,r<-r+1	- - - - - -
LD	(rr)+	(r)+	2	18	rr<-rr+1,r<-r+1	- - - - - -
LDPP,LDDP,LDPD, LDDD : Load from / to program / data memory						
LDPP	(rr)+	(rr)+	2	16	dst<-src (note 1), rr<-rr+1	- - - - - -
LDDP	(rr)+	(rr)+	2	16	dst<-src (note 2), rr<-rr+1	- - - - - -
LDPD	(rr)+	(rr)+	2	16	dst<-src (note 3), rr<-rr+1	- - - - - -
LDDD	(rr)+	(rr)+	2	16	dst<-src (note 4), rr<-rr+1	- - - - - -

Notes:
1. dst in Program Memory, src in Program Memory
2 dst in Data Memory, src in Program Memory
3. dst in Program Memory, src in Data Memory
4 dst in Data Memory, src in Data Memory

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
LDW : Load word instructions						
LDW	rr	rr	2	10	dst<-src	- - - - - -
LDW	RR	RR	3	10	dst<-src	- - - - - -
LDW	rr	RR	3	10	dst<-src	- - - - - -
LDW	RR	rr	3	10	dst<-src	- - - - - -
LDW	rr	(r)	3	10	dst<-src	- - - - - -
LDW	RR	(r)	3	10	dst<-src	- - - - - -
LDW	rr	(rr)	2	16	dst<-src	- - - - - -
LDW	RR	(rr)	3	18	dst<-src	- - - - - -
LDW	rr	NN	4	22	dst<-src	- - - - - -
LDW	rr	N(rx)	3	16	dst<-src	- - - - - -
LDW	rr	N(rrx)	4	28	dst<-ssc	- - - - - -
LDW	RR	N(rrx)	4	28	dst<-src	- - - - - -
LDW	rr	NN(rrx)	5	30	dst<-src	- - - - - -
LDW	RR	NN(rrx)	5	30	dst<-src	- - - - - -
LDW	rr	rr(rrx)	3	24	dst<-src	- - - - - -
LDW	rr	(rr)+	3	20	dst<-src,rr<-rr+2	- - - - - -
LDW	RR	(rr)+	3	20	dst<-src,rr<-rr+2	- - - - - -
LDW	rr	-(rr)	3	22	rr<-rr-2,dst<-src	- - - - - -
LDW	RR	-(rr)	3	22	rr<-rr-2,dst<-src	- - - - - -
LDW	(r)	rr	3	10	dst<-src	- - - - - -
LDW	(r)	RR	3	10	dst<-src	- - - - - -
LDW	(rr)	rr	2	18	dst<-src	- - - - - -
LDW	(rr)	RR	3	20	dst<-src	- - - - - -
LDW	(rr)+	rr	3	24	rr<-rr+2,dst<-src	- - - - - -
LDW	(rr)+	RR	3	24	rr<-rr+2,dst<-src	- - - - - -
LDW	NN	rr	4	22	dst<-src	- - - - - -
LDW	N(rx)	rr	3	14	dst<-src	- - - - - -
LDW	N(rrx)	RR	4	26	dst<-src	- - - - - -
LDW	N(rrx)	rr	4	26	dst<-src	- - - - - -
LDW	NN(rrx)	RR	5	28	dst<-src	- - - - - -
LDW	NN(rrx)	rr	5	28	dst<-src	- - - - - -
LDW	rr(rrx)	rr	3	24	dst<-src	- - - - - -
LDW	-(rr)	rr	3	26	rr<-rr-2,dst<-src	- - - - - -
LDW	-(rr)	RR	3	26	rr<-rr-2,dst<-src	- - - - - -
LDW	rr	#NN	4	12	dst<-src	- - - - - -
LDW	RR	#NN	4	12	dst<-src	- - - - - -
LDW	(rr)	#NN	4	22	dst<-src	- - - - - -
LDW	N(rrx)	#NN	5	28	dst<-src	- - - - - -
LDW	NN(rrx)	#NN	6	30	dst<-src	- - - - - -
LDW	NN	#NN	6	26	dst<-src	- - - - - -
LDW	(rr)	(rr)	2	22	dst<-src	- - - - - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
MUL : Multiply						
MUL	rr	r	2	22	dst <- dst x src, 8 x 8 multiply	note 1
NOP : No operation						
NOP			1	6	No Operation	- - - - - -
OR : Logical OR between 2 bytes						
OR	r	r	2	6	dst<-dst OR src	- ^ ^ 0 - -
OR	R	R	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	r	R	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	R	r	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	r	(r)	2	6	dst<-dst OR src	- ^ ^ 0 - -
OR	R	(r)	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	r	(rr)	3	12	dst<-dst OR src	- ^ ^ 0 - -
OR	R	(rr)	3	12	dst<-dst OR src	- ^ ^ 0 - -
OR	r	NN	4	18	dst<-dst OR src	- ^ ^ 0 - -
OR	r	N(rrx)	4	24	dst<-dst OR src	- ^ ^ 0 - -
OR	R	N(rrx)	4	24	dst<-dst OR src	- ^ ^ 0 - -
OR	r	NN(rrx)	5	26	dst<-dst OR src	- ^ ^ 0 - -
OR	R	NN(rrx)	5	26	dst<-dst OR src	- ^ ^ 0 - -
OR	r	rr(rrx)	3	22	dst<-dst OR src	- ^ ^ 0 - -
OR	r	(rr)+	3	16	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	R	(rr)+	3	16	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	r	-(rr)	3	16	rr<-rr-1 dst<-dst OR src	- ^ ^ 0 - -
OR	R	-(rr)	3	16	rr<-rr-1 dst<-dst OR src	- ^ ^ 0 - -
OR	(r)	r	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	(r)	R	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	r	3	18	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	R	3	18	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)+	r	3	22	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	(rr)+	R	3	22	dst<-dst OR src rr<-rr+1	- ^ ^ 0 - -
OR	NN	r	4	20	dst<-dst OR src	- ^ ^ 0 - -
OR	N(rrx)	r	4	26	dst<-dst OR src	- ^ ^ 0 - -
OR	N(rrx)	R	4	26	dst<-dst OR src	- ^ ^ 0 - -
OR	NN(rrx)	r	5	28	dst<-dst OR src	- ^ ^ 0 - -
OR	NN(rrx)	R	5	28	dst<-dst OR src	- ^ ^ 0 - -
OR	rr(rrx)	r	3	24	dst<-dst OR src	- ^ ^ 0 - -
OR	-(rr)	r	3	22	dst<-dst OR src rr<-rr-1	- ^ ^ 0 - -
OR	-(rr)	R	3	22	dst<-dst OR src rr<-rr-1	- ^ ^ 0 - -
OR	r	#N	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	R	#N	3	10	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	#N	3	16	dst<-dst OR src	- ^ ^ 0 - -
OR	NN	#N	5	24	dst<-dst OR src	- ^ ^ 0 - -
OR	(rr)	(rr)	3	20	dst<-dst OR src	- ^ ^ 0 - -
OR	(RR)	(rr)	3	20	dst<-dst OR src	- ^ ^ 0 - -

Note 1. Refer to programming manual for detailed information.

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
ORW : Logical OR between two words						
ORW	rr	rr	2	10	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	RR	3	12	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	RR	3	12	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	rr	3	12	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	(r)	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	(r)	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	(rr)	2	16	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	(rr)	3	18	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	NN	4	22	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	N(rrx)	4	28	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	N(rrx)	4	28	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	NN(rrx)	5	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	NN(rrx)	5	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	rr(rrx)	3	26	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	(rr)+	3	22	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	RR	(rr)+	3	22	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	(r)	rr	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	(r)	RR	3	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	rr	2	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	RR	3	30	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)+	rr	3	32	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	(rr)+	RR	3	32	dst<-dst OR src rr<-rr+2	- ^ ^ 0 - -
ORW	NN	rr	4	32	dst<-dst OR src	- ^ ^ 0 - -
ORW	N(rrx)	rr	4	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	N(rrx)	RR	4	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN(rrx)	rr	5	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN(rrx)	RR	5	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	rr(rrx)	rr	3	34	dst<-dst OR src	- ^ ^ 0 - -
ORW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst OR src	- ^ ^ 0 - -
ORW	rr	#NN	4	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	RR	#NN	4	14	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	#NN	4	32	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN	#NN	6	36	dst<-dst OR src	- ^ ^ 0 - -
ORW	N(rrx)	#NN	5	36	dst<-dst OR src	- ^ ^ 0 - -
ORW	NN(rrx)	#NN	6	38	dst<-dst OR src	- ^ ^ 0 - -
ORW	(rr)	(rr)	2	32	dst<-dst OR src	- ^ ^ 0 - -

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
PEA : Push effective address on system stack						
PEA		N(rrx)	4	20	SSP<-USP-2, (SSP)<-rrx+N	- - - - - -
PEA		NN(rrx)	5	26	SSP<-USP-2, (SSP)<-rrx+N	- - - - - -
PEA		N(RRx)	4	20	SSP<-USP-2, (SSP)<-RRx+N	- - - - - -
PEA		NN(RRx)	5	26	SSP<-USP-2, (SSP)<-RRx+N	- - - - - -
PEAU : Push effective address on user stack						
PEAU		N(rrx)	4	20	USP<-USP-2, (USP)<-rrx+N	- - - - - -
PEAU		NN(rrx)	5	26	USP<-USP-2, (USP)<-rrx+N	- - - - - -
PEAU		N(RRx)	4	20	USP<-USP-2, (USP)<-RRx+N	- - - - - -
PEAU		NN(RRx)	5	26	USP<-USP-2, (USP)<-RRx+N	- - - - - -
POP : Pop system stack						
POP	r		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POP	R		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POP	(r)		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POP	(R)		2	10	dst<-(SSP), SSP<-SSP+1	- - - - - -
POPU : Pop user stack						
POPU	r		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPU	R		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPU	(r)		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPU	(R)		2	10	dst<-(USP), USP<-USP+1	- - - - - -
POPUW : Pop word from user stack						
POPUW	rr		2	14	dst<-(USP), USP<-USP+2	- - - - - -
POPUW	RR		2	14	dst<-(USP), USP<-USP+2	- - - - - -
POPW : Pop word from system stack						
POPW	rr		2	14	dst<-(SSP), SSP<-SSP+2	- - - - - -
POPW	RR		2	14	dst<-(SSP), SSP<-SSP+2	- - - - - -
PUSH : Push system stack						
PUSH		r	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		R	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		(r)	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		(R)	2	10	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSH		#N	3	16	SSP<-SSP-1, (SSP)<-src	- - - - - -
PUSHU : Push user stack						
PUSHU		r	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		R	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		(r)	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		(R)	2	10	USP<-USP-1, (USP)<-src	- - - - - -
PUSHU		#N	3	16	USP<-USP-1, (USP)<-src	- - - - - -
PUSHUW : Push word on user stack						
PUSHUW		rr	2	12	USP<-USP-2, (USP)<-src	- - - - - -
PUSHUW		RR	2	12	USP<-USP-2, (USP)<-src	- - - - - -
PUSHUW		#NN	4	20	USP<-USP-2, (USP)<-src	- - - - - -
PUSHW : Push Word on System Stack						
PUSHW		rr	2	12	SSP<-SSP-2, (SSP)<-src	- - - - - -
PUSHW		RR	2	12	SSP<-SSP-2, (SSP)<-src	- - - - - -
PUSHW		#NN	4	20	SSP<-SSP-2, (SSP)<-src	- - - - - -

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
RCF : Reset carry flag						
RCF			1	6	C <- 0	0 - - - - -
RET : Return from subroutine						
RET			1	12	PC <- (SSP), SSP <- SPP+2	- - - - - -
RLC : Rotate left through carry						
RLC	r		2	6	dst(0)<-C, C<-dst(7) dst(n+1)<-dst(n) n=0-6	^ ^ ^ ^ - -
RLC	R		2	6	" "	^ ^ ^ ^ - -
RLC	(r)		2	6	" "	^ ^ ^ ^ - -
RLC	(R)		2	6	" "	^ ^ ^ ^ - -
RLCW : Rotate word left through carry						
RLCW	rr		2	8	dst(0)<-C, C<-dst(15) dst(n+1)<-dst(n) n=0-14	
RLCW	RR		2	8	" "	
ROL : Rotate left						
ROL	r		2	6	C<-dst(7), dst(0)<-dst(7) dst(n+1)<-dst(n) n=0-6	^ ^ ^ ^ - -
ROL	R		2	6	" "	^ ^ ^ ^ - -
ROL	(r)		2	6	" "	^ ^ ^ ^ - -
ROL	(R)		2	6	" "	^ ^ ^ ^ - -
ROR : Rotate right						
ROR	r		2	6	C<-dst(0), dst(7)<-dst(0) dst(n)<-dst(n+1) n=0-6	^ ^ ^ ^ - -
ROR	R		2	6	" "	^ ^ ^ ^ - -
ROR	(r)		2	6	" "	^ ^ ^ ^ - -
ROR	(R)		2	6	" "	^ ^ ^ ^ - -
RRC : Rotate right through carry						
RRC	r		2	6	dst(7)<-C, C<-dst(0) dst(n)<-dst(n+1) n=0-6	^ ^ ^ ^ - -
RRC	R		2	6	" "	^ ^ ^ ^ - -
RRC	(r)		2	6	" "	^ ^ ^ ^ - -
RRC	(R)		2	6	" "	^ ^ ^ ^ - -
RRCW : Rotate word right through carry						
RRCW	rr		2	8	dst(15)<-C, C<-dst(0) dst(n)<-dst(n+1) n=0-14	^ ^ ^ ^ - -
RRCW	RR		2	8	" "	^ ^ ^ ^ - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SBC : Subtraction of 2 bytes with carry						
SBC	r	r	2	6	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	R	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	R	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	r	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	(r)	2	6	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	(r)	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	(rr)	3	12	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	(rr)	3	12	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	NN	4	18	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	N(rrx)	4	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	N(rrx)	4	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	NN(rrx)	5	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	NN(rrx)	5	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	rr(rrx)	3	22	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	(rr)+	3	16	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	R	(rr)+	3	16	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	r	-(rr)	3	16	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	-(rr)	3	16	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(r)	r	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(r)	R	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	r	3	18	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	R	3	18	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)+	r	3	22	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	(rr)+	R	3	22	dst<-dst-src-C rr<-rr+1	^ ^ ^ ^ 1 ^
SBC	NN	r	4	20	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	N(rrx)	r	4	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	N(rrx)	R	4	26	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	NN(rrx)	r	5	28	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	NN(rrx)	R	5	28	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	rr(rrx)	r	3	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	-(rr)	r	3	22	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	-(rr)	R	3	22	rr<-rr-1 dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	r	#N	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	R	#N	3	10	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	#N	3	16	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	NN	#N	5	24	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(rr)	(rr)	3	20	dst<-dst-src-C	^ ^ ^ ^ 1 ^
SBC	(RR)	(rr)	3	20	dst<-dst-src-C	^ ^ ^ ^ 1 ^

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SBCW : Subtract word with carry						
SBCW	rr	rr	2	10	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	RR	3	12	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	RR	3	12	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	rr	3	12	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	(r)	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	(r)	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	(rr)	2	16	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	(rr)	3	18	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	NN	4	22	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	N(rrx)	4	28	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	N(rrx)	4	28	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	NN(rrx)	5	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	NN(rrx)	5	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	rr(rrx)	3	26	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	(rr)+	3	22	dst<-dst-src-C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	RR	(rr)+	3	22	dst<-dst+src+C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(r)	rr	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(r)	RR	3	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	rr	2	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	RR	3	30	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)+	rr	3	32	dst<-dst-src-C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	(rr)+	RR	3	32	dst<-dst-src-C rr<-rr+2	^ ^ ^ ^ ? ?
SBCW	NN	rr	4	32	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	N(rrx)	rr	4	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	N(rrx)	RR	4	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN(rrx)	rr	5	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN(rrx)	RR	5	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr(rrx)	rr	3	34	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	rr	#NN	4	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	RR	#NN	4	14	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	#NN	4	32	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN	#NN	6	36	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	N(rrx)	#NN	5	36	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	NN(rrx)	#NN	6	38	dst<-dst-src-C	^ ^ ^ ^ ? ?
SBCW	(rr)	(rr)	2	32	dst<-dst-src-C	^ ^ ^ ^ ? ?

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SCF : Set carry flag						
SCF			1	6	C <- 1	1 - - - - -
SDM : Set data memory						
SDM			1	6	Set Data Memory DP<-1 Note 1	- - - - - -
SLA : Shift left arithmetic						
SLA	r R (rr)		2 3 3	6 10 20	dst C<-dst(7), dst (0)<-0 dst(n+1)<-dst(n)n=0-6 " " " "	^ ^ ^ ^ 0 - ^ ^ ^ ^ 0 - ^ ^ ^ ^ 0 -
SLAW : Shift word left arithmetic						
SLAW	rr RR (rr)		2 3 2	10 12 32	C<-dst(15), dst (0)<-0 dst(n+1)<-dst(n)n=1-14 " " " "	^ ^ ^ ^ - - ^ ^ ^ ^ - - ^ ^ ^ ^ - -
SPM : Set program memory						
SPM			1	6	Set Program Memory DP<-0 Note 2	- - - - - -
SPP : Set page pointer						
SPP		#N	2	6	Set Page Pointer	- - - - - -
SRA : Shift right arithmetic						
SRA SRA SRA SRA	r R (r) (R)		2 2 2 2	6 6 6 6	dst(7)<-dst(7), C<-dst(0) dst(n)<-dst(n+1)n=0-6 " " " " " "	^ ^ ^ ^ 0 ^ ^ ^ ^ ^ 0 ^ ^ ^ ^ ^ 0 ^ ^ ^ ^ ^ 0 ^
SRAW : Shift word right arithmetic						
SRAW SRAW	rr RR		2 2	6 8	dst(15)<-dst(15), C<-dst(0) dst(n)<-dst(n+1)n=0-14 " "	^ ^ ^ 0 - - ^ ^ ^ 0 - -

Note 1 : Following this instruction, all addressing modes refering to address spaces will refer to the Data Space.

Note 2 : Following this instruction, all addressing modes refering to address spaces will refer to the Program Space, except for the following instructions which operate with Dataspace independently of the setting of the DP flag :
PUSH(W)/PUSHU(W), POP(W)/POPU(W), PEA/PEAU, and CALL, RET, IRET and interrupt execution (assuming the Stack Pointers are not pointing to the Register File.

Mnemo	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SRP : Set register pointer						
SRP		#N	2	6	Set Register Pointer	- - - - - -
SRP0 : Set register pointer 0						
SRP0		#N	2	6	Set Register Pointer 0	- - - - - -
SRP1 : Set register pointer 1						
SRP1		#N	2	6	Set Register Pointer 1	- - - - - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SUB : Subtraction of 2 bytes without carry						
SUB	r	r	2	6	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	R	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	R	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	r	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	(r)	2	6	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	(r)	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	(rr)	3	12	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	(rr)	3	12	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	NN	4	18	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	N(rrx)	4	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	N(rrx)	4	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	NN(rrx)	5	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	NN(rrx)	5	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	rr(rrx)	3	22	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	(rr)+	3	16	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	R	(rr)+	3	16	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	r	-(rr)	3	16	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	-(rr)	3	16	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(r)	r	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(r)	R	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	r	3	18	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	R	3	18	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)+	r	3	22	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	(rr)+	R	3	22	dst<-dst-src rr<-rr+1	^ ^ ^ ^ 1 ^
SUB	NN	r	4	20	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	N(rrx)	r	4	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	N(rrx)	R	4	26	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	NN(rrx)	r	5	28	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	NN(rrx)	R	5	28	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	rr(rrx)	r	3	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	-(rr)	r	3	22	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	-(rr)	R	3	22	rr<-rr-1 dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	r	#N	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	R	#N	3	10	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	#N	3	16	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	NN	#N	5	24	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(rr)	(rr)	3	20	dst<-dst-src	^ ^ ^ ^ 1 ^
SUB	(RR)	(rr)	3	20	dst<-dst-src	^ ^ ^ ^ 1 ^

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
SUBW : Subtract words						
SUBW	rr	rr	2	10	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	RR	3	12	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	RR	3	12	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	rr	3	12	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	(r)	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	(r)	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	(rr)	2	16	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	(rr)	3	18	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	NN	4	22	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	N(rrx)	4	28	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	N(rrx)	4	28	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	NN(rrx)	5	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	NN(rrx)	5	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	rr(rrx)	3	26	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	(rr)+	3	22	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	RR	(rr)+	3	22	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(r)	rr	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(r)	RR	3	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	rr	2	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	RR	3	30	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)+	rr	3	32	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	(rr)+	RR	3	32	dst<-dst-src rr<-rr+2	^ ^ ^ ^ ? ?
SUBW	NN	rr	4	32	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	N(rrx)	rr	4	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	N(rrx)	RR	4	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN(rrx)	rr	5	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN(rrx)	RR	5	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr(rrx)	rr	3	34	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	rr	#NN	4	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	RR	#NN	4	14	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	#NN	4	32	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN	#NN	6	36	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	N(rrx)	#NN	5	36	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	NN(rrx)	#NN	6	38	dst<-dst-src	^ ^ ^ ^ ? ?
SUBW	(rr)	(rr)	2	32	dst<-dst-src	^ ^ ^ ^ ? ?
SWAP : Swap nibbles						
SWAP	r		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -
SWAP	R		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -
SWAP	(r)		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -
SWAP	(R)		2	8	dst(0-3)<--->dst(4-7)	? ^ ^ ? - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TCM : Test and complement byte under mask						
TCM	r	r	2	6	NOT dst AND src	- ^ ^ 0 - -
TCM	R	R	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	r	R	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	R	r	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	r	(r)	2	6	NOT dst AND src	- ^ ^ 0 - -
TCM	R	(r)	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	r	(rr)	3	12	NOT dst AND src	- ^ ^ 0 - -
TCM	R	(rr)	3	12	NOT dst AND src	- ^ ^ 0 - -
TCM	r	NN	4	18	NOT dst AND src	- ^ ^ 0 - -
TCM	r	N(rrx)	4	24	NOT dst AND src	- ^ ^ 0 - -
TCM	R	N(rrx)	4	24	NOT dst AND src	- ^ ^ 0 - -
TCM	r	NN(rrx)	5	26	NOT dst AND src	- ^ ^ 0 - -
TCM	R	NN(rrx)	5	26	NOT dst AND src	- ^ ^ 0 - -
TCM	r	rr(rrx)	3	22	NOT dst AND src	- ^ ^ 0 - -
TCM	r	(rr)+	3	16	NOT dst AND src rr<-rr+1	- ^ ^ 0 - -
TCM	R	(rr)+	3	16	NOT dst AND src rr<-rr+1	- ^ ^ 0 - -
TCM	r	-(rr)	3	16	rr<-rr-1 NOT dst AND src	- ^ ^ 0 - -
TCM	R	-(rr)	3	16	rr<-rr-1 NOT dst AND src	- ^ ^ 0 - -
TCM	(r)	r	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	(r)	R	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	r	3	18	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	R	3	18	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)+	r	3	22	NOT dst AND src rr<-rr+1	- ^ ^ 0 - -
TCM	(rr)+	R	3	22	dst<-ds AND src rr<-rr+1	- ^ ^ 0 - -
TCM	NN	r	4	20	NOT dst AND src	- ^ ^ 0 - -
TCM	N(rrx)	r	4	26	NOT dst AND src	- ^ ^ 0 - -
TCM	N(rrx)	R	4	26	NOT dst AND src	- ^ ^ 0 - -
TCM	NN(rrx)	r	5	28	NOT dst AND src	- ^ ^ 0 - -
TCM	NN(rrx)	R	5	28	NOT dst AND src	- ^ ^ 0 - -
TCM	rr(rrx)	r	3	24	NOT dst AND src	- ^ ^ 0 - -
TCM	-(rr)	r	3	22	NOT dst AND src rr<-rr-1	- ^ ^ 0 - -
TCM	-(rr)	R	3	22	NOT dst AND src rr<-rr-1	- ^ ^ 0 - -
TCM	r	#N	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	R	#N	3	10	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	#N	3	16	NOT dst AND src	- ^ ^ 0 - -
TCM	NN	#N	5	22	NOT dst AND src	- ^ ^ 0 - -
TCM	(rr)	(rr)	3	18	NOT dst AND src	- ^ ^ 0 - -
TCM	(RR)	(rr)	3	18	NOT dst AND src	- ^ ^ 0 - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TCMW : Test and complement word under mask						
TCMW	rr	rr	2	10	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	RR	3	12	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	RR	3	12	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	rr	3	12	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	(r)	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	(r)	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	(rr)	2	16	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	(rr)	3	18	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	NN	4	22	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	N(rrx)	4	28	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	N(rrx)	4	28	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	NN(rrx)	5	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	NN(rrx)	5	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	rr(rrx)	3	26	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	(rr)+	3	22	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	RR	(rr)+	3	22	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	rr	-(rr)	3	24	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	-(rr)	3	24	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	(r)	rr	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	(r)	RR	3	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	rr	2	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	RR	3	28	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)+	rr	3	30	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	(rr)+	RR	3	30	NOT dst AND src rr<-rr+2	- ^ ^ 0 - -
TCMW	NN	rr	4	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	N(rrx)	rr	4	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	N(rrx)	RR	4	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN(rrx)	rr	5	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN(rrx)	RR	5	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	rr(rrx)	rr	3	32	NOT dst AND src	- ^ ^ 0 - -
TCMW	-(rr)	rr	3	30	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	-(rr)	RR	3	30	rr<-rr-2 NOT dst AND src	- ^ ^ 0 - -
TCMW	rr	#NN	4	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	RR	#NN	4	14	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	#NN	4	30	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN	#NN	6	34	NOT dst AND src	- ^ ^ 0 - -
TCMW	N(rrx)	#NN	5	34	NOT dst AND src	- ^ ^ 0 - -
TCMW	NN(rrx)	#NN	6	36	NOT dst AND src	- ^ ^ 0 - -
TCMW	(rr)	(rr)	2	32	NOT dst AND src	- ^ ^ 0 - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TM : Test byte under mask						
TM	r	r	2	6	dst AND src	- ^ ^ 0 - -
TM	R	R	3	10	dst AND src	- ^ ^ 0 - -
TM	r	R	3	10	dst AND src	- ^ ^ 0 - -
TM	R	r	3	10	dst AND src	- ^ ^ 0 - -
TM	r	(r)	2	6	dst AND src	- ^ ^ 0 - -
TM	R	(r)	3	10	dst AND src	- ^ ^ 0 - -
TM	r	(rr)	3	12	dst AND src	- ^ ^ 0 - -
TM	R	(rr)	3	12	dst AND src	- ^ ^ 0 - -
TM	r	NN	4	18	dst AND src	- ^ ^ 0 - -
TM	r	N(rrx)	4	24	dst AND src	- ^ ^ 0 - -
TM	R	N(rrx)	4	24	dst AND src	- ^ ^ 0 - -
TM	r	NN(rrx)	5	26	dst AND src	- ^ ^ 0 - -
TM	R	NN(rrx)	5	26	dst AND src	- ^ ^ 0 - -
TM	r	rr(rrx)	3	22	dst AND src	- ^ ^ 0 - -
TM	r	(rr)+	3	16	dst AND src rr<-rr+1	- ^ ^ 0 - -
TM	R	(rr)+	3	16	dst AND -src rr<-rr+1	- ^ ^ 0 - -
TM	r	-(rr)	3	16	rr<-rr-1 dst AND src	- ^ ^ 0 - -
TM	R	-(rr)	3	16	rr<-rr-1 dst AND src	- ^ ^ 0 - -
TM	(r)	r	3	10	dst AND src	- ^ ^ 0 - -
TM	(r)	R	3	10	dst AND src	- ^ ^ 0 - -
TM	(rr)	r	3	18	dst AND src	- ^ ^ 0 - -
TM	(rr)	R	3	18	dst AND src	- ^ ^ 0 - -
TM	(rr)+	r	3	22	dst AND src rr<-rr+1	- ^ ^ 0 - -
TM	(rr)+	R	3	22	dst AND src rr<-rr+1	- ^ ^ 0 - -
TM	NN	r	4	20	dst AND src	- ^ ^ 0 - -
TM	N(rrx)	r	4	26	dst AND src	- ^ ^ 0 - -
TM	N(rrx)	R	4	26	dst AND src	- ^ ^ 0 - -
TM	NN(rrx)	r	5	28	dst AND src	- ^ ^ 0 - -
TM	NN(rrx)	R	5	28	dst AND src	- ^ ^ 0 - -
TM	rr(rrx)	r	3	24	dst AND src	- ^ ^ 0 - -
TM	-(rr)	r	3	22	rr->rr-1 dst AND src	- ^ ^ 0 - -
TM	-(rr)	R	3	22	rr->rr-1 dst AND src	- ^ ^ 0 - -
TM	r	#N	3	10	dst AND src	- ^ ^ 0 - -
TM	R	#N	3	10	dst AND src	- ^ ^ 0 - -
TM	(rr)	#N	3	16	dst AND src	- ^ ^ 0 - -
TM	NN	#N	5	22	dst AND src	- ^ ^ 0 - -
TM	(rr)	(rr)	3	18	dst AND src	- ^ ^ 0 - -
TM	(RR)	(rr)	3	18	dst AND src	- ^ ^ 0 - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
TMW : Test word under mask						
TMW	rr	rr	2	10	dst AND src	- ^ ^ 0 - -
TMW	RR	RR	3	12	dst AND src	- ^ ^ 0 - -
TMW	rr	RR	3	12	dst AND src	- ^ ^ 0 - -
TMW	RR	rr	3	12	dst AND src	- ^ ^ 0 - -
TMW	rr	(r)	3	14	dst AND src	- ^ ^ 0 - -
TMW	RR	(r)	3	14	dst AND src	- ^ ^ 0 - -
TMW	rr	(rr)	2	16	dst AND src	- ^ ^ 0 - -
TMW	RR	(rr)	3	18	dst AND src	- ^ ^ 0 - -
TMW	rr	NN	4	22	dst AND src	- ^ ^ 0 - -
TMW	rr	N(rrx)	4	28	dst AND src	- ^ ^ 0 - -
TMW	RR	N(rrx)	4	28	dst AND src	- ^ ^ 0 - -
TMW	rr	NN(rrx)	5	30	dst AND src	- ^ ^ 0 - -
TMW	RR	NN(rrx)	5	30	dst AND src	- ^ ^ 0 - -
TMW	rr	rr(rrx)	3	26	dst AND src	- ^ ^ 0 - -
TMW	rr	(rr)+	3	22	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	RR	(rr)+	3	22	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	rr	-(rr)	3	24	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	RR	-(rr)	3	24	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	(r)	rr	3	14	dst AND src	- ^ ^ 0 - -
TMW	(r)	RR	3	14	dst AND src	- ^ ^ 0 - -
TMW	(rr)	rr	2	28	dst AND src	- ^ ^ 0 - -
TMW	(rr)	RR	3	28	dst AND src	- ^ ^ 0 - -
TMW	(rr)+	rr	3	30	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	(rr)+	RR	3	30	dst AND src rr<-rr+2	- ^ ^ 0 - -
TMW	NN	rr	4	30	dst AND src	- ^ ^ 0 - -
TMW	N(rrx)	rr	4	36	dst AND src	- ^ ^ 0 - -
TMW	N(rrx)	RR	4	36	dst AND src	- ^ ^ 0 - -
TMW	NN(rrx)	rr	5	36	dst AND src	- ^ ^ 0 - -
TMW	NN(rrx)	RR	5	36	dst AND src	- ^ ^ 0 - -
TMW	rr(rrx)	rr	3	32	dst AND src	- ^ ^ 0 - -
TMW	-(rr)	rr	3	30	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	-(rr)	RR	3	30	rr<-rr-2 dst AND src	- ^ ^ 0 - -
TMW	rr	#NN	4	14	dst AND src	- ^ ^ 0 - -
TMW	RR	#NN	4	14	dst AND src	- ^ ^ 0 - -
TMW	(rr)	#NN	4	30	dst AND src	- ^ ^ 0 - -
TMW	NN	#NN	6	34	dst AND src	- ^ ^ 0 - -
TMW	N(rrx)	#NN	5	34	dst AND src	- ^ ^ 0 - -
TMW	NN(rrx)	#NN	6	36	dst AND src	- ^ ^ 0 - -
TMW	(rr)	(rr)	2	32	dst AND src	- ^ ^ 0 - -
WFI : Wait for Interrupt						
WFI			2	18	wait for interrupt	- - - - - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
XCH : Exchange Register						
XCH	r	r	3	12	dst <-> src	- - - - - -
XCH	R	R	3	12	dst <-> src	- - - - - -
XCH	r	r	3	12	dst <-> src	- - - - - -
XCH	R	R	3	12	dst <-> src	- - - - - -
XOR : Logical exclusive OR						
XOR	r	r	2	6	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	R	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	R	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	r	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	(r)	2	6	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	(r)	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	(rr)	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	(rr)	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	NN	4	18	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	N(rrx)	4	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	N(rrx)	4	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	NN(rrx)	5	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	NN(rrx)	5	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	rr(rrx)	3	22	dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	(rr)+	3	16	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	R	(rr)+	3	16	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	r	-(rr)	3	16	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	-(rr)	3	16	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	(r)	r	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(r)	R	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	r	3	18	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	R	3	18	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)+	r	3	22	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	(rr)+	R	3	22	dst<-dst XOR src rr->rr+1	- ^ ^ 0 - -
XOR	NN	r	4	20	dst<-dst XOR src	- ^ ^ 0 - -
XOR	N(rrx)	r	4	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	N(rrx)	R	4	26	dst<-dst XOR src	- ^ ^ 0 - -
XOR	NN(rrx)	r	5	28	dst<-dst XOR src	- ^ ^ 0 - -
XOR	NN(rrx)	R	5	28	dst<-dst XOR src	- ^ ^ 0 - -
XOR	rr(rrx)	r	3	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	-(rr)	r	3	22	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	-(rr)	R	3	22	rr->rr-1 dst<-dst XOR src	- ^ ^ 0 - -
XOR	r	#N	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	R	#N	3	10	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	#N	3	16	dst<-dst XOR src	- ^ ^ 0 - -
XOR	NN	#N	5	24	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(rr)	(rr)	3	20	dst<-dst XOR src	- ^ ^ 0 - -
XOR	(RR)	(rr)	3	20	dst<-dst XOR src	- ^ ^ 0 - -

Mnemo.	dst	src	Bytes	Clock cycles	Operation	Flags C Z S V D H
XORW : Logical exclusive OR between words						
XORW	rr	rr	2	10	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	RR	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	RR	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	rr	3	12	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	(r)	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	(r)	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	(rr)	2	16	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	(rr)	3	18	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	NN	4	22	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	N(rrx)	4	28	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	N(rrx)	4	28	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	NN(rrx)	5	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	NN(rrx)	5	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	rr(rrx)	3	26	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	(rr)+	3	22	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	RR	(rr)+	3	22	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	rr	-(rr)	3	24	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	-(rr)	3	24	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	(r)	rr	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(r)	RR	3	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	rr	2	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	RR	3	30	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)+	rr	3	32	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	(rr)+	RR	3	32	dst<-dst XOR src rr<-rr+2	- ^ ^ 0 - -
XORW	NN	rr	4	32	dst<-dst XOR src	- ^ ^ 0 - -
XORW	N(rrx)	rr	4	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	N(rrx)	RR	4	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN(rrx)	rr	5	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN(rrx)	RR	5	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr(rrx)	rr	3	34	dst<-dst XOR src	- ^ ^ 0 - -
XORW	-(rr)	rr	3	32	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	-(rr)	RR	3	32	rr<-rr-2 dst<-dst XOR src	- ^ ^ 0 - -
XORW	rr	#NN	4	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	RR	#NN	4	14	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	#NN	4	32	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN	#NN	6	36	dst<-dst XOR src	- ^ ^ 0 - -
XORW	N(rrx)	#NN	5	36	dst<-dst XOR src	- ^ ^ 0 - -
XORW	NN(rrx)	#NN	6	38	dst<-dst XOR src	- ^ ^ 0 - -
XORW	(rr)	(rr)	2	32	dst<-dst XOR src	- ^ ^ 0 - -

CHAPTER 4

INTERRUPT AND DMA

4.1 INTRODUCTION

The ST9 responds to peripheral events and external events through its Interrupt and on-chip DMA channels. When such an event occurs, if previously enabled and according to a priority mechanism, the current program execution can be suspended to allow the ST9 to execute a specific response routine. If the event generates an interrupt request, the current program status is saved after the current instruction is completed and the CPU control passes to the Interrupt Service Routine. Similarly, if the event occurrence requires a DMA transaction, this will take place at the end of the current instruction execution.

The ST9 CPU can receive requests from the following type of sources

- On-chip peripherals
- External pins
- Top-Level Pseudo-non-maskable interrupt

According to the on-chip peripheral features, an event occurrence can generate an Interrupt request or a DMA transaction, depending on the selected mode.

Up to eight external interrupt channels, with programmable input trigger edge, are available. In addition, a dedicated interrupt channel, set to the Top-level priority, can be devoted either to the external pin NMI (to provide a Non-Maskable-Interrupt) or to the Timer/Watchdog. Interrupt service routines are addressed through a vector table mapped in Program Memory.

DMA transactions provide high speed data transmission from/to the peripheral data register to/from Register File or Data Memory or Program Memory. DMA channels support up to 64k bytes block transfer with Memory spaces. When DMA operates with the Register File, the data block dimension cannot be larger than 222 bytes. A special DMA mode called Swap Mode is implemented to support real time applications.

4.2 INTERRUPT VECTORIZATION

The ST9 implements an interrupt vectoring structure that allows the on-chip peripheral to identify the location of the first instruction of the Interrupt Service Routine (IVR) automatically.

When the interrupt request is acknowledged, the peripheral interrupt module provides, through its Interrupt Vector Register (IVR), a vector to point into the vector table of locations containing the start addresses of the Interrupt Service Routines (defined by the programmer).

Each peripheral has a specific IVR mapped within its Register File pages.

The Interrupt Vector table, containing the list of the addresses of the Interrupt Service Routines, is located in the first 256 locations of the Program Memory. The first 6 locations of the Program Memory are reserved for:

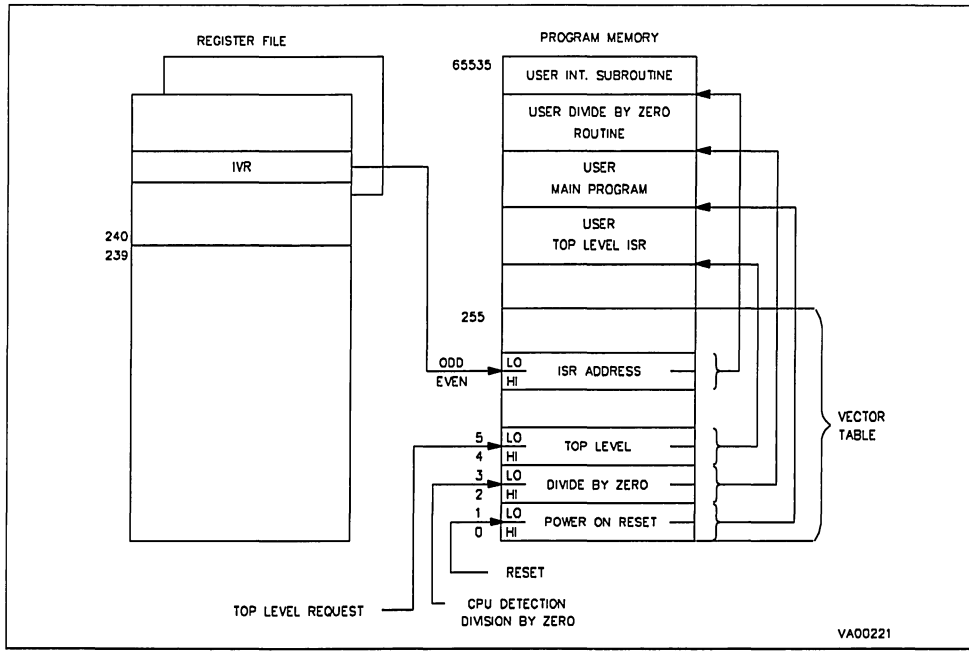
Address Content

0	Address high of Power on Reset routine
1	Address low of Power on Reset routine
2	Address high of Divide by zero trap routine
3	Address low of Divide by zero trap routine
4	Address high of Top Level Interrupt routine
5	Address low of Top Level Interrupt routine

With one Interrupt Vector register, it is possible to address more interrupt service routines; in fact, several peripherals share the same interrupt vector register among several interrupt channels. The most significant bits of the vector are user programmable to define the base vector address inside the vector table in the program memory, the least significant bits are controlled by the interrupt module in hardware to select the specific vector.

Note: The first 256 locations of the program memory can contain program code. Other than the Reset vector, they are not exclusively reserved to the vector table.

Figure 4-1. Vectors and Associated Routines



4.3 INTERRUPT AND DMA PRIORITY LEVEL ARCHITECTURE

The ST9 supports a fully programmable interrupt and DMA priority structure. Figure 4.2 shows a conceptual description.

9 priority levels are available to define the channel priority relationship. Each channel has a 3 bit field, PRL (Priority Level), that defines its priority level among 8 programmable levels. The ninth level (Top Level Priority) is reserved for the Timer/Watch-Dog or the External Pseudo Non-Maskable Interrupt. The On-chip peripheral channel and the eight external interrupt sources can be programmed within eight priority levels: level 7 has the lowest priority, level 0 has the highest priority.

If several units are located at the same priority level, an internal daisy chain, fixed for each ST9 device, defines the priority relationship within that level.

The PRL bits are used to define the priority level both for interrupt requests and for DMA transactions. If both interrupt and DMA requests are generated at the same priority level, the DMA request will be acknowledged first.

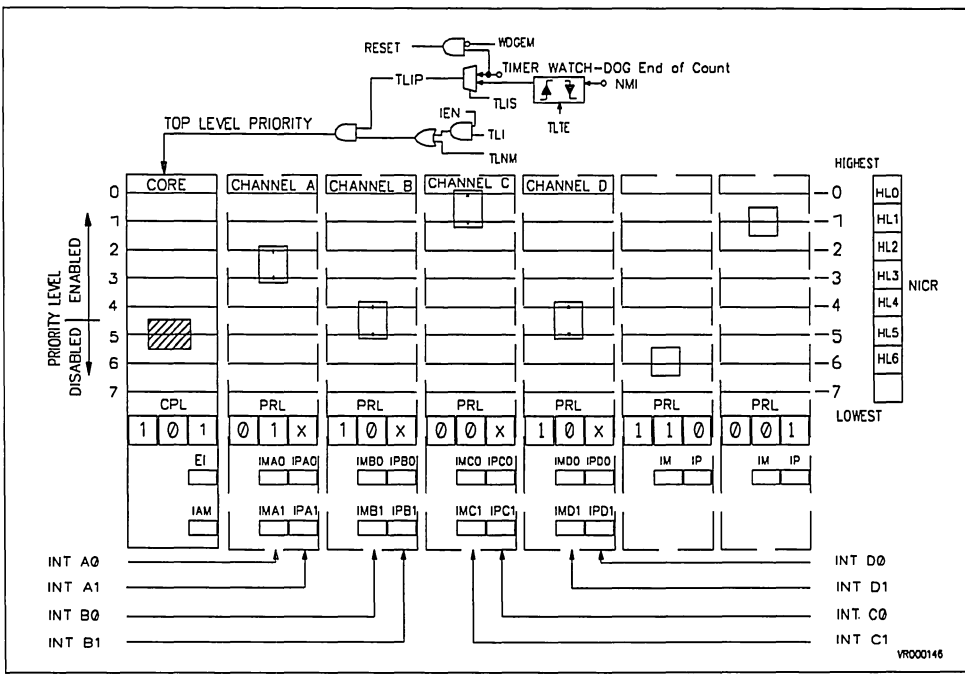
Top level priority interrupt (highest) can be assigned either to the external Pseudo Non-Maskable interrupt or to the internal Timer/Watch-Dog. An Interrupt service routine at this level cannot be interrupted in any arbitration mode. Its mask can be both maskable (TLI) or non-maskable (TLNM).

4.4 PRIORITY LEVEL ARBITRATION

The 3 bits of CPL (Current Priority Level) in the Central Interrupt Control Register contain the priority of the currently running program (CPU priority). CPL is set to 7 (lowest priority) upon reset and can be modified during program execution either by software or automatically by hardware according to the selected Arbitration Mode.

During every instruction an arbitration phase is made between every channel capable of generating an Interrupt or DMA request, each priority level is compared to all the other requests. If the highest priority request is an interrupt, it must be *higher* than the CPL value in order to be acknowledged. If the highest priority request is a DMA transaction request, it must be *equal to or higher than* the CPL value in order to be executed.

Figure 4-2. ST9 Interrupt Architecture



The priority of the Top Level Interrupt overrides every other priority.

If two or more requests occur at the same instant of time and at the same priority level, an on-chip daisy chain, specific to every ST9 version, selects the channel with the highest position in the chain. The position in the chain for the ST9030, ST9040, ST9020 families is shown in table 4.1.

Table 4-1. Daisy Chain Priorities

	ST9030 ST9040	ST9020
Highest Position	INTA0	INTA0
	INTA1	INTA1
	INTB0	INTB0
	INTB1	INTB1
	INTC0	INTC0
	INTC1	INTC1
	INTD0	INTD0
	INTD1	INTD1
	TIMER0	SCI
	SCI	TIMER
	A/D	
Lowest Position	TIMER1	

The 8 priority levels used for interrupts are also used to prioritize the DMA requests, which are arbitrated in the same arbitration phase as interrupt requests. When an interrupt and a DMA request occur simultaneously, on the same priority level, the DMA request is serviced before the interrupt.

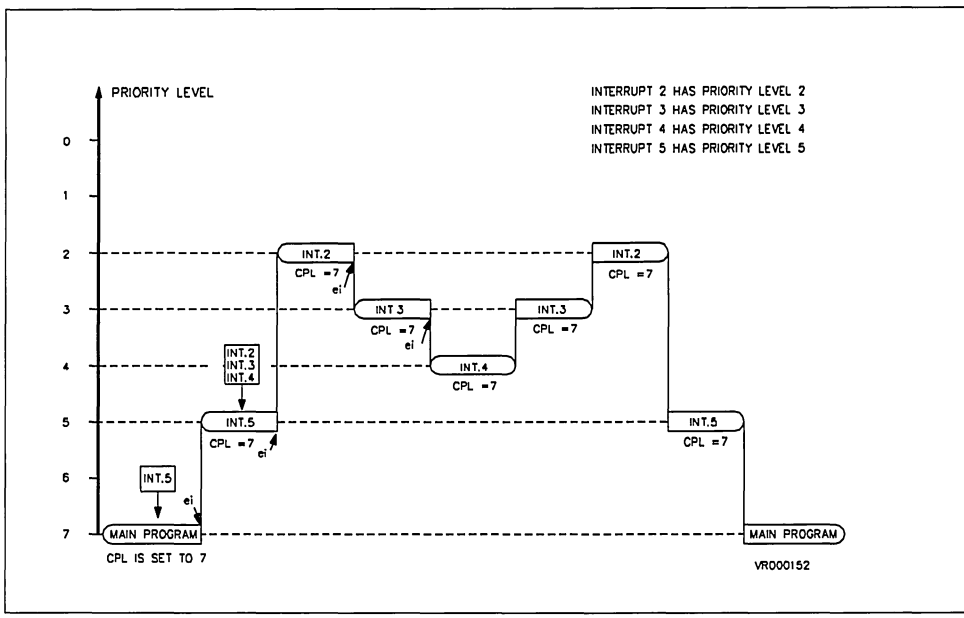
ST9 provides two interrupt arbitration modes: Concurrent and Nested modes. The Concurrent mode is the standard interrupt arbitration mode while the Nested mode improves the effective interrupt response time when a nesting of the service routines is required according to the request priority levels. The control bit IAM (CICR.3) selects the Concurrent Arbitration mode (when reset to "0") or the Nested Arbitration Mode (when set to "1").

4.4.1 Concurrent Mode

This mode is selected when the IAM bit is cleared (reset condition). The arbitration phase, performed during every instruction, selects the request with the highest priority level.

If the highest priority request is an interrupt request and its priority value is higher than the Current Priority Value CICR.2,1,0 (R230.2,1,0), the interrupt request will be acknowledged at the end of the

Figure 4-3. Example of a Sequence of Interrupt Requests with :
- Concurrent mode
- EI set to 1 during the interrupt routine execution



current instruction. The interrupt Machine Cycle performs the following steps:

- 1. Disables all the maskable interrupt requests by clearing CICR.IEN
- 2. Pushes the PC low byte into the system stack
- 3. Pushes the PC high byte into the system stack
- 4. Pushes the Flag register into the system stack
- 5. Loads the PC with the 16-bit vector stored in the Vector Table, pointed to by the Interrupt Vector Register (IVR).

The Interrupt Service Routine must be concluded with the `IRET` instruction. The `IRET` instruction executes the following operations:

- 1. Pops off the Flag register from the system Stack
- 2. Pops off PC high byte from the system Stack
- 3. Pops off PC low byte from the system Stack
- 4. Enables all the un-masked Interrupts, by setting the CICR.IEN bit

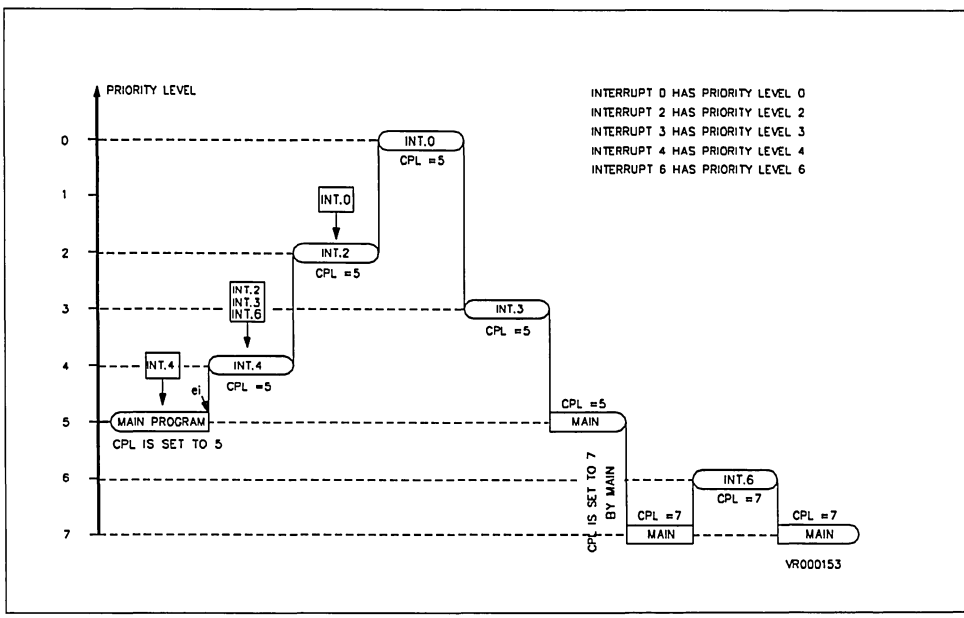
The suspended program execution is thus recovered at the interrupted instruction. All pending interrupts existing, or having occurred during the interrupt service routine execution, remain pending until the Enable Interrupt instruction (even if it is executed during the interrupt service routine).

NOTE: When Concurrent mode is selected, the source priority level is meaningful only during the arbitration phase, where it is compared to all the other priority levels and the CPL, but no trace is kept of its value during the Interrupt Service Routine. Therefore, if other requests are issued, once the global CICR.IEN is enabled again, they will be acknowledged regardless of the Interrupt Service Routine priority value; if no care is taken by the programmer, unpleasant side effects can take place.

A typical case is the following: 3 pending requests with different priority levels (ie 2,3,4) generate requests at the same time (because the associated events occurred during the same instruction). The three interrupt service routines set Interrupt Enable (IEN, CICR.4) by the EI instruction at the beginning of the routine to avoid a high interrupt response time to requests with a priority higher than the one under service (usually, the higher the priority, the sooner the routine must be executed). Unfortunately, what will happen in this case is that the three interrupt servicing routines will be executed exactly in the opposite order of their priority. Interrupt routine level 2 will be acknowledged first, then, when the EI instruction is executed, it will be interrupted by interrupt routine level 3, which itself will be interrupted by interrupt routine level 4. When interrupt routine level 4 is completed, interrupt routine level 3 will be recovered and finally, interrupt routine level 2.

Therefore, **it is recommended, in concurrent mode, to avoid the insertion of the EI instruction in the interrupt subroutine**, which can trigger this LIFO (Last In, First Out) sequence of interrupt processing.

Figure 4-4. Example of a Sequence of Interrupt Requests with :
- Concurrent mode
- EI unchanged by the interrupt routines



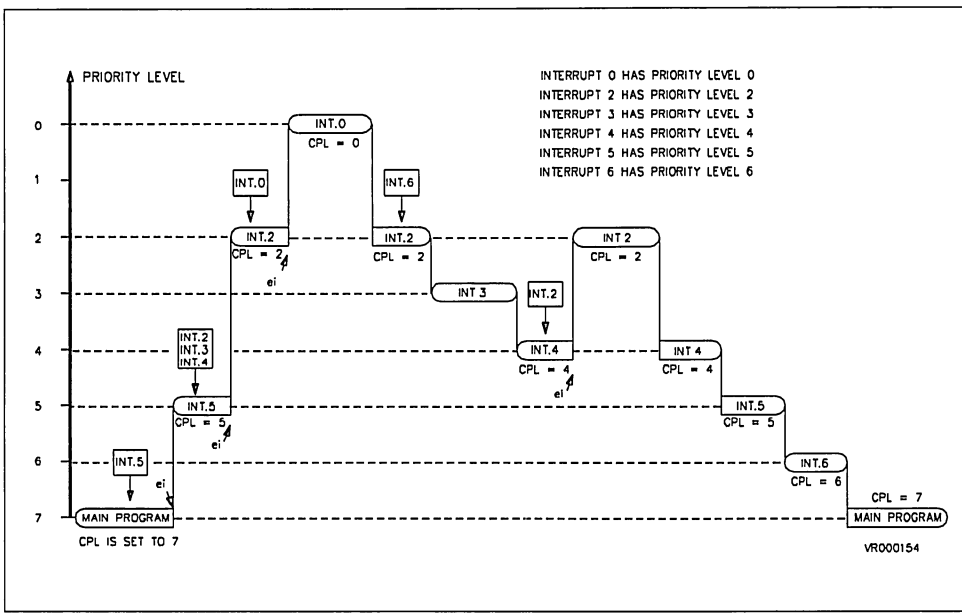
4.4.2 Nested Mode

The difference of the Nested mode to the Concurrent mode consists of the modification of the CPL value during the interrupt processing. The arbitration phase is basically identical to the concurrent Mode, however once the request is acknowledged, the current CPL value is saved in the Nested Interrupt Control Register (NICR, R247 page 0) by setting the NICR bit corresponding to the CPL value (i.e. if the CPL is 3, NICR.3 bit will be set). The CPL value is then updated with the Priority value of the request just acknowledged, in this way the next arbitration cycle will be performed against the priority level of the Service Routine in progress. The above procedure is done only when the request is an Interrupt request, if it is a DMA request it is not necessary to modify the CPL value as the DMA transaction is non-interruptable.

The Interrupt Machine Cycle will perform the following steps:

- Disable all the maskable interrupts by clearing IEN
- Push the CPL value into the special stack NICR to hold the priority level of the suspended routine
- Store in CPL the priority level of the acknowledged routine, so that the next request priority will be compared with the one of the routine under service
- Push the PC-low byte into the System Stack
- Push the PC-high byte into the System Stack
- Push the Flag Register into the System Stack
- Load the PC with the vector pointed by IVR.

Figure 4-5. Example of a Sequence of Interrupt Requests with :
- Nested mode
- EI set to 1 during the interrupt routines execution



The IRET Interrupt Return instruction executes the following steps:

- 1. Pop off the Flag Register from the System Stack
- 2. Pop off the PC-high byte from the System Stack
- 3. Pop off the PC-low byte from the System Stack
- 4. Enable all the unmasked interrupts by setting the IEN bit
- 5. Recover the interrupted routine priority level by popping the value from the special register (NICR) and by copying it into CPL.

The suspended execution is thus recovered at the interrupted instruction.

REMARKS

1) Dynamic priority level modification: the main program and routines can be specifically prioritized. Since CPL is represented by 3 bits in a read/write register, it is possible to modify dynamically the current priority value during the program execution. This means that a critical section can have a higher priority with respect to other interrupt and DMA requests. Furthermore it is possible to prioritize even the Main Program execution by modifying CPL during its execution.

2) Maximum number of nestings: No more than 8 routines can be nested. If an interrupt routine at level N is being serviced, no other Interrupts located at level N can interrupt it. This guarantees a maximum number of 8 nested levels including the Top Level Interrupt request.

3) Priority level 7: Interrupt requests at level 7 cannot be acknowledged as their priority cannot be higher than the CPL value. This can be of use in a fully polled interrupt environment. Only DMA transaction requests at this level can be acknowledged.

A nested/concurrent mode sequence is given on Figure 4.7. This example clearly shows that Nested and Concurrent modes are defined by the user. Note that here the Y axis is referenced by CPL, instead of the source priority level, and *that Interrupt 1 stays pending*, having a priority level lower than CPL.

Figure 4-6. Example of a Sequence of Interrupt Requests with :
- Nested mode
- EI unchanged by the interrupt routines execution

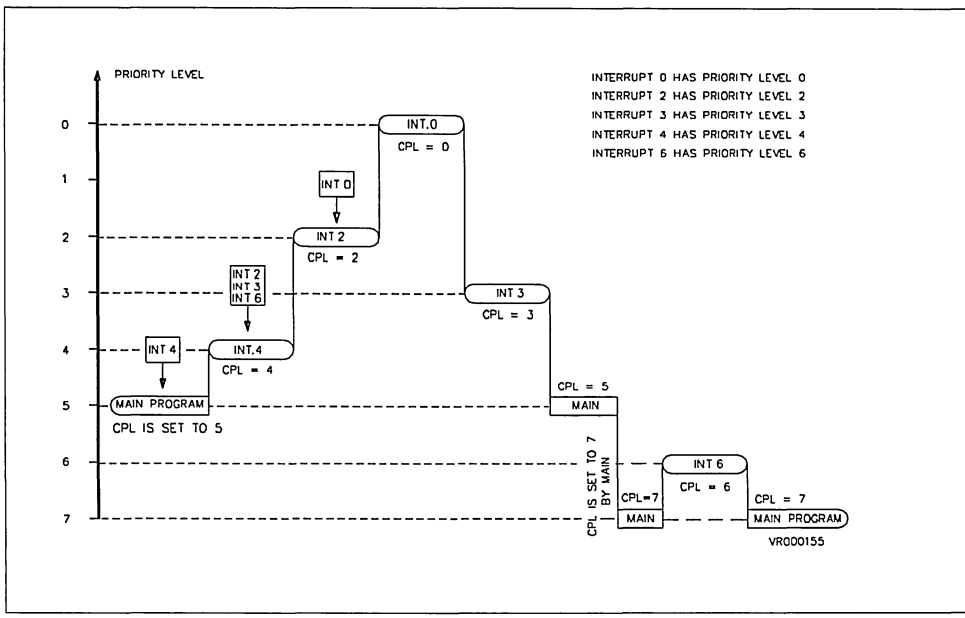


Figure 4-7. Example of a Nested and Concurrent Mode Sequence

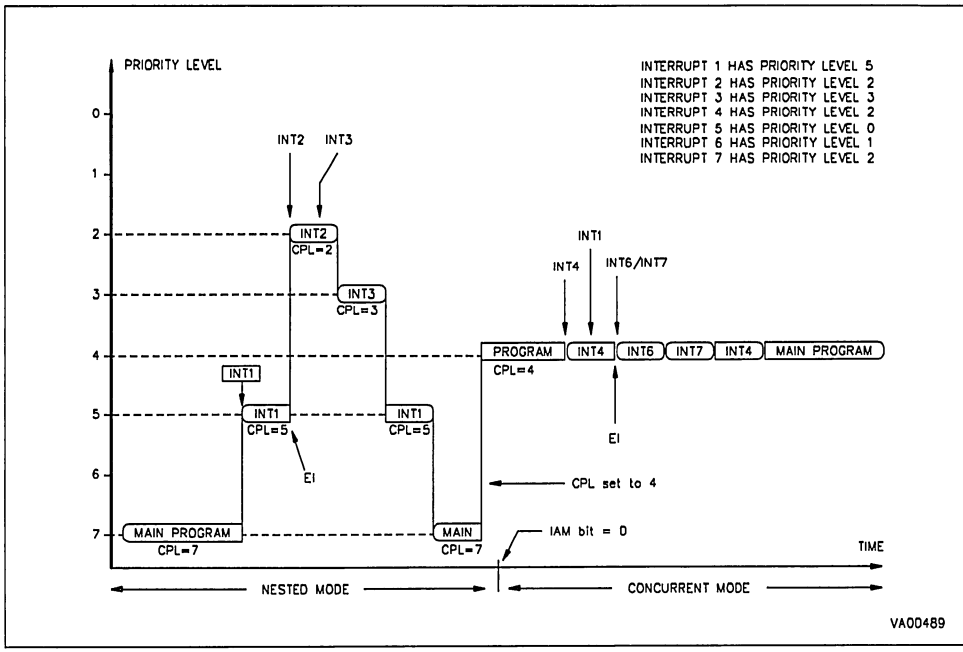
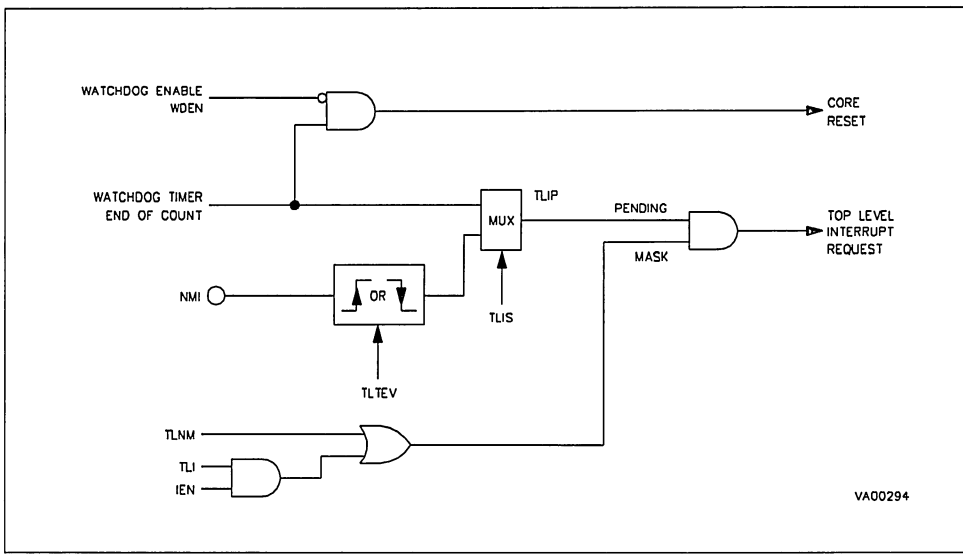


Figure 4-8. Top Level Interrupt Structure



4.5 EXTERNAL INTERRUPTS

The standard ST9 core contains 8 external interrupts sources grouped into four pairs.

Table 4-2.

Pair	Sources
A	INTA0, INTA1
B	INTB0, INTB1
C	INTC0, INTC1
D	INTD0, INTD1

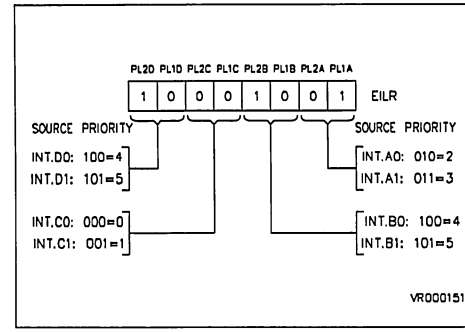
Each source has a trigger control bit TEA0,..TED1 (R242,EITR.0,..,7 Page 0) to select triggering on the rising or falling edge of the external pin. If the Trigger control bit is set to "1", the corresponding pending bit IPA0,..,IPD1 (R243,EIPR.0,..,7 Page 0) is set on the input pin rising edge, if it is cleared, the pending bit is set on the falling edge of the input pin. Each source can be individually masked through the corresponding control bit IMA0,..,IMD1 (EIMR.7,..,0). See Figure 4.9.

The priority level of the external interrupt sources can be programmed among the eight priority levels with the control register EIPLR (R245). The priority level of each pair is software defined using the bits PRL2,PRL1. For each pair, the even channel (A0,B0,C0,D0) of the group has the even priority level and the odd channel (A1,B1,C1,D1) has the odd (lower) priority level. Table 4.3 shows an example of priority levels.

- The source of the interrupt channel A0 can be selected between the external pin INT0 (when IA0S = "1", the reset value) or the On-chip Timer/Watchdog peripheral (when IA0S = "0").
- The source of the interrupt channel B0 can be selected between the external pin INT2 (when (SPEN,BMS)=(0,0)) or the on-chip SPI peripheral.
- The source of the interrupt channel C0 can be selected between the external pin INT4 (when EEIEN = "0") and the on-chip EEPROM write completion interrupt (when EEIEN="1") for ST9 devices with on-chip EEPROM (eg the ST9040).

All other interrupt channels have an input pin as source. According to specific ST9 version, however, the input line may be multiplexed with an on-chip peripheral I/O or connected to an input pin that performs also other function (as in the case of the handshake feature).

Table 4-3. Priority Level Examples



4.6 TOP LEVEL INTERRUPT

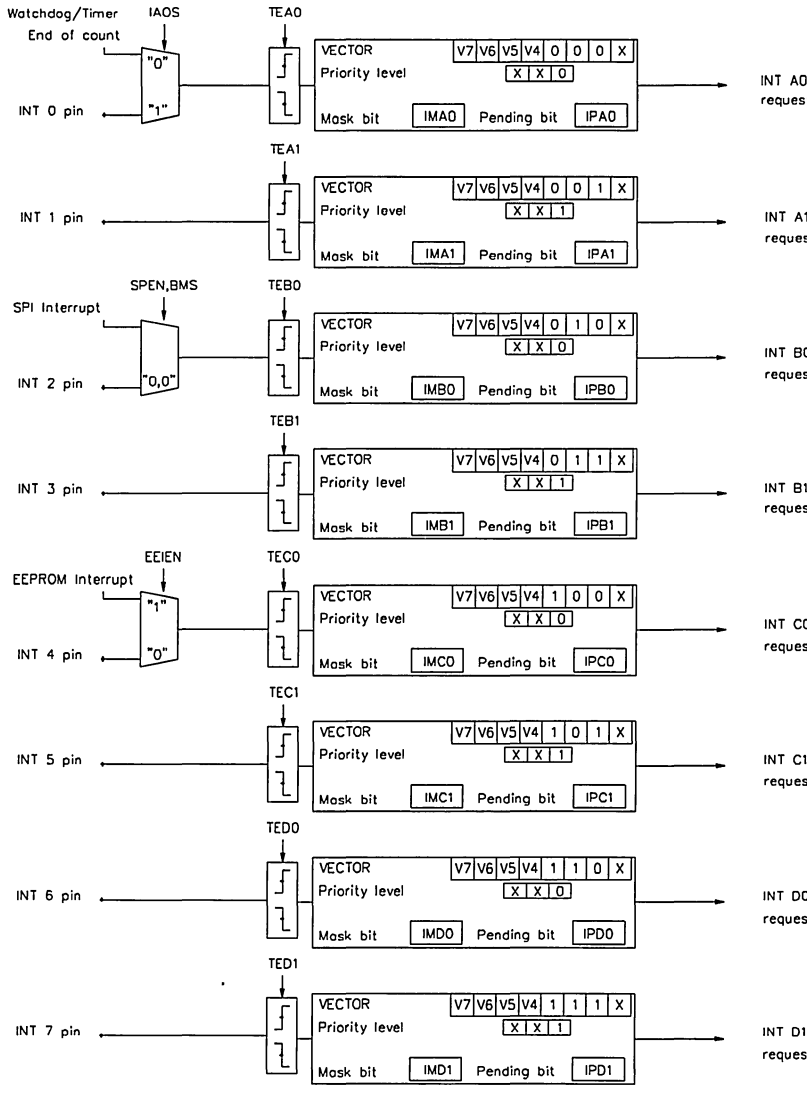
The Top Level Interrupt channel can be assigned either to the external pin NMI or to the Timer/Watchdog according to the status of the control bit EIVR.TLIS (R246.2, Page 0). If this bit is high (the reset condition) the source is the external pin NMI, if it is low, the source is the Timer/ Watchdog End Of Count. When the source is the NMI external pin, the control bit EIVR.TLTEV (R246.3; Page 0) selects between the rising (if set) or falling (if cleared) edge generating the interrupt request. When the selected event occurs, the CICR.TLIP bit (R230.6) is set. Depending on the mask situation, a Top Level Interrupt request may be generated. Two kinds of masks are available, a Maskable mask and a Non-Maskable mask. The first mask is the bit CICR.TLI (R230.5): it can be set or cleared to enable or disable respectively the Top Level Interrupt request. If it is enabled, the global Enable Interrupt bit CICR.IEN (R230.4) must also be enabled in order to allow a Top Level Request.

The second mask NICR.TLNM (R247.7) is a set-only mask. Once set, it enables the Top Level Interrupt request independent of the value of CICR.IEN and it cannot be cleared by program. Only the processor RESET cycle can clear this bit.

The Top Level Interrupt Service Routine cannot be interrupted by any other interrupt request, in any arbitration mode, even by another Top Level Interrupt request.

The interrupt machine cycle of the Top Level Interrupt does not clear the CICR.IEN bit, and the corresponding `IRET` does not set it.

Figure 4-9. External Interrupts Control Bits and Vectors



4.7 ON-CHIP PERIPHERAL INTERRUPTS

The general structure of the peripheral interrupt unit is described here, however each on-chip peripheral has its own specific interrupt unit containing one or more interrupt channels, or interrupt and DMA channels. Please refer to the specific peripheral chapter for the description of its interrupt features and control registers.

The on-chip peripheral interrupt channels provide the following control bits:

- Interrupt Pending bit (IP)
  Set by hardware when the Trigger Event occurs. Can be set/cleared by software to generate/cancel pending interrupts and give the status for Interrupt polling.
- Interrupt Mask bit (IM)
  If IM = "0", no interrupt request is generated. If IM ="1" an interrupt request is generated whenever IP = "1" and CICR.IEN = "1".
- Priority Level (PRL, 3 bits)
  These bits define the source priority level
  PRL=0: the highest priority
  PRL=7: the lowest priority (the interrupt cannot be acknowledged)
- Interrupt Vector Register (IVR, up to 7 bits)
  The IVR points to the vector table which itself contains the interrupt routine start address.

4.8 ST9 ON-CHIP DMA

4.8.1 Introduction

ST9 has on chip Direct Memory Access (DMA) channels to provide high-speed data transaction between peripherals and memory or Register File. Multi-channel DMA is fully supported as each peripheral can have its own DMA channel(s). Each DMA channel transfers data to or from contiguous locations of the Register File, Program Memory or Data Memory. The maximum number of transactions that each DMA channel can perform is 222 if the Register File is selected, or 65536 if Program or Data Memory is selected.

DMA transfer to (or from) the Register File takes 8 CPUCLK cycles; DMA transfer to (or from) Memory takes 16 CPUCLK cycles. If the ST9 is in the idle mode (during a Wait For Interrupt instruction execution), DMA requests are acknowledged and the time required for Register File transfers becomes 10 CPUCLK cycles and for Memory 18 CPUCLK cycles.

Each DMA channel has its own control registers located in the I/O page(s) related to the peripheral.

4.8.2 DMA Transactions

The purpose of on-chip DMA channel is to transfer a block of data from/to the peripheral to/from Register File or Memory. Each DMA transfer consists of three operations:

- A load from/to the peripheral data register to a location of Register File (or Memory) addressed through the DMA Address Register (or Register pair)
- A post-increment of the DMA Address Register (or Register pair)
- A post-decrement of the DMA transaction counter, which contains the number of transactions that have still to be performed.

If the DMA transaction is made between **the peripheral and the Register File** (figure 4.10), one register is required to hold the DMA Address and one to hold the DMA transaction counter. These two registers must be located in the Register File: the DMA Address Register in the even addressed register, the DMA transaction Counter in the following register (odd address). They are pointed to by the DMA Transaction Counter Pointer Register (DCPR) located in the page registers of the peripheral. In order to select the DMA transaction with the Register File, the control bit DCPR.RM (bit 0 of DCPR) must be set.

Figure 4-10. DMA Between Registers and Peripheral

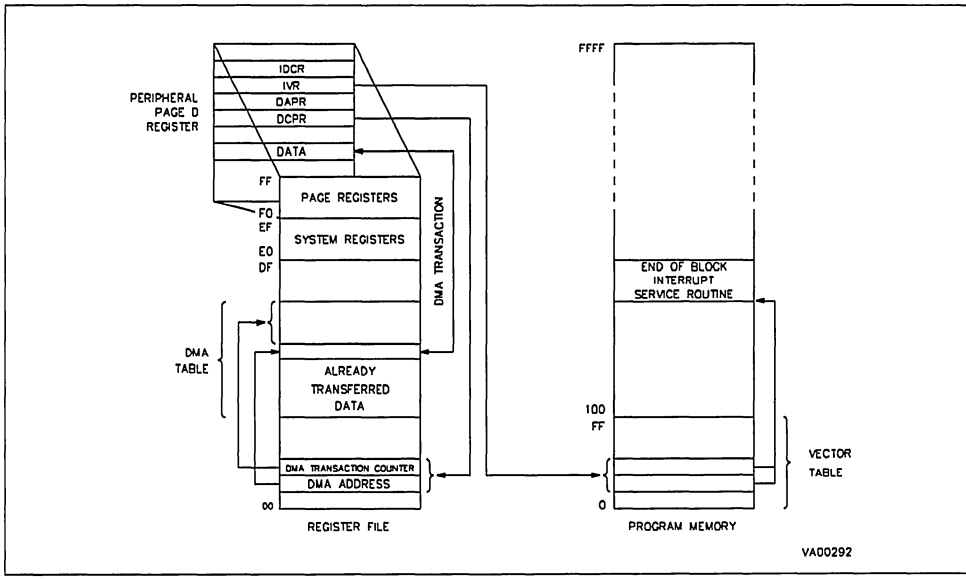
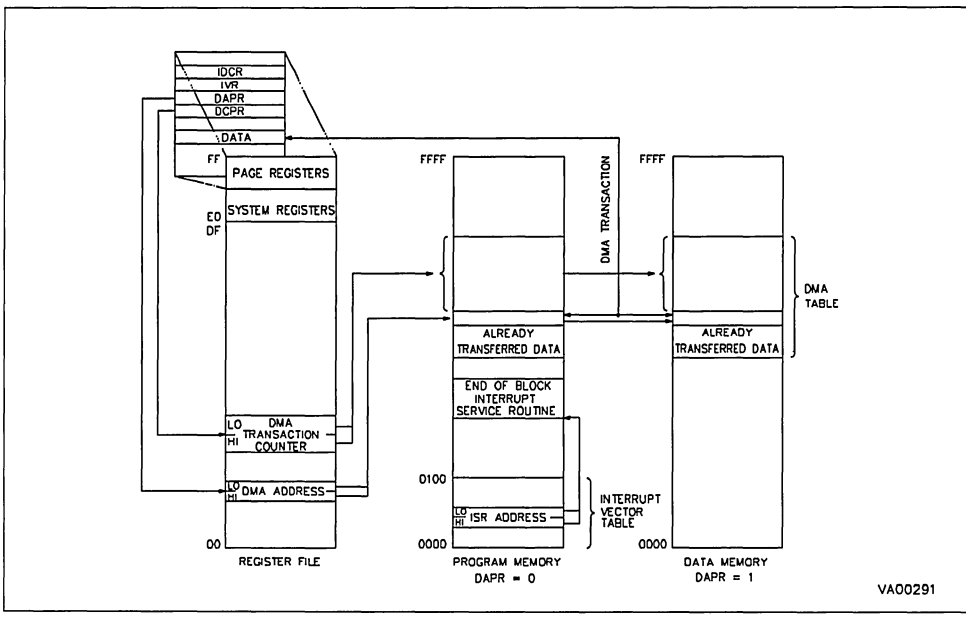


Figure 4-11. DMA Between Memory and Peripheral



The Transaction Counter Register must be initialized with the number of DMA transfers to perform and will be decremented after each transaction. The DMA Address Register must be initialized with the starting address of the DMA table in the Register File, and is increased after each transaction. These two registers must be located between addresses 00h and DFh of the Register File.

If the transaction is made between **the peripheral and Memory Space (Program or Data Memory)**, a register pair (16 bits) is required for the DMA Address and for the DMA transaction Counter (figure 4.11). Thus, two register pairs must be located in the Register File. The DMA Transaction Counter is pointed to by the DMA Transaction Counter Pointer Register (DCPR), the DMA Address is pointed to by the DMA Address Pointer Register (DAPR), both DCPR and DAPR are located in the page registers of the peripheral. When selecting the DMA transaction with memory, the control bit DCPR.RM (bit 0 of DCPR) must be cleared to "0".

To select between Program or Data Memory, the control bit DAPR.DP (bit 0 of DAPR) must be cleared or set respectively.

Once the DMA table is completed (the transaction counter reaches 0 value), an Interrupt request to the CPU is generated.

The DMA transaction Counter must be initialized with the number of transactions to perform and will be decremented after each transaction. The DMA Address must be initialized with the starting address of the DMA table and is increased after each transaction. These two register pairs must be located betweeen addresses 00h and DFh of the Register File.

Once a DMA channel is initialized, a transfer can start. The direction of the transfer (data from/to peripheral to/from memory or Register File) is automatically defined by the type of peripheral involved.

When the Request Pending bit is set by a hardware event (or by software), and the DMA Mask bit is set, a DMA request is generated. If the Priority Level of the DMA source is higher than or equal to the Priority Level under service (CPL) the DMA transfer is executed at the end of the current instruction. DMA transfer reads/writes data from/to the location pointed by the DMA Address Register, increments the DMA Address register and decrements the Transaction Counter Register. When the content of the Transaction Counter is decremented to zero, the DMA Mask bit (DM) is cleared and an interrupt request is generated according to the Interrupt Mask bit (End of Block interrupt). This End-of-Block interrupt request is taken into account depending on the PRL value.

4.8.3 The Swap-Mode

An extra feature of ST9 DMA channels of some peripherals (i.e the Multi Function TIMER) is the SWAP mode. This feature allows transaction from two DMA tables alternatively. All the DMA descriptors in the Register File are thus doubled. Two DMA transaction counters and two DMA address pointers allow the definition of two fully independent tables (they only have to belong the same space, Register file or Data memory or Program memory). The DMA transaction is programmed to start on one of the two tables (say table 0) and, at the end of block, the DMA controller automatically swaps to the other table (table 1) by pointing to the other DMA descriptors. In this case, the DMA mask (DM bit) control bit is not cleared, but the End Of Block interrupt request is generated to allow the optional updating of the data table (table 0).

Until the swap mode is not disabled, the DMA controller will continue to swap between DMA Table 0 and DMA Table 1.

Figure 4-12. DMA Transaction to Memory

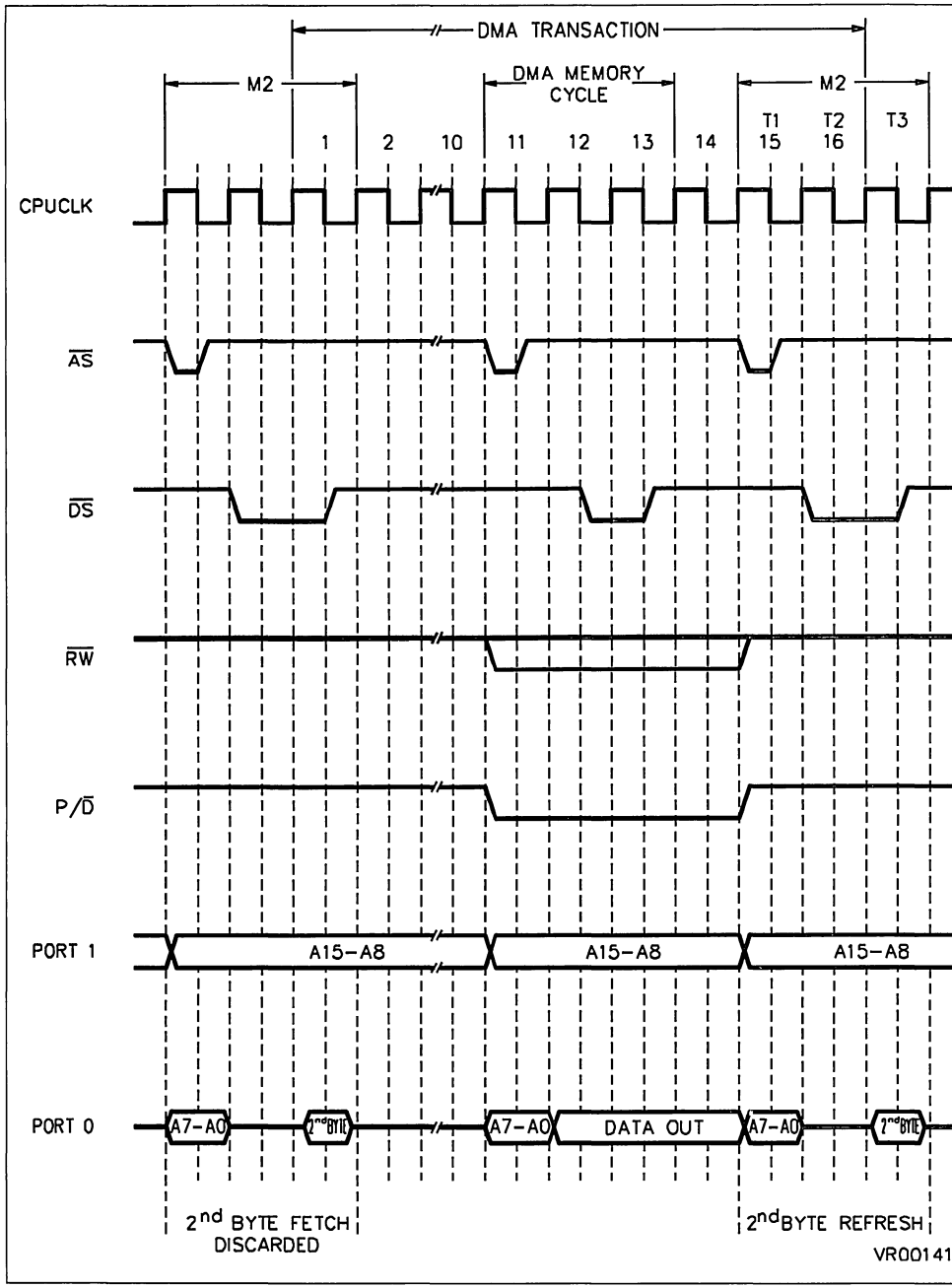
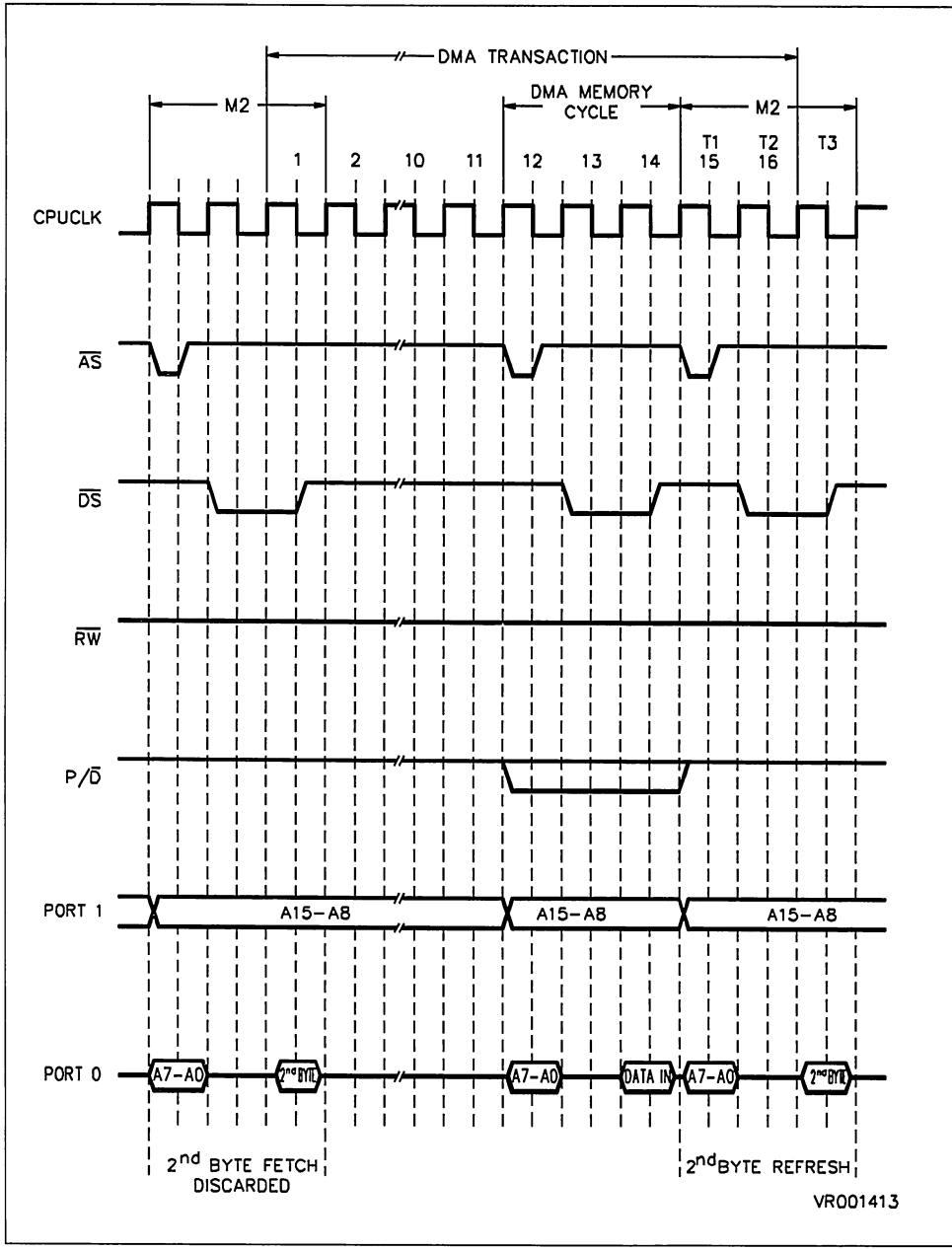


Figure 4-13. DMA Transaction from Memory



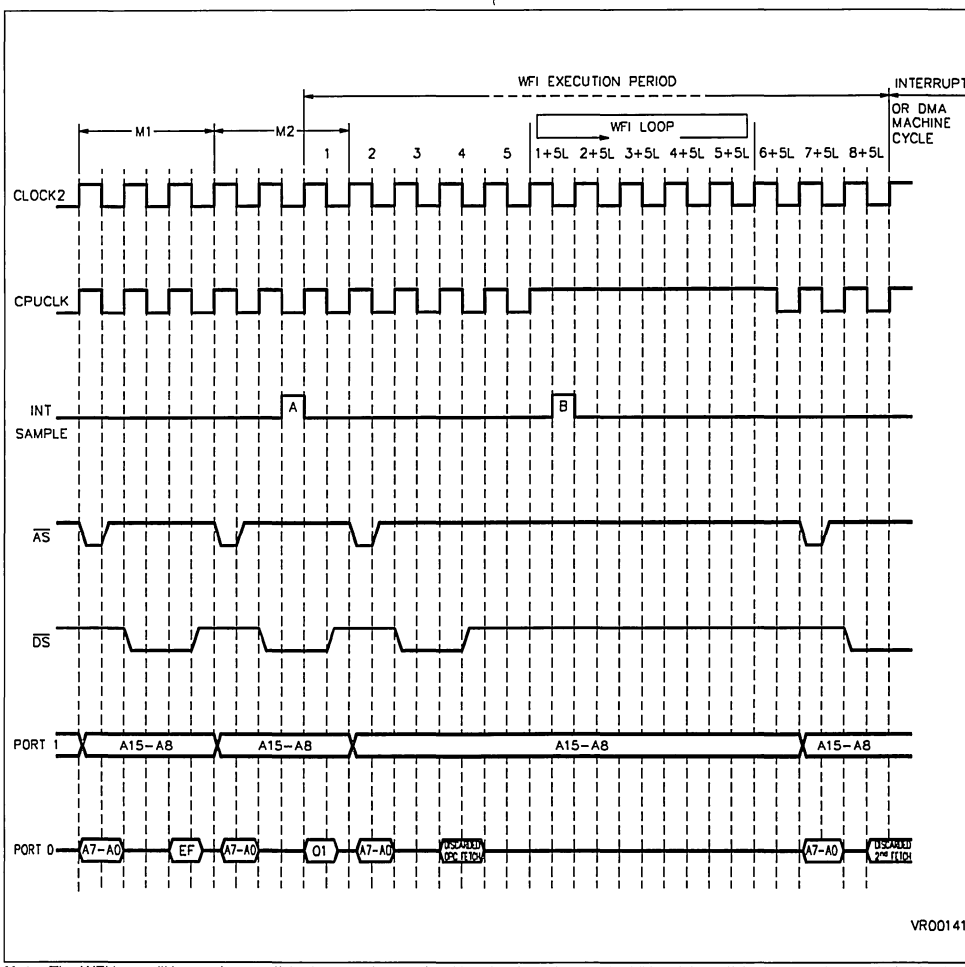
4.9 WAIT FOR INTERRUPT INSTRUCTION

The Wait For Interrupt instruction suspends program execution until an interrupt request is acknowledged. During this period, DMA transactions are executed if their priority allows it. During the WFI instruction, the CPUCLK is halted while INTCLK keeps running. Under this state, the power consumption of the processor is lowered by the CORE power consumption value.

4.10 INTERRUPT AND DMA RESPONSE TIME

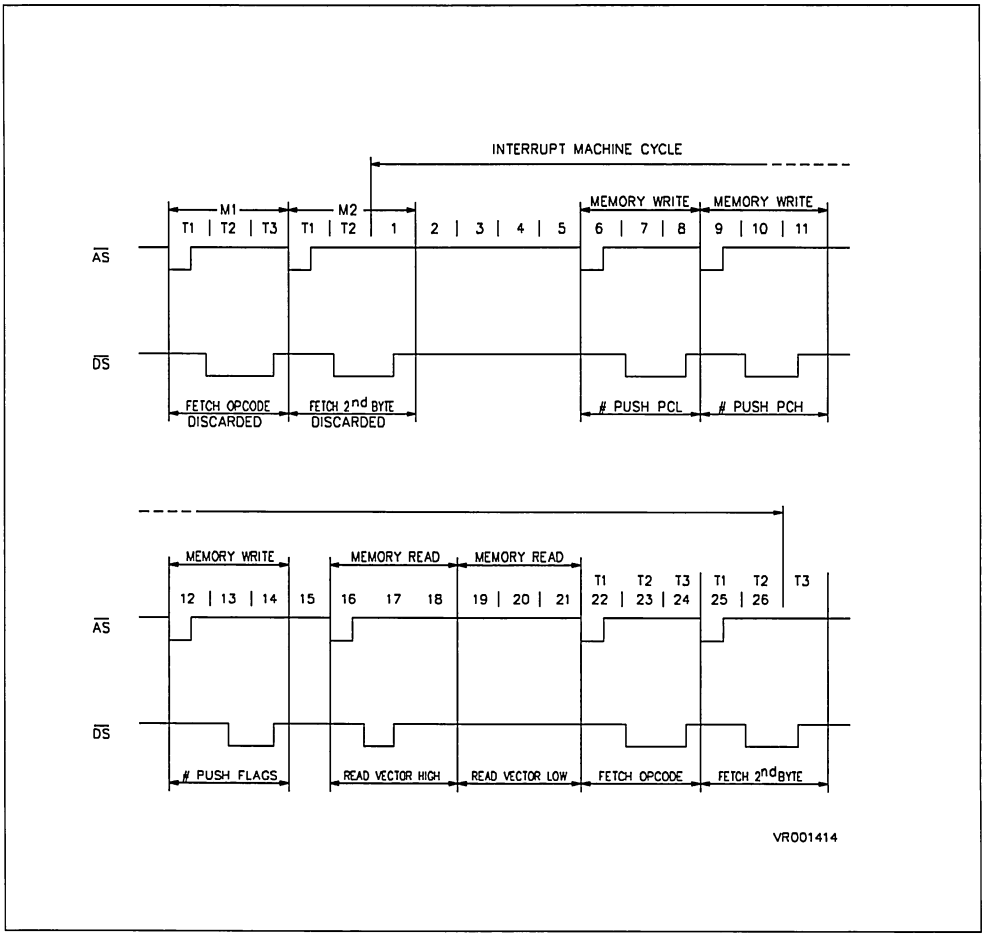
Interrupt and DMA requests are sampled 6 INTCLK cycles before the end of the instruction. If Wait For Interrupt is in progress, requests are sampled every 5 INTCLK cycles. If the interrupt request comes from an external pin, the programmed event has to be set a minimum of one CPUCLK cycle before the sampling time.

Figure 4-14. Wait for Interrupt Timing



Note: The WFI loop will be made once if the interrupt is sample with pulse A, and several additional times if the interrupt is sampled pulse B.

Figure 4-15. ST9 Interrupt Acknowledge Timing



A DMA transfer with the Register file takes 8 CPUCLK cycles, except when the Wait For Interrupt is in progress (10 CPUCLK cycles).

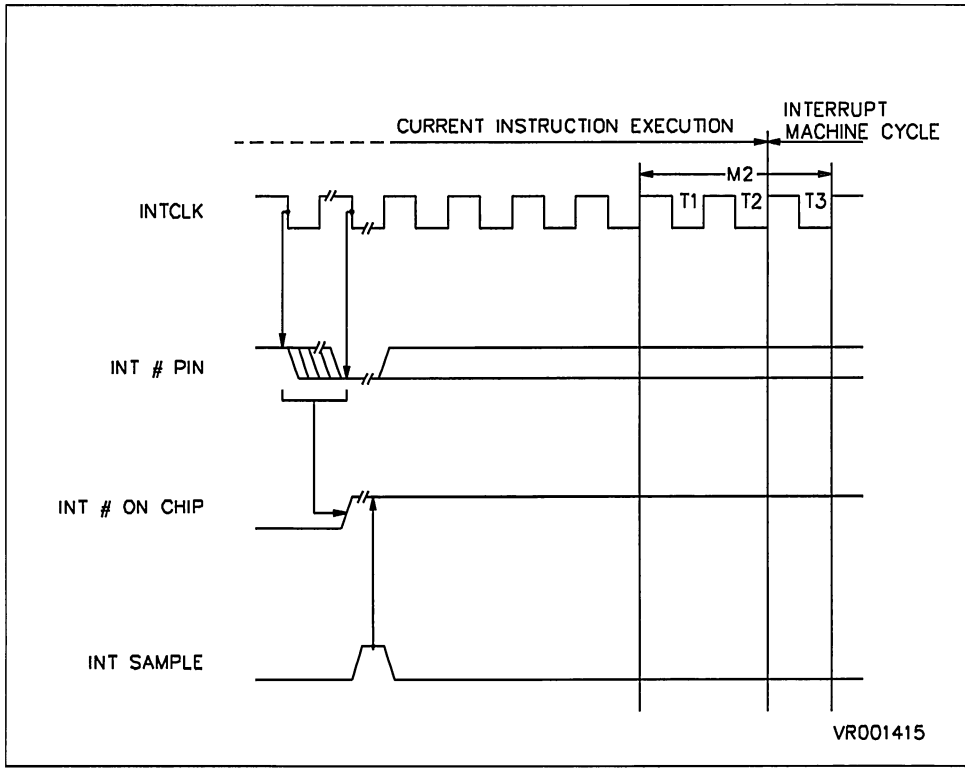
A DMA transfer with the memory takes 16 CPUCLK cycles except when the Wait For Interrupt is in progress (18 CPUCLK cycles).

In order to guarantee the falling/rising edge detection, input signals must be kept low/high for a minimum of one CPUCLK cycle.

An interrupt machine cycle takes 26 internal clock cycles (CPUCLK), with some exceptions as follows:

- 28 internal clock cycles (CPUCLK), if a Wait For Interrupt is in progress
- 32 internal clock cycles (CPUCLK), if the acknowledge cycle follows a DMA transfer with Register File

Figure 4-16. External Interrupt Response Time



4.11 ST9 INTERRUPT REGISTERS

CICR - R230 (0E6h) Sys. Reg.; Read/Write
Central Interrupt Control Register

Reset value: 1000 0111 (87h)

7							0
GCEN	TLIP	TLI	IEN	IAM	CPL2	CPL1	CPL0

b7 = **GCEN:** *Global Counter Enable bit.* When set the 16 bit MultiFunction Timers are enabled (see Timer Control Register in MULTI FUNCTION TIMER chapter)

b6 = **TLIP:** *Top Level Interrupt Pending bit.* Set by hardware when the Trigger Event occurs. Cleared by hardware when the Top Level Interrupt is aknowledged.

b5 = **TLI:** *Top Level Interrupt bit.* If TLI ="1", and IEN is set, a Top Level Interrupt request is generated as TLIP is set. If TLI = "0", a request is generated only if TLNM is set.

b4 = **IEN:** *Interrupt Enable.* If IEN = "0", no maskable Interrupt requests are generated. This bit is cleared by the interrupt machine cycle and it is set by the IRET instruction of maskable routines.

b3 = **IAM:** *Interrupt Arbitration Mode.* If IAM = "0", Concurrent Arbitration Mode is selected; If IAM = "1" Nested Mode is selected.

b2-b0 = **CPL2, CPL1, CPL0:** *Current Priority Level.* Defines the Current Priority Level under service. CPL=0 is the highest priority. CPL=7 is the lowest priority.

EITR - R242 (0F2h) Page 0; Read/Write
External Interrupt Trigger Event Register

Reset value: XXXX 0000 (00h)

7							0
TED1	TED0	TEC1	TEC0	TEB1	TEB0	TEA1	TEA0

If TExy bit is set, the pending bit will be set upon the rising edge of the input signal.

If TExy is cleared, the pending bit will be set upon the falling edge of the input signal.

All bits are set/reset only by software.

b7 = **TED1:** *Trigger Event of Interrupt Channel D1*

b6 = **TED0:** *Trigger Event of Interrupt Channel D0*

b5 = **TEC1:** *Trigger Event of Interrupt Channel C1*

b4 = **TEC0:** *Trigger Event of Interrupt Channel C0*

b3 = **TEB1:** *Trigger Event of Interrupt Channel B1*

b2 = **TEB0:** *Trigger Event of Interrupt Channel B0*

b1 = **TEA1:** *Trigger Event of Interrupt Channel A1*

b0 = **TEA0:** *Trigger Event of Interrupt Channel A0*

IDPR - R243 (0F3h) Page 0; Read/Write
External Interrupt Pending Register

Reset value: 0000 0000 (00h)

7							0
IPD1	IPD0	IPC1	IPC0	IPB1	IPB0	IPA1	IPA0

b7 = **IPD1:** *Interrupt Pending bit Channel D1*

b6 = **IPD0:** *Interrupt Pending bit Channel D0*

b5 = **IPC1:** *Interrupt Pending bit Channel C1*

b4 = **IPC0:** *Interrupt Pending bit Channel C0*

b3 = **IPB1:** *Interrupt Pending bit Channel B1*

b2 = **IPB0:** *Interrupt Pending bit Channel B0*

b1 = **IPA1:** *Interrupt Pending bit Channel A1*

b0 = **IPA0:** *Interrupt Pending bit Channel A0*

IP bits are hardware set upon the occurence of the trigger event and are reset by the interrupt acknowledge machine cycle.

Note: IP bits may be set by the programmer to implement a software interrupt.

EIMR - R244 (0F4h) Page 0; Read/Write
External Interrupt Mask-bit Register

Reset value: 0000 0000 (00h)

7							0
IMD1	IMD0	IMC1	IMC0	IMB1	IMB0	IMA1	IMA0

EIMR bits are set/reset by software

When the IM bit is set (and the global IEN is enabled), an interrupt request is generated if the corresponding IP bit is set. When IM = "0", no request will be generated.

- IMxy = "1": an interrupt request can be acknowledged (depending on IEN)
- IMxy = "0": an interrupt request is masked.

b7 = **IMD1:** *Interrupt Mask of Interrupt Channel D1*

b6 = **IMD0:** *Interrupt Mask of Interrupt Channel D0*

b5 = **IMC1:** *Interrupt Mask of Interrupt Channel C1*

b4 = **IMC0:** *Interrupt Mask of Interrupt Channel C0*

b3 = **IMB1:** *Interrupt Mask of Interrupt Channel B1*

b2 = **IMB0:** *Interrupt Mask of Interrupt Channel B0*

b1 = **IMA1:** *Interrupt Mask of Interrupt Channel A1*

b0 = **IMA0:** *Interrupt Mask of Interrupt Channel A0*

EIPLR - R245 (0F5h) Page 0; Read/Write
External Interrupt Priority Level Register

Reset value: 1111 1111 (FFh)

7							0
PL2D	PL1D	PL2C	PL1C	PL2B	PL1B	PL2A	PL1A

EIPLR bits are set/reset by software

b7-b6 = **PL1D, PL2D:** *Priority level for the Group INTD0, INTD1*

b5-b4 = **PL1C, PL2C:** *Priority level for the Group INTC0, INTC1*

b3-b2 = **PL1B, PL2B:** *Priority level for the Group INTB0, INTB1*

b1-b0 = **PL1A, PL2A:** *Priority level for the Group INTA0, INTA1*

EIVR - R246 (0F6h) Page 0; Read/Write
External Interrupt Vector Register

Reset value: xxxx 0110 (X6h)

7							0
V7	V6	V5	V4	TLTEV	TLIS	IAOS	EWEN

b7-b4 = **V7 to V4:** *Most significant nibble of External Interrupt Vector.* Not initialized by reset.

b3 = **TLTEV:** *Top Level Trigger Event bit* When set, the Top Level event is triggered on rising edge of NMI input pin. Triggering on the falling edge of the NMI input pin is activated when this bit is "0" (reset value)

b2 = **TLIS:** *Top Level Input Selection bit* This bit selects the source of the Top Level Interrupt between the external NMI pin (when "1", the reset value) and the Timer/Watchdog End of Count (when "0").

b1 = **IA0S:** *Interrupt A0 Selection bit* When set, the External Interrupt pin is selected as the External Interrupt Channel A0 source. When reset the source is the Timer/Watchdog End of Count interrupt.

b0 = **EWEN:** *External Wait Enable bit* When set, this bit enables the WAIT input pin to stretch the external memory access cycle. For more details of the WAIT mode, the reader should refer to the Clock and Wait chapter or External memory Interface chapter.

NICR - R247 (0F7h) Page 0; Read/Write
Nested Interrupt Control Register

Reset value: 0000 0000 (00h)

7							0
TLNM	HL6	HL5	HL4	HL3	HL2	HL1	HL0

b7 = **TLNM:** *Top Level Not Maskable.*

If TLNM = "1", a top level request is generated as TLIP is set. Once TLNM is set, it can be cleared only with an hardware reset

bx = **HLx:** *Hold Level x* These bits are set to "1" when, in Nested Mode, an interrupt service routine at level x is interrupted from a request with higher priority (other than the Top Level interrupt request). It is cleared by the `IRET` execution when the routine at level x is recovered.

b6 = **HL6:** *Hold Level 6*

b5 = **HL5:** *Hold Level 5*

b4 = **HL4:** *Hold Level 4*

b3 = **HL3:** *Hold Level 3*

b2 = **HL2:** *Hold Level 2*

b1 = **HL1:** *Hold Level 1*

b0 = **HL0:** *Hold Level 0*

4.12 ST9 DMA REGISTERS

As each peripheral DMA channel has its own control registers, the following register list should be considered as a general example. The names and register bit allocation shown here may be different from those found in the peripheral chapters.

IDCR Address set by Peripheral; Read/Write
Generic External Peripheral Interrupt and DMA Control Register

Reset value: undefined

7							0
		IP	DM	IM	PRL2	PRL1	PRL0

b5 = **IP:** *Interrupt Pending* Set by hardware when the Trigger Event occurs Cleared by hardware when the request is acknowledged. Can be set/cleared by software in order to generate/cancel a pending request. Identical in function to IP of ICR.

b4 = **DM:** *DMA Mask* If DM = "0" no DMA request is generated when the trigger event occurs. This bit is cleared whenever the transaction counter reaches zero (unless SWAP mode is active).

b3 = **IM:** *Interrupt Mask.* If IM = "0" no interrupt request is generated. If IM = "1" DMA requests depend on DM bit value as shown below.

DM	IM	
1	0	DMA request (without interrupt at End of Block)
1	1	DMA request (with interrupt at request End of Block)
0	0	No request interrupt
0	1	Interrupt request (without DMA associated is not used)

b2-b0 = **PRL2, PRL1, PRL0:** *Priority Level* Definition of the source priority level. PRL = 0 is highest priority. If PRL = 7, no interrupt can be acknowledged, DMA request will be.

DCPR Address set by Peripheral; Read/Write
DMA Counter Pointer Register

Reset value: undefined

7							0
C7	C6	C5	C4	C3	C2	C1	RM

b7-b1 = **C7-C1:** *DMA Transfer Counter Register(s) Address*

b0 = **RM:** *Register File/Memory Selector* If set, the DMA transactions are done with the Register File; if cleared, the DMA transactions are done with the Program or Data memory (see DAPR.DP)

DAPR Address set by Peripheral; Read/Write
DMA Address Pointer Register

Reset value: undefined

7							0
A7	A6	A5	A4	A3	A2	A1	DP

b7-b1 = **A7-A1:** *DMA Address Register(s) Address*

b0 = **DP:** *Data/Program Memory Selector:* (DAPR.RM is "0") if set the DMA transactions are made with the Data Memory; if cleared the DMA transactions are made with the Program Memory.

RM	DP	DMA Source/Destination
0	0	Program Memory
0	1	Data Memory
1	0	Register File
1	1	Register File

5.1 CLOCK

5.1.1 Introduction

The ST9 Clock generator module generates the internal clock for the ST9 core and the on-chip peripherals. The Clock generator can be driven by an external crystal circuit, connected to the OSCIN and OSCOUT pins, or by an external pulse generator, connected to OSCIN.

5.1.2 Clock Management

The oscillator circuit generates an internal clock signal CLOCK1 with the same period and phase as at the OSCIN input pin. The maximum frequency allowed for CLOCK1 is 24MHz.

The signal CLOCK1 drives a programmable divider by two. If the control bit MODER.DIV2 (R235.5) is set, the internal clock CLOCK2 is CLOCK1 divided by two; otherwise, if DIV2 bit is cleared, the clock signal CLOCK2 has the same period and phase as CLOCK1. CLOCK2 drives the internal clock INTCLK delivered to all ST9 on-chip peripherals and acts as the central timebase for all timing functions (eg Multifunction Timer or Serial Communications Interface Baud Rate generator). The maximum frequency allowed for INTCLK is 12MHz.

CLOCK2 also drives a programmable prescaler which generates the basic time base, CPUCLK, for the instruction executer ofthe ST9 core. This allows the user to slow the program execution time to reduce power dissipation, and to speed up certain code segments for time critical routines. The internal peripherals are not affected by the CPUCLK prescaler. The prescaler value divides the input clock by the value programmed in the control bits MODER.PRS2,1,0 (R235.4,3,2). If the prescaler value is zero, no prescale is made, thus CPUCLK has the same period and phase as CLOCK2 and INTCLK. If the value is different from 0, the prescaling is equal to the value plus one, ranging thus from two (PRS2,1,0 = 1) to eight (PRS2,1,0 = 7). The clock generated is shown in Figure 5.2. It must be noticed that the prescaling of the clock does not keep the duty cycle to 50%, but stretches the high level of the clock until completion.

When External Memory Wait (or Bus Request or Wait for Interrupt) events occur, CPUCLK is stretched on the high level for the whole period required by the function.

Note: The added wait cycles refer to the INTCLK frequency and not the original CPUCLK.

Figure 5.3 shows an example of a memory access cycle with the CPUCLK prescaled by 2 and with 5 added Wait states.

Figure 5-1. Peripheral and Core Clocks

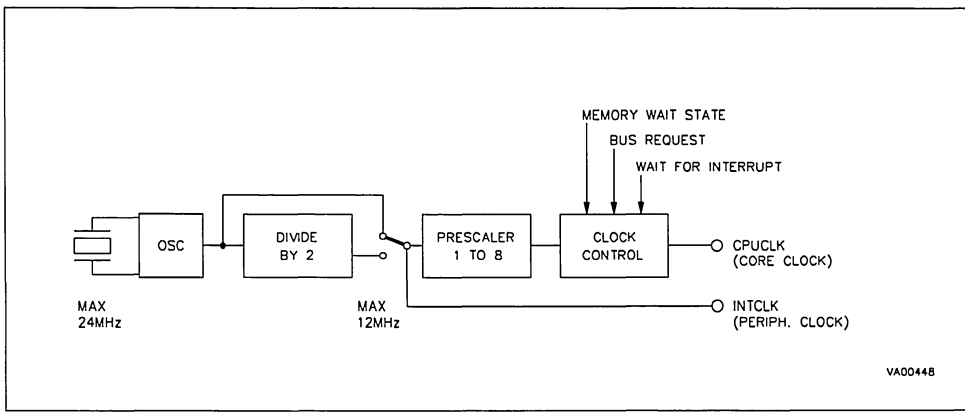
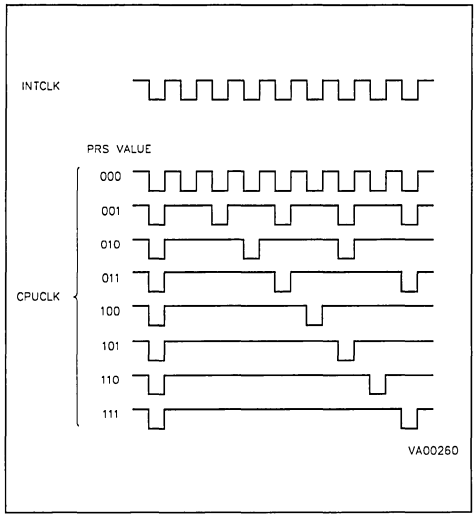


Figure 5-2. ST9 Core Clock Prescaling



5.1.3 Clock Control Registers

The ST9 clock division by 2 and the clock prescaling are controlled by the MODER register. SSP, USP, BRQEN and HIMP control bits are not related to the control of the clock.

MODER - R235 (0EBh) Sys.Reg. Read/Write

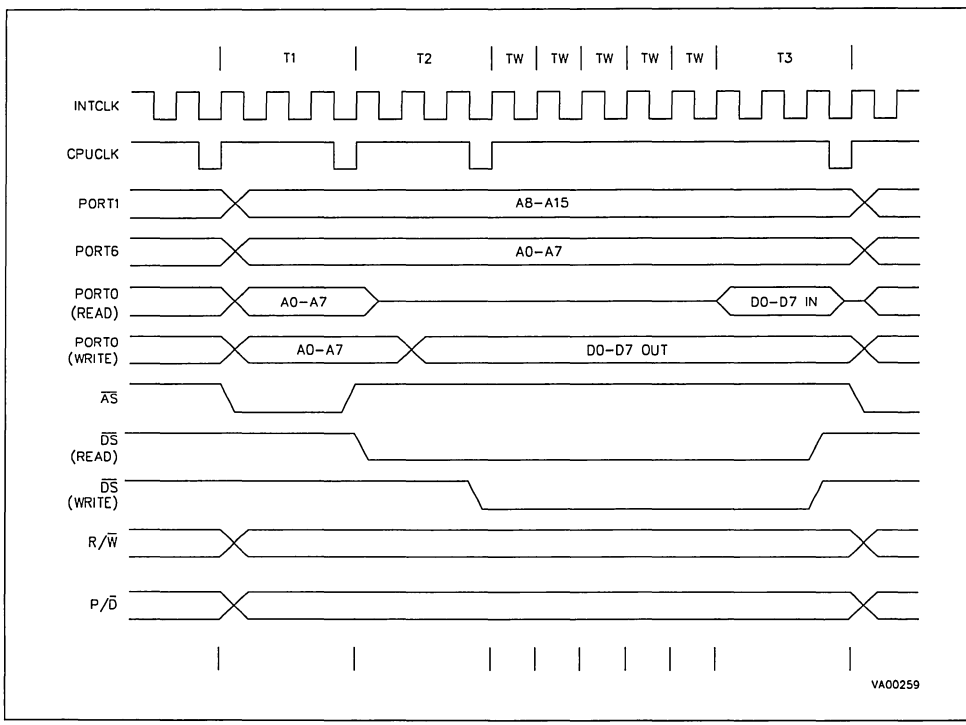
Mode Register
Reset Value : 1110 0000 (OE0h)

7							0
SSP	USP	DIV2	PRS2	PRS1	PRS0	BRQEN	HIMP

b5 = **DIV2:** *OSCIN Divided by 2.* This bit controls the divide by 2 circuit which operates on the OSCIN Clock. A logical "1" value means that the OSCIN clock is internally divided by 2, and a logical "0" value means that no division of OSCIN Clock occurs.

b4- b2 = **PRS2, PRS1, PRS0:** *Prescaling of ST9 Clock.* These bits define the prescaler value used to prescale the CPUCLK from INTCLK. When these three bits are reset, the internal clock is not prescaled, and is equal to OSCIN frequency; in all other cases, the internal clock is prescaled by the value of (PRS2,1,0 + 1)

Figure 5-3. Memory Access with a Clock Prescaled by 2 and 5 Wait Cycles



5.1.4 Oscillator Characteristics

The on-chip oscillator circuit (Figure. 5.4) is an inverting gate circuit.

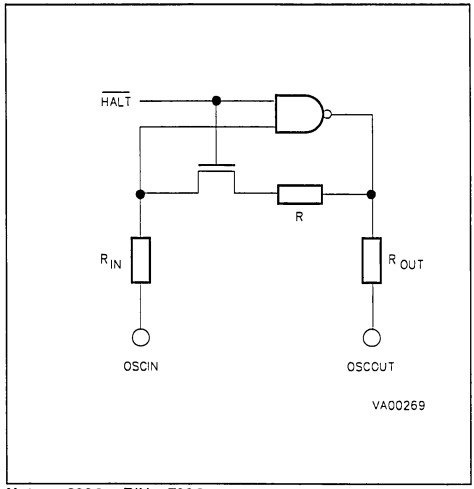
Note: Owing to the Q factor required of the external oscillator, Ceramic Resonators may not provide a reliable oscillator source.

In Halt mode, set by means of the HALT instruction, the parallel resistor R is disconnected and the oscillator is disabled, forcing the internal clock CLOCK1 to a high level and OSCOUT to a low level.

To exit the HALT condition and restart the oscillator, an external RESET pulse is required of a minimum duration of 10ms (Figure. 5.6).

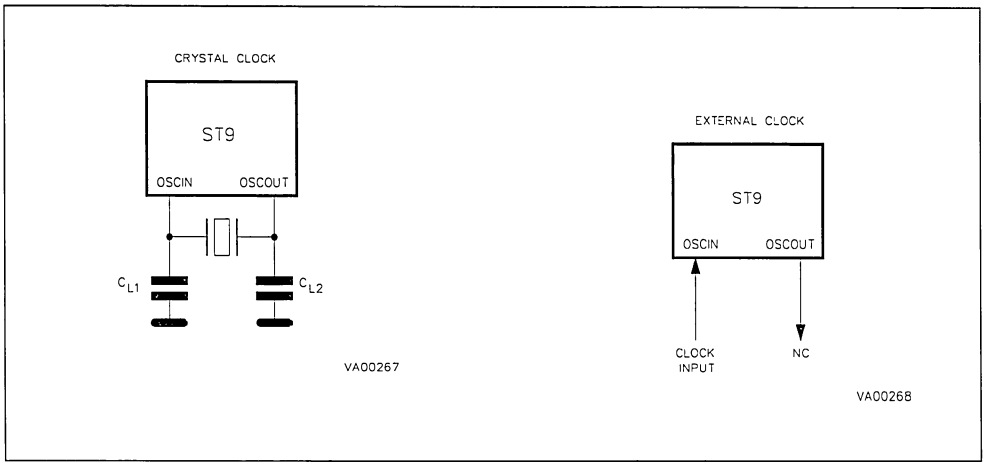
It must be noted that if the Timer/Watchdog watchdog function is enabled, a HALT instruction will not disable the oscillator. This to avoid to stopping the watchdog if, by an error, a HALT code is executed. When this occurs, the ST9 CPU falls into an endless loop ended by the watchdog (or external) reset.

Figure 5-4. ST9 Internal Oscillator Schematic



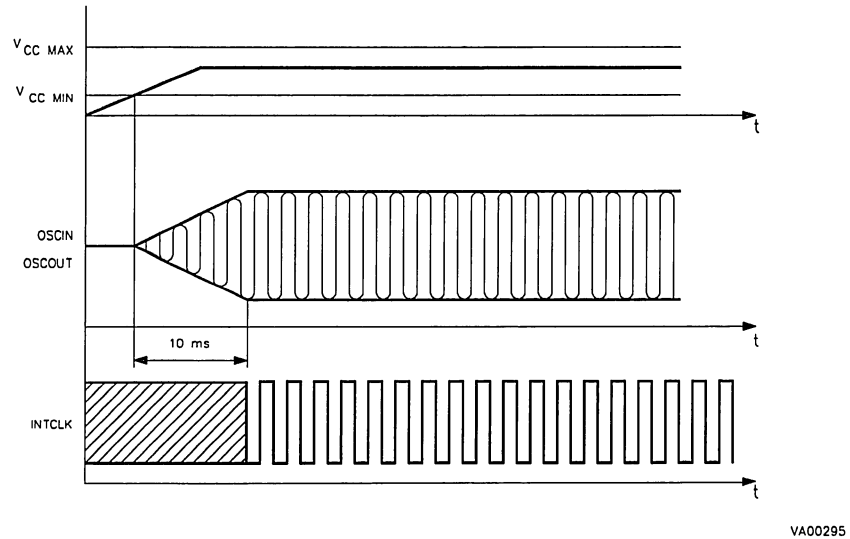
Note : 300Ω < RIN < 700Ω
ROUT > 1.5MΩ
R>50Ω

Figure 5-5. ST9 Oscillator Drive by Crystal and External Oscillator Drive



Note : When the oscillator is used with a crystal, the associated capacitors C1 and C2 are typically 22pF for crystal ranging from 8MHz to 24 MHz. When the oscillator is driven with an external generator, the oscout pin must stay unconnected

Figure 5-6. Oscillator Start-up Sequence



5.2 RESET

5.2.1 Introduction

The processor Reset overrides all the other conditions and forces the ST9 to the reset state. During reset, the processor internal registers are set to the reset value, as shown in figure. 5.7.

5.2.2 Reset Generation

The reset condition can be generated by the external pin $\overline{\text{RESET}}$ or by the on-chip Timer/Watchdog.

The on-chip Timer/Watchdog generates a reset condition if the watchdog mode is enabled (WCR.WDEN cleared, R252 page 0), and if the programmed period elapses without the specific code (AAh,55h) written to the appropriate register. The input pin RESET is not driven low by the on-chip reset generated by the Timer/Watchdog.

During reset, the $\overline{\text{DS}}$ output signal is kept low and the $\overline{\text{AS}}$ output is toggled with the crystal frequency (input at OSCIN) divided by 32. This condition is recognized by off-chip Z-bus peripherals as a reset condition.

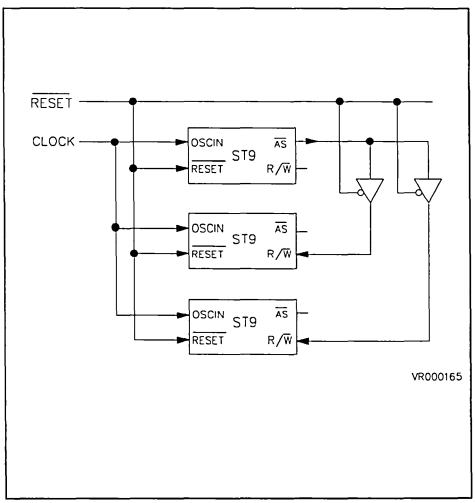
5.2.3 Processor Synchronization Under Reset

During reset, a specific procedure has been implemented to synchronize two or more oscillators in a multi-micro ST9 based system, for example a majority voting high reliability system. Figure. 5.8 shows the principle schematic for the multi-microprocessor synchronization. The master processor delivers the synchronous signal, output at its $\overline{\text{AS}}$ pin, to the R/$\overline{\text{W}}$ pin of the slave processors. The R/$\overline{\text{W}}$ pin is, under reset status, set to input mode with a weak (10kΩ typical) pull up resistor. The slave processor(s) synchronizes its internal clock phase with the clock received at its R/$\overline{\text{W}}$ pin. To guarantee the phase synchronization, the reset status must be at least 32x31 = 992 crystal periods. All the processors must have the same input clock.

Figure 5-7. Internal Registers Reset Values

Register Number	System Register Reset Value	Page 0 Register Reset Value
F	(SSPLR) = undefined	Reserved
E	(SSPHR) = undefined	(SPICR) = 00h
D	(USPLR) = undefined	(SPIDR) = undefined
C	(USPHR) = undefined	(WCR) = 7Fh
B	(MODER) = E0h	(WDTCR) = 12h
A	(Page Ptr) = undefined	(WDTPR) = undefined
9	(Reg Ptr 1) = undefined	(WDTLR) = undefined
8	(Reg Ptr 0) = undefined	(WDTHR) = undefined
7	(FLAGR) = undefined	(NICR) = 00h
6	(CICR) = 87h	(EIVR) = x2h
5	(PORT5) = FFh	(EIPLR) = FFh
4	(PORT4) = FFh	(EIMR) = 00h
3	(PORT3) = FFh	(EIPR) = 00h
2	(PORT2) = FFh	(EITR) = 00h
1	(PORT1) = FFh	(EEPROM) = xx00 0000b
0	Reserved	Reserved

Figure 5-8. Synchronization Under Reset



5.2.4 Eprom Programming Pin

The ST9 versions with on-chip EPROM memory require an external programming voltage Vpp to perform the programming procedure. The Vpp voltage must be applied to the RESET pin during the whole programming phase. Refer to the EPROM Programming Board Manual for specifications.

5.2.5 Reset Pin Timing

The $\overline{\text{RESET}}$ pin has a Schmitt trigger input circuit with hysteresis. The internal reset is generated by the external pin synchronized with the internal clock. The power up reset circuit must keep the $\overline{\text{RESET}}$ input low for a minimum of the crystal startup period plus 53 crystal periods.

Once the $\overline{\text{RESET}}$ pin reaches a logical 1, the processor exits from the reset status after 67 crystal periods ($\overline{\text{DS}}$ is set). The processor then fetches from Program Memory locations 0 and 1 (power-on reset vector) and begins program execution from the address contained in the vector. If the ST9 is a ROMLESS version, without on-chip program memory, ports Port0, Port1 and Port6 are set to external memory mode (i.e Alternate Function) and the memory accesses are made to external Program memory with wait cycles insertion (see chapter 6 External Memory Interface).

Figure 5-9. Signal To Be Applied On Reset Pin

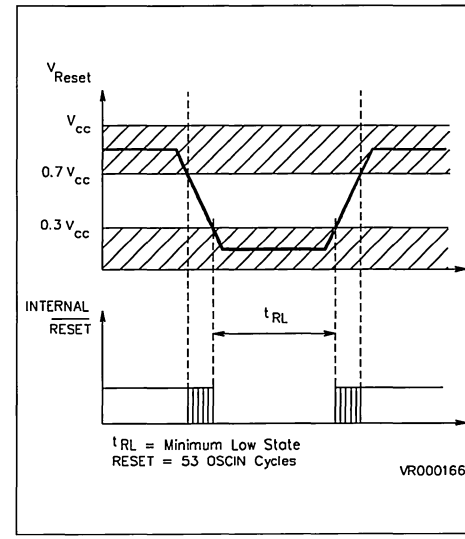
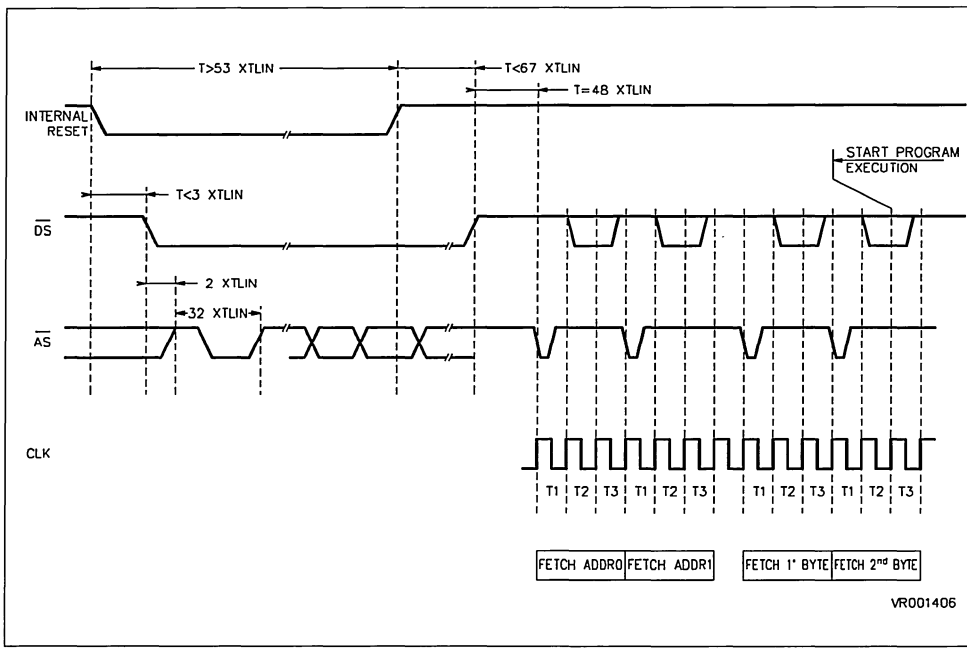


Figure 5-10. Exit From Reset Timing



INTERFACING EXTERNAL MEMORY

6.1 GENERAL OVERVIEW

The ST9 microcontroller provides an external memory interface with enhanced features including non-multiplexed bus capabilities, programmable memory wait cycles, bus request/acknowledge cycles and shared memory bus access control. Data and Address are provided on Port 0, Port 1 and, depending upon the family member, Port 6. Control signals are generated at the $\overline{AS}$, $\overline{DS}$ and R/$\overline{W}$ pins. The control signals P/$\overline{D}$ (to access up to 128 kbytes of memory address space), $\overline{WAIT}$, $\overline{BREQ}$, $\overline{BACK}$ are provided, as Alternate Functions of the general purpose parallel ports.
The ST9 Memory Control Unit automatically recognizes if a memory location belongs to on-chip memory. When the memory location is on-chip, it performs a machine cycle without $\overline{DS}$ generation, and the access is performed on-chip. If the location does not belong to on-chip memory, an access to off-chip memory is performed (generating the $\overline{DS}$ low pulse) through the Ports 0, 1 and 6.

During Reset, $\overline{AS}$ and $\overline{DS}$ are driven to perform external peripherals reset and to implement, in conjunction with the R/W pin, a multi-microprocessor synchronization procedure.

6.2 CONTROL SIGNALS

$\overline{AS}$

Address Strobe (Output, Active low, Tristate). The rising edge of AS indicates that Memory Address, Read/Write and Program/Data Memory signals are valid.

$\overline{DS}$

Data Strobe (Output, Active low, Tristate). Data Strobe provides the memory data timing during external memory access cycle. When internal memory is accessed, $\overline{DS}$ is kept high during the whole memory cycle.

R/$\overline{W}$

Read/Write (Output, Active low, Tristate). The R/$\overline{W}$ output signal identifies the type of memory cycle: Read if R/$\overline{W}$ = "1", Write if R/$\overline{W}$ = "0".

P/$\overline{D}$

Program/Data Memory (Alternate Function Output, Active low). The P/$\overline{D}$ output signal selects between Program and Data Memory. P/$\overline{D}$ = "1" for program memory, P/$\overline{D}$ = "0" for data memory.

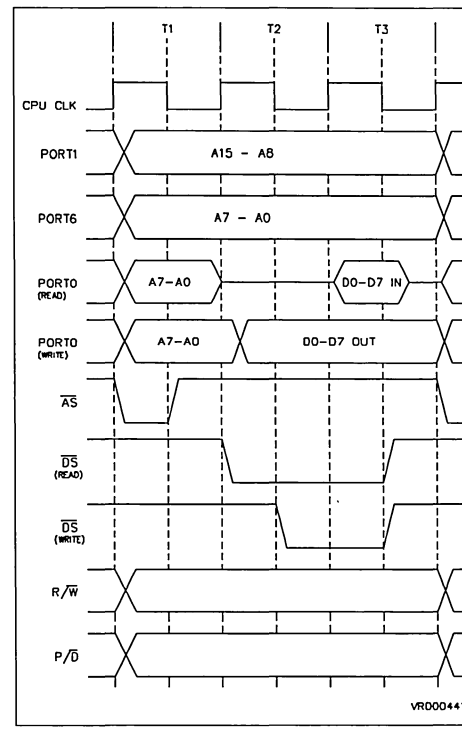
$\overline{WAIT}$

External Memory Wait (Alternate Function Input, Active low). The $\overline{WAIT}$ input signal indicates to the ST9 that the external memory requires more time to complete the memory access cycle. The memory cycle will then be stretched.

$\overline{BREQ}$

Bus Request (Alternate Function Input, Active low). The $\overline{BREQ}$ input signal indicates to the ST9 that a bus request has tried or is trying to gain control of the memory bus.

Figure 6-1. External Memory Read/Write



$\overline{BACK}$

Bus Acknowledge (Alternate Function Output, Active low). The $\overline{BACK}$ output signal indicates that the ST9 has relinquished control of the memory bus in response to a bus request.

P0

Port 0 (Input/Output, Push-Pull/Open-Drain/Weak Pull-up). Port0 can be configured as a bit programmable Parallel I/O port or as External Memory interface for multiplexed Low-Address/Data (A0-7/D0-7).

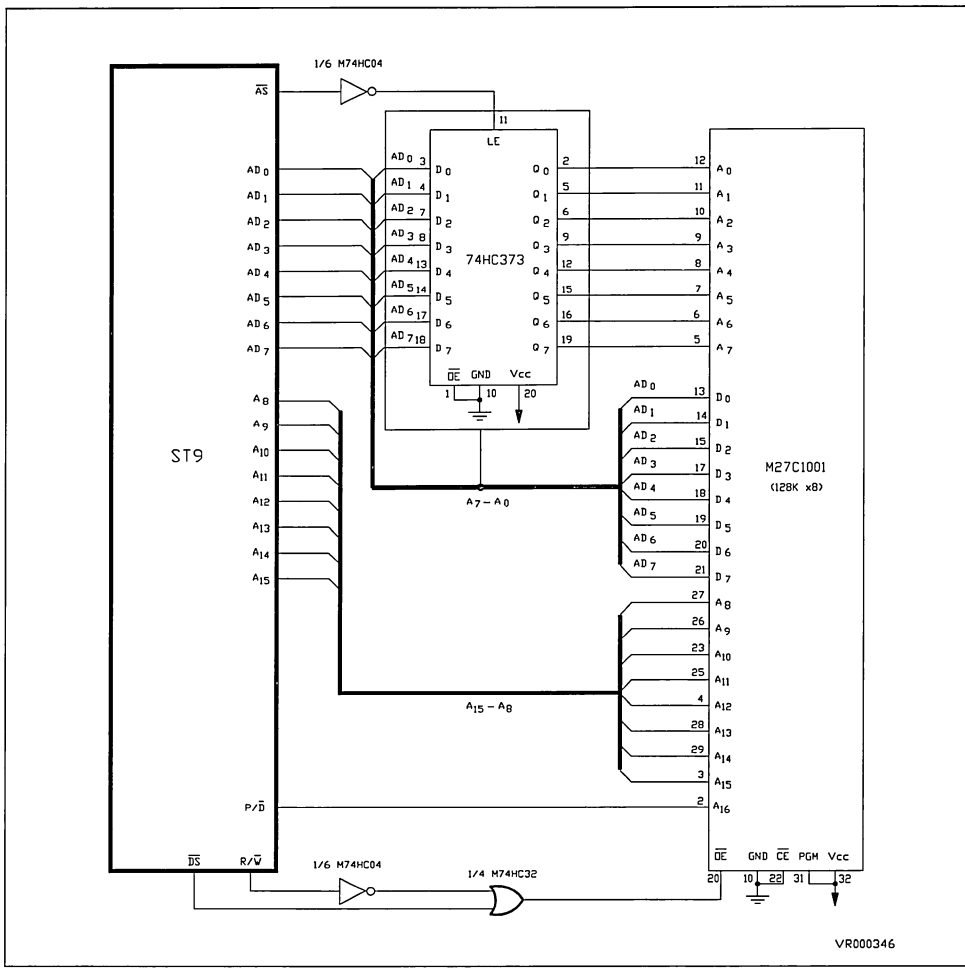
P1

Port 1 (Input/Output, Push-Pull/Open-Drain/Weak Pull-up). Port1 can be configured as a bit programmable Parallel I/O port or as External Memory interface for the High-Address (A8-A15).

P6 (When available)

Port 6 (Input/output, Push-Pull/Open-Drain/Weak Pull-up). This port, when available, can be configured as bit programmable Parallel I/O port or as External Memory interface for the Low-Address (A0-7), allowing a non-multiplexed memory bus capability.

Figure 6-2. ST9 Accessing External Program and Data Memory



6.3 MEMORY ACCESS CYCLE

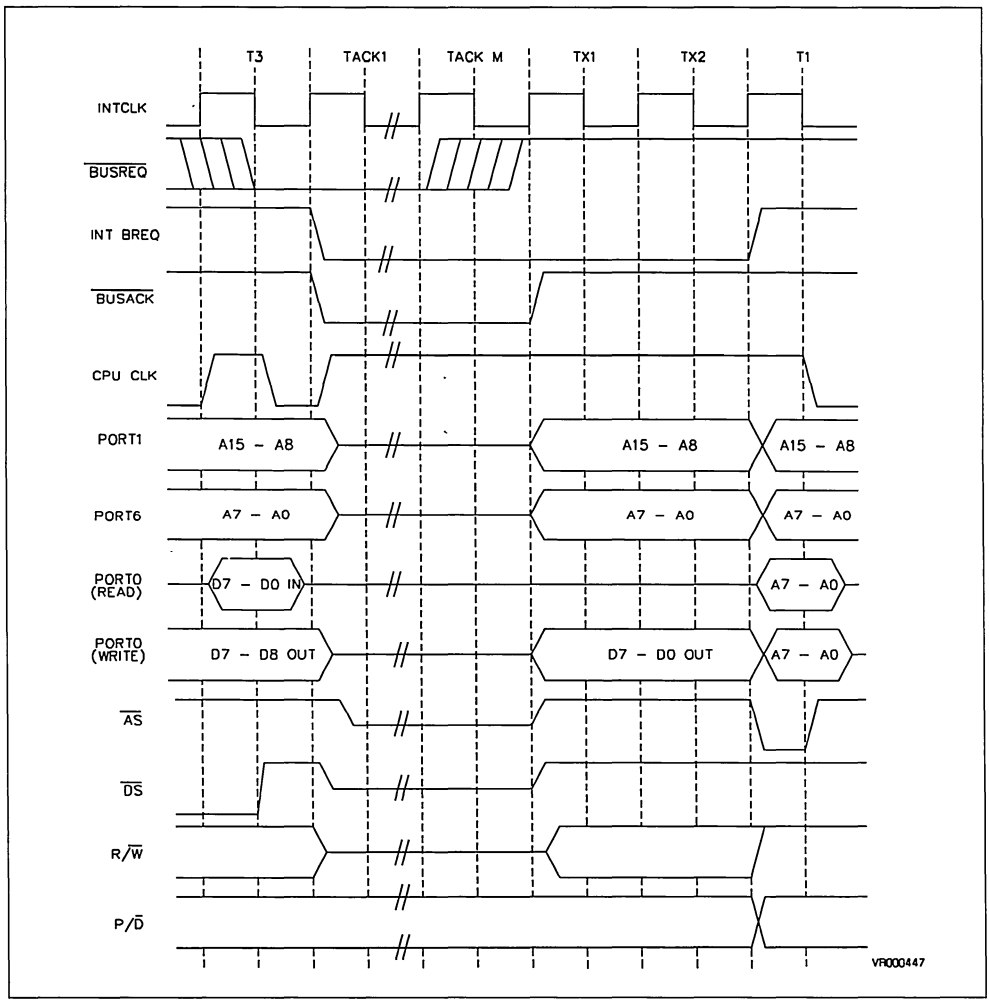
The access to external memory is made using the following signals:

$\overline{AS}$

$\overline{AS}$ is active during the internal clock high-level phase of each T1 memory cycle. $\overline{AS}$ is released in high-impedance during Bus acknowledge cycle or under processor control by setting the HIMP bit (MODER.0).

Under the ST9 Reset state, the $\overline{AS}$ pin outputs the external clock divided by 32 (f[AS] = f[OSCIN/32]. This signal is used, in conjunction with $\overline{DS}$, to perform an external peripheral reset ($\overline{AS}$ low and $\overline{DS}$ low) according to the Z-bus protocol, and, in conjunction with R/$\overline{W}$, to synchronize Slave processors in a multi-microcontroller system.

Figure 6-3. Bus Request/Acknowledge Timing



$\overline{DS}$

During an External memory write cycle, the data output at Port 0 is valid when $\overline{DS}$ is active. During a read cycle, the data at Port 0 must be valid before the trailing edge of $\overline{DS}$. $\overline{DS}$ is released in high-impedance during Bus acknowledge cycle or under processor control by set of HIMP bit (MODER.0).
When the processor accesses on-chip memory, the $\overline{DS}$ is held high. Under Reset status, the $\overline{DS}$ pin is kept low to generate the external peripheral reset command.

R/$\overline{W}$

When R/$\overline{W}$ = "1", the memory cycle is a Memory Read cycle; when R/$\overline{W}$ = "0", it is a Memory Write Cycle. R/$\overline{W}$ output signal is defined at the beginning of the memory cycle and is stable until the next Memory cycle. R/$\overline{W}$ is released in high-impedance during Bus acknowledge cycle or under processor control by setting the HIMP bit.
Under Reset status, the pin is set as an input and held high with a weak internal pull-up. To synchronize the processor phase with the phase of a Master Processor in a multi-processor application, the R/$\overline{W}$ pin must receive the $\overline{AS}$ signal of the Master.

P/$\overline{D}$

When P/$\overline{D}$ = "1", the memory referenced by the processor is the Program Memory; when P/$\overline{D}$ = "0", the memory referenced is the Data Memory. The P/$\overline{D}$ output signal is defined at the beginning of the memory cycle and is stable until the next Memory cycle. It is enabled by software as the Alternate Function output of a parallel port bit (refer to specific ST9 version to identify the specific port and pin).

Under Reset status, the associated bit of the port is set into bidirectional weak pull-up mode. To enable this function, the program must set the port bit as an Alternate Function.

$\overline{WAIT}$

$\overline{WAIT}$ is sampled by the ST9 (once this function is enabled by setting EWEN (EIVR.0 R246 page 0) with the falling edge of the processor internal clock during phase T2 of every machine cycle. If the signal was sampled active, one more internal clock cycle is added to the memory cycle; on the falling edge of the added clock cycle, $\overline{WAIT}$ is sampled again to continue or finish the memory cycle stretching (see paragraph 6.4).

$\overline{BREQ}$, $\overline{BACK}$

Once enabled by setting BRQEN (MODER.1 R235), $\overline{BREQ}$ is sampled by the ST9 upon the falling edge of the internal clock during the phase T3. When the $\overline{BREQ}$ signal is sampled low, the CPUCLK clock is stretched and the External Memory signals ($\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, P0, P1(, P6)) are released to high-impedance. The input signal $\overline{BREQ}$ is then continuously monitored, and when it is sampled high the External Memory interface pins are driven again by the ST9 after two additional internal clock cycles. These cycles are used to fully drive and propagate the control and data signals through the external memory bus before CPUCLK is restarted.
The output signal $\overline{BACK}$ is driven low during the whole period when the External Memory interface is released to high impedance.

Under the Reset status, the bits of the I/O port(s) associated to $\overline{BREQ}$ and $\overline{BACK}$ are set to Bidirectional Weak pull-up mode and the enable bit BRQEN is cleared. To enable this function, the program must set the $\overline{BACK}$ port as an Alternate Function and enable (set to "1") the bit BRQEN.

Port 0. When Port0 is used as a parallel I/O port, it has the same features as a regular port (see I/O port chapter). When set as an Alternate Function, Port0 provides the Low Address bus (A0 to A7) and the Data bus (D0 to D7) to interface to external memory.

Port 1. When Port1 is used as a parallel I/O port, it has the same features as a regular port (see I/O port chapter). When set as an Alternate Function, Port1 provides the High Address bus (A8 to A15 to interface to external memory.

Port 6. When Port6 (if available) is used as a parallel I/O port, it has the same features as a regular port (see I/O port chapter). When set as an Alternate Function, Port6 provides the Low Address bus (A0 to A7) to interface to external memory with a non-multiplexed bus.

Each memory memory cycle is composed of three CPUCLK phases: T1, T2, T3. During phase T1, the memory address is output upon $\overline{AS}$ falling edge and is valid upon the rising edge of $\overline{AS}$. Port1 and Port6 maintain the address stable until the next T1 phase.
If the Memory access cycle is a Read cycle, Port0 pins are released to high impedance with the falling edge of $\overline{DS}$ until the next $\overline{AS}$ falling edge.
If the Memory access is a Write cycle, Port0 is held active, the data is output during T2 and is maintained until the next address is output (upon the falling edge of $\overline{AS}$). DS is pulled low during T2 only if the Memory access is an External Memory access. If the memory cycle is a Memory Read, it is pulled low at the beginning of T2. If it is a Memory Write, $\overline{DS}$ is kept low from the middle of T2 until the middle of T3.

6.4 STRETCHED ACCESS CYCLE

The ST9 can interface to memory with slow access times by automatically inserting additional Wait cycles during the External Memory cycle. On-chip memory accesses do not require $\overline{WAIT}$ cycles and run at the full speed of CPUCLK.

Three Wait cycle sources are available:

- The input pin WAIT from external sources
- The internal programmable Wait cycle generator
- Internal memories with stretched access cycle (EEPROM)

The input pin $\overline{WAIT}$ (when enabled) is sampled on the CPUCLK falling edge of phase T2. If active (low), one INTCLK clock cycle will be added. During the added clock cycle, the $\overline{WAIT}$ pin is sampled again. CPUCLK is stretched for as long as the $\overline{WAIT}$ input is active.

The internal programmable $\overline{WAIT}$ cycle generator allows the extension of the External Memory cycle automatically by the programmed number of $\overline{WAIT}$ cycles. Two three bit fields in the $\overline{WAIT}$ Control Register WCR (R252 page 0) allow the stretching of Program and Data Memory access cycles independently by 0 to 7 cycles. The three least significant bits WPM2,1,0 (WCR.5,4,1) contain the number of Program memory wait cycles to be added, WDM2,1,0 (WCR.2,1,0) contain the number of Data memory wait cycles to be added.

	WDM2 WPM2	WDM1 WPM1	WDM0 WPM0	Nb of Clock Cycles Added	
min	0	0	0	0	- No Wait Cycle
	0	0	1	1	
	0	1	0	2	
	0	1	1	3	
	1	0	0	4	
	1	0	1	5	
	1	1	0	6	
max	1	1	1	7	- Reset Value

WCR R253 Page 0 Read/Write
Wait Control Register

Reset Value: 0111 1111 (7Fh)

7							0
X	WDGEN	WDM2	WDM1	WDM0	WPM2	WPM1	WPM0

b7 = Reserved, reads as a "0".

b6 = **WDGEN:** refer to Timer/Watchdog chapter.

b5-b3 = **WDM2-0:** *Data Space Wait Cycles.* These bits contain the number of INTCLK cycles to be added automatically to external Data memory accesses. (WDM = 0 gives no additional wait cycles, WDM = 7 provides the maximum 7 INTCLK cycles (this is the reset condition in order to allow the use of slow access time external memory, if faster memory is used, then this value may be modified by the User).

b2-b0 =**WPM2-0:** *Program Space Wait Cycles.* These bits contain the number of INTCLK cycles to be added automatically to external Program memory accesses. (WPM = 0 gives no additional wait cycles, WPM = 7 provides the maximum 7 INTCLK cycles (this is the reset condition in order to allow the use of slow access time external memory, if faster memory is used, then this value may be modified by the User).

Note the number of clock cycles added refer to INTCLK and NOT to CPUCLK.

Figure 6-4. External Memory Read/Write Sequence With External Wait

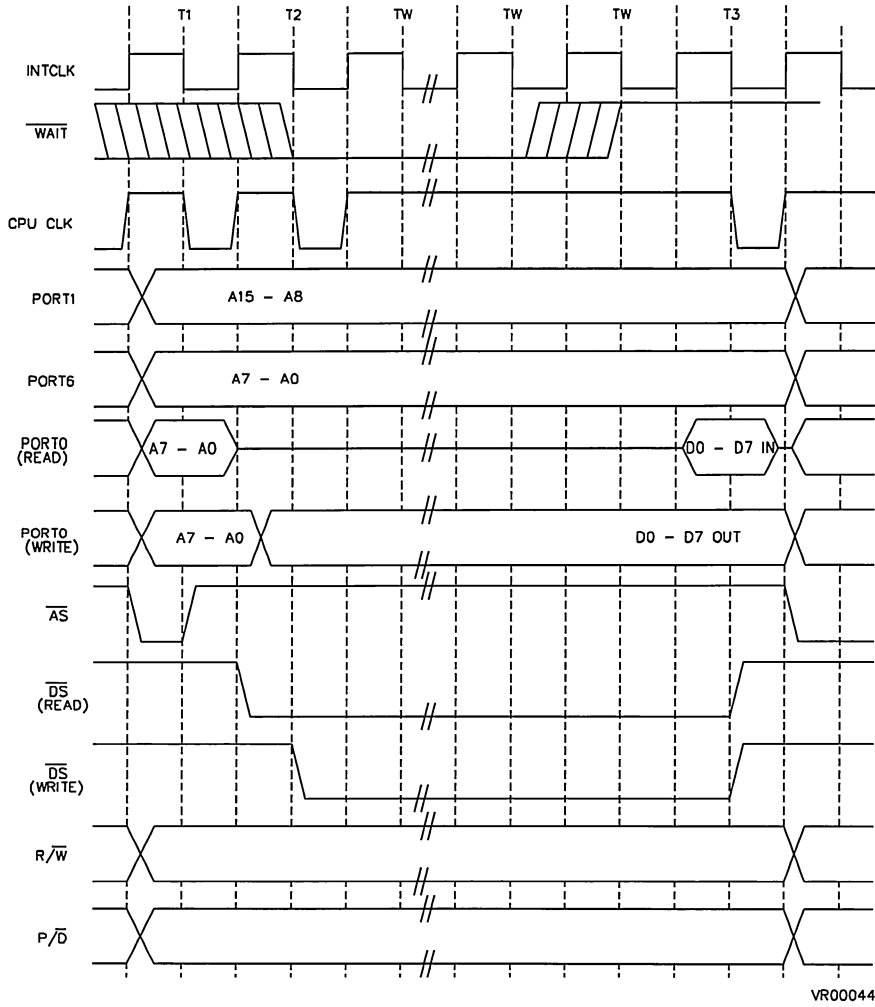
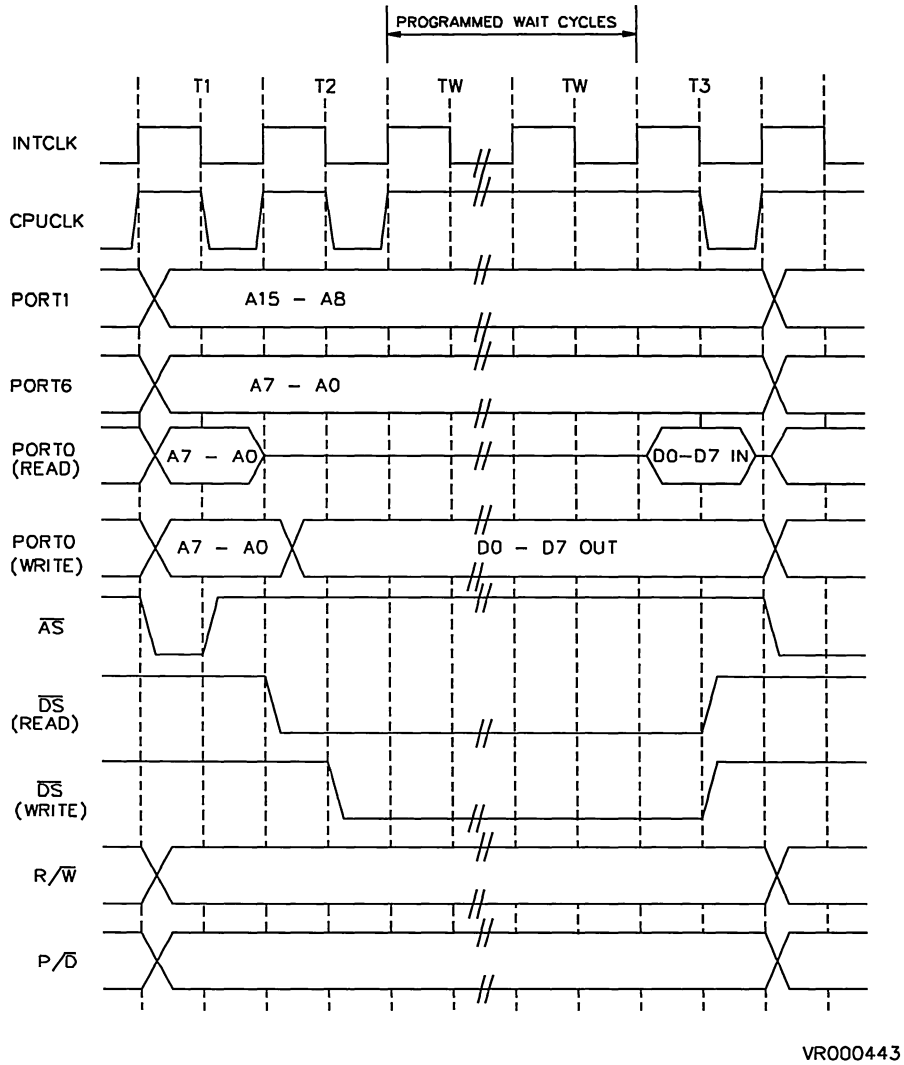


Figure 6-5. External Memory Read/Write with a Programmable Wait



6.5 SHARED BUS

When the ST9 runs in a multi-master bus system, it is necessary to release the bus control to other bus master(s). This operation can be performed by the Bus Request/Acknowledge capability supported by the ST9.

When it is required to disable the external bus, but to keep the processor running in the on-chip memory, the external memory bus can be disabled by software programming of the HIMP (MODER.0) control bit. By setting HIMP, the External Memory Interface ($\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ and Port0, Port1 and Port6 if not configured as I/O lines) is set into a high impedance state. In this way, the external memory bus can be used by another resource (e.g. diagnostic equipment or external programming of system memories) and the ST9 program can continue accessing the on-chip memory. This feature can also be useful for high security applications where the flow and addresses of the on-chip security algorithms must not be shown on the external address pins.

The disabling of the External Memory Interface by setting HIMP = "1" can be interrupt driven by applying the "Bus Request" input signal to an External Interrupt pin. In this case the bus disable response time will be longer than the automatic system using the $\overline{BREQ}$ request, however the ST9 can continue to execute the program written in the on-chip memory.

6.6 PORTS P0, P1, P6 INITIALISATION AFTER RESET

The Port 0, Port 1 and Port 6 initialisation after reset depends on the configuration of the ST9 as shown in Table 6.1.

Table 6-1.

ST9 Device	Port 0, 1, 6 Initialization
ROM, EPROM	Bidirectional Weak-Pull-Up (PxC0,PxC1,PxC2 = 0,0,0 ; Data = 1)
ROMLESS	Memory Address and Data Alternate Function Push-Pull (PxC0,PxC1,PxC2 = 1,1,0 ; Data = 1)

If the device has on-chip Program memory (ROM or EPROM), the ports (or the existing parts of them) are set to Bidirectional Weak Pull-up Mode.

If the device is ROMLESS or a ROM device with the ROMless function enabled, the ports (or the existing part of them) are set to Alternate Function Push-Pull Mode, providing the Address and Data lines to interface to the external Program and Data Memory from Reset.

6.7 ROMless FUNCTION

In order to accomodate the use of ROM based ST9 devices in the event of a subsequent ROM code change, a ROMless function may be enabled on a specified Port I/O pin by Mask Option. This function is activated by pulling the ROMless select pin to ground with a 100k resistor. This status is latched on the rising edge of the $\overline{RESET}$ pin and, when low, the on-chip PROGRAM memory (ROM or EPROM) is disabled, causing all instruction fetch cycles to be external. On-chip Data memory (RAM or EEPROM) is not affected.
If the ROMless function is enabled by the mask option, and the internal program is to be used, then the ROMless pin must be held to a high level (via a 100k resistor to V_{DD}) during the Reset cycle. After the Reset cycle the ROMless pin may be programmed for any I/O or Alternate function.

6.8 BANKSWITCH LOGIC

The Bankswitch (BS) logic of the ST9 family is an address expander allowing the ST9 extend its addressing range from 64K Program space + 64K Data space to up to 8 Megabytes of Program memory plus 8M bytes of Data memory organised in 32K byte segments mapped into the address 8000h to FFFFh, and a 32K byte common segment in the address range 0000h to 7FFFh.

This expansion is achieved using the 8 bit data present on the I/O Port mapped as the Bankswitch port as address bits A23:A16, and internally using A15 as the control signal to control the common segment (seg0) selection.
An alternative use of the Bankswitch Port is to use the bits directly as chip selects to external memory. For this reason, the Bankswitch Port has the value FEh after the Reset State and whenever address bit A15 is low (indicating the common segment), this allows the initialisation program and common subroutines to be held in the external memory mapped to bit 0 of the Bankswitch output port.

The data present on the Bankswitch port (i.e. Addresses A23 to A16) changes according to the following conditions:

- the address is in Program memory
- the address is in Data memory
- a DMA transaction is being made with Program memory
- a DMA transaction is being made with Data memory
- the common segment is being addressed

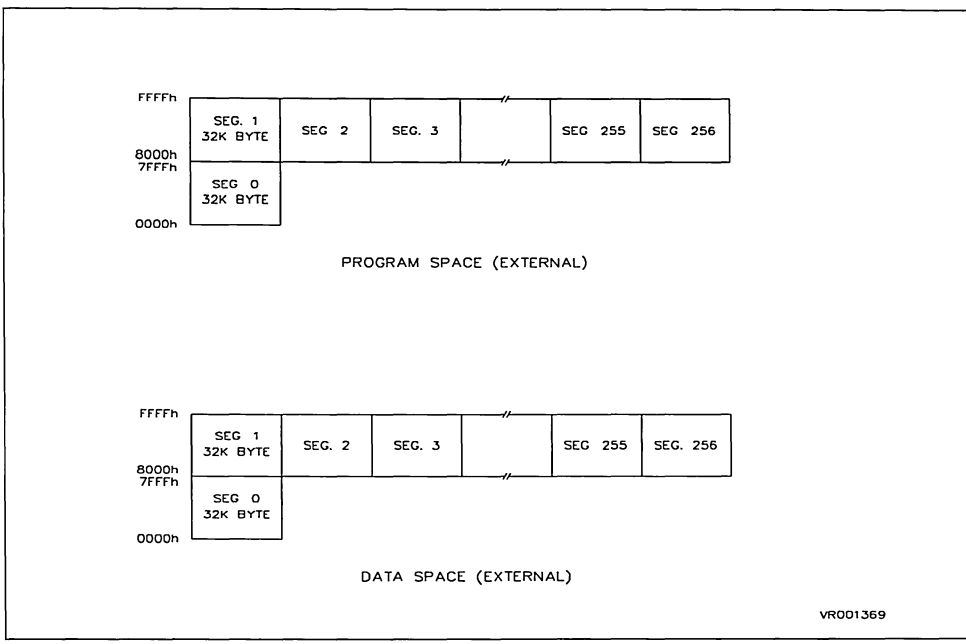
The Bankswitch logic includes 4 registers whose contents are user definable as the segment numbers to be used with these memory addressing conditions as shown in the following table:

- BS_PSR when A15 = "1" and the segment is in Program Memory
- BS_DSR when A15 = "1" and the segment is in Data Memory
- BS_PDSR when A15 = "1" and DMA is using Program Memory
- BS_DDSR when A15 = "1" and DMA is using Data Memory

also

- BS Port = FEh whenever A15 = "0"

Figure 6-6. Bankswitch Memory Maps



6.8.1 Bankswitch Register Mapping

When the Bankswitch port is used as an I/O port, the data and control registers are mapped as I/O Port 2 in the System page E (for the Data Register) and Page 2 (for the Control Registers). Refer to figure 6-7, for the Bankswitch Register Mapping.

When the Bankswitch function is enabled, the four Bankswitch Registers are mapped into Page 2 (BS_PDSR and BS_DDSR) and in System Group E (BS_PSR and BS_DSR). The mapping of the Program and Data Memory Bankswitch Registers into Group E optimises the software overhead during memory segment changes, while the mapping of the DMA Bankswitch registers in an I/O page does not give a great overhead as, once initialised, the values are used automatically.

6.8.2 Bankswitch Port Programming

The Bankswitch Port functionality can be nibble (4bits or half a byte) programmed as Bankswitch outputs or I/O by latching (in the Reset cycle) the state of the Alternate Function pins BSH_EN1 and BSL_EN1.

These port pins are set to INPUT CMOS status during the Reset cycle so programming is made through the use of external pull-up or pull-down resistors. After the Reset cycle both port pins can be independently reprogrammed as any other I/O pin.

The programming configurations 01 and 10 both enable the lower nibble as active Bankswitch outputs.

Figure 6-7. Bankswitch Register Mapping

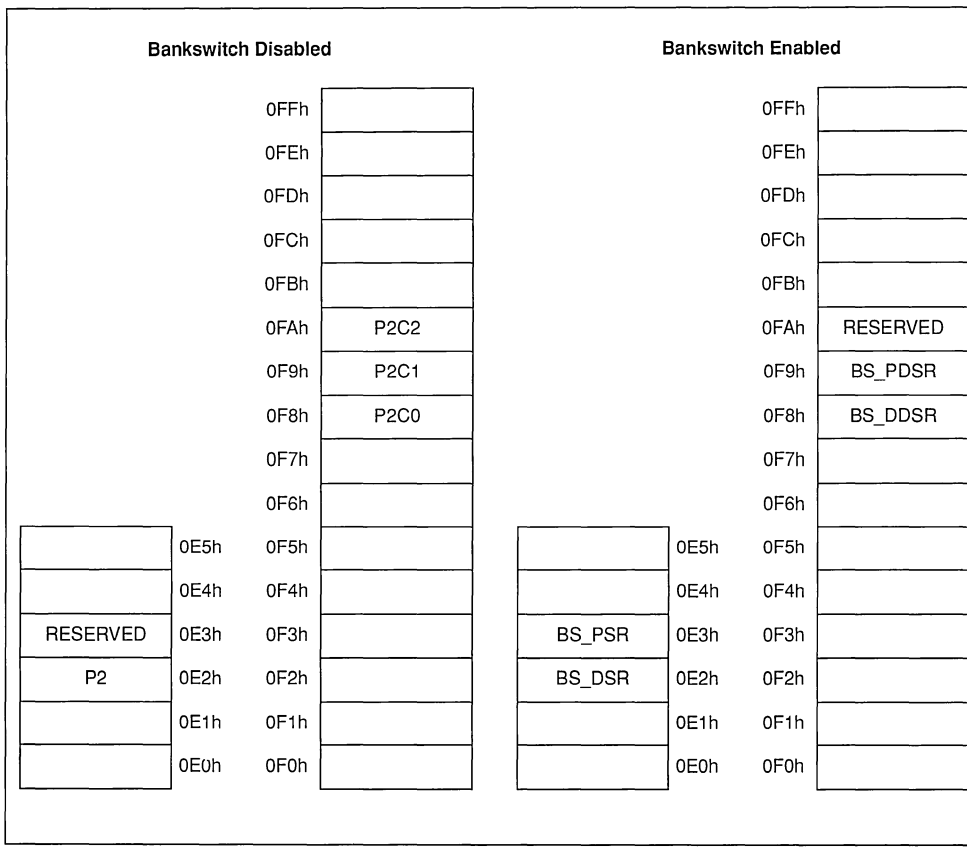


Table 6-1. Port 2 Nibble Programming for Bankswitch and I/O

BSH_EN1	BSL_EN1	BS Port Nibble		BS Port Reset Value
		High	Low	
0	0	I/O	I/O	0FFh
0	I	I/O	BS	0FEh
I	0	I/O	BS	0FEh
I	I	BS	BS	0FEh

WARNING *BS_PSR and P2C2 share the same physical register, so that when the Bankswitch port is nibble programmed, caution must be taken when writing to this register. If the lower nibble is programmed as Bankswitch and the upper as I/O, programming of the Bankswitch register with a byte value can cause the erroneous reconfiguration of the I/O nibble.*

6.8.3 Output Timing

In order to prevent potential bus conflicts on Port 0 (Address/Data multiplexed) during the address strobe time when using the Bankswitch logic, the timing of the Bankswitch outputs may be modified by software. This is achieved by setting to "1" bit 1 of Register 0FFh in I/O page 0. This causes the Bankswitch outputs to be all high during the address strobe period. The reset condition provides normal timing and status. Refer to figure 6-8.

Also the timing of the Read/Write signal may be modified as shown in figure 6-9 by setting to "1" bit 0 of Register 0FFh in I/O page 0. This allows the use of different types of external memories. When this bit is "0" (the reset state) normal timing is generated.

Note: The LST9 ST9 Incremental Linker supports the paging mechanism of the Bankswitch and is able to allocate program and data code into specific segments if required

Figure 6-8. Bankswitch Output Timing Modification

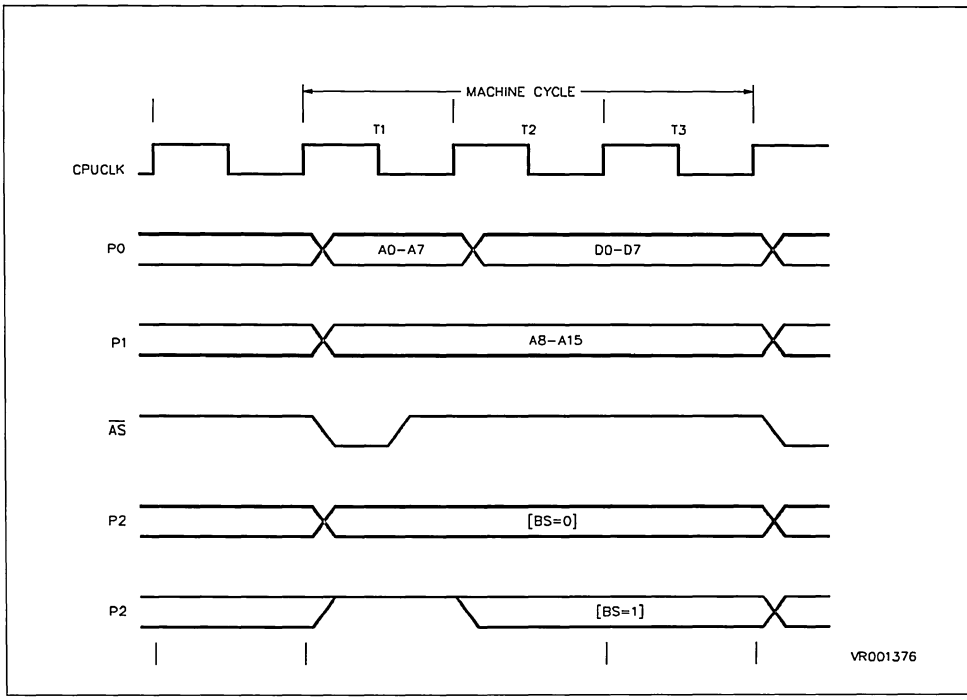
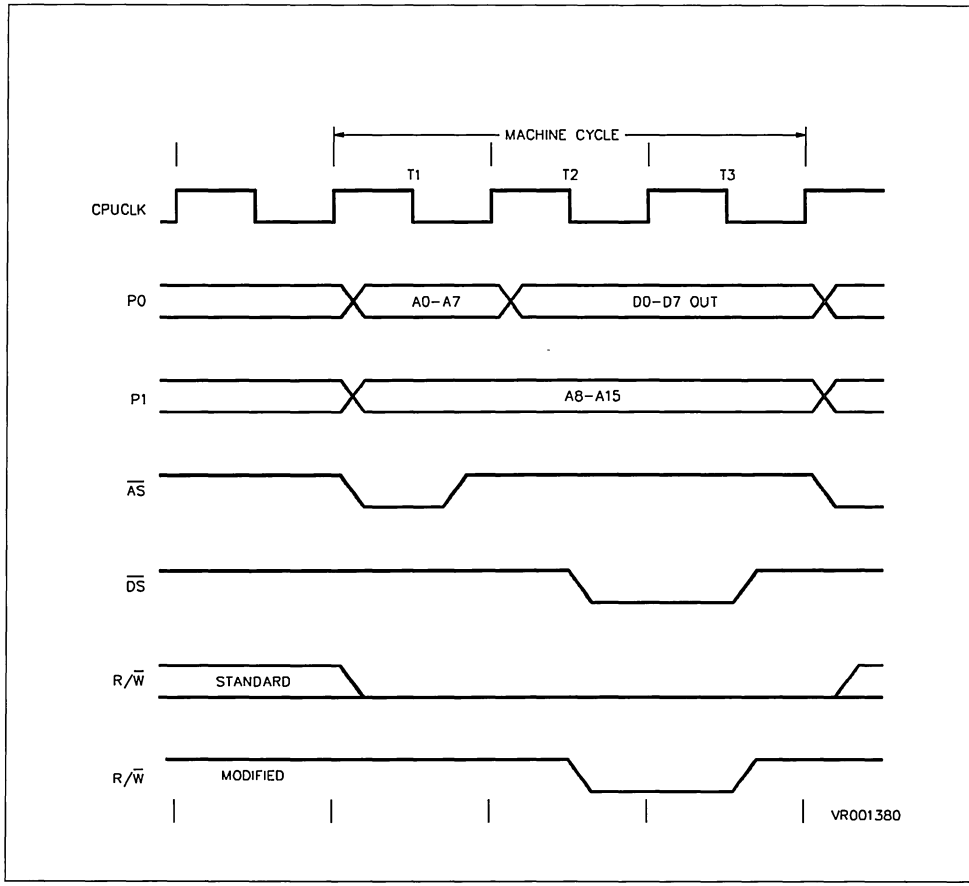


Figure 6-9. R/$\overline{W}$ Output Timing Modification

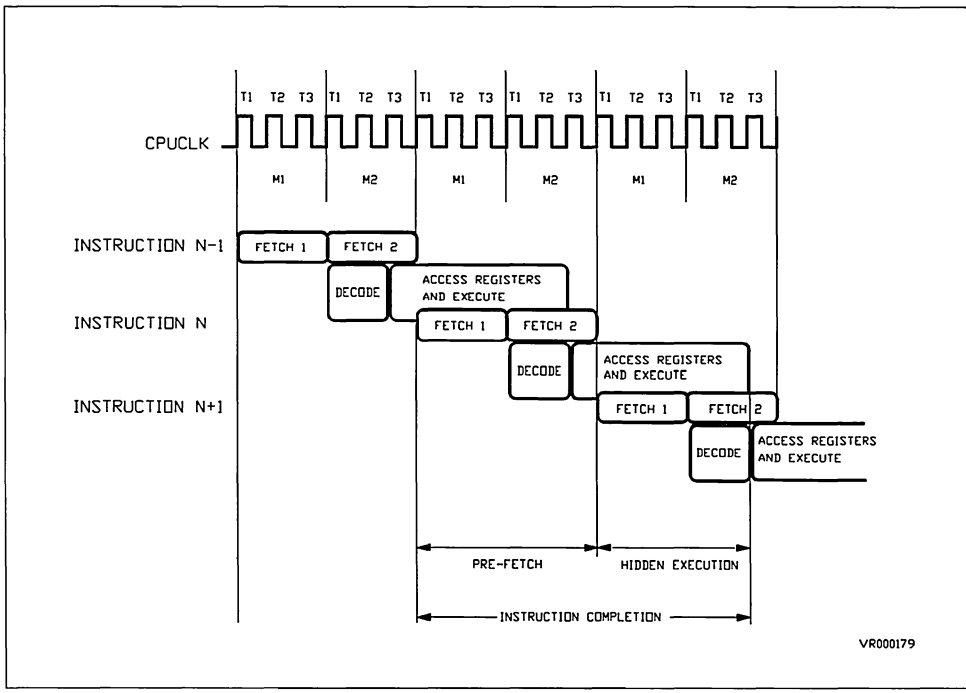


6.9 ST9 PIPELINE

The ST9 implements pipe-line stages on instruction fetch and execution in order to increase the execution speed. The instruction execution is in fact hidden by the Memory access cycles: the execution of one instruction is overlapped with the pre-fetch of the two successive bytes. The fetch of the first byte (opcode) is identified by the machine cycle M1, the fetch of the second byte by M2.

The 2 bytes instructions, whose execution time is 6 CPUCLK cycles, have the instruction execution hidden by the following instruction prefetch. For those instructions that require an execution time longer than the time to prefetch the following bytes, perform memory access during their execution or interrupt the sequential memory access, the pipe is flushed.

Figure 6-10. Instruction Pipe Line Stages



6.10 "SPURIOUS" MEMORY ACCESSES

The ST9 in certain cases produces external memory accesses which may be regarded as "Spurious" in their nature. While these do not affect the correct operation of the ST9, these accesses may cause misunderstandings when developing and debugging applications as the signals $\overline{AS}$ and $\overline{DS}$ are produced, and Ports 0, 1, and 6 output updated addresses (if used to interface to external memory).

The spurious reading cycle is produced when executing specific instructions. This is one of 4 cases: double reading, reading before writing, reading when the stack is internal or prefetch reading.

– DOUBLE READING A memory location read by the ST9 is read two times consecutively (instead of one).

Involved instruction(s):

`DIVrr,r` ; divide (16/8)

The first byte of the code following DIV is fetched two times. The double reading does not occur if the Overflow flag was set by DIV, or if Divide by zero was trapped. The P/$\overline{D}$ line remains high during the cycle.

– READING BEFORE WRITING A memory location which is to be written to by the ST9 is previously read.

Involved instruction(s):

`LD(rr)+,(r)+`; load (byte) Memory, Register

The destination memory location is read before being written. The P/$\overline{D}$ line reflects the memory space of the destination memory location.

– READING WHEN THE STACK IS INTERNAL If the System and/or User Stack has been programmed to use the Register File, a memory location of Data Space is POPed in parallel.

Involved instruction(s):

```
POP     R   ; POP (byte) from System Stack
POP     (R) ;      "     "     "
POPU    R   ; POP (byte) from User Stack
POPU    (R) ;      "     "     "
```

While a byte is being `POP`ed from the Register File, a memory location in Data Space is read in parallel with its address given by SSPHR+SSPLR for `POP` instructions and by USPHR+USPLR for `POPU` instructions. The external data is ignored.

```
POPW    RR ; POP (word) from System Stack
POPUW   RR ; POP (word) from User Stack
```

While the higher address byte is being popped from the Register File, a memory location in Data Space is read in parallel with its address given by SSPHR+SSPLR for `POPW` instructions and by USPHR+USPLR for `POPUW` instructions. No spurious reading is made for the lower byte.

`RET`; Return from Subroutine

While the Program Counter Higher and Lower bytes are POPed from the Register File, two memory locations are read at addresses given by SSPHR+SSPLR.

`IRET`; Return from Interrupt

While the Program Counter Higher and Lower bytes and the FLAGS are `POP`ed from the Register File, three memory locations are read at addresses given by SSPHR+SSPLR. When working with Internal Stacks, SSPHR and USPHR contents are don't care from the point of view of program execution, but they must be considered RESERVED by the User as the instructions listed in this section perform updating of SSPHR/USPHR, together with the spurious reading.

– PREFETCH READING Due to the ST9 Pipeline, instructions which stop the Core or which perform program branches can fetch bytes of the following program code while the pipeline is being flushed.

Involved instruction(s):

`WFI`; Wait For Interrupt

reads two bytes of the following code.

`HALT`; Halt `CALL(rr)`; Unconditional Call subroutine

read one byte of the following code in Program space (P/$\overline{D}$ high).

SERIAL PERIPHERAL INTERFACE

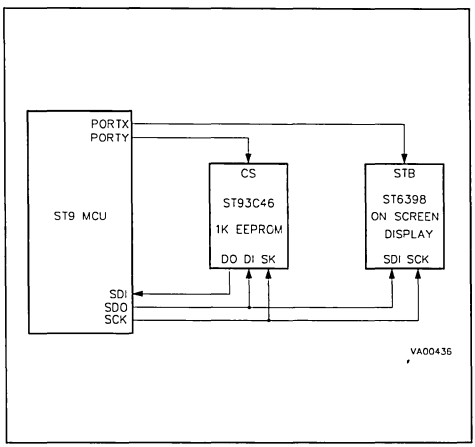
7.1 SPI FEATURES

The ST9 Serial Peripheral Interface (SPI) allows several external peripherals to be linked through an SPI protocol bus, as well as, with reduced software overhead, other different protocols: S-bus, I^2C-bus and IM-bus.

Its Main Features are:

- Full duplex 3-wire synchronous transfer
- Master operation only
- 1.5MHz max bit transfer frequency (INTCLK = 12MHz)
- 4 Programmable bit rates
- Programmable clock polarity and phase
- Busy Flag
- End of transmission interrupt
- Additional hardware to facilitate more complex protocols

Figure 7-1. A Typical SPI Network



7.2 FUNCTIONAL DESCRIPTION

A block diagram of the Serial Peripheral Interface (SPI) is shown in Fig 7.2.

The SPI (when enabled) receives input data from the ST9 Core (into SPIDR) and originates the Serial Clock (SCK) based upon dividing of the internal processor clock (INTCLK). The data is parallel loaded into the 8 bit shift register (from the internal bus) during a write cycle and then shifted out serially through the Serial Data Out pin (SDO) to the slave device which responds by sending its data to the master device via the SDI pin. This implies full duplex transmission with data-out and data-in both synchronized with the same clock signal. Thus the transmitted byte is replaced by the byte received, eliminating the need to separate "Tx empty" and "Rx full" status bits.

When the shift register is loaded, data is parallel transferred to the read buffer and data becomes available for the ST9 during a read cycle.

7.3 SIGNAL DESCRIPTION

The SPI requires three alternate function pins on an I/O port:

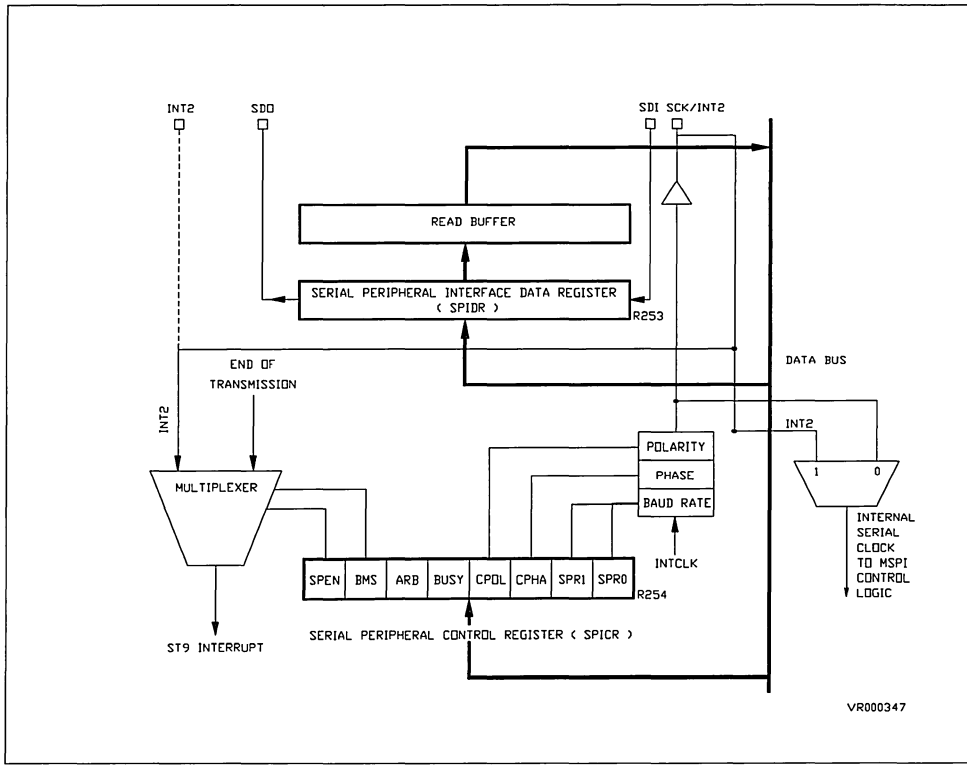
SCK . . . . . Serial Clock signal
SDO . . . . . Serial Data Out
SDI . . . . . Serial Data In

An additional output bit of an I/O port may be used to perform the slave chip select signal.

Serial Data Out (SDO)

The SDO pin is configured as an output for the master device. This is obtained by programming the corresponding I/O pin as an output alternate function. Data is transferred serially from a master to a slave on SDO, most significant bit first. This pin is forced to the high impedance state when the SPI is disabled and is set to "1" when arbitration is lost (during an S-bus/I^2C-bus protocol transmission).The master device always allows data to be applied on the SDO line one half cycle before the clock edge in order to latch the data for the slave device.

Figure 7-2. SPI Block Diagram



Master Serial Clock (SCK)

The master device uses SCK to latch the incoming data on the SDI line. This pin is forced to a high impedance state when SPI is disabled (SPEN, SPICR.7 = "0"), in order to avoid clock contention from different masters in a multi-master system.
The master device generates SCK from INTCLK. SCK is used to synchronize the transfer of data both in and out of the device through its SDI and SDO pins. The SCK type and its relationship to data are controlled by the CPOL and CPHA bits in the Serial Peripheral Control Register.
This input is provided with a digital filter which cleans spikes lasting less than one INTCLK period.

Two bits (SPR1 and SPR0) in the Serial Peripheral Control Register, SPICR (R254) select the clock rate. Four frequencies can be selected, two in a high frequency range (mostly used with the SPI protocol) and two in a medium frequency range (mostly used for more complex protocols).

Serial Data In (SDI)

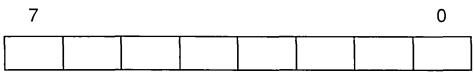
Data is transferred serially from a slave to a master on this line, most significant bit first. In an S-BUS/I^2C-bus configuration, SDI line senses the value forced on the data line (by SDO or by another peripheral connected to the S-bus/I^2C-bus environment).

7.4 SPI REGISTERS

SPI uses two registers mapped on page 0 of the register file:

SPIDR-R253 (FDh) Page 0 Read/Write
SPI Data Register (R253)

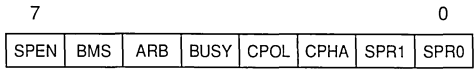
Reset Value: 0000 0000b (00h)



b7-b0 = **SPI Data Register**. This register contains the data transmitted and received by the SPI. Data is transmitted b7 first, and receives incoming data into b0. Transmission is started by writing to this register.

SPICR-R254 (FEh) Page 0 Read/Write
SPI Control Register (R254)

Reset Value: 0000 0000b (00h)



b7 = **SPEN:** *Serial Peripheral Enable*. When set, the two alternate functions SCK and SDO are enabled. When disabled, SCK and SDO are kept in high impedance. Furthermore, SPEN affects the selection of the source for interrupt channel B0. Transmission will start by simply writing the data into the SPIDR Register (see paragraph 7.5).

b6 = **BMS:** *S-bus/I²C-bus Mode Selector*. This bit should be set to "1" when the SPI is used in an S-bus/I²C-bus protocol. It enables S-bus/I²C-bus arbitration, clock synchronization and Start/ Stop detection (refer to paragraph 7.6 for more details). When this bit is reset to "0", a reinitialisation of the SPI logic is performed allowing recovery procedures after a R_X/T_X failure. BMS (and SPEN) affects the selection of the source for interrupt channel B0 (see paragraph 7.5)

b5 = **ARB:** *Arbitration flag bit*. This bit is set when the SPI, in S-bus/I²C-bus mode, loses arbitration, and is reset when an S-bus/I²C-bus stop condition is detected. ARB can be reset by software. When ARB is set automatically, the SDO pin is set to high value until a write instruction on SPIDR is performed.

b4 = **BUSY:** *SPI Busy Flag*. BUSY flag is set when a transmission is in process. This bit allows the user to monitor the SPI status by polling its value.

b3 = **CPOL:** *Transmission Clock Polarity*. CPOL controls the normal or steady state value of the clock when data is not being transferred.

As the SCK line is held in a high impedance state when the SPI is disabled (SPEN = "0"), the SCK pin must be connected to V_{SS} or V_{CC} through a resistor according to the CPOL state. Polarity should be selected during the reset routine according to the value set into all peripherals and must not be changed during program execution.

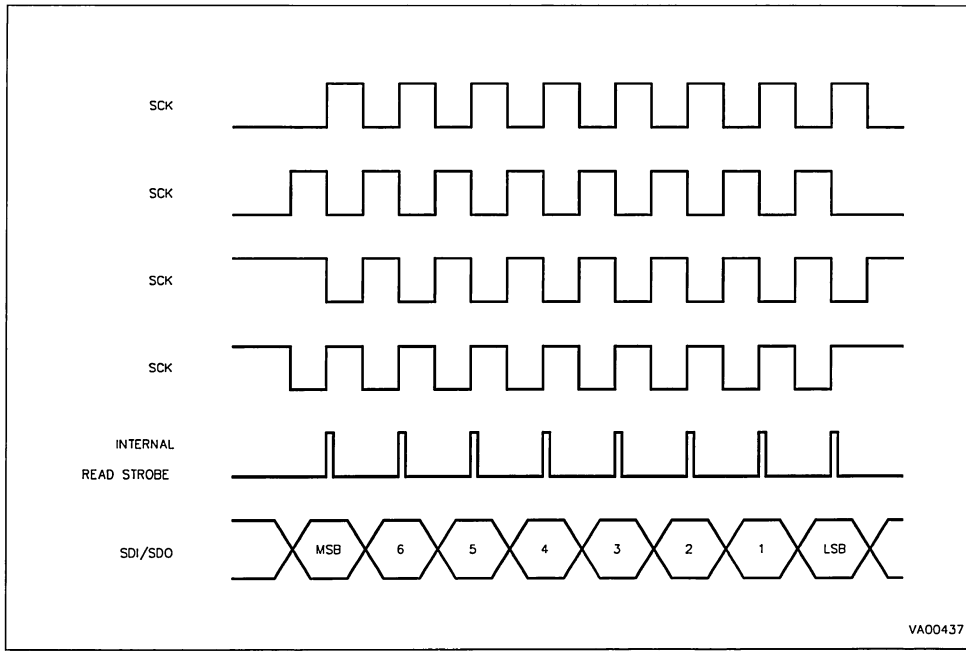
CPOL	CPHA	SCK on Fig. 7-3
0	0	(a)
0	1	(b)
1	0	(c)
1	1	(d)

b2 = **CPHA:** *Transmission Clock Phase*. CPHA controls the relationship between the data on the SDI and SDO pins and the clock produced at the SCK pin. CPHA bit selects the clock edge which captures data and allows it to change state. It has its greatest impact on the first bit transmitted (MSB) because it does (or does not) allow a clock transition before the first data capture edge.
Figure 7.3 shows the relationship between CPHA, CPOL and SCK, and indicates active clock edges and strobe times.

b1-b0 = **SPR1,SPR0:** *SPI Rate*. These two bits select one (out of four) baud rates to be used as SCK.

SPR1	SPR0	Clock Divider	SCK Frequency (INTCLK = 12MHz)
0	0	8	1500kHz (T = 0.67µs)
0	1	16	750kHz (T = 1.33µs)
1	0	128	93.75kHz (T = 10.66µs)
1	1	256	46.87kHz (T = 21.33µs)

Figure 7-3. SPI Data and Clock Timing



7.5 INTERRUPT STRUCTURE

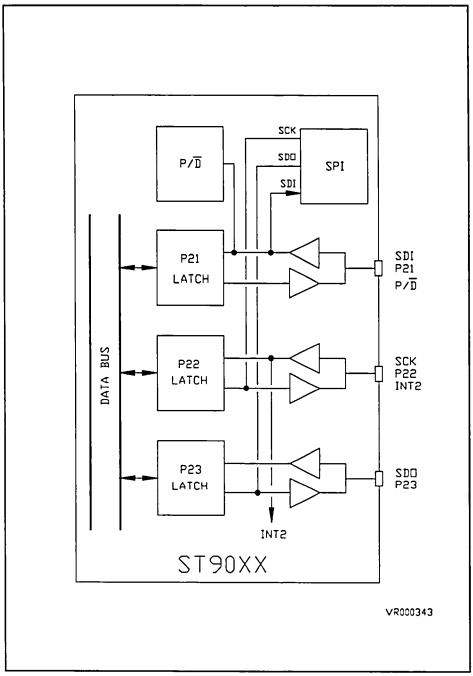
SPI peripheral is associated with external interrupt channel B0 (pin INT2) as described in chapter 4.5. Multiplexing between the external pin and SPI internal source is controlled by the SPEN and BMS bits according to the following table:

SPEN	BMS	Interrupt Source
0	0	External channel INT2
0	1	S bus/I²C bus start or stop condition
1	X	End of one byte transmission

The two possible SPI interrupt sources are: End of transmission (after each byte) and S-bus/I²C-bus start condition. Care should be taken when toggling SPEN or/and BMS bits from (0,0) status, this should be done by masking the interrupt channel B0 (reset of EIMR.IMB0, bit 2 of External Interrupt Mask Register). Furthermore it is necessary to clear possible spurious requests on the corresponding channel by resetting the interrupt pending bit EIPR.IPB0 (bit 2 of External Interrupts Pending Register).

The INT2 input Alternate Function is always mapped together with the SCK input Alternate Function to allow start/stop bit detection when using S-bus/I²C-bus protocols. A delay instruction (e.g. a NOP instruction) should be inserted between the SPEN toggle instruction and the interrupt pending bit reset instruction.

Figure 7-4. SPI I/O Pins



7.6 WORKING with DIFFERENT PROTOCOLS

The SPI peripheral offers the following facilities to work with S-bus/I²C-bus and IM-bus protocols:

- Interrupt request on start/stop detection
- Hardware clock synchronisation
- Arbitration lost flag with an automatic set of data line

The following paragraphs provide information to manage these protocols.

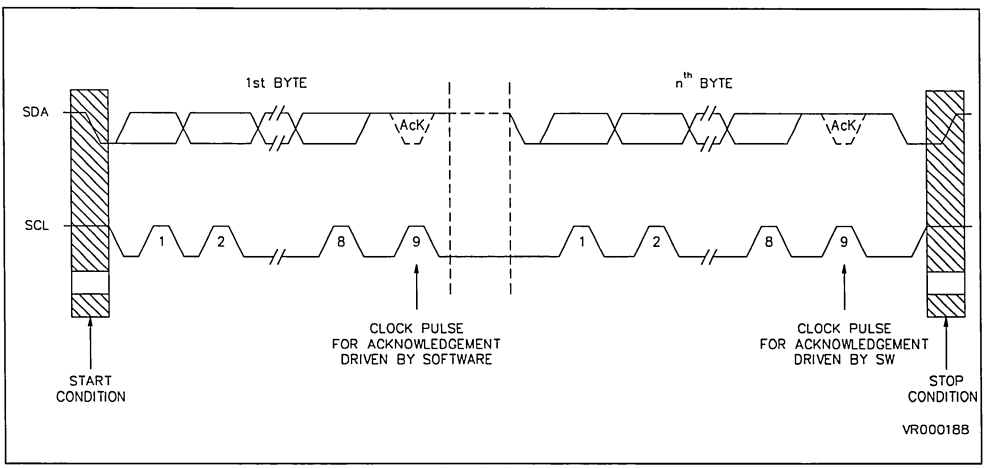
7.6.1 I²C-Bus Interface

I²C-bus is a two-wire bidirectional data-bus, the two lines being SDA (Serial DAta) and SCL (Serial CLock). Both are open drain lines to allow arbitration. As shown in figure 7.5, data is toggled with clock low and Start and Stop conditions are detected when a high to low (start) or a low to high (stop) transition on the SDA line occurs with the SCL line high.

Each transmission consists of nine clock pulses (SCL line). The first 8 pulses transmit the byte (msb first), the ninth is used by the receiver to acknowledge.

The data on the SDA line is sampled with the low to high transition on the SCL line.

Figure 7-5. I²C Bus Configuration



SPI Working With I²C-bus

To use the SPI with the I²C-bus protocol, the SCK line is used as SCL, the SDI and SDO lines, externally wired-OR'd, are used as SDA. All the pins must be configured as open drain (see figure 7.5).

Table 7-1 shows the typical I²C-bus sequence divided in 5 phases: initialize, start, transmission, acknowledge and stop.

Software and hardware will take care of each phase. A master to slave transmission can be managed as example according to the following table.

During the transmission phase, the following I²C BUS features are also supported by hardware.

Clock Synchronization

In a multimaster I²C-bus system, when more masters generate their own clock, synchronization is needed. The first master which releases the SCL line stops internal counting, restarting only when the SCL line goes high (released by all the other masters). In this way, devices using different clock sources and different frequencies can be interfaced.

Figure 7-6. S-Bus/I²C Bus Peripheral Compatibility Without S Bus Chip Select

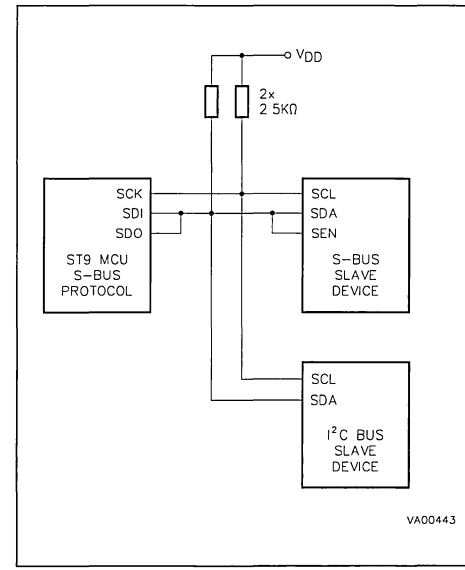


Table 7-1

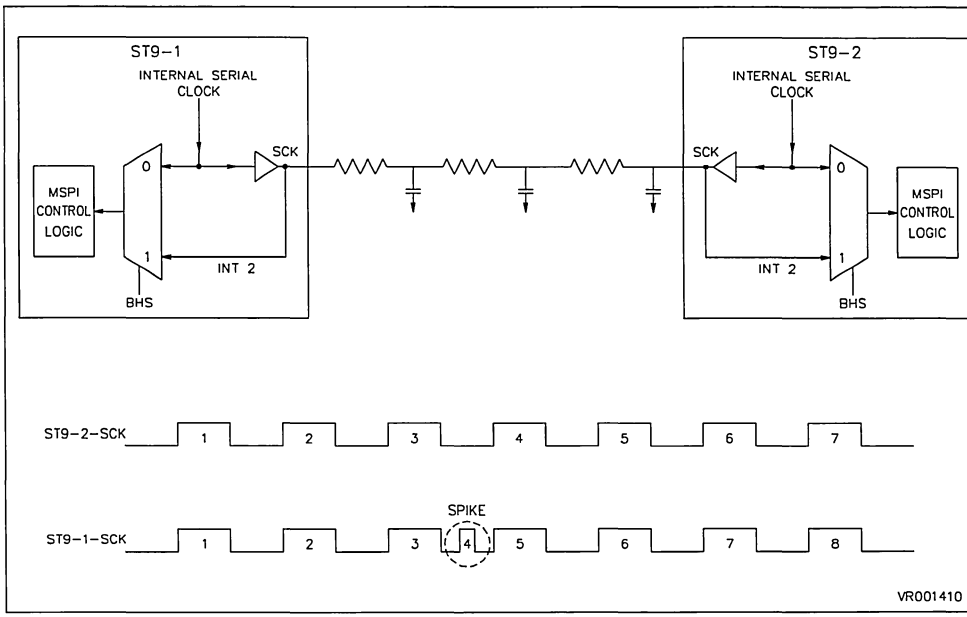
Phase	Software	Hardware	Notes
INITIALIZE	SPICR.CPOL, CPHA = 0, 0 SPICR.SPEN = 0 SPICR.BMS = 1 SCK pin set as AF output SDI pin set as input (*) Set SCK, SDO port bit to 1	SCK, SDO IN HI-Z SCL, SDA = 1, 1	Set polarity and phase SPI disable START/STOP interrupt Enable
START	SDO pin set as output Open Drain Set SDO port bit to 0	SDA = 0, SCL = 1 interrupt request	START condition receiver START detection
TRANSMISSION	SPICR.SPEN = 1 SDO pin as Alternate Function outupt load data into SPIDR	SCL = 0 Start transmission interrupt request	Managed by interrupt routine load FFh when receiving end of transmission detection
ACKNOWLEDGE	SPICR.SPEN = 0 Poll SDA line Set SDA line SPICR.SPEN = 1	SCK, SDO in HI-Z SCL, SDA = 1 SCL = 0	SPI disable only if transmitting only if receiving only if transmitting
STOP	Set SDO port bit to 1	SDA = 1 interrupt request	STOP condition

Arbitration Lost

When more masters are sending data on SDA line, the following mechanism is performed: if the transmitter sends a "1" and SDA line is forced low by another device the ARB flag (SPICR.5) is set and the SDO buffer is "switched off". (ARB is reset and SDO buffer is "switched on" when SPIDR is written to again). When BMS is set to "1" the peripheral clock is supplied through the INT2 line by the external clock line (SCL). Due to potential noise spikes (which must last longer than one INTCLK period to be detected), RX or TX may gain a clock pulse.

Referring to Figure 7.7, if ST9-1 detects a noise spike and gains a clock pulse, it will stop its transmission in advance and hold the clock line low causing ST9-2 to be frozen at the 7th bit. To exit and recover from this condition the BMS bit must be reset to "0", this will cause the reset of the SPI logic, aborting the current transmission. An End of Transmission interrupt is generated after this reset sequence.

Figure 7-7. SPI Arbitration



7.6.2 S-Bus Interface

S-bus is a three-wire bidirectional data-bus, with functional features similar to I^2C-bus. Differently from I^2C-bus, the START/STOP conditions are given by encoding the information on 3 wires instead of 2, as shown in Figure 7.8. The additional line is referred as SEN.

SPI Working With S-bus

The S-bus protocol uses the same pin configuration as I^2C-bus for generating the SCL and SDA lines. The additional SEN line is managed through a standard ST9 I/O port under software control (see Figure. 7.9).

Figure 7-8. Mixed S-bus and I^2C-bus system

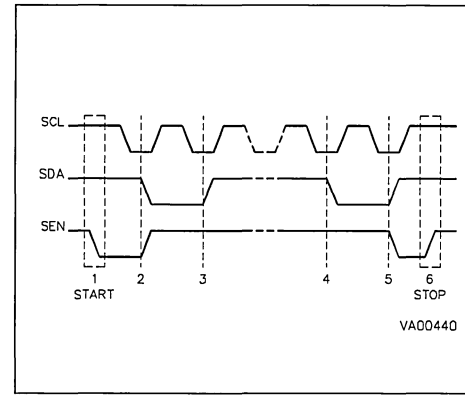


Figure 7-9. S-Bus Configuration

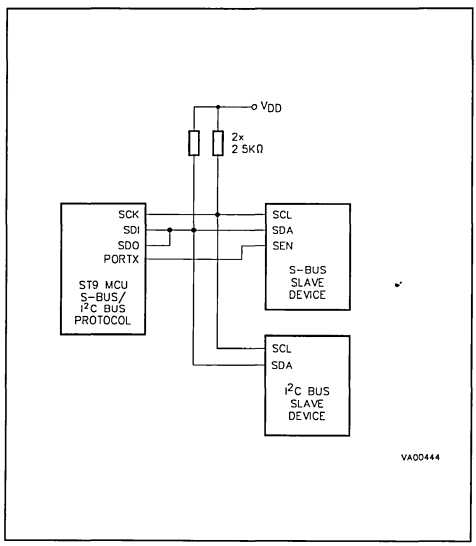
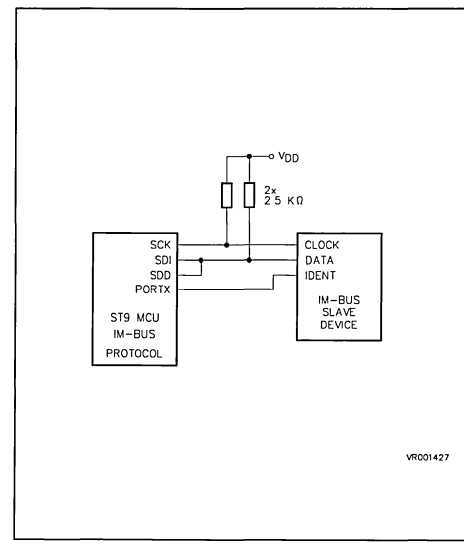


Figure 7-10. ST9 and Intermetal Peripheral



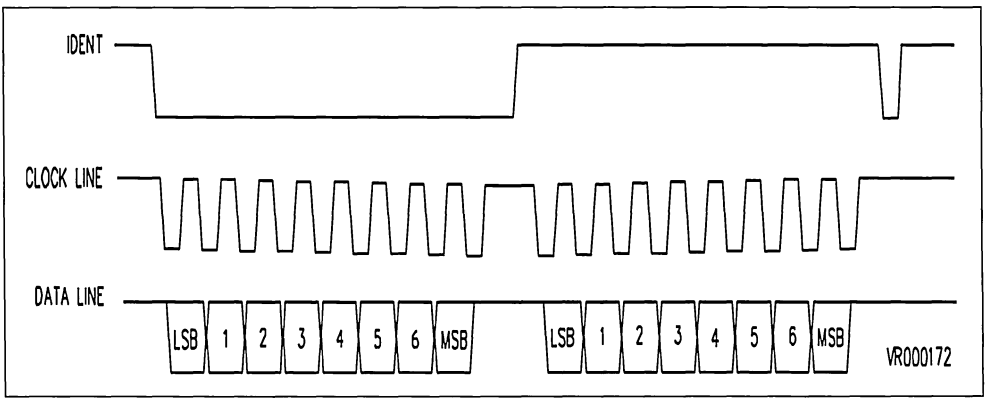
7.6.3 IM-Bus Interface

The IM-bus has a bidirectional data line and a clock line, and in addition it requires an IDENT line that distinguishes an address from a data byte (Figure 7.11). Unlike the I^2C-bus protocol, the IM-bus protocol sends the least significant bit first, this requires a software routine which reverses the bit order before sending, and after receiving a data byte. Fig 7.10 shows the connections for an IM-bus peripheral to an ST9 SPI. The SDO and SDI pins are connected to the bidirectional data pin of the peripheral device. The SDO alternate function is set in Open Drain (external 2.5K ohm pull-up resistors are required).

With this type of configuration, data is sent to the pripheral by writing the data byte to SPIDR. To receive data from the peripheral, the User should write FFh into SPIDR in order to generate the shift clock pulses. As the SDO line is set to the Open Drain configuration, the incoming data bits that are set to one do not affect the SDO/SDI line status (which defaults to a high level due to the FFh in the transmit register), while incoming bits that are set to "0" pull the input line low.

In software it is necessary to initialise the ST9 SPI with CPOL and CPHA set to "1", "1". By using a general purpose I/O as the IDENT line and forcing it to a logical "0" when writing to SPIDR, an address is sent (or read). Then by setting this bit to a logical "1" and writing to SPIDR, data is sent to the peripheral. When all the address and data pairs are sent it is necessary to drive the IDENT line low and high to create a short pulse. In this way the stop condition is generated.

Figure 7-11. IM bus TIMING



8.1 INTRODUCTION

The ST9 core includes a programmable 16-bit down counter with an 8-bit prescaler. The Timer/Watchdog can be programmed to be used as a Watchdog (in order to detect hardware or software failures) or as a Timer (with Single and Continuous counting modes).

The Timer/Watchdog functions may use inputs from an external pin and output as an Alternate-Function of an I/O Port. The Input pin can be used in one of the four programmable input modes:

- event counter,
- gated external input mode,
- triggerable input mode,
- retriggerable input mode.

The output pin can be used to generate a square or a Pulse Width Modulated signal.

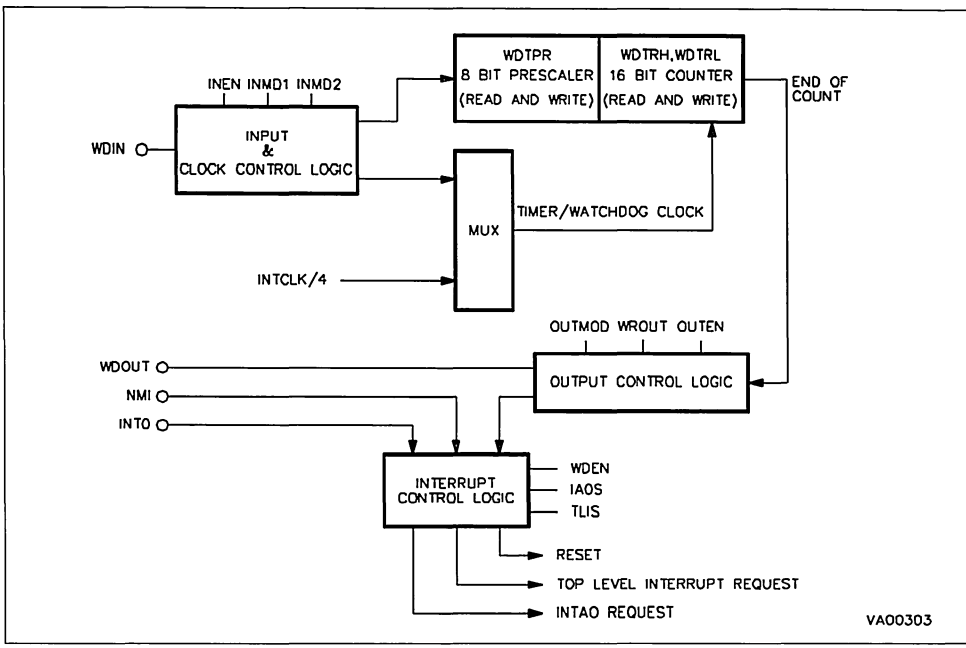
An interrupt generated by the unit (when running as a 16-bit Timer/counter and not as Watchdog) can be used as a Top Level Interrupt or as a source connected to channel 0 of the external interrupt structure.

The Timer is composed of a 16-bit down counter with an 8-bit prescaler. The clock for the counter can be driven either by an external clock or an internal clock equal to INTCLK divided by 4.
When using an external 24MHz crystal (INTCLK = 12MHz), the End Of Count rate is:

5.59 sec. for Max. Count (Timer Const. = FFFFh, Prescaler Const. = FFh)

333 nsec. for Min. Count (Timer Const. = 0000h, Prescaler Const. = 00h)

Figure 8-1. Timer/Watchdog Block Diagram



8.2. SELECTION OF WATCHDOG TIMER/COUNTER MODES

8.2.1 Timer/Counter Control

Start/Stop

ST_SP (WDTCR.7) enables down-counting. An instruction which sets this bit will cause the Timer to start at the beginning of the following instruction. Resetting this bit will stop the counter.

If the counter is stopped and restarted, counting will resume from the last value unless a new constant has been entered in the Timer registers. A new constant can be written with the counter running. The new value will be loaded at the following End Of Count (EOC).

WARNING: In order to prevent incorrect counting of the Timer/Watchdog, the prescaler (WDTPR) and counter (WDTRL, WDTRH) registers must be initialised before the starting of the Timer/Watchdog. If this is not done, counting will start with the reset (un-initialised) values.

Single/Continuous Mode

SINGLE MODE: At End Of Count the Timer stops, reloads the constant, and resets the Start/Stop bit (WDTCR.6) (user may check the current status by reading this bit). Restarting is done by setting the Start/Stop bit. Note that the Timer constant is reloaded only if it has been modified during the stop period.

CONTINUOUS MODE: At End Of Count the counter automatically reloads the constant and restarts. It is stopped only if the Start/Stop bit is reset. This Mode bit can be written with the Timer stopped or running. It is possible to toggle the Start/stop and start the counter with the same instruction.

8.2.2 Timer/Counter Input Modes

Setting the Input Enable (INEN) bit enables the input mode which is selected via the INMD1 and INMD2 bits. When INEN is reset to zero, the input section is disabled and the values of INMD1 and INMD2 are don't-care.

Event Counter Mode (INMD1 = "0", INMD2 = "0")

The Timer is driven by the signal applied to the input pin which acts as an external clock. The unit works therefore as an event counter. The event is a high to low transition of the input signal.
Spacing between trailing edges should be at least 333ns (i.e. the maximum Watchdog Timer input frequency is 3MHz with INTCLK = 12MHz).

Gated Input Mode (INMD1 = "0", INMD2 = "1")

The Timer uses the Watchdog internal clock (INTCLK divided by 4) and starts and stops the Timer according to the input pin. When the status of the Input pin is High the Timer Watchdog count operation proceeds, and when Low, counting is stopped.

Retriggerable Input Mode
(INMD1 = "1", INMD2 = "1")

A Timer/Watchdog start is caused by:
a) a set of the Start-Stop bit, or
b) a High to Low (low trigger) transition on the input pin.

In order to stop the Timer, it is only necessary to reset the Start-Stop bit to zero.

Triggerable Input Mode
(INMD1 = "1", INMD2 = "0")

In this mode when the Timer is running (TIMER/WATCHDOG internal clock), a High to Low transition of the input pin causes the counting to start from the initial value. When the Timer is stopped (ST_SP bit equal to zero), a High to Low transition of the input pin has no effect.

8.2.3 Watchdog Mode

In this mode (WDGEN = "0") the counter generates a fixed time basis. When End Of Count is reached the Timer generates a system Reset.

The time base is user-defined and must be written in the Timer registers before entering Watchdog mode. In Watchdog mode it is possible to modify only the Prescaler Constant. This new value will be loaded when the counter restarts.
Resetting WDGEN (bit 6 of the Wait Control Register) causes the counter to start regardless of the value of the Start-Stop. In order to prevent a system reset the sequence AAh, 55h should be entered in WDTRL (Watchdog Timer register low). Once the writing of 55h has been performed the Timer reloads the constant and counting restarts from the preset value. The minimum time between the writing of the AAh and 55h codes is zero, i.e. the writing is sequential, and the maximum time is given by the Watchdog timeout period.

In Watchdog-mode a HALT instruction stops the CPU but does not stop the Watchdog Timer, which will cause a System Reset when reaching the End of Count. Furthermore ST_SP, S_C and input mode selection bits are "don't-care". Hence regardless of their status, the counter always runs in Continuous Mode driven by the internal clock.

The Output mode should not be enabled since that particular mode of operation is meaningless.

8.2.4 Timer/Watchdog Output Modes

OUTPUT modes are selected using 2 bits of WDTCR (R251): OUTEN (Output Enable) and OUTMD (Output Mode).

When OUTMD = "0", the Timer outputs a signal with a frequency equal to half the End Of Count repetition rate. With INTCLK = 12MHz, this allows generation of a square wave with a period ranging from 666ns to 11.18 seconds.

The value of the WROUT bit is transferred to the output pin at the End Of Count and the value is held until the next End Of Count when OUTMD = "1". This allows the user to generate PWM signals, by modifying the status of WROUT between End of Count events, based on software counters decremented on the Timer/Watchdog interrupt.

OUTEN = "1" enables the output function selected via OUTMD

When OUTEN = "0", the output is disabled and the output pin is held at a "1" level to allow several alternate functions on the same pin.

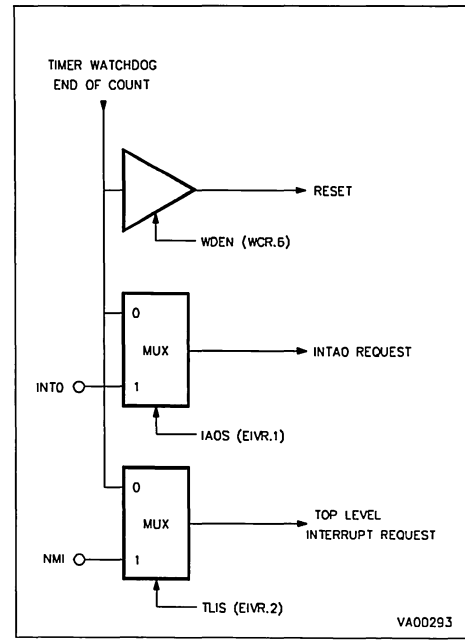
8.3 TIMER/WATCHDOG INTERRUPT

When enabled, the Timer/Watchdog will issue an interrupt request at every End Of Count.

A pair of control bits, IA0S (EIVR.1, Interrupt A0 selection bit) and TLIS (EIVR.2, Top Level Input Selection bit) allow the selection of 2 interrupt sources (the Timer/Watchdog End of Count or an external pin) in two different ways, as a top level non maskable interrupt (Software Reset) or as a source for channel A0 of the external interrupt logic.

In the Watchdog mode the End Of Count always causes a system reset.

Figure 8-2.



A block diagram of the interrupt logic is given below (Note: software traps can be generated by setting the interrupt pending bit):

The following table shows all the possible configurations of the interrupt/reset sources which involve the Timer/Watchdog:

Control Bits			Enabled Sources			Watchdog Timer Status
WDGEN	IAOS	TLIS	Reset	INTA0	Top Level	
0	0	0	WDG/Ext Reset	SW TRAP	SWT TRAP	Watchdog
0	0	1	WDG/Ext Reset	SW TRAP	Ext Pin	Watchdog
0	1	0	WDG/Ext Reset	Ext Pin	SW TRAP	Watchdog
0	1	1	WDG/Ext Reset	Ext Pin	Ext Pin	Watchdog
1	0	0	Ext Reset	Timer	Timer	Timer
1	0	1	Ext Reset	Timer	Ext Pin	Timer
1	1	0	Ext Reset	Ext Pin	Timer	Timer
1	1	1	Ext Reset	Ext Pin	Ext Pin	Timer

8.4. TIMER/WATCHDOG REGISTERS

The Timer/Watchdog has 4 registers mapped into Goup F, Page 0 of the Register File.
WDTCR (R251): Timer/Watchdog Control Register
WDTPR (R250): Timer/Watchdog Prescaler Register
WDTRL (R249): Timer/Watchdog Counter low register
WDTRH (R248): Timer/Watchdog Counter high register

Three additional control bits are mapped in the following registers of page 0:

- watchdog mode enable, WCR.6
- top level interrupt selection, EIVR.2
- interrupt A0 channel selection, EIVR.1

Note: The registers containing these bits also contain other functions only the bits relevant to the operation of the Timer/Watchdog are shown here.

WDTPR -(R250) is used to select the prescaling factor from 1 (loading 00h) to 256 (loading FFh).

WDTRL -(R248), WDTRH -(R249) This 16 bit register is used to load the 16 bit counter value. The registers can be read or written "on the fly".

WARNING: In order to prevent incorrect counting of the Timer/Watchdog, the prescaler (WDTPR) and counter (WDTRL, WDTRH) registers must be initialised before the starting of the Timer/Watchdog. If this is not done, counting will start with the reset (un-initialised) values.

WDTCR - R251 (0FBh) Page 0; Read/Write
Timer/Watchdog Control Register

Reset value: 0001 0010 (12h)

7							0
ST_SP	S_C	INMD1	INMD2	INEN	OUTMD	WROUT	OUTEN

b7 = **ST_SP:** *Start/Stop Bit.* Setting this bit to a "1" starts the counting operation (see Warning above). When this bit is "0", the counter is stopped (reset status)

b6 = **S_C:** *Single/Continuous.* When this bit is set, the counter operates in Single Count Mode. Continuous Mode is set when this bit is "0"

b5-b4 = **INMD1, INMD2:** *Input mode selection bits.*

INMD1	INMD2	Input Modes
0	0	Event Counter
0	1	Gated (Reset status)
1	0	Triggerable
1	1	Retriggerable

b3 = **INEN:** *Input Enable.* This bit enables ("1") and disables ("0") the input section

b2 = **OUTMD:** *Output Mode.* When this bit is "1", and the output is enabled, the value of WROUT is transferred to the output pin on every End Of Count. When "0", the output is toggled on every End of Count

b1 = **WROUT:** *WROUT bit.* The status of this bit is transferred to the Output pin when OUTMD = "1", it is user definable to allow PWM output (at reset WROUT = "1")

b0 = **OUTEN:** *Output Enable bit.* The output is enabled by setting this bit to "1", and disabled by resetting to "0"

WCR - R252 (0FCh) Page 0; Read/Write
Wait Control Register

Reset value: 0111 1111 (7Fh)

7							0
X	WDGEN	X	X	X	X	X	X

b6 = **WDGEN:** *Watchdog Enable Bit (active low).* Resetting this bit to zero via software enters the Watchdog mode. Once reset , it cannot be set to "1" by the user program. At system reset, the Watchdog mode is disabled

EIVR - R246 (0F6h) Page 0; Read/Write
External Interrupt Vector Register

Reset value: xxxx 0110 (X6h)

7							0
X	X	X	X	X	TLIS	IAOS	X

b2 = **TLIS:** *Top Level Input Selection bit.* This bit selects the Top Level interrupt source. When "0", the Top Level interrupt source is the Watchdog/Timer end of count, when = "1", it is the external pin NMI.

b1 = **IAOS:** *Interrupt A0 channel Selection Bit.* This bit allows the Timer/Watchdog interrupt to channel through the external Interrupt A0 source, allowing the setting of user-defined priority levels.

WARNING: To avoid spurious interrupt requests, an access to the IAOS bit must be made only when the interrupt logic is disabled (i.e. after the DI instruction). It is also necessary to clear a possible interrupt pending request on channel A0 before enabling this interrupt channel. A delay instruction (e.g. a NOP instruction) must be inserted between the reset of the interrupt pending bit and the IAOS write instruction.

CHAPTER 9

I/O PORTS AND HANDSHAKE TRANSFERS

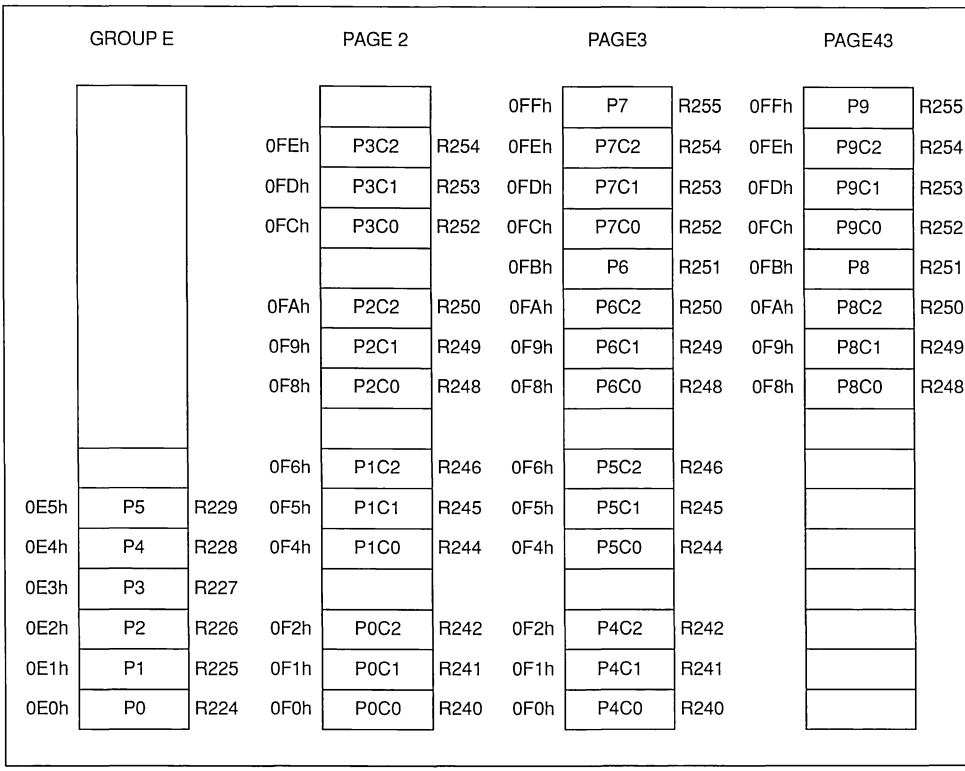
9.1 INTRODUCTION

The ST9 is provided with dedicated lines for input/output. These lines, grouped into 8-bit ports, can be independently programmed to provide parallel input/output with or without handshake, or to carry in/out signals to/from the on-chip peripherals and Core (e.g. Timers and SCI). All ports have active pull-ups and pull-down resistors compatible with TTL loads. In addition, pull-ups can be turned off for open-drain operation and weak pull-ups can be turned on to save off-chip resistive pull-ups. Input buffers can be either TTL or CMOS compatible.

9.2 CONTROL REGISTERS

Each port PX (P0 - P7) has three associated control registers (PXC0, PXC1, PXC2) which define the port lines configuration and allow dynamic change in port configuration during program execution. Ports and control registers are mapped into the Register File as shown in Fig 9.1. Ports and control registers are treated like any other general-purpose register. There are no special instruction for port manipulation, any instruction that addresses a register can address the ports. Data can be directly accessed in the port register, without passing through other memory or "accumulator" locations.

Figure 9-1. Ports and Control Register Map in the Register File



During the reset state, all the Ports are set as bidirectional/weak pull-up mode, with output data register set to 0FFh. This condition is also held after reset (except for Ports 0, 1, 6 in romless devices, see chapter 6) and can be redefined under software control at any time.

9.3 PORT BIT STRUCTURE AND PROGRAMMING

By programming the control bits PXC0n and PXC1n (see Figure 9.2a) it is possible to configure bit PXn as Input, Output, Bidirectional or Alternate Function, where X is the number of the I/O port, and n the bit within the port (n = 0 to 7).

By programming the control bit PXC2n it is possible to select the input level as TTL or CMOS.

The output buffer can be programmed as Push-pull or Open-drain. A Weak Pull-up configuration can be used when the port bit is programmed as Bidirectional. It is an Open-drain configuration with a high pull-up resistor value (turned on by writing a "1"), to avoid the requirement for external resistances.

The basic structure of the bit PXn (n = 0 to 7) of a general purpose port PX is shown in Figure 9.3.

Independently of the chosen configuration, when the User addresses the port as an destination register of an instruction, the port is written to and the data is transferred from the Internal Data Bus into the Output Master Latches. When the port is addressed as a source register for an instruction, the port is read and the data stored in the Input Latch is transferred onto the Internal Data Bus.

When PXn is programmed as Input: (Fig. 9.4)

- The Output Buffer is forced tristate
- The data present on the I/O pin is sampled into the Input Latch at the beginning of the execution of each instruction
- The data stored in the Output Master Latch is copied into the Output Slave Latch at the end of the execution of each instruction. So if bit PXn is reconfigured as Output or Bidirectional, the data stored in the Output Slave Latch is reflected on the I/O pin.

Figure 9-2a. Control Bits

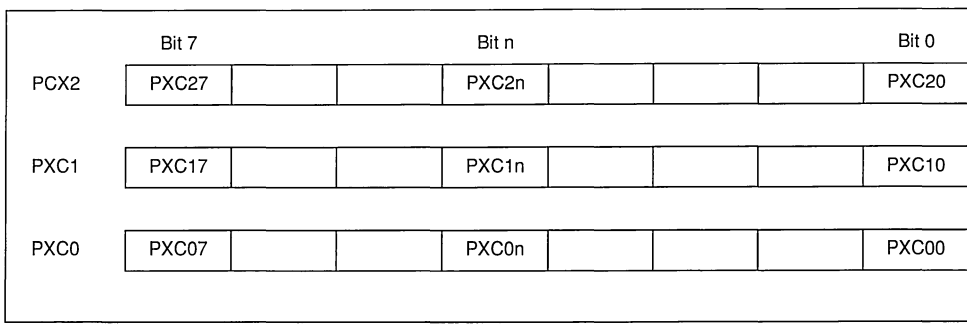


Figure 9-2b. Port Bit Configuration Table

PXC2n PXC1n PXC0n	1 0 0	0 0 0	1 0 1	0 0 1	1 1 0	0 1 0	1 1 1	0 1 1
PXn Configuration	BID*	BID*	IN*	IN*	OUT*	OUT*	AF*	AF*
PXn Output	OD	WP	TRI	TRI	OD	PP	OD	PP
PXn Input	TTL	TTL	TTL	CMOS	TTL	TTL	TTL	TTL

Notes:

BID* : BIDIRECTIONAL
IN* : INPUT
OUT* : OUTPUT
AF* : OUTPUT ALTERNATE FUNCTION
TRI : TRISTATE
OD : OPEN DRAIN
WP : WEAK PULL-UP
PP : PUSH-PULL
TTL : TTL STANDARD INPUT
CMOS : CMOS STANDARD INPUT

* These configurations are shown in Figure 9-4, 9-5, 9-6 and 9-7.

Figure 9-3. Basic Structure of an I/O Port Bit

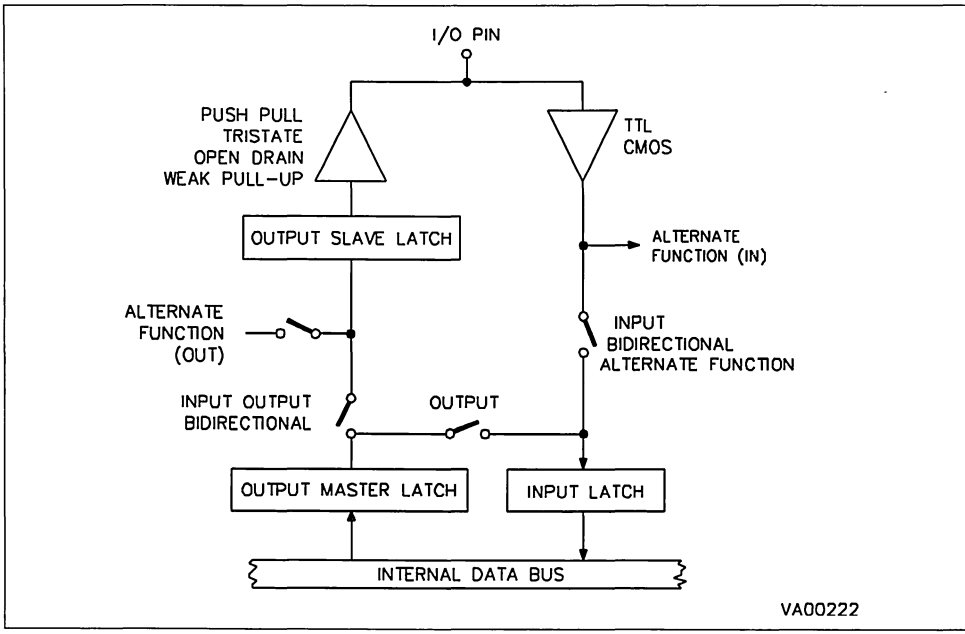


Figure 9-4. Input Configuration

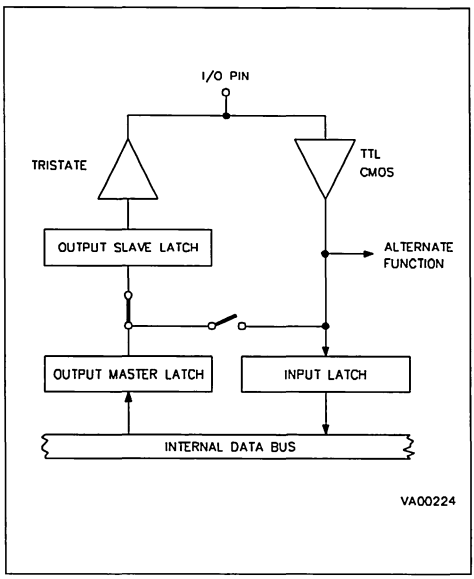
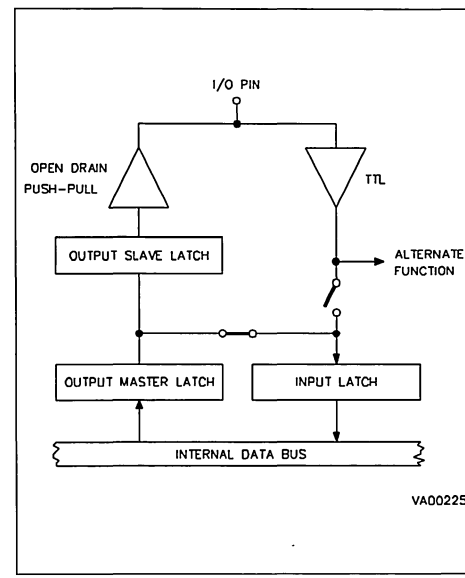


Figure 9-5. Output Configuration



When PXn is programmed as Output: (Fig. 9.5)

- The Output Buffer is turned on in an Open-drain or Push-pull configuration
- The data stored in the Output Master Latch is copied both into the Input Latch and into the Output Slave Latch, driving the I/O pin, at the end of the execution of each instruction.

When PXn is programmed as Bidirectional: (Fig. 9.6)

- The Output Buffer is turned on in an Open-drain or Weak Pull-up configuration
- The data present on the I/O pin is sampled into the Input Latch at the beginning of the execution of each instruction
- The data stored in the Output Master Latch is copied into the Output Slave Latch, driving the I/O pin, at the end of the execution of each instruction.

Due to the unique feature of the bidirectional mode of reading the external pin instead of the output latch, particular care must be taken with arithmetic/logic and boolean instructions performed on a bidirectional port pin.

These instructions use a read-modify-write sequence, and the result written in the port register depends on the logical level present on the external pin.

This may bring unwanted modifications to the port output register content.

For example:

Port register content	external port value
0Fh	03h

(Bits 3 and 2 are externally forced to 0)

Making a `BSET` instruction on bit 7 will return:

Port register content	external port value
83h	83h

(Bit 3,2 have been cleared.)

To avoid this situation, it is suggested that all the operations on a port using at least one bit in bidirectional mode, are performed on a copy of the port register, then transferring the result with a load instruction to the I/O port.

When PXn is programmed as Alternate Function: (Fig. 9.7)

- The Output Buffer is turned on in an Open-drain or Push-pull configuration
- The data present on the I/O pin is sampled into the Input Latch at the beginning of the execution of each instruction
- A signal coming from an on-chip Module is allowed to load the Output Slave Latch driving the I/O pin. Signal timing is under the Module control. If no module is connected to PXn the I/O pin is driven to a high level in Push-pull configuration and is driven to high impedance in open drain configuration.

Figure 9-6. Bidirectional Configuration

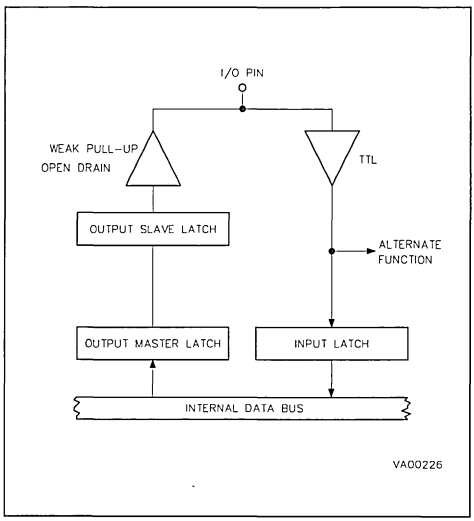
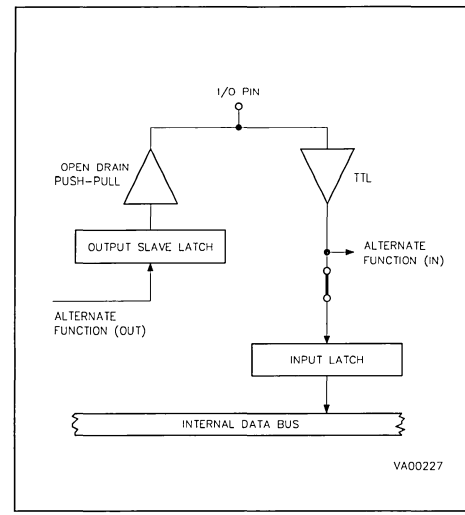


Figure 9-7. Alternate Function Configuration



9.4 ALTERNATE FUNCTION ARCHITECTURE

Each single I/O pin may access three different types of ST9 internal signals:

- Data bus line (I/O)
- Alternate Function Input
- Alternate Function Output

Each pin configuration is done by software, thus allowing the User to choose the type of signal to access a pin. The choice of type of signal is made with the registers PxC2, PxC1, PxC0 of the I/O Port x (Please refer to the previous section for more details)

Pins Declared as an I/O
A pin declared as an I/O is a pin connected to the I/O buffer. In such a case, this pin may either be an Input or an Output or an I/O depending on the value stored in (PxC2, PxC1, PxC0)

Pin Declared As An Alternate Function Input
One single pin may access several Alternate Function inputs. In such a case, the User has to select by software the Alternate Function module (by enabling it) and unselect all other Alternate Functions (by disabling them).

Figure 9-8. Example of 3 Alternate Function Inputs

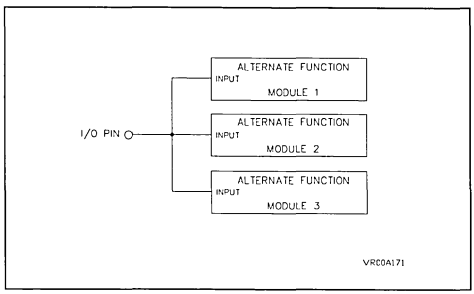
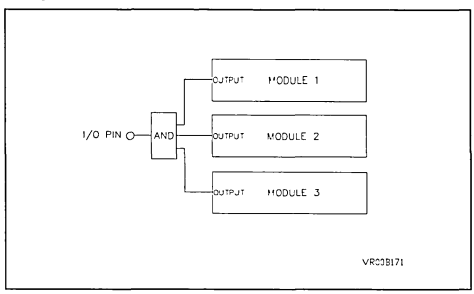


Figure 9-9. Example of 3 Alternate Function Outputs

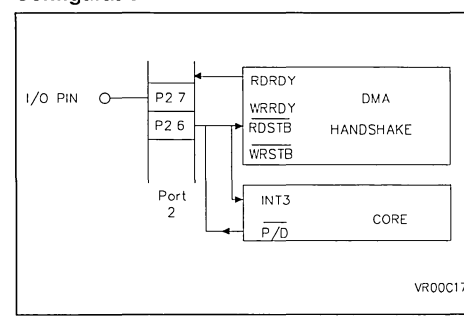


No specific configuration of the port is required to enable the input Alternate Function, as the input buffer is directly connected to each module using it. As more than one module can use the same input Alternate Function line, it is under the software control to enable a module to use the input Alternate Function.

Pin Declared As An Alternate Function Input
A single pin may be directly connected to several Alternate Function inputs. In such a case, the User has to select the required input mode (TTL or CMOS levels) and to enable, by software, the selected Alternate Function module (by enabling it) and unselect all other Alternate Functions (by disabling them).
No specific configuration of the port is required to enable the input Alternate Function, as the input buffer is directly connected to each module using it. The digital I/O remains operational, even when using the Alternate Function input. The exception to this is for an I/O port bit connected to analog voltages (for the Analog to Digital Converter), see the following section.

Pin Declared As An Alternate Function Output
A pin declared as an Alternate function output corresponds to (PxC2,PxC1,PxC0) = 111 or 011. Several Alternate Function outputs may drive a common pin. In such a case, the Alternate Function output signals are ANDed before driving the common pin. The User has therefore to select by software the Alternate Function module by enabling it and has to disable all other modules (a disabled module outputs a "1").

Figure 9-10. Example of One I/O Pin Configuration



General Configuration
A single pin may be used, according to different phases of the software, as an I/O or connected to an Alternate Function input or an Alternate Function output. An example is given in figure 9.10

WARNING. When a common pin is declared to be connected to an Alternate Function input and to an Alternate Function output, the User must be aware of the fact that the Alternate Function output signal always input to the Alternate Function module(s) declared as input(s). Figure 9.10 shows an example where the signal P/D also enters RDSTB and INT3.

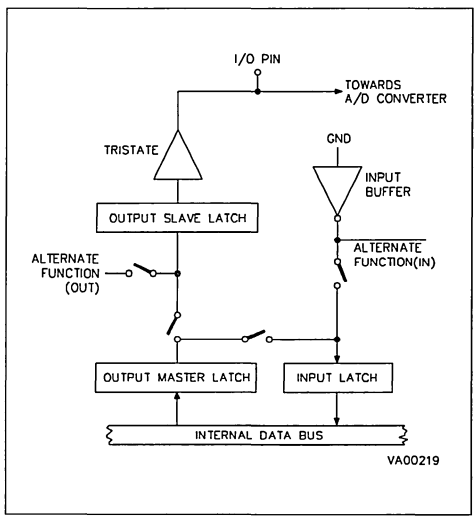
9.5 SPECIAL PORTS

9.5.1 Bit Structure For A/D Converter Inputs

When a port bit is used as input for an on-chip A/D Converter, its structure is modified as shown in Figure 9.11.

The behaviour of this bit is identical to the general purpose bit described in paragraph 9.3 except when it is programmed as Alternate Function. In this case, the Output Buffer is forced Tristate and the input of the Input Buffer is disconnected from the I/O pin and forced low. In this way the I/O pin is free to assume any analog value without causing power consumption in the Input Buffer. The bit MUST be programmed to (PxC2, PxC1, PxC0)=111) to assume this special configuration.

Figure 9-11. A/D Input Port Bit Structure



9.6 I/O STATUS AFTER WFI, HALT AND RESET

The status of the ST9 I/O ports during the Wait For Interrupt, Halt and Reset operational modes is shown in the following table. The External Memory Interface ports are shown separately, however, if only the internal memory is being used and the ports are acting as I/O, the status is the same as shown for the other I/O ports.

Mode	P0	P1 [P6]	I/O
WFI	HIgh Impedance	Next Address	No Affect (clocks output from ST9 running)
HALT	HIgh Impedance	Next Address	No Affect (clocks output from ST9 stopped)
RESET	Note 1		Bidirectional Weak Pull-up except: ROMless = Input TTL BS_EN = Input CMOS

Note 1: P0, P1 and P6 (when used to provide non-multiplexed low order address) setup depends on the ROMLESS condition.

- if ROMLESS (ST9 memory is Off-chip):
  P0 is set to A.F Tristate
  P1,P6 are set to A.F.
  Push-pull, Output value is undefined.
- if not ROMLESS (ROM or EPROM parts)
  P0, P1 and P6 are set to Bidirectional Weak Pull-up, Output value is FFh (all pins high).

9.7 HANDSHAKE/DMA CONTROLLER

9.7.1 Introduction

This module allows the User to configure an I/O Port under handshake control or to support DMA operations, driven by an on-chip 16 bit Multifunction TIMER, between Data/Program Memory or Register File and an I/O port.

A block diagram of the module is shown in Figure 9.12.

The module supports data exchange with handshake through port PX with 4 handshake lines ($\overline{RDSTB}$, RDRDY, $\overline{WRSTB}$, WRRDY) connected as Alternate Functions. Input, Output and Bidirectional Handshake modes are available.

Input Alternate Function $\overline{RDSTB}$ and $\overline{WRSTB}$ are always associated to external interrupt channels. To synchronize handshake protocols generating interrupt requests (as the following paragraph will show) the User must program the interrupt control register and the vector associated to the used line(s) ($\overline{RDSTB}$ and/or $\overline{WRSTB}$). The active high output lines RDRDY and WRRDY are held high when not active in order to allow the Alternate Function connection with other ST9 peripherals.

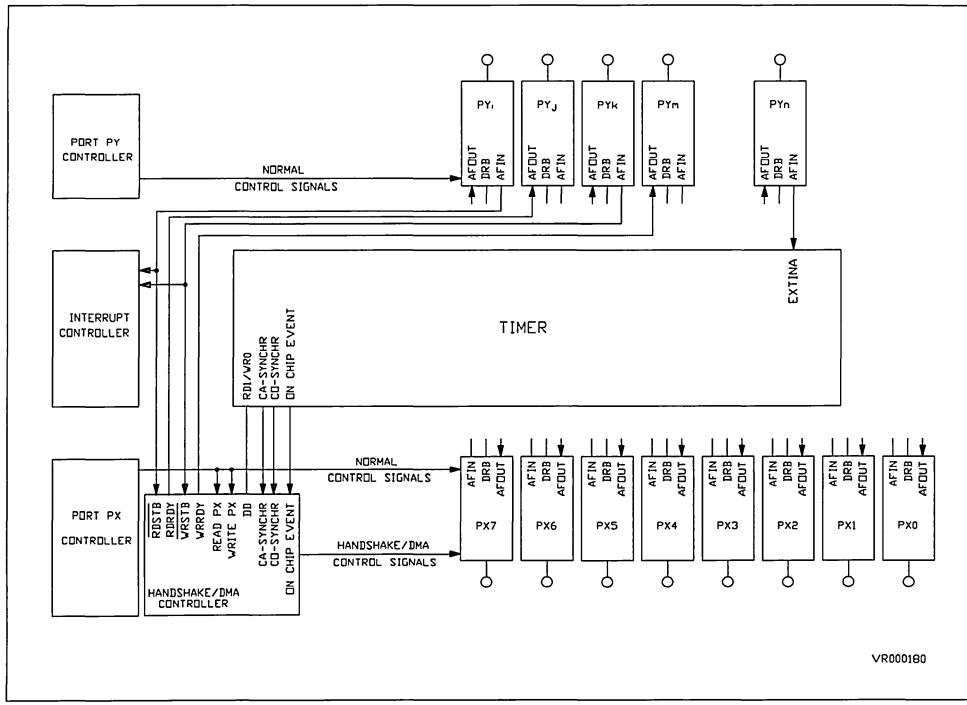
DMA transfers can move data from Data/Program Memory or Register File to the port or viceversa, using either the Multifunction Timer CAPT0 or COMP0 DMA Channels. In Figure 9.12 the four on-chip lines that connect the module to the on-chip Multifunction Timer to support DMA transfers are shown (DD (Data Direction), CO_SYNCHR(COmpare SYNCHronism), CA_SYNCHR (CApture SYNCHronism) and On Chip Event).

To program the module the User has to write the Handshake/DMA Control register (HDCTL) according to the table shown in Figure 9.13. Different handshake protocols and the Port behaviour during DMA operations are explained in the following paragraphs.

ST9 HANDSHAKE/DMA CONTROL REGISTER ADDRESSES

	PAGE 0			PAGE 2			PAGE3	
			0FFh	HDCTL3	R255	0FFh	P7	R255
			0FEh	P3C2	R254	0FEh	P7C2	R254
			0FDh	P3C1	R253	0FDh	P7C1	R253
			0FCh	P3C0	R252	0FCh	P7C0	R252
			0FBh	HDCTL2	R251	0FBh	P6	R251
			0FAh	P2C2	R250	0FAh	P6C2	R250
			0F9h	P2C1	R249	0F9h	P6C1	R249
			0F8h	P2C0	R248	0F8h	P6C0	R248
						0F7h	HDCTL5	R247
			0F6h	P1C2	R246	0F6h	P5C2	R246
0E5h	P5	R229	0F5h	P1C1	R245	0F5h	P5C1	R245
0E4h	P4	R228	0F4h	P1C0	R244	0F4h	P5C0	R244
0E3h	P3	R227				0F3h	HDCTL4	R243
0E2h	P2	R226	0F2h	P0C2	R242	0F2h	P4C2	R242
0E1h	P1	R225	0F1h	P0C1	R241	0F1h	P4C1	R241
0E0h	P0	R224	0F0h	P0C0	R240	0F0h	P4C0	R240

Figure 9-12. Handshake/DMA Controller Module Block Diagram



HDCTL Read/Write
Handshake/DMA Control Register

Reset Value: 1111 1111 (0FFh)

7							0
HS7	HS6	HS5	DEN	DD	DST	DCH	1

b7-b5 = **HS7, HS6, HS5:** *Handshake Mode Selection.* These bits allow selection of the Handshake direction and the number of wires used in the handshake as shown in the following table.

b4 = **DEN:** *DMA Enable.* This bit (when reset) enables the DMA function with handshake through an I/O Port. DMA is disabled when this bit = "1".

b3 = **DD:** *DMA Data Direction.* The direction of the DMA transfers through an I/O Port is set by this bit. A "1" sets DMA Input and a "0" sets DMA Output.

b2 = **DST:** *DMA Strobe.* This bit enables the use of a Multifunction Timer On-Chip Event to trigger the DMA transaction when set.

b1 = **DCH:** *DMA Channel* When DST is set, allowing the DMA transactions to be triggered by a Multifunction Timer, DCH selects the MFT source, a "1" selects the COMP0 source, a "0" selects the CAPT0 source.

b0 = **D0.** This bit is fixed by hardware to a high level.

Figure 9-13. Module Configuration Table

Handshake Modes			HS7	HS6	HS5
HS	Disabled		X	1	1
HS	Output	(2 lines)	1	1	0
HS	Output	(1 line)	0	1	0
HS	Input	(2 lines)	1	0	1
HS	Input	(1 line)	0	0	1
HS	Bidirectional	(2 lines)	X	0	0

9.7.2 Programmable Handshake Modes

9.7.2.1 INPUT HANDSHAKE

Two Input Handshake Modes are available to synchronise input transitions on port bits programmed as Input or Bidirectional. Output or Alternate Function bits are not affected.
In the timings, READ PORT is an ST9 internal signal that transfers data from the Input Latches onto the Data bus.

Two Lines Input Handshake

When this mode is selected WRRDY is set to indicate that data can be loaded into the Input Latches of the Input and Bidirectional port pins. Data present on the pins are sampled when the peripheral forces a low level on $\overline{WRSTB}$.

When a rising edge on $\overline{WRSTB}$ occurs, WRRDY goes low signifying that the Input Latch is full and further loading must be inhibited until the ST9 reads the port. When the port register is read, WRRDY is set. Both low and high levels on WRSTB must last at least one INTCLK cycle.

The User is suggested to program the External Interrupt Channel associated with the $\overline{WRSTB}$ line to generate an interrupt request when a rising edge occurs. The ST9 can thus, in the course of its interrupt service routine, read the data furnished by the peripheral as soon as it is available.

One Line Input Handshake

Figures 9.14c and 9.14d illustrate the timing associated with the One Line (WRRDY) Input Handshake Mode.

When this mode is selected WRRDY is set to indicate that data can be loaded into the Input Latches of the Input and Bidirectional port pins.

Data present on the pins is continuously sampled. When the ST9 is reading the port WRRDY goes low. If data is strobed into the port only when WRRDY is high, the forced low state of WRRDY will prevent Input Latches data from changing while ST9 is reading the port. When the ST9 reading cycle finishes, WRRDY is set.

Figure 9-14a. Two Lines Input Handshake

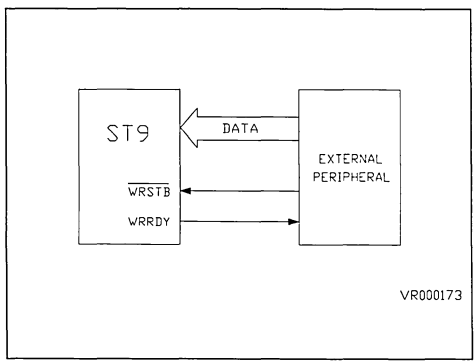
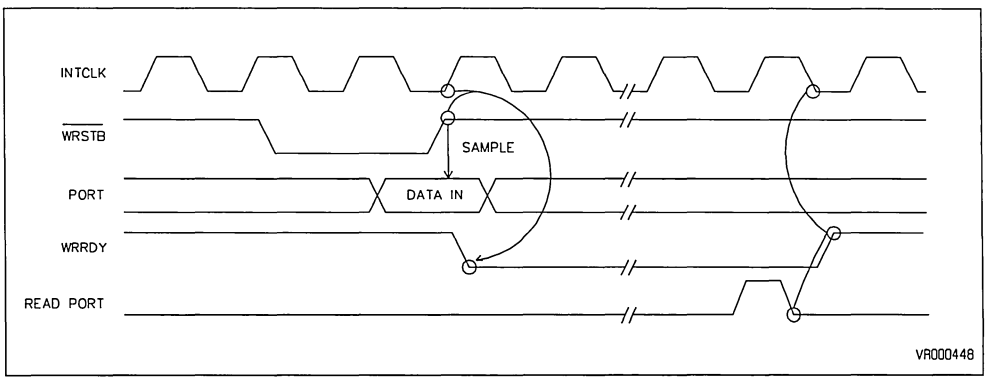


Figure 9-14b. Two Lines Input Handshake Timing



9.7.2.2 OUTPUT HANDSHAKE

Two Output Handshake Modes are available to synchronize output transitions on port bits programmed as Output or Bidirectional. Input or Alternate Function bits are not affected.

In the timings WRITE PORT is the internal signal that transfers data from the Internal Data Bus into the Port Output Master Latches.

Two Lines Output Handshake

Figure 9.15b illustrates the timing associated with the Two Lines (RDRDY, $\overline{\text{RDSTB}}$) Output Handshake Mode (Figure 9-15a).

When this mode is selected RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written, RDRDY is set to indicate that data is ready for the peripheral device. In most systems the rising edge of RDRDY can be used as a latching signal in the peripheral device. RDRDY will remain high until a rising edge is received on $\overline{\text{RDSTB}}$ indicating that the peripheral has taken the data. Both low and high level on $\overline{\text{RDSTB}}$ must last at least one ST9 INTCLK cycle. The User is suggested to program the External Interrupt Channel associated with the $\overline{\text{RDSTB}}$ line to generate an interrupt request when a rising edge occurs. The ST9 can thus, in the course of its interrupt service routine, furnish new data as soon as the previous data is taken by the peripheral.

One Line Output Handshake

Figure 9.15 illustrates the timing associated with the One Line (RDRDY) Output Handshake Mode Figure 9-15c.

When this mode is selected RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written to, RDRDY is set to indicate that data is ready for the peripheral device. In most systems the rising edge of RDRDY can be used as a latching signal in the peripheral device. No peripheral acknowledge is waited for. While ST9 is writing into the Output Slave Latches RDRDY goes low, RDRDY is set again when the new data is ready on the port pins.

Figure 9-14c. One Line Input Handshake

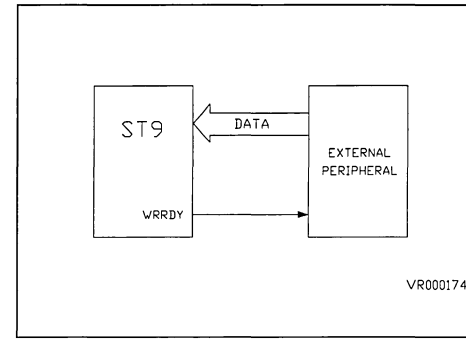


Figure 9-15a. Two Lines Output Handshake

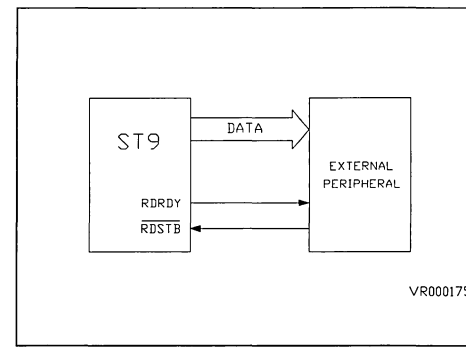


Figure 9-15c. One Line Output Handshake

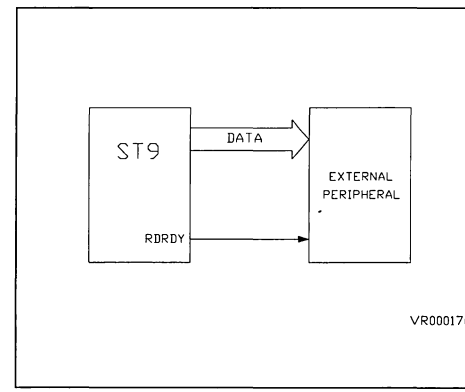


Figure 9-14d. One Line Input Handshake Timing

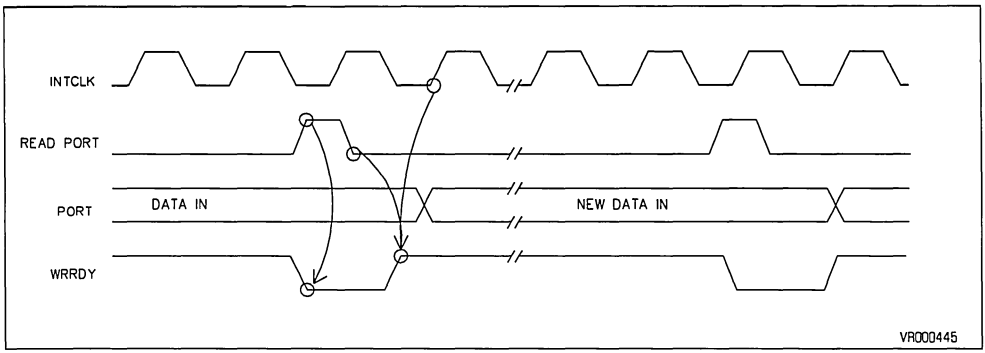


Figure 9-15b. Two Lines Output Handshake Timing

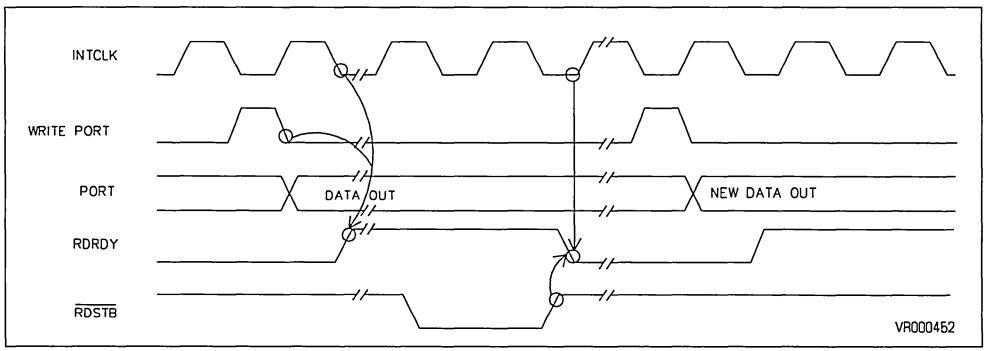
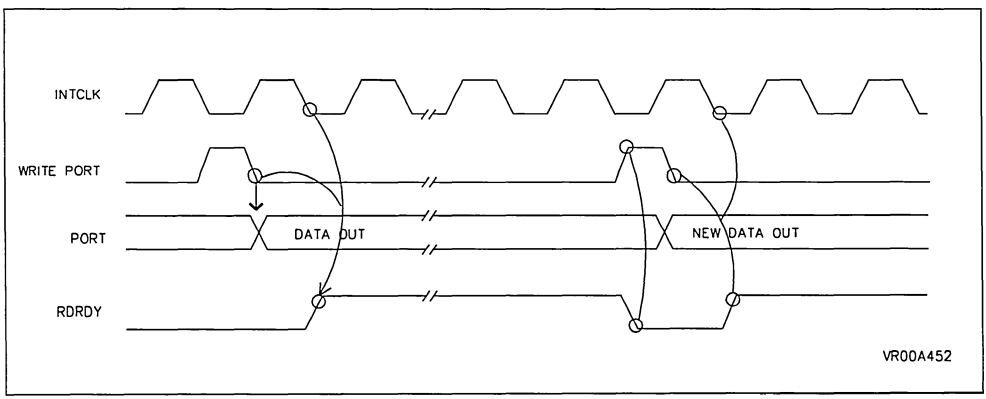


Figure 9-15d. One Line Output Handshake Timing



9.7.2.3 BIDIRECTIONAL HANDSHAKE

A Bidirectional Handshake Mode is available to synchronise bidirectional transitions on Port bits programmed as Bidirectional. When this mode is selected, the Output Buffer configuration of Bidirectional port pins programmed as Weak Pull-up become Push-pull. Open-drain configuration is not modified. Bits set to Input, Output or Alternate Function are not affected.

Figure 9.16b illustrates the timing associated with the Bidirectional Handshake Mode. This mode is a combination of Two Lines Output Mode and Two Lines Input Mode using all four handshake lines, two for output (RDRDY, $\overline{\text{RDSTB}}$) and two for input control (WRRDY, $\overline{\text{WRSTB}}$). In the timing INTCLK is the ST9 internal not stretched clock, WRITE PORT is the signal that transfers data from the Internal Data Bus into the port Output Master Latches and READ PORT is the signal that transfers data from the Input Latches onto the Data Bus.
When Bidirectional Handshake mode is selected the Output Buffers of the Bidirectional port pins are forced tristate, WRRDY is set to indicate that data can be loaded into the Input Latches and RDRDY is reset to indicate that no significant data is present in the Output Slave Latches.

Input Transitions. Data present on the pins is sampled when the peripheral forces a low level on $\overline{\text{WRSTB}}$. When a rising edge on $\overline{\text{WRSTB}}$ occurs, WRRDY goes low signifying that the Input Latches are full and further loading must be inhibited until the ST9 reads the port. When the port register is read, WRRDY is set. Both low and high levels on $\overline{\text{WRSTB}}$ must last at least an ST9 INTCLK cycle.

The User is suggested to program the External interrupt Channel associated with the $\overline{\text{WRSTB}}$ line to generate an interrupt request when a rising edge occurs.The ST9 can thus, in the course of its interrupt service routine, read the data furnished by the peripheral as soon as it is available.

Output Transitions. When the Output Slave Latches are written, RDRDY is set to indicate that data is ready for the peripheral device. When $\overline{\text{RDSTB}}$ goes low, data is allowed out onto the port pins. When a rising edge is received on $\overline{\text{RDSTB}}$, indicating that the peripheral has taken the data, the Output Buffers are forced tristate and RDRDY goes low. Both low and high level on $\overline{\text{RDSTB}}$ must last at least an ST9 INTCLK cycle.

The User is suggested to program the External Interrupt Channel associated to the $\overline{\text{RDSTB}}$ line to generate an interrupt request when a rising edge occurs; The ST9 can thus, in the course of its interrupt service routine, write new data into the Output Slave Latches as soon as the previous data is taken by the peripheral.

Figure 9-16a. Four Lines Bidirectional

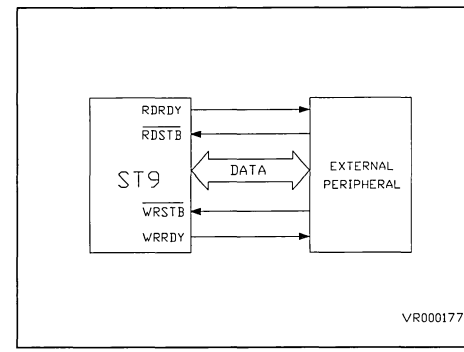
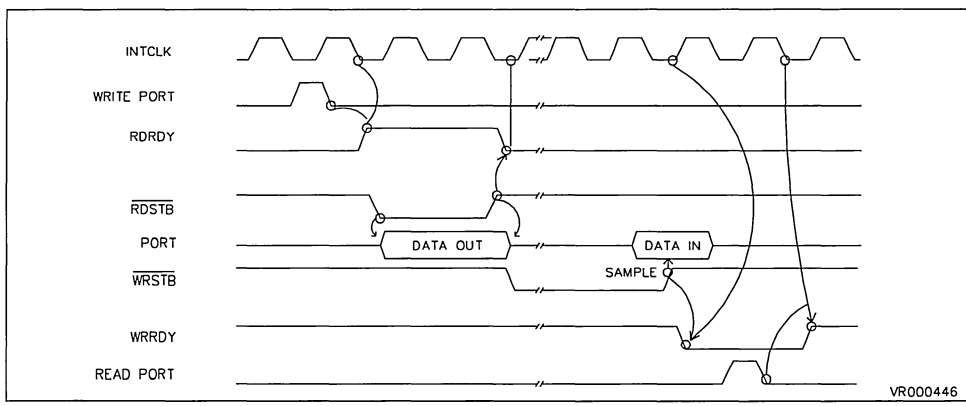


Figure 16-b. Bidirectional Handshake Timing



APPLICATION EXAMPLE: MAPPING AN ST9 ONTO THE MEMORY BUS OF ANOTHER ST9

Fig 9.17 shows a possible application of the bidirectional handshake protocol, used to connect an ST9 to a slave of another (master) ST9.

PX of the slave ST9 is connected to the Address/Data Memory Bus of the master ST9. A decoder enables, with a low level, the generation of $\overline{RDSTB}$ or $\overline{WRSTB}$ when $\overline{DS}$ is low and the master is reading from, or writing to, the memory.

To synchronize data transfers with the slave, the master ST9 uses RDRDY and WRRDY as External Interrupt Sources, programmed to generate an interrupt request when a rising edge occurs. The slave ST9 interrupts the master raising RDRDY when new data is ready in the port Output Slave Latches and raising WRRDY when the Input Latches can be filled with new data. According to the interrupt request received, the master ST9 can read from the slave the ready data (RDRDY interrupt routine) or write into the slave other data (WRRDY interrupt routine).

9.7.3 Programmable DMA Modes

The Module supports DMA operations controlled by either the Timer CAPT0 or COMP0 DMA Channel. The User enables this function writing a "0" in the DEN bit in the HDCTL register and selects the Channel writing the DCH bit: "0" for CAPT0, "1" for COMP0.

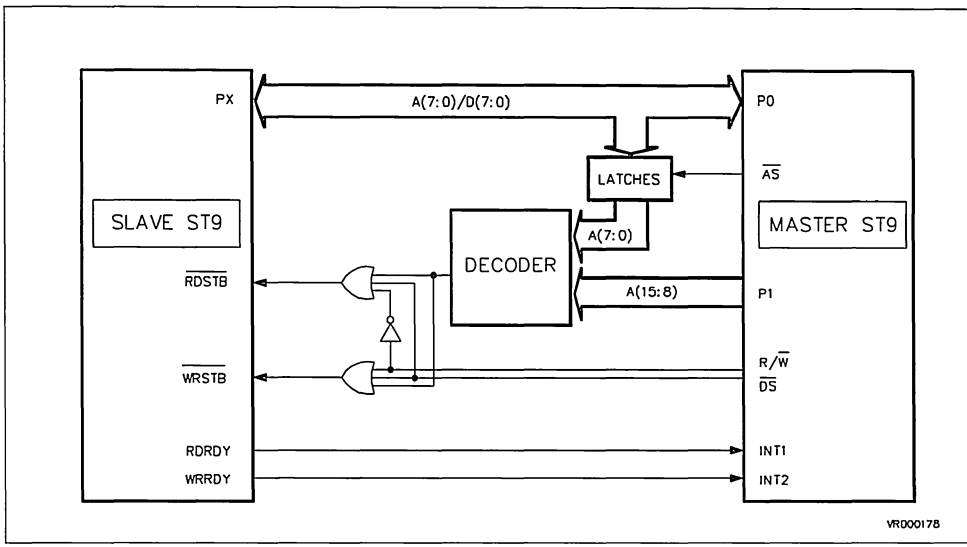
When the CAPT0 Channel is chosen, the DD bit selects the Data Direction: "0" to move data from Data/Program Memory or Register File to the port (DMA Output), "1" to perform the opposite transfer (DMA Input). Signal CA_SYNCHR is sent by the Timer to the Handshake/DMA Controller for writing the port Output Master Latches or reading the Input latches (depending on DD), during the DMA operations when a capture occurs on the Timer.

If the Handshake section of the module is enabled, the data transfer from the Output Master Latches into the Output Slave Latches (Output Strobe, for pins programmed as Output or Bidirectional) or from the Pins into the Input Latches (Input Strobe, for pins programmed as Input or Bidirectional) is controlled by the logic supporting the chosen Handshake protocol.

If no Handshake is programmed the User can choose how to drive the Output or Input Strobe by writing the DST bit: a "0" leaves the Strobes under the normal port control, according to the chosen port bit configuration (see Paragraph 9.3), a "1" selects the On Chip Event generated by the Timer as the Output or Input Strobe.

When the COMP0 Channel is selected, DMA output transfers are only allowed independent of DD, and CO_SYNCHR is used for output Master Latch. If Handshake is disabled, DST selects how to control the Output Strobe. If enabled, the Handshake controls the Output Strobe.

Figure 9-17. Bidirectional Application Example



9.7.4 DMA Transfers Driven By Timer CAPT0 Channel With Handshake

The following descriptions are made assuming that DMA transfers are driven by Multifunction Timer 0. The following list shows the DMA Port capabilities of the ST9 family:

- For the ST902X family, MFTimer On Chip Event and DMA channels may be internally connected (by software) to I/O Port 5 to provide external DMA/Handshake transfer.
- For the ST903X and ST904X family, MFTimer 0 is internally connected to the A/D converter trigger. MFTimer 1 On Chip Event and DMA channels may be internally connected to I/O Port 5 to provide external DMA/Handshake transfer.
- For the ST905X family, the On Chip Event of MFTimer 0 controls DMA/Handshake transfer with I/O Port 4, the On Chip Event of MFTimer 1 controls DMA/Handshake with I/O Port 5 and the On Chip Event of MFTimer 3 is connected to the internal trigger of the A/D converter.

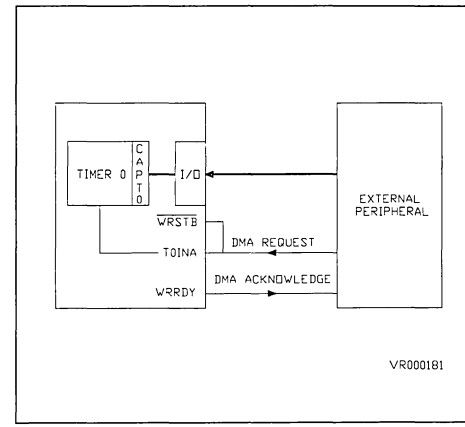
9.7.4.1 INPUT TRANSFERS WITH TWO LINE INPUT HANDSHAKE

When

- Two Lines Input Handshake mode is selected (HS7="1", HS6="0", HS5="1")
- the port is enabled to support DMA input transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DD="1", DCH="0")
- the Handshake $\overline{\text{WRSTB}}$ line is connected off-chip to the Timer T0INA line
- T0INA DMA requests are enabled on rising edges
- $\overline{\text{WRSTB}}$ interrupt requests are disabled data transfers on port pins programmed as Input (or Bidirectional) can be synchronized using the Handshake $\overline{\text{WRSTB}}$ line as DMA Request and the WRRDY line as DMA Acknowledge.

WRRDY is set to indicate that data can be loaded into the Input Latches of the Input (or Bidirectional) port pins. Data present on the port pins is sampled when the peripheral forces a low level on $\overline{\text{WRSTB}}$. When a rising edge on $\overline{\text{WRSTB}}$ (T0INA) occurs WRRDY goes low, signifying that the Input Latches are full and further loading must be inhibited until the ST9 reads the port, and a DMA request is issued. When the port register is read, during the DMA transfer, WRRDY is set.

Figure 9-18. DMA with 2 Lines Input Handshake Mode



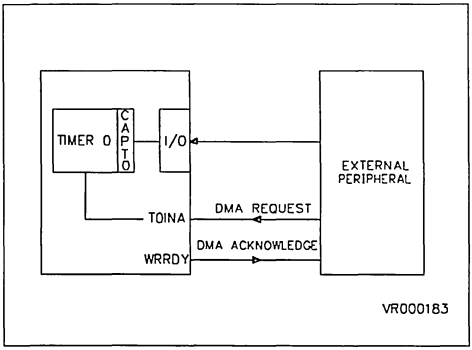
9.7.4.2 INPUT TRANSFERS WITH ONE LINE INPUT HANDSHAKE

When

- One Line Input Handshake is selected (HS7="0", HS6="0", HS5="1")
- the port is enabled to support DMA input operations driven by the Timer CAPT0 DMA Channel (DEN="0", DD="1", DCH="0")
- the Timer T0INA DMA requests are enabled on rising (or falling) edges data transfers on port pins programmed as Input (or Bidirectional) can be synchronized by using the Timer T0INA line as DMA Request, and the Handshake WRRDY line as DMA Acknowledge.

When One Line Input Handshake is selected WRRDY is set to indicate that data can be loaded into the Input Latches of the Input and Bidirectional port pins. Data present on the port pins is continuously sampled. While ST9 is reading the port, during the DMA transfer requested by a rising (or falling) edge on the Timer T0INA line, WRRDY goes low. If data is strobed into the port only when WRRDY is high, the forced low state of WRRDY will prevent Input Latches data from changing while ST9 is reading the port. When ST9 reading cycle finishes, WRRDY is set.

Figure 9-19.DMA Input Transfer with 1 Line Input Handshake



9.7.4.3 OUTPUT TRANSFERS WITH TWO LINES OUTPUT HANDSHAKE

When

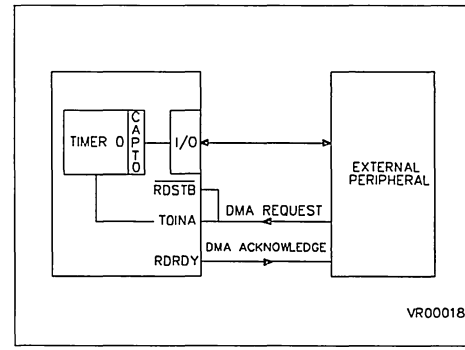
- Two Lines Output Handshake is selected (HS7="1", HS6="1", HS5="0")
- the port is enabled to support DMA output transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DD="0", DCH="0")
- the Handshake $\overline{\text{RDSTB}}$ line is connected off-chip to the Timer T0INA line
- T0INA DMA requests are enabled on rising edges
- $\overline{\text{RDSTB}}$ interrupt requests are disabled

data transfers on port pins programmed as Output (or Bidirectional) can be synchronized when using the Handshake $\overline{\text{RDSTB}}$ and RDRDY lines as DMA Request and DMA Acknowledge.

When Two Lines Output Handshake is selected, RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written, RDRDY is set to indicate that data is ready for the peripheral device. The first data, whose usual meaning is that ST9 is ready to provide the following data by DMA transfers, is normally written by the DMA initialization routine.

When a rising edge is received on $\overline{\text{RDSTB}}$ (T0INA), indicating that the peripheral has taken the data, RDRDY is reset and a DMA request is issued to get the next data. When the ST9 Output Slave Latches are written, during the DMA transfer, RDRDY is set again. If the User wants to get data from ST9 as soon as $\overline{\text{RDSTB}}$ goes low, external latches clocked by $\overline{\text{RDSTB}}$ can be added to create a pipeline stage, that is at each $\overline{\text{RDSTB}}$ low pulse on the falling edge the peripheral gets data transferred into the port by the previous DMA transfer and on the rising edge a DMA request is issued to get the next data.

Figure 9-20. DMA Output Transfer with 2 Lines Output Handshake



9.7.4.4 OUTPUT TRANSFERS WITH ONE LINE OUTPUT HANDSHAKE

When

- One Line Output Handshake is selected (HS7="0", HS6="1", HS5="0")
- the port is enabled to support DMA output transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DD="0", DCH="0")
- the Timer T0INA DMA requests are enabled on rising (or falling) edges

data transfers on port pins programmed as Output or Bidirectional can be synchronized using the Timer T0INA line as DMA Request, and the Handshake RDRDY line as DMA Acknowledge. RDRDY is reset to indicate that no significant data is present on the Output (or Bidirectional) port pins. When the ST9 Output Slave Latches are written, RDRDY is set to indicate that data are ready for the peripheral device. The first data, whose usual meaning is that the ST9 is ready to provide the following data by DMA transfers, is normally written by the DMA initialization routine. When a rising (or falling) edge is received on T0INA, a DMA request is issued to get the next data. While ST9 is writing into the Output Slave Latches, during the DMA transfer, RDRDY goes low. RDRDY is set again when the new data is ready on the port pins.

Figure 9-21. Output Transfer with 1 Line Output Handshake

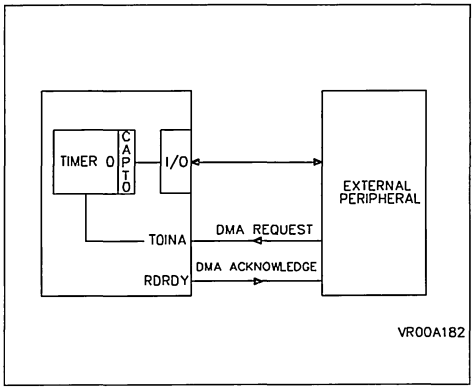
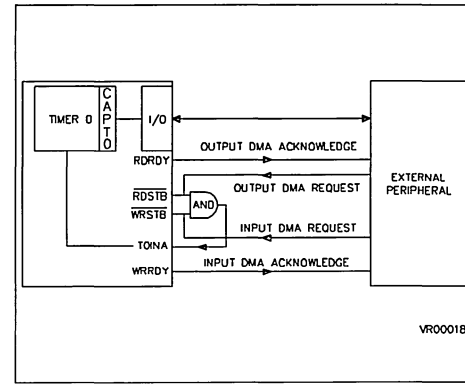


Figure 9-22. Input/Output Transfers with Bidirectional Handshake Configuration



9.7.4.5 INPUT/OUTPUT TRANSFERS WITH BIDIRECTIONAL HANDSHAKE

When

- Bidirectional Handshake is selected (HS7="X", HS6="0", HS5="0")
- the port is enabled to support DMA transfers driven by the Timer CAPT0 DMA Channel (DEN="0", DCH="0")
- the Handshake $\overline{\text{WRSTB}}$ and $\overline{\text{RDSTB}}$ lines are ANDed and connected off-chip to the Timer T0INA line
- T0INA DMA requests are enabled on rising edges
- $\overline{\text{WRSTB}}$ and $\overline{\text{RDSTB}}$ interrupt requests are disabled

data transfers on port pins programmed as Bidirectional can be synchronized using the Handshake $\overline{\text{WRSTB}}$ and WRRDY lines as DMA Request and DMA Acknowledge for DMA Input transfers (DD="1") and the Handshake $\overline{\text{RDSTB}}$ and RDRDY lines as DMA Request and DMA Acknowledge for DMA Output transfers (DD="0").

DMA Input Transfers. When Bidirectional Handshake is selected WRRDY is set to indicate that data can be loaded into the Input Latches of the Bidirectional port pins. Data present on the pins is sampled when the peripheral forces a low level on $\overline{\text{WRSTB}}$. When a rising edge on $\overline{\text{WRSTB}}$ (T0INA) occurs WRRDY goes low, signifying that the Input Latches are full and further loading must be inhibited until the ST9 reads the port, and a DMA request is issued. When the port register is read, during the DMA transfer, WRRDY is set.

DMA Output Transfers. When Bidirectional Handshake is selected, RDRDY is reset to indicate that no significant data is present on the Bidirectional port pins. When the Output Slave Latches are written, RDRDY is set to indicate that data is ready for the peripheral device. The first data, whose usual meaning is that ST9 is ready to provide the following data by DMA transfers, is normally written by the DMA initialization routine.

When $\overline{\text{RDSTB}}$ goes low data is allowed onto the port pins. When a rising edge is received on $\overline{\text{RDSTB}}$ (T0INA), indicating that the peripheral has taken the data, the Output Buffers are forced tristate, RDRDY is reset and a DMA request is issued to get the next data. When the Output Slave Latches are written during the DMA transfer, RDRDY is set again.

In the output data flow there is one pipeline stage, that is at each $\overline{RDSTB}$ low pulse on the falling edge the peripheral gets data transferred into the port by the previous DMA transfer and on the rising edge issues a DMA request to get the next data.

Example. As the direction of DMA transfers is controlled by software, the User must define a protocol to control the sequence of input/output data transfers.

The initialization routine defines the direction (DD) of the first DMA transfer and the address and size of the data buffer (Pointer and Counter associated to the DMA Channel). In the interrupt routine called when the DMA Transaction Counter = 0, the User must define the new address and size of the data buffer and can change (according to the chosen protocol) the direction of next DMA operations.

Figure 9.23 shows how the application example of Figure 9.17 (an ST9 connected as a slave of another ST9) is modified when data transfer from/to the slave ST9 is performed by DMA transfers.

9.7.5 DMA Transfers Driven By Timer Comp0 Channel With Handshake

9.6.5.1 OUTPUT TRANSFERS WITH ONE LINE OUTPUT HANDSHAKE

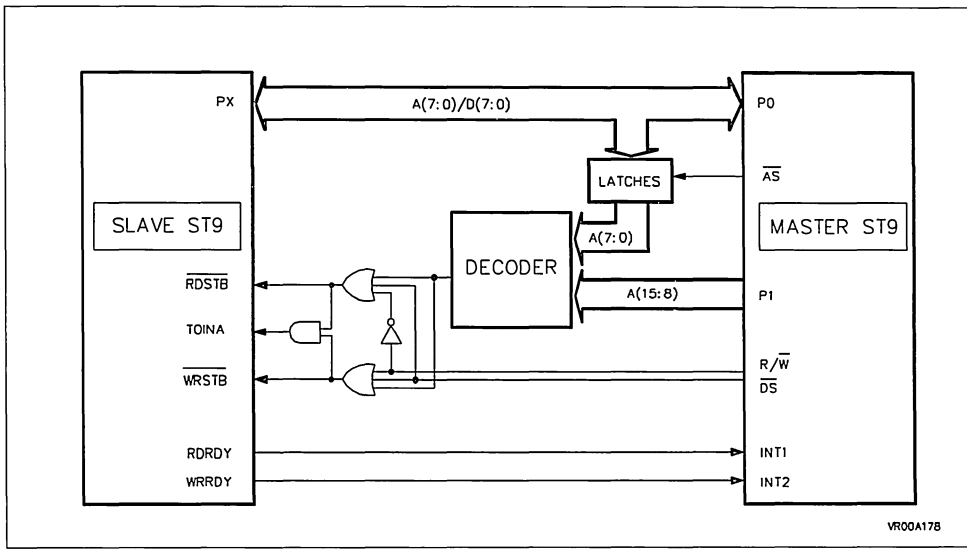
When

- One Line Output Handshake is selected (HS7="0", HS6="1", HS5="0")
- the port is enabled to support DMA output transfers driven by the Timer COMP0 DMA Channel (DEN="0", DCH="1")

data transferred by DMA transfers on port pins programmed as Output or Bidirectional can be strobed using the Handshake RDRDY line.

When One Line Output Handshake is selected RDRDY is reset to indicate that no significant data is present on the Output and Bidirectional port pins. When the Output Slave Latches are written RDRDY is set. The rising edge of RDRDY can be used as a latching signal. At every DMA transfer triggered by the COMP0 event new data is written into the port. While data is changing on the Output Slave Latches, RDRDY goes low. RDRDY is set again when the new data is ready on the port pins.

Figure 9-23. Bidirectional Application Example With DMATransfer



10.1 MULTIFUNCTION TIMER ARCHITECTURE

10.1.1 General Description

The ST9 Multifunction Timer has 2 input pins and 2 output pins available as programmable alternate functions on I/O pins.

The timer contains one 16 bit counter, with an 8 bit prescaler, two Capture/Reload 16 bit registers (REG0R,REG1R) and two 16 bit output Compare registers (CMP0R,CMP1R).

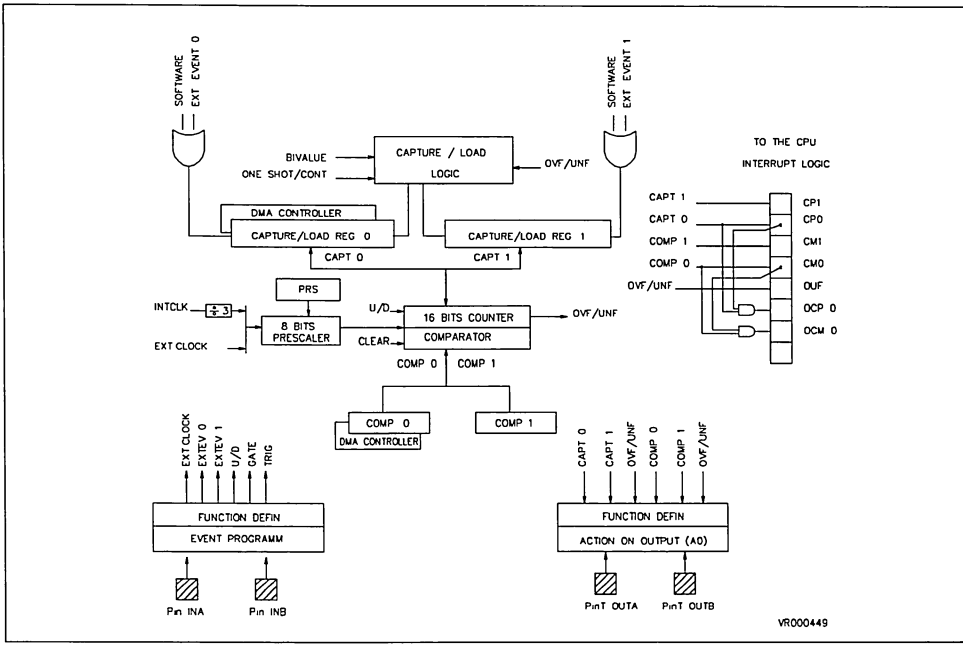
The timer function can be selected by programming two dedicated control registers (TCR-Timer Control Register/TMR-Timer Mode Register). Several functional configurations are possible,e.g.:

- 2 input captures on two different external lines and 2 independent output compare functions (counter in free running mode), or 1 output compare on a fixed repetition rate
- 1 input capture, 1 counter reload and 2 independent output compares
- 2 alternate autoreloads and 2 independent output compares
- 2 alternate captures on the same external line and 2 independent output compares on a fixed repetition rate.

When two timers are present on ST9 chip, a combined mode is available (see section 10.2.11).

Four internal signals are also available for timing of on-chip functions: the On Chip Event signal can be used to control other peripherals on the chip itself, and 3 other signals which can be internally connected to I/O port(s) in order to allow automatic, timed, DMA transfers (See chapter 4).

Figure 10-1. Multifunction Timer Architecture



10.1.2 Timer Input/Output Configuration

The two external inputs (T0INA/T0INB) of the timer can be individually programmed to catch a particular external configuration, i.e.:

- rising edge
- falling edge
- rising and falling edges

The configuration of each input is fixed by the Input Control Register (ICR).

Each of the two output pins (T0OUTA/T0OUTB) can be driven from any of three possible sources:

- Compare Register 0 logic
- Compare Register 1 logic
- Overflow/Underflow logic

Each of these three sources can cause one of the following four effects, independently, on each of the two outputs:

- Nop
- Set
- Reset
- Toggle

Furthermore an additional on-chip Event signal can be generated by two of the three sources mentioned above, i.e. Over/Underflow event and Compare 0 event. This signal can be used internally as synchronism for another on-chip peripheral or as strobe for an I/O port (see I/O port chapter).

10.1.3 Interrupt/DMA Section

Five maskable interrupt sources referring to an End Of Count condition, 2 input captures and 2 output compares, can generate 3 different interrupt requests (with hardware fixed priority), pointing to 3 interrupt routine vectors.

Two independent DMA channels are available for a MFtimer and can be used for quick data flow operations. Each DMA request (associated to a capture on REG0R register, or a compare on CMP0R register) has priority on the INT request generated by the same source.

Each DMA channel can be employed in external transfers to/from memory from/to an I/O port using three internal lines (one for setting the data flow direction, and two for the transfer synchronization).

A SWAP mode is also available to allow high speed continuous transfers (see Interrupt and DMA chapter).

10.2 TIMER OPERATING MODE DESCRIPTION

The different operating modes of the timer can be selected by programming the Timer Control Register (TCR) and the Timer Mode Register (TMR).

10.2.1 One Shot Mode

When the counter generates an overflow (in up-count mode) or an underflow (in down-count mode), i.e. an End Of Count is reached, the counter stops and no counter reload occurs. The counter can be restarted only by an external or software trigger. The One Shot Mode is entered by setting TMR bit CO.

10.2.2 Continuous Mode

Whenever the counter reaches an End Of Count, the counting sequence is automatically restarted and the counter is reloaded from REG0R (or REG1R when selected in Biload Mode). Continuous Mode is entered by resetting TMR bit C0.

10.2.3 Trigger And Retrigger Mode

A trigger event may be generated either by software action (setting either CP0 or CP1 bit in timer register FLAGR), or by an external source which may be programmed to be active on the rising edge, the falling edge or both, using the fields A0-A1 and B0-B1 in ICR.

In One Shot and Trigger Mode, every trigger event (used as a reload and start count) arriving before an End Of Count, is masked. In One Shot and Retrigger Mode, every trigger (used as a reload and start count) received while the counter is running automatically reloads the counter from REG0R (or REG1R when the register is selected in Biload Mode). Trigger/Retrigger Mode is set by the REN bit in TMR.

T0INA input refers to REG0R and T0INB input refers to REG1R.

10.2.4 Gate Mode

In this mode the counting operation is performed only when the external gate input is active (logical state "0"). The selection of T0INA or T0INB input as gate input is made through IN0-IN3 bits in ICR.

10.2.5 Capture Mode

REG0R and REG1R registers may be independently set in Capture Mode by setting RM0 or RM1 in TMR, so that a capture of the current count value can be performed either on REG0R or

REG1R, via software action (by setting CP0 or CP1 in the FLAGR register) or a programmable event on the external input pins.

WARNING: Care should be taken when two software captures have to be performed on the same register. In this case, at least one extra instruction must be present between the first CP0/CP1 bit set and the subsequent CP0/CP1 bit reset.

10.2.6 Up/Down Mode

The counter can count up or down depending on the state of the UDC bit (Software Up/Down) in TCR, or on the configuration of the external input pins, which have priority over UDC (see Input pin assignment in ICR). When read, the UDCS bit always returns the counter up/down current status (see also the Up/Down Autodiscrimination mode in the Input Pin Assignment Section, section 10.3.13).

10.2.7 Free Running Mode

The timer performs full range counting (in up or down mode) without reloading from REG0R at an End Of Count. This mode is automatically selected either in Bicapture Mode or by setting REG0R for capture function (Continuous Mode must also be set). In Autoclear Mode, free running with modulo less than 2^{16} may be obtained (see Autoclear Mode).

10.2.8 Monitor Mode

When RM1 bit in TMR is reset and the timer is not in Bivalue Mode, then REG1R acts as monitor, reproducing the current U/D counter content enabling the ST9 to read the counter "on the fly".

10.2.9 Autoclear Mode

A clear command forces the counter to the value 0000h or 0FFFFh, when counting in up or down count mode respectively. The counter reset may be obtained either directly, through CCL bit in TCR, or by entering the Autoclear Mode, through CCP0 and CCMP0 fields in TCR.

Every capture performed on REG0R (if CCP0 = "1"), or every successful compare performed by CMP0R (if CCMP0 = "1"), clears the counter and reloads the prescaler.

The Clear On Capture mode allows the direct measurement of delta time between successive captures on REG0R, while the Clear On Compare mode allows free running with modulo less than 2^{16}.

10.2.10 Bivalue Mode

Depending on the value of RM0 bit in TMR, the Biload Mode (RM0 = "0") or the Bicapture Mode (RM0 = "1") can be selected as explained in the following table:

Table 10-1. Bivalue Modes

TMR bits			Timer Operating Modes
RM0	RM1	BM	
0	X	1	Biload mode
1	X	1	Bicapture mode

A) Biload Mode

The Biload Mode is entered by selecting the Bivalue Mode (BM = "1" in TMR) and programming REG0R as a reload register (RM0 = "0" in TMR).

At any End Of Count, the counter reloading is performed alternately from REG0R and REG1R, (a low level for BM bit always sets REG0R as the current register, so that, after a Low to High transition of BM bit, the first reload is always from REG0R).

Every software or external trigger event on REG0R performs a reload from REG0R resetting the Biload cycle. In One Shot mode (reload made by a software or external trigger), the reload is always from REG0R.

B) Bicapture Mode

The Bicapture Mode is entered selecting the Bivalue Mode (BM = "1" in TMR) and programming REG0R as a capture register (RM0 = "1" in TMR).

Every capture event, software simulated (setting CP0 flag) or coming from T0INA input line, captures the current counter value alternately into REG0R and REG1R. A low level for BM bit always sets REG0R as current register, so that the first capture, after setting BM bit, is always into REG0R.

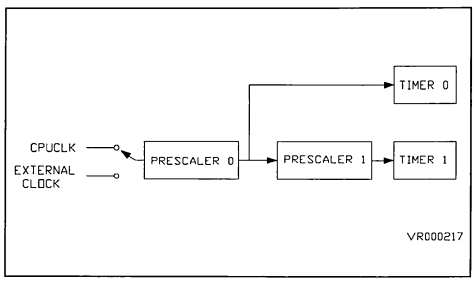
10.2.11 Parallel Mode

When there are two timers on ST9 chip, the parallel mode is entered with ECK ="1" in TMR of Timer 1. Timer 1 prescaler input is internally connected to the Timer 0 prescaler output. Timer 0 prescaler input may be connected to the system clock line or external input pin (depending on IN0-IN3 configuration in ICR) if ECK = "0" (in TMR of Timer0).

By loading the Prescaler Register of Timer 1 with the value 00h the two timers (Timer 0 and Timer 1) are driven by the same frequency in parallel mode.

The parallel mode can also be used for other timer pairs (Timer 2 and Timer 3), where available.

Figure 10-2. Parallel Mode Description



10.2.12 Autodiscriminator Mode

The phase difference sign of two overlapped pulses (respectively on TxINB and TxINA) generates a one step up(down) count, so that the up/down control and the counter clock are both external. The setting of the UDC bit in the TCR register has no effect in this configuration.

This mode is especially useful in determining the rotation direction.

10.3 INPUT PIN ASSIGNMENT

The two external inputs (TxINA and TxINB) of the timer can be individually configured to catch a particular external event (i.e. rising edge, falling edge, rising and falling edges) by programming the two relevant bits (A0, A1 and B0, B1) for each input in the external Input Control Register (ICR).

The 16 different functional modes of the two external inputs can be selected by programming IN0 - IN3 bits of the ICR as explained in the following table 10.2.

Table 10-2. Input Pin Function

I C Reg. IN3-IN0 bits	TxINA Input Function	TxINB Input Function
0000	I/O	I/O
0001	I/O	Trigger
0010	Gate	I/O
0011	Gate	Trigger
0100	I/O	Ext. Clock
0101	Trigger	I/O
0110	Gate	Ext. Clock
0111	Trigger	Trigger
1000	Clock Up	Clock Down
1001	Up/Down	Ext. Clock
1010	Trigger Up	Trigger Down
1011	Up/Down	I/O
1100	Autodiscr.	Autodiscr.
1101	Trigger	Ext. Clock
1110	Ext. Clock	Trigger
1111	Trigger	Gate

Some choices in the external input pin assignment are defined in conjunction with RM0 and RM1 bits in TMR.

For input pin assignment codes using the input pins as Trigger Inputs (except for code 1010, Trigger Up:Trigger Down):

- a trigger signal on TxINA input pin performs an U/D counter load if RM0 ="0", or an external capture if RM0 = "1".
- a trigger signal on TxINB input pin always performs an external capture on REG1R. The TxINB input pin is disabled when the Bivalue Mode is set.

NOTE: For proper operation of the External Input pins, the following must be observed:

– the minimum external clock/trigger pulse width cannot be less than the system clock (INTCLK) period if the input pin is programmed as rising or falling edge sensitive.

– the minimum external clock/trigger pulse width cannot be less than the prescaler clock period (INTCLK/3) if the input pin is programmed as rising and falling edges sensitive (valid also in Autodiscrimination mode). - the minimum delay between two clock/trigger pulse active edges must be greater than the prescaler clock period (INTCLK/3), while the minimum delay between two consecutive clock/trigger pulses must be greater than the system clock (INTCLK) period.

– the minimum gate pulse width must be at least twice the prescaler clock period (INTCLK/3).

– in Autodiscrimination mode, the minimum delay between the input pin A pulse edge (inside the input pin B pulse) and the edges of the input pin B pulse, must be at least the system clock (INTCLK) period.

– if a number N of external pulses must be counted using a Compare Register of a Timer in External Clock mode, then the Compare Register used must be loaded with the value [X +/- (N-1)], where X is the starting counter value and the sign is chosen depending if in Up or Down count mode respectively.

Here below is a description of the sixteen external input functional modes referring to table 10.2.

10.3.1 TxINA = I/O - TxINB = I/O

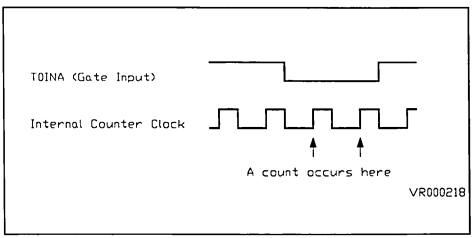
Input pins A and B are general purpose I/O bits. The counter clock is internally generated and the up/down control may be done only by software action through the UDC (Software Up/Down) bit in the TCR register.

10.3.2 TxINA = I/O - TxINB = Trigger

The signal applied to input pin B acts as a trigger signal on REG1R register. The prescaler clock is internally generated and the up/down control may be done only by software action through the UDC bit in the TCR register.

10.3.3 TxINA = Gate - TxINB = I/O

The signal applied to input pin A acts as a gate signal for the internal clock (i.e. the counter runs only when the gate signal is at a low level). The counter clock is internally generated and the up/down control may be done only by software action through the UDC bit in the TCR register.



10.3.4 TxINA = Gate - TxINB = Trigger

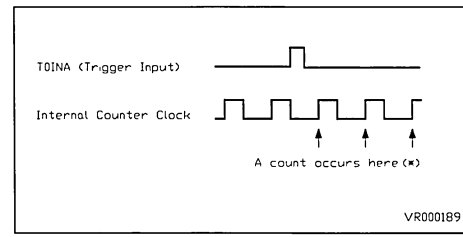
Both input pins A and B are connected to the timer, with the resulting effect of combining the actions due to the above explained configurations 10.3.2 and 10.3.3.

10.3.5 TxINA = I/O - TxINB = Ext. Clock

The signal applied to input pin B is used as the external clock for the prescaler. The up/down control may be done only by software action through the UDC bit in the TCR register.

10.3.6 TxINA = Trigger - TxINB = I/O

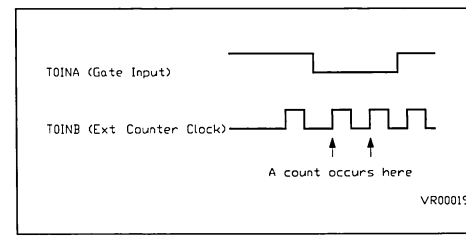
The signal applied to input pin A acts as a trigger signal on REG0R register performing the action for which the register was programmed (i.e. a reload or capture). The prescaler clock is internally generated and the up/down control may be done only by software action through the UDC bit in the TCR register.



(*) The timer is in One shot mode and REG0R in Reload mode

10.3.7 TxINA = Gate - TxINB = Ext. Clock

The signal applied to input pin B, gated by the signal applied to input pin A, acts as external clock for the prescaler. The up/down control may be done only by software action through the UDC bit in the TCR register.

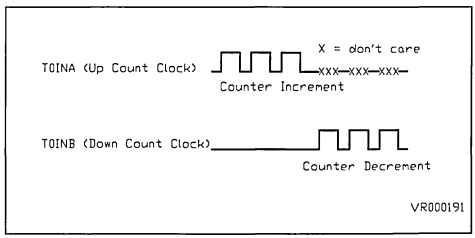


10.3.8 TxINA = Trigger - TxINB = Trigger

The signal applied to input pin A (or B) acts as trigger signal for the REG0R (or REG1R) register performing the action for which the register has been programmed. The counter clock is internally generated and the up/down control may be done only by software action through the UDC bit in the TCR register.

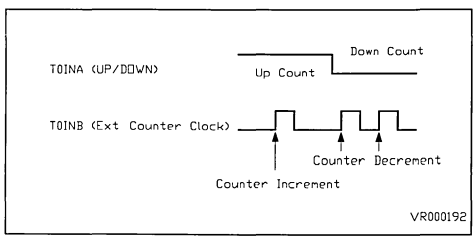
10.3.9 TxINA = Clock Up - TxINB = Clock Down

The pulse received on input pin A (or B) performs a one step up (or down) count, so that the counter clock and the up/down control are external. Setting the UDC bit in the TCR register has no effect in this configuration while input pin B has priority on input pin A.



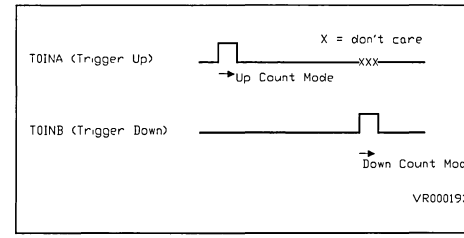
10.3.10 TxINA = Up/Down - TxINB = Ext Clock

An High (or Low) level of the signal applied on input pin A sets the counter in the up (or down) count mode, while the signal applied to input pin B is used as clock for the prescaler. Setting the UDC bit in the TCR register has no effect in this configuration.



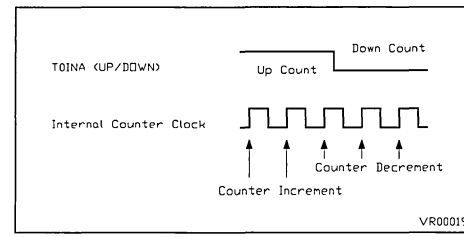
10.3.11 TxINA = Trigger Up - TxINB = Trigger Down

Up/down control is performed through both input pins A and B. A pulse on input pin A sets the up count mode, while a pulse on input pin B (which has priority on input pin A) sets the down count mode. The counter clock is internally generated while setting the UDC bit in the TCR register has no effect in this configuration.



10.3.12 TxINA = Up/Down - TxINB = I/O

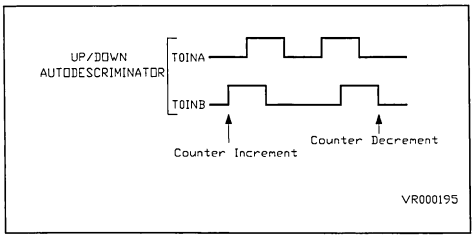
An High (or Low) level of the signal applied on input pin A sets the counter in the up (or down) count mode. The counter clock is internally generated. Setting the UDC bit in the TCR register has no effect in this configuration.



10.3.13 Autodiscrimination Mode

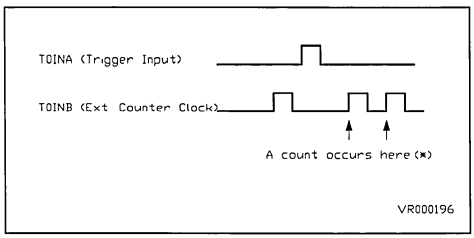
The phase between two pulses (respectively on input pin B and input pin A) generates a one step up (or down) count, so that the up/down control and the counter clock are both external. Thus, if the rising edge of TxINB arrives when TxINA is at level "0" the timer is incremented (no action if the rising edge of TxINB is coming when TxINA is at level "1"). If the falling edge of TxINB arrives when TxINA is at level "0" the timer is decremented (no action if the falling edge of TxINB arrives when TxINA is at level "1").

Setting the UDC bit in the TCR register has no effect in this configuration.



10.3.14 TxINA = Trigger - TxINB = Ext. Clock

The signal applied to input pin A acts as a trigger signal on REG0R register performing the action for which the register was programmed (i.e. a reload or capture), while the signal applied to input pin B is used as clock for the prescaler.



(*) The timer is in One shot mode and REG0R in reload mode

10.3.15 TxINA = Ext. Clock - TxINB = Trigger

The signal applied to input pin B acts as a trigger, performing a capture on REG1R register, while the signal applied to the input pin A is used as clock for the prescaler.

10.3.16 TxINA = Trigger - TxINB = Gate

The signal applied to input pin A acts as a trigger signal on REG0R register performing the action for which the register was programmed (i.e. a reload or capture), while the signal applied to input pin B acts as a gate signal for the internal clock (i.e. the counter runs only when the gate signal is at a low level).

10.4 OUTPUT PIN ASSIGNMENT

Two external outputs are available for each timer when programmed as alternate functions of the I/O pins.

Two registers for every timer, Output A Control Register (OACR) and Output B Control Register (OBCR) define the driver for the outputs and the actions to be performed.

Each of the two output pins can be driven from any of the three possible sources:

- Compare Register 0 event logic
- Compare Register 1 event logic
- Overflow/Underflow event logic.

Each of these three sources can cause one of the following four effects on any of the two outputs:

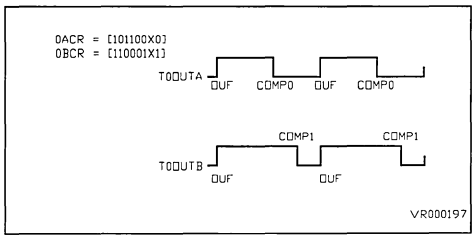
- Nop
- Set
- Reset
- Toggle.

Furthermore an On Chip Event signal can be driven by two of the three sources: the Over/Underflow event and Compare 0 event by programming the CEV bit of the OACR register and the OEV bit of OBCR register respectively. This signal can be used for another on-chip peripheral or as strobe for an I/O port (see I/O chapter).

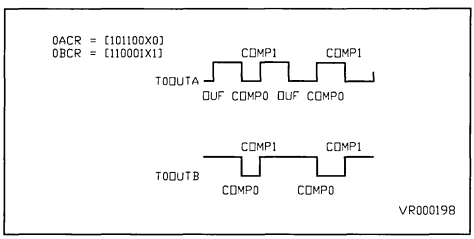
10.4.1 Output Waveforms

Depending on the different programmed values of OACR and OBCR the following example waveforms can be generated on TxOUTA and TxOUTB pins.

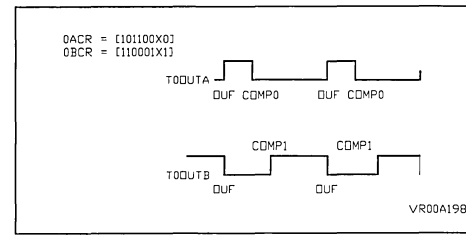
1) Configuration where TxOUTA is driven by Over/Underflow (OUF) and Compare 0 event (CM0), while TxOUTB is driven by the Over/Underflow and Compare 1 event (CM1).
OACR is programmed with TxOUTA preset to "0", OUF sets TxOUTA, CM0 resets T0OUTA and CM1 does not affect the output.
OBCR is programmed with TxOUTB preset to "0", OUF sets TxOUTB, CM1 resets TxOUTB while CM0 does not affect the output.



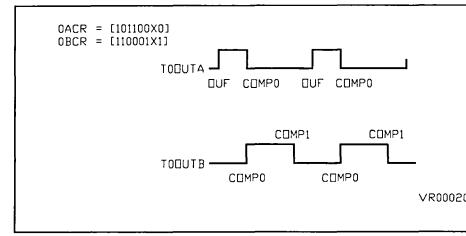
2) Configuration where TxOUTA is driven by Over/Underflow, Compare 0 and Compare 1, while TxOUTB is driven by both Compare 0 and Compare 1.
OACR is programmed with TxOUTA preset to "0". OUF toggles the Output 0 as do CM0 and CM1.
OBCR is programmed with TxOUTB preset to "1". OUF does not affect the output while CM0 resets TxOUTB and CM1 sets it.



3) Configuration where TxOUTA is driven by Over/Underflow and Compare 0, while TxOUTB is driven by Over/Underflow and Compare 1.
OACR is programmed with TxOUTA preset to "0". OUF sets TxOUTA while CM0 resets it and CM1 has no affect.
OBCR is programmed with TxOUTB preset to "1". OUF toggles TxOUTB, CM1 sets it and CM0 has no affect.



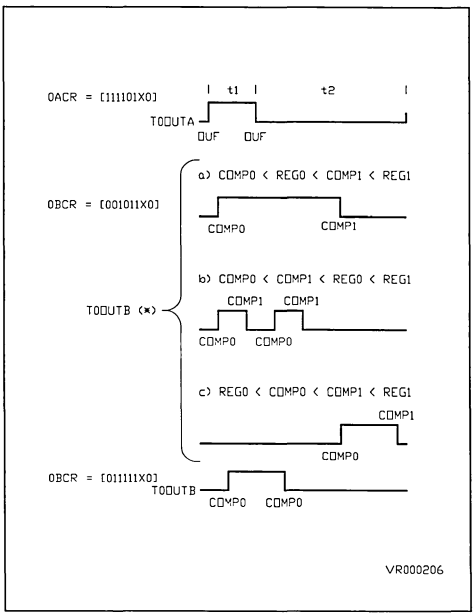
4) Configuration where TxOUTA is driven by Over/Underflow and Compare 0, while TxOUTB is driven by Compare 0 and 1.
OACR is programmed with TxOUTA preset to "1". OUF sets TxOUTA, CM0 resets it and CM1 has no affect.
OBCR is programmed with TxOUTB preset to "0". OUF has no affect, CM0 sets TxOUTB and CM1 toggles it.



10.4.2 Output Waveform Samples In Biload Mode

TxOUTA is programmed to monitor the two time intervals (t1 and t2) of the Biload Mode while TxOUTB is independent from the Over/Underflow and is driven by the different values of Compare 0 and Compare 1.
OACR is programmed with TxOUTA preset to "0". OUF toggles the output and CM0 and CM1 do not affect TxOUTA.
OBCR is programmed with TxOUTB preset to "0". OUF has no effect, while CM1 resets TxOUTB and CM0 sets it.

Depending on the CM1/CM0 values, three different example waveforms have been drawn starting from the above mentioned configuration of OBCR. In the last case, with a different programmed value of OBCR, only Compare 0 drives TxOUTB, toggling the output.



Note (*) Depending on the CMP1R/CMP0R values

10.5 TIMER INTERRUPT STRUCTURE

Reader should refer to the Interrupt chapter of this manual for more details of the ST9 Interrupt architecture.

The timer has 5 different Interrupt sources, grouped into 3 independent groups, assigned to the following Interrupt vectors:

Table 10-5. Timer Interrupt Structure

Interrupt Source	Vector Address
COMP 0 COMP 1	xxxx x110
CAPT 0 CAPT 1	xxxx x100
Overflow/Underflow	xxxx x000

The three least significant bits of the vector pointer address represent the relative priority assigned to each group, (000 value is the highest priority level) and are fixed by hardware depending on the source which generates the interrupt request. The 5 most significant bits are programmed by the user in the Interrupt Vector Register (IVR) of each Timer.

Each source can be masked by a dedicated bit in the Interrupt/DMA Mask Register (IDMR) of each timer, as well as a global mask enable bit (IDMR.7), masking all interrupts.

If an interrupt request (CM0 or CP0) happens before the corresponding pending bit is reset, an overrun condition occurs. This condition is flagged in two dedicated overrun bits, concerning the Comp0 and Capt0 sources, and placed in the Timer Flag Register (FLAGR).

A map of the Interrupt and DMA Registers available for every timer (the absolute address of every register and the meaning of every bit is detailed in the paragraph 10.8).

10.6 TIMER DMA STRUCTURE

Two Independent DMA channels, associated to Compare 0 and Capture 0 sources, respectively allow DMA transfers from Register File/Memory to Comp0 Register and vice versa from Capt0 Register to Register File/Memory (also transfers in/from Memory from/into an I/O port are available; see par. 10.9). Their priority is hardware set as following:

– Compare 0 Destination Lower Priority

– Capture 0 Source Higher Priority

The two DMA request sources are independently maskable by two DMA Mask bits, mapped in the Timer Interrupt/DMA Mask register (IDMR).

The two End of Block procedures, associated to each Interrupt mask and DMA mask combination, follow the standard architecture as shown in the Interrupt and DMA chapter in this manual.

10.6.1 DMA Pointers

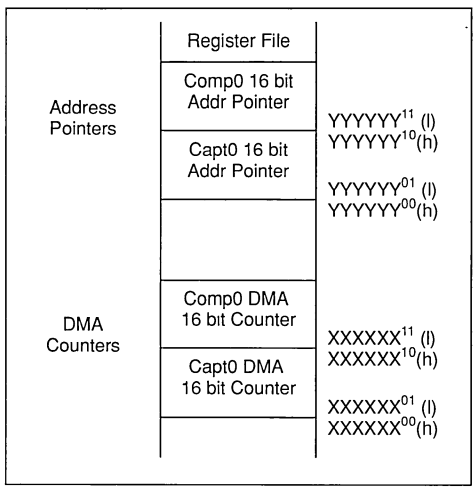
The 6 programmable most significant bits of the Timer Address and Counter Pointer registers (DAPR-DCPR) are common to both channels (Comp0 and Capt0 sources). As a consequence, the Comp0 and Capt0 Address pointers are mapped by pair in the Register File, as well as the Comp0 and Capt0 DMA Counter pair.

The different address specification, in order to point either Capt0 or Comp0 pointers, is provided by ST9 according to the channel under service (replacing the address bit 1 with "0" for CAPT0 or with "1" for COMP0), when D0 bit on DCPR register is equal to zero (Word address in Register File). In this condition (register with program/data memory transfer), the pointers will be split in two groups of adjacent Address pointer and Counter pairs respectively.

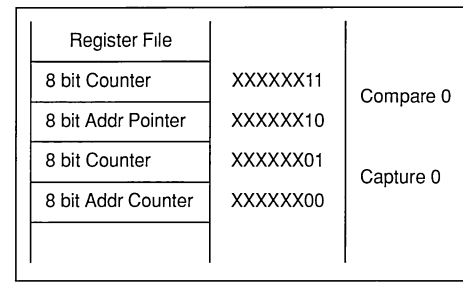
In the case of register to register transfers (selected by programming the value "1" into bit 0 of the DCPR register), only one pair of pointers are required and the pointers are mapped into one group of adjacent positions.

DAPR (the DMA/Address Pointer Register) in this case in not used, but must be considered reserved.

MAP POINTER FOR REGISTER TO PROG/DATA MEMORY TRANSFER



MAP POINTER FOR REGISTER TO REGISTER TRANSFER



10.6.2 Priority During The DMA Transactions

Each Timer DMA transaction is a 16 bit operation, therefore two different bytes must be transferred subsequently. This is accomplished by two DMA transfers. In order to speed up each word transfer, the second byte transfer is executed by forcing automatically the peripheral priority to the highest level (000) regardless to the previous set level. It will be then restored to the original value after executing this transfer. Furthermore, once one request is being served, its hardware priority is kept at the highest level regardless to the other Timer internal sources, i.e. once a Comp0 request is being served, it keeps a higher priority on the Capt0 channel, even if a Capt0 request occurs between the two byte transfers.

10.6.3 The DMA Swap Mode

After a complete data table transfer, the transaction counter is reset and an End Of Block condition occurs, the block transfer is completed.

The End Of Block Interrupt routine has at this point to reload both address and counter pointers of the channel referred by the End Of Block interrupt source if the application requires a continuous high speed data flow. This procedure causes speed limitations because of the time consumed by the reload routine.

The SWAP feature overcomes this drawback, allowing high speed continous transfers. Bit 2 of the Timer Address and Counter Pointer registers (DAPR-DCPR), toggles after any End Of Block condition, alternately providing odd and even address (D2-D7) for the couple of pointers, thus pointing to an updated couple, after a block has been completely transfered. This allows the User to be updating or reading the first block, and update the pointer values while the second is being transferred. These two toggle bits are software writable and readable, mapped in DCPR bit 2 for the CM0 channel, and in DAPR bit 2 for the CP0 channel (though a DMA event on a channel, in Swap mode, modifies a field in DAPR and DCPR common to both channels, the DAPR/DCPR content used in the transfer is always the one related to the correct channel).

The SWAP mode can be enabled by a control bit placed in the Interrupt Control Register.

WARNING: this mode is always set for both channel (CM0 and CP0).

10.6.4 The DMA End Of Block Interrupt Routine

This Interrupt request is generated after each block transfer (EOB) and its priority is the same as assigned in the usual Interrupt request, for the two channels. As a consequence, they will be served only when no DMA request occurs, and will be submitted to a possible OUF Interrupt request, which has higher priority.

Here is a typical EOB procedure (with swap mode enabled):

- Toggle bit test and Jump
- Pointers (odd or even depending on toggle bit status) reload
- Reset EOB bit: this bit must be reset only after the old couple of pointers has been restored, so that, if a new EOB condition occurs, the next pointers are ready to be swapped
- Verify the software protection condition
- Read the corresponding Overrun bit: this make the user sure that NO DMA request has been lost meantime
- Return.

WARNING: The EOB bits are read/write bits only for testing reasons. Writing a logical "1" by software (when SWEN bit is set) will cause a spurious interrupt request. During normal operation, these bits have only to be reset by software.

10.6.5 DMA Software Protection

A second EOB condition may occur before the first EOB routine is completed, this would cause a not yet updated pointer couple to be addressed, with consequent overwriting of memory. To prevent these errors, a protection mechanism is provided, such that the attempted setting of the EOB bit before it has been reset by software will cause the DMA mask on that channel to be reset (DMA disabled), locking any further DMA operation. As shown above, this mask bit should always be checked in each EOB routine, to ensure all DMA transfers are properly served.

10.7 TIMER DMA EXTERNAL MODES ON I/O PORTS

Each Timer DMA channel can also be employed in external transfers to/from memory from/to an I/O port. In this case only Byte transfers are executed for any request. Two control bits (DCTS and DCTD) in the Interrupt/DMA Control Register (IDCR) set each channel in INT/EXT (Internal = Register to Memory/External = Memory to/from I/O ports) mode.

The relevant I/O port must then be programmed in DMA mode and the right direction of the port chosen by the HDCxR register of that port (see I/O port chapter).

The two modes, however, are not the same for both channels as explained in the following section.

10.7.1 CM0 Channel External Mode

This mode is enabled when DCTD (DMA Compare Transaction Destination) bit is equal to "1" in the IDCR register.

This mode allows only Output transfers, from Register File/memory to the I/O port, under a request caused by a CM0 event or a software request (writing "1" in the CM0 flag). An application for this is a data flow under DMA to be output at fixed times.

The synchronization with the I/O port is accomplished by an internal signal, active when the data to be transfered is present on the internal Data Bus. If programmed, the on-chip event pulse can also be generated and used to strobe the output data on the selected handshake port.

In either case the DMA Output mode must be selected in the HDCTL Register of the port (see I/O port chapter).

10.7.2 CP0 Channel In External Mode

This mode is enabled when DCTS (DMA Capture Transaction Source) bit is equal to "1" in the IDCR register.

This mode allows bi-directional transfers controlled (when the I/O port is programmed in DMA Input/Output mode in the HDCTL register) by the value of the DD bit of the HDCTL register (the DD bit selects the DMA input or DMA Output mode).

The DMA request can be either an External CPT0 request (Timer External input A) or a software request (by writing "1" in the CP0 Flag).

This, along with a further internal synchronization signal, generated by the Timer Unit, allows handshake operations managed by the I/O port while the direction of the data to read or write on the I/O port is fixed by the value of the DD bit in the HDCTL register (see I/O port chapter).

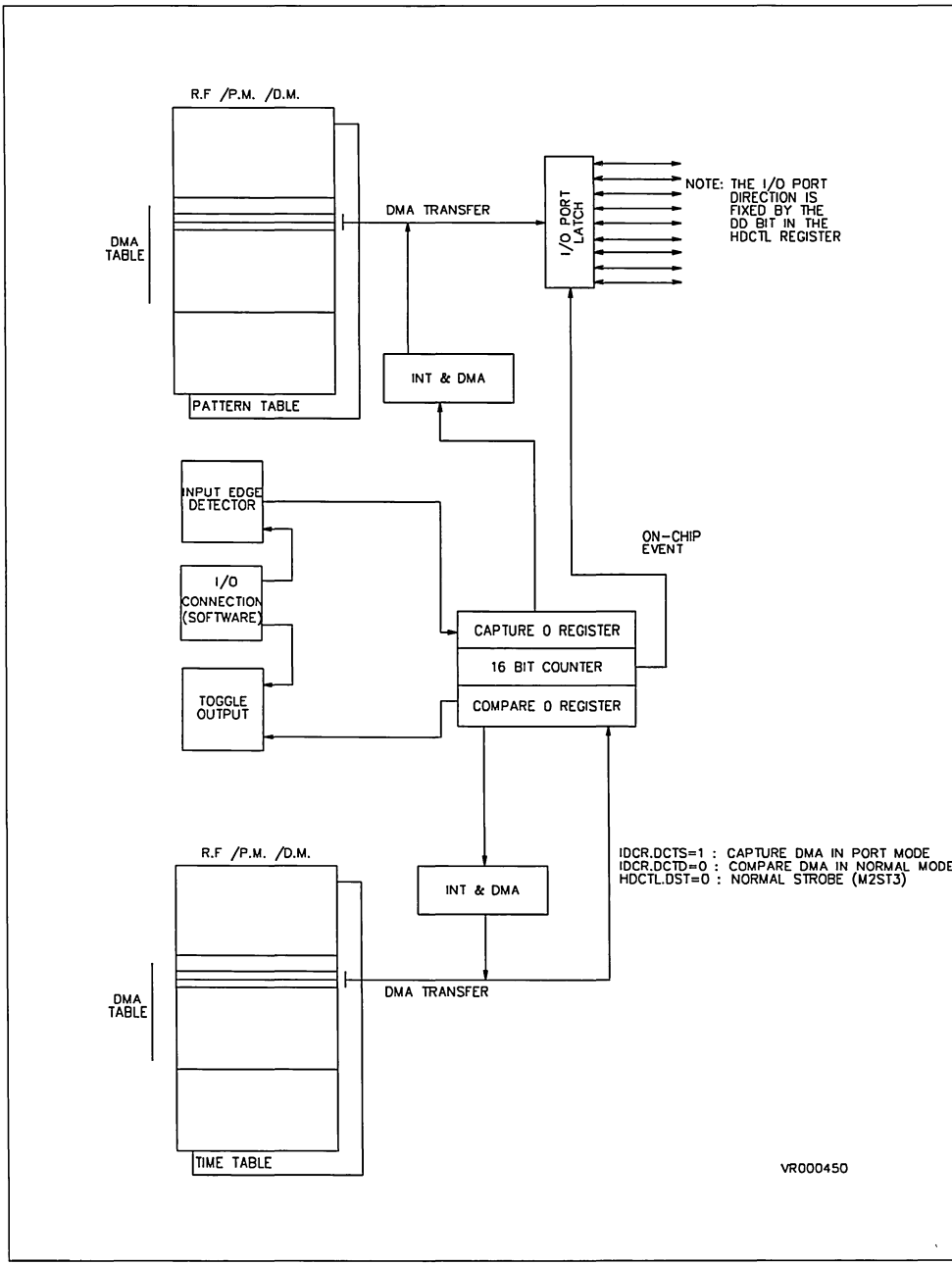
10.7.3 DMA Channel Synchronization

A CP0 DMA request can be generated also by a CM0 event, simply by setting the Timer External Input A on rising and falling edges sensitive, connecting it by hardware or software (though the IOCR register) to the Timer OUT 0, and programming the CM0 action as output toggle.

This will cause a CP0 request to be generated after each CM0 condition, thus synchronizing the 2 DMA channels (see the following application example).

The DCTS bit must be set and DCTD bit must be reset in the IDCR register. Fig 10.3 shows an example of two channel synchronisation. A new byte will be sent out through the I/O port at an interval specified by the COMP0 value mapped in the look-up table.

Figure 10-3. Timer DMA Channels Synchronization



10.8 MULTIFUNCTION TIMER REGISTER DESCRIPTION

Twenty-one control and data registers are associated to each Multifunction timer, and are located in the Group F I/O pages of the ST9 Register File.

The registers of Timer 0 are located (in the case of at least one timer on the ST9 device, otherwise these are reserved registers) into I/O page 10 and page 9 as follows:

Page 10 (0Ah)		Page 9 (09h)
IMDR - TIM0	-15-	
FLAGR - TIM0		
OBCR - TIM0		
OACR - TIM0		
PRSR - TIM0		
ICR - TIM0		
TMR - TIM0		
TCR - TIM0		IOCR
CMP1LR - TIM0		
CMP1HR - TIM0		
CMP0LR - TIM0		
CMP0HR - TIM0		
REG1LR - TIM0		IDCR - TIM0
REG1HR - TIM0		IVR - TIM0
REG0LR - TIM0		DAPR - TIM0
REG0HR - TIM0	-00-	DCPR - TIM0

The registers of Timer 1 are located (in the case of at least two timers on the ST9 device, otherwise are reserved registers) into I/O page 8 and page 9 as follows:

Page 8 (08h)		Page 9 (09h)
IMDR - TIM1	-15-	
FLAGR - TIM1		
OBCR - TIM1		
OACR - TIM1		
PRSR - TIM1		
ICR - TIM1		
TMR - TIM1		
TCR - TIM1		IOCR
CMP1LR - TIM1		IDCR - TIM1
CMP1HR - TIM1		IVR - TIM1
CMP0LR - TIM1		DAPR - TIM1
CMP0HR - TIM1		DCPR - TIM1
REG1LR - TIM1		
REG1HR - TIM1		
REG0LR - TIM1		
REG0HR - TIM1	-00-	

The registers of Timer 2 are located (in the case of at least three timers on the ST9 device, otherwise are reserved registers) into I/O page 14 and page 13 as follows:

IMDR - TIM2	-15-	
FLAGR - TIM2		
OBCR - TIM2		
OACR - TIM2		
PRSR - TIM2		
ICR - TIM2		
TMR - TIM2		
TCR - TIM2		IOCR
CMP1LR - TIM2		
CMP1HR - TIM2		
CMP0LR - TIM2		
CMP0HR - TIM2		
REG1LR - TIM2		IDCR - TIM2
REG1HR - TIM2		IVR - TIM2
REG0LR - TIM2		DAPR - TIM2
REG0HR - TIM2	-00-	DCPR - TIM2
Page 14 (0Eh)		Page 13 (0Dh)

The registers of Timer 3 are located (in the case of four timers on the ST9 device, otherwise are reserved registers) into I/O page 12 and page 13 as follows:

IMDR - TIM3	-15-	
FLAGR - TIM3		
OBCR - TIM3		
OACR - TIM3		
PRSR - TIM3		
ICR - TIM3		
TMR - TIM3		
TCR - TIM3		IOCR
CMP1LR - TIM3		IDCR - TIM3
CMP1HR - TIM3		IVR - TIM3
CMP0LR - TIM3		DAPR - TIM3
CMP0HR - TIM3		DCPR - TIM3
REG1LR - TIM3		
REG1HR - TIM3		
REG0LR - TIM3		
REG0HR - TIM3	-00-	
Page 12 (0Ch)		Page 13 (0Dh)

In the following pages there is a detailed description of every register with the meaning and the function of every bit. The register is referred without the absolute address which is depending on the number of the timer used (of course the configuration and the functions of the internal bits of i.e. TCR - TIM0 are the same of TCR - TIM1 and so on.

10.8.1 Register 0 (REG0R) Registers

This couple of registers (REG0LR and REG0HR) is used to capture values from the U/D counter or to load preset values into the U/D counter.

REG0HR R240 (F0h) Read/Write
Capture Load Register 0 (High)

Reset value: undefined

7							0
R15	R14	R13	R12	R11	R10	R9	R8

REG0LR R241 (F1h) Read/Write
Capture Load Register 0 (Low)

Reset value: undefined

7							0
R7	R6	R5	R4	R3	R2	R1	R0

10.8.2 Register 1 (REG1R) Registers

This pair of registers (REG1LR and REG1HR) is used (as REG0R) to capture values from the U/D counter or to load preset values into the U/D counter.

REG1LR R243 (F3h) Read/Write
Capture Load Register 1 (Low)

Reset value: undefined

7							0
R7	R6	R5	R4	R3	R2	R1	R0

REG1HR R242 (F2h) Read/Write
Capture Load Register 1 (High)

Reset value: undefined

7							0
R15	R14	R13	R12	R11	R10	R9	R8

10.8.3 Compare 0 (CMP0R) Registers

This pair of Registers (CMP0L and CMP0H) is used to store 16 bit values to be compared to the U/D counter content.

CMPOLR R245 (F5h) Read/Write
Compare 0 Register (Low)

Reset value: undefined

7							0
R7	R6	R5	R4	R3	R2	R1	R0

CMPOHR R244 (F4h) Read/Write
Compare 0 Register (High)

Reset value: undefined

7							0
R15	R14	R13	R12	R11	R10	R9	R8

10.8.4 Compare 1 (CMP1R) Registers

This pair of Registers (CMP1L and CMP1H) is used (as CMP0R) to store 16 bit values to be compared to the U/D counter content.

CMP1LR R247 (F7h) Read/Write
Compare 1 Register (Low)

Reset value: undefined

7							0
R7	R6	R5	R4	R3	R2	R1	R0

CMP1HR R246 (F6h) Read/Write
Compare 1 Register (High)

Reset value: undefined

7							0
R15	R14	R13	R12	R11	R10	R9	R8

10.8.5 Timer Control Register (TCR)

This register is used to control the status of the timer.

TCR R248 (F8h) Read/Write Timer
Control Register

Reset value: 0000 0xxxb

7							0
CEN	CCP0	CCMP0	CCL	UDC	UDCS	OF0	CS

b7 = **CEN:** *Counter Enable.* This bit is ANDed with the Global Counter Enable bit (GCEN bit on R230 - Central Interrupt Control Register; the GCEN bit is set after the Reset cycle). Setting the CEN bit starts the counter and prescaler (without reload). When this bit is reset, the counter and prescaler stop.

b6 = **CCP0:** *Clear on Capture.* When this bit is set, a clear of the counter and a reload of the prescaler are performed on REG0R or REG1R capture. No effect when this bit is reset.

b5 = **CCMP0:** *Clear on Compare.* When this bit is set, a clear of the counter and a reload of the prescaler are performed on CMP0R compare. No effect when this bit is reset.

b4 = **CCL:** *Counter clear.* When this bit is set, the counter is cleared without generation of interrupt request. No effect when this bit is reset.

b3 = **UDC:** *Software Up/Down.* When the direction of the counter is not fixed by T0INA and/or T0INB (see par. 10.3) it can be software controlled by the UDC bit. Setting the UDC bit selects the Up mode counting. Resetting this bit the Down counting is performed.

b2 = **UDCS:** *Up/Down Count status.* This bit is read only and monitors the direction of the counter. Reading "1" means that the counter is using the Up mode counting. Reading "0" means that the Down mode counting is in use.

b1 = **OF0:** *OVF/UNF state.* This bit is read only and is set if an Overflow or an Underflow occurs during a Capture on Register 0.

b0 = **CS:** *Counter Status.* This bit is read only and monitors the status of the counter. Reading "1" means that the counter is running. Reading "0" indicates that the counter is halted.

10.8.6 Timer Mode Register (TMR)

This register is used to select the operating mode of the timer.

TMR R249 (F9h) Read/Write
Timer Mode Register

Reset value: 0000 0000b (00h)

7							0
0E1	0E0	BM	RM1	RM0	ECK	REN	C0

b7 = **OE1:** *Output 1 Enable.* Setting this bit enables the Output 1 (TxOUTB) of the relevant timer. When this bit is reset, the TxOUTB is disabled and forced to the logic state "1". The relevant I/O bit must also be set to Alternate Function.

b6 = **OE0:** *Output 0 Enable.* Setting this bit enables the Output 0 (TxOUTA) of the relevant timer. When this bit is reset, the TxOUTA is disabled and forced to the logic state "1". The relevant I/O bit must also be set to Alternate Function.

b5 = **BM:** *Bivalue Mode.* This bit enables the Bivalue mode when is set. When the bit is reset, the Bivalue mode is disabled. After that, depending on the value of RM0 bit (TMR - bit 3), the Biload or Bicapture mode is selected (see par. 10.4.10).

b4 = **RM1:** *REG1R mode.* When this bit is set, the REG1R can be used to capture the value of the counter. When the bit is reset, the REG1R monitors the value of the counter. The selection performed by this bit has no effect when the Bivalue Mode is enabled.

b3 = **RM0:** *REG0R mode.* When this bit is reset, the REG0R can be used to capture the value of the counter (also the Bicapture mode can be selected if the BM bit is equal to 1). When the bit is reset, the REG0R can be used to load the new value of the counter (also the Biload mode can be selected if the BM bit is equal to "1").

b2 = **ECK:** *Timer clocking mode.* This bit selects the clock source which drives the prescaler. When the ECK bit is reset, either the Internal or External clock is used depending on IN0 - IN3 configuration in ICR. When ECK bit is set, different functions are performed depending on the number of the relevant timer. For odd timers (Timer 1, Timer 3 and so on) setting the ECK bit enables the Parallel mode (see par. 10.4.11) where the prescaler of the odd timer is driven by the prescaler output of the even timer.

b1 = **REN:** *Retrigger mode.* When this bit is reset, the Retriggerable mode is enabled. When the bit is set, this operating mode is disabled.

b0 = **CO:** *Continous/One shot mode.* When this bit is reset, the Continuous mode is selected (with autoreload on condition). The bit must be set to select the one shot mode. The following table summarizes the different operating modes depending on the values of RM0, RM1 and BM bits.

Table 10-3. Timer Operating Modes

TMR Bits			Timer Operating Modes
RM0	RM1	BM	
0	X	1	Biload mode
1	X	1	Bicapture mode
0	0	0	Load from REG0R and Monitor on REG1R
0	1	0	Load from REG0R and Capture on REG1R
1	0	0	Capture on REG0R and Monitor on REG1R
1	1	0	Capture on REG0R and REG1R

10.8.7 External Input Control Register(ICR)

By this register it is possible to program the function and the operation to be performed on TxINA and TxINB inputs.

ICR R250 (FAh) read/write
External Input Control Register

Reset value: 0000 xxxxb (0Xh)

7							0
IN3	IN2	IN1	IN0	A0	A1	B0	B1

b7-b4 = **IN3,IN2,IN1,IN0:** *Input pin assignment.* The different functions of TxINA and TxINB inputs of every timer can be selected by IN0 - IN3 bits as explained below.

b3-b2 = **A0, A1:** *TxINA event programming.* The following TxINA configurations can be selected according to the values of A0 and A1 bits:

A0	A1	TxINA Configuration
0	0	No operation
0	1	Falling edge sensitive
1	0	Rising edge sensitive
1	1	Rising and falling edges

I C Reg. IN3-IN0 bits	TxINA Input Function	TxINB Input Function
0000	I/O	I/O
0001	I/O	Trigger
0010	Gate	I/O
0011	Gate	Trigger
0100	I/O	Ext. Clock
0101	Trigger	I/O
0110	Gate	Ext. Clock
0111	Trigger	Trigger
1000	Clock Up	Clock Down
1001	Up/Down	Ext. Clock
1010	Trigger Up	Trigger Down
1011	Up/Down	I/O
1100	Autodiscr.	Autodiscr.
1101	Trigger	Ext. Clock
1110	Ext. Clock	Trigger
1111	Trigger	Gate

b1-b0 = **B0, B1:** *TxINB event programming.* The following TxINB configurations can be selected according to the values of B0 and B1 bits:

B0	B1	TxINB Configuration
0	0	No operation
0	1	Falling edge sensitive
1	0	Rising edge sensitive
1	1	Rising and falling edges

10.8.8 Prescaler Register (PRSR)

This register holds the preset value for the 8-bit prescaler. The PRSR content may be modified at any time, but it will be loaded into the prescaler at the following prescaler underflow, or as a consequence of a counter reload (either by software or upon external request). On an external RESET condition, the prescaler is automatically loaded with the 00h value, so that the prescaler divides by 1 and the maximum counter clock is generated (OSCIN frequency divided by 6 when MODER.5 = DIV2 bit is set).

PRSR R251 (FBh) read/write
Prescaler Register

Reset value: 0000 0000b (00h)

7							0
P7	P6	P5	P4	P3	P2	P1	P0

The binary value stored (by programmer) in the PRSR register is equal to [divider value - 1]. For example, loading PRSR with 24 makes the prescaler divide by 25.

10.8.9 Output A Control Register (OACR)

This register selects the sources that can perform actions on TxOUTA pin.

TxOUTA can be driven from any of three possible sources:

- OVF/UNF being an Overflow or Underflow event on the U/D counter,
- COMP0 being a successful compare event on CMP0R register, and
- COMP1 being a successful compare event on CMP1R.

By programming bits B0 and B1 of the relevant source can cause one of the following four effects on TxOUTA (which can be previously preset):

B0	B1	Event
0	0	Set
0	1	Toggle
1	0	Reset
1	1	Nop

Note: In any case of contemporary events the action will be taken which results from 'ANDing' the B1-B0 fields. Through this register the action of COMP0 on the on-chip event can be also selected.

OACR R252 (FCh) Read/Write
Output A Control Register

Reset value: xxxx xx0xb

7							0
B0	B1	B0	B1	B0	B1	CEV	OP
< COMP0 >		< COMP1 >		< OVF/UNF >			

b7-b6 = **B0, B1:** *Control bits of COMP0.* Control bits for event driven by COMP0.

b5-b4 = **B0, B1:** *Control bits of COMP1.* Control bits for event driven by COMP1.

b3-b2 = **B0, B1:** *Control bits of OVF/UNF.* Control bits for event driven by OVF/UNF.

b1 = **CEV:** *On-Chip Event on CMP0R.* When this bit is set, a successful compare on CMP0R activates the on-chip event signal (a single pulse is generated). No action when this bit is reset.

b0 = **OP:** *Control bit of TxOUTA preset.* The value of this bit is the preset value of TxOUTA output pin. Reading this bit returns the current state of the TxOUTA output pin (i.e. useful when this output is selected in toggle mode).

10.8.10 Output B Control Register (OBCR)

This register selects the sources that can perform actions on TxOUTB output pin. TxOUTB can be driven from any of three possible sources:

- OVF/UNF being an Overflow or Underflow event on the U/D counter,
- COMP0 being a successful compare event on CMP0R register, and
- COMP1 being a successful compare event on CMP1R.

By programming bits B0 and B1 of the relevant source can cause one of the following four effects on TxOUTB (which can be previously preset):

B0	B1	Event
0	0	Set
0	1	Toggle
1	0	Reset
1	1	Nop

Note: In any case of contemporary events the action will be taken which results from 'ANDing' the B1-B0 fields Through this register the action of Overflow/Underflow on the on-chip event can be also selected

OBCR R253 (FD h) Read/Write
Output B Control Register

Reset value: xxxx xx0x b

7							0
B0	B1	B0	B1	B0	B1	OEV	OP
< COMP0 >		< COMP1 >		< OVF/UNF >			

b7-b6 = **B0, B1:** *control bits of COMP0.* Control bits for event driven by COMP0.

b5-b4 = **B0, B1:** *control bits of COMP1.* Control bits for event driven by COMP1.

b3-b2 = **B0, B1:** *control bits of OVF/UNF.* Control bits for event driven by OVF/UNF.

b1 = **OEV:** *On-Chip Event on OVF/UNF.* When this bit is set, a successful Overflow/Underflow activates the on-chip event signal (a single pulse is generated). No action when this bit is reset.

b0 = **OP:** *control bit of TxOUTB preset.* The value of this bit is the preset value of TxOUTB output pin. Reading this bit, it returns the current state of the TxOUTB output pin (i.e. useful when this output is selected in toggle mode).

10.8.11 Flag Register (FLAGR)

This register contains the flags of the successful captures or comparisons together with the Overflow/Underflow and overrunning indications. Also the mode of the Interrupt on capture can be selected. By writing into the capture flags it is possible to generate software captures. It is necessary to clear the capture flag before subsequent sofware captures can be generated. By reading this register, user can know which source has generated an interrupt (several sources may share the same interrupt vector).

FLAGR R254 (FEh) Read/Write
Flags Register

Reset value: 0000 0000b (00h)

7							0
CP0	CO1	CM0	CM1	OUF	OCP0	OCM0	A0

b7 = **CP0:** *Flag on Capture 0.* This bit is set after a capture on REG0R register. Writing "1" acts as a software load/capture from/on REG0R.

b6 = **CP1:** *Flag on Capture 1.* This bit is set after a capture on REG1R register. Writing "1" acts as a software capture on REG1R, except when in Bicapture mode.

b5 = **CM0:** *Flag on Compare 0.* This bit is set after a successful compare on CMP0R register.

b4 = **CM1:** *Flag on Compare 1.* This bit is set after a successful compare on CMP1R register.

b3 = **OUF:** *Flag on Overflow/Underflow.* This bit is set after a counter Over/Underflow condition.

b2 = **OCP0:** *Flag of overrun on Capture 0.* This bit is set when more than one INT/DMA request occurs before having reset the event flag CP0 or whenever a capture is software simulated.

b1 = **OCM0:** *Flag of overrun on Compare 0.* This bit is set when more than one INT/DMA request occurs before having reset the event flag CM0.

b0 = **AO:** *Capture Interrupt Function.* When this bit is set the Interrupt is generated by an AND function of REG0R/REG1R captures while when the AO bit is reset, the Interrupt is generated by an OR function of REG0R/REG1R captures.

10.8.12 Interrupt/DMA Mask Register (IDMR)

This register contains the Global Timer Interrupt enable bit and the INT/DMA enable bits of the following events:

- Capture on REG0R (CP0 field),
- Capture on REG1R (CP1I bit - only Interrupt mask),
- Compare on CMP0R (CM0 field),
- Compare on CMP1R (CM1I bit- only Interrupt mask), and
- Overflow/Underflow (OUI bit - only Interrupt mask).

IDMR R255 (FF h) Read/Write
Interrupt/DMA Mask Register

Reset value: 0000 0000b (00h)

7							0
GTIEN	CP0D	CP0I	CP1I	CM0D	CM0I	CM1I	OUI
	< CP0 >		< CP1 >	< CM0 >		< CM1 >	

b7 = **GTIEN:** *Global Timer Interrupt Enable.* When this bit is set, all the Interrupts (of the enabled sources) of the timer are enabled. When the bit is reset, all the Interrupts of timer are disabled.

b6 = **CP0D:** *Capture 0 DMA Mask.* Capture on REG0R DMA is enabled when CP0D = "1."

b5 = **CP0I:** *Capture 0 Interrupt Mask.* Capture on REG0R interrupt is enabled when CP0I = "1".

b4 = **CP1I:** *Capture 1 Interrupt Mask.* Capture on REG1R interrupt is enabled when CP1I = "1".

b3 = **CM0D:** *Compare 0 DMA Mask.* Compare on CMP0R DMA is enabled when CM0D = "1".

b3 = **CM0I:** *Compare 0 Interrupt Mask.* Compare on CMP0R interrupt is enabled when CM0I = "1".

b1 = **CM1I:** *Compare 1 Interrupt Mask.* Compare on CMP1R interrupt is enabled when CM1I = "1".

b0 = **OUI:** *Overflow/Underflow Interrupt Mask.* Overflow/Underflow condition interrupt is enabled when OUI = "1".

10.8.13 DMA Counter Pointer Register (DCPR)

This register is not used only as DMA Counter pointer but also to define the DMA area and the DMA source.

DCPR R24O or 244 (F0h or F4h) Read/Write
DMA Counter Pointer Register

Reset value: undefined

7							0
D7	D6	D5	D4	D3	D2	DMA SRCE	REG MEM

b7-b2 = **D7,....,D2:** *MSB of DMA counter register address.* Those bits contain the most significant bits of the DMA counter register address and are user programmable. Though user programmable, the D2 bit may be hardware toggled if the Swap mode is set for the Timer DMA section related to Compare 0 channel.

b1 = **DMA-SRCE:** *DMA source selection (hardware programmed).* This bit is hardware fixed by the Timer DMA logic and is set if the DMA destination is a Compare on CMP0R register and reset if the DMA source is a Capture on REG0R register.

b0 = **REG-MEM:** *DMA area selection.* When this bit is set, it selects the Source/Destination of the DMA area from/into Register File while when it is reset, the Source/Destination of the DMA area is from/to the External Program or Data Memory (according with the value of D0 bit in DAPR).

10.8.14 DMA Address Pointer Register (DAPR)

This register is not used only as DMA Address pointer but also to define the DMA area and the DMA source.

DAPR R241 or 245 (F1h or F5h) Read/Write
DMA Address Pointer Register

Reset value: undefined

7							0
D7	D6	D5	D4	D3	D2	DMA SRCE	PRG/ DAT

b7-b2 = **D7,...., D2:** *MSB of DMA Address register location.* Those bits contain the most significant bits of the DMA Address register location and are user programmable. Through user programmable, the bit D2 may be hardware toggled if the Swap mode is set for the Timer DMA section related to Capture 0 channel.

b1 = **DMA-SRCE:** *DMA source selection (hardware programmed).* This bit is hardware fixed by the Timer DMA logic and is set if the DMA destination is a Compare on CMP0R register and reset if the DMA source is a Capture on REG0R register.

b0 = **PRG/DAT:** *DMA memory selection.* When this bit is set it selects the Source/Destination of the DMA area from/into Data Memory while when it is reset the Source/Destination of the DMA area is from/into the External Program Memory (according with the value of D0 bit in DCPR).

DCPR.0	DAPR.0	DMA Source/Destination
0	0	Program memory
0	1	Data memory
1	0	Register file
1	1	Register file

10.8.15 Interrupt Vector Register (IVR)

This register is used as a vector pointing to the 16-bit interrupt vectors in the program memory which contain the starting addresses of the three interrupt subroutines managed by every timer.

Only one Interrupt Vector Register is available for every timer and is able to manage the three interrupt groups because the 3 less significant bits are fixed by hardware depending on the group which generated the interrupt request.

In order to understand which request generated the interrupt inside the same group, the FLAGR register can be used to check the relevant flag of the interrupt source.

IVR R242 or 246 (F2h or F6h) Read/Write
Interrupt Vector Register

Reset value: undefined

7							0
V4	V3	V2	V1	V0	W1	W0	D0

b7-b3 = **V4 to V0:** *MSB of the Vector address.* These bits are user programmable and contain the five most significant bits of the Timer interrupt vector addresses in the program memory. In any case, an 8-bit address can be used to indicate the Timer interrupt vector locations because they are within the first 256 locations of the program memory (see Interrupt and DMA chapter).

b2-b1 = **W1 and W0:** *Vector Address bits.* These bits are equivalent to bit 1 and bit 2 of the Timer interrupt vector addresses in the program memory. They are fixed by hardware depending on the group of sources which generated the interrupt request as follows:

b0 = **D0.** This bit is fixed by hardware. It always returns the value "0" if read.

W1	W0	Interrupt Source
0	0	Overflow/Underflow even interrupts
0	1	Not available
1	0	Capture event interrupts
1	1	Compare event interrupts

10.8.16 Interrupt/DMA Control Register (IDCR)

This register is used to control the Interrupt and DMA priority level, the DMA transfer source and destination and the Swap mode. This register contains also the two End Of Block bits.

IDCR R243 or 247 (F3h or F7h) Read/Write
Interrupt/DMA Control Register

Reset value: 1100 0111b (C7h)

7							0
CPE	CME	DCTS	DCTD	SWEN	PL2	PL1	PL0

b7 = **CPE:** *Capture 0 EOB.* This bit is set by hardware when the End Of Block condition is reached during a Capture 0 DMA operation with the Swap mode enabled. When the Swap mode is disabled (SWEN bit = "0") the CPE bit is forced by hardware to "1".

b6 = **CME:** *Compare 0 EOB.* This bit is set by hardware when the End Of Block condition is reached during a Compare 0 DMA operation with the Swap mode enabled. When the Swap mode is disabled (SWEN bit = "0") the CME bit is forced by hardware to "1".

b5 = **DCTS:** *DMA Capture Transfer Source.* This bit selects the source of the DMA operation related to the channel associated to the Capture 0. When the DCTS bit is reset the selected source is the REG0R register. When the DCTS bit is set the ST9 port is selected as DMA transfer source (with this DMA channel the ST9 port can also be destination depending on the value of the DD bit in the HDCTL register of the port - see I/O port chapter 9).

b4 = **DCTD:** *DMA Compare Transfer Destination.* This bit selects the destination of the DMA operation related to the channel associated to the Compare 0. When this bit is reset, the selected destination is the CMP0R register. When the bit is set, the ST9 port is selected as DMA transfer destination.

b3 = **SWEN:** *Swap function Enable.* When this bit is set, the Swap function is enabled for the two DMA channels. Resetting the SWEN bit disables the Swap mode.

b2-b0 = **PL2 to PL0:** *Interrupt/DMA priority level.* With these three bits it is possible to select the Interrupt and DMA priority level of every single timer within eight different levels (see Interrupt/DMA chapter).

10.8.17 I/O Connection Register (IOCR)

This register allows user to select (or not) an on-chip connection between input A and output A of one same timer.

IOCR R248 (F8h) Read/Write
I/O Connection Register

Reset value: 1111 1100b (0FCh)

7							0
						SC1	SC0

b7-b2 = not used.

b1 = **SC1:** *Select Connection Odd.* SC1 selects if connection between TxOUTA and TxINA for ODD timers (Timers 1, 3) is made on-chip or externally (physically on pins)

SC1 = "0": TxOUTA and TxINA unconnected

SC1 = "1": TxOUTA and TxINA connected internally

b0 = **SC0:** *Select Connection Even.* SC0 selects if connection between TxOUTA and TxINA for EVEN timers (Timers 0, 2) is made on-chip or externally (physically on pins)

SC0=0: TxOUTA and TxINA unconnected

SC0=1: TxOUTA and TxINA connected internally

ANALOG TO DIGITAL CONVERTER

11.1 MAIN FEATURES AND FUNCTIONAL DESCRIPTION

11.1.1 Features

The ST9 A/D Converter Unit is an 8 channel Analog to Digital converter, with 8 bit ± 1/2 LSB maximum DNL error, and a channel conversion time of 11µs at INTCLK = 12MHz (including sample and hold time).

The converter uses a fully differential analog input configuration for the best noise immunity and precision performance, along with two separate supply lines, allowing the best supply noise rejection and possible analog supplies lower than the Digital V_{CC}. In fact, the converted digital value, is referred to the Analog V_{CC} (AV_{CC}) as the full scale value, so that using, for example a value of 4 Volt for this supply, all conversions will accordingly refer to this 4 Volt full scale value ($AV_{SS} = V_{SS}$).

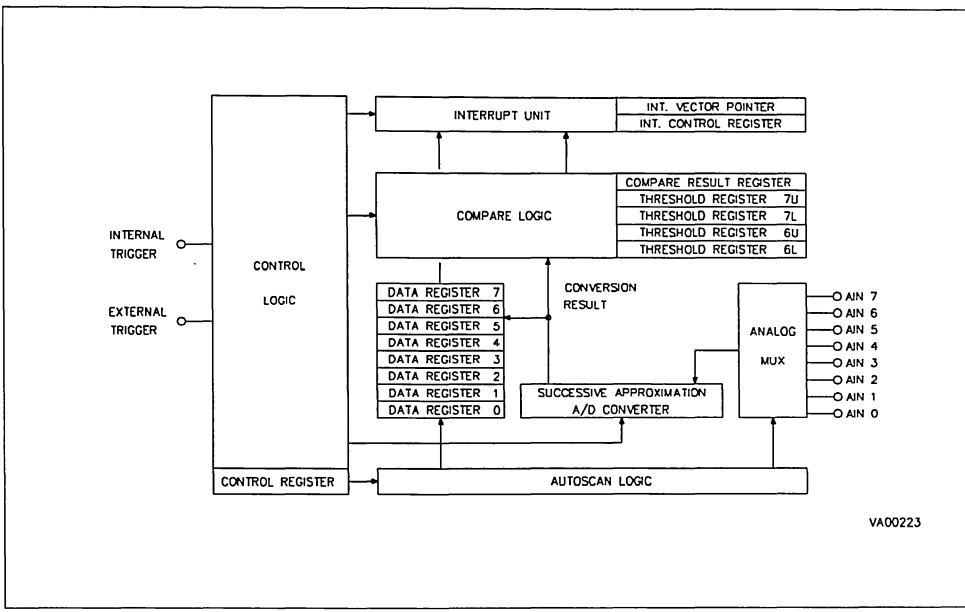
A internal, fast Sample and Hold cell, with a sample time of 3µs at INTCLK=12MHz, allows quick sampling of signal for the minimum warping effect and Integral conversion error.

Up to 8 multiplexed Analog Inputs are available. A group of signals can be converted sequentially by simply programming the starting address of the first analog channel to be converted and using the AUTOSCAN feature.

Two Analog Watchdogs are provided, on analog input channels 6 and 7, allowing a continuous hardware monitoring of these two inputs. An alarm Interrupt request will be generated whenever the converted value of either of these two analog inputs exceed one of the two programmed threshold values (Upper and Lower) for each channel. The comparison result is stored in a dedicated register.

Single, continuous, or externally triggered conversion modes are available, internal clock sample synchronization is also available through the 'On

Figure 11-1. A/D Block Diagram



chip Event" synchronization logic of a Multifunction Timer Unit,

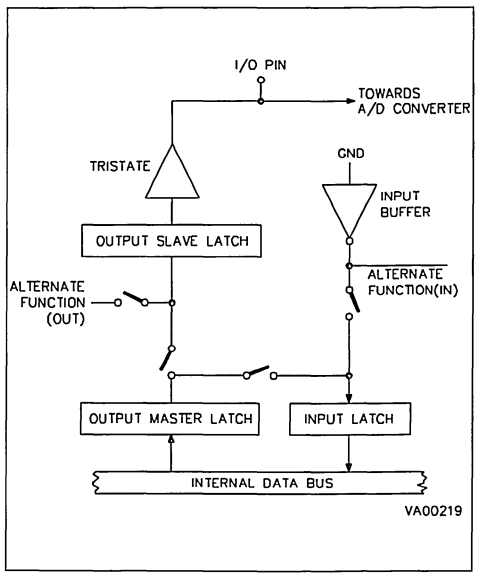
A Power-Down programmable bit allows to set the A/D converter to a minimum consumption idle status.

The ST9 A/D Interrupt Unit provides two maskable channels (Analog Watchdog and End of Conversion) with hardware fixed priority, and up to 7 programmable priority levels.

WARNING: A/D INPUT PIN DECLARATION

The I/O Port pins connected to the A/D must be configured as an A/D input by declaring it as an Alternate Function to prevent possible high power dissipation within the input buffer. The Alternate Function of ports connected to the A/D converter are modified as shown in figure 11-2.
An I/O Port PX (associated with the A/D converter) pin is configured as an A/D input for: (PXC2, PXC1, PXC0) = (1, 1, 1).

Figure 11-2. A/D Input Configuration Status



11.2 A/D OPERATION

11.2.1 Operational Modes

Two main operational modes are available: Continuous Mode and Single Mode. To enter one of these modes it is necessary to program the CONT bit of the Control Logic Register, the Continuous Mode is selected when CONT = "1", while CONT = "0" enables the Single Mode.

Both modes operate in the AUTOSCAN configuration, allowing a sequential conversion flow of the input channels. It is possible to choose by software the number of analog inputs to be converted by writing into the Control Register (SC2, SC1, SC0 bits) the number of the first channel to be converted. Subsequentially, after each conversion is completed, the channel number is automatically incremented, up to channel 7. For example, if (SC2, SC1, SC0) = 011, the conversion flow runs from channel 3 up to channel 7. If (SC2, SC1, SC0) = 111, only channel 7 is converted.

When the ST bit of the Control Logic Register is written to "1", the analog inputs are sequentially converted (from the first selected channel up to channel 7) and the results are stored in the relevant Data Registers.

In Single Mode (CONT = "0"), the ST bit is reset by hardware at the end of conversion of channel 7, an End of Conversion (ECV) interrupt request is issued, and the A/D waits for a new start event.

In Continuous Mode (CONT = "1"), a continuous conversion flow is entered by the start event. After the conversion of channel 7 ends, the conversion of channel 's' starts (where 's' is specified in the (SC2, SC1, SC0) bits), this will continue until the ST bit is reset by software. In all cases, an ECV interrupt is issued each time the channel 7 conversion ends.

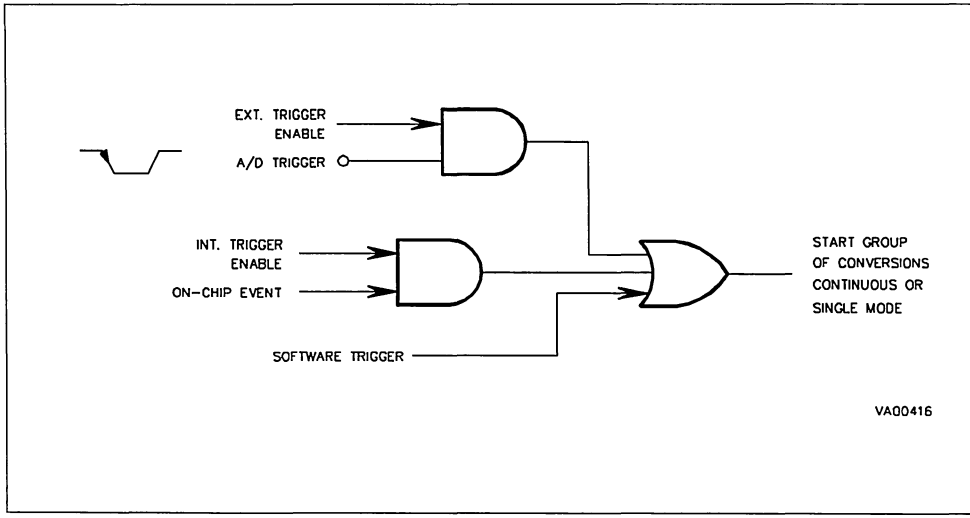
When channel 'i' is converted ('s' < 'i' < 7), the Data Register is reloaded with the new conversion result and the previous value is lost. The ECV interrupt routine can be used to save the current values before a new conversion sequence (so as to create signal sample tables within Register File or Memory).

11.2.2 Synchronisation

Conversion start synchronisation for all modes may be internal or external. The external (ADTRG, as Alternate Function of an I/O port) or the internal (INTRG, produced by an on-chip peripheral) can be used to synchronise the converion start with a trigger pulse. Both external and internal events can be seperately masked by the programming of the EXTG/INTG bits of the Control Logic Register. The events are internally OR'ed, thus avoiding potential hardware conflicts, however the correct procedure is to always enable only one alternate synchronisation input at any time.

The effect of the alternate synchronisation is to set the ST bit by hardware. This bit is reset, only in Single Mode, at the end of each group of conversions. In Continuous Mode all trigger pulses, following the first, are ignored.

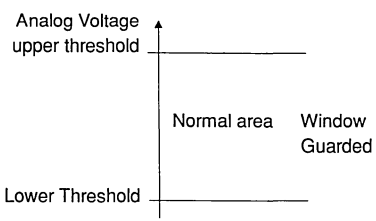
Figure 11-3. A/D Trigger Sources



The two synchronisation sources must have a clock cycle minimum length of 83ns (at INTCLK = 12MHz), and a period greater (in Single Mode) than the total time of a group of conversions (11.5µs x the number of channels scanned (at INTCLK = 12MHz)). If a trigger occurs when the ST bit is still "1" and conversion is still in progress, it is ignored.

11.2.3 Analog Watchdog

Two internal Analog Watchdogs are available, allowing great flexibility in automatic threshold monitoring in those applications experiencing a maximum range of fluctuations. Analog channels 6 and 7 define a voltage window for the allowed values of the converted analog input. The range of values of the external voltage applied to input 6 and 7 are accepted as normal whenever below the Upper threshold and above or equal to the Lower threshold.



When the external voltage is greater or equal to the upper, or is less than the lower programmed voltage limits, a maskable interrupt request (see AWD and AWDI in the Interrupt Control Register) is generated and the Compare Results Register is updated to inform which threshold (Upper or Lower) of which channel (6 or 7) has been exceeded. The 4 threshold voltages are user programmable in 4 dedicated registers (8 up to B) of the A/D register page, storing their 8 bit binary code. Only the 4 MSB of the Compare Results Register are used (the 4 LSB always return "1" if read), each bit for each threshold possible overflow or underflow status.

After an hardware reset, these bit values are "0". During the normal A/D operation, the CRR bits are set to "1" to flag an over-range and must be reset by the User in the watchdog interrupt routine, immediately before the `IRET` instruction, in order to allow further input monitoring.

11.2.4 Powerdown Mode

Before enabling any A/D conversion, it is mandatory to set the POW bit of the Control Logic Register to "1" at least 60µs before the first conversion start. This is in order to correctly bias the analog section of the converter, if this is not done, then functionality of the converter will be locked.

Setting POW to "0" is useful when the A/D is not required in order to reduce the total power consumption. This is the reset configuration, and is also entered automatically when the ST9 is in Halt Mode (following the execution of the `HALT` instruction).

Figure 11-4. Analog Watchdog Functional Diagram

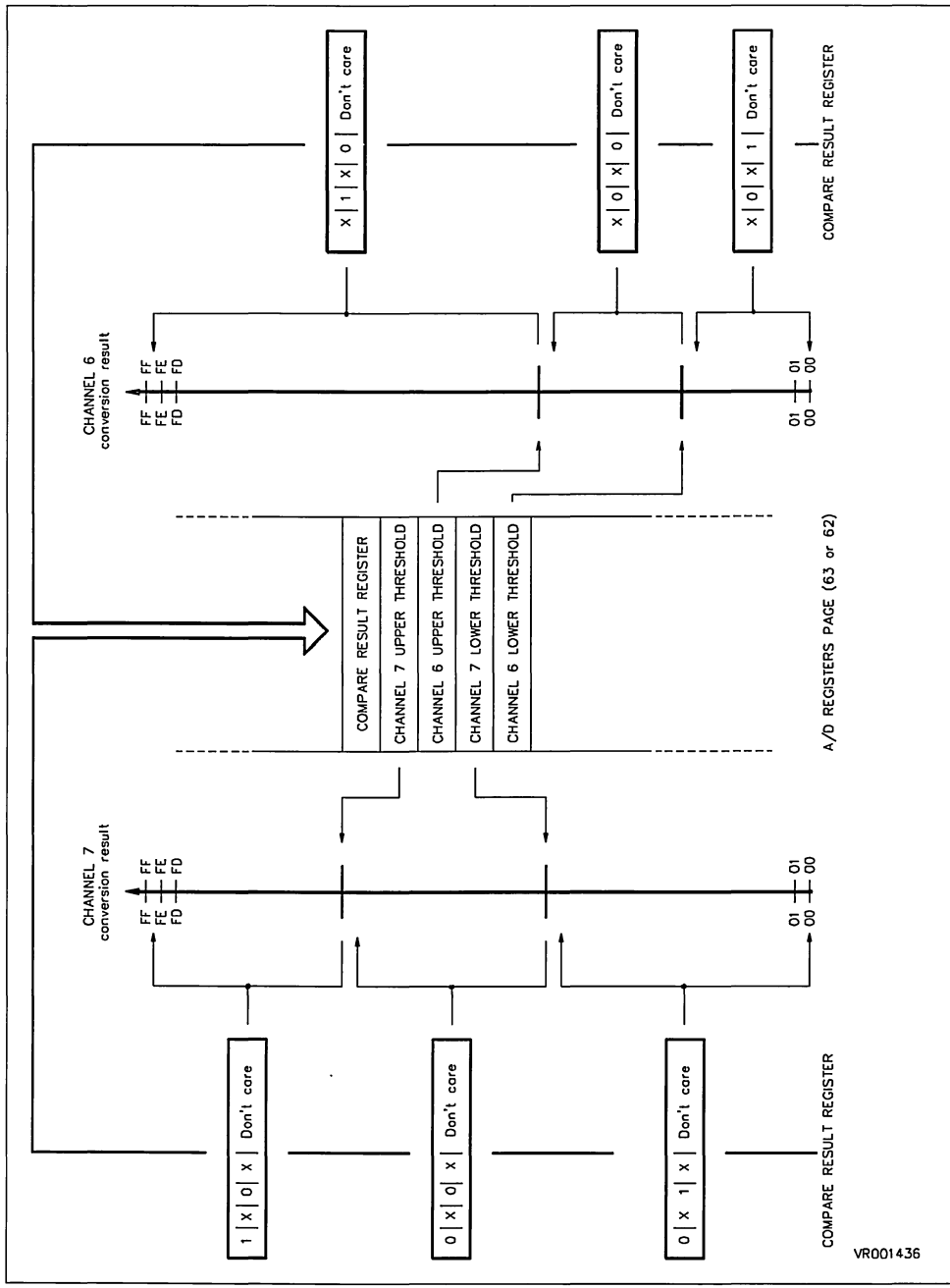
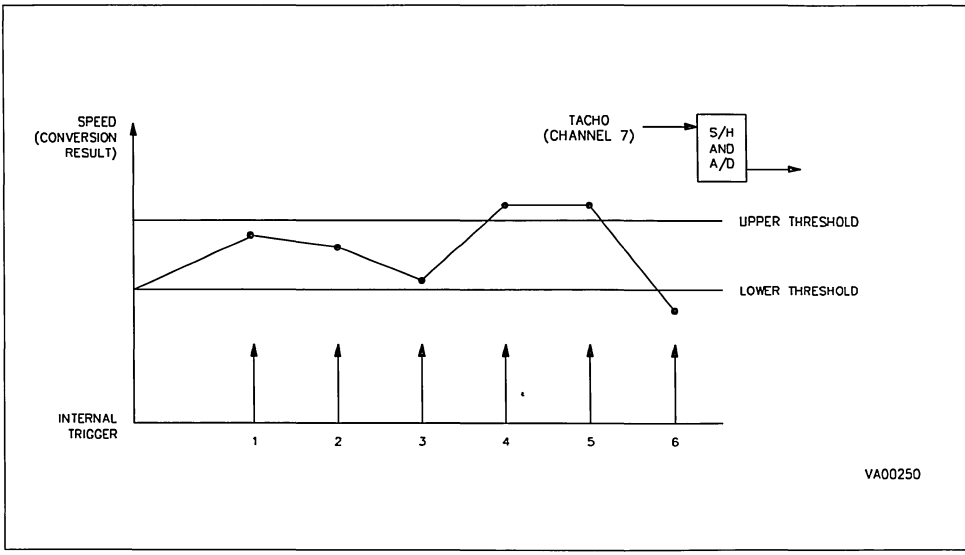


Figure 11-5. Analog Watchdog Used in Motor Speed Control



11.3 REGISTERS

11.3.1. Register Mapping

Up to two identical A/D converters can be implemented on the ST9 microcontroller. Each A/D peripheral has 16 registers, mapped in a ST9 register file page, addressed at page 63 (& 62 for the second A/D peripheral).

11.3.2 Interrupt Vector Register (IVR)

IVR R255 (FFh) Read/Write
Interrupt Vector Register

Reset Value: xxxx xx10 (x2h)

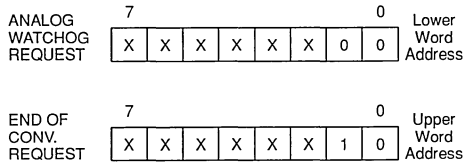
7							0
V7	V6	V5	V4	V3	V2	W1	D0

b7-b2 = **V7-V2:** *A/D Interrupt Vector.* This vector should be programmed by the User to point to the first memory location in the Interrupt Vector table containing the starting addresses of the A/D interrupt service routines.

1111	IVR : Interrupt Vector Register
1110	ICR : Interrupt Control Register
1101	CLR : Control Logic Register
1100	CRR : Compare Result Register
1011	Channel 7 Upper Threshold
1010	Channel 6 Upper Threshold
1001	Channel 7 Lower Threshold
1000	Channel 6 Lower Threshold
0111	Channel 7 Data Register
0110	Channel 6 Data Register
0101	Channel 5 Data Register
0100	Channel 4 Data Register
0011	Channel 3 Data Register
0010	Channel 2 Data Register
0001	Channel 1 Data Register
0000	Channel 0 Data Register

Page 63 (or 62)

b1 = **W1:** *Word Select.* This bit is set by hardware, according to the A/D interrupt source. It is set to "0" if the source is the Analog Watchdog, pointing to the lower word of the A/D interrupt service block (defined by V7-V2). It is set to "1" if the source is the End of Conversion interrupt, thus pointing to the upper word.



When 2 requests occur simultaneously the Analog watchdog request has priority over the End of Conversion request which is held pending, to be served after the current routine.

b0 = **D0:** This bit is fixed by hardware. It always returns the value "0" if read.

11.3.3 Interrupt Control Register (ICR)

This register contains the three priority level bits, the two sources flags, and their bit mask:

ICR R254 (FEh) Read/Write
Interrupt Control Register

Reset Value: 0000 1111 (0Fh)

7							0
ECV	AWD	ECI	AWDI	X	PL2	PL1	PL0

b7 = **ECV:** *End of Conversion.* ECV is automatically set by hardware after a group of conversions is completed.

b6 = **AWD:** *Analog Watchdog.* AWD is automatically hardware set whenever any of the two guarded analog inputs goes out of bounds. The threshold values are stored in registers F8h and FAh for channel 6, and in registers F9h and FBh for channel 7 respectively. The Compare Result Register (CRR) keeps track of the analog inputs exceeding the thresholds.

AWD and ECV must be reset by the user, before returning from the Interrupt service routine. Setting either of them by software will cause a software interrupt request to be generated.

b5 = **ECI:** *End of Conversion Interrupt Enable* This bit masks the End of Conversion interrupt request. A logical level "1" enables the request, a logical level "0" masks the request.

b4 = **AWDI:** *Analog Watchdog Interrupt Enable.* This bit masks the Analog Watchdog interrupt request.
A logical level "1" enables the request, a logical level "0" masks the request.

b3 = **D3:** *Undefined*

b2-b0 = **PL2, PL1, PL0:** *A/D Interrupt Priority Level.* With these three bits it is possible to select the Interrupt priority level of the A/D Converter.

11.3.4 Control Logic Register (CLR)

This register manages the A/D logic operations. Writing into this register will cause the current conversion to be aborted and the autoscan logic to be re-initialized to the starting configuration. CLR is programmable as following:

CLR R253 (FDh) Read/Write
Control Logic Register

Reset Value: 0000 0000 (00h)

7							0
SC2	SC1	SC0	EXTG	INTG	POW	CONT	ST

b7-b5 = **SC2, SC1, SC0:** *Start Conversion Address.* These 3 bits define the starting analog input in Autoscan mode. The first channel addressed by SC2-SC0 is converted, then the address is incremented for the successive conversion, until channel 7 (111) is converted. The (SC2, SC1, SC0) bits define the group of channels to be scanned. When setting SC2=1 SC1=1 SC0=1 only channel 7 is converted.

b4 = **EXTG:** *External Trigger.* When set to a logical "1", this bit allows to start a group of conversion synchronized on the following edge of the external signal applied on pin ADTRG (when enabled for Alternate Function)..

b3 = **INTG:** *Internal Trigger.* When set to a logical level "1", this bit enables the start of a group of conversion, synchronized with an internal signal (On chip Event signal) from another peripheral (e.g. a Multifunction Timer Unit).

Both External and Internal Trigger inputs are internally OR'ed, thus avoiding Hardware conflicts, however the correct procedure is to enable only one alternate synchronization input at time.

b2 = **POW:** *Power Up/Power Down* A logical "1" enables the A/D logic and analog circuitry.
A logical "0" disables all power consuming logic, thus allowing a low power idle status.

b1 = **CONT:** *Continuous/Single.* When this bit is set to "1" (Continuous Mode), the first group of conversions are started either by software (setting to "1"the ST bit) or by hardware (on an Internal or

external trigger, depending on the INTG and EXTG bits status), and a continuous conversion flow is then processed.

When this bit is set to "0" (single mode), only a single group of conversions (1 up to 8) are started whenever any External (or Internal) trigger occurs, or the ST bit is set to "1" by software.

The effect of the alternate synchronization is to hardware set the START/STOP bit which is hardware reset when in SINGLE mode, at the end of each group of conversions.

Requirements:

The External Synchronisation Input must receive a pulse (low level) wider than an INTCLK period (83ns) minimum and for both External and On chip Event synchronisation, a period greater than the time required for a group of conversion (number of channels times * 11µs).

b0 = **ST:** *Start/Stop* A logical "1" level enables the starting of a group of conversions; a logical level "0" stops the conversion. When the A/D is running in the Single Mode, this bit is hardware reset at the end of a group of conversions.

11.3.5 Compare Result Register (CRR)

The result of comparison between the current value of data registers 6 and 7 and the threshold registers is stored in this 4 bit register.

CRR **R252** (FCh) Read/Write
Compare Result Register

Reset Value: 0000 1111 (0Fh)

7							0
C7U	C6U	C7L	C6L	X	X	X	X

b7 = **C7U:** *Compare Reg 7 Upper threshold* Set to "1" when converted data is greater than or equal to the threshold value. Not affected otherwise

b6 = **C6U:** *Compare Reg 6 Upper threshold* Set to "1" when converted data is greater than or equal to the threshold value. Not affected otherwise

b5 = **C7L:** *Compare Reg 7 Lower threshold* Set to "1" when converted data is less than the threshold value. Not affected otherwise.

b4 = **C6L:** *Compare Reg 6 Lower threshold* Set to "1" when converted data is less than the threshold value. Not affected otherwise.

These bits should be Software reset at the end of the 'Out of Bounds' Interrupt routine.

b3-b0 = undefined, return "1" when read.

Note: any Software request reset in the ICR, will cause also these bits to be hardware forced to zero, to prevent possible overwriting, if an Interrupt request occurs between their Software reset and the Interrupt Request Software reset The correct procedure is to reset, before exiting the Interrupt routine, only the requests flags.

11.3.6 Upper Threshold Registers (UTiR)

The 2 upper threshold registers are used to store the 2 user programmable upper threshold voltages (i.e. their 8 bit binary code) to be compared with the present channel 6 or 7 conversion result. They fix the upper voltage window limit.

UT7R **R251** (FBh) Read/Write
Channel 7 Upper Threshold Register

Reset Value: Undefined

7							0
UT7.7	UT7.6	UT7.5	UT7.4	UT7.3	UT7.2	UT7.1	UT7.0

b7-b0 = **UT7.7-UT7.0** : *Channel 7 Upper Threshold*

UT6R **R250** (FAh) Read/Write
Channel 6 Upper Threshold Register

Reset Value: Undefined

7							0
UT6.7	UT6.6	UT6.5	UT6.4	UT6.3	UT6.2	UT6.1	UT6.0

b7-b0 = **UT6.7-UT6.0** : *Channel 6 Upper Threshold*

11.3.7 Lower Threshold Registers (LTiR)

The 2 lower threshold registers are used to store the 2 user programmable lower threshold voltages (i.e. their 8 bit binary code) to be compared with the present channel 6 or 7 conversion result. They fix the lower voltage window limit.

LT7R **R249** (F9h) Read/Write
Channel 7 Lower Threshold Register

Reset Value: Undefined

7							0
LT7.7	LT7.6	LT7.5	LT7.4	LT7.3	LT7.2	LT7.1	LT7.0

b7-b0 = **LT7.7-LT7.0** : *Channel 7 Lower Threshold*

LT6R **R248** (F8h) Read/Write
Channel 6 Lower Threshold Register

Reset Value: Undefined

7							0
LT6.7	LT6.6	LT6.5	LT6.4	LT6.3	LT6.2	LT6.1	LT6.0

b7-b0 = **LT6.7-LT6.0** : *Channel 6 Lower Threshold*

11.3.8 Data Registers (DiR)

The result of the conversions of the 8 available channels are loaded in the 8 DiR (channel0→D0R ....channel7→D7R); every Data Register is re-loaded with a new value at the end of the conversion of the correspondent analog input.

D7R R247 (F7h) Read/Write
Channel 7 Data Register

Reset Value: Undefined

7							0
D7.7	D7.6	D7.5	D7.4	D7.3	D7.2	D7.1	D7.0

b7-b0 = **D7.7-D7.0** : Channel 7 Data

D6R R246 (F6h) Read/Write
Channel 6 Data Register

Reset Value: Undefined

7							0
D6.7	D6.6	D6.5	D6.4	D6.3	D6.2	D6.1	D6.0

b7-b0 = **D6.7-D6.0** : Channel 6 Data

D5R R245 (F5h) Read/Write
Channel 5 Data Register

Reset Value: Undefined

7							0
D5.7	D5.6	D5.5	D5.4	D5.3	D5.2	D5.1	D5.0

b7-b0 = **D5.7-D5.0** : Channel 5 Data

D4R R244 (F4h) Read/Write
Channel 4 Data Register

Reset Value: Undefined

7							0
D4.7	D4.6	D4.5	D4.4	D4.3	D4.2	D4.1	D4.0

b7-b0 = **D4.7-D4.0** : Channel 4 Data

D3R R243 (F3h) Read/Write
Channel 3 Data Register

Reset Value: Undefined

7							0
D3.7	D3.6	D3.5	D3.4	D3.3	D3.2	D3.1	D3.0

b7-b0 = **D3.7-D3.0** : Channel 3 Data

D2R R242 (F2h) Read/Write
Channel 2 Data Register

Reset Value: Undefined

7							0
D2.7	D2.6	D2.5	D2.4	D2.3	D2.2	D2.1	D2.0

b7-b0 = **D2.7-D2.0** : Channel 2 Data

D1R R241 (F1h) Read/Write
Channel 1 Data Register

Reset Value: Undefined

7							0
D1.7	D1.6	D1.5	D1.4	D1.3	D1.2	D1.1	D1.0

b7-b0 = **D1.7-D1.0** : Channel 1 Data

D0R R240 (F0h) Read/Write
Channel 0 Data Register

Reset Value: Undefined

7							0
D0.7	D0.6	D0.5	D0.4	D0.3	D0.2	D0.1	D0.0

b7-b0 = **D0.7-D0.0** : Channel 0 Data

SERIAL COMMUNICATION INTERFACE

12.1 SCI FEATURES

The ST9 Serial Communications Interface (SCI) has the following features:

- Full duplex character-oriented asynchronous operation
- Synchronous serial port expansion capability
- Transmit, recieve, line status, and device address interrupt generation
- Integral baud rate generator capable of dividing the input clock by any value from 2 to 2^{16}-1 (16 bit word) and generating the internal 16X clock for asynchronous operation or 1X clock for synchronous operation
- Fully programmable serial-interface characteristics:
  - 5, 6, 7, or 8 bit word length
  - Even, odd, or no parity generation and detection
  - 1, 1-1/2, 2, 2-1/2 stop bit generation
  - False start bit detection
  - Complete status reporting capabilities
  - Line break generation and detection
- Programmable address indication bit (wake-up bit) and User invisible compare logic to support network communication of multiple microcomputers. Optional character search function.
- Internal diagnostic capabilities:
  - Local loopback for communications link fault isolation
  - Auto-echo for communications link fault isolation
- Separation interrupt/DMA channels for both transmit and receive

12.2 FUNCTIONAL DESCRIPTION

12.2.1 Serial Frame Format

Every character sent (or received) by the SCI has the following format:

START	LSB Data MSB	PARITY	ADDRESS/9TH	STOP Bits

This format is used by the SCI for the 3 modes:

- Asynchronous
- Synchronous
- Serial expansion mode

START: the start bit indicates the beginning of a data frame in the asynchronous mode. START bit is detected as a high to low transition

DATA: the DATA word is programmable to be 5 to 8 bits long for both synchronous and asynchronous modes

PARITY: The Parity Bit is optional, and can be used with any length of word. It is used for error checking and resets in a resultant state (odd or even) depending on number of "1"s in DATA.

ADDRESS/9TH: The Address/9th Bit is optional. It can be used with any word format. It is used in both synchronous or asynchronous mode to indicate that the data is an address (bit = "1"). The ADDRESS/9TH bit is useful when several microcontrollers are exchanging data on the same serial bus. Individual microcontrollers can stay idle on the serial bus, waiting for a transmitted address. When a microcontroller recognizes its own address, it can begin Data Reception, likewise, on the transmit side, the microcontroller can transmit another address to begin communication with a different microcontroller.

The ADDRESS/9TH bit can be used as an additional data bit or to mark control words (9th bit).

STOP: Indicates the end of a data frame for both asynchronous and synchronous modes. The stop bit is programmed to be 1, 1.5, 2, or 2.5 bits long. It returns the SCI to the quiescent marking state (i.e., a constant high-state condition which lasts until a new start bit indicates an incoming word).

12.2.2 Architecture

12.2.2.1 EXTERNAL PINS

SOUT: Serial Data Output. This signal is the serial data output from the SCI transmitter

SIN: Serial Data Input. This signal is the serial data input to the SCI receiver

TXCLK: External Transmitter Clock Input. This pin inputs the clock driving the SCI transmitter; TXCLK frequency is greater than or equal to 16 times the transmitter data rate (depending on the selection of X16 or X1 clock operating mode). TXCLK pin is optional.

RXCLK: External Receiver Clock Input. This input can be the clock sent to the SCI receiver. INTCLK is normally the SCI baud rate generator clock, but this input RXCLK can also be the clock source for the SCI baud rate generator. A 50/50 duty cycle is not required for this input. However, the shorter pulse must last more than two INTCLK periods. RXCLK pin is optional.

CLKOUT: Clock Output. This pin outputs either the data clock from the transmitter to an external shift register in the serial expansion mode or the clock output from the Baud rate generator. In serial expansion mode it will clock only the data portion of the frame. The data is valid on the rising edge of the clock. The CLKOUT idle state is low.

12.2.2.2 FUNCTIONAL ARCHITECTURE

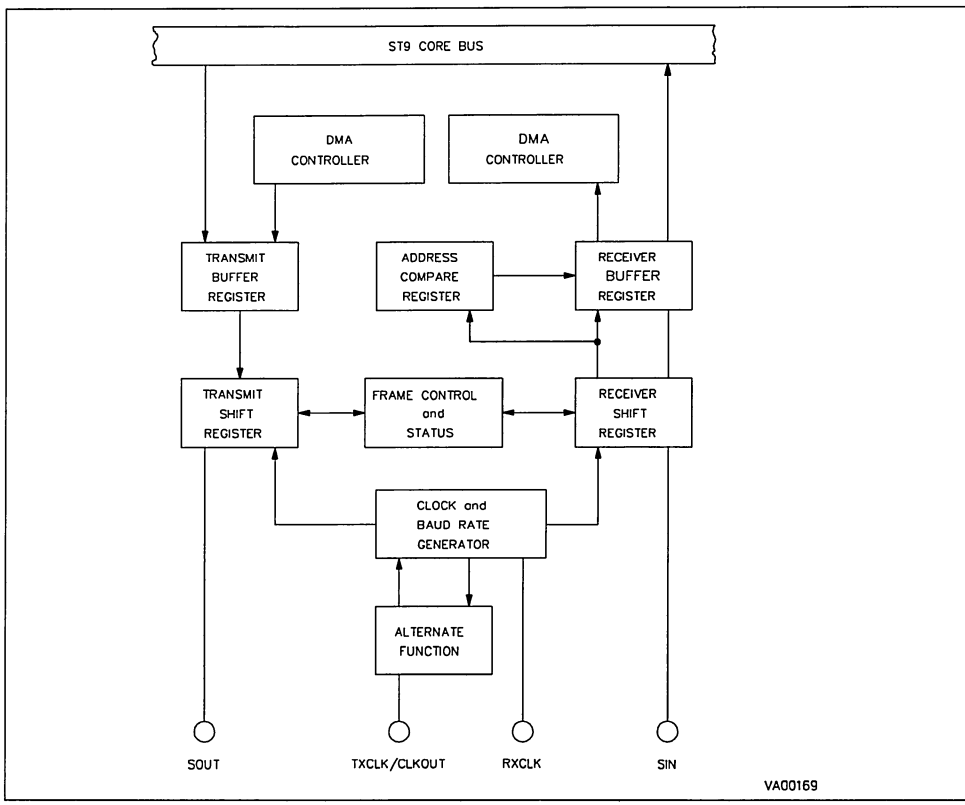
Reader should refer to figure 12.1 when reading the following information.

Transmitter Shift Register. The Transmitter Shift Register converts parallel data (coming from TRANSMIT BUFFER REGISTER) into serial format for transmission

Transmitter Buffer Register. The Transmitter Buffer Register is loaded by the ST9 when a data has to be transmitted. The SCI will transfer the data into the Shift Register as it becomes available. At the transfer, the transmitter buffer register interrupt will be updated.
If the selected word format is less than 8 bits, the unused most significant bits are "don't care".

Receiver Shift Register. The Receiver Shift Register converts incoming serial data into parallel data for reception

Figure 12-1. SCI Block Diagram



Receiver Buffer Register. The Receiver Buffer Register stores the data portion of the received word. The data will be transferred from the "RECEIVER SHIFT REGISTER" into this register at the end of the word. All Receiver interrupt conditions will be updated at the time of transfer. If the selected character format is less than 8 bits, unused most significant bits will have the value "1".

Frame Control And Status. This block contains the character configuration, that means it defines the Data length, the stop bits, the source for the transmitter/receiver clock.

Clock And Baud Rate Generator. The Baud Rate Generator contains a programmable divide by "N" counter which can be used to generate the clocks for the transmitter and/or receiver. The minimum baud rate is 2 and the maximum is 2^{16}-1. The baud rate generator can use INTCLK or the Receiver clock input RXCLK.

Data is latched upon the rising edge of the clock.

Address/Data Compare Register. With the 9th bit address mode, the received address will be compared to the value of this register. If true, the Receive Address Pending bit is set and all received data will be transferred to the RECEIVER BUFFER REGISTER.

DMA Controller. This contains the Transaction Counters and Source Addresses for Transmission and Reception, the interrupt vectors, the interrupt masks and the interrupts status and priority.

Remark: If properly initialized, the DMA controller starts a data transfer after and only if the running program has loaded the Transmitter Buffer Register with a value. In order to execute properly a DMA transmission, the End Of Block interrupt routine must include the following actions:

- Load the Transmitter Buffer Register (TXBR) with the first byte to transmit.
- Restore the DMA counter (TDCPR)
- Restore the DMA pointer (TDAPR)
- Reset the transmitter end of block bit TXEOB (IMR.5)
- Reset the transmitter holding empty bit TXHEM (ISR.1)
- Enable DMA

12.2.2.3 CLOCKS AND SERIAL TRANSMISSION RATES

The maximum data transfer rate is in synchronous mode (1x mode):

- Maximum bit rate = INTCLK/8 = 12MHz/8 = 1.5 Mbit/s
- Maximum byte rate= 1.5 Mbit/10 = 150 kbytes/s

(one byte = 8 bits of data + 1 stop bit + 1 start bit = 10 bits)

Table 12-1. SCI Timings

	Min.	Max.	Conditions
INTCLK Frequency	0	12MHz	
RXCLK Frequency	0	INTCLK/8	1x mode
	0	INTCLK/4	16x mode
	0	INTCLK/2	16x mode internal clock
RXCLK Pulse	$4xT_{INTCLK}$		1x mode
	$2xT_{INTCLK}$		16x mode
TXCLK Frequency		INTCLK/8	1x mode
		INTCLK/4	16x mode
		INTCLK/2	16x mode internal clock
TXCLK Pulse	$4xT_{INTCLK}$		1x mode
	$2xT_{INTCLK}$		16x mode
Data set-up time before rising edge of RXCLK	$T_{INTCLK}/2$		
Data hold time (after falling edge of TXCLK)		$5xT_{INTCLK}/2$	1x mode

12.2.3 SCI Modes: asynchronous, synchronous, serial expansion mode

SCI can run in three modes:

- Asynchronous mode
- Synchronous mode
- Serial expansion mode

Each of these three modes output data with the same serial frame format (as described in 12.2.1). The differences are coming from the clock rate (x1, x16) and the sampling clock (for the serial expansion mode).

ASYNCHRONOUS MODE

In this mode, data and clock can be asynchronous (usually emitter and receiver have their own clock to sample received data), each data is sampled 16 times per clock period.

SYNCHRONOUS MODE

In this mode, data and clock are synchronous, each data is sampled once per clock period.

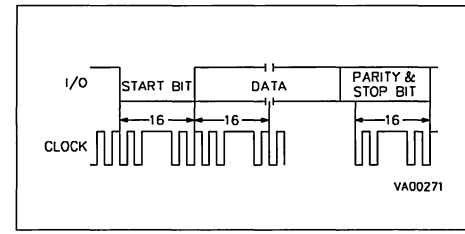
SERIAL EXPANSION MODE

This mode is used to access an external synchronous peripheral.

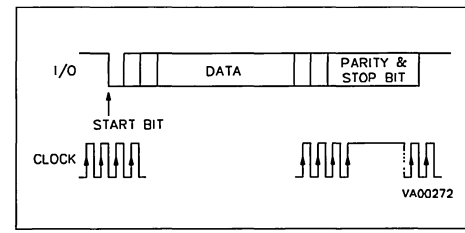
The transmitter will provide the clock waveform only during the transmitted data through CLKOUT pin. The data is latched on the rising edge of this clock.

Whenever the SCI has to receive data in the serial port expansion mode, this clock waveform must be supplied synchronously with the data to the ST9. The SCI will latch the incoming data on the rising edge of the receiver I/O expansion clock. The clock is supplied on RXCLK pin.

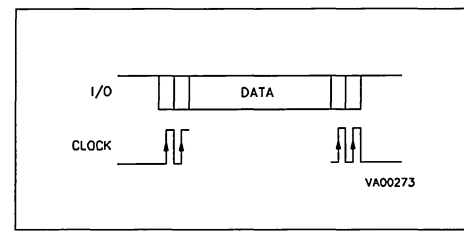
ASYNCHRONOUS MODE



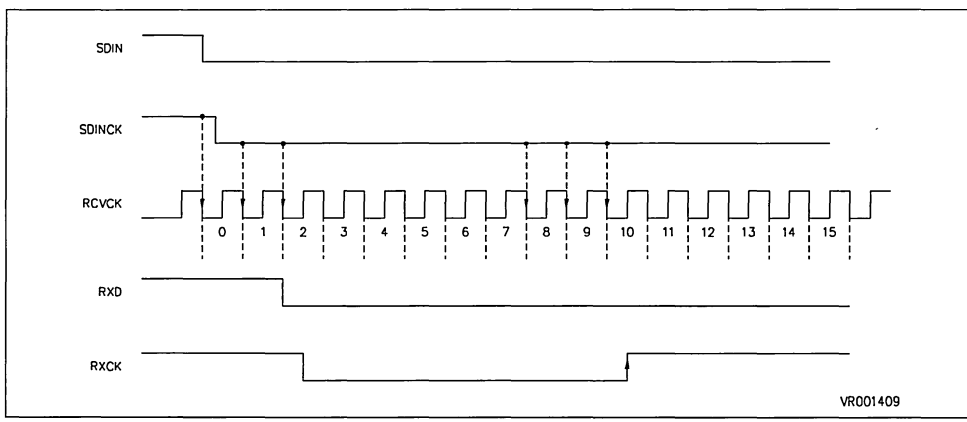
SYNCHRONOUS MODE



SERIAL EXPANSION MODE



SAMPLING TIMES IN ASYNCHRONOUS MODE



12.3 CONTROL REGISTERS

The SCI is controlled by the following registers

Address	Register
0000 (0h)	Receiver DMA Transaction Counter Pointer Register
0001 (1h)	Receiver DMA Source Address Pointer Register
0010 (2h)	Transmitter DMA Transaction Counter Pointer Register
0011 (3h)	Transmitter DMA Destination Address Pointer Register
0100 (4h)	Interrupt Vector Register
0101 (5h)	Address Compare Register
0110 (6h)	Interrupt Mask Register
0111 (7h)	Interrupt Status Register
1000 (8h)	Receive Buffer Register same Address as Transmitter Buffer Register (Read Only)
1000 (8h)	Transmitter Buffer Register same Address as Receive Buffer Register (Write only)
1001 (9h)	Interrupt/DMA Priority Register
1010 (Ah)	Character Configuration Register
1011 (Bh)	Clock Configuration Register
1100 (Ch)	Baud Rate Generator Register
1101 (Dh)	

The relative pages of the SCI into a ST9 are:

- SCI number 1: page 24 (18h)
- SCI number 2: page 25 (19h)
- SCI number 3: page 26 (1Ah)
- SCI number 4: page 27 (1Bh)

RDCPR R240 (F0h) Read/Write
Receiver DMA Transaction Counter Pointer Register

Reset value: undefined

7							0
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RR/M

b7-b1 = **RC7-RC1:** *Receive DMA Counter Pointer.* RDCPR contains the address of the pointer (in the Register File) of the DMA receiver transaction counter.

b0 = **RR/M:** *Receiver Register File/Memory Selector.* If this bit = "1" the Register File will be selected as Destination, if this bit = "0" the Memory space will be selected.

RDAPR R241 (F1h) Read/Write
Receiver DMA Source Address Pointer

Reset value: undefined

7							0
RA7	RA6	RA5	RA4	RA3	RA2	RA1	RD/P

b7-b1 = **RA7-RA1:** *Receive DMA Address Pointer.* RDAPR contains the address of the pointer (in the Register File) of the receiver DMA data source.

b0 = **RD/P:** *Receive DMA Data/Program Memory Selector.* If memory (RR/M = "0") has been selected for DMA transfers, when this bit = "1" receiver DMA transfers will go to Data Memory. If this bit = "0" receiver DMA transfers will go to Program Memory.

TDCPR R242 (F2h) Read/Write
Transmitter DMA Transaction Counter Pointer

Reset value: undefined

7							0
TC7	TC6	TC5	TC4	TC3	TC2	TC1	TR/M

b7-b1 = **TC7-TC1:** *Transmitter DMA Counter Pointer.* TDCPR contains the address of the pointer (in the Register File) of the DMA transmitter transaction counter.

b0 = **TR/M:** *Transmitter Register File/Memory Selector.* If this bit = "1" the Register File will be selected as Source, if this bit = "0" the Memory space will be selected.

TDAPR R243 (F3h) Read/Write
Transmitter DMA Destination Address Pointer Register

Reset value: undefined

7							0
TA7	TA6	TA5	TA4	TA3	TA2	TA1	TD/P

b7-b1 = **TA7-TA1:** *Transmitter DMA Address Pointer.* TDAPR contains the address of the pointer (in the Register File) of the transmitter DMA data source.

b0 = **TD/P:** *Transmitter DMA Data/Program Memory Selector.* If memory (TR/M = "0") has been selected for DMA transfers, when this bit = "1" transmitter DMA transfers come from Data Memory. If this bit = "0" transmitter DMA transfers come from Program Memory

IVR R244 (F4h) Read/Write and Read only
Interrupt Vector Register

Reset value: undefined

7							0
V7	V6	V5	V4	V3	EV2	EV1	0

b7-b3 = **V7-V3:** *SCI Interrupt Vector Base Address.* User programmable interrupt vector bits for transmitter and receiver

b2-b1 = **EV2-EV1:** *Encoded Interrupt Source (Read only).* EV2 and EV1 are set by hardware according to the interrupt source.

b0 = **D0:** This bit is fixed by hardware. It always returns the value "0" when read.

EV2	EV1	Encoded Status Information
0	0	Receiver error
0	1	Break detect or address word match
1	0	Receiver data ready/receiver End of Block
1	1	Transmitter buffer or shift register empty/transmitter End of Block

ACR R245 (F5h)) Read/Write
Address/Data Compare Register

Reset value: undefined

7							0
AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0

b7-b0 = **AC7-AC0:** *Address/Compare Character.* With either 9th bit address mode, address after break mode, or character search, the received address will be compared to the value stored in this register. When a valid address matches this register content, the Receive Address Pending bit is set. After the RXAP bit is set in an addressed mode all received data words will be transferred to the Receiver Buffer Register.

IMR R246 (F6h) Read/Write
Interrupt Mask Register

Reset value: 0xx0 0000b

7							0
HSN	RXEOB	TXEOB	RXE	RXA	RXB	RXDI	TXDI

b7 = **HSN:** *Holding or shift register empty interrupt.* This bit selects the source of interrupt/DMA as the transmitter register empty event. If this bit is set to "1", a holding register empty will generate a transmitter register empty interrupt.
If this bit has a "0" value, a shift register empty will generate a transmitter register empty interrupt.

b6 = **RXEOB:** *Received End of Block.* This bit is set after a receiver DMA cycle to mark the end of a block of data. The last DMA data word will cause a DMA cycle followed by a receiver data ready interrupt. This sequence will signal to the ST9 core to reinitialize the receiver DMA block counter. RXEOB should be reset by software in order to avoid undesired interrupt routines, especially in initialisation routine (after reset) and after entering the End Of Block interrupt routine.
Writing "0" in this bit will cancel the interrupt request.

NOTE: RXEOB can only be written with a "0" (RXEOB = set only by the ST9 core)

b5 = **TXEOB:** *Transmitter End of Block.* This bit is set in a transmitter DMA cycle to mark the end of a block of data. The last DMA data word will cause a DMA cycle followed by a transmitter interrupt. This sequence will signal to the ST9 core to reinitialize the transmitter DMA block counter. TXEOB should be reset by software in order to avoid undesired interrupt routines, especially in initialisation routine(after reset) and after entering the End Of Block interrrupt routine.
Writing "0" in this bit will cancel the interrupt request.

NOTE: TXEOB can only be written with a 0 (TXEOB is set only by the ST9 core)

b4 = **RXE:** *Receiver Error Mask.* When this bit is set to "0", the receiver error bits: Overrun Error (OE), Parity Error (PE), and Framing Error (FE), cannot generate an interrupt.

b3 = **RXA:** *Receiver Address Mask.* When this bit is set to "0", the Receiver Address Pending (RXAP) bit cannot generate an interrupt.

b2 = **RXB:** *Receiver Break Mask.* When this bit is set to "0", the Receiver Break Pending (RBP) bit cannot generate an interrupt.

b1 = **RXDI:** *Receiver Data Interrupt Mask.* When this bit is set to "0", the Receiver Data Pending (RDP) bit and the Receiver End of Block (RXEOB) bit cannot generate an interrupt. RXDI has no effect on DMA transfers.

b0 = **TXDI:** *Transmitter Data Interrupt Mask.* When this bit is set to "0", neither the Transmitter Holding or Shift Register Empty (TXHEM) bit or the Transmitter End of Block (TXEOB) bit can generate an interrupt. TXDI has no effect on DMA transfers.

ISR R247 (F7h) Read/Write
Interrupt Status Register

Reset value: xxxx xxxb (XXh)

7							0
OE	FE	PE	RXAP	RXBP	RXDP	TXHEM	TXSEM

b7 = **OE:** *Overrun Error Pending.* This bit is set to a logic "1" if the data in the Receiver Buffer Register was not read by the CPU before the next character was transferred into the Receiver Buffer Register (the previous data is lost). It is cleared by writing a zero into OE.

b6 = **FE:** *Framing Error Pending bit.* This bit is set to a logic "1" if the received data word did not have a valid stop bit. It is cleared by writing a zero to the bit. In the case where a framing error occurs when the SCI is programmed in an address mode, and is monitoring for an address, this interrupt is asserted and the corrupted data element is transferred to the Receiver Buffer Register.

b5 = **PE:** *Parity Error Pending.* This bit is set to a logic "1" if the received word did not have the correct even or odd parity bit. It is cleared by writing a zero into PE.

b4 = **RXAP:** *Receiver Address Pending.* RXAP is set to "1" after an interrupt acknowledged in the address mode. The source of this interrupt is given by the couple of bits (AMEN, AM) as detailed in the "Interrupt/DMA Priority Register" description. RXAP is cleared by software.

b3 = **RXBP:** *Receiver Break Pending bit.* This bit is set to a logic "1" if the received data input is held low for the full word transmission time (start bit, data bits, parity bit, stop bit). It is cleared by writing a zero into RXBP.

b2 = **RXDP:** *Receiver Data Pending bit.* This bit is set to a logic "1" when data is loaded into the Receiver Holding Register. It is cleared by writing a zero into RXDP.

b1 = **TXHEM:** *Transmitter buffer register Empty.* This bit is set to a logic "1" if the Holding Register is empty. It is cleared by writing a zero into TXHEM.

b0 = **TXSEM:** *Transmitter Shift Register Empty.* This bit is set to a logic "1" if the Shift Register has completed the transmission of the available data. It is cleared by writing a "0" into TXSEM.

NOTE:
- The Interrupt Status Register bits can be reset by writing a "0" but it is not possible to write a "1" into any bit in this register. It is mandatory to clear the interrupt source by writing a "0" in the pending bit when executing the interrupt service routine.
- When servicing an interrupt routine, the User should reset ONLY the pending bit relative to the serviced interrupt routine (and not reset the other pending bits)

RXBR R248 (F8h) Read only
Receive Buffer Register

Reset value: undefined

7							0
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0

b7-b0 = **RD7-RD0:** *Received Data.* This register stores the data portion of the received word. The data will be transferred from the Receiver Shift Register into the Receiver Buffer Register at the end of the word. All receiver interrupt conditions will be updated at the time of transfer. If the selected character format is less than 8 bits, unused most significant bits will forced to "1".

TXBR R248 (F8h) Write only
Transmitter Buffer Register

Reset value: undefined

7							0
TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

b7-b0 = **TD7-TD0:** *Transmit Data.* The ST9 core will load the data for transmission into this register. The SCI will transfer the data from the buffer into the Shift Register when available. At the transfer, the Transmitter Buffer Register interrupt will be updated. If the selected word format is less than 8 bits, the unused most significant bits are not significant.

IDPR R249 (F9h) Read/Write
Interrupt/DMA Priority Register

Reset value: undefined

7							0
AMEN	SB	SA	RXD	TXD	PRL2	PRL1	PRL0

b7 = **AMEN:** *Address Mode Enable.* This bit, with AM, decodes the desired addressing/ninth data bit/character match operation.

In an addressed mode the SCI will monitor the input serial data until its address is detected.

AMEN	AM	
0	0	Address interrupt if ninth data bit = 1
0	1	Address interrupt if character match
1	0	Address interrupt if character match and ninth data bit = 1
1	1	Address interrupt if character match with word immediately following break

Upon reception of address, the RXAP bit (in the Interrupt Status Register) is set and an interrupt cycle can begin. The address character will not be transferred into the Receiver Buffer Register but, all data following the matched SCI address and preceeding the next address word will be transferred to the Receiver Buffer Register and the proper interrupts updated. If the address does not match, all data following this unmatched address will not be transferred to the Receiver Buffer Register.

In any of the cases the RXAP bit must be reset by software before the next word is transferred into the Buffer Register.

b6 = **SB:** *Set Break.* If this bit is set, a break will be transmitted following the transmission of all data in the Transmitter Shift Register and the Buffer Register. The break will be a "0" value on the transmitter data output for at least one complete word format. If software does not reset SB before the minimum break length has finished, the break condition will continue until software resets SB. The SCI terminates the break condition with a "1" on the transmitter data output for one transmission clock period.

b5 = **SA:** *Set Address.* If an address/9th data bit mode is selected, SA value will be loaded for transmission. Setting this bit indicates an address word. SA will be cleared by hardware after it is loaded into the Shift Register. Proper procedure would be, when the Transmitter Buffer Register is empty, to load the value of SA and then load the data into the Transmitter Buffer Register.

b4 = **RXD:** *Receiver DMA Mask.* If this bit is "0", no receiver DMA request will be generated, and the RXDP bit in the Interrupt Status Register can request an interrupt. If RXD is set to "1", the RXDP bit can request a DMA transfer. This bit is reset by hardware when the transaction counter value decrements to zero. At that time a receiver "end of block" interrupt can occur.

b3 = **TXD:** *Transmitter DMA Mask.* If this bit is "0" no transmitter DMA request will be generated and the TXHEM (or TXSEM) bit in the Interrupt Status Register can request an interrupt. If TXD is set, the TXHEM (or TXSEM) bit can request a DMA transfer. This bit is reset by hardware when the transaction counter value decrements to zero. At that time a transmitter End Of Block interrupt can occur.

b2-b0 = **PRL2, PRL2, PRL0:** *SCI Interrupt/DMA Priority bits.* The priority for the SCI is encoded with (PRL2,PRL1,PRL0). A priority value of "0" has the highest priority, a value of "7" has no priority.

When user has defined a priority level for the SCI, priorities inside the SCI are hardware defined. These SCI internal priorities are:

receiver DMA request	higher priority
transmitter DMA request	
receiver interrupt	
transmitter interrupt	lower priority

CHCR R250 (FAh) Read/Write
Character Configuration Register

Reset value: undefined

7							0
AM	EP	PEN	AB	SB2	SB1	WL1	WL0

b7 = **AM:** *Address Mode.* decodes the desired addressing/ninth data bit/character match operation in conjunction with AMEN (bit 7 of INT/DMA priority register). There are four basic operating modes:

AMEN	AM	
0	0	Address interrupt if 9th data bit = 1
0	1	Address interrupt if character match
1	0	Address interrupt if character match and 9th data bit = 1
1	1	Address interrupt if character match with word immediately following break

b6 = **EP:** *Even Parity.* When parity is enabled, this bit selects between even or odd parity. If this bit is equal to 0, odd parity will be selected. If this bit is equal to 1, even parity will be selected.

b5 = **PEN:** *Parity Enable.* When this bit is equal to 1, a parity bit is generated (transmit data) or checked (received data) between the last word bit and the stop bits. If the address/9th bit is enabled, the parity bit will precede the address/9th bit (The parity bit is used to produce an even or odd number of 1's when the parity bit and all data bits are summed. The 9th bit is never included in the parity calculation).

b4 = **AB:** *Address/9th Bit.* If this bit equals "1" the transmit and receive character format will include a bit between the parity bit and the first stop bit. This bit can be used to address the SCI or as a ninth data bit.

b3-b2 = **SBx:** *Stop Bits.* This bit field specifies the number of stop bits to be included in the data format

SB2	SB1	Number of stop bits	
		in 16X mode	in 1X mode
0	0	1	1
0	1	1.5	2
1	0	2	2
1	1	2.5	3

b1-b0 = **WL1, WL0:** These two bits specify the number of data bits in each transmitted or received character. The following table shows the coding of WL.

WL1	WL0	Data Length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

When AMEN = "0" and AM = "1", a useful character search function is performed. This allows the SCI to generate an interrupt whenever a specific character is encountered (e.g. Carriage Return). Figure 12.2 shows the use of the SCI addressing modes.

CCR R251 (FBh) Read/Write
Clock Configuration Register

Reset value: 0000 0000 (00h)

7							0
XTCLK	OCLK	XRX	XBRG	CD	AEN	LBEN	STPEN

b7 = **XTCLK:**

b6 = **OCLK:** These two bits select the source for the transmitter clock. The following table shows the coding of XTCLK and OCLK.

XTCLK	OCLK	Pin Function
0	0	Pin is used as a general I/O
0	1	Pin = TXCLK (used as an input)
1	0	Pin = CLKOUT (outputs the Baud Rate Generator clock)
1	1	Pin = CLKOUT (outputs the serial exp. mode clock)

b5 = **XRX:** *External Receiver Clock Source.* If this bit is "1", the receiver will use the external receiver clock pin for its clock source. The external clock must be equal to 16 times the data rate or equal to the data rate depending on the bit CD.

b4 = **XBRG:** *Baud Rate Generator Clock Source.* If this bit is "1", the baud rate generator will use the external receiver clock pin for its clock source. If this bit is "0", the baud rate generator will use the ST9 system clock (INTCLK).

b3 = **CD:** *Clock Divisor.* If CD = "1", both the receiver and the transmitter will be in 1X clock mode. In 1X clock mode, the transmitter will transmit data at one data bit per clock period. If this bit is "0", both the receiver and the transmitter will be in 16X mode. In 16X mode each data bit period will be 16 clock periods long.

The CD value will determine the synchronous/asynchronous SCI configuration mode.

b2 = **AEN:** *Auto Echo Enable.* If AEN = "1", the SCI is in auto echo mode. In this mode the SCI transmitter is disconnected from the transmitter data-out pin (SOUT). The transmitter data-out pin (SOUT) is driven directly by the receiver data-in pin (SIN). The receiver remains connected to the receiver data-in pin (SIN) and is operational, unless loopback mode is also selected.

b1 = **LBEN:** *Loopback Enable.* If this bit is set to "1", the loopback mode is enabled. In this mode the transmitter output is set to "1", the receiver input is disconnected, and the output of the Transmitter Shift Register is looped back into the Receiver Shift Register input. All interrupt sources for both the transmitter and the receiver are operational.

b0 = **STPEN:** *Stick Parity Enable.* If this bit is set to "1", the transmitter and the receiver will use the opposite parity type selected by the even parity bit (EP).

EP	SPEN	Parity (transmitter & receiver)
0 (odd)	0	Odd
1 (even)	0	Even
0 (odd)	1	Even
1 (even)	1	Odd

Figure 12-2. SCI Addressing Modes

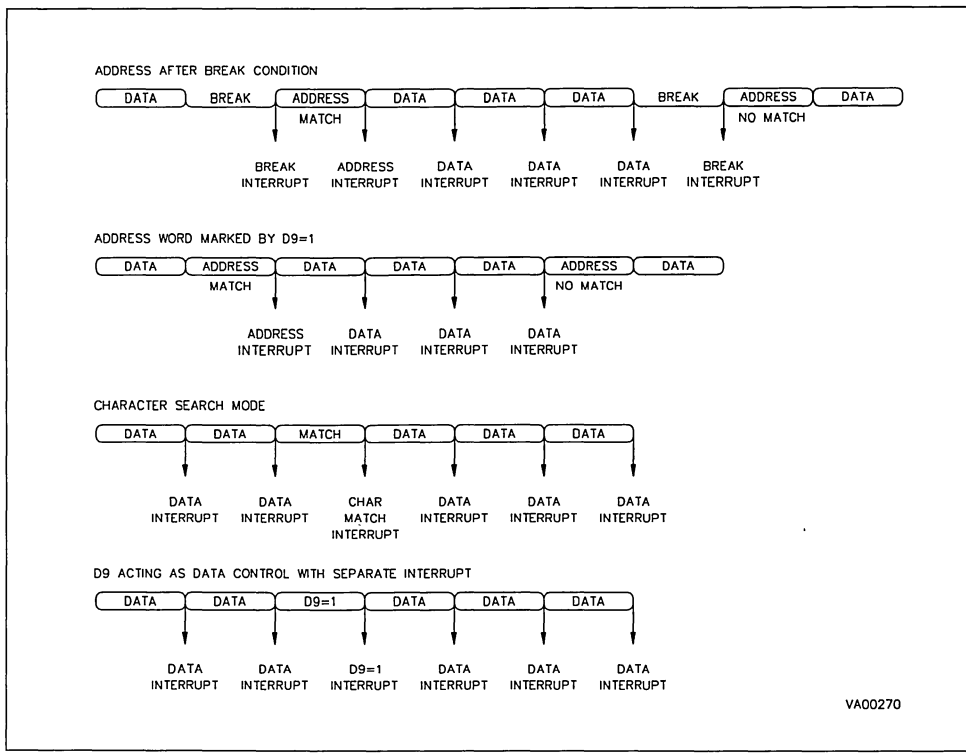


Figure 12-3. SCI Functional Scheme

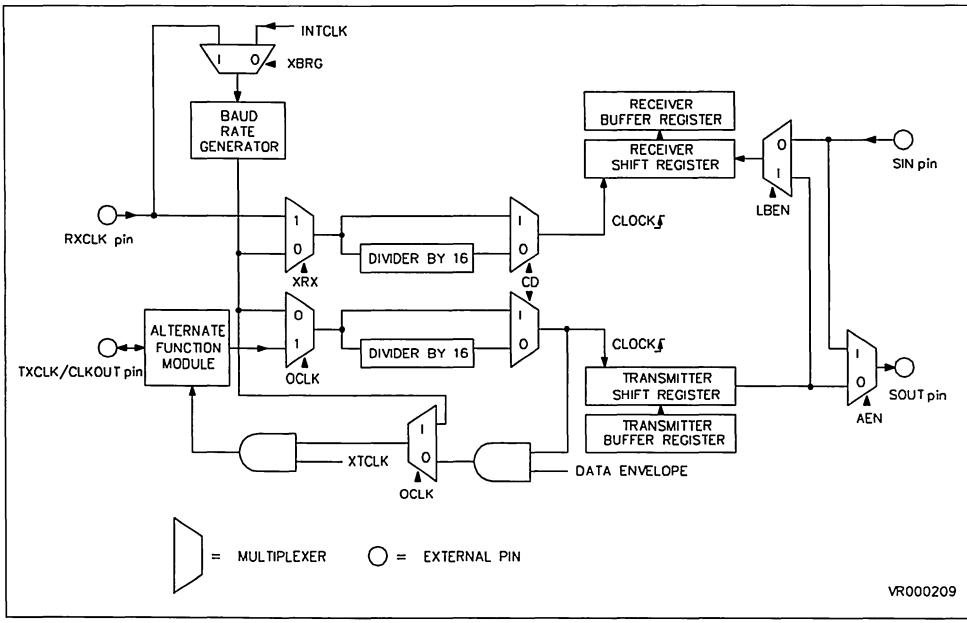


Figure 12-4a. Auto Echo Configuration

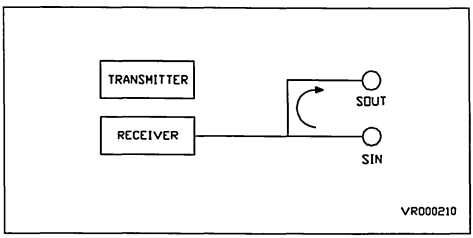


Figure 12-4b. Loop Back Configuration

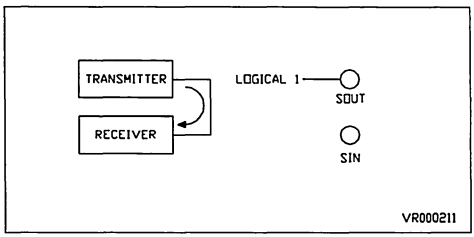
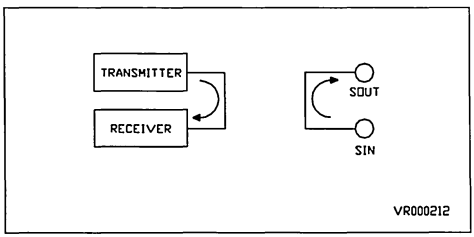


Figure 12-4c. Auto Echo and Loop Back Configuration



BRGHR R252 (FCh) Read/Write
Baud Rate Generator Register, High byte.

Reset value: undefined

15							8
BG15	BG14	BG13	BG12	BG11	BG10	BG9	BG8

BRGLR R253 (FDh) Read/Write
Baud Rate Generator Register, Low byte.

Reset value: undefined

7							0
BG7	BG6	BG5	BG4	BG3	BG2	BG1	BG0

b15-B0: *The Baud Rate generator* contains a programmable divide by "N" counter which can be used to generate the clocks for the transmitter and/or receiver. This counter divides the clock input by the value in the Baud Rate Generator Register. The minimum baud rate divisor is 2 and the maximum divisor is 2^{16}-1. After initialization of the baud rate generator, the divisor value is immediately loaded into the counter. This prevents potentially long random counts on the initial load.

Baud Rate generator frequency = Input Clock frequency/Divisor

The baud rate generator clock provides a 16X or 1X clock for the receiver and the transmitter. An additional divide by 16 may be appropriate to compute the SCI data rate if in this normal operating mode.

The Baud Rate generator can use INTCLK clock for the input clock source. An alternate source is the receiver clock input pin.

The output of the baud generator has an exact 50% duty cycle. The output can provide either the 16X clock for asynchronous operation or a 1X clock for synchronous and serial port expansion modes.

NOTES:

1) Writing to a Baud Rate Generator Register immediately disables and resets both the SCI baud rate generator, the transmitter and receiver circuitry. After writing to the remaining Baud Rate Generator Register, the transmitter and receiver circuits are enabled. The Baud Rate generator will load the new value and start counting.

To initialize the SCI, user should first initialize one Baud Rate Generator Divisor Register. This will reset all SCI circuitry. Initialize all other SCI registers for the desired operating mode. To enable the SCI, initialize the remaining Baud Rate Generator Register

2) For synchronous receive operation, the data and receive clock must not have significant skew between clock and data. The received data and clock are internally synchronized to INTCLK clock

For synchronous transmit operation, a general purpose I/O port pin must be programmed to output the CLKOUT signal from the baud rate generator. If the SCI is provided with an external transmission clock source, there will be skew equivalent to two INTCLK periods between clock and data

The synchronous data will be transmitted on the fall of the transmit clock. The synchronous received data will be latched into the SCI on the rising edge of the provided receive clock

ON-CHIP MEMORY

13.1 INTRODUCTION

Within the standard ST9 addressing space (64K bytes of Program memory plus 64K bytes of Data memory) 4 types of memory can be included on the same device.

- ROM up to 32K (in blocks of 4K bytes)
- EPROM up to 32K (in blocks of 4K bytes)
- RAM up to 2K (in blocks of 256 bytes)
- EEPROM up to 1K (in blocks of 256 bytes)

ST9 microcontrollers with program memory on-chip require non-volatile memory from 0000h (the Reset vector). If this memory is not present, the ST9 is defined as **ROMLESS**, that is, the Reset vector is automatically fetched from external memory (see Chapter 6).

On-chip memory access times are not affected by Hardware or Software Wait states. Access is performed at the maximum speed (3 CPUCLK cycles) unless otherwise stated.

When on-chip and off-chip memories are mapped at the same address in the same address map, the internal memory takes priority and the external DS is not generated.

13.2 EPROM

On-chip EPROM memory, once programmed, is functionally equivalent to ROM memory. It is recommended to cover the EPROM window with an opaque label when the device is in operation. This protects RAM and other on-chip logic from malfunctions.

13.2.1 EPROM Programming Board

EPROM programming is made by the ST9 Programming board. When programming has been completed successfully, the ST9 is ready to be used as ROM device.

13.2.2 EPROM Erasure

The recommended EPROM erasure procedure is exposure to short wave ultraviolet light which has a wavelength 2537 Å. The integrated dose for erasure should be a minimum of 15 joules/cm^2.

The erasure time with this dosage is approximately 20 minutes using an ultraviolet lamp with 12000 µW/cm^2 power rating.

The ST9 should be placed within 1 inch of the lamp tube during the erasure.

13.3 EEPROM DESCRIPTION

The EEPROM memory can be mapped in data space, or in memory space, or both. When mapped in program memory, the EEPROM generates one internal WAIT cycle in Read mode. Operations on EEPROM are controlled by programming the EECR register mapped in the Register File (page 0).

13.3.1 EEPROM Control Register

EECR R241 (F1h) Page 0 Read/Write (except EEBUSY: read only) EEPROM Control Register

Reset value : 0000 0000b (00h)

7							0
0	VERIFY	EESTBY	EEIEN	PLLST	PLLEN	EEBUSY	EEWEN

bit 7 = **B7:** This bit is forced to "0" after reset and **MUST** not be modified by the user.

bit 6 = **VERIFY:** *Set Verify mode.* Verify (active high) is used to activate the verify mode.

The verify mode provides a guarentee of good retention of the programmed bit. When active, the reading voltage on the cell gate is decreased from 1.2V to 0.0V, decreasing the current from the programmed cell by 20%. If the cell is well programmed (to "1"), a "1" will still be read, otherwise a "0" will be read.

NOTE : The verify mode must not be used during an erasing or a programming cycle).

bit 5 = **EESTBY:** *EEPROM Stand-By.* STBY = "1" switches off all power consumption sources inside the EEPROM. Any attempt to access the EEPROM when STBY = "1" will produce unpredictable errors.

NOTE: After EESTBY is reset, the user must wait 6 CPUCLK cycles (e.g 1 NOP instruction) before selecting the EEPROM

bit 4 = **EEIEN:** *EEPROM Interrupt Enable.* INTEN = "1" disables the external interrupt source and enables the EEPROM to send its interrupt request to the central interrupt unit at the end of each write procedure.

bit 3 = **PLLST:** *Parallel Write Start.* Setting PLLST to "1" starts the parallel writing procedure. It can be set only if PLLEN is already set. PLLST is internally reset at the end of the programming sequence.

bit 2 = **PLLEN:** *Parallel write Enable.* Setting PLLEN to "1" enables the parallel writing mode which allows the user to write up to 16 bytes at the same time. PLLEN is internally reset at the end of the programming sequence.

bit 1 = **EEBUSY:** *BUSY.* When this read only bit is high, an EEPROM write operation is in progress and any attempt to access the EEPROM is aborted.

bit 0 = **EEWEN:** *EEPROM Write Enable.* Setting this bit allows programming of the EEPROM, when low a writing attempt has no effect.

13.3.2 EEPROM Programming Time

No timing routine is required to control the programming time as dedicated circuitry takes care of the EEPROM programming time (The typical programming time is 6 ms).

13.3.3 EEPROM Interrupt Management

At the end of each write procedure the EEPROM sends an interrupt request (if EEIEN bit is set). The EEPROM shares its interrupt channel with the external interrupt source INT4, from which the priority level is derived.

Care must be taken when EEIEN is reset. The associated external interrupt channel must be disabled (by reseting bit 4 of EIMR, R244) along with reseting the interrupt pending bit (bit 4 of EIPR, R243). A delay instruction (at least 1 NOP instruction) must be inserted between these two operations

13.3.4 EEPROM Programming Procedure

The programming of an byte of EEPROM memory is equivalent to writing a byte into a RAM location after verifying that EEBUSY bit is low. Instructions operating on word data (16 bits) will not access the EEPROM.

13.3.5 EEPROM Programming Voltage

No external Vpp voltage is required, an internal 18 Volt charge-pump gives the required energy by a dedicated oscillator pumping at a typical frequency of 5 MHz, regardless of the external clock.

NOTE: After a programming procedure, the user must wait 6 CPUCLK cycles (1 NOP instruction) after the EEBUSY bit is reset In most applications, this NOP instruction can be saved when the time necessary to test EECR and branch to the next instruction is longer than 6 CPUCLK cycles

13.3.6 PARALLEL Programming Procedure

Parallel programming is a feature of the EEPROM macrocell. One up to sixteen bytes of a same row can be programmed at once (same row = addresses to be programmed only differ by their bits A0, A1, A2 and A3).

The software procedure is:

- set PLLEN, access the EEPROM by writing at the desired locations, providing that each address location is on the same row (i.e the distance between their addresses is of 128 bytes). While PLLST = "0", the programming procedure will not start and data is loaded in the latches.
- the programing procedure is performed by setting PLLST. Both PLLEN and PLLST are internally reset at the end of the programming session.

Example:

If the 512 bytes of EEPROM is mapped from 0000h to 01FFh in Data Space, it will be composed of 4 modules of 128 bytes each, allowing up to 4 bytes to be programmed in parallel.

Programming in parallel all the row of location 0012 will program, after setting PLLEN:

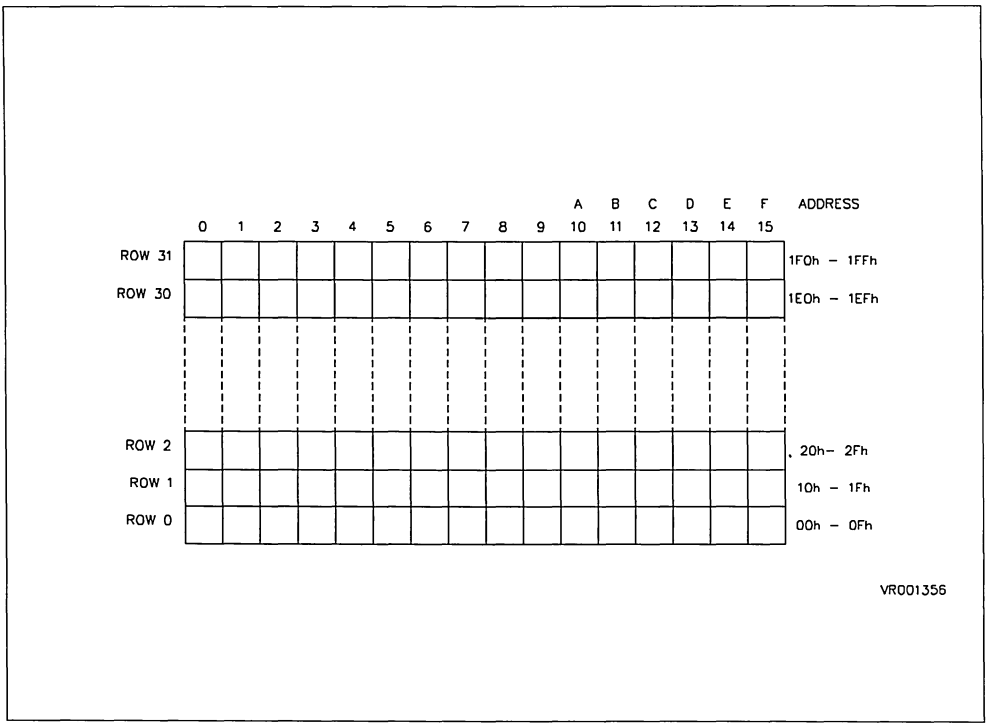
- Address 0012 - Address 0092 - Address 0112 - Address 0192

NOTE: User should take care when using PLLEN bit, and avoid the following sequence:

- Set PLLEN
- Load latches
- Reset PLLEN (without modifying PLLST)

Such a sequence will lose the data loaded in the latches.

Figure 13-1. EEPROM Parallel Programming Rows



13.4 MEMORY PROTECTION

13.4.1 Introduction

The ST9 family allows protection of its on-chip memory to provide the security required by many applications. The on-chip memory can be protected on Read and/or Write access by Hardware (as defined by the device specification).

The protection mechanism inhibits access to the on-chip memories in these two cases:

- When an access is attempted during the execution of an instruction fetch from an off-chip memory.
- When an access is attempted under a DMA transfer.

If the access is made under the two above conditions, the read operation will always return FFh, a write operation will have no affect on the memory content.

The protection mechanism for ROM devices is activated under customer request at the mask programming level. EPROM versions allow the activation of the memory protection after loading the EPROM content.

WARNING: If protection is enabled, the following must be taken into consideration:

- *No DMA will be allowed to/from the memory if protected, even if the DMA transfer is internal to the device.*
- *Care must be taken when using the protected memory as the SYSTEM STACK:*

A `RET` (or `IRET`) instruction from a subroutine resident in an off-chip memory will fail when accessing the (protected) stack to restore the program counter.

Also, if an Interrupt occurs when the instruction executed has been fetched from off-chip, the `PUSH` operation of the Program Counter will fail when accessing the protected stack.

13.4.2 Security Register

When available, ST9 Security Register provides the access control to the internal memory from external sources by both external program and DMA by a series of User programmable fuses. These fuses allow the level of security to be defined by the user at each stage of the product development (e.g. installation of factory codes and subsequently end customer personal codes).
The fuses themselves are one time programmable based on EEPROM technology, thus they require the ST9 to have existing EEPROM or an external voltage supply in order to generate the programming voltage.

Each memory element on-chip (ROM, RAM, EEPROM) has an independent protection enable selectable for each level.

The protection enable options are selectable by metal mask during manufacture and are activated by the programming of fuses present in the Security Register mapped at register R255 of I/O page 59 decimal (3Bh). These fuses are based on EEPROM technology and require a high voltage to program the fuse. In the ST9040, this is supplied by the charge pump present in the EEPROM memory, so that the EEPROM must be in an active state (STBY low), before security fuse programming is activated. Please check with SGS-THOMSON for the reference pin of the external voltage of other devices with the Security Register.
The Security fuses are TESTLOCK (TLCK) and HARDWARE LOCK (HLCK). These are both one-time programmable, once these are programmed THE PROTECTION CANNOT BE DISABLED, so care must be made in the use of this feature.

Figure 13-2. Security Mask Options

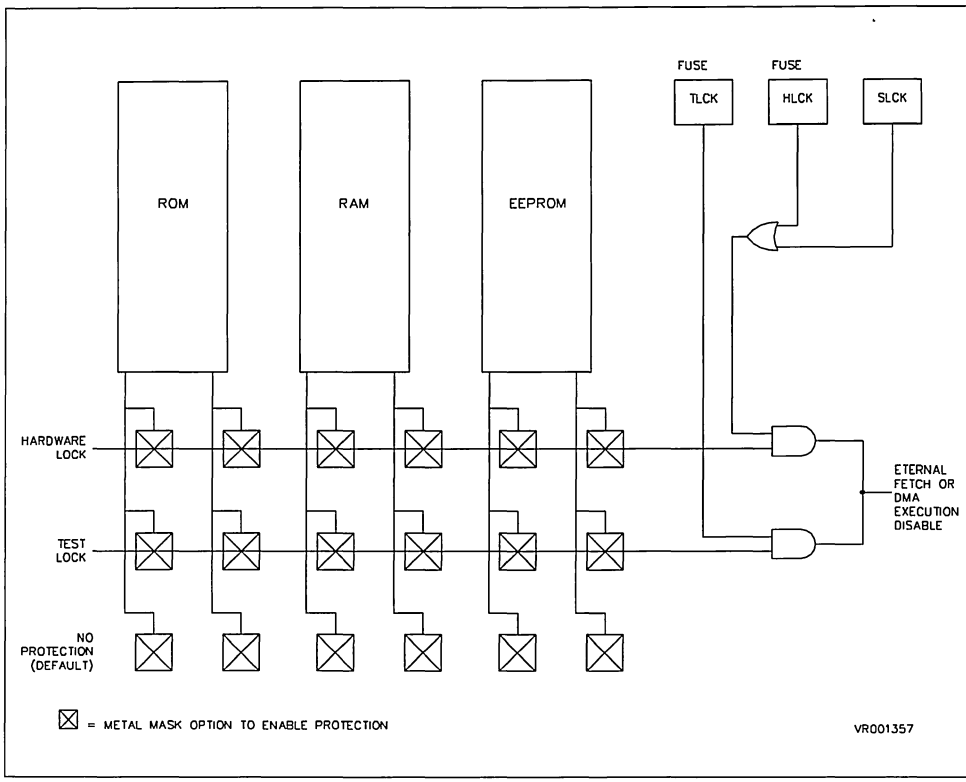
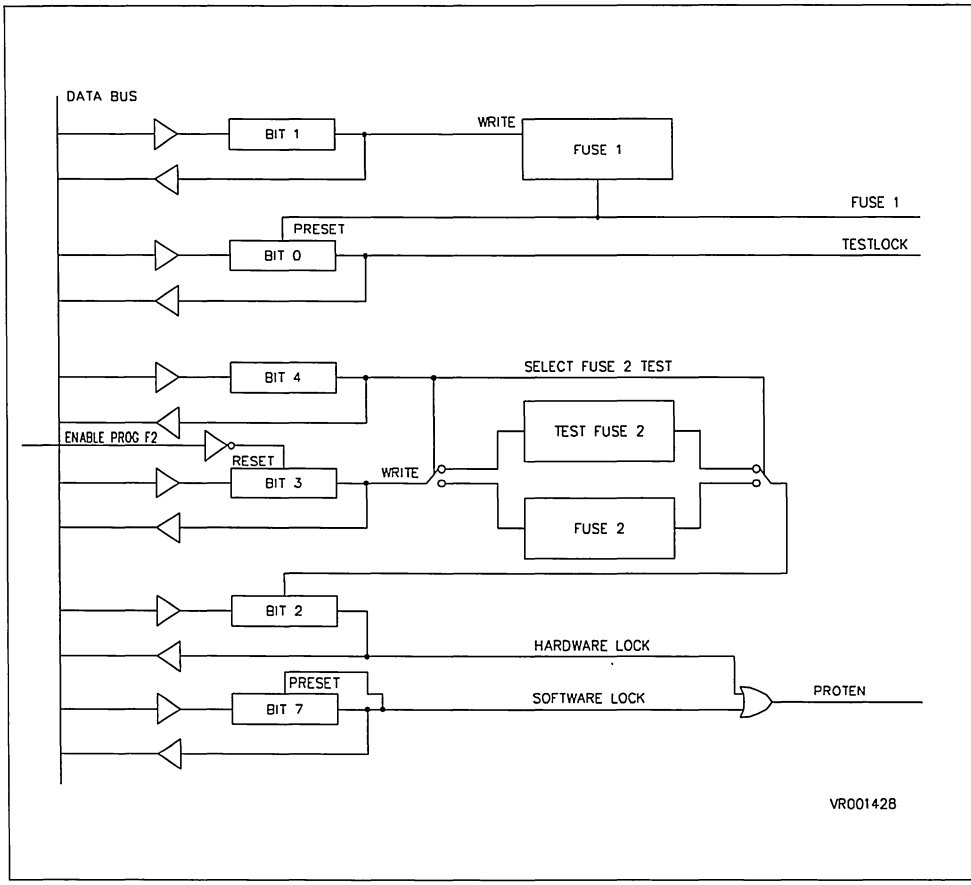


Figure 13-3. Security Fuses



13.4.3 Testlock

The TESTLOCK protection level may be programmed by SGS-THOMSON during the manufacturing cycle, if requested, or may be programmed by the user before the release of the end equipment. If programmed by SGS-THOMSON, the user must include in the masked ROM the routines to program RAM and EEPROM. The Reset vector must also be provided. There is no possibility to test the ST9, or to use external memory to program RAM and EEPROM, once this bit is programmed. The TLCK bit may also be programmed by the user after the internal read/write memory has been programmed. Consult SGS-THOMSON for further information.

The TESTLOCK level of protection allows the basic protection of the user-designated on-chip memory e.g. the ROM contents, while, optionally, allowing the further programming of the on-chip RAM and EEPROM memory from external programs.

13.4.4 Hardware Lock

The HARDWARE lock protection level is provided to give a final high security protection after the programming of the internal memory (e.g. access codes, serial numbers or PIN codes). If the on-chip memory has been mask selected for protection by this level, then the programming of the HLCK bit will give the full protection.

WARNING: THIS IS TOTAL PROTECTION, there is no method to access or test on-chip memory from any external source once this level is programmed.

A third fuse is present in parallel with HLCK, and is used by SGS-THOMSON as part of the final device check to test the security functions.

When the Testlock and Hardware lock bits are virgin, the value read from the bits are the value previously written, allowing verification of the operation of the protection mechanism. The fuses are programmed by setting the appropriate Write Fuse bit (WF1 for TLCK and WF2 for HLCK) with the programming voltage applied. For the ST9040 this is achieved by making a dummy read from the EEPROM memory (STBY must be active). This triggers the charge pump to generate the high voltage necessary to program the fuse.
The Write Fuse bits must be held to a "1" state for the whole of the programming cycle, the end of programming may be monitored on the TLCK or HLCK bit.

13.4.5 Software Lock

A third level of security may be achieved by the latching of a third bit, SLCK which provides an additional level of security in parallel with the Hardware lock. This is provided in the case of a failure, by externally induced means, of the OTP hardware locks.

It should also be noted that the ST9 on-chip programmable memories (RAM and EEPROM) are mapped into Data Space, preventing the operation of "Trojan Horse" programs (external programs loaded into internal memory to bypass the read out protection), and that the High Impedance mode can be activated to prevent the external address lines of the ST9 from echoing the addresses used by the internal security program.

SEC R255 (FFh) Page 59 Read/Write
SECURITY Control Register

Reset value : 0xx0 0*0*b

7							0
SLCK	-	-	TF2	WF2	HLCK	WFI	tlck

* depends on the state of the fuses

bit 7 = **SLCK:** *Software Lock* The reset value of this bit is "0". Writing a "1" to this bit will set the Protection Enable and will lock the "1" bit.

bit 6 = not used.

bit 5 = not used.

bit 4 = **TF2:** *Test Fuse 2.* This bit must be maintained at a "0" level

bit 3 = **WF2:** *Write Fuse 2.* Writing a "1" in this bit will blow Fuse 2 when the high voltage Vpp rises (when writing to on-chip EEPROM memory or rising an external Vpp supply voltage).
This bit must remain at "1" for the whole programming of the Fuse. The end of the programming can be monitored with bit 2. Care must be taken before writing this bit to "1" because the programming of this bit is irreversible.

bit 2 = **HLCK:** *Hardware Lock.* When Fuse 2 is virgin, the value read in this bit is the value previously written.
When Fuse 2 is programmed, this bit is forced to "1", thus the method of verifying that Fuse 2 is programmed is to write a "0" in this bit and to read back a "1". When Fuse 2 is programmed, the Hardware Lock is forced to "1".
When Fuse 2 is virgin, the accessibility of the on-chip memories can be tested by writing this bit.

bit 1 = **WF1:** *Write Fuse 1.* Writing a "1" in this bit will blow Fuse 1 when the high voltage Vpp rises (when writing to on-chip EEPROM memory or rising an external Vpp supply voltage).
This bit must remain at "1" for the whole programming of the Fuse. The end of the programming can be monitored with bit 0. Care must be taken before writing this bit to "1" because the programming of this bit is irreversible.

bit 0 = **TLCK:** *Testlock.* When Fuse 1 is virgin, the value read in this bit is the value previously written.
When Fuse 1 is programmed, this bit is forced to "1", thus the method of verifying that Fuse 1 is programmed is to write a "0" in this bit and to read back a "1". Testlock controls the accessibility of the on-chip memories
When Fuse 1 is virgin, the accessibility of the on-chip memories can be tested by programming this bit.

ELECTRICAL CHARACTERISTICS

Table 14-1. DC Electrical Characteristics
(V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Value			Unit
			Min.	Typ.	Max.	
V_{IHCK}	Clock Input High Level	External Clock	0.7 V_{DD}		V_{DD} + 0.3	V
V_{ILCK}	Clock Input Low Level	External Clock	– 0.3		0.3 V_{DD}	V
V_{IH}	Input High Level	TTL	2.0		V_{DD} + 0.3	V
		CMOS	0.7 V_{DD}		VDD + 0.3	V
V_{IL}	Input Low Level	TTL	– 0.3		0.8	V
		CMOS	– 0.3		0.3 V_{DD}	V
V_{IHRS}	Reset Input High Level		0.7 V_{DD}		V_{DD} + 0.3	V
V_{ILRS}	Reset Input Low Level		–0.3		0.3 V_{DD}	V
V_{HYRS}	Reset Input Hysteresis		0.3		1.5	V
V_{OH}	Output High Level	Push Pull, Iload = – 0.8mA	V_{DD} – 0.8			V
V_{OL}	Output Low Level	Push Pull or Open Drain, Iload = – 1.6mA			0.4	V
I_{WPU}	Weak Pull-up Current	Bidirectional Weak Pull-up, V_{OL} = 0V	– 80	– 200	– 420	µA
I_{APU}	Active Pull-up Current, for INT0 and INT7 only	V_{IN} < 0.8V	– 80	– 200	– 420	µA
I_{LKIO}	I/O Pin Input Leakage	Input/Tri-State, 0V < V_{IN} < V_{DD}	– 10		+ 10	µA
I_{LKRS}	Reset Pin Input Leakage	0V < V_{IN} < V_{DD}	– 30		+ 30	µA
I_{LKAD}	A/D Pin Input Leakage	Alternate Function, Open Drain, 0V < V_{IN} < V_{DD}	– 3		+ 3	µA
I_{LKAP}	Active Pull-up Input Leakage	0V < V_{IN} < 0.8V	– 10		+ 10	µA
I_{LKOS}	OSCIN Pin Input Leakage	0V < V_{IN} < V_{DD}	– 10		+ 10	µA

Table 14-1. DC Electrical Characteristics (Continued)

Symbol	Parameter	Test Conditions	Value			Unit
			Min.	Typ.	Max.	
I_{DD}	Run Mode Current	24MHz, Note 1		32	70	mA
		4MHz, Note 1		6	12	mA
I_{DP2}	Run Mode Current Prescale by 2	24MHz, Note 1		19	40	mA
		4MHz, Note 1		4	8	mA
I_{WFI}	WFI Mode Current	24MHz, Note 1		9	18	mA
		4MHz, Note 1		2.5	5	mA
I_{HALT}	HALT Mode Current	24MHz, Note 1			100	µA

Note.
1. All I/O Ports are configured in Bidirectional Weak Pull-up Mode with no DC load, External Clock pin (OSCIN) is driven by square wave external clock No peripheral working. External interface not active (Internal Program Execution).

Figure 14-1. DC Test Conditions

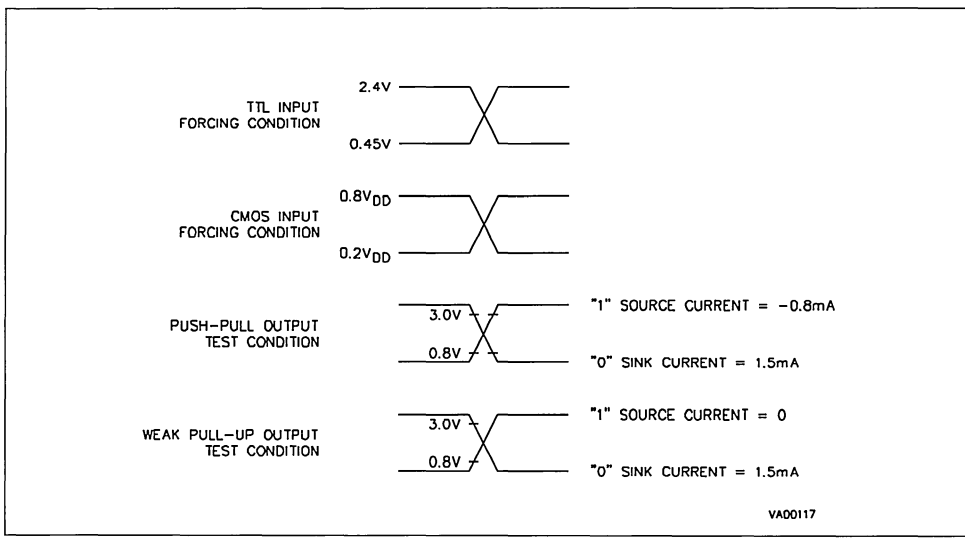


Table 14-2. Clock Timing Table
(V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, unless otherwise specified)

N°	Symbol	Parameter	Value		Unit	Note
			Min.	Max.		
1	TpC	OSCIN Clock Period	41.5		ns	1
			83		ns	2
2	TrC, TfC	OSCIN Rise and Fall Time		12	ns	
3	TwCL, TwCH	OSCIN Low and High Width	25	12	ns	1
			38		ns	2

Notes:
1. Clock divided by 2 internally (MODER.DIV2=1)
2. Clock not divided by 2 internally (MODER.DIV2=0)

Figure 14-2. Clock Timing

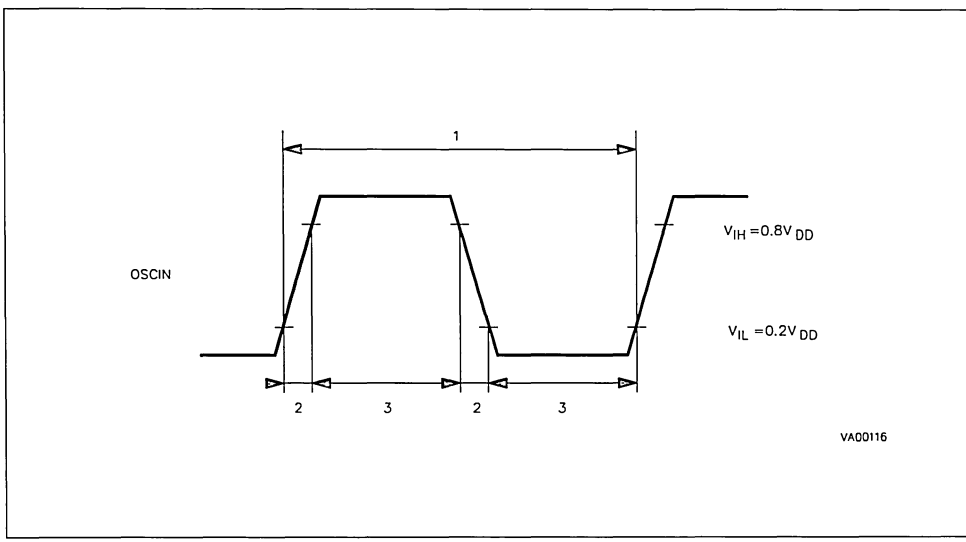


Table 14-3. External Bus Timing Table
(V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, CPUCLK = 12MHz, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TsA (AS)	Address Set-up Time before $\overline{AS}$ ↑	TpC (2P+1) –22	TWCH+PTpC –18	20		ns
2	ThAS (A)	Address Hold Time after $\overline{AS}$ ↑	TpC –17	TwCL –13	25		ns
3	TdAS (DR)	$\overline{AS}$ ↑ to Data Available (read)	TpC (4P+2W+4) –52	TpC (2P+W+2) –51		115	ns
4	TwAS	$\overline{AS}$ Low Pulse Width	TpC (2P+1) –7	TwCH+PTpC –3	35		ns
5	TdAz (DS)	Address Float to $\overline{DS}$ ↓	0	0	0		ns
6	TwDSR	$\overline{DS}$ Low Pulse Width (read)	TpC (4P+2W+3) –20	TwCH+TpC (2P+W+1) –16	105		ns
7	TwDSW	$\overline{DS}$ Low Pulse Width (write)	TpC (2P+2W+2) –13	TpC (P+W+1) –13	70		ns
8	TdDSR (DR)	$\overline{DS}$ ↓ to Data Valid Delay (read)	TpC (4P+2W-3) –50	TwCH+TpC(2P+W+1) – 46		75	ns
9	ThDR (DS)	Data to $\overline{DS}$ ↑ Hold Time (read)	0	0	0		ns
10	TdDS (A)	$\overline{DS}$ ↑ to Address Active Delay	TpC –7	TwCL –3	35		ns
11	TdDS (AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	TpC –18	TwCL –14	24		ns
12	TsR/W (AS)	R/$\overline{W}$ Set-up Time before $\overline{AS}$ ↑	TpC (2P+1) –22	TwCH+PTpC –18	20		ns
13	TdDSR (R/W)	$\overline{DS}$ ↑ to R/$\overline{W}$ and Address Not Valid Delay	TpC –9	TwCL –5	33		ns
14	TdDW (DSW)	Write Data Valid to $\overline{DS}$ ↓ Delay (write)	TpC (2P+1) –32	TwCH+PTpC –28	10		ns
15	ThDS (DW)	Data Hold Time after $\overline{DS}$ ↑ (write)	TpC –9	TwCL –5	33		ns
16	TdA (DR)	Address Valid to Data Valid Delay (read)	TpC (6P+2W+5) –68	TwCH+TpC (3P+W+2) –64		140	ns
17	TdAs (DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	TpC –18	TwCL –14	24		ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescaler value of zero and zero wait status

Legend:
P = Clock Prescaling Value
W = Wait Cycles
TpC = OSCIN Period
TwCH = High Level OSCIN half period
TwCL = Low Level OSCIN half period

Figure 14-3. External Bus Timing

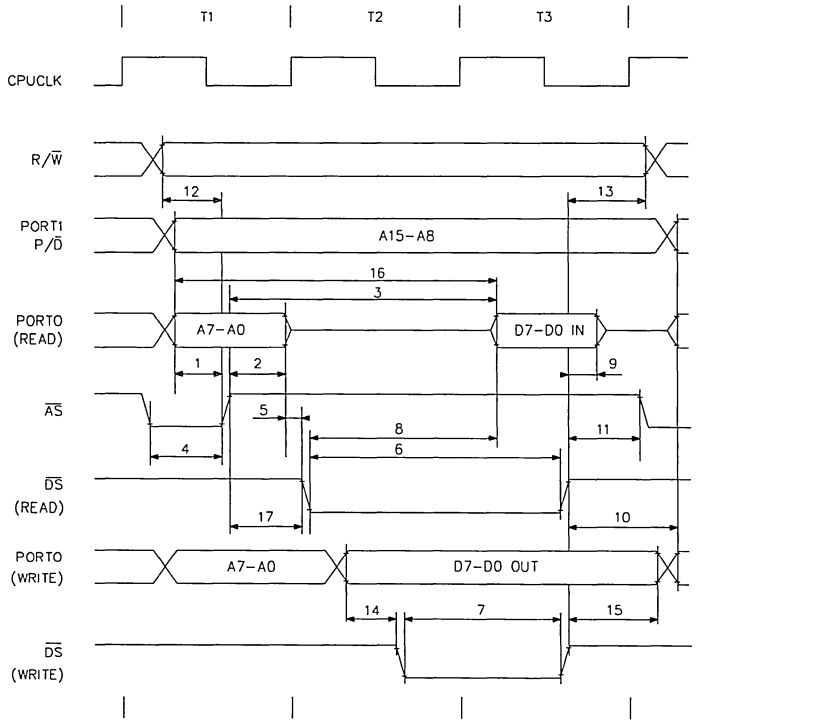


Table 14-4. External Wait Timing Table (V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TdAS (WAIT)	$\overline{AS}$ ↑ to $\overline{WAIT}$ ↓ Delay	2(P+1)TpC –29	(P+1)TpC –29		40	ns
2	TdAS (WAIT)	$\overline{AS}$ ↑ to $\overline{WAIT}$ ↑ Minimum Delay	2(P+W+1)TpC –4	(P+W+1)TpC –4	80		ns
3	TdAS (WAIT)	$\overline{AS}$ ↑ to $\overline{WAIT}$ ↑ Maximum Delay	2(P+W+1)TpC –29	(P+W+1)TpC –29		83W+40	ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value right hand two columns show the timing minimum and maximum for an external clock at 24MHz divided by 2, prescale value of zero and zero wait status

Figure 14-4. External Wait Timing

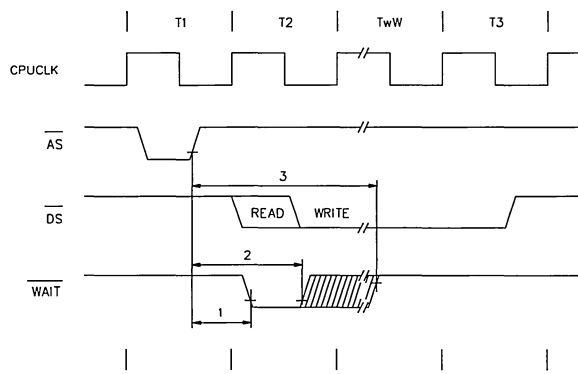


Table 14-5. Bankswitch, AC-Timing Table

N°	Symbol	Parameter	Value		Unit	Note
			Min.	Max.		
1	TdBAS	Bank switch from $\overline{AS}$ High to low transition		45	ns	1
2	TdBLH	Bank switch rising edge from $\overline{AS}$ High to Low transition		40	ns	2
3	TwBSW	Bank switch -FFh- pulse width		CPUCLK	ns	1

Notes .
1 Page 0 R255 BS=0
2 Page0 R255 BS=1

Figure 14-5. Bankswitch, AC-Timing

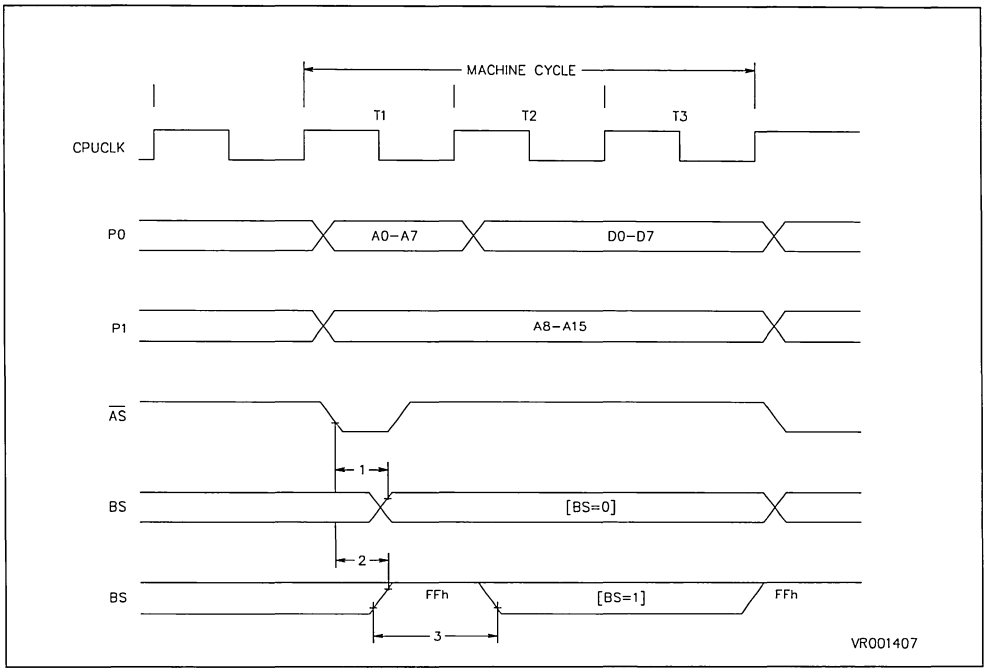


Table 14-6. Bankswitch R/$\overline{W}$ Modified from $\overline{DS}$ Delay

N°	Symbol	Parameter	Value		Unit
			Min.	Max.	
1	TdRMD	R/$\overline{W}$ modified from $\overline{DS}$ delay		35	ns

Figure 14-6. Bankswitch R/$\overline{W}$ Modified from $\overline{DS}$ Delay

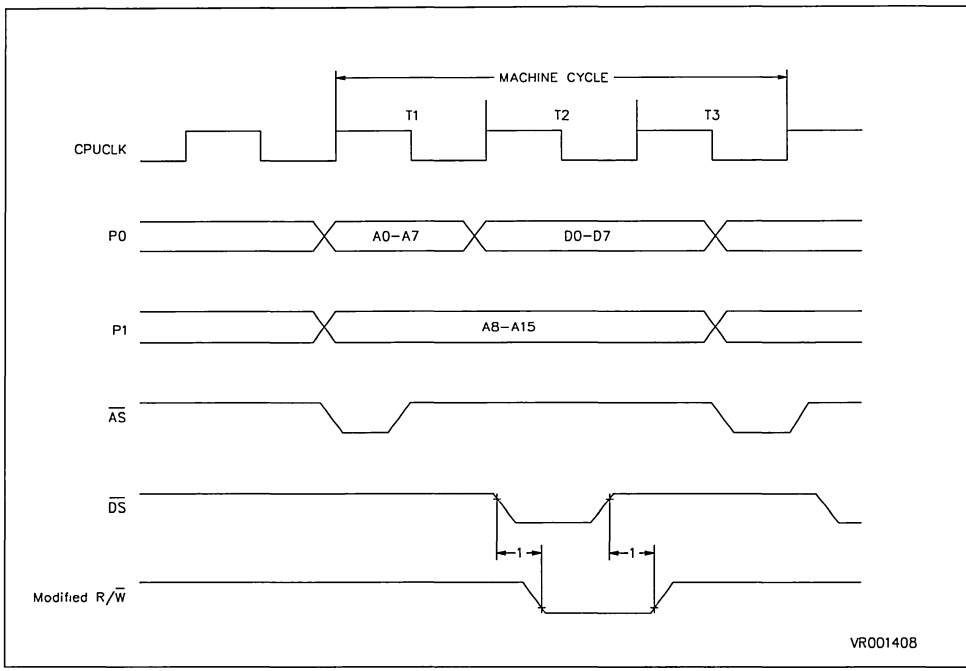
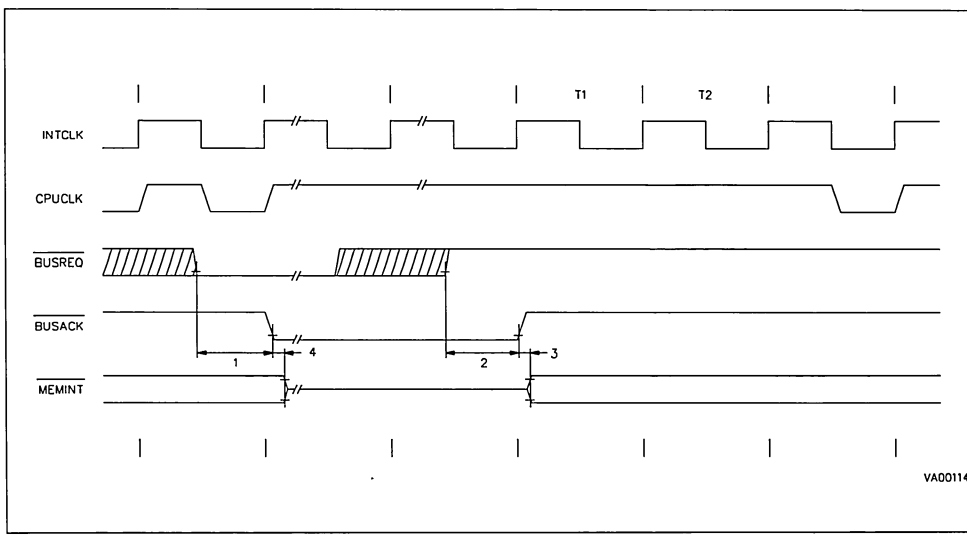


Table 14-7. Bus Request/Acknowledge Timing Table (V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2	OSCIN Not Divided By 2	Min.	Max.	
1	TdBR (BACK)	$\overline{BREQ}$ ↓ to $\overline{BUSACK}$ ↓	TpC+8	TwCL+12	50		ns
			TpC(6P+2W+7)+65	TpC(3P+W+3)+TwCL+65		360	ns
2	TdBR (BACK)	$\overline{BREQ}$ ↑ to $\overline{BUSACK}$ ↑	3TpC+60	TpC+TwCL+60		185	ns
3	TdBACK (BREL)	$\overline{BUSACK}$ ↓ to Bus Release	20	20		20	ns
4	TdBACK (BACT)	$\overline{BUSACK}$ ↑ to Bus Active	20	20		20	ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value right hand two columns show the timing minimum and maximum for an external clock at 24MHz divided by 2, prescale value of zero and zero wait status

Figure 14-7. Bus Request/Acknowledge Timing



Note: MEMINT = group of memory interface signals: $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, P00-P07, P10-P17.

Table 14-8. Handshake Timing Table (V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Min.	Max.	Unit
			OSCIN Divided By 2		OSCIN Not Divided By 2				
			Min.	Max.	Min.	Max.			
1	TwRDY	RDRDY, WRRDY Pulse Width in One Line Handshake	2TpC (P+W+1) –18		Tp (P+W+1) – 18		65		ns
2	TwSTB	$\overline{RDSTB}$, $\overline{WRSTB}$ Pulse Width	2TpC+12		TpC+12		95		ns
3	TdST (RDY)	$\overline{RDSTB}$, or $\overline{WRSTB}$ ↑ to RDRDY or WRRDY ↓		TpC+45		(TpC-TwCL) +45		87	ns
4	TsPD (RDY)	Port Data to RDRDY ↑ Set-up Time	(2P+2W+1) TpC –25		TwCH+(W+P) TpC –25		16		ns
5	TsPD (RDY)	Port Data to WRRDY ↓ Set-up Time in One Line Handshake	43		43		43		ns
6	ThPD (RDY)	Port Data to WRRDY ↓ Hold Time in One Line Handshake	0		0		0		ns
7	TsPD (STB)	Port Data to $\overline{WRSTB}$ ↑ Set-up Time	10		10		10		ns
8	ThPD (STB)	Port Data to WRSTB ↑ Hold Time	25		25		25		ns
9	TdSTB (PD)	$\overline{RDSTBD}$ ↑ to Port Data Delay Time in Bidirectional Handshake		35		35		35	ns
10	TdSTB (PHZ)	$\overline{RDSTB}$ ↑ to Port High-Z Delay Time in Bidirectional Handshake		25		25		25	ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted
The value right hand two columns show the timing minimum and maximum for an external clock at 24 MHz divided by 2, prescaler value of zero and zero wait status.

Legend:
P = Clock Prescaling Value (R235 4,3,2)
W = Programmable Wait Cycles (R252.2 1 0/5,4,3) + External Wait Cycles

Figure 14-8. Handshake Timing

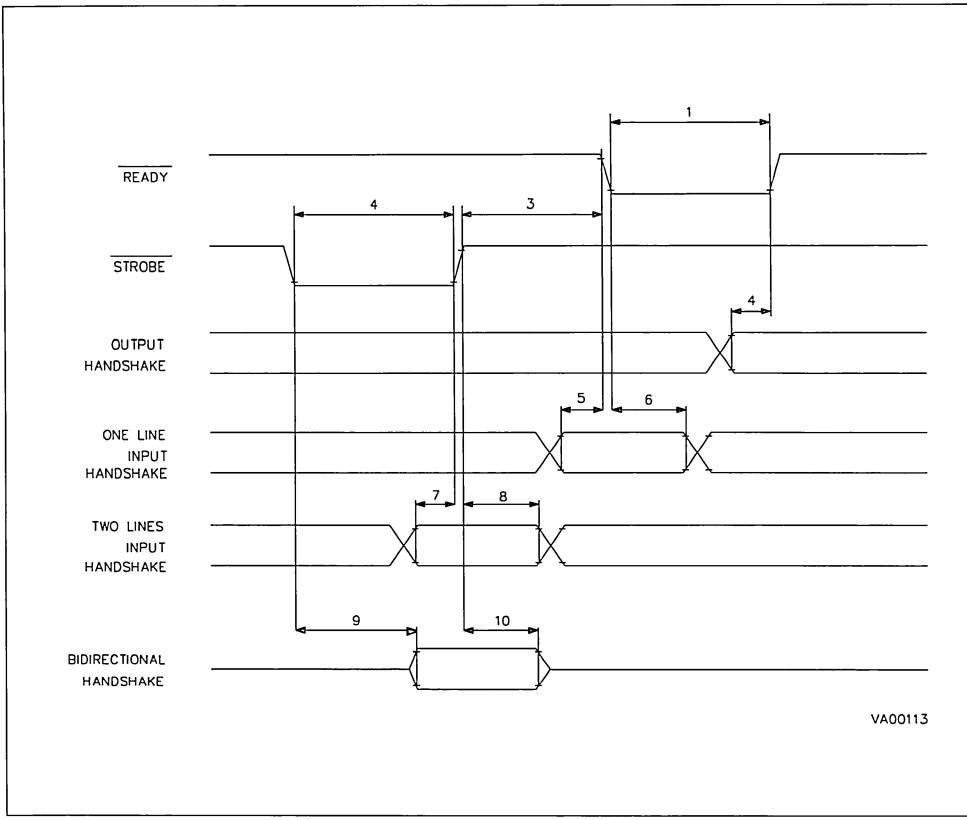


Table 14-9. External Interrupt Timing Table (V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, INTCLK = 12MHz, Push-pull output configuration, unless otherwise specified)

N°	Symbol	Parameter	Value (Note)				Unit
			OSCIN Divided By 2 Min.	OSCIN Not Divided By 2 Min.	Min.	Max.	
1	TwLR	Low Level Minimum Pulse Width in Rising Edge Mode	2TpC+12	TpC+12	95		ns
2	TwHR	High Level Minimum Pulse Width in Rising Edge Mode	2TpC+12	TpC+12	95		ns
3	TwHF	High Level Minimum Pulse Width in Falling Edge Mode	2TpC+12	TpC+12	95		ns
4	TwLF	Low Level Minimum Pulse Width in Falling Edge Mode	2TpC+12	TpC+12	95		ns

Note: The value left hand two columns show the formula used to calculate the timing minimum or maximum from the oscillator clock period, prescale value and number of wait cycles inserted.
The value right hand two columns show the timing minimum and maximum for an external clock at 24MHz divided by 2, prescale value of zero and zero wait status

Figure 14-9. External Interrupt Timing

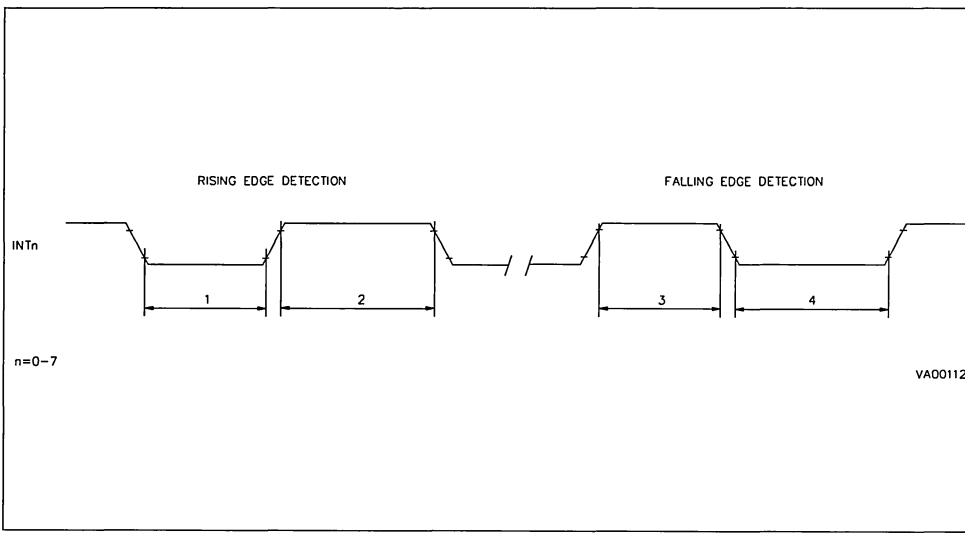


Table 14-10. Timer/Watchdog Timing Table (V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, INTCLK = 12MHz, Output alternate function set as Push-pull)

N°	Symbol	Parameter	Value		Unit
			Min.	Max.	
1	TwWDOL	WDOUT Low Pulse Width	620		ns
2	TwWDOH	WDOUT High Pulse Width	620		ns
3	TwWDIL	WDIN Low Pulse Width	350		ns
4	TwWDIH	WDIN High Pulse Width	350		ns

Figure 14-10. Timer/Watchdog Timing

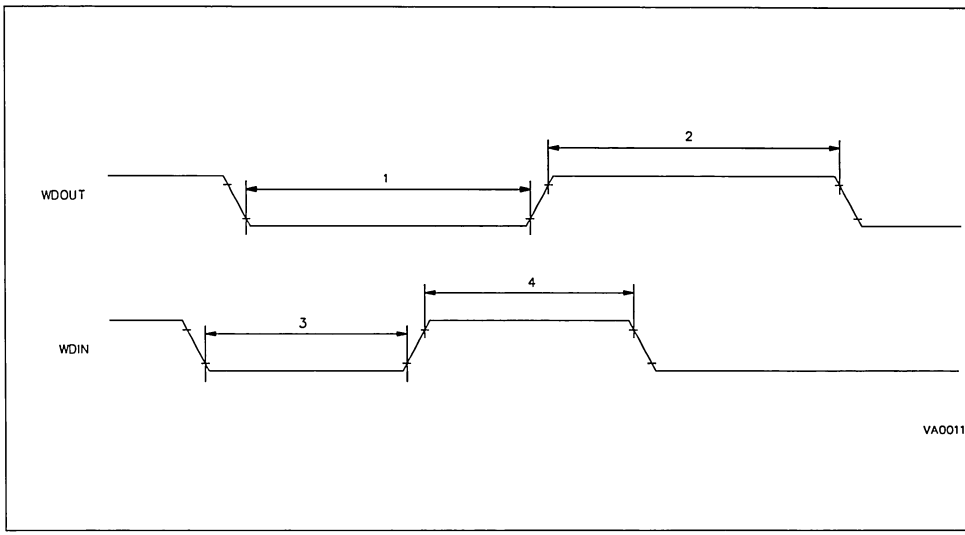


Table 14-11. SPI Timing Table (V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, INTCLK = 12MHz, Output alternate function set as Push-pull)

N°	Symbol	Parameter	Value		Unit
			Min.	Max.	
1	TsDI	Input Data Set-up Time	100		ns
2	ThDI (1)	Input Data Hold Time	1/2 TpC+100		ns
3	TdOV	SCK to Output Data Valid		100	ns
4	ThDO	Output Data Hold Time	-20		ns
5	TwSKL	SCK Low Pulse Width	300		ns
6	TwSKH	SCK High Pulse Width	300		ns

Note: 1 TpC is the Clock period.

Figure 14-11. SPI Timing

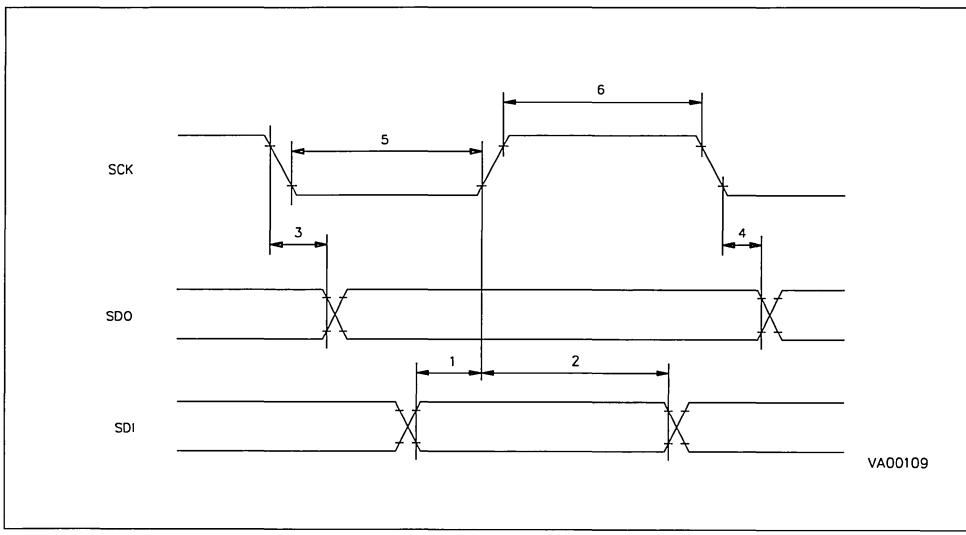


Table 14-12. A/D External Trigger Timing

N°	Symbol	Parameter	OSCIN Divided by 2 (2)		OSCIN Not Divided by 2 (2)		Value (3)		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	
1	Tw_{LOW}	External trigger pulse width	2 x Tpc		Tpc		83	-	ns
2	Tw_{HIGH}	External trigger pulse distance	2 x Tpc		Tpc		83	-	ns
3	Tw_{EXT}	External trigger active edges distance (1)	276n x Tpc		138n x Tpc		n x 11.5	-	µs
4	Td_{STR}	ADTRG falling edge and first conversion start	Tpc	3 x Tpc	.5 x Tpc	1.5 x Tpc	41.5	125	ns

Notes:
1. n = number of autoscanned channels (1 < n < 8)
2 Variable clock (Tpc = OSCIN clock period)
3. CPUCLK = 12MHz

Figure 14-12. A/D External Trigger Timing

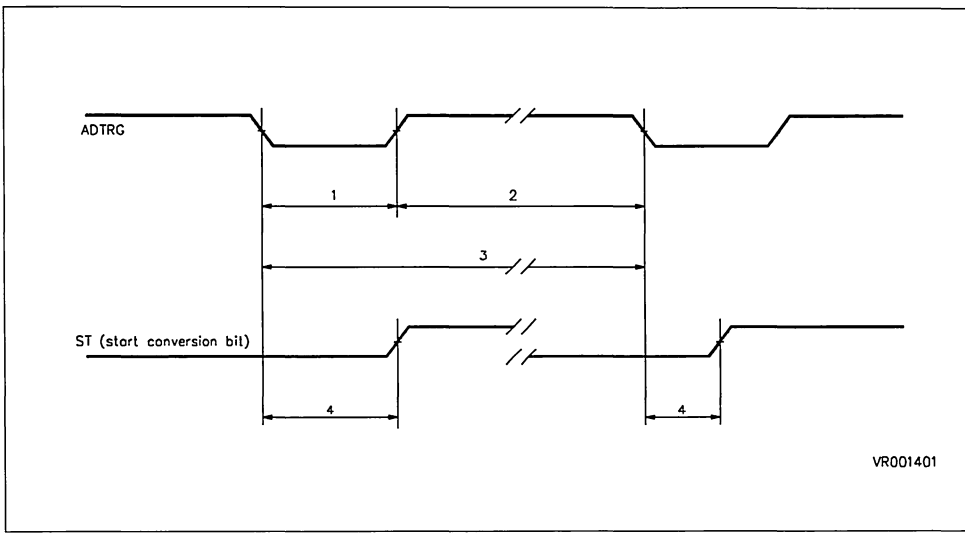


Table 14-13. A/D Internal Trigger Timing

N°	Symbol	Parameter	OSCIN Divided by 2 (2)		OSCIN Not Divided by 2 (2)		Value (3)		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	
1	TW_{HIGH}	Internal trigger pulse width	Tpc		.5 x Tpc		41.5	-	ns
2	TW_{LOW}	Internal trigger pulse distance	6 x Tpc		3 x Tpc		250	-	ns
3	TW_{EXT}	Internal trigger active edges distance (1)	276n x Tpc		138n x Tpc		n x 11.5	-	µs
4	TW_{STR}	Internal delay between INTRG rising edge and first conversion start	Tpc	3 x Tpc	.5 x Tpc	1.5 x Tpc	41.5	125	ns

Notes:
1. n = number of autoscanned channels (1 < n < 8)
2. Variable clock (Tpc = OSCIN clock period)
3. CPUCLK = 12MHz

Figure 14-13. A/D Internal Trigger Timing

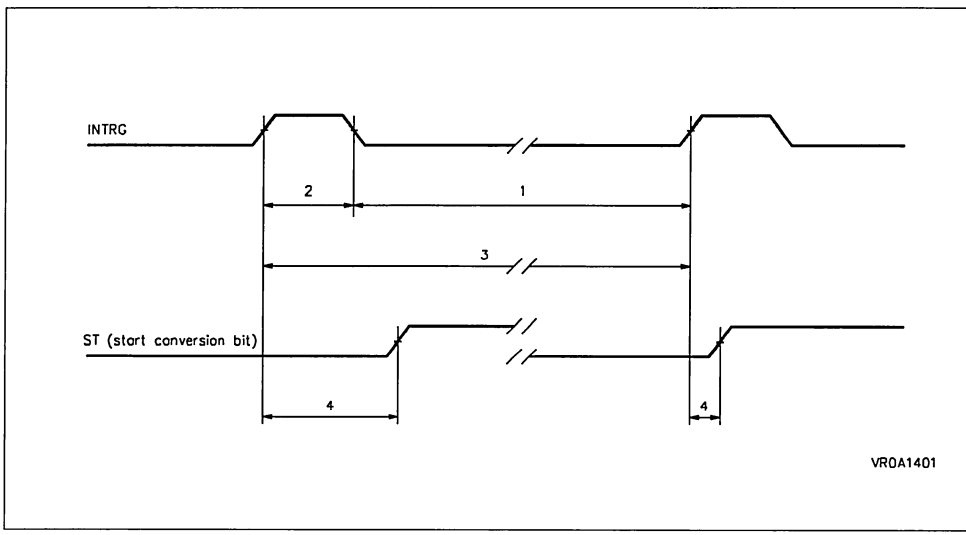


Table 14-14. A/D Channel Enable Timing

N°	Symbol	Parameter	OSCIN Divided by 2 (2)		OSCIN Not Divided by 2 (2)		Value (3)		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	
1	T_{WEXT}	CEn Pulse width (1)	276n x Tpc		138n x Tpc		n x 11.5	-	µs

Notes:
1. n = number of autoscanned channels (1 < n < 8)
2. Variable clock (Tpc = OSCIN clock period)
3. CPUCLK = 12MHz

Figure 14-14. A/D Channel Enable Timing

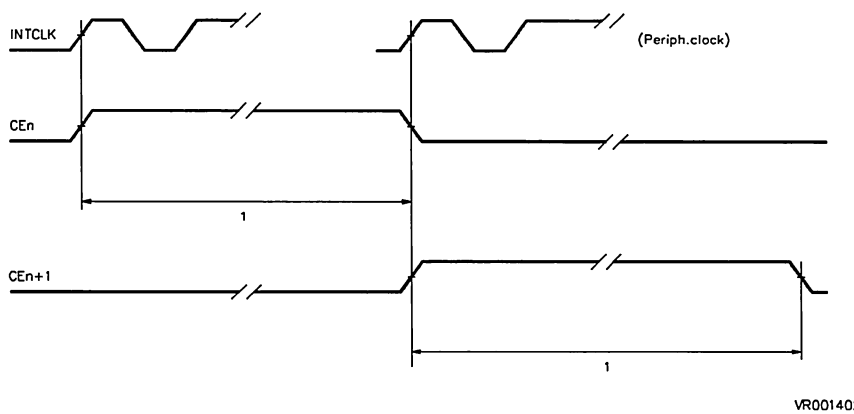


Table 14-15. A/D Analog Specifications

Parameter	Typical (1)	Minimum	Maximum	Units (2)	Notes
Analog Input Range A_{VCC}		3	A_{VCC} V_{CC}	V	
Conversion time		11.5		V	
Sample time		3		µs	(3, 4)
Power-up time		60		µs	(2)
Resolution	8	8		µs	
Monotonicity	GUARANTEED			bits	
No missing codes	GUARANTEED				
Zero input reading		00		Hex	
Full scale reading			FF	Hex	
Offset error	.5		1	LSBs	(6)
Gain error	.5		1	LSBs	(6)
Diff. Non Linearity	±.3	±.2	±.5	LSBs	(6)
Int. Non Linearity			1	LSBs	(6)
Absolute Accuracy			1	LSBs	(6)
S/N		45	49	dB	
A_{VCC}/A_{VSS} Resistance	13.5	16	11	Kohm	
Input Resistance	12	8	15	Kohm	(5)
Hold Capacitance			30	pF	
Input Leakage			3	µA	

Notes:
1 The values are expected at 25 degree Centigrade with A_{VCC} = 5V
2. "LSBs", as used here, has a value of $A_{VCC}/256$
3 @ 24MHz external clock
4. Including sample time
5. It must be intended as the internal series resistance before the sampling capacitor
6 This is a typical expected value, but not a tested production parameter.
If V(i) is the value of the i-th transition level (0 < i < 255), the performance of the A/D converter has been valued as follows
OFFSET ERROR = deviation between the actual V(0) and the ideal V(0) (=1/2 LSB)
GAIN ERROR = deviation between the actual V(255) and the ideal V(255) (=AVCC-3/2 LSB)
DNL ERROR = max {[V(i) - V(i-1)]/LSB - 1}
INL ERROR = max {[V(i) - V(0)]/LSB - i}
ABS. ACCURACY= overall max conversion error
S/N ratio has been valued by sampling a sinusoidal input waveform and then calculating its Fast Fourier Transform.

Table 14-16. Multifunction Timer Unit External Timing

N°	Symbol	Parameter	OSCIN Divided by 2 (3)	OSCIN Not Divided by 2 (3)	Value (4)		Unit	Note
					Min.	Max.		
1	Tw_{CTW}	External clock/trigger pulse width	2n x Tpc	n x Tpc	n x 83	-	ns	1
2	Tw_{CTD}	External clock/trigger pulse distance	2n x Tpc	n x Tpc	n x 83	-	ns	1
3	Tw_{AED}	Distance between two active edges	6 x Tpc	3 x Tpc	249	-	ns	
4	Tw_{GW}	Gate pulse width	12 x Tpc	6 x Tpc	498	-	ns	
5	Tw_{LBA}	Distance between TINB pulse edge and the following TINA pulse edge	2 x Tpc	Tpc	83	-	ns	2
6	Tw_{LAB}	Distance between TINA pulse edge and the following TINB pulse edge	0		0	-	ns	2
7	Tw_{AD}	Distance between two TxINA pulses	0		0	-	ns	2
8	Tw_{OWD}	Minimum output pulse width/distance	6 x Tpc	3 x Tpc	249	-	ns	

Notes:
1. n = 1 if the input is rising OR falling edge sensitive
   n = 3 if the input is rising AND falling edge sensitive
2. In Autodiscrimination mode
3. Variable clock (Tpc = OSCIN period)
4. INTCLK = 12 MHz

Figure 14-15. Multifunction Timer Unit External Timing

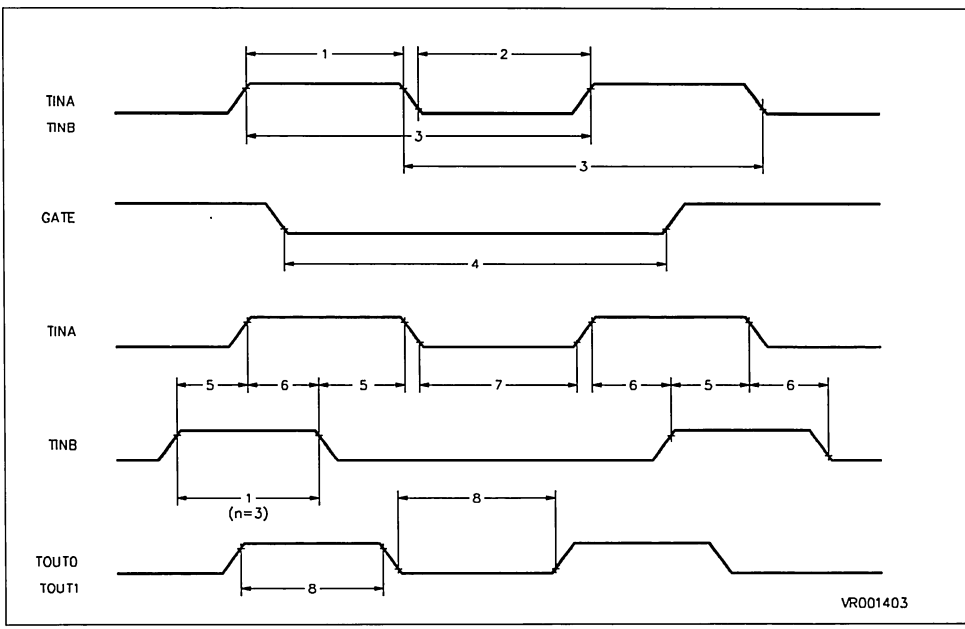
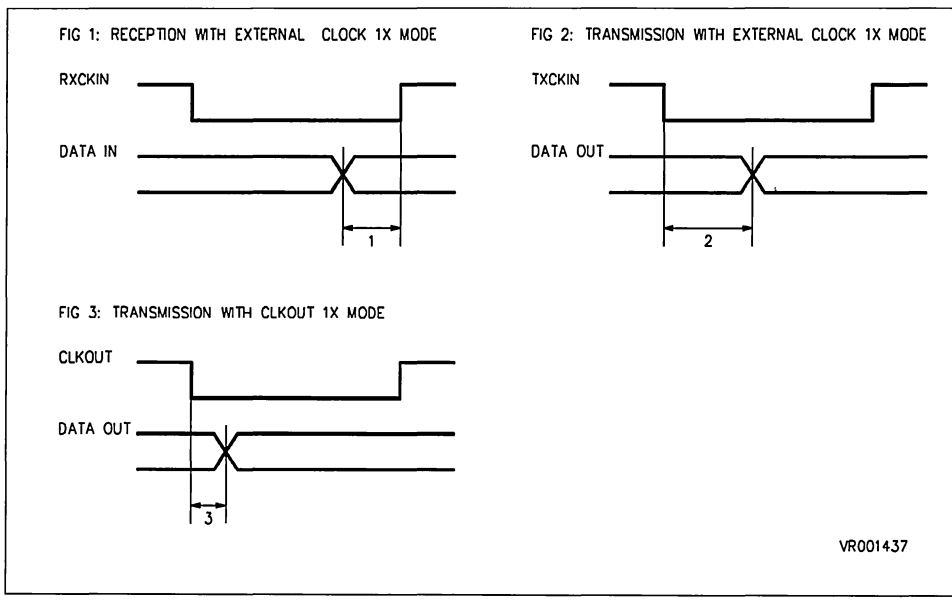


Table 14-17. SCI Timing Table (V_{DD} = 5V ± 10% T_A = - 40°C to + 85°C, Cload = 50pF, INTCLK = 12MHz, Output alternate function set as Push-pull)

N°	Symbol	Parameter	Condition	Value		Unit
				Min.	Max.	
	F_{RxCKIN}	Frequency of RxCKIN	1 x mode		$F_{CK}/8$	Hz
			16 x mode		$F_{CK}/4$	Hz
	Tw_{RxCKIN}	RxCKIN shortest pulse	1 x mode	4 T_{CK}		s
			16 x mode	2 T_{CK}		s
	F_{TxCKIN}	Frequency of TxCKIN	1 x mode		$F_{CK}/8$	Hz
			16 x mode		$F_{CK}/4$	Hz
	Tw_{TxCKIN}	TxCKIN shortest pulse	1 x mode	4 T_{CK}		s
			16 x mode	2 T_{CK}		s
1	Ts_{DS}	DS (Data Stable) before rising edge of RxCKIN	1 x mode reception with RxCKIN	$T_{PC}/2$		ns
2	Td_{D1}	TxCKIN to Data out delay Time	1 x mode transmission with external clock C load <100pF		2.5 T_{PC}	ns
3	Td_{D2}	CLKOUT to Data out delay Time	1 x mode transmission with CLKOUT	350		ns

Note: $F_{CK} = 1/T_{CK}$



SALES OFFICES

EUROPE

DENMARK

2730 HERLEV
Herlev Torv, 4
Tel (45-42) 94 85 33
Telex 35411
Telefax (45-42) 948694

FINLAND

LOHJA SF-08150
Karjalankatu, 2
Tel (358-12) 155 11
Telefax (358-12) 155 66

FRANCE

94253 GENTILLY Cedex
7 - avenue Gallieni - BP 93
Tel (33-1) 47 40 75 75
Telex 632570 STMHQ
Telefax (33-1) 47 40 79 10

67000 STRASBOURG
20, Place des Halles
Tel (33) 88 75 50 66
Telefax (33) 88 22 29 32

GERMANY

6000 FRANKFURT
Gutleutstrasse 322
Tel (49-69) 237492-3
Telex 176997 689
Telefax (49-69) 231957
Teletex 6997689=STVBP

8011 GRASBRUNN
Bretonischer Ring 4
Neukeferloh Technopark
Tel (49-89) 46006-0
Telex 528211
Telefax (49-89) 4605454
Teletex 897107=STDISTR

3000 HANNOVER 51
Rotenburger Strasse 28A
Tel (49-511) 615960
Telex 175118418
Teletex 5118418 CSFBEH
Telefax (49-511) 6151243

5202 HENNEF
Reuther Strasse 1A-C
Tel (49-2242) 6088
(49-2242) 4019/4010
Telefax (49-2242) 84181

8500 NÜRNBERG 20
Erlenstegenstrasse, 72
Tel (49-911) 59893-0
Telex 626243
Telefax (49-911) 5980701

7000 STUTTGART 31
Mittlerer Pfad 2-4
Tel (49-711) 13968-0
Telex 721718
Telefax (49-711) 8661427

ITALY

20090 ASSAGO (MI)
Vle Milanofiori - Strada 4 - Palazzo A/4/A
Tel (39-2) 89213 1 (10 linee)
Telex 330131 - 330141 SGSAGR
Telefax (39-2) 8250449

40033 CASALECCHIO DI RENO (BO)
Via R Fucini, 12
Tel (39-51) 591914
Telex 512442
Telefax (39-51) 591305

00161 ROMA
Via A Torlonia, 15
Tel (39-6) 8443341
Telex 620653 SGSATE I
Telefax (39-6) 8444474

NETHERLANDS

5652 AR EINDHOVEN
Meerenakkerweg 1
Tel (31-40) 550015
Telex 51186
Telefax (31-40) 528835

SPAIN

08021 BARCELONA
Calle Platon, 6 4th Floor, 5th Door
Tel (34-3) 4143300-4143361
Telefax. (34-3) 2021461

28027 MADRID
Calle Albacete, 5
Tel (34-1) 4051615
Telex 46033 TCCEE
Telefax (34-1) 4031134

SWEDEN

S-16421 KISTA
Borgarfjordsgatan, 13 - Box 1094
Tel (46-8) 7939220
Telex 12078 THSWS
Telefax (46-8) 7504950

SWITZERLAND

1218 GRAND-SACONNEX (GENEVA)
Chemin Francois-Lehmann, 18/A
Tel (41-22) 7986462
Telex 415493 STM CH
Telefax (41-22) 7984869

UNITED KINGDOM and EIRE

MARLOW, BUCKS
Planar House, Parkway
Globe Park
Tel (44-628) 890800
Telex 847458
Telefax (44-628) 890391

ORDER CODE: DBST9TMST/1