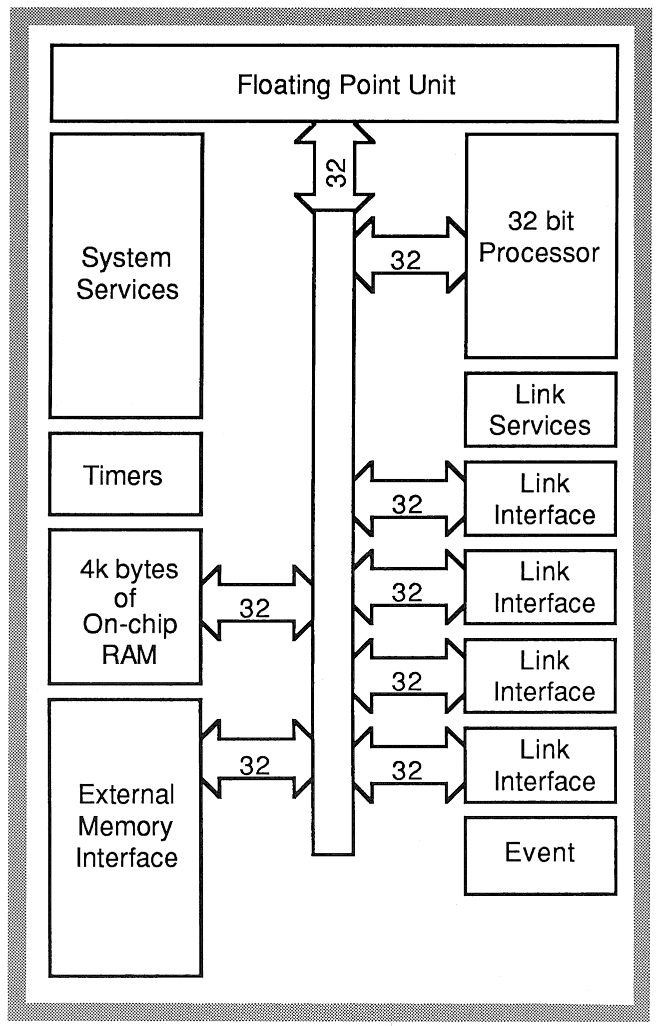
Preliminary Data

IMS T800 transputer

inmos®

FEATURES

Integral hardware 64 bit floating point unit
ANSI-IEEE 754-1985 floating point representation
Sustained 1.5(2.25†) Mflops
32 bit architecture with 15 MIPS† performance
Hardware and pin compatible with IMS T414-20
4 Kbytes on chip RAM for 120 Mbytes/sec† data rate
32 bit configurable memory interface
Directly addresses 4 Gbytes at 40 Mbytes/sec†
High performance graphics support
Sub-microsecond context switch & interrupt latency
Four 5/10/20 Mbits/sec INMOS serial links
Hardware scheduler for concurrent programs
Internal timers for real time processing
External event interrupt
Support for run-time error diagnostics
Boots from communication link or ROM
On-chip DRAM controller
Internal program continues during DMA
Optional external memory wait states
Single 5 MHz clock input
Single +5V ±10% power supply



APPLICATIONS

Scientific and mathematical applications
High speed multi processor systems
High performance graphics processing
Supercomputers
Workstations and workstation clusters
Digital signal processing
Accelerator processors
Distributed databases
System simulation
Telecommunications
Robotics
Fault tolerant systems
Image processing
Molecular modelling
Pattern recognition
Artificial intelligence

Note: † indicates a reference to a 30 MHz device.

42 1082 00

April 1987

CONTENTS

1 Introduction

2 Pin designations

3 Processor
- 3.1 Registers
- 3.2 Instructions
- 3.3 Processes and concurrency
- 3.4 Priority
- 3.5 Communications
- 3.6 Timers
- 3.7 Instruction set summary

4 Floating Point Unit

5 System Services
- 5.1 Power
- 5.2 CapPlus, CapMinus
- 5.3 ClockIn
- 5.4 ProcSpeedSelect0-2
- 5.5 Reset
- 5.6 Boot
- 5.7 Peek and poke
- 5.8 Analyse
- 5.9 Error, ErrorIn

6 Memory

7 External Memory Interface
- 7.1 ProcClockOut
- 7.2 Tstates
- 7.3 Internal access
- 7.4 MemAD2-31
- 7.5 MemnotWrD0
- 7.6 MemnotRfD1
- 7.7 notMemRd
- 7.8 notMemS0-4
- 7.9 notMemWrB0-3
- 7.10 MemConfig
- 7.11 notMemRf
- 7.12 MemWait
- 7.13 MemReq, MemGranted

8 Events

9 Links

10 Electrical specifications
- 10.1 DC electrical characteristics
- 10.2 Equivalent circuits
- 10.3 AC timing characteristics
- 10.4 Power rating

11 Package specifications
- 11.1 Pin grid array package

12 Ordering details

1 Introduction

The IMS T800 transputer is a 32 bit CMOS microcomputer with a 64 bit floating point unit and graphics support. It has 4 Kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the occam model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond.

The processor speed of a device can be pin-selected in stages from 17.5 MHz up to the maximum allowed for the part. A device running at 30 MHz achieves an instruction throughput of 15 MIPS.

The IMS T800 provides high performance arithmetic and floating point operations. The 64 bit floating point unit provides single and double length operation to the ANSI-IEEE 754-1985 standard for floating point arithmetic. It is able to perform floating point operations concurrently with the processor, sustaining a rate of 1.5 Mflops at a processor speed of 20 MHz and 2.25 Mflops at 30 MHz.

High performance graphics support is provided by microcoded block move instructions which operate at the speed of memory. The two dimensional block move instructions provide for contiguous block moves as well as block copying of either non-zero bytes of data only or zero bytes only. Block move instructions can be used to provide graphics operations such as text manipulation, windowing, panning, scrolling and screen updating.

Cyclic redundancy checking (CRC) instructions are available for use on arbitrary length serial data streams, to provide error detection where data integrity is critical. Another feature of the IMS T800, useful for pattern recognition, is the facility to count bits set in a word.

The IMS T800 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 100 nanoseconds (40 Mbytes/sec) for a 30 MHz device. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

System Services include processor reset and boot control, together with facilities for error analysis. Error signals may be daisy-chained in multi-transputer systems.

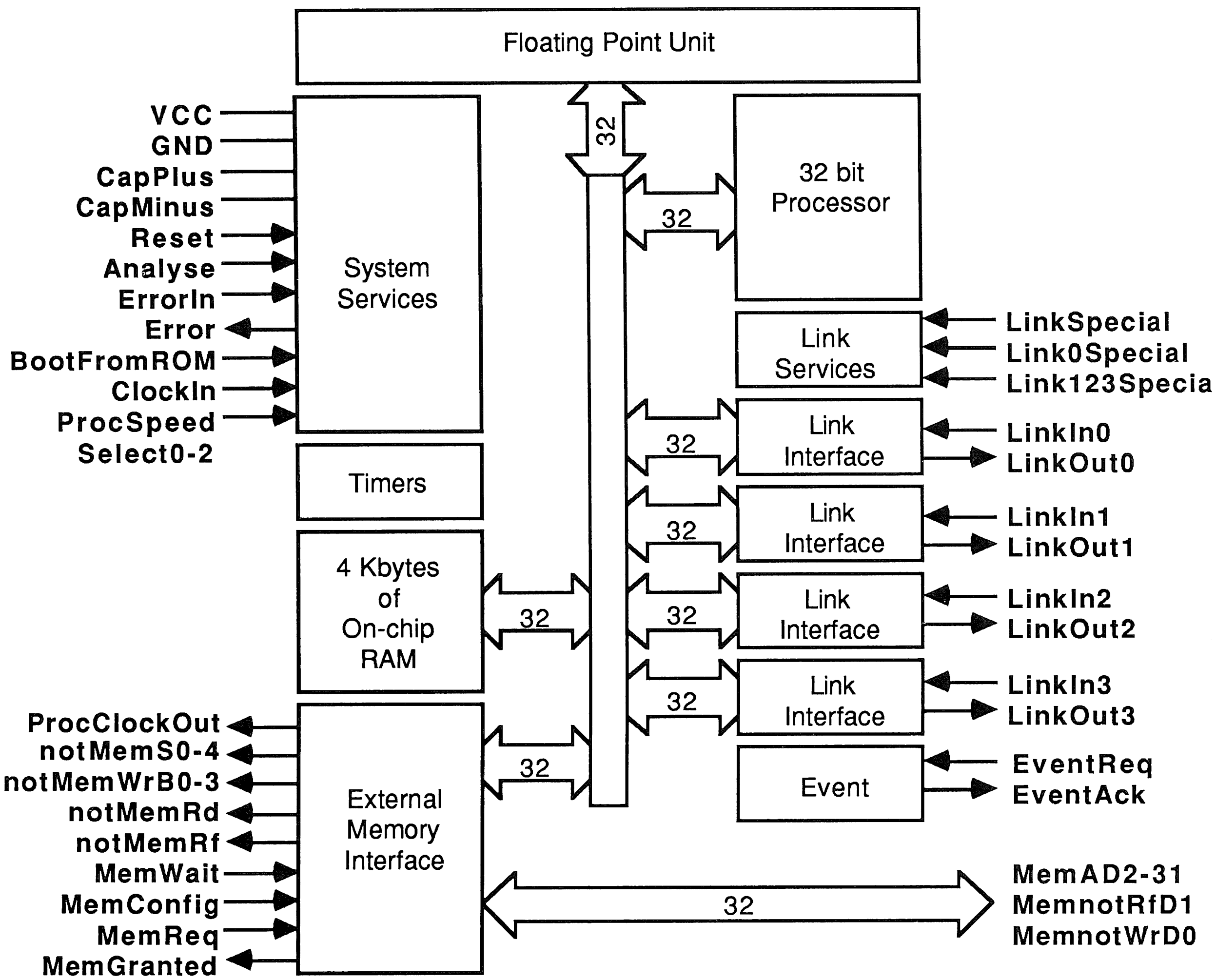
The standard INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T800 links support the standard operating speed of 10 Mbits per second, but also operate at 5 or 20 Mbits per second. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

The IMS T800-20 is pin compatible with the IMS T414-20, as the extra inputs used are all held to ground on the IMS T414. The IMS T800-20 can thus be plugged directly into a circuit designed for a 20 MHz version of the IMS T414. Software should be recompiled, although no changes to the source code are necessary.

The transputer is designed to implement the occam language, detailed in the Occam Reference Manual, but also efficiently supports other languages such as C, Pascal and Fortran. Access to the transputer at machine level is seldom required, but if necessary refer to The Transputer Instruction Set - A Compiler Writers' Guide.

This data sheet supplies hardware implementation and characterisation details for the IMS T800. It is intended to be read in conjunction with the Transputer Reference Manual, which details the architecture of the transputer and gives an overview of occam.

For convenience of description, the IMS T800 operation is split into the basic blocks shown in the Block Diagram.



IMS T800 Block Diagram

2 Pin designations

System Services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
ClockIn	in	Input clock
ProcSpeedSelect0-2	in	Processor speed selectors
Reset	in	System reset
Error	out	Error indicator
ErrorIn	in	Error daisychain input
Analyse	in	Error analysis
BootFromRom	in	Boot from external ROM or from link
HoldToGND		Must be connected to **GND**
DoNotWire		Must not be wired

External Memory Interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemnotWrD0	in/out	Multiplexed data bit 0 and write cycle warning
MemnotRfD1	in/out	Multiplexed data bit 1 and refresh warning
MemAD2-31	in/out	Multiplexed data and address bus
notMemRd	out	Read strobe
notMemWrB0-3	out	Four byte-addressing write strobes
notMemS0-4	out	Five general purpose strobes
notMemRf	out	Dynamic memory refresh indicator
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemConfig	in	Memory configuration data input

Event

Pin	In/Out	Function
EventReq	in	Event request
EventAck	out	Event request acknowledge

Link

Pin	In/Out	Function
LinkIn0-3	in	Four serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec
Link0Special	in	Select special speed for Link 0
Link123Special	in	Select special speed for Links 1,2,3

Notes

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for the different packages are given in section 11.

3 Processor

The 32 bit processor contains instruction processing logic, instruction pointer, workspace pointer, and an operand register. It directly addresses 4 Gbytes of memory, 4 Kbytes of which is fast on-chip RAM.

3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The **A**, **B** and **C** registers which form an evaluation stack.

A, **B** and **C** are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes **B** into **C**, and **A** into **B**, before loading **A**. Storing a value from **A**, pops **B** into **A** and **C** into **B**.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicity. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

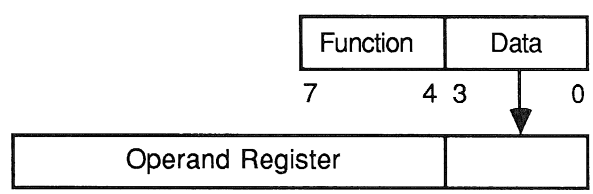
Any location in memory can be accessed relative to the workpointer register, enabling the workspace to be of any size.

Further register details are given in The Transputer Instruction Set - A Compiler Writers' Guide.

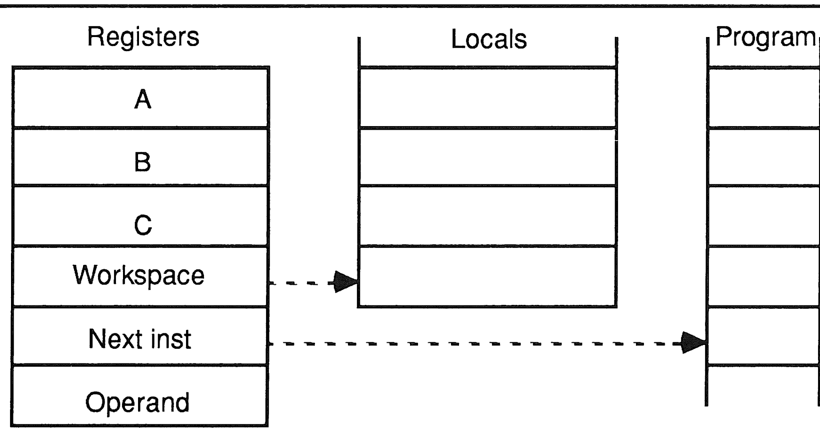
3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4 bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.



Instruction Format



Registers

3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Thirteen of these are used to encode the most important functions. These include

load constant	*add constant*
load local *load local pointer*	*store local*
load non-local	*store non-local*
jump *call*	*conditional jump*

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the **A** register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as occam, C, Fortran, Pascal or ADA.

3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative prefix*.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occuring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as *add*, *exclusive or* and *greater than*. Less frequently occuring operations have encodings which require a single prefix operation.

3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Program	Mnemonic	
x := 0	ldc	0
	stl	x
x := #24	pfix	2
	ldc	4
	stl	x
x := y + z	ldl	y
	ldl	z
	add	
	stl	x

3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte (i.e. without the use of prefix instructions). Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive several instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (section 3.4).

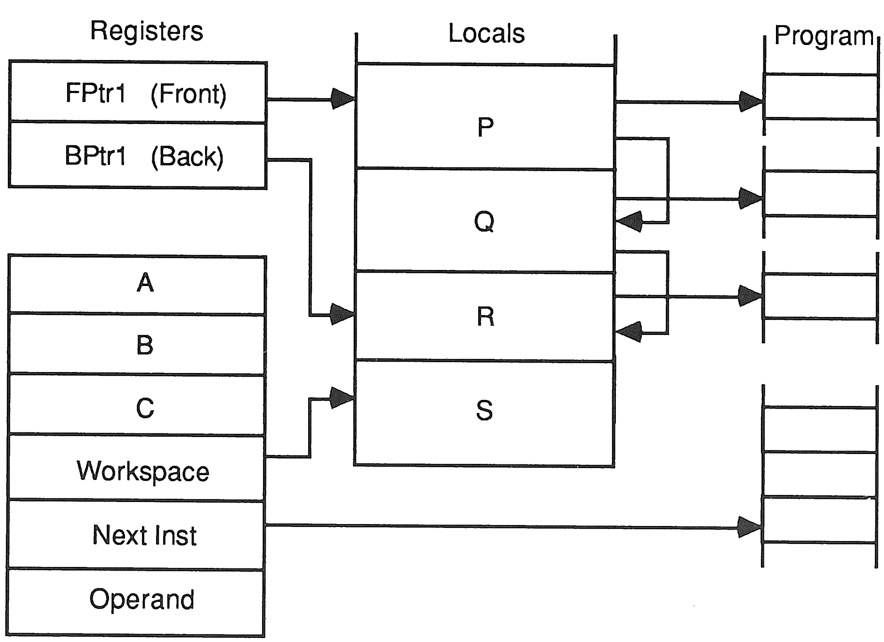
The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active - Being executed
- On a list waiting to be executed

Inactive - Ready to input
- Ready to output
- Waiting until a specified time

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (section 3.4). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List diagram, process **S** is executing and **P**, **Q** and **R** are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.



Linked Process List

High Priority Queue Control Registers

Fptr0	Pointer to front of active process list
Bptr0	Pointer to back of active process list

Low Priority Queue Control Registers

Fptr1	Pointer to front of active process list
Bptr1	Pointer to back of active process list

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next available descheduling point (section 3.7.1). The time slice period is 5120 cycles of **ClockIn**, giving ticks approximately 1ms apart.

A process can only be descheduled on certain instructions, known as descheduling points (section 3.7.1). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1 μs, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

3.4 Priority

The IMS T800 supports two levels of priority. The priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are **n** low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is 2**n**-2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (section 3.7.1).

Each timeslice period lasts for 5120 cycles of the input clock **ClockIn** (approximately 1 millisecond at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum 78 cycles (assuming use of on-chip RAM). If the floating point unit is not being used at the time then the maximum interrupt latency is only 58 cycles. To ensure this latency, certain instructions are interruptable.

3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-to-point, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being *input message* and *output message*.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

3.6 Timers

The transputer has two 32 bit timers which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

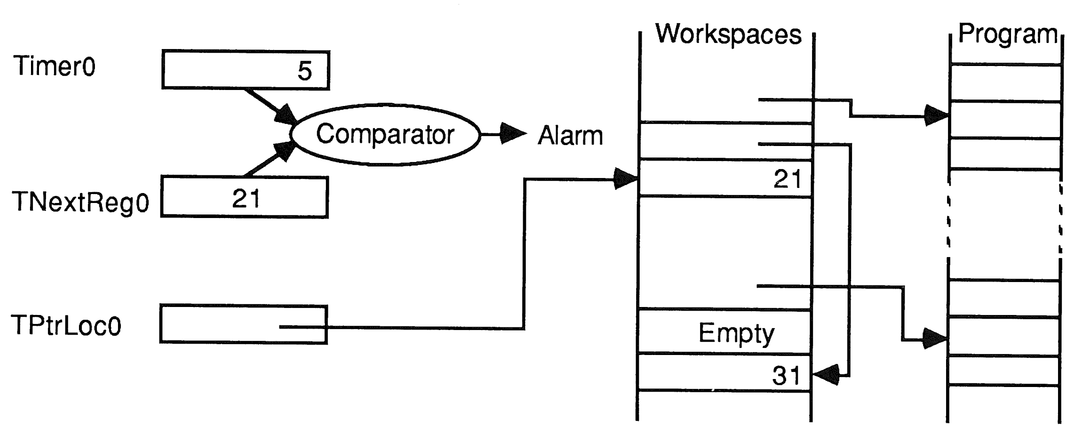
One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks of this timer in one second. It cycles in approximately 76 hours.

Timer Registers

Timer0	Current value of high priority (level 0) process timer
Timer1	Current value of low priority (level 1) process timer
TNextReg0	Indicates time of earliest event on high priority (level 0) timer queue
TNextReg1	Indicates time of earliest event on low priority (level 1) timer queue

The current value of a timer can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

The Timer Registers diagram shows two processes waiting on a timer queue, one waiting for time 21, the other for time 31.



Timer Registers

3.7 Instruction set summary

The Function Codes table gives the basic function code set (section 3.2.1). Where the operand is less than 16 a single byte encodes the complete instruction. If the operand is greater than 15 one prefix instruction (*pfix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*.

Mnemonic		Function code	Memory code
ldc	#3	#4	#43
ldc	#35		
is coded as			
pfix	#3	#2	#23
ldc	#5	#4	#45
ldc	#987		
is coded as			
pfix	#9	#2	#29
pfix	#8	#2	#28
ldc	#7	#4	#47
ldc	-31 (ldc #FFFFFFE1)		
is coded as			
nfix	#1	#6	#61
ldc	#1	#4	#41

The Operation Codes tables give details of operation codes. Where an operation code is less than 16 (e.g. *add*: operation code **05**), the operation can be stored as a single byte comprising the *operate* function code **F** and the operand (**5** in the example). Where an operation code is greater than 15 (e.g. *ladd*: operation code **16**), the *prefix* function code **2** is used to extend the instruction.

Mnemonic		Function code	Memory code
add	(op. code #5)		#F5
is coded as			
opr	add	#F	#F5
ladd	(op. code #16)		#21F6
is coded as			
pfix	#1	#2	#21
opr	#6	#F	#F6

In the Floating Point Operation Codes tables a selector sequence code (section 4) is indicated in the Memory Code column by *s*. The code given in the Operation Code column is the indirection code, the operand for the *ldc* instruction.

The FPU and processor operate concurrently, so the actual throughput of floating point instructions is better than that implied by simply adding up instruction times. For full details see The Transputer Instruction Set - A Compiler Writers' Guide.

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where relevant the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from the table below.

b is the bit number of the highest bit set in register A. Bit 0 is the least significant bit.

m is the bit number of the highest bit set in the absolute value of register A. Bit 0 is the least significant bit.

n is the number of places shifted.

w is the number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.

p is the number of words per row.

r is the number of rows.

The Desch/Error column of the tables indicate if an instruction is a descheduling point (section 3.3) or if it will affect **Error** (section 5.9) or **FP_Error** (section 4).

3.7.1 Descheduling points

The following instructions are the only ones at which a process may be descheduled (section 3.3). They are also the ones at which the processor will halt if **Analyse** is asserted (section 5.8).

input message	*output message*
output byte	*output word*
timer alt wait	*timer input*
stop on error	*alt wait*
jump	*loop end*
end process	*stop process*

3.7.2 Error instructions

The following instructions are the only ones which can affect **Error** (section 5.9) directly. Note, however, that the floating point unit error flag **FP_Error** is set by certain floating point instructions (section 3.7.3), and that **Error** can be set from this flag by *fpcheckerror*.

add	*add constant*
subtract	*multiply*
divide	*remainder*
long add	*long subtract*
long divide	*fractional multiply*
set error	*testerr*
check word	*check subscript from 0*
check single	*check count from 1*
fpcheckerror	

3.7.3 Floating point errors

The following instructions are the only ones which can affect the floating point error flag **FP_Error** (section 4). **Error** is set from this flag by *fpcheckerror* if **FP_Error** is set.

fpadd	*fpsub*
fpmul	*fpdiv*
fpldnladdsn	*fpldnladddb*
fpldnlmulsn	*fpldnlmuldb*
fpremfirst	*fpusqrtfirst*
fpgt	*fpeq*
fpuseterror	*fpuclearerror*
fptesterror	*fprtoi32*
fpuexpincby32	*fpuexpdecby32*
fpumulby2	*fpudivby2*
fpur32tor64	*fpur64tor32*
fpucki32	*fpucki64*
fpuabs	*fpint*

Function Codes

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
0	0X	j	3	jump	Desch
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	
3	3X	ldnl	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	ldnlp	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	ldl	2	load local	
8	8X	adc	1	add constant	Error
9	9X	call	7	call	
A	AX	cj	2	conditional jump (not taken)	
			4	conditional jump (taken)	
B	BX	ajw	1	adjust workspace	
C	CX	eqc	2	equals constant	
D	DX	stl	1	store local	
E	EX	stnl	2	store non-local	
F	FX	opr	-	operate	

General Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
00	F0	rev	1	reverse	
3A	23FA	xword	4	extend to word	
56	25F6	cword	5	check word	Error
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	Error
42	24F2	mint	1	minimum integer	

Arithmetic/Logical Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	
32	23F2	not	1	bitwise not	
41	24F1	shl	n+2	shift left	
40	24F0	shr	n+2	shift right	
05	F5	add	1	add	Error
0C	FC	sub	1	subtract	Error
53	25F3	mul	38	multiply	Error
72	27F2	fmul	35	fractional multiply (no rounding)	Error
			40	fractional multiply (rounding)	Error
2C	22FC	div	39	divide	Error
1F	21FF	rem	37	remainder	Error
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product for positive register A	
			m+5	product for negative register A	

Long Arithmetic Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
16	21F6	ladd	2	long add	Error
38	23F8	lsub	2	long subtract	Error
37	23F7	lsum	2	long sum	
4F	24FF	ldiff	2	long diff	
31	23F1	lmul	33	long multiply	
1A	21FA	ldiv	35	long divide	Error
36	23F6	lshl	n+3	long shift left (n<32)	
			n-28	long shift left(n≥32)	
35	23F5	lshr	n+3	long shift right (n<32)	
			n-28	long shift right (n≥32)	
19	21F9	norm	n+5	normalise (n<32)	
			n-26	normalise (n≥32)	
			3	normalise (n=64)	

Indexing/Array Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
02	F2	bsub	1	byte subscript	
0A	FA	wsub	2	word subscript	
81	28F1	wsubdb	3	form double word subscript	
34	23F4	bcnt	2	byte count	
3F	23FF	wcnt	5	word count	
01	F1	lb	5	load byte	
3B	23FB	sb	5	store byte	
4A	24FA	move	2w+8	move message	

Timer Handling Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
22	22F2	ldtimer	2	load timer	
2B	22FB	tin	30	timer input (time future)	Desch
			3	timer input (time past)	Desch
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	Desch
			48	timer alt wait (time future)	Desch
47	24F7	enbt	8	enable timer	
2E	22FE	dist	23	disable timer	

Input/Output Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
07	F7	in	2w+19	input message	Desch
0B	FB	out	2w+19	output message	Desch
0F	FF	outword	23	output word	Desch
0E	FE	outbyte	23	output byte	Desch
12	21F2	resetch	3	reset channel	
43	24F3	alt	2	alt start	
44	24F4	altwt	5	alt wait (channel ready)	Desch
			17	alt wait (channel not ready)	Desch
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Control Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
20	22F0	ret	5	return	
1B	21FB	ldpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	
5A	25FA	dup	1	duplicate top of stack	
06	F6	gcall	3	general call	
21	22F1	lend	10	loop end (loop)	Desch
			5	loop end (exit)	Desch

Scheduling Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
0D	FD	startp	12	start process	Desch
03	F3	endp	13	end process	Desch
39	23F9	runp	10	run process	
15	21F5	stopp	11	stop process	
1E	21FE	ldpri	1	load current priority	

Error Handling Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
13	21F3	csub0	2	check subscript from 0	Error
4D	24FD	ccnt1	3	check count from 1	Error
29	22F9	testerr	2	test error false and clear (no error)	
			3	test error false and clear (error)	Error
10	21F0	seterr	1	set error	Error
55	25F5	stoperr	2	stop on error	Desch
57	25F7	clrhalterr	1	clear halt-on-error	
58	25F8	sethalterr	1	set halt-on-error	
59	25F9	testhalterr	2	test halt-on-error	

Processor Initialisation Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	
18	21F8	sthf	1	store high priority front pointer	
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stlf	1	store low priority front pointer	
17	21F7	stlb	1	store low priority back pointer	
54	25F4	sttimer	1	store timer	

Floating Point Load/Store Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
8E	28FE	fpldnlsn	2	fp load non-local single	
8A	28FA	fpldnldb	3	fp load non-local double	
86	28F6	fpldnlsni	4	fp load non-local indexed single	
82	28F2	fpldnldbi	6	fp load non-local indexed double	
9F	29FF	fpldzerosn	2	load zero single	
A0	2AF0	fpldzerodb	2	load zero double	
AA	2AFA	fpldnladdsn	2+fpadd	fp load non local & add single	FP_Error
A6	2AF6	fpldnladddb	3+fpadd	fp load non local & add double	FP_Error
AC	2AFC	fpldnlmulsn	2+fpmul	fp load non local & multiply single	FP_Error
A8	2AF8	fpldnlmuldb	3+fpmul	fp load non local & multiply double	FP_Error
88	28F8	fpstnlsn	2	fp store non-local single	
84	28F4	fpstnldb	3	fp store non-local double	
9E	29FE	fpstnli32	4	store non-local int32	

Floating Point General Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
AB	2AFB	fpentry	1	floating point unit entry	
A4	2AF4	fprev	1	fp reverse	
A3	2AF3	fpdup	1	fp duplicate	

Floating Point Rounding Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
22	s	fpurn	1	set rounding mode to round nearest	
06	s	fpurz	1	set rounding mode to round zero	
04	s	fpurp	1	set rounding mode to round positive	
05	s	fpurm	1	set rounding mode to round minus	

Floating Point Error Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
83	28F3	fpchkerror	1	check fp error	Error
9C	29FC	fptesterror	2	test fp error false and clear	FP_Error
23	s	fpuseterror	1	set fp error	FP_Error
9C	s	fpuclearerror	1	clear fp error	FP_Error

Floating Point Comparison Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
94	29F4	fpgt	3/6	fp greater than	FP_Error
95	29F5	fpeq	3/5	fp equality	FP_Error
92	29F2	fpordered	3/4	fp orderability	
91	29F1	fpnan	2/3	fp NaN	
93	29F3	fpnotfinite	2/2	fp not finite	
0E	s	fpuchki32	3/4	check in range of type int32	FP_Error
0F	s	fpuchki64	3/4	check in range of type int64	FP_Error

Processor cycles are shown as **Minimum/Maximum** cycles.

Floating Point Conversion Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
07	s	fpur32tor64	3/4	real32 to real64	FP_Error
08	s	fpur64tor32	6/9	real64 to real32	FP_Error
9D	29FD	fprtoi32	7/9	real to int32	FP_Error
96	29F6	fpi32tor32	8/10	int32 to real32	
98	29F8	fpi32tor64	8/10	int32 to real64	
9A	29FA	fpb32tor64	8/8	bit32 to real64	
0D	s	fpunoround	2/2	real64 to real32, no round	
A1	2AF1	fpint	5/6	round to floating integer	FP_Error

Processor cycles are shown as **Typical/Maximum** cycles.

Floating Point Arithmetic Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles		Name	Desch/ Error
			Single	Double		
87	28F7	fpadd	6/9	6/9	fp add	FP_Error
89	28F9	fpsub	6/9	6/9	fp subtract	FP_Error
8B	28FB	fpmul	11/18	18/27	fp multiply	FP_Error
8C	28FC	fpdiv	16/28	31/43	fp divide	FP_Error
0B	s	fpuabs	2/2	2/2	fp absolute	FP_Error
8F	28FF	fpremfirst	36/46	36/46	fp remainder first step	FP_Error
90	29F0	fpremstep	32/36	32/36	fp remainder iteration	
01	s	fpusqrtfirst	27/29	27/29	fp square root first step	FP_Error
02	s	fpusqrtstep	42/42	42/42	fp square root step	
03	s	fpusqrtlast	8/9	8/9	fp square root end	
0A	s	fpuexpinc32	6/9	6/9	multiply by 2^{32}	FP_Error
09	s	fpuexpdec32	6/9	6/9	divide by 2^{32}	FP_Error
12	s	fpumulby2	6/9	6/9	multiply by 2.0	FP_Error
11	s	fpudivby2	6/9	6/9	divide by 2.0	FP_Error

Processor cycles are given for single and double length operations. In each column, figures are shown as **Typical/Maximum** cycles.

Block Move Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
5B	25FB	move2dinit	8	initialise data for 2D block move	
5C	25FC	move2dall	(2p+23)*r	2D block copy	
5D	25FD	move2dnonzero	(2p+23)*r	2D block copy non-zero bytes	
5E	25FE	move2dzero	(2p+23)*r	2D block copy zero bytes	

CRC and Bit Operation Codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	Desch/ Error
74	27F4	crcword	35	calculate crc on word	
75	27F5	crcbyte	11	calculate crc on byte	
76	27F6	bitcnt	b+2	count bits set in word	
77	27F7	bitrevword	36	reverse bits in word	
78	27F8	bitrevnbits	n+4	reverse bottom n bits in byte	

4 Floating Point Unit

The 64 bit floating point unit (FPU) provides single and double length arithmetic to floating point standard ANSI-IEEE 754-1985. It is able to perform floating point arithmetic concurrently with the central processor unit (CPU), sustaining in excess of 2.25 Mflops on a 30 MHz device. All data communication between memory and the FPU occurs under control of the CPU.

The FPU consists of a microcoded computing engine with a three deep floating point evaluation stack for manipulation of floating point numbers. These stack registers are **FA**, **FB** and **FC**, each of which can hold either 32 bit or 64 bit data; an associated flag, set when a floating point value is loaded, indicates which. The stack behaves in a similar manner to the CPU stack (section 3.1).

As with the CPU stack, the FPU stack is not saved when rescheduling (section 3.3) occurs. The FPU can be used in both low and high priority processes. When a high priority process interrupts a low priority one the FPU state is saved inside the FPU. The CPU will service the interrupt immediately on completing its current operation. The high priority process will not start, however, before the FPU has completed its current operation.

Points in an instruction stream where data need to be transferred to or from the FPU are called *synchronisation points*. At a synchronisation point the first processing unit to become ready will wait until the other is ready. The data transfer will then occur and both processors will proceed concurrently again. In order to make full use of concurrency, floating point data source and destination addresses can be calculated by the CPU whilst the FPU is performing operations on a previous set of data. Device performance is thus optimised by minimising the CPU and FPU idle times.

The FPU has been designed to operate on both single length (32 bit) and double length (64 bit) floating point numbers, and returns results which fully conform to the ANSI-IEEE 754-1985 floating point arithmetic standard. Denormalised numbers are fully supported in the hardware. All rounding modes defined by the standard are implemented, with the default being round to nearest.

The basic addition, subtraction, multiplication and division operations are performed by single instructions. However, certain less frequently used floating point instructions are selected by a value in register **A** (when allocating registers, this should be taken into account). A *load constant* instruction *ldc* is used to load register **A**; the *floating point entry* instruction *fpentry* then uses this value to select the floating point operation. This pair of instructions is termed a *selector sequence*.

Names of operations which use *fpentry* begin with *fpu*. A typical usage, returning the absolute value of a floating point number, would be

ldc fpuabs; fpentry;

Since the indirection code for *fpuabs* is **0B**, it would be encoded as

Mnemonic		Function code	Memory code
ldc	fpuabs	#4	#4B
fpentry	(op. code #AB)		#2AFB
is coded as			
pfix	#A	#2	#2A
opr	#B	#F	#FB

The *remainder* and *square root* instructions take considerably longer than other instructions to complete. In order to minimise the interrupt latency period of the transputer they are split up to form instruction sequences. As an example, the instruction sequence for a single length square root is

fpusqrtfirst; fpusqrtstep; fpusqrtstep; fpusqrtlast;

The FPU has its own error flag **FP_Error**. This reflects the state of evaluation within the FPU and is set in circumstances where invalid operations, division by zero or overflow exceptions to the ANSI-IEEE 754-1985 standard would be flagged (section 3.7.3). **FP_Error** is also set if an input to a floating point operation is infinite or is not a number (NaN). The **FP_Error** flag can be set, tested and cleared without affecting the main **Error** flag, but can also set **Error** when required (sections 3.7.2). Depending on how a program is compiled, it is possible for both unchecked and fully checked floating point arithmetic to be performed.

Further details on the operation of the FPU can be found in The Transputer Instruction Set - A Compiler Writers' Guide.

Typical Floating Point Operation Times

operation	T800-20 single length	T800-20 double length	T800-30 single length	T800-30 double length
add	350 ns	350 ns	233 ns	233 ns
subtract	350 ns	350 ns	233 ns	233 ns
multiply	550 ns	1000 ns	367 ns	667 ns
divide	850 ns	1600 ns	567 ns	1067 ns

Timing is for operations where both operands are normalised fp numbers

5 System Services

System services include all the necessary logic to initialise and sustain operation of the transputer. They also include error handling and analysis facilities.

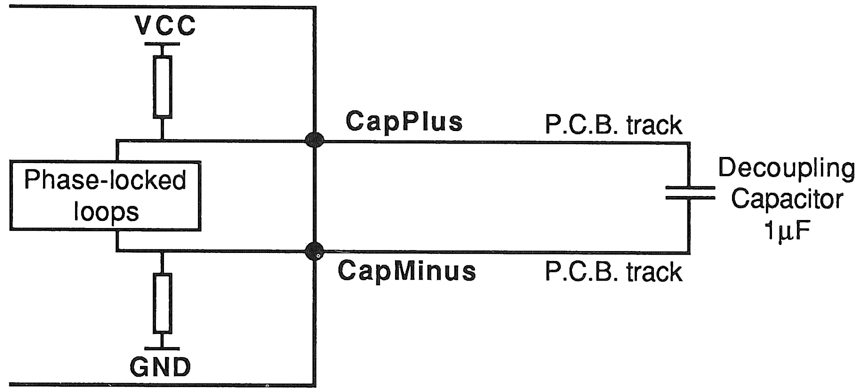
5.1 Power

Power is supplied to the transputer via the **VCC** and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100nF low inductance (e.g. ceramic) capacitor between **VCC** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to **VCC** and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

5.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance 1μF capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50mm. The connections must not touch power supplies or other noise sources.



Recommended PLL Decoupling

5.3 ClockIn

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5MHz, regardless of word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. **ClockIn** must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. **ClockIn** must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

5.4 ProcSpeedSelect0-2

Processor speed of the IMS T800 is variable in discrete steps. The desired speed can be selected, up to the maximum rated for a particular component, by the three speed select lines **ProcSpeedSelect0-2**. The pins are tied high or low, according to the table below, for the various speeds. The **ProcSpeedSelect0-2** pins are designated **HoldToGND** on the IMS T414, and coding is so arranged that the IMS T800 can be plugged directly into a board designed for a 20MHz IMS T414.

Only six of the possible speed select combinations are currently used; the other two are not valid speed selectors. The frequency of **ClockIn** for the speeds given in the table is 5 MHz.

5.5 Reset

Reset can go high with **VCC**, but must at no time exceed the maximum specified voltage for **VIH**. After **VCC** is valid **ClockIn** should be running for a minimum period **TDCVRL** before the end of **Reset**. The falling edge of **Reset** initialises the transputer, triggers the memory configuration sequence and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and **EventReq** should be held low. Memory request (DMA) must not occur whilst **Reset** is high but can occur before boot (section 7.13).

After the end of **Reset** there will be a delay of 144 periods of **ClockIn** (Post-Reset Sequence diagram). Following this, the **MemWrD0**, **MemRfD1** and **MemAD2-31** pins will be scanned to check for the existence of a pre-programmed memory interface configuration (section 7.10.1). This lasts for a further 144 periods of **ClockIn**. Regardless of whether a configuration was found, 36 configuration read cycles will then be performed on external memory using the default memory configuration (section 7.10.2), in an attempt to access the external configuration ROM. A delay will then occur, its period depending on the actual configuration. Finally eight complete and consecutive refresh cycles will initialise any dynamic RAM, using the new memory configuration. If the memory configuration does not enable refresh of dynamic RAM the refresh cycles will be replaced by an equivalent delay with no external memory activity.

If **BootFromRom** is high bootstrapping will then take place immediately, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

Input Clock

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TDCLDCH	ClockIn pulse width low	40			ns	
TDCHDCL	ClockIn pulse width high	40			ns	
TDCLDCL	ClockIn period		200		ns	1,3
TDCerror	ClockIn timing error			±0.5	ns	2
TDC1DC2	Difference in ClockIn for 2 linked devices			400	ppm	3
TDCr	ClockIn rise time			10	ns	4
TDCf	ClockIn fall time			8	ns	4

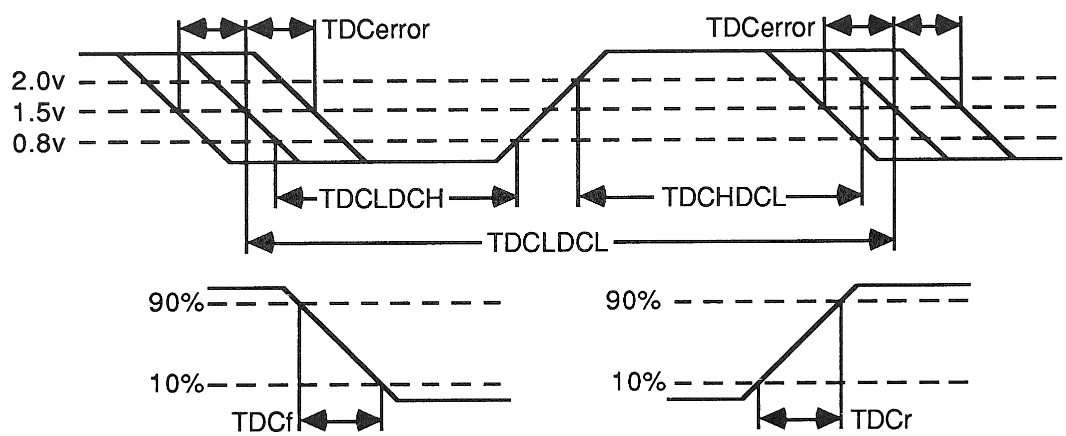
Notes

1 Measured between corresponding points on consecutive falling edges.

2 Variation of individual falling edges from their nominal times.

3 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.

4 Clock transitions must be monotonic within the range **VIH** to **VIL** (section 10.1).

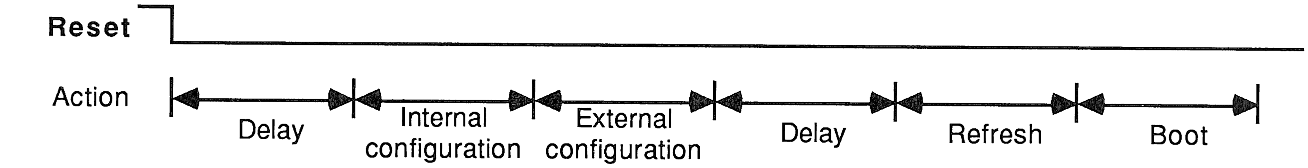


ClockIn Timing

Processor Speed Selection

Proc Speed Select2	Proc Speed Select1	Proc Speed Select0	processor clock speed MHz	processor cycle time nS	notes
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			Invalid
1	1	0	17.5	57.1	
1	1	1			Invalid

Note: Inclusion of a speed selection in this table does not imply immediate availability.



Post-Reset Sequence

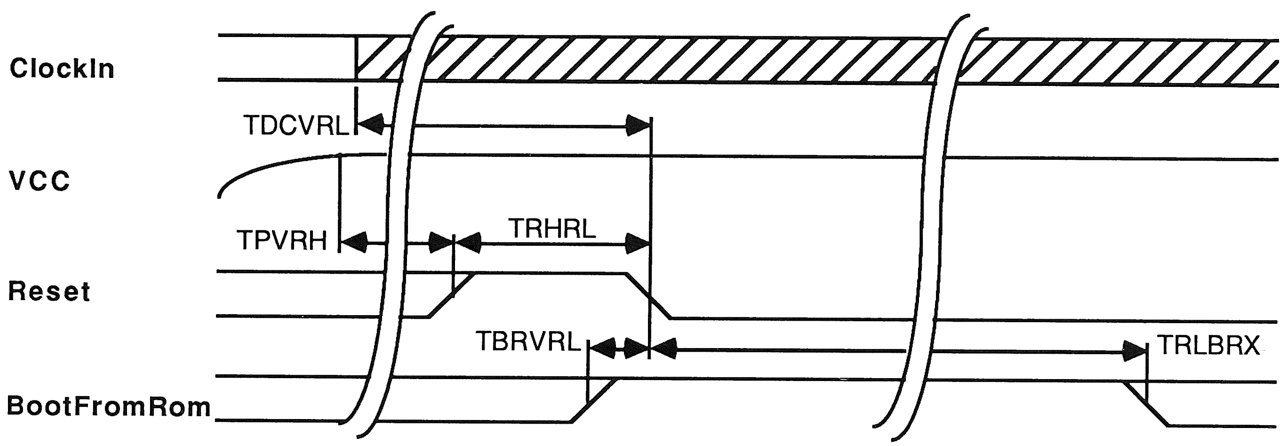
Reset, Analyse

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			ClockIn	1
TDCVRL	ClockIn running before Reset end	10			ms	2
TAHRH	Analyse setup before Reset	3			ms	
TRLAL	Analyse hold after Reset end	1			ns	
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	50			ms	
TALBRX	BootFromRom hold after Analyse	50			ms	

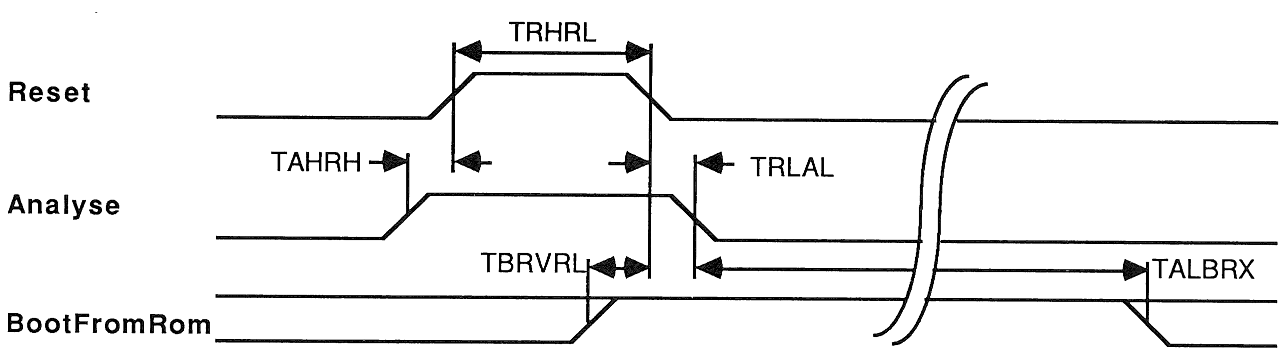
Notes

1 Full periods of **ClockIn TDCLDCL** required.

2 At power-on reset.



Reset Timing with Analyse Low



Reset and Analyse Timing

5.6 Boot

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **BootFromRom** may be dynamically changed, but must obey the specified timing restrictions.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address #7FFFFFFE. This location should contain a backward jump to a program in ROM. The processor is in the low priority state. The **W** register points to **MemStart** (section 6).

If **BootFromRom** is connected low (e.g. to **GND**) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles **TPCLPCL** after **Reset** goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location **MemStart**. Following reception of the last byte the transputer will start executing code at **MemStart** as a low priority process. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the booting link after the last bootstrap byte will be retained until a process inputs from them.

5.7 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a boot from link. If the control byte is 0 then eight more bytes are expected on the same link. The first four byte word is taken as an internal or external memory address at which to poke (write) the second four byte word. If the control byte is 1 the next four bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its boot code. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

5.8 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (section 3.7.1). From **Analyse** being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Memory refresh continues.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

Reset should not be asserted before the transputer has halted and link transfers have ceased. When **Reset** is taken low whilst **Analyse** is high, neither the memory configuration sequence nor the block of eight refresh cycles will occur; the previous memory configuration will be used for any external memory accesses. If **BootFromRom** is high the transputer will boot as soon as **Analyse** is taken low, otherwise it will await a control byte on any link.

If **Analyse** is taken low without **Reset** going high the transputer state and operation are undefined.

After the end of a valid **Analyse** sequence the registers have the following values:

- **I** **MemStart** if booting from a link, or the external memory boot vector if booting from ROM.
- **W** **MemStart** if booting from ROM, or the address of the first free word after the boot code if booting from link.
- **A** The value of **I** when the processor halted.
- **B** The value of **W** when the processor halted, together with the priority of the process when the transputer was halted (i.e. the **W** descriptor).
- **C** The ID of the booting link if booting from link.

5.9 Error, ErrorIn

The **Error** pin carries the OR'ed ouput of the internal **Error** flag and the **ErrorIn** input. If **Error** is high it indicates either that **ErrorIn** is high or that an error was detected in one of the processes. An internal error can be caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (section 3.7.2). It can also be set from the floating point unit under certain circumstances (sections 3.7.3 and 4).

A process can be programmed to stop if **Error** is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data. **ErrorIn** does not directly affect the status of a processor in any way.

By setting the **HaltOnError** flag the transputer itself can be programmed to halt if **Error** becomes set. If **Error** becomes set after **HaltOnError** has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting **HaltOnError** after **Error** will not cause the transputer to halt; this allows **Reset** and **Analyse** to function with the flags in indeterminate states.

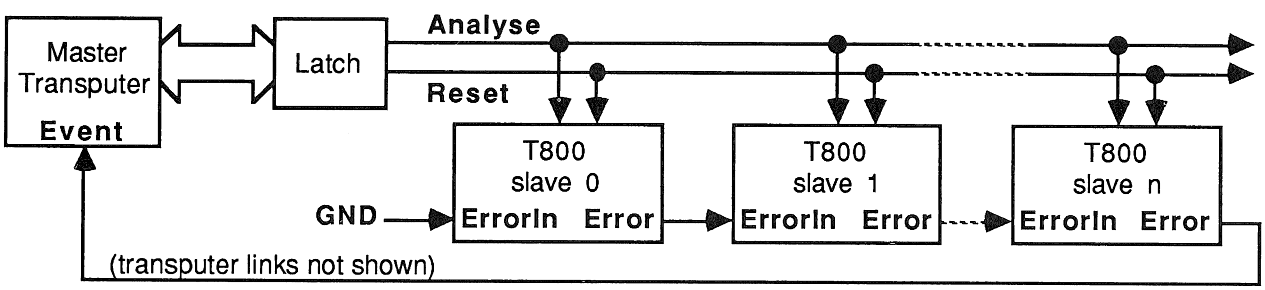
An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by 'daisy-chaining' the **ErrorIn** and **Error** pins of a number of processors and applying the final **Error** output signal to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the **Analyse** function to debug the fault.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an undefined effect.

If a high priority process pre-empts a low priority one, status of the **Error** and **HaltOnError** flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of both flags is transmitted to the high priority process. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of **HaltOnError**, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue, but outputs will not make another access to memory for data. Memory refresh will continue to take place.

After halting due to **Error** changing from 0 to 1 whilst **HaltOnError** is set, register **I** points two bytes past the instruction which sets **Error**. After halting due to **Analyse** being taken high, register **I** points one byte past the instruction being executed. In both cases **I** will be copied to register **A**.



Error Handling in a Multi-Transputer System

6 Memory

The IMS T800 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (section 7.1). The transputer can also access 4 Gbytes of external memory space. Internal and external memory are part of the same linear address space.

Transputer memory is byte addressed, with words aligned on four-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address #80000000 and extends to #80000FFF. User memory begins at #80000070; this location is given the name **MemStart**.

A reserved area at the bottom of internal memory is used to implement link and event channels.

Two words of memory are reserved for timer use, **TPtrLoc0** for high priority processes and **TPtrLoc1** for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer queue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved **IntSaveLoc** locations when a high priority process pre-empts a low priority one. Other locations are reserved for extended features such as block moves and floating point operations.

External memory space starts at #80001000 and extends up through #00000000 to #7FFFFFFF. Memory configuration data and ROM bootstrapping code must be in the most positive address space, starting at #7FFFFF6C and #7FFFFFFE respectively. Address space immediately below this is conventionally used for ROM based code.

---

hi **Machine Map** lo	Byte address		Word offsets	**Occam Map**
Reset Inst	#7FFFFFFE			
Memory configuration	#7FFFFFF8			
	#7FFFFF6C			
	#0			
	#80001000	- Start of external memory -	#0400	
	#80000070	**MemStart**	**MemStart** #1C	
Reserved for Extended Functions	#8000006C	Note 1		
	#80000048			
EregIntSaveLoc	#80000044			
STATUSIntSaveLoc	#80000040			
CregIntSaveLoc	#8000003C			
BregIntSaveLoc	#80000038			
AregIntSaveLoc	#80000034			
IptrIntSaveLoc	#80000030			
WdescIntSaveLoc	#8000002C			
TPtrLoc1	#80000028			
TPtrLoc0	#80000024			
Event	#80000020		#08	Event
Link 3 Input	#8000001C		#07	Link 3 Input
Link 2 Input	#80000018		#06	Link 2 Input
Link 1 Input	#80000014		#05	Link 1 Input
Link 0 Input	#80000010		#04	Link 0 Input
Link 3 Output	#8000000C		#03	Link 3 Output
Link 2 Output	#80000008		#02	Link 2 Output
Link 1 Output	#80000004		#01	Link 1 Output
Link 0 Output	#80000000	(Base of memory)	#00	Link 0 Output

Memory Map

Notes

1 These locations are used as auxiliary processor registers and should not be manipulated by the user. Like processor registers, their contents may be useful for implementing debugging tools (see **Analyse** section 5.8). For details see The Transputer Instruction Set - A Compiler Writers' Guide.

7 External Memory Interface

The External Memory Interface (EMI) allows access to a 32 bit address space, supporting dynamic and static RAM as well as ROM and EPROM. EMI timing can be configured at **Reset** to cater for most memory types and speeds, and a program is supplied with the Transputer Development System to aid in this configuration.

There are 13 internal configurations which can be selected by a single pin connection (section 7.10.1). If none are suitable the user can configure the interface to specific requirements, as shown in section 7.10.2.

7.1 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from **ClockIn**. Its period is equal to one internal microcode cycle time, and can be derived from the formula

TPCLPCL = TDCLDCL / PLLx

where **TPCLPCL** is the **ProcClockOut Period**, **TDCLDCL** is the **ClockIn Period** and **PLLx** is the phase lock loop factor for the relevant speed part, obtained from the ordering details (section 12).

The time value **Tm** is used to define the duration of **Tstates** and, hence, the length of external memory cycles; its value is exactly half the period of one **ProcClockOut** cycle (0.5∗**TPCLPCL**), regardless of mark/space ratio of **ProcClockOut**.

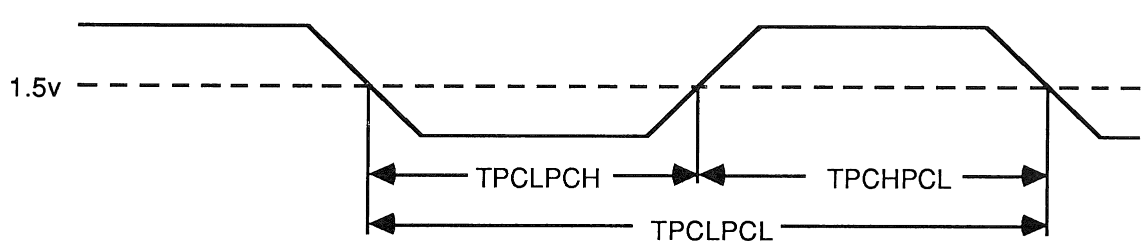
Edges of the various external memory strobes coincide with rising or falling edges of **ProcClockOut**. It should be noted, however, that there is a skew associated with each coincidence. The value of skew depends on whether coincidence occurs when the **ProcClockOut** edge and strobe edge are both rising, when both are falling or if either is rising when the other is falling. Timing values given in the strobe tables show the best and worst cases. If a more accurate timing relationship is required, the exact **Tstate** timing and strobe edge to **ProcClockOut** relationships should be calculated and the correct skew factors applied from the edge skew timing table.

ProcClockOut

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TPCLPCL	ProcClockOut period	a-1	a	a+1	ns	1
TPCHPCL	ProcClockOut pulse width high	b-2.5	b	b+2.5	ns	2
TPCLPCH	ProcClockOut pulse width low		c		ns	3
Tm	ProcClockOut half cycle	b-0.5	b	b+0.5	ns	2
TPCstab	ProcClockOut stability			4000	ppm	4

Notes

1. **a** is **TDCLDCL/PLLx**.
2. **b** is 0.5∗**TPCLPCL** (half the processor clock period).
3. **c** is **TPCLPCL-TPCHPCL**.
4. Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.



ProcClockOut Timing

7.2 Tstates

The external memory cycle is divided into six **Tstates** with the following functions:

T1 Address setup time before address valid strobe
T2 Address hold time after address valid strobe
T3 Read cycle tristate or write cycle data setup
T4 Extendable data setup time
T5 Read or write data
T6 Data hold

Under normal conditions each **Tstate** may be from one to four periods **Tm** long, the duration being set during memory configuration. The default condition on **Reset** is that all **Tstates** are the maximum four periods **Tm** long to allow external initialisation cycles to read slow ROM.

Period **T4** can be extended indefinitely by adding externally generated wait states.

An external memory cycle is always an even number of periods **Tm** in length and the start of **T1** always coincides with a rising edge of **ProcClockOut**. If the total configured quantity of periods **Tm** is an odd number, one extra period **Tm** will be added at the end of **T6** to force the start of the next **T1** to coincide with a rising edge of **ProcClockOut**. This period is designated **E** in configuration diagrams (section 7.10.2).

7.3 Internal access

During an internal memory access cycle the external memory interface bus **MemAD2-31** reflects the word address used to access internal RAM, **MemnotWrD0** reflects the read/write operation and **MemnotRfD1** is high; all control strobes are inactive. This is true unless and until a memory refresh cycle or DMA (memory request) activity takes place, when the bus will carry the appropriate external address or data.

The bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjuction with peek and poke (section 5.7).

7.4 MemAD2-31

External memory addresses and data are multiplexed on one bus. Only the top 30 bits of address are output on the external memory interface, using pins **MemAD2-31**. They are normally output only during **Tstates T1** and **T2**, and should be latched during this time. Byte addressing is carried out internally by the IMS T800 for read cycles. For write cycles the relevant bytes in memory are addressed by the write strobes **notMemWrB0-3**.

The data bus is 32 bits wide. It uses **MemAD2-31** for the top 30 bits and **MemnotRfD1** and **MemnotWrD0** for the lower two bits. Read cycle data may be set up on the bus at any time after the start of **T3**, but must be valid when the IMS T800 reads it at the end of **T5**. Data may be removed any time during **T6**, but must be off the bus no later than the end of that period.

Write data is placed on the bus at the start of **T3** and removed at the end of **T6**. If **T6** is extended to force the next cycle **Tmx** (section 7.8) to start on a rising edge of **ProcClockOut**, data will be valid during this time also.

7.5 MemnotWrD0

During **T1** and **T2** this pin will be low if the cycle is a write cycle, otherwise it will be high. During **Tstates T3** to **T6** it becomes bit 0 of the data bus. In both cases it follows the general timing of **MemAD2-31**.

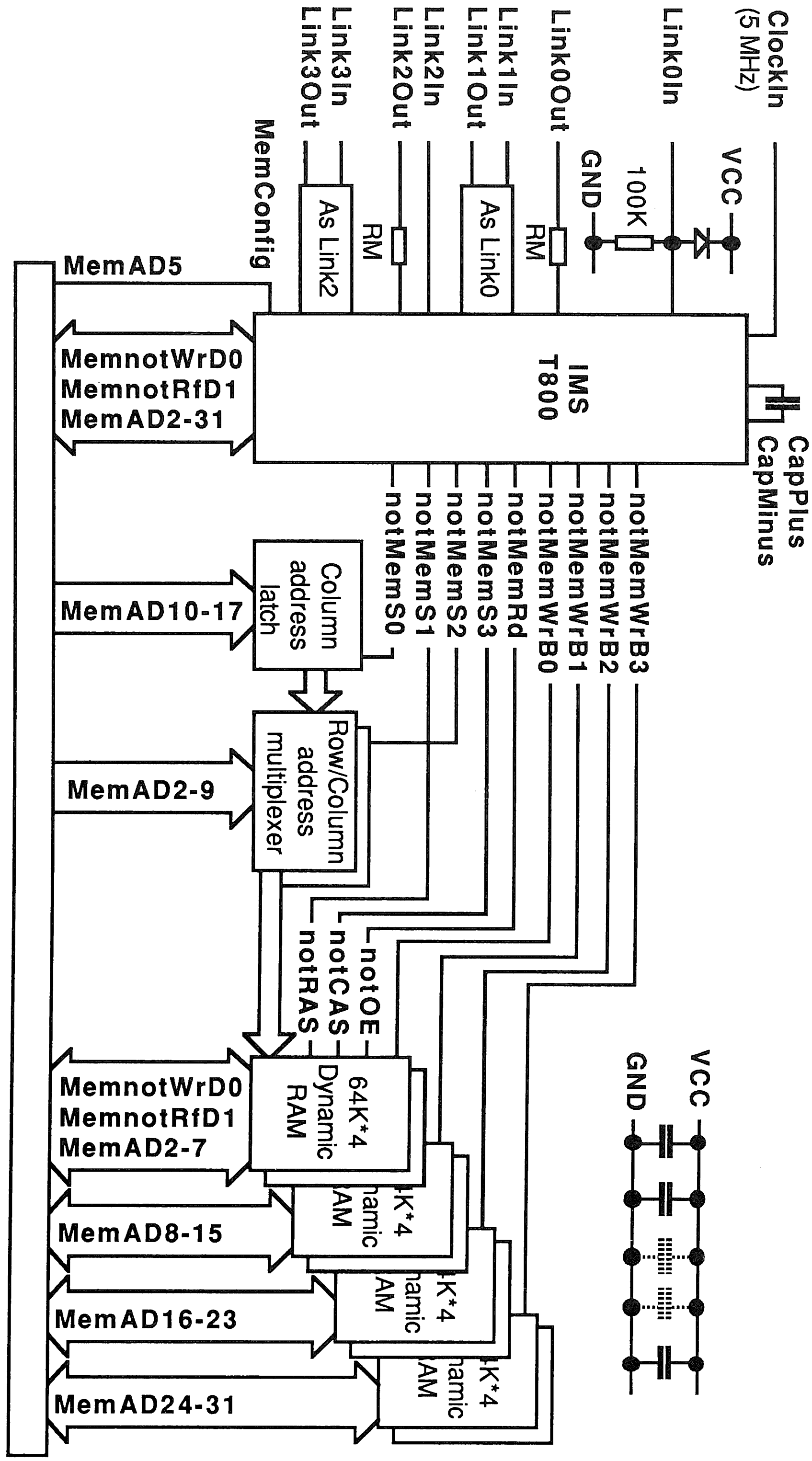
7.6 MemnotRfD1

During **T1** and **T2**, this pin is low if the address on **MemAD2-31** is a refresh address, otherwise it is high. During **Tstates T3** to **T6** it becomes bit 1 of the data bus. In both cases it follows the general timing of **MemAD2-31**.

7.7 notMemRd

For a read cycle the read strobe **notMemRd** is low during **T4** and **T5**. Data is read by the transputer on the rising edge of this strobe, and may be removed immediately afterward. If the strobe duration is insufficient it may be extended by adding extra periods **Tm** to either or both of the **Tstates T4** and **T5**. Further extension may be obtained by inserting wait states at the end of **T4**.

In the read cycle timing diagrams **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**.



Dynamic RAM application

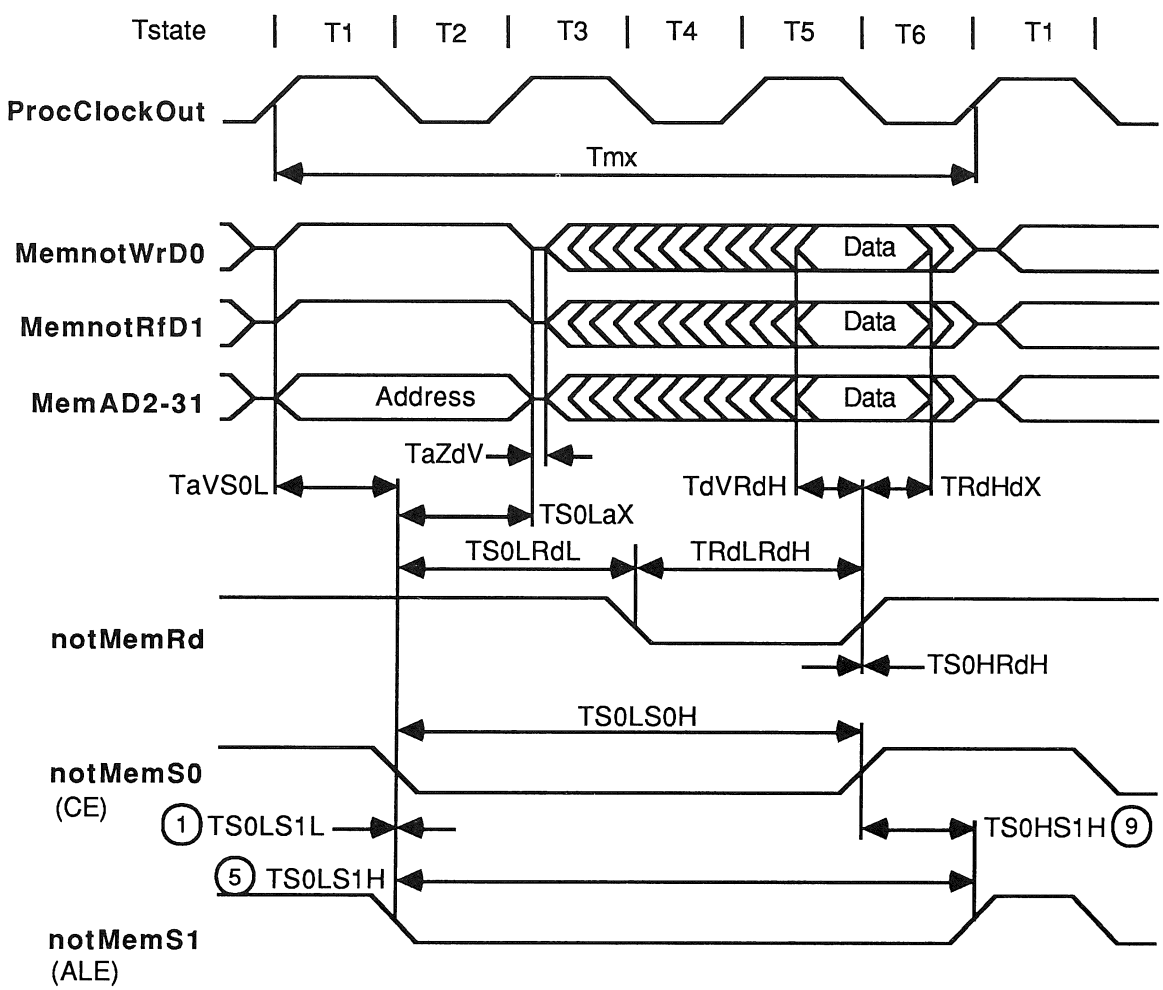
Read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TaZdV	Address tristate to data valid	0			ns	
TdVRdH	Data setup before read	20			ns	
TRdHdX	Data hold after read	0			ns	
TS0LRdL	notMemS0 before start of read	a-2	a	a+2	ns	1
TS0HRdH	End of read from end of notMemS0	-1		1	ns	
TRdLRdH	Read period	b		b+6	ns	2

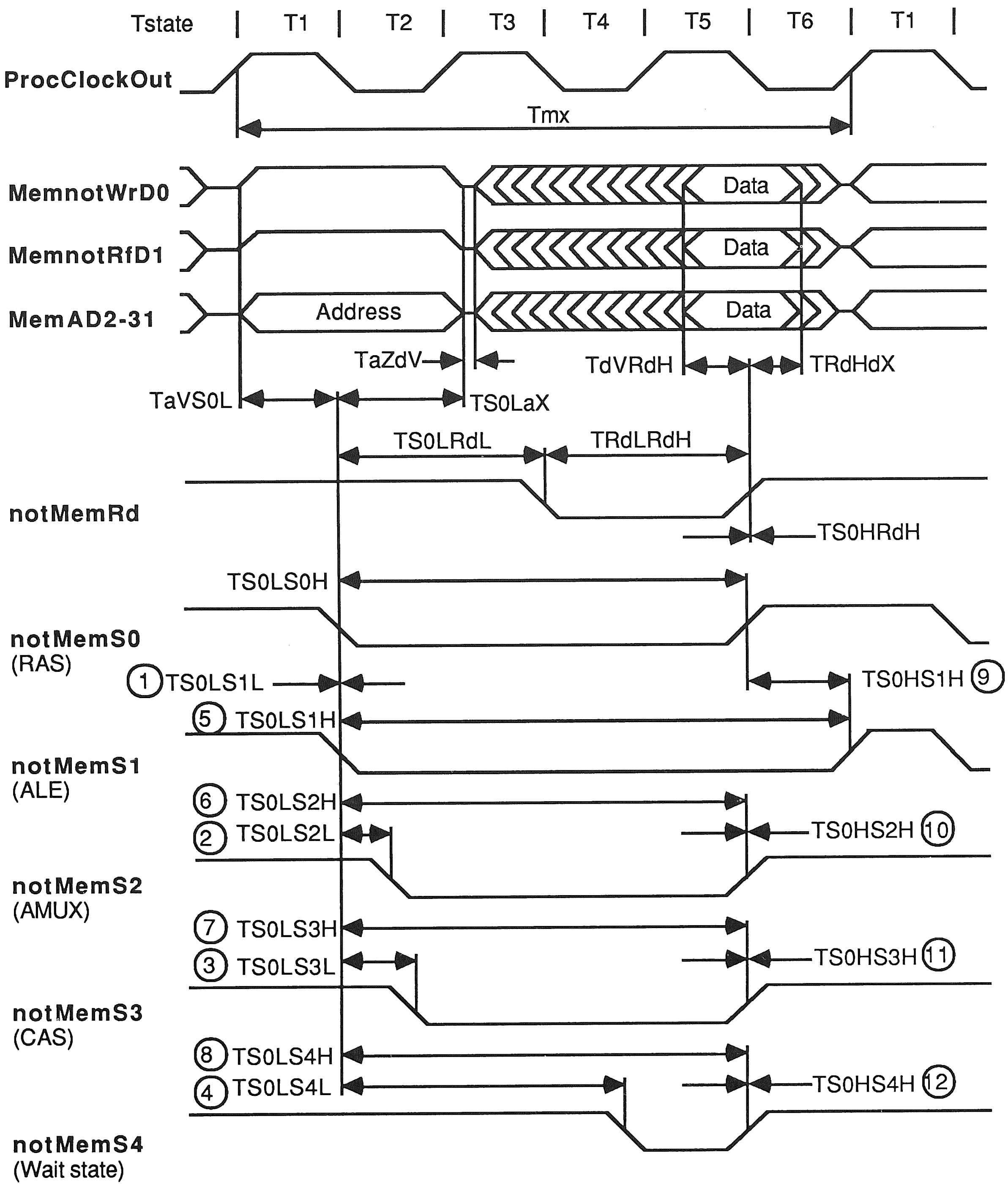
Notes

1 **a** is total of **T2+T3** where **T2**, **T3** can be from one to four periods **Tm** each in length.

2 **b** is total of **T4+Twait+T5** where **T4**, **T5** can be from one to four periods **Tm** each in length and **Twait** may be any number of periods **Tm** in length.



External Read Cycle: Static Memory



External Read Cycle: Dynamic Memory

Strobe Timing

SYMBOL	(n)	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TaVS0L		Address setup before notMemS0		a		ns	1
TS0LaX		Address hold after notMemS0		b		ns	2
TS0LS0H		notMemS0 pulse width low	c		c+6	ns	3
TS0LS1L	1	notMemS1 from notMemS0	0		2	ns	
TS0LS1H	5	notMemS1 end from notMemS0	d		d+6	ns	4,6
TS0HS1H	9	notMemS1 end from notMemS0 end	e-1		e+4	ns	5,6
TS0LS2L	2	notMemS2 delayed after notMemS0	f-1		f+4	ns	7
TS0LS2H	6	notMemS2 end from notMemS0	c+4		c+8	ns	3
TS0HS2H	10	notMemS2 end from notMemS0 end	0		2	ns	
TS0LS3L	3	notMemS3 delayed after notMemS0	f-1		f+3	ns	7
TS0LS3H	7	notMemS3 end from notMemS0	c+4		c+8	ns	3
TS0HS3H	11	notMemS3 end from notMemS0 end	0		2	ns	
TS0LS4L	4	notMemS4 delayed after notMemS0	f-1		f+2	ns	7
TS0LS4H	8	notMemS4 end from notMemS0	c+4		c+8	ns	3
TS0HS4H	12	notMemS4 end from notMemS0 end	0		2	ns	
Tmx		Complete external memory cycle		g			8

Notes

1 **a** is **T1** where **T1** can be from one to four periods **Tm** in length.

2 **b** is **T2** where **T2** can be from one to four periods **Tm** in length.

3 **c** is total of **T2+T3+T4+Twait+T5** where **T2, T3, T4, T5** can be from one to four periods **Tm** each in length and **Twait** may be any number of periods **Tm** in length.

4 **d** can be from zero to 31 periods **Tm** in length.

5 **e** can be from -27 to +4 periods **Tm** in length.

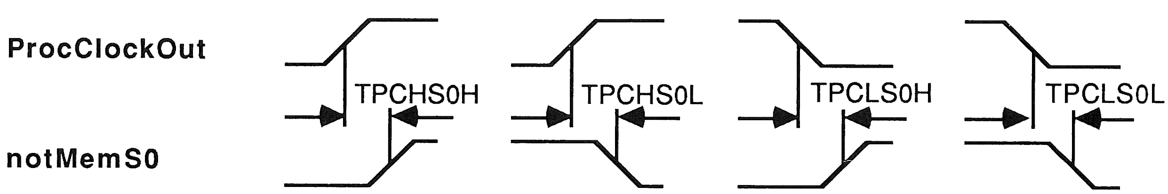
6 If the configuration would cause the strobe to remain active past the end of **T6** it will go high at the end of **T6**. If the strobe is configured to zero periods **Tm** it will remain high throughout the complete cycle **Tmx**.

7 **f** can be from zero to 31 periods **Tm** in length. If this length would cause the strobe to remain active past the end of **T5** it will go high at the end of **T5**. If the strobe value is zero periods **Tm** it will remain low throughout the complete cycle **Tmx.**

8 **g** is one complete external memory cycle comprising the total of **T1+T2+T3+T4+Twait+T5+T6** where **T1, T2, T3, T4, T5** can be from one to four periods **Tm** each in length, **T6** can be from one to five periods **Tm** in length and **Twait** may be zero or any number of periods **Tm** in length.

Strobe S0 to ProcClockOut Skew

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TPCHS0H	Strobe rising from ProcClockOut rising	0		3	nS	
TPCLS0H	Strobe rising from ProcClockOut falling	1		4	nS	
TPCHS0L	Strobe falling from ProcClockOut rising	-3		0	nS	
TPCLS0L	Strobe falling from ProcClockOut falling	-1		2	nS	



Skew of notMemS0 to ProcClockOut

7.8 notMemS0-4

To facilitate control of different types of memory and devices, the EMI is provided with five strobe outputs, four of which can be configured by the user. The strobes are conventionally assigned the functions shown in the read and write cycle diagrams, although there is no compulsion to retain these designations.

notMemS0 is a fixed format strobe. Its leading edge is always coincident with the start of **T2** and its trailing edge always coincident with the end of **T5**.

The leading edge of **notMemS1** is always coincident with the start of **T2**, but its duration may be configured to be from zero to 31 periods **Tm**. Regardless of the configured duration, the strobe will terminate no later than the end of **T6**. The strobe is sometimes programmed to extend beyond the normal end of **Tmx**. When wait states are inserted into an EMI cycle the end of **Tmx** is delayed, but the potential active duration of the strobe is not altered. Thus the strobe can be configured to terminate relatively early under certain conditions (section 7.12). If **notMemS1** is configured to be zero it will never go low.

notMemS2, **notMemS3** and **notMemS4** are identical in operation. They all terminate at the end of **T5**, but the start of each can be delayed from one to 31 periods **Tm** beyond the start of **T2**. If the duration of one of these strobes would take it past the end of **T5** it will stay high. This can be used to cause a strobe to become active only when wait states are inserted. If one of these strobes is configured to zero it will never go high. One of the diagrams shows the effect of **Wait** on strobes in more detail; each division on the scale is one period **Tm**.

7.9 notMemWrB0-3

Because the IMS T800 uses word addressing, four write strobes are provided; one to write each byte of the word. **notMemWrB0** addresses the least significant byte.

The IMS T800 has both early and late write cycle modes. For a late write cycle the relevant write strobes **notMemWrB0-3** are low during **T4** and **T5**; for an early write they are also low during **T3**. Data should be latched into memory on the rising edge of the strobes in both cases, although it is valid until the end of **T6**. If the strobe duration is insufficient, it may be extended at configuration time by adding extra periods **Tm** to either or both of **Tstates T4** and **T5** for both early and late modes. For an early cycle they may also be added to **T3**. Further extension may be obtained by inserting wait states at the end of **T4**. If the data hold time is insufficient, extra periods **Tm** may be added to **T6** to extend it.

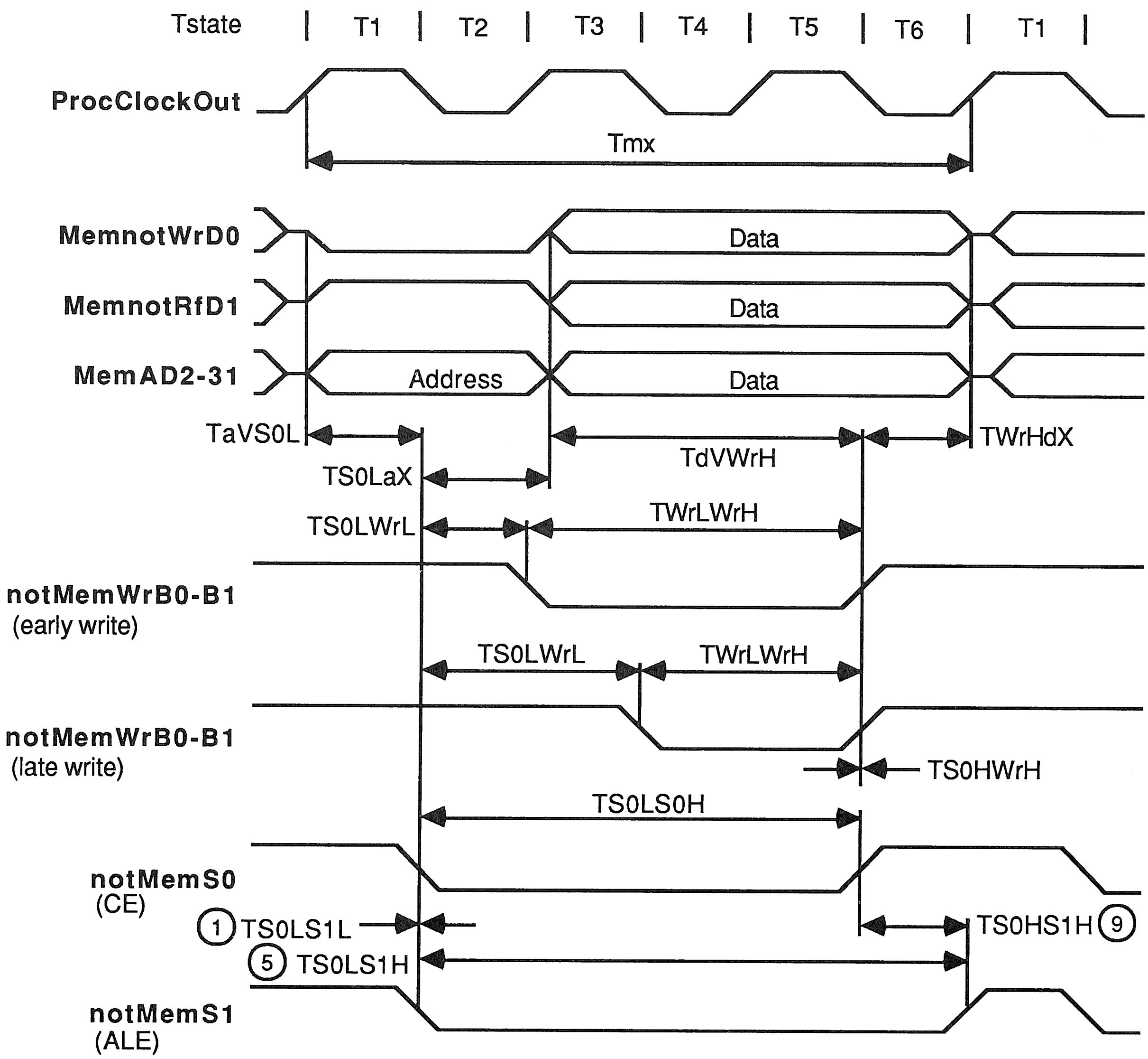
In the write cycle timing diagram **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**. The strobe is inactive during internal memory cycles.

Write

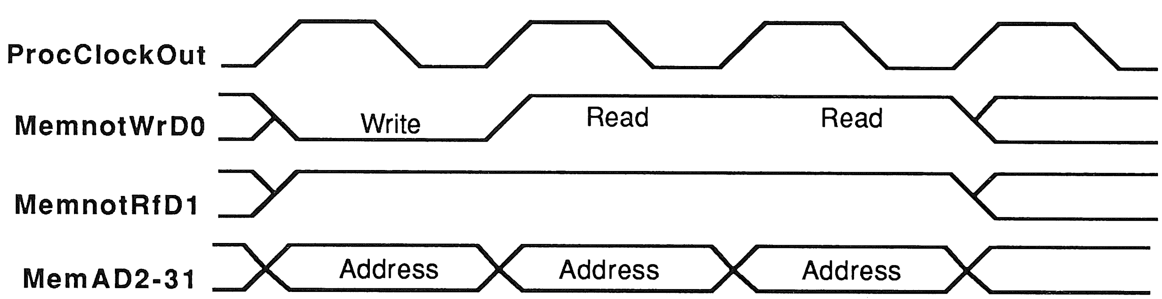
SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TdVWrH	Data setup before write	d			ns	1,5
TWrHdX	Data hold after write	a			ns	1,2
TS0LWrL	notMemS0 before start of early write	b-3		b+2	ns	1,3
	notMemS0 before start of late write	c-3		c+2	ns	1,4
TS0HWrH	End of write from end of notMemS0	-2		2	ns	1
TWrLWrH	Early write pulse width	d		d+6	ns	1,5
	Late write pulse width	e		e+6	ns	1,6

Notes

1. Timing is for all write strobes **notMemWrB0-3**.
2. **a** is **T6** where **T6** can be from one to five periods **Tm** in length.
3. **b** is **T2** where **T2** can be from one to four periods **Tm** in length.
4. **c** is total of **T2+T3** where **T2**, **T3** can be from one to four periods **Tm** each in length.
5. **d** is total of **T3+T4+Twait+T5** where **T3**, **T4**, **T5** can be from one to four periods **Tm** each in length and **Twait** may be zero or any number of periods **Tm** in length.
6. **e** is total of **T4+Twait+T5** where **T4**, **T5** can be from one to four periods **Tm** each in length and **Twait** may be zero or any number of periods **Tm** in length.



External Write Cycle



Bus Activity for Internal Memory Cycle

7.10 MemConfig

MemConfig is an input pin used to read configuration data when setting external memory interface (EMI) characteristics. It is read by the processor on two occasions after **Reset** goes low; first to check if one of the preset internal configurations is required, then to determine a possible external configuration.

7.10.1 Internal configuration

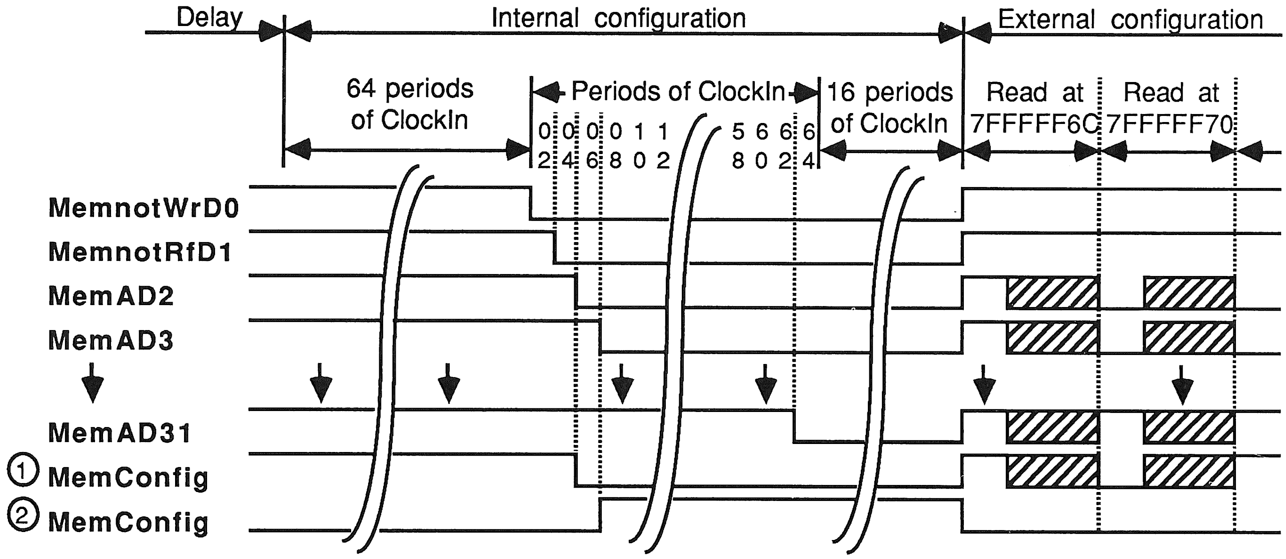
The internal configuration scan comprises 64 periods **TDCLDCL** of **ClockIn** during the internal scan period of 144 **ClockIn** periods. **MemnotWrD0**, **MemnotRfD1** and **MemAD2-32** are all high at the beginning of the scan. Starting with **MemnotWrD0**, each of these lines goes low successively at intervals of two **ClockIn** periods and stays low until the end of the scan. If one of these lines is connected to **MemConfig** the preset internal configuration mode associated with that line will be used as the EMI configuration. The default configuration is that defined in the table for **MemAD31**; connecting **MemConfig** to **VCC** will also produce this default configuration. Note that only 13 of the possible configurations are valid.

Internal Configuration Coding

	Duration of each Tstate periods Tm						Strobe coefficient				Write cycle	Refresh interval	Cycle time	Extra cycles
Pin	T1	T2	T3	T4	T5	T6	s1	s2	s3	s4	type	ClockIn cycles	Proc cycles	e
MemnotWrD0	1	1	1	1	1	1	30	1	3	5	late	72	3	2
MemnotRfD1	1	2	1	1	1	2	30	1	2	7	late	72	4	3
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5	4
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6	5
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3	2
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4	3
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5	4
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6	5
MemAD8	1	1	1	1	1	1	30	1	2	3	early	---	3	2
MemAD9	1	1	2	1	2	1	30	2	5	9	early	---	4	3
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7	6
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9	8
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12	11

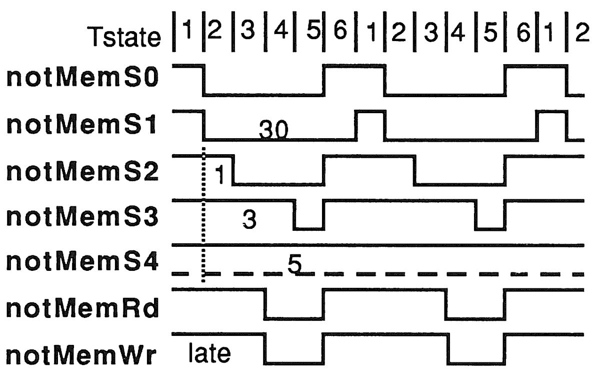
Internal configuration description

Pin	Configuration
MemnotWrD0	Dynamic RAM in 3 processor cycles
MemnotRfD1	Dynamic RAM in 4 processor cycles
MemAD2	Dynamic RAM in 5 processor cycles
MemAD3	Dynamic RAM in 6 cycles
MemAD4	Multiplexed address dynamic RAM in 3 processor cycles
MemAD5	Multiplexed address dynamic RAM in 4 processor cycles
MemAD6	Multiplexed address dynamic RAM in 5 processor cycles
MemAD7	Multiplexed address dynamic RAM in 6 processor cycles
MemAD8	Fast static RAM in 3 processor cycles
MemAD9	Static RAM in 4 cycles with wait generator
MemAD10	General purpose configuration in 7 processor cycles
MemAD11	General purpose configuration in 9 processor cycles
MemAD31	General purpose configuration in 12 processor cycles

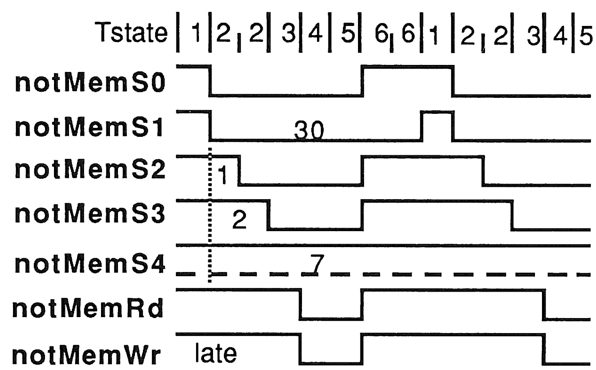


1 Internal configuration: **MemConfig** connected to **MemAD2**.
2 External configuration: **MemConfig** connected to inverse of **MemAD3**.

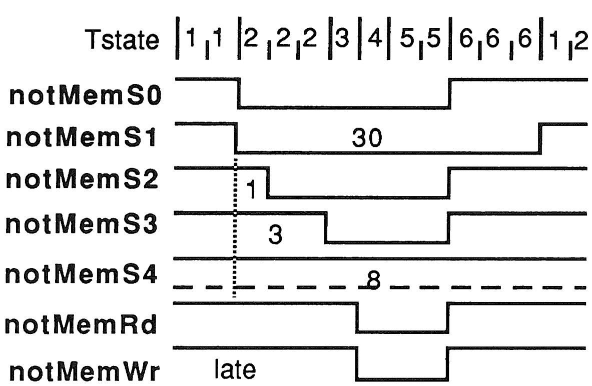
Internal Configuration Scan



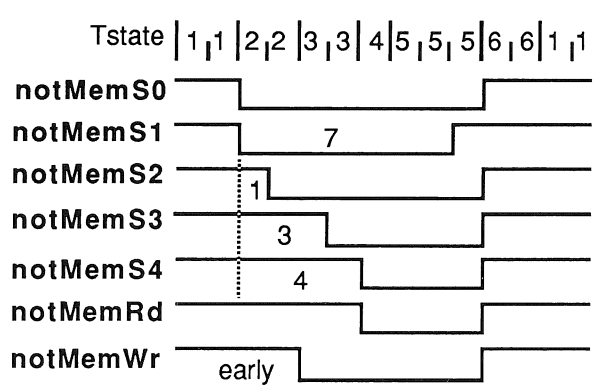
Internal Configuration: MemConfig=MemnotWrD0



Internal Configuration: MemConfig=MemnotRfD1



Internal Configuration: MemConfig=MemAD3



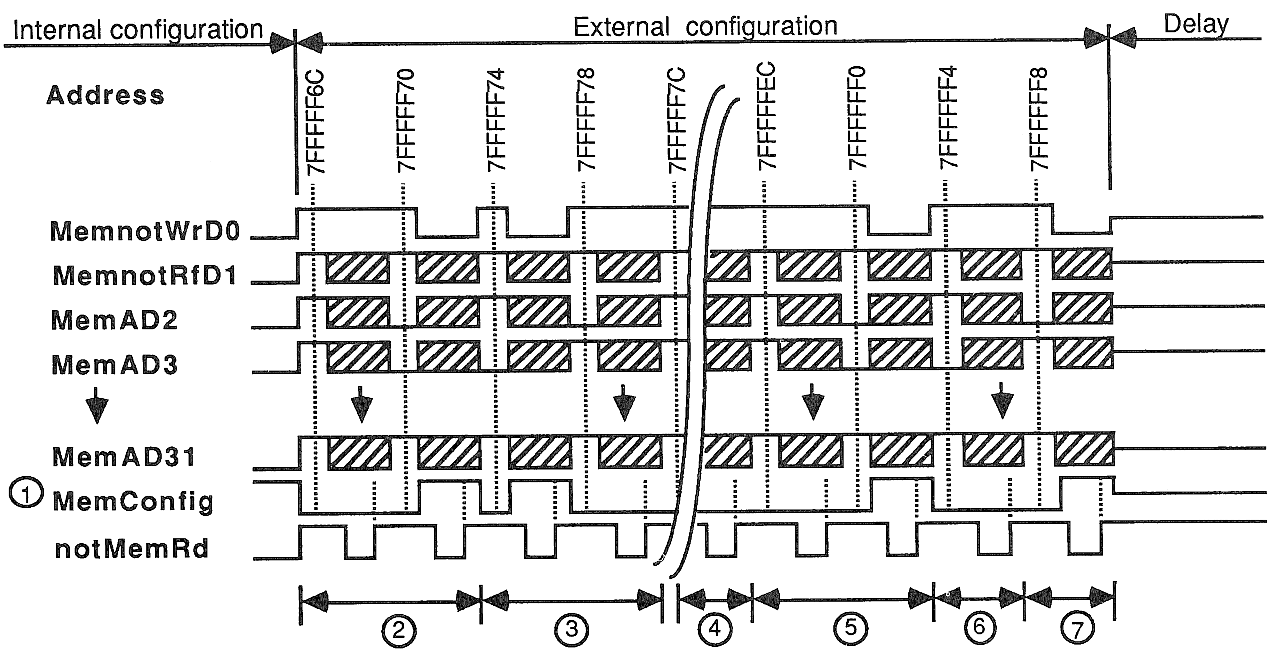
Internal Configuration: MemConfig=MemAD7

7.10.2 External configuration

If **MemConfig** is held low until **MemnotWrD0** goes low the internal configuration is ignored and an external configuration will be loaded instead. An external configuration scan always follows an internal one, but if an internal configuration occurs any external configuration is ignored.

The external configuration scan comprises 36 successive external read cycles, using the default EMI configuration preset by **MemAD31**. However, instead of data being read on the data bus as for a normal read cycle, only a single bit of data is read on **MemConfig** at each cycle. Addresses put out on the bus for each read cycle are shown in the External Configuration Coding table, and are designed to address ROM at the top of the memory map. The table shows the data to be held in ROM; data required at the **MemConfig** pin is the inverse of this.

MemConfig is typically connected via an inverter to **MemnotWrD0**. Data bit zero of the least significant byte of each ROM word then provides the configuration data stream. By switching **MemConfig** between various data bus lines up to 32 configurations can be stored in ROM, one per bit of the data bus. **MemConfig** can be permanently connected to a data line or to **GND**. Connecting **MemConfig** to **GND** gives all **Tstates** configured to four periods; **notMemS1** pulse of maximum duration; **notMemS2-4** delayed by maximum; refresh interval 72 periods of **ClockIn**; refresh enabled; early write.



External Configuration Scan

Notes

1 **MemConfig** connected to inverse of **MemnotWrD0**.

2 Configuration field 1; **T1** configured for 2 periods **Tm**.

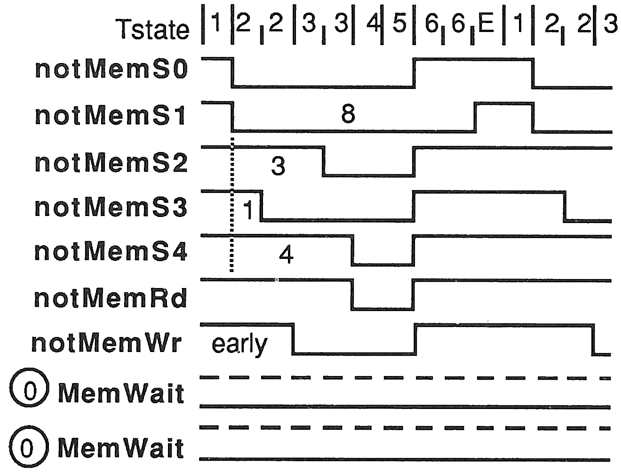
3 Configuration field 2; **T2** configured for 3 periods **Tm**.

4 Configuration field 10; most significant bit of **notMemS4** configured high.

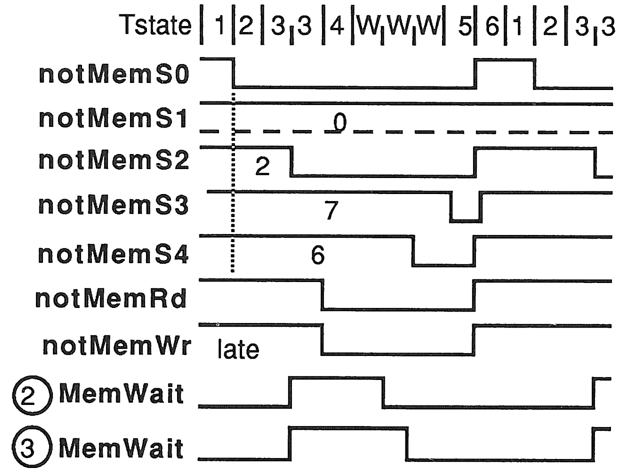
5 Configuration field 11; refresh interval configured for 36 periods **ClockIn**.

6 Configuration field 12; refresh enabled.

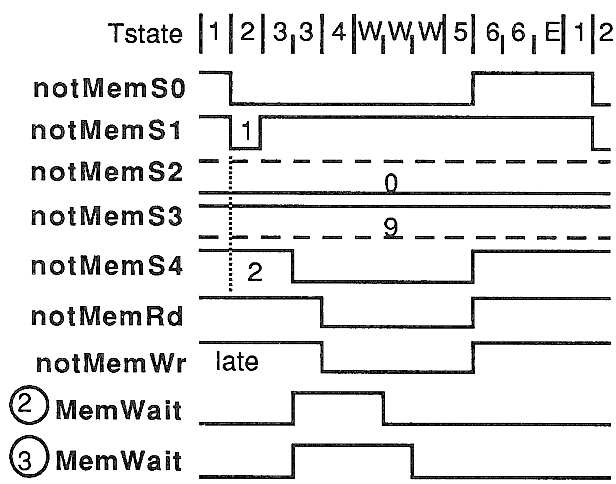
7 Configuration field 13; early write cycle.



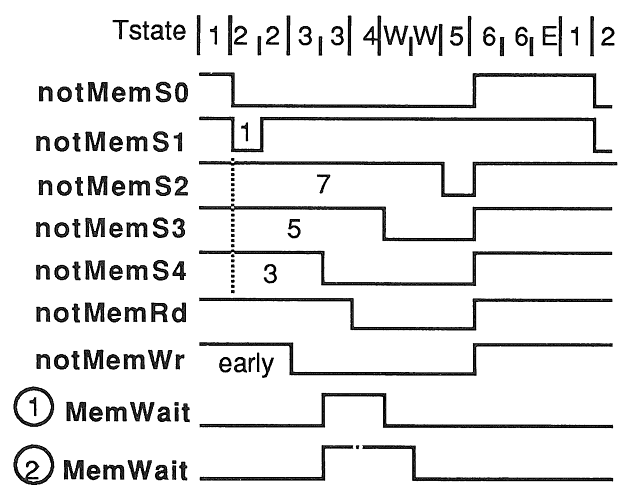
External Configuration: Example 1



External Configuration: Example 2



External Configuration: Example 3



External Configuration: Example 4

Notes

0 No wait states inserted.

1 One wait state inserted.

2 Two wait states inserted.

3 Three wait states inserted.

The external memory configuration table shows the contribution of each memory address to the 13 configuration fields. The lowest 12 words (#7FFFFF6C to #7FFFFF98, fields 1 to 6) define the number of extra periods **Tm** to be added to each **Tstate**. If field 2 is 3 then three extra periods will be added to **T2** to extend it to the maximum of four periods.

The next five addresses (field 7) define the duration of **notMemS1** and the following fifteen (fields 8 to 10) define the delays before strobes **notMemS2-4** become active. The five bits allocated to each strobe allow durations of from 0 to 31 periods **Tm**, as described in strobes section 7.8.

Addresses #7FFFFFEC to #7FFFFFF4 (fields 11 and 12) define the refresh interval and whether refresh is to be used, whilst the final address (field 13) supplies a high bit to **MemConfig** if a late write cycle is required.

The columns to the right of the coding table show the values of each configuration bit for the four sample external configuration diagrams. Note the inclusion of period **E** at the end of **T6** in some diagrams. This is inserted to bring the start of the next **Tstate T1** to coincide with a rising edge of **ProcClockOut** (section 7.1).

Wait states **W** have been added to show the effect of them on strobe timing; they are not part of a configuration. In each case which includes wait states, two wait periods are defined. This shows that if a wait state would cause the start of **T5** to coincide with a falling edge of **ProcClockOut**, another period **Tm** is generated by the EMI to force it to coincide with a rising edge of **ProcClockOut**. This coincidence is only necessary if wait states are added, otherwise coincidence with a falling edge is permitted.

External Configuration Coding

Scan cycle	MemAD address	Field	Function	Example diagram 1	2	3	4
1	7FFFFF6C	1	T1 least significant bit	0	0	0	0
2	7FFFFF70	1	T1 most significant bit	0	0	0	0
3	7FFFFF74	2	T2 least significant bit	1	0	0	1
4	7FFFFF78	2	T2 most significant bit	0	0	0	0
5	7FFFFF7C	3	T3 least significant bit	1	1	1	1
6	7FFFFF80	3	T3 most significant bit	0	0	0	0
7	7FFFFF84	4	T4 least significant bit	0	0	0	0
8	7FFFFF88	4	T4 most significant bit	0	0	0	0
9	7FFFFF8C	5	T5 least significant bit	0	0	0	0
10	7FFFFF90	5	T5 most significant bit	0	0	0	0
11	7FFFFF94	6	T6 least significant bit	1	0	1	1
12	7FFFFF98	6	T6 most significant bit	0	0	0	0
13	7FFFFF9C	7	notMemS1 least significant bit	0	0	1	1
14	7FFFFFA0	7	↓	0	0	0	0
15	7FFFFFA4	7	↓	0	0	0	0
16	7FFFFFA8	7	↓	1	0	0	0
17	7FFFFFAC	7	notMemS1 most significant bit	0	0	0	0
18	7FFFFFB0	8	notMemS2 least significant bit	1	0	0	1
19	7FFFFFB4	8	↓	1	1	0	1
20	7FFFFFB8	8	↓	0	0	0	1
21	7FFFFFBC	8	↓	0	0	0	0
22	7FFFFFC0	8	notMemS2 most significant bit	0	0	0	0
23	7FFFFFC4	9	notMemS3 least significant bit	1	1	1	1
24	7FFFFFC8	9	↓	0	1	0	0
25	7FFFFFCC	9	↓	0	1	0	1
26	7FFFFFD0	9	↓	0	0	1	0
27	7FFFFFD4	9	notMemS3 most significant bit	0	0	0	0
28	7FFFFFD8	10	notMemS4 least significant bit	0	0	0	1
29	7FFFFFDC	10	↓	0	1	1	1
30	7FFFFFE0	10	↓	1	1	0	0
31	7FFFFFE4	10	↓	0	0	0	0
32	7FFFFFE8	10	notMemS4 most significant bit	0	0	0	0
33	7FFFFFEC	11	Refresh Interval least significant bit	-	-	-	-
34	7FFFFFF0	11	Refresh Interval most significant bit	-	-	-	-
35	7FFFFFF4	12	Refresh Enable	-	-	-	-
36	7FFFFFF8	13	Late write	0	1	1	0

Memory refresh configuration coding

Refresh interval	Interval in μs	Field 11 encoding	Complete cycle (mS)
18	3.6	00	0.922
36	7.2	01	1.843
54	10.8	10	2.765
72	14.4	11	3.686

Refresh intervals are in periods of **ClockIn** and **ClockIn** frequency is 5MHz:

Interval = 18 ∗ 200 = 3600ns

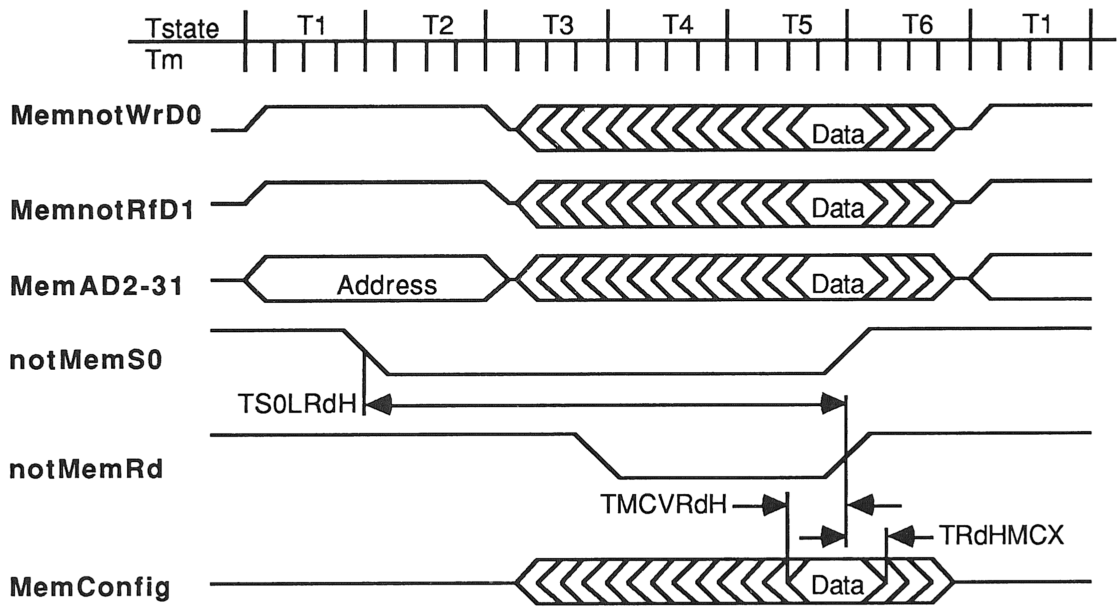
Refresh interval is between successive incremental refresh addresses. Complete cycles are shown for 256 row DRAMS.

Memory Configuration

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TMCVRdH	Memory configuration data setup	20			ns	
TRdHMCX	Memory configuration data hold	0			ns	
TS0LRdH	notMemS0 to configuration data read	a		a+6	ns	1

Notes

1 **a** is 16 periods **Tm**.



External Configuration Read Cycle Timing

7.11 notMemRf

The IMS T800 can be operated with memory refresh enabled or disabled. The selection is made during memory configuration, when the refresh interval is also determined. Refresh cycles do not interrupt internal memory accesses, although the internal addresses cannot be reflected on the external bus during refresh.

When refresh is disabled no refresh cycles occur. During the post-**Reset** period eight dummy refresh cycles will occur with the appropriate timing but with no bus or strobe activity.

A refresh cycle uses the same basic external memory timing as a normal external memory cycle, except that it starts two periods **Tm** before the start of **T1**. If a refresh cycle is due during an external memory access, it will be delayed until the end of that external cycle. Two extra periods **Tm** (periods **R** in the diagram) will then be inserted between the end of **T6** of the external memory cycle and the start of **T1** of the refresh cycle itself. The refresh address and various external strobes become active approximately one period **Tm** before **T1**. Bus signals are active until the end of **T2**, whilst **notMemRf** remains active until the end of **T6**.

For a refresh cycle, **MemnotRfD1** goes low before **notMemRf** goes low and **MemnotWrD0** goes high with the same timing as **MemnotRfD1**. All the address lines share the same timing, but only **MemAD2-11** give the refresh address. **MemAD12-30** stay high during the address period, whilst **MemAD31** remains low. Refresh cycles generate strobes **notMemS0-4** with timing as for a normal external cycle, but **notMemRd** and **notMemWrB0-3** remain high.

7.12 MemWait

Taking **MemWait** high with the timing shown will extend the duration of **T4**. **MemWait** is sampled near to, but independent of, the falling edge of **ProcClockOut**, and should not change state in this region. By convention, **notMemS4** is used to synchronize wait state insertion. If this or another strobe is used, its delay should be such as to take the strobe low an even number of periods **Tm** after the start of **T1**, to coincide with a rising edge of **ProcClockOut**.

MemWait may be kept high indefinitely, although if dynamic memory refresh is used it should not be kept high long enough to interfere with refresh timing.

If the start of **T5** would coincide with a falling edge of **ProcClockOut** an extra wait period **Tm** (**EW**) is generated by the EMI to force coincidence with a rising edge. Rising edge coincidence is only forced if wait states are added, otherwise coincidence with a falling edge is permitted.

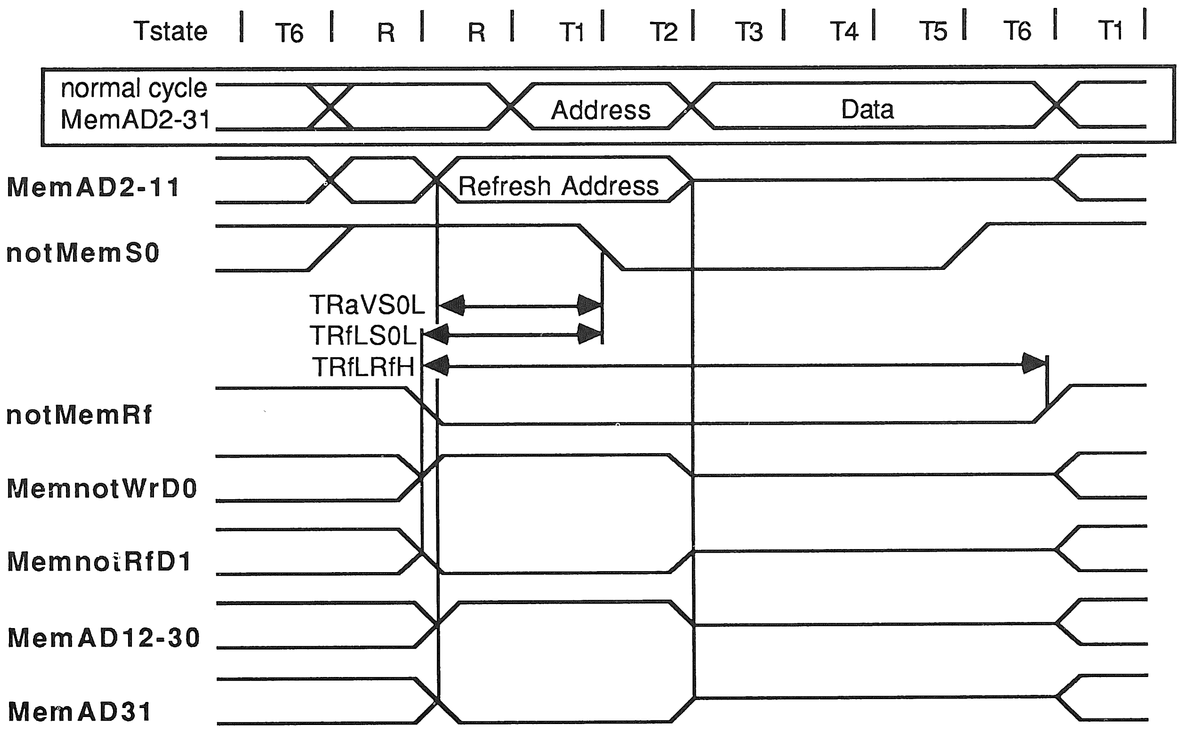
Memory Refresh

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TRfLRfH	Refresh pulse width low	a		a+6	ns	1
TRaVS0L	Refresh address set up before notMemS0		b		ns	2
TRfLS0L	Refresh indicator setup before notMemS0		b		ns	2

Notes

1 **a** is total **Tmx**+(2 periods **Tm**).

2 **b** is total **T1**+(2 periods **Tm**) where **T1** can be from one to four periods **Tm** in length.



Refresh Cycle Timing

Memory Wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TPCHWtH	Wait setup	-(a+3)			ns	1,4
TPCHWtL	Wait hold	b+3			ns	2,3,4
TWtLWtH	Delay before re-assertion of Wait	2			Tm	

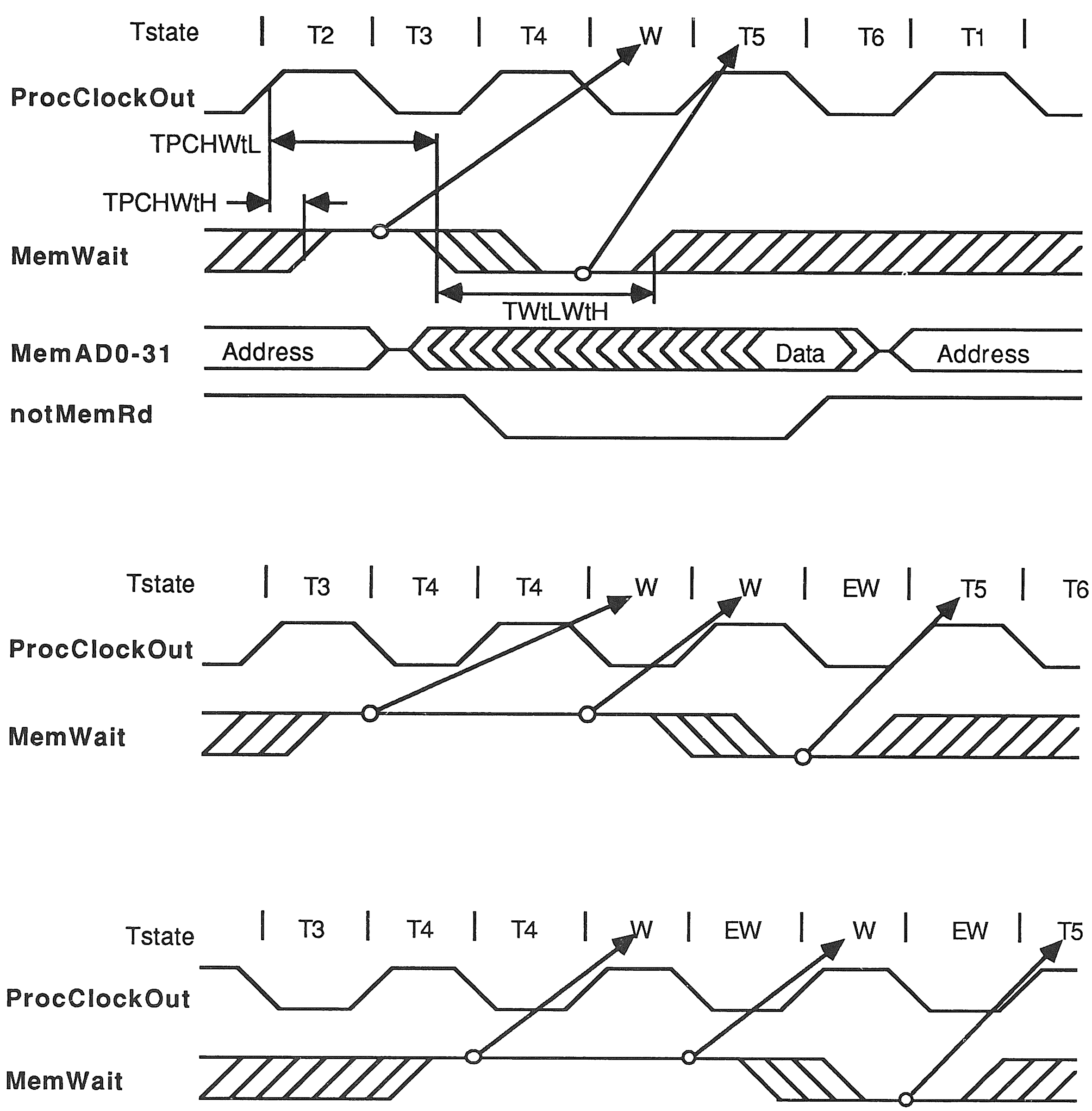
Notes

1 **a** is 0.5 periods **Tm**.

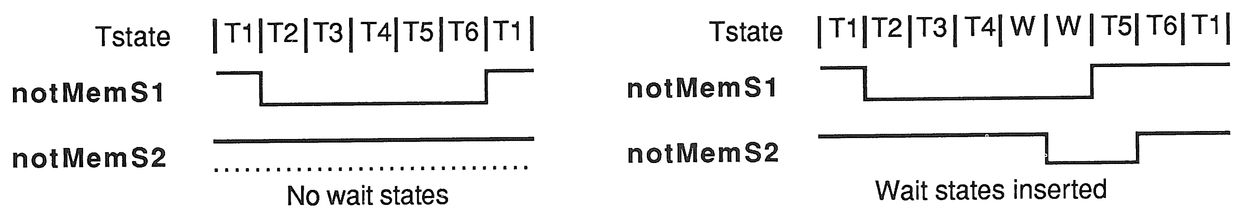
2 **b** is 1.5 periods **Tm**.

3 If wait period exceeds refresh interval, refresh cycles will be lost.

4 Wait timing is independent of falling edge of **ProcClockOut**.



Memory Wait Timing



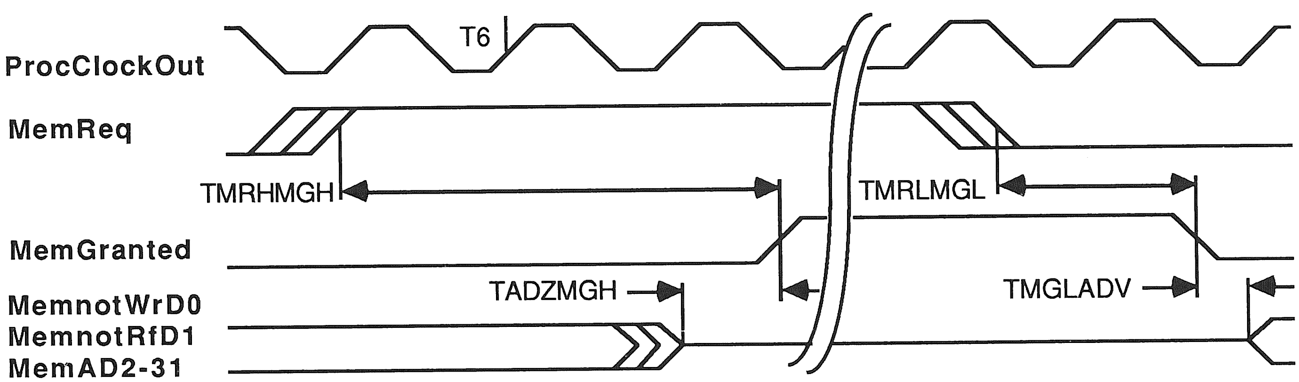
Effect of Wait States on Strobes

Memory Request

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TMRHMGH	Memory request response time	4		6	Tm	1
TMRLMGL	Memory request end response time	2		4	Tm	
TADZMGH	Bus tristate before memory granted		1		Tm	
TMGLADV	Bus active after end of memory granted		1		Tm	

Notes

1 These values assume no external memory cycle is in progress. If an external cycle is active, maximum time could be (1 EMI cycle **Tmx**)+(1 refresh cycle **TRfLRfH**)+(6 periods **Tm**).



Memory Request Timing

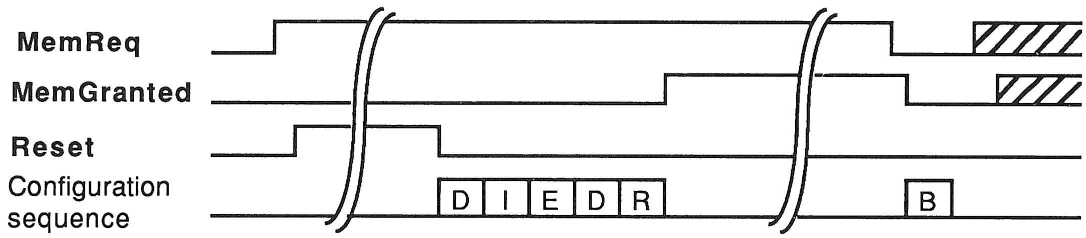
7.13 MemReq, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The IMS T800 samples **MemReq** during the final period **Tm** of **T6** of both refresh and external memory cycles. To guarantee taking over the bus immediately following either, **MemReq** must be set up at least two periods **Tm** before the end of **T6**. In the absence of an external memory cycle, **MemReq** is sampled during every low period of **ProcClockOut**. The address bus is tristated two periods **Tm** after the **ProcClockOut** rising edge which follows the sample. **MemGranted** is asserted one period **Tm** after that.

Removal of **MemReq** is sampled during each low period of **ProcClockOut** and **MemGranted** is removed synchronously with the next falling edge of **ProcClockOut**. If accurate timing of DMA is required, **MemReq** should be set low coincident with a falling edge of **ProcClockOut**. Further external bus activity, either refresh, external cycles or reflection of internal cycles, will commence at the next rising edge of **ProcClockOut**.

Strobes are left in their inactive states during DMA. DMA cannot interrupt a refresh or external memory cycle, and outstanding refresh cycles will occur before the bus is released to DMA. DMA does not interfere with internal memory cycles in any way, although a program running in internal memory would have to wait for the end of DMA before accessing external memory. DMA cannot access internal memory. If DMA extends longer than one refresh interval (Memory Refresh Configuration Coding table, section 7.10.2), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

If **MemReq** is taken high at least one period **TDCLDCL** of **ClockIn** before **Reset** is asserted and remains high during **Reset**, **MemGranted** will be asserted immediately before the boot sequence begins. This allows a boot program to be loaded to external memory. The circuit should be designed to ensure correct operation if **Reset** could interrupt a normal DMA cycle.



DMA sequence at Reset

Notes

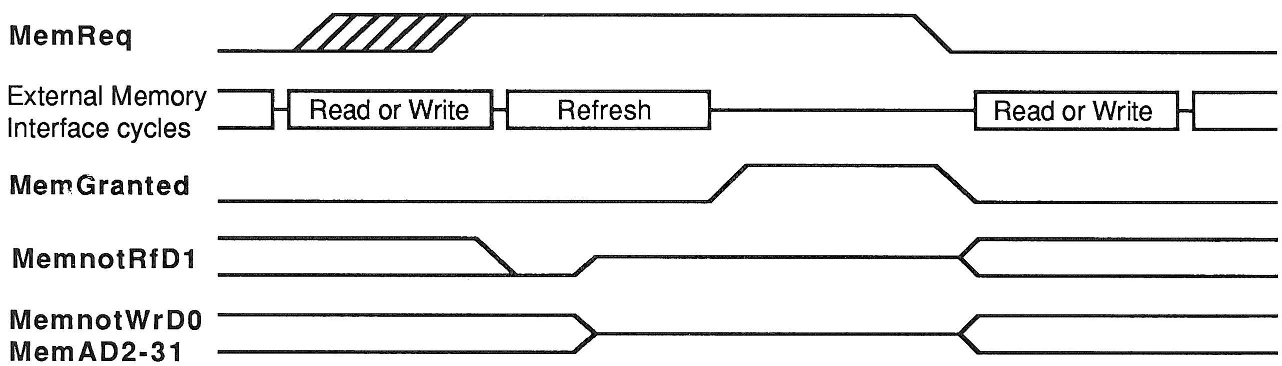
D Pre- and post-configuration delays (see Reset Sequence diagram).

I Internal configuration sequence.

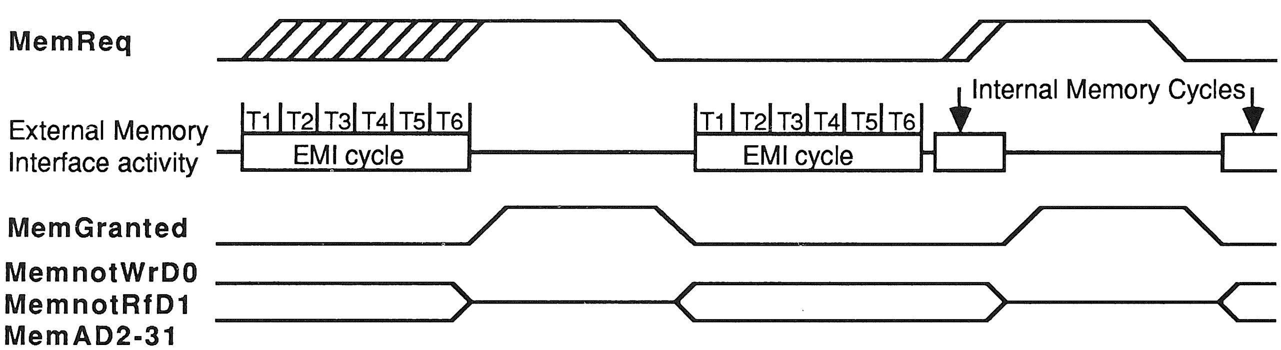
E External configuration sequence.

R Initial refresh sequence.

B Boot sequence.



Operation of MemReq and MemGranted with External and Refresh Memory Cycles



Operation of MemReq and MemGranted with External and Internal Memory Cycles

8 Events

EventReq and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes **EventReq** high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes **EventAck** high and the process, if waiting, is scheduled. **EventAck** is removed after **EventReq** goes low.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

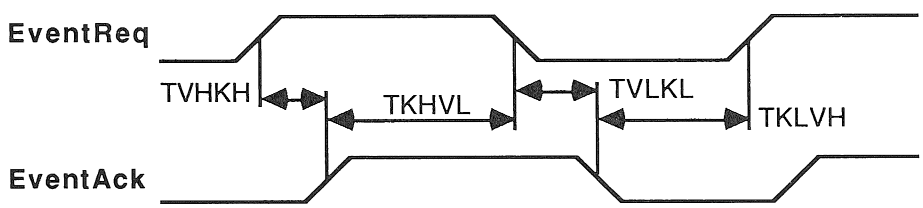
If the process is a high priority one and no other high priority process is running, the maximum latency is as described in section 3.4. Setting a high priority task to wait for an event input is a way of interrupting a transputer program.

Event

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0			ns	
TVLKL	Delay before removal of event acknowledge	0		a	ns	1
TKLVH	Delay before re-assertion of event request	0			ns	

Notes

1 **a** is 2 periods **Tm**.



Event Timing

9 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

Link performance is improved over previous transputers by allowing an acknowledge packet to be sent before the data packet has been fully received. This overlapped acknowledge technique is fully compatible with all other INMOS transputer links.

The IMS T800 links support the standard INMOS communication speed of 10 Mbits per second. In addition they can be used at 5 or 20 Mbits per second. Links are not synchronised with **ClockIn** or **ProcClockOut** and are insensitive to their phases.

Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

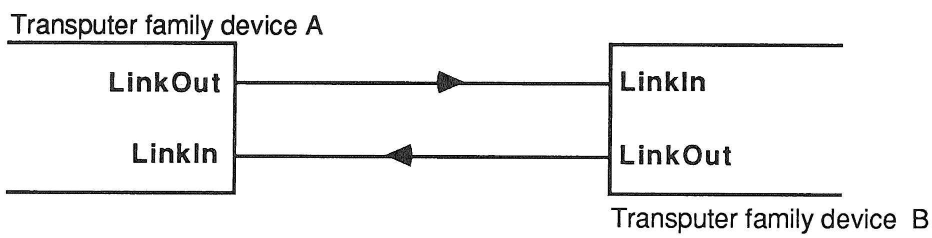
Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 Ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

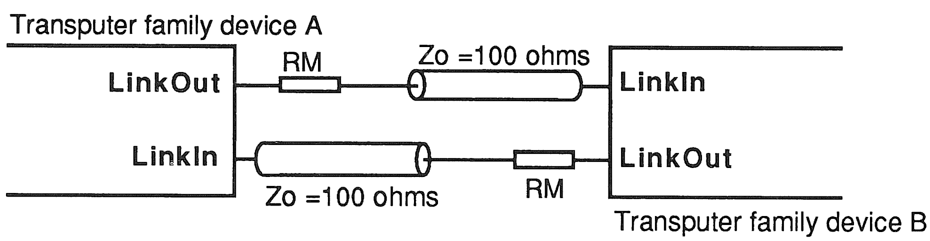
Link speeds can be set by **LinkSpecial**, **Link0Special** and **Link123Special**. The link 0 speed can be set independently. The table shows uni-directional and bi-directional data rates in Kbytes/second for each link speed; **LinknSpecial** is to be read as **Link0Special** when selecting link 0 speed and as **Link123Special** for the others. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Speed settings for Links

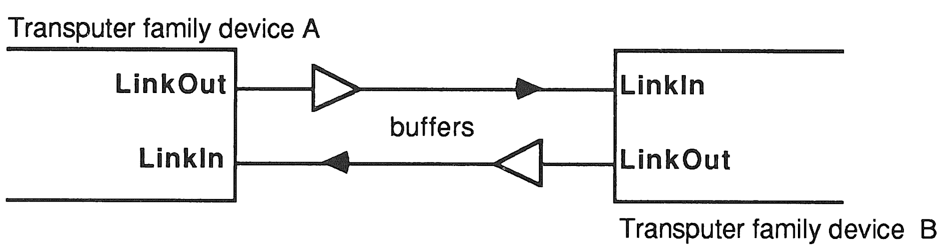
Link Special	Linkn Special	Mbits /sec	Kbytes/sec	
			Uni	Bi
0	0	10	910	1250
0	1	5	450	670
1	0	10	910	1250
1	1	20	1740	2350



Links Directly Connected



Links Connected by Transmission Line



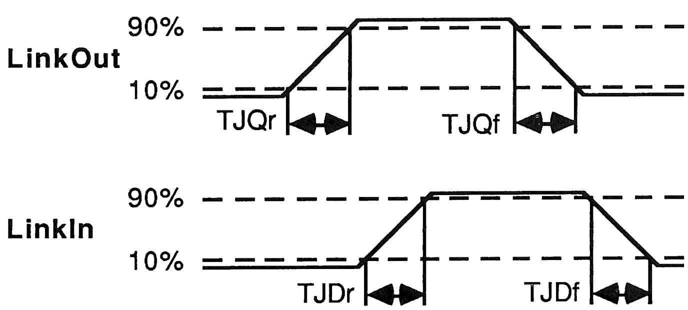
Links Connected by Buffers

Link

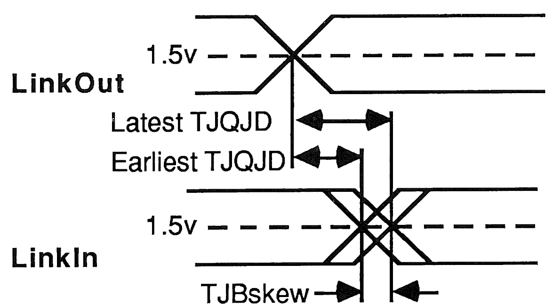
SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TJQr	LinkOut rise time			20	ns	
TJQf	LinkOut fall time			10	ns	
TJDr	LinkIn rise time			20	ns	
TJDf	LinkIn fall time			20	ns	
TJQJD	Buffered edge delay	0			ns	
TJBskew	Variation in TJQJD: 05 Mbits/S			30	ns	1
	10 Mbits/S			10	ns	1
	20 Mbits/S			3	ns	1
CLI Z	LinkIn input capacitance @ f = 1 MHz			7	pF	
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission line		56		ohms	

Notes

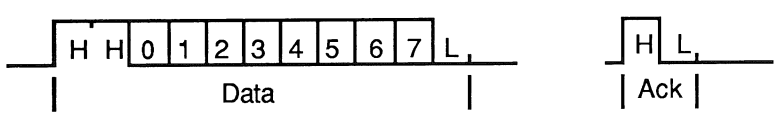
1 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.



Link Timing



Buffered Link Timing



Link Data and Acknowledge Packets

10 Electrical specifications

10.1 DC electrical characteristics

Absolute Maximum Ratings

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
VCC	DC supply voltage	0		7.0	V	1,2,3
VI,VO	Voltage on input and output pins	-0.5		VCC+0.5	V	1,2,3
I I	Input current			±25	mA	4
OSCT	Output short circuit time (one pin)			1	s	2
TS	Storage temperature	-65		150	°C	2
TA	Ambient temperature under bias	-55		125	°C	2
PDmax	Maximum allowable dissipation			2	W	

Notes

1 All voltages are with respect to **GND**.

2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic level such as **VCC** or **GND**.

4 The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VCC**.

Recommended Operating Conditions

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
VCC	DC supply voltage	4.75		5.25	V	1
VI,VO	Input or output voltage	0		VCC	V	1,2
CL	Load capacitance on any pin			50	pF	
TA	Operating temperature range (still air)	0		70	°C	

Notes

1 All voltages are with respect to **GND**.

2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.

DC Characteristics

SYMBOL	PARAMETER		MIN	NOM	MAX	UNITS	NOTES
VIH	High level input voltage		2.0		VCC+0.5	V	1,2
VIL	Low level input voltage		-0.5		0.8	V	1,2
II	Input current	@ GND<VI<VCC			±10	µA	1,2
VOH	Output high voltage	@ IOH=2mA	VCC-1			V	1,2
VOL	Output low voltage	@ IOL=4mA			0.4	V	1,2
IOS	Output short circuit current	@ GND<VO<VCC			50	mA	1,2
IOZ	Tristate output current	@ GND<VI<VCC			±10	µA	1,2
PD	Power dissipation				1.2	W	1,2,3,4
CIN	Input capacitance	@ f=1MHz			7	pF	
COZ	Output capacitance	@ f=1MHz			10	pF	

Notes

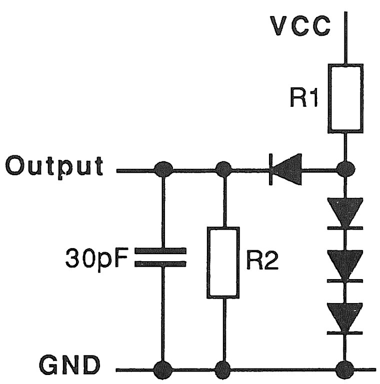
1 All voltages are with respect to **GND**.

2 Parameters measured at 4.75V<**VCC**<5.25V and 0°C<**TA**<70°C. Input clock frequency = 5MHz.

3 Dissipation for processor operating at 20MHz.

4 Power dissipation varies with output loading and with executing program content.

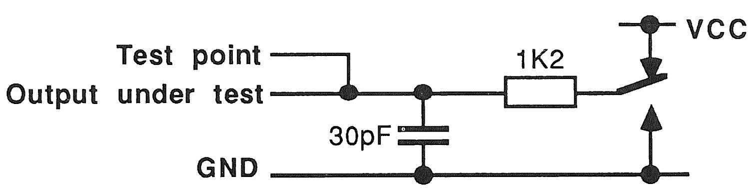
10.2 Equivalent circuits



Load for:	R1	R2	Equivalent load:
Link outputs	1k96	47k	1 Schottky TTL input
Other outputs	970R	24k	2 Schottky TTL inputs

Diodes are 1N916

Load Circuit for AC Measurements



Tristate Load Circuit for AC Measurements

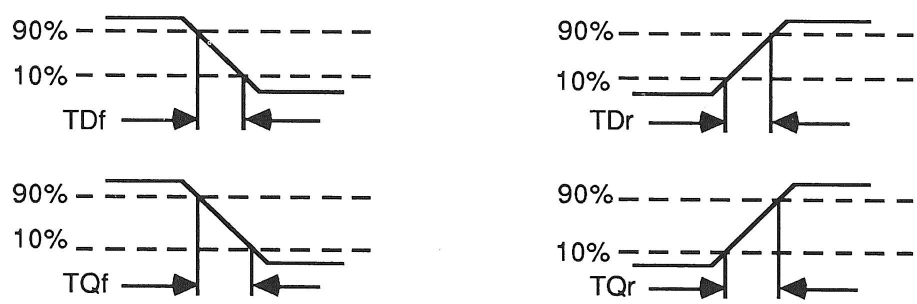
10.3 AC timing characteristics

Input, Output Edges

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
TDr	Input rising edges	2		20	ns	
TDf	Input falling edges	2		20	ns	
TQr	Output rising edges			25	ns	
TQf	Output falling edges			15	ns	
TS0LaHZ	Address high to tristate	a		a+6	ns	1
TS0LaLZ	Address low to tristate	a		a+6	ns	1

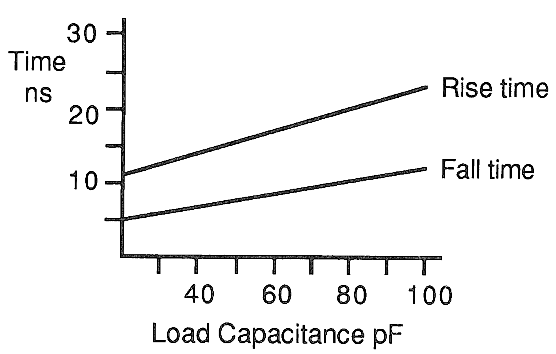
Notes

1 a is **T2** where **T2** can be from one to four periods **Tm** in length. Address lines include **MemnotWrD0**, **MemnotRfD1**, **MemAD2-31**.

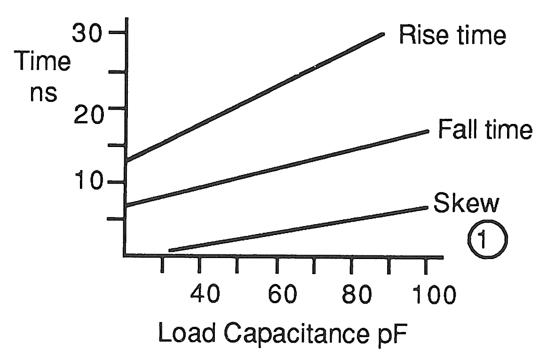


Input and Output Edge Timing

Tristate Timing Relative to notMemS0



Typical Link Rise/Fall Times



Typical EMI Rise/Fall Times

Notes

1 Skew is measured between **notMemS0** with a standard load (2 Schottky TTL inputs and 30pF) and **notMemS0** with a load of 2 Schottky TTL inputs and varying capacitance.

10.4 Power rating

Average junction temperature TJ of the chip in °C is obtained from

$$TJ = TA + (PD * \theta JA) \qquad \textbf{(1)}$$

where

$TA =$ ambient temperature in °C
$\theta JA =$ package junction-to-ambient thermal resistance in °C/W
$PD = PINT + PIO$
$PINT = ICC * VCC$ Watts (internal power)
$PIO =$ power dissipation on input/output pins (application dependant)

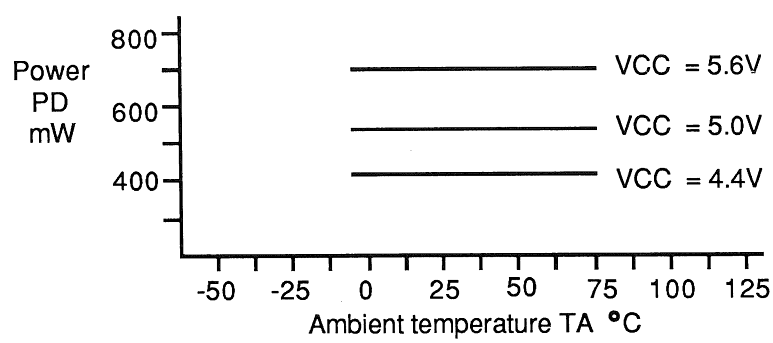
The relationship between PD and TJ with negligible PIO is approximated by

$$PD = K/(TJ + 273) \qquad \textbf{(2)}$$

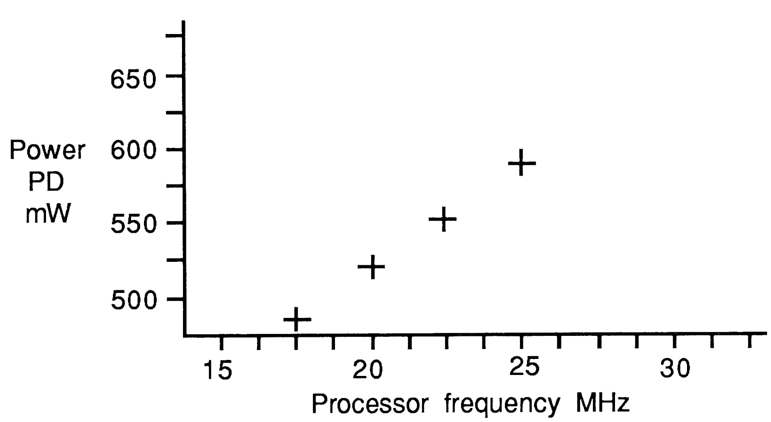
From equations (1) and (2)

$$K = PD * (TA + 273) + \theta JA * PD^2 \qquad \textbf{(3)}$$

where K is a constant for a particular part. K is determined from (3) by measuring PD at equilibrium for a known temperature TA. The values of PD and TJ can be obtained using K to iteratively solve (1) and (2) for any value of TA.



Typical Power Dissipation with Temperature



Typical Power Dissipation with Processor Speed

11 Package specifications

11.1 Pin grid array package

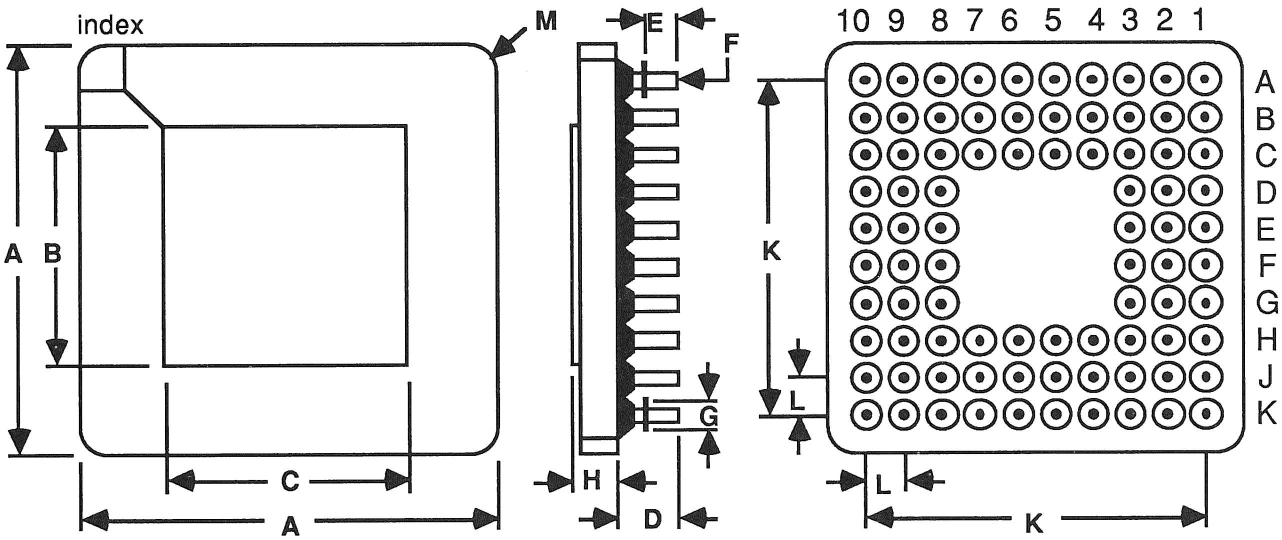
11.1.1 Pin grid array pinout

	1	2	3	4	5	6	7	8	9	10
A	DoNot Wire	Link Special	Proc Clock Out	Link123 Special	Link In0	Link Out1	Link In2	Event Ack	GND	Mem Wait
B	Proc Speed Select0	ClockIn	DoNot Wire	Link0 Special	Link Out0	Link Out2	Link Out3	Event Req	Mem Req	notMem WrB3
C	GND	VCC	Cap Minus	Cap Plus	VCC	Link In1	Link In3	Mem Config	Mem Granted	notMem WrB1
D	Error	Proc Speed Select2	ErrorIn					notMem Rf	notMem WrB2	notMem WrB0
E	HoldTo GND	Boot From ROM	Reset					notMem Rd	notMem S0	VCC
F	Proc Speed Select1	Analyse	Mem AD31					notMem S3	notMem S2	notMem S4
G	Mem AD30	GND	Mem AD27					Memnot WrD0	GND	notMem S1
H	Mem AD29	Mem AD25	Mem AD23	VCC	Mem AD16	Mem AD12	Mem AD8	Mem AD4	Mem AD3	Memnot RfD1
J	Mem AD28	Mem AD24	Mem AD22	Mem AD19	Mem AD17	Mem AD13	GND	Mem AD6	Mem AD5	Mem AD2
K	Mem AD26	Mem AD21	Mem AD20	Mem AD18	Mem AD15	Mem AD14	Mem AD11	Mem AD10	Mem AD9	Mem AD7

Index

IMS T800
84 Pin Grid Array
Top View

11.1.2 Pin grid array dimensions



DIM	Millimetres NOM	Millimetres TOL	Inches NOM	Inches TOL	NOTES
A	26.924	±0.254	1.060	±0.010	
B	17.018	±0.203	0.670	±0.008	
C	18.796	±0.203	0.740	±0.008	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.051	0.018	±0.002	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
H	2.456	±0.278	0.097	±0.011	
K	22.860	±0.254	0.900	±0.010	
L	2.540	±0.127	0.100	±0.005	
M	0.508		0.020		Chamfer

Package weight is approximately 7.2 grams

11.1.3 Pin grid array thermal characteristics

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTES
θJA	Junction to ambient thermal resistance			35	°C/W	1

Notes

1 Measured in still air.

12 Ordering details

The following table indicates the designation of the IMS T800 speed and package selections. Speed of **ClockIn** is 5MHz for all parts. Processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in section 7.1.

INMOS designation	Instruction throughput	Processor clock speed	Processor cycle time	PLLx	Package
IMS T800B-G20S	10 MIPS	20 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T800B-G30S‡	15 MIPS	30 MHz	33 ns	6.0	Ceramic Pin Grid

Notes

‡ For availability contact INMOS.

INMOS REPRESENTATIVES

ALABAMA
Huntsville
Macro-Marketing Associates
205-883-9630

ARIZONA
Phoenix
Compass Marketing & Sales
602-996-0635

Scottsdale
Compass Marketing & Sales
602-505-0202

Tucson
Compass Marketing & Sales
602-293-1220

CALIFORNIA
Santa Clara
Criterion Sales
408-988-6300

San Diego
Hadden Associates
619-565-9444

COLORADO
Wheatridge
Waugaman Associates
303-423-1020

CONNECTICUT
Brookfield
M & M Associates
203-775-6888

FLORIDA
Coral Springs
Graham Associates
305-755-6733

Indiatlantic
Graham Associates
305-773-6631

Palm Beach Gardens
Graham Associates
305-622-4049

Winter Garden
Graham Associates
305-656-9369

GEORGIA
Norcross (Atlanta)
Macro-Marketing Associates
404-662-5580

ILLINOIS
Chicago
Micro-Tex
312-382-3001

KANSAS
Kansas City
B.C. Electronic Sales
913-342-1211

Wichita
B.C. Electronic Sales
316-722-0104

MASSACHUSETTS
Needham Heights
Kanan Associates
617-449-7400

MARYLAND
Severna Park (Baltimore)
New Era Sales
301-544-4100

MINNESOTA
Edina (Minneapolis)
Mel Foster Tech Sales
612-941-9790

MISSOURI
St. Louis
B. C. Electronic Sales
314-521-6683

NEW JERSEY
Teaneck (New York City)
Technical Marketing Group
201-692-0200

NEW MEXICO
Albuquerque
Compass Marketing & Sales
505-888-0800

NEW YORK
Endwell
Tri-Tech Electronics
607-754-1094

Fayetteville
Tri-Tech Electronics
315-446-2881

Fishkill
Tri-Tech Electronics
914-897-5611

Melville
Technical Marketing Group
516-351-8833

Rochester
Tri-Tech Electronics
716-385-6500

NORTH CAROLINA
Raleigh
J & B Sales, Inc.
919-783-9440

Myrtle Beach
J & B Sales, Inc.
803-272-3475

OHIO
Beachwood
E.S.I.
216-831-9555

OKLAHOMA
Tulsa
B.P. Sales
918-744-9964

OREGON
Portland
W3C Technology Group
503-629-9871

PENNSYLVANIA
Feasterville (Philadelphia)
Knowles Associates
215-322-7100

TEXAS
Austin
B.P. Sales
512-346-9186

Houston
B.P. Sales
713-782-4144

Richardson (Dallas)
B.P. Sales
214-234-8438

UTAH
Salt Lake City
Waugaman Associates
801-261-0802

WASHINGTON
Seattle
W3C Technology Group
206-285-0210

WISCONSIN
Waukesha (Milwaukee)
Micro-Tex
414-542-5352

CANADA
Downsview, Ontario
Har-Tech Electronics
416-665-7773

Nepean, Ontario
Har-Tech Electronics
613-726-9410

Pointe Claire, Quebec
Har-Tech Electronics
514-694-6110

INMOS DISTRIBUTORS

ALABAMA
Huntsville
Arrow Electronics
205-837-6955

Huntsville
Pioneer-Standard
205-837-9300

ARIZONA
Phoenix
Wyle
602-866-2888

Tempe
Anthem Electronics
602-966-6600

Tempe
Arrow Electronics
602-968-4800

CALIFORNIA
Calabasas
Wyle
818-880-9001

Chatsworth
Anthem Electronics
818-700-1000

Chatsworth
Arrow Electronics
818-701-7500

El Segundo
Wyle
213-322-8100

Garden Grove
Wyle (Microelectronics)
714-891-1717

Irvine
Anthem Electronics
714-768-4444

Irvine
Wyle
714-863-9953

Irvine
Wyle (Military)
714-851-9953

Rancho Cordova
Wyle
916-638-5282

Sacramento
Anthem Electronics
916-922-6800

San Diego
Anthem Electronics
619-453-4871

San Diego
Arrow Electronics
619-565-4800

San Diego
Wyle
619-565-9171

San Jose
Anthem Electronics
408-295-4200

Santa Clara
Wyle
408-727-2500

Sunnyvale
Arrow Electronics
408-745-6600

Tustin
Arrow Electronics
714-838-5422

COLORADO
Aurora (Denver)
Arrow Electronics
303-696-1111

Englewood (Denver)
Anthem Electronics
303-790-4500

Thornton (Denver)
Wyle
303-457-9953

CONNECTICUT
Meriden
Anthem-Lionex Corporation
203-237-2282

Milford
Falcon Electronics
203-878-5272

Wallingford
Arrow Electronics
203-265-7741

FLORIDA
Ft. Lauderdale
Arrow Electronics
305-429-8200

Ft. Lauderdale
Pioneer-Standard
305-428-8877

Orlando
Pioneer-Standard
305-834-9090

Palm Bay
Arrow Electronics
305-725-1480

GEORGIA
Norcross
Arrow Electronics
404-449-8252

Norcross
Pioneer-Standard
404-448-1711

ILLINOIS
Elk Grove Village
Anthem Electronics
312-640-6066

Schaumburg (Chicago)
Arrow Electronics
312-397-3440

IOWA
Cedar Rapids
Arrow Electronics
319-395-7230

INDIANA
Indianapolis
Arrow Electronics
317-243-9353

KANSAS
Lenexa
Arrow Electronics
913-541-9542

MASSACHUSETTS
Wilmington
Anthem-Lionex Corporation
617-657-5170

Woburn
Arrow Electronics
617-933-8130

MARYLAND
Columbia
Arrow Electronics
800-842-7769

Columbia
Anthem-Lionex Corporation
301-995-6640

Gaithersburg
Pioneer-Standard
301-921-0660

MICHIGAN
Ann Arbor
Arrow Electronics
313-971-8220

Grand Rapids
Arrow Electronics
616-243-0912

MINNESOTA
Edina (Minneapolis)
Arrow Electronics
612-830-1800

Eden Praire
Anthem-Lionex Corporation
612-944-5454

MISSOURI
St. Louis
Arrow Electronics
314-567-6888

NORTH CAROLINA
Charlotte
Pioneer-Standard
704-527-8188

Raleigh
Arrow Electronics
919-876-3132

Winston-Salem
Arrow Electronics
919-725-8711

NEW HAMPSHIRE
Manchester
Arrow Electronics
603-668-6968

NEW JERSEY

Fairfield
Anthem-Lionex Corporation
201-227-7960

Parsippany
Arrow Electronics
201-538-0900

Marlton
Arrow Electronics
609-596-8000

NEW MEXICO
Albuquerque
Arrow Electronics
505-243-4566

NEW YORK
Hauppauge
Arrow Electronics
516-231-1000

Hauppauge
Anthem-Lionex Corporation
516-273-1660

Melville
Arrow Electronics
516-694-6800

Rochester
Arrow Electronics
716-427-0300

OHIO
Centerville (Dayton)
Arrow Electronics
513-435-5563

Columbus
Arrow Electronics
614-885-8362

Solon (Cleveland)
Arrow Electronics
216-248-3990

OKLAHOMA
Tulsa
Arrow Electronics
918-665-7700

OREGON
Beaverton
Anthem Electronics
503-643-1114

Portland
Arrow Electronics
503-684-1690

Portland
Wyle
503-640-6000

PENNSYLVANIA
Horsham
Anthem-Lionex Corporation
215-443-5150

Horsham
Pioneer-Standard
215-674-4000

Monroeville
Arrow Electronics
412-856-7000

TEXAS
Austin
Arrow Electronics
512-835-4180

Austin
Wyle
512-834-9957

Carrollton (Dallas)
Arrow Electronics
214-380-6464

Houston
Wyle
713-879-9953

Richardson (Dallas)
Wyle
214-235-9953

Houston
Arrow Electronics
713-530-4700

UTAH
Salt Lake City
Anthem Electronics
801-973-8555

Salt Lake City
Arrow Electronics
801-972-0404

Salt Lake City
Wyle
801-974-9953

WASHINGTON
Bellevue (Seattle)
Arrow Electronics
206-643-4800

Bellevue (Seattle)
Wyle
206-453-8300

Redmond (Seattle)
Anthem Electronics
206-881-0850

WISCONSIN
Brookfield (Milwaukee)
Arrow Electronics
414-792-0150

CANADA
Baxter Center , Ottawa
Future Electronics
613-820-8313

Calgary, Alberta
Future Electronics
403-235--5325

Downsview, Ontario
Future Electronics
416-638-4771

Pointe Claire, Quebec
Future Electronics
514-694-7710

Vancouver, B.C.
Future Electronics
604-438-3321

INMOS SALES OFFICES

CALIFORNIA
Santa Clara
INMOS Sales
408-727-7771

Santa Ana
INMOS Sales
714-957-6018

COLORADO
Denver
INMOS Sales
303-252-4100

GEORGIA
Norcross
INMOS Sales
404-242-7444

MASSACHUSETTS
Westborough
INMOS Sales
617-366-4020

MARYLAND
Columbia (Baltimore)
INMOS Sales
301-995-6952

MINNESOTA
Minneapolis
INMOS Sales
612-831-5626

TEXAS
Dallas
INMOS Sales
214-490-9522

INTERNATIONAL
France, Paris
INMOS SARL
(1)-4687-22-01

United Kingdom, Bristol
INMOS Limited
454-616616

West Germany
INMOS GmbH
089-319-1028

Japan, Tokyo
INMOS Japan K.K.
03-505-2840

inmos®

INMOS Corporation • P.O. Box 16000 • Colorado Springs, CO • 80935 • USA • (303) 630-4000 • Easy Link 62944936
INMOS Limited • 1000 Aztec West • Almondsbury • Bristol BS12 4SQ • England • Tel (0454) 616616 • TLX 851-444723
INMOS SARL • Immeuble Monaco • 7 rue Le Corbusier SILIC 219 • 94518 Rungis Cedex • France • Tel (1) 4687-22-01 • TLX 201222
INMOS GmbH • Danziger Strasse 2 • 8057 Eching • Munich • West Germany • Tel (089) 319-1028 • TLX 522645
INMOS Japan K.K. • No. 16 Annex, Room 308 • 9-20 Akasaka 1-Chome • Minato-Ku/Tokyo • 107 Japan • Tel (035) 052840 • TWX 29507