IMS T212
transputer

The IMS T222 is recommended for new designs

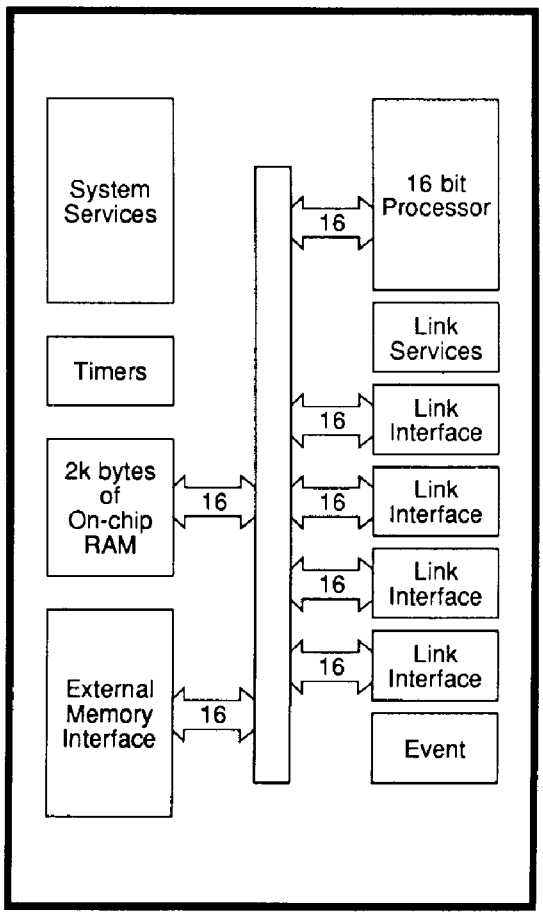
Engineering Data

FEATURES

16 bit architecture
50 ns internal cycle time
20 MIPS (peak) instruction rate
Pin compatible with IMS T222
2 Kbytes on-chip static RAM
80 Mbytes/sec sustained data rate to internal memory
64 Kbytes directly addressable external memory
20 Mbytes/sec sustained data rate to external memory
950 ns response to interrupts
Four INMOS serial links 5/10/20 Mbits/sec
Bi-directional data rate of 1.6 Mbytes/sec per link
Internal timers of 1 μs and 64 μs
Boot from ROM or communication links
Single 5 MHz clock input
Single +5V ±5% power supply, less than 1 Watt

APPLICATIONS

Real time processing
Microprocessor applications
High speed multi processor systems
Industrial control
Robotics
System simulation
Digital signal processing
Telecommunications
Fault tolerant systems
Medical instrumentation
Pattern recognition
Image processing
Graphics processing
Artificial intelligence
Supercomputers



1 Introduction

The IMS T212 transputer is a 16 bit CMOS microcomputer with 2 Kbytes on-chip RAM for high speed processing, an external 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. A device running at 20 MHz achieves an instruction throughput of 10 MIPS.

For convenience of description, the IMS T212 operation is split into the basic blocks shown in figure 1.1.

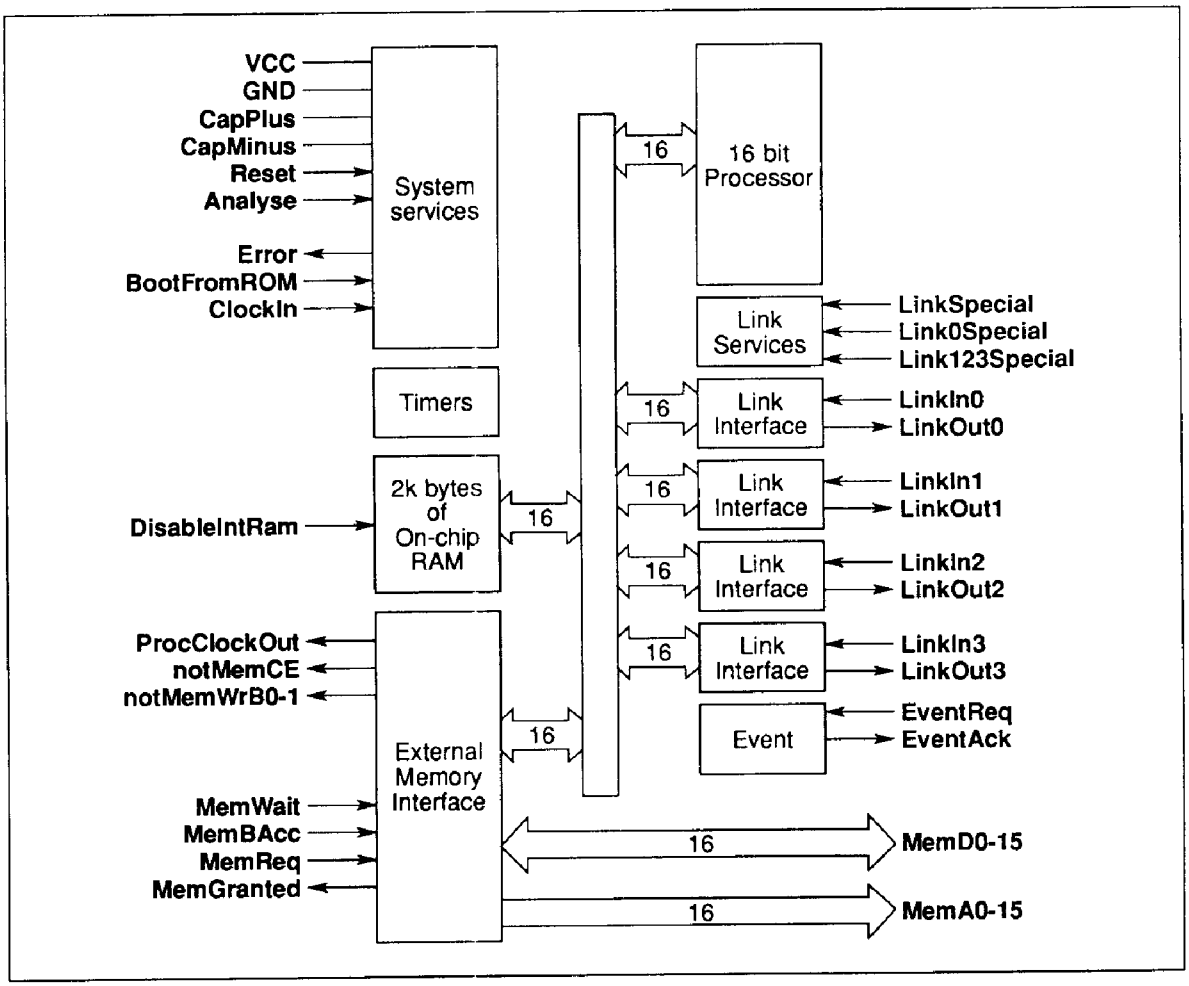


Figure 1.1 IMS T212 block diagram

2 Pin designations

Table 2.1 IMS T212 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
Reset	in	System reset
Error	out	Error indicator
Analyse	in	Error analysis
BootFromRom	in	Bootstraps from external ROM or from link
DisableIntRAM	in	Disable internal RAM
HoldToGND		Must be connected to **GND**

Table 2.2 IMS T212 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemA0-15	out	Sixteen address lines
MemD0-15	in/out	Sixteen data lines
notMemWrB0-1	out	Two byte-addressing write strobes
notMemCE	out	Chip enable
MemBAcc	in	Byte access mode selector
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted

Table 2.3 IMS T212 event

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

Table 2.4 IMS T212 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

Signal names are prefixed by **not** if they are active low, otherwise they are active high.
Pinout details for various packages are given on page 345.

3 Processor

The 16 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 2 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 64 Kbytes of memory via the External Memory Interface (EMI).

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 implicitly. 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.

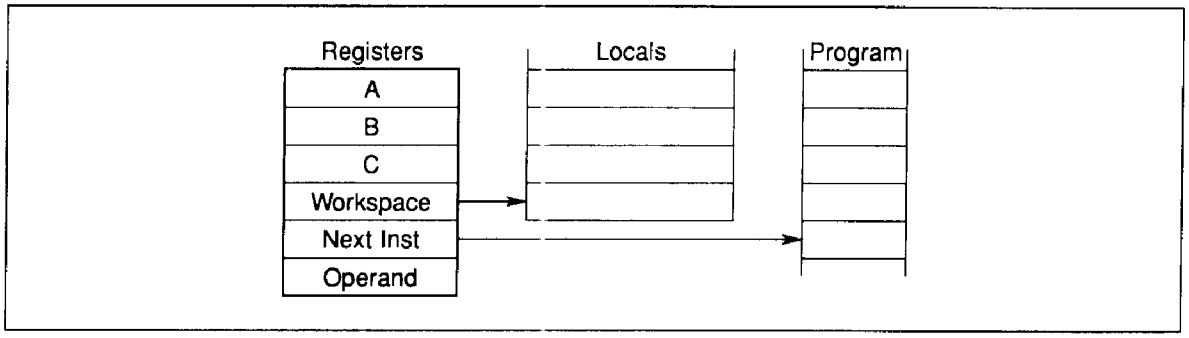


Figure 3.1 Registers

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.

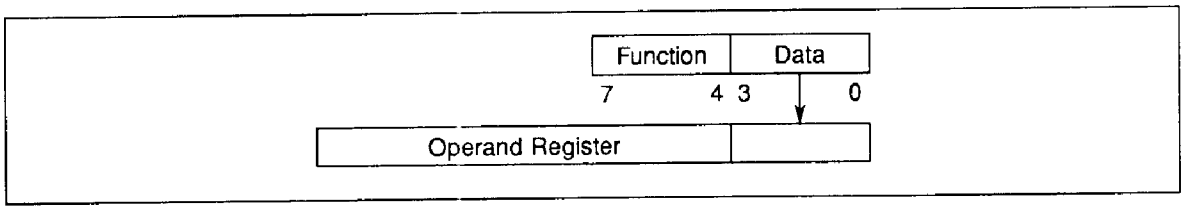


Figure 3.2 Instruction format

3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

Table 3.1 Direct functions

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

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 or Pascal.

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 occurring 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 occurring 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.

Table 3.2 Expression evaluation

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; that is, 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 two 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 (page 305).

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 (page 305). 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 figure 3.3, 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.

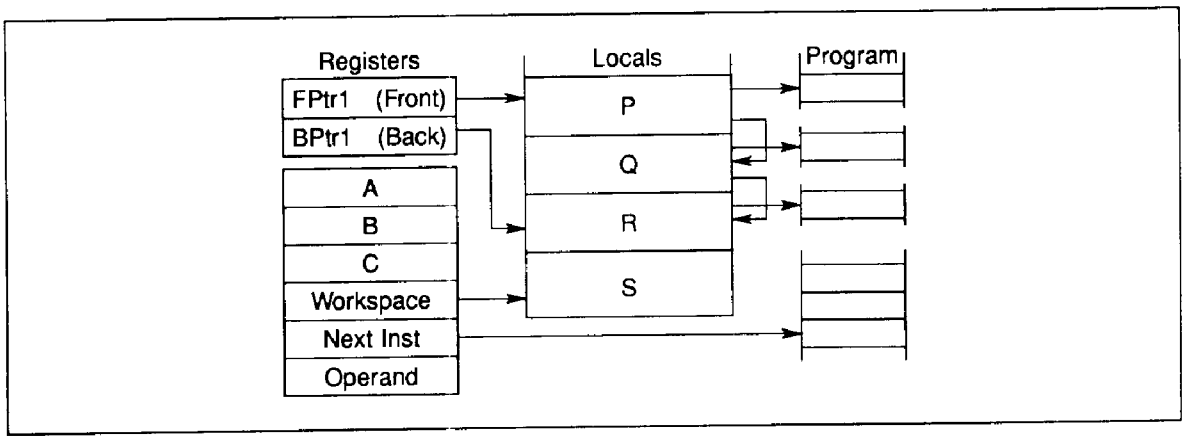


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	*Fptr0*	*Fptr1*
Pointer to back of active process list	*Bptr0*	*Bptr1*

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 descheduling point (page 308). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

A process can only be descheduled on certain instructions, known as descheduling points (page 308). 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 T212 supports two levels of priority. 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 (page 308).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms 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 of 53 cycles (assuming use of on-chip RAM).

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 16 bit timer clocks 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 approximately 65 milliseconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately four seconds.

Table 3.4 Timer registers

Clock0	Current value of high priority (level 0) process clock
Clock1	Current value of low priority (level 1) process clock
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 the processor clock 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.

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

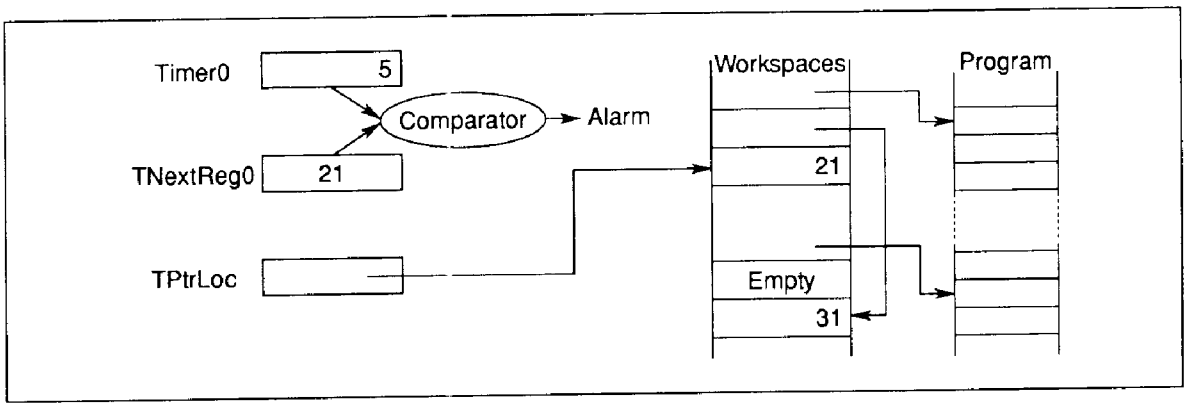


Figure 3.4 Timer registers

4 Instruction set summary

The Function Codes table 4.7. gives the basic function code set (page 302). 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*.

Table 4.1 *prefix* coding

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* #FFE1)		
is coded as				
nfix	#1		#6	#61
ldc	#1		#4	#41

Tables 4.8 to 4.17 give details of the 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.

Table 4.2 *operate* coding

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

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 the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

Table 4.3 Instruction set interpretation

Ident	Interpretation
b	Bit number of the highest bit set in register *A*. Bit 0 is the least significant bit.
n	Number of places shifted.
w	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.

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

Ident	Feature	See page:
D	The instruction is a descheduling point	308
E	The instruction will affect the *Error* flag	308, 317

4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 304). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 316).

Table 4.5 Descheduling point instructions

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*

4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the *Error* flag (page 317) directly.

Table 4.6 Error setting instructions

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

Table 4.7 IMS T212 function codes

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
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	E
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	

Table 4.8 IMS T212 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
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	E
0C	FC	sub	1	subtract	E
53	25F3	mul	23	multiply	E
2C	22FC	div	24	divide	E
1F	21FF	rem	21	remainder	E
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product	

Table 4.9 IMS T212 long arithmetic operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	E
38	23F8	lsub	2	long subtract	E
37	23F7	lsum	3	long sum	
4F	24FF	ldiff	3	long diff	
31	23F1	lmul	17	long multiply	
1A	21FA	ldiv	19	long divide	E
36	23F6	lshl	**n**+3	long shift left (**n**<16)	
			n-12	long shift left(**n**≥16)	
35	23F5	lshr	**n**+3	long shift right (**n**<16)	
			n-12	long shift right (**n**≥16)	
19	21F9	norm	**n**+5	normalise (**n**<16)	
			n-10	normalise (**n**≥16)	
			3	normalise (**n**=32)	

Table 4.10 IMS T212 general operation codes

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

Table 4.11 IMS T212 indexing/array operation codes

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

Table 4.12 IMS T212 timer handling operation codes

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

Table 4.13 IMS T212 input/output operation codes

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

Table 4.14 IMS T212 control operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	
1B	21FB	ldpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	
06	F6	gcall	4	general call	
21	22F1	lend	10	loop end (loop)	D
			5	loop end (exit)	D

Table 4.15 IMS T212 scheduling operation codes

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

Table 4.16 IMS T212 error handling operation codes

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

Table 4.17 IMS T212 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
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	

5 System services

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

5.1 Power

Power is supplied to the device 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 100 nF 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 20 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

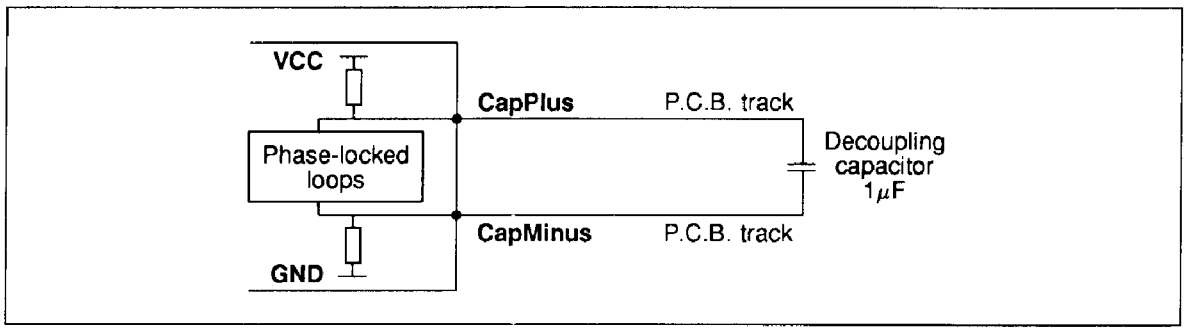


Figure 5.1 Recommended PLL decoupling

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 5 MHz, regardless of device type, transputer 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.

Table 5.1 Input clock

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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** (table 10.3).

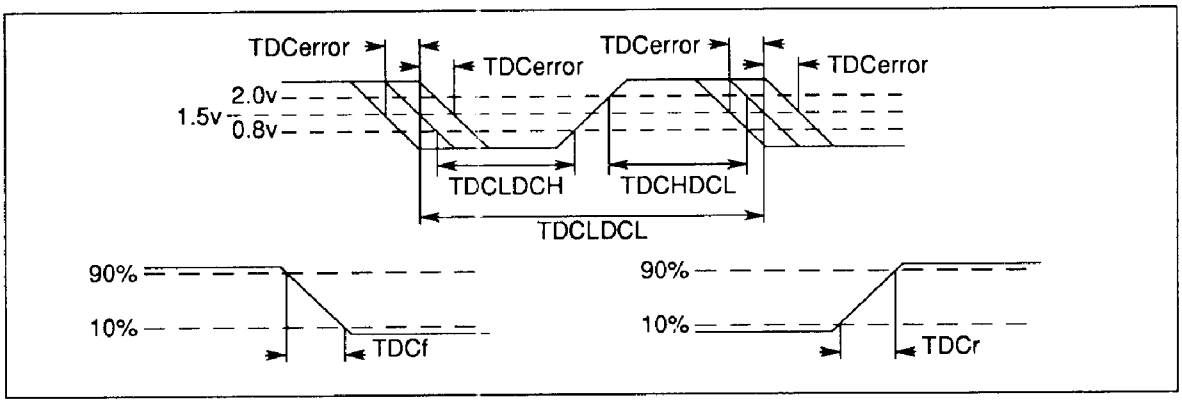


Figure 5.2 ClockIn timing

5.4 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 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 bootstrap (page 329). If **BootFromRom** is high bootstrapping will take place immediately after **Reset** goes low, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

5.5 Bootstrap

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. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

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 #7FFE. This location should contain a backward jump to a program in ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority state, and the *W* register points to *MemStart* (page 318).

Table 5.2 Reset and Analyse

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0			ms	3
TALBRX	BootFromRom hold after Analyse					3

Notes

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

2 At power-on reset.

3 Must be stable until after end of bootstrap period. See Bootstrap section.

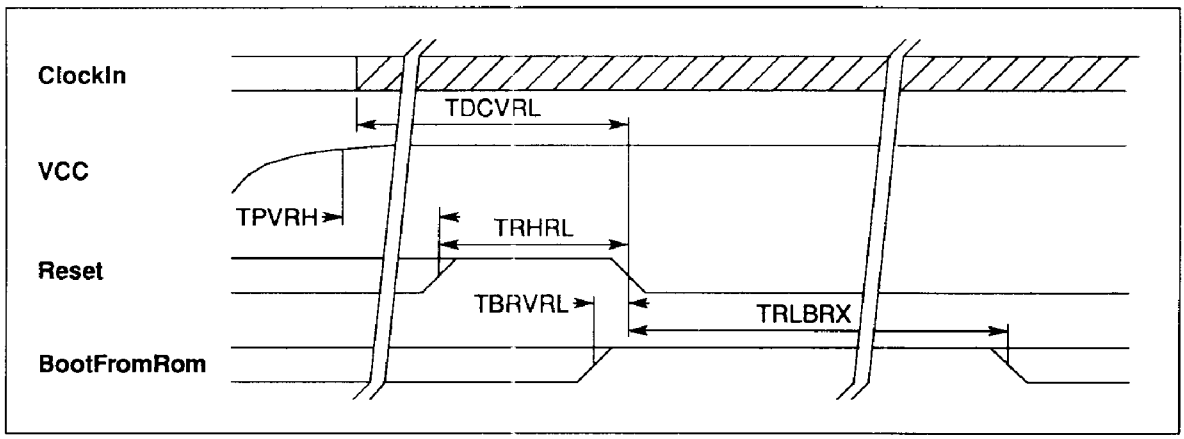


Figure 5.3 Transputer reset timing with Analyse low

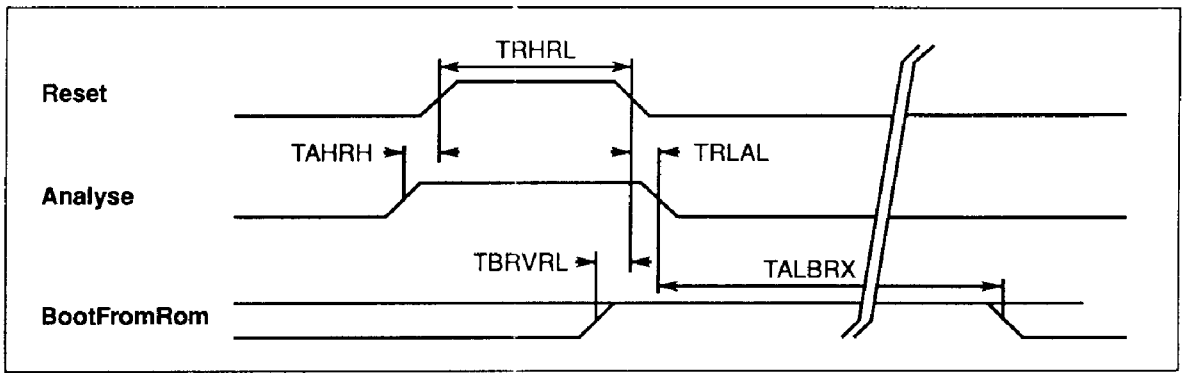


Figure 5.4 Transputer reset and analyse timing

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. **BootFromRom** may be taken high after reception of the last byte, if required. 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 bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

5.6 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then four more bytes are expected on the same link. The first two byte word is taken as an internal or external memory address at which to poke (write) the second two byte word. If the control byte is 1 the next two 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 bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

5.7 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 308). 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. Processor flags **Error** and **HaltOnError** are not altered at reset, whether **Analyse** is asserted or not.

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. If **BootFromRom** is high the transputer will bootstrap 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 values given in table 5.3.

Table 5.3 Register values after Analyse

I	*MemStart* if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
W	*MemStart* if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping 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 bootstrapping link if bootstrapping from link.

5.8 Error

The **Error** pin is connected directly to the internal *Error* flag and follows the state of that flag. If **Error** is high it indicates an error in one of the processes caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 308). Once set, the *Error* flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 316).

A process can be programmed to stop if the *Error* flag 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.

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 the processor reset and analyse facilities 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 applying the **Error** output signal of the errant transputer 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. When using such a circuit, note that the *Error* flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary 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 the *Error* flag is transmitted to the high priority process but the *HaltOnError* flag is cleared before the process starts. 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.

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

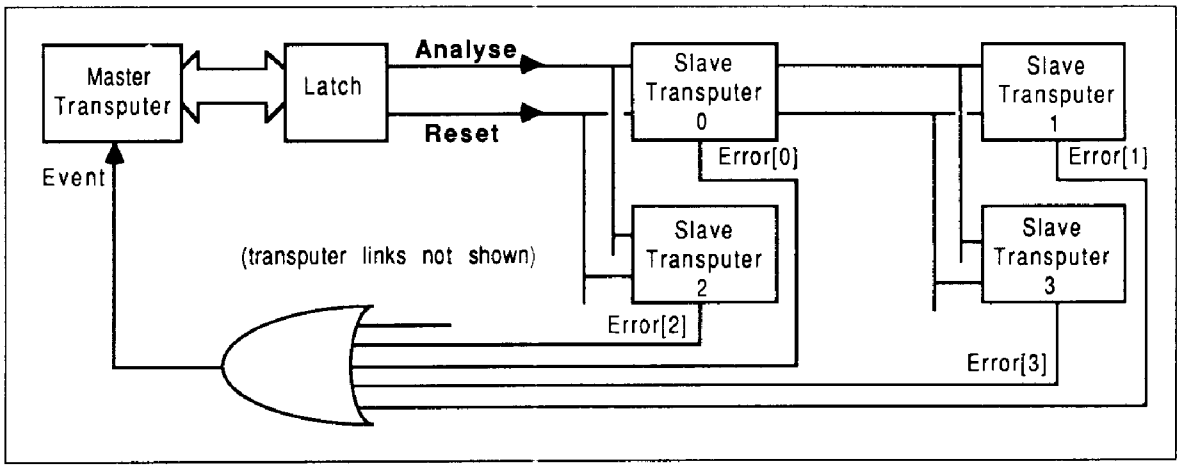


Figure 5.5 Error handling in a multi-transputer system

6 Memory

The IMS T212 has 2 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 320). The transputer can also access an additional 62 Kbytes of external memory space. Internal and external memory are part of the same linear address space. Internal RAM can be disabled by holding **DisableIntRAM** high. All internal addresses are then mapped to external RAM. This pin should not be altered after **Reset** has been taken low.

IMS T212 memory is byte addressed, with words aligned on two-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 #8000 and extends to #87FF. User memory begins at #8024; 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.

External memory space starts at #8800 and extends up through #0000 to #7FFF. ROM bootstrapping code must be in the most positive address space, starting at #7FFE. Address space immediately below this is conventionally used for ROM based code

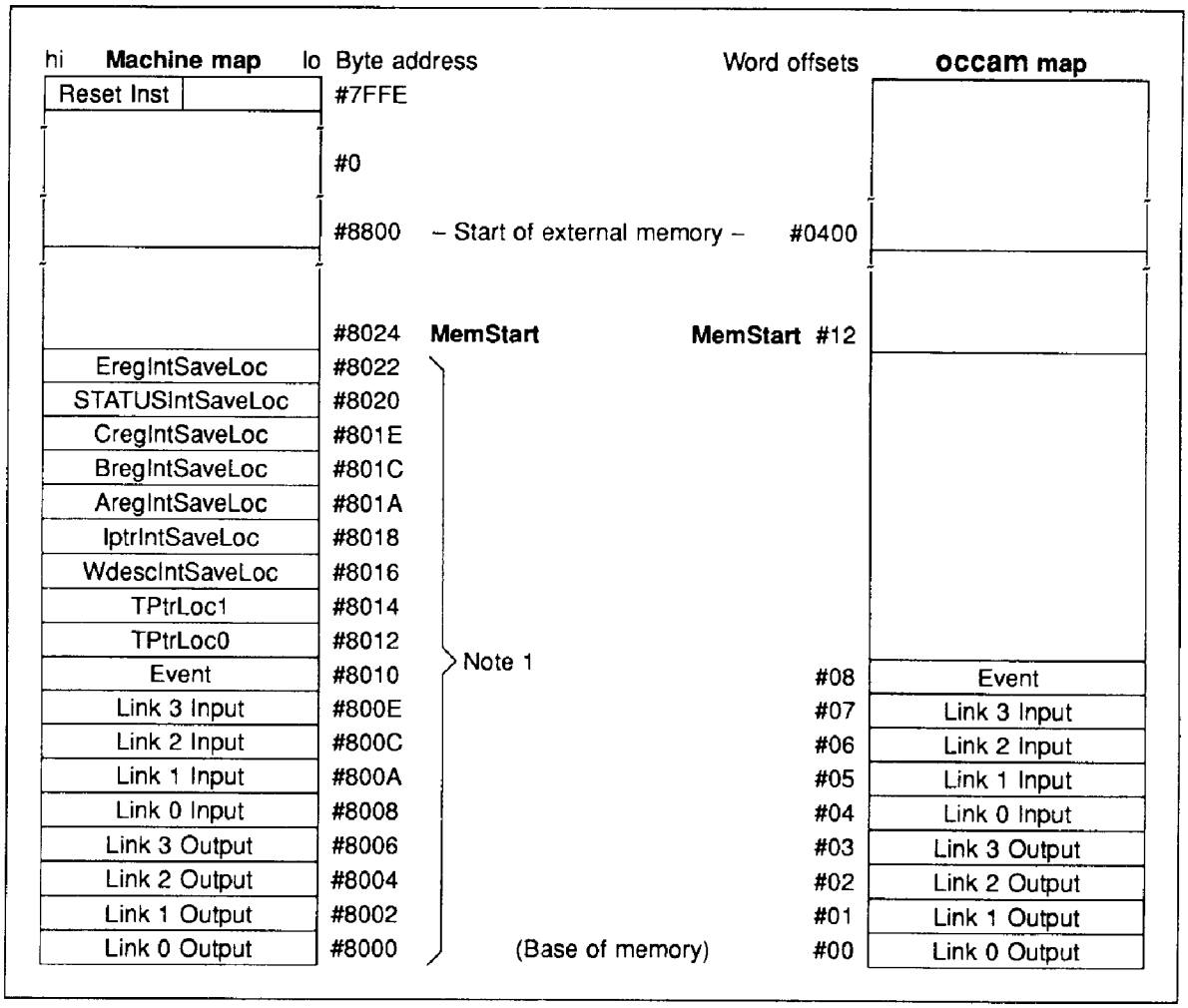


Figure 6.1 IMS T212 memory map

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 (**Analyse**, page 316). For details see The Transputer Instruction Set - A Compiler Writers' Guide.

7 External memory interface

The IMS T212 External Memory Interface (EMI) allows access to a 16 bit address space via separate address and data buses. The data bus can be configured for either 16 bit or 8 bit memory access, allowing the use of a single bank of byte-wide memory. Both word-wide and byte-wide access may be mixed in a single memory system (page 326).

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 (ordering section).

Edges of the various external memory strobes are synchronised by, but do not all coincide with, rising or falling edges of **ProcClockOut**.

Table 7.1 **ProcClockOut**

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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
TPCstab	ProcClockOut stability			4	%	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.

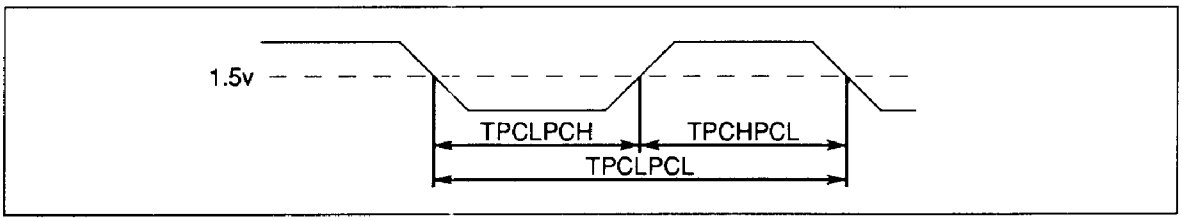


Figure 7.1 IMS T212 ProcClockOut timing

7.2 Tstates

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

T1 Address and control setup time.

T2 Data setup time.

T3 Data read/write.

T4 Data and address hold after access.

Each **Tstate** is half a processor cycle **TPCLPCL** long, displaced by approximately one fourth of a cycle from **ProcClockOut** edges. **T2** can be extended indefinitely by adding externally generated wait states of one complete processor cycle each.

An external memory cycle is always a complete number of cycles **TPCLPCL** in length. The start of **T1** always coincides with the low phase of **ProcClockOut**.

7.3 Internal access

During an internal memory access cycle the external memory interface address bus **MemA0-15** reflects the word address used to access internal RAM, **notMemWrB0-1** reflect the internal read/write operation, **notMemCE** is inactive and the data bus **MemD0-15** is tristated. This is true unless and until a DMA (memory request) activity takes place, when the lines will be placed in a high impedance state by the transputer.

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 (page 316).

7.4 MemA0-15

External memory addresses are output on a non-multiplexed 16 bit bus. The address is valid at the start of **T1** and remains so until the end of **T4**, with the timing shown. Byte addressing is carried out internally by the IMS T212 for read cycles. For write cycles the relevant bytes in memory are addressed by the write enables **notMemWrB0-1**.

The transputer places the address bus in a high impedance state during DMA.

7.5 MemD0-15

The non-multiplexed data bus is 16 bits wide. Read cycle data may be set up on the bus at any time after the start of **T1**, but must be valid when the IMS T212 reads it during **T3**. Data can be removed any time during **T4**, 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 T2 and removed at the end of **T4**. It is normally written into memory in synchronism with **notMemCE** going high.

The data bus is high impedance except when the transputer is writing data. If only one byte is being written, the unused 8 bits of the bus are high impedance at that time. In byte access mode **MemD8-15** are high impedance during the external memory cycle which writes the most significant (second) byte (page 326).

If the data setup time for read or write is too short it can be extended by inserting wait states at the end of **T2** (page 327).

Table 7.2 Read

SYMBOL	PARAMETER	T212-20		T212-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TAVEL	Address valid before chip enable low	13	16	15	19	ns	
TELEH	Chip enable low	56	63	65	72	ns	
TEHEL	Delay before chip enable re-assertion	35	46	40	51	ns	1
TEHAX	Address hold after chip enable high	20	24	21	27	ns	
TELDrV	Data valid from chip enable low		40		43	ns	
TDrVEH	Data setup before chip enable high	11		15		ns	
TEHDrZ	Data hold after chip enable high	0		0		ns	
TWEHEL	Write enable setup before chip enable low	14		18		ns	2

Notes

1 These values assume back-to-back external memory accesses.

2 Timing is for both write enables **notMemWrB0-1**.

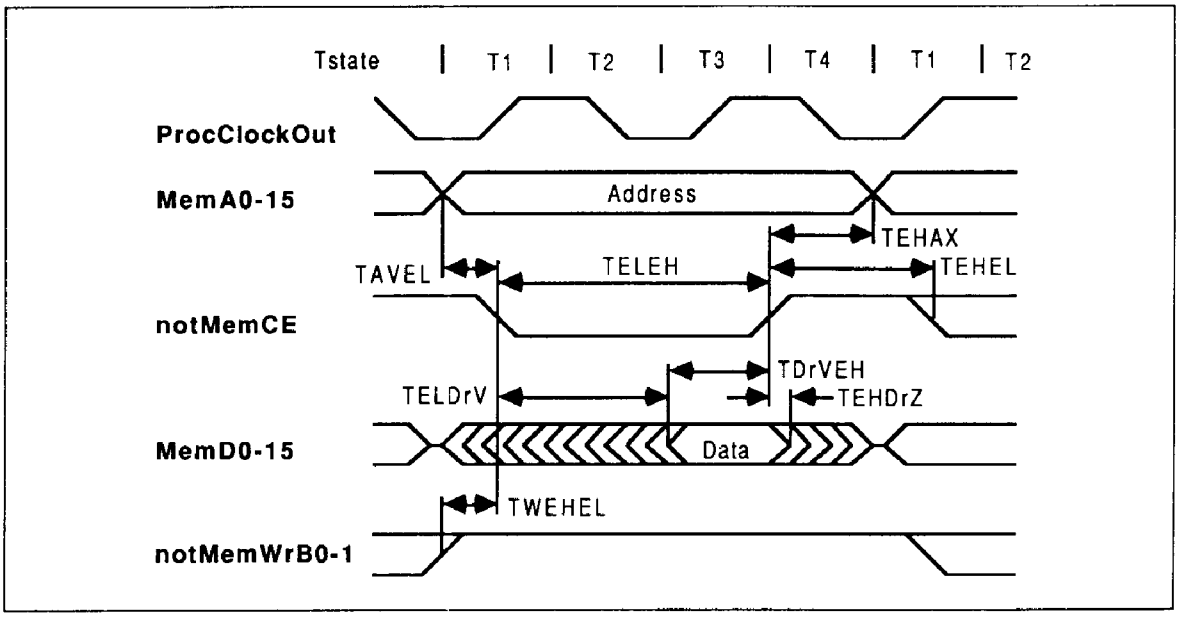


Figure 7.2 IMS T212 external read cycle

7.6 notMemWrB0-1

Two write enables are provided, one to write each byte of the word. When writing a word, both write enables are asserted; when writing a byte only the appropriate write enable is asserted. **notMemWrB0** addresses the least significant byte. The write enables are active before the chip enable signal **notMemCE** becomes active, thus reducing memory access time and the risk of bus contention.

The write enables are active before the chip enable signal **notMemCE**, thus reducing memory access time and the risk of bus contention.

Data must be strobed into memory by, or in conjunction with, **notMemCE**, as the write enables are not guaranteed to go high between consecutive write cycles. The write enables are placed in a high impedance state during DMA.

Table 7.3 Write

SYMBOL	PARAMETER	T212-20		T212-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TDwVEH	Data setup before chip enable high	36		42		ns	
TEHDwZ	Data hold after write	22	30	24	32	ns	
TWELEL	Write enable setup before chip enable low	4	20	4	24	ns	1
TEHWEH	Write enable hold after chip enable high	17	25	18	27	ns	1

Notes

1 Timing is for both write enables **notMemWrB0-1**.

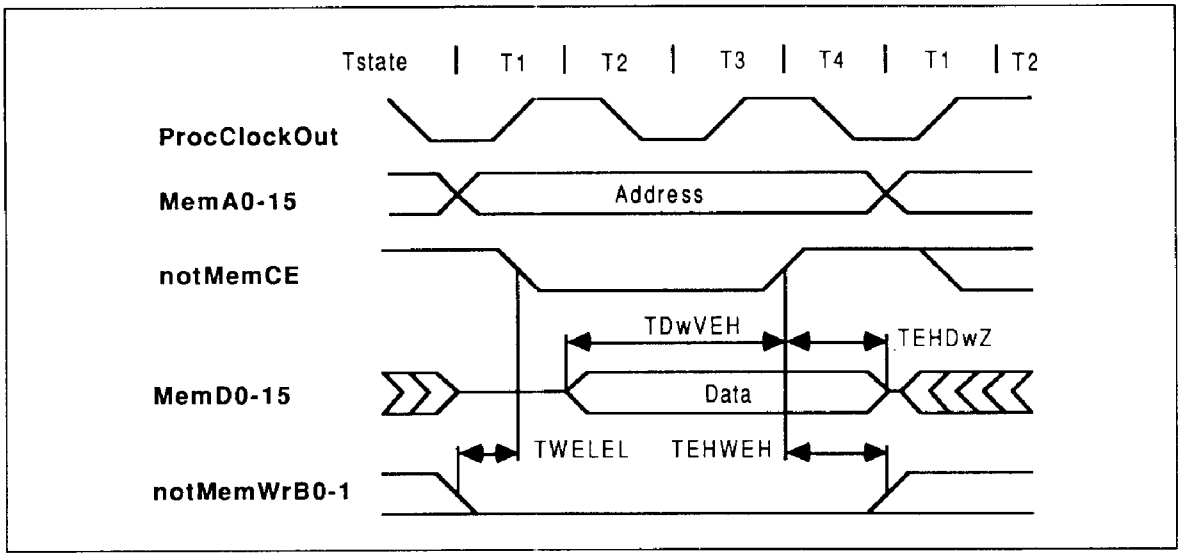


Figure 7.3 IMS T212 external write cycle

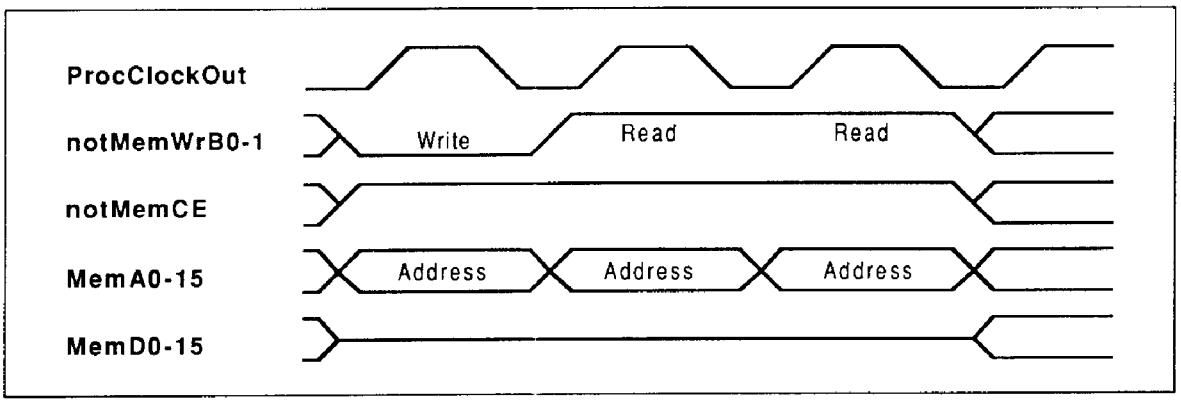


Figure 7.4 IMS T212 typical bus activity for internal memory cycles

7.7 notMemCE

The active low signal **notMemCE** is used to enable external memory on both read and write cycles. It must be used, in conjunction with the write enables **notMemWrB0-1**, to write data into memory; the write enable lines only select the byte of memory to be written.

Table 7.4 **notMemCE** to **ProcClockOut** skew

SYMBOL	PARAMETER	T212-20		T212-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TPCHEL	notMemCe falling from ProcClockOut rising	1	5	2	8	ns	
TEHPCL	notMemCe rising to ProcClockOut falling	8	14	10	15	ns	

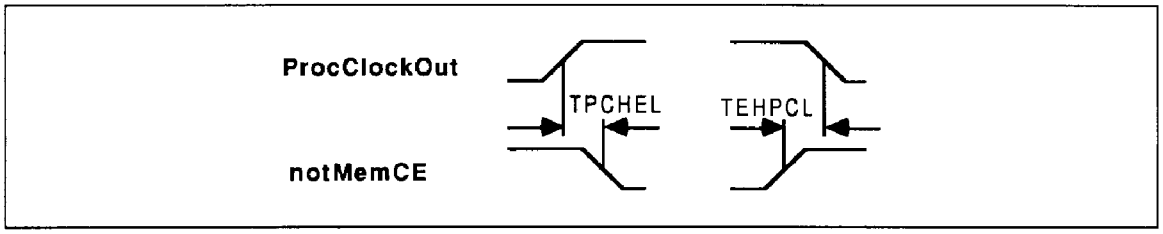


Figure 7.5 IMS T212 skew of **notMemCE** to **ProcClockOut**

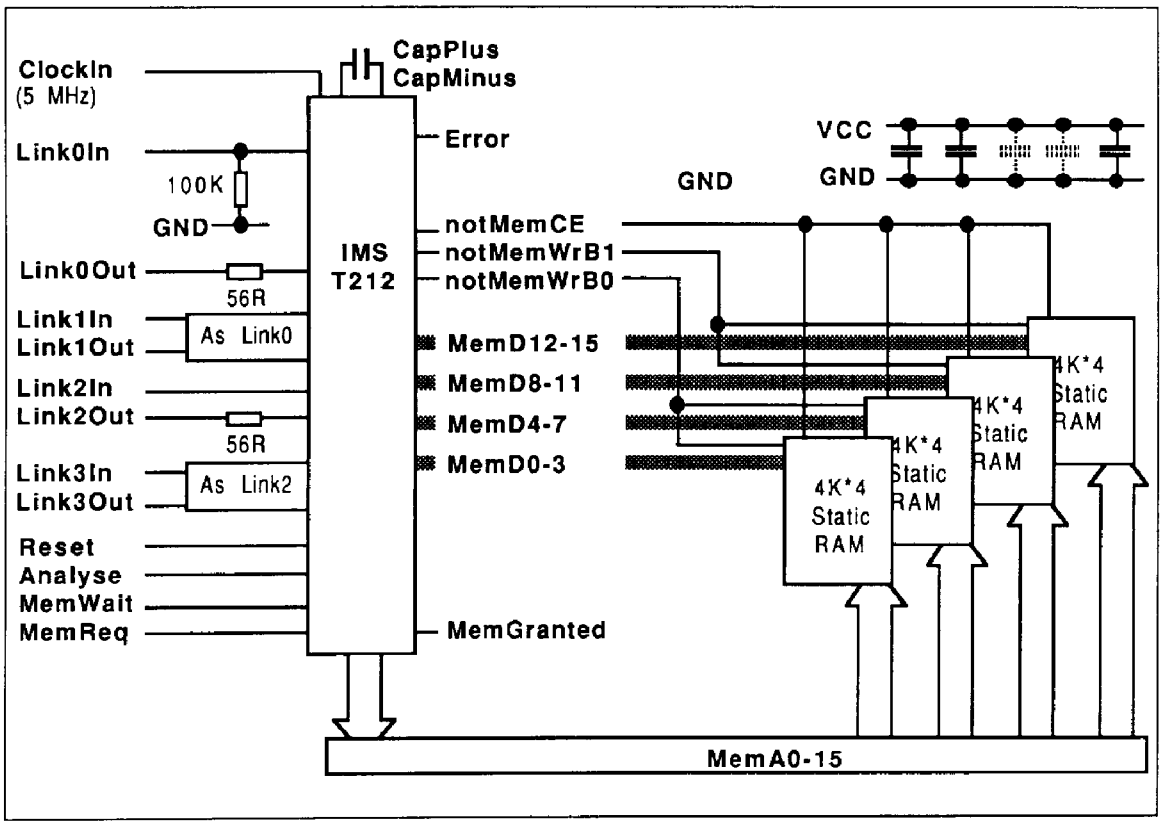


Figure 7.6 IMS T212 application

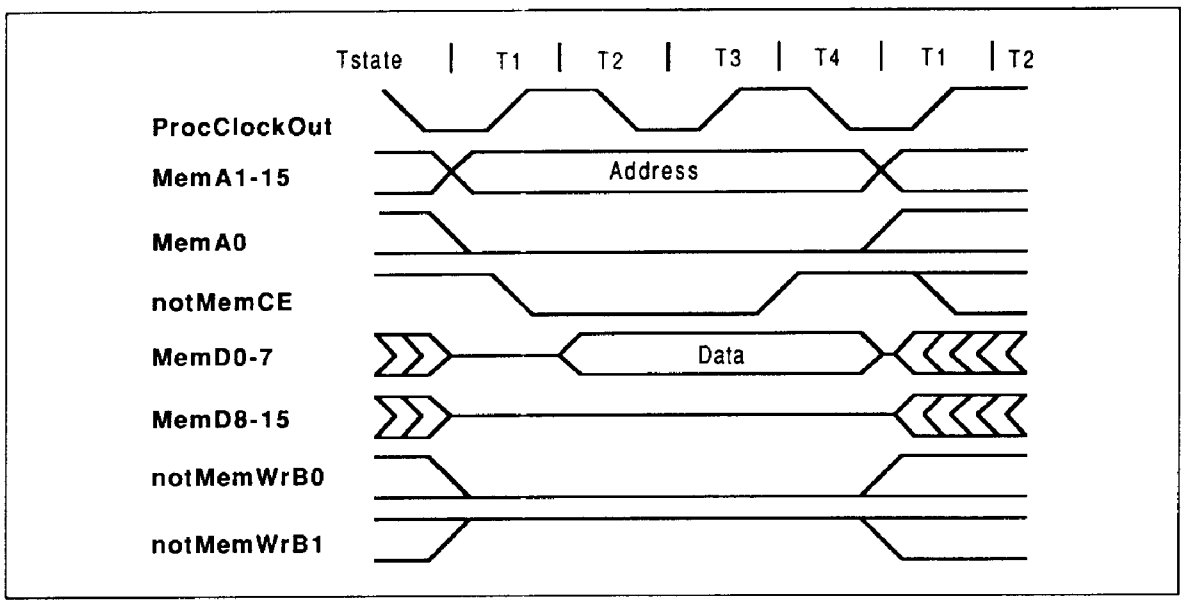


Figure 7.7 IMS T212 Least significant byte write in word access mode

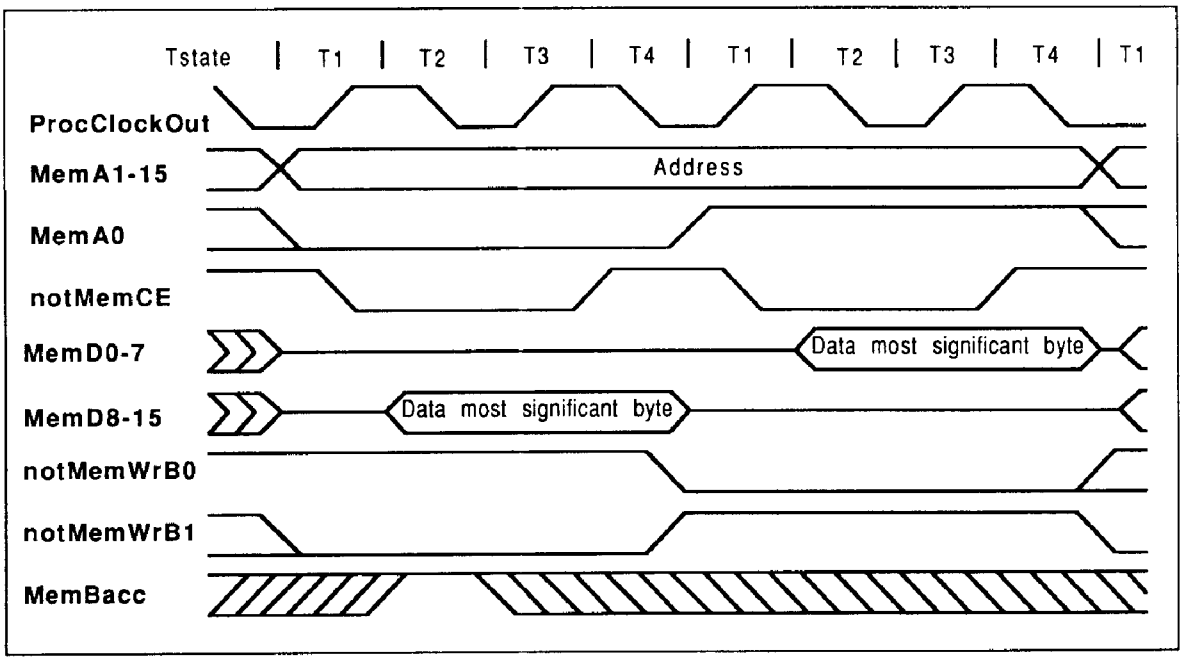


Figure 7.8 IMS T212 Most significant byte write to byte-wide memory

7.8 MemBAcc

The IMS T212 will, by default, perform word access at even memory locations. Access to byte-wide memory can be achieved by taking **MemBAcc** high with the timing shown. Where all external memory operations are to byte-wide memory, **MemBAcc** may be wired permanently high. The state of this signal is latched during **T2**.

If **MemBAcc** is low then a full word will be accessed in one external memory cycle, otherwise the high and low bytes of the word will be separately accessed during two consecutive cycles. The first (least significant) byte is accessed at the word address (**MemA0** is low). The second (most significant) byte is accessed at the word address +1 (**MemA0** is high).

With **MemBAcc** high, the first cycle is identical with a normal word access cycle. However, it will be immediately followed by another memory cycle, which will use **MemD0-7** to read or write the second (most significant) byte of data. During this second cycle **notMemWrB1** remains high, both for read and write, and **MemD8-15** are high impedance. When writing a single byte with **MemBAcc** high, both the first and second cycles are performed with **notMemWrB0** asserted in the appropriate cycle.

Table 7.5 Byte-wide memory access

SYMBOL	PARAMETER	T212-20		T212-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TELBAH	MemBAcc high from chip enable		12		15	ns	
TELBAL	MemBAcc low from chip enable	26		29		ns	

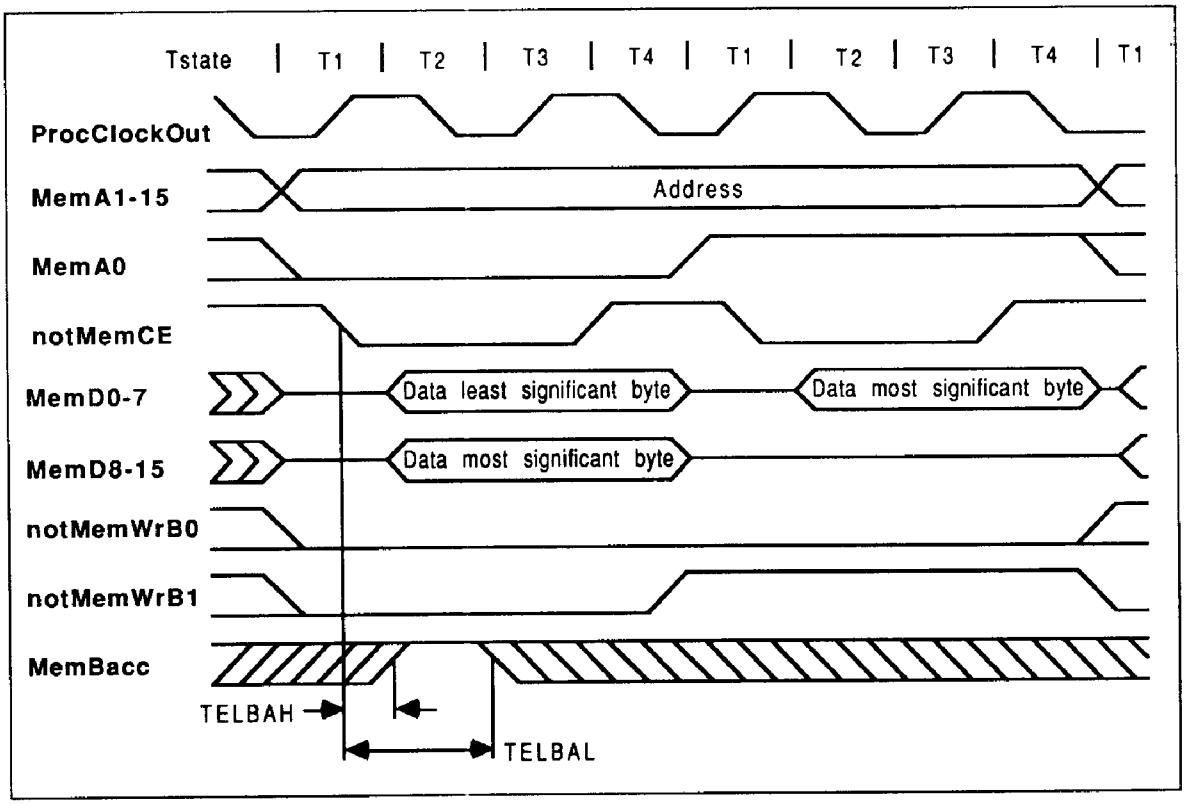


Figure 7.9 IMS T212 word write to byte-wide memory

7.9 MemWait

Taking **MemWait** high with the timing shown in the diagram will extend the duration of **T2** by one processor cycle **TPCLPCL**. One wait state comprises the pair **W1** and **W2**. **MemWait** is sampled near the falling edge of **ProcClockOut** during **T2**, and should not change state in this region. If **MemWait** is still high when sampled near the falling edge of **ProcClockOut** in **W2** then another wait period will be inserted. This can continue indefinitely. Internal memory access is unaffected by the number of wait states selected.

The wait state generator can be a simple digital delay line, synchronised to **notMemCE**. The **Single Wait State Generator** circuit in figure 7.11 can be extended to provide two or more wait states, as shown in figure 7.12.

The **Programmable Wait State Generator** circuit in figure 7.13 is designed to be interfaced directly to any memory or peripheral enable signal; 'F' series devices should be employed to ensure minimum delay between **notMemCE** and a valid **notWaitX** input. Only one wait select input line should be low at any one time; for zero wait states **notWait0** must be asserted.

Table 7.6 Memory wait

SYMBOL	PARAMETER	T212-20		T212-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TELWtH	MemWait asserted after chip enable low		13		13	ns	
TELWtL	Wait hold after chip enable low	23	**a**+13	23	**a**+13	ns	1

Notes

1 **a** is **w*****c** where **w** is the number of wait states and **c** is the toleranced clock period of 49 ns for IMS T212-20, 56 ns for IMS T212-17.

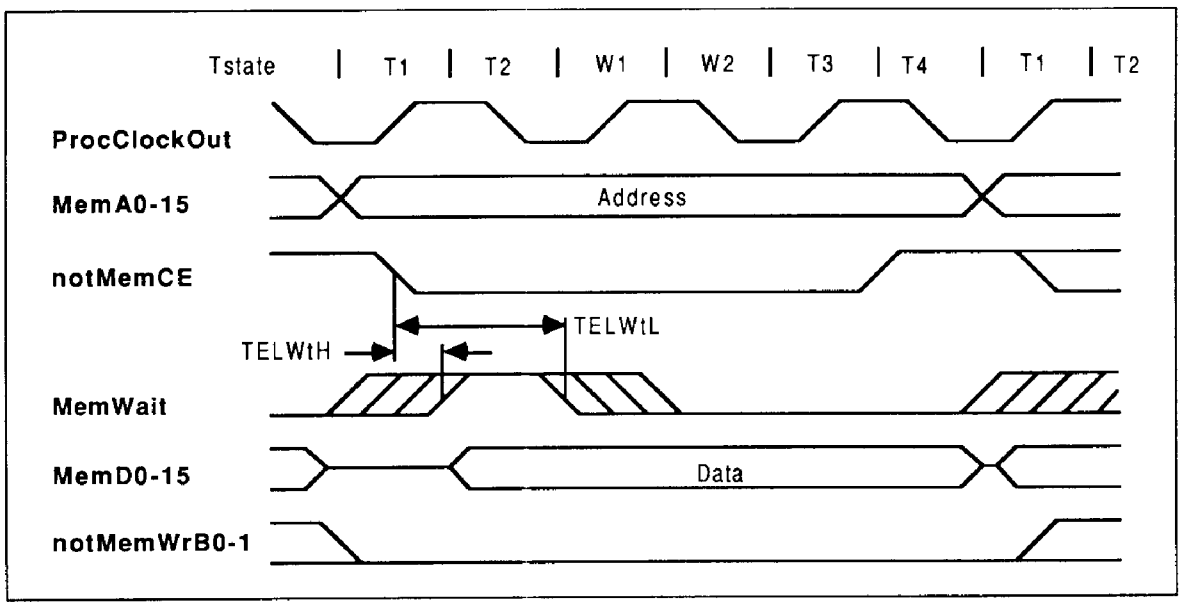


Figure 7.10 IMS T212 memory wait timing

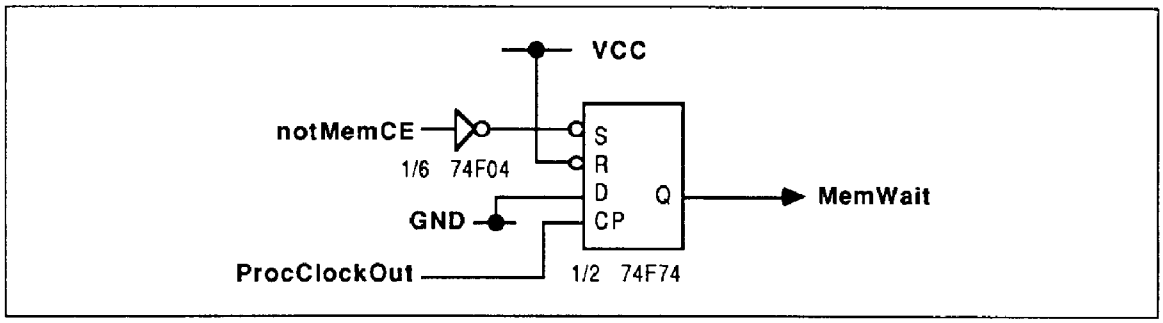


Figure 7.11 Single wait state generator

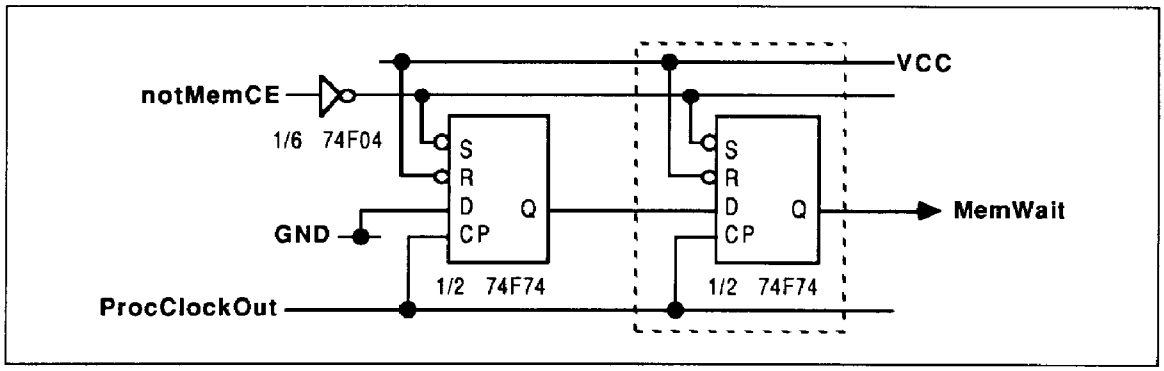


Figure 7.12 Extendable wait state generator

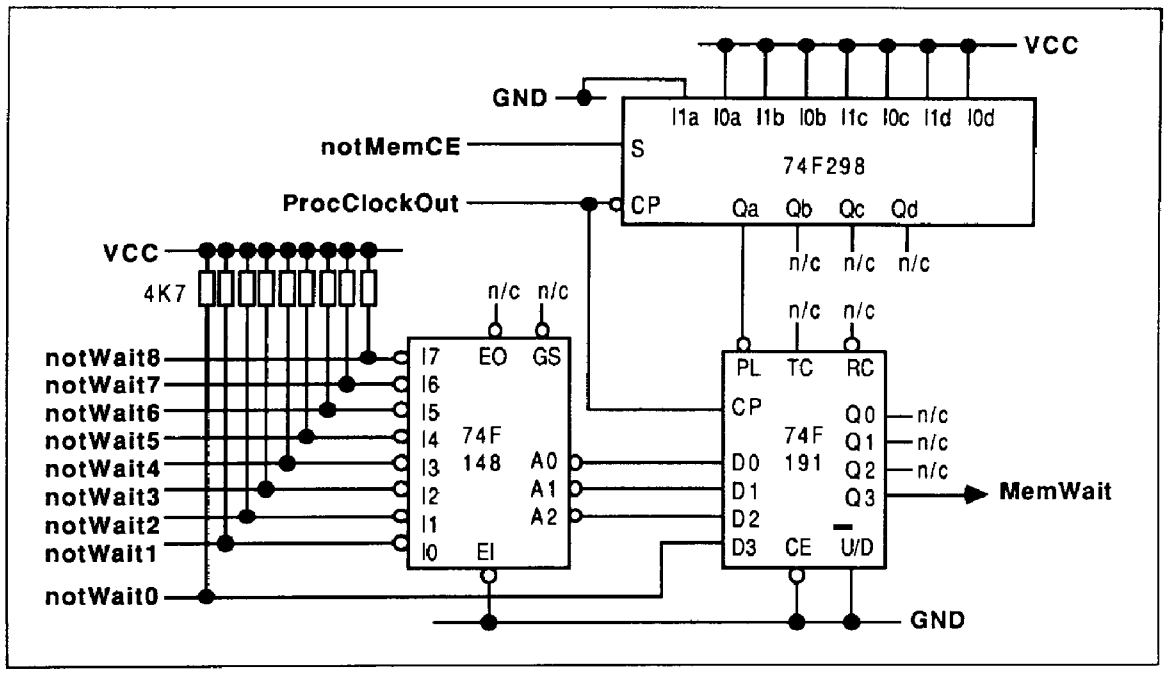


Figure 7.13 Programmable wait state generator

7.10 MemReq, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. For external memory cycles, the IMS T212 samples **MemReq** during the first high phase of **ProcClockOut** after **notMemCE** goes low. In the absence of an external memory cycle, **MemReq** is sampled during every high phase of **ProcClockOut**. **MemA0-15**, **MemD0-15**, **notMemWrB0-1** and **notMemCE** are tristated before **MemGranted** is asserted.

Removal of **MemReq** is sampled during each high phase of **ProcClockOut** and **MemGranted** removed with the timing shown. Further external bus activity, either external cycles or reflection of internal cycles, will commence during the next low phase of **ProcClockOut**.

Chip enable, write enables, address bus and data bus are in a high impedance state during DMA. External circuitry must ensure that **notMemCE** and **notMemWrB0-1** do not become active whilst control is being transferred; it is recommended that a 10K resistor is connected from **VCC** to each pin. DMA cannot interrupt an external memory cycle. 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.

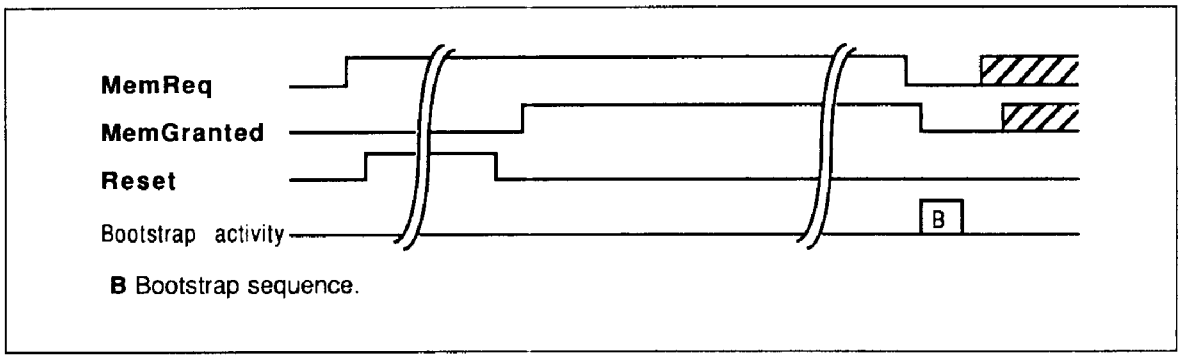


Figure 7.14 IMS T212 DMA sequence at reset

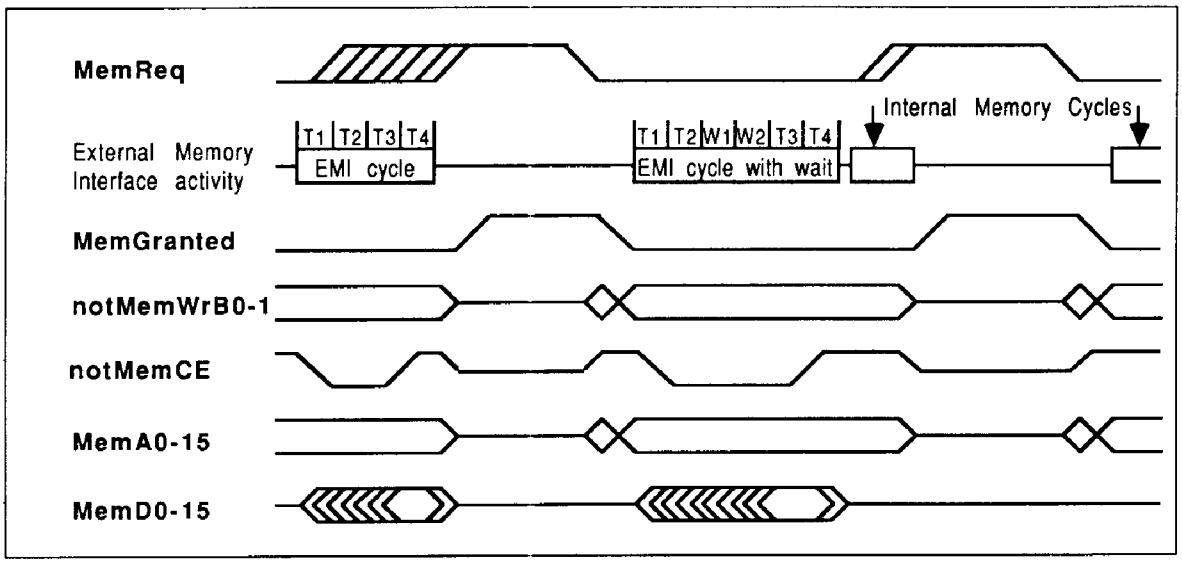


Figure 7.15 IMS T212 operation of **MemReq** and **MemGranted** with external and internal memory cycles

DMA allows a bootstrap program to be loaded into external RAM ready for execution after reset. If **MemReq** is held high throughout reset, **MemGranted** will be asserted before the bootstrap sequence begins. **MemReq** must be high at least one period **TDCLDCL** of **ClockIn** before **Reset**. The circuit should be designed to ensure correct operation if **Reset** could interrupt a normal DMA cycle.

Table 7.7 Memory request

SYMBOL	PARAMETER	T212-20		T212-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TMRHMGH	Memory request response time	85	**a**	100	**a**	ns	1
TMRLMGL	Memory request end response time	90	100	100	114	ns	
TAZMGH	Addr. bus tristate before MemGranted	0		0		ns	
TAVMGL	Addr. bus active after MemGranted end	0		0		ns	
TDZMGH	Data bus tristate before MemGranted	0		0		ns	
TEZMGH	notMemCE tristate before MemGranted	0		0		ns	
TEVMGL	notMemCE active after MemGranted end	0		0		ns	
TWEZMGH	Write enable tristate before MemGranted	0		0		ns	
TWEVMGL	Write enable active after MemGranted end	0		0		ns	

Notes

1 Maximum response time **a** depends on whether an external memory cycle is in progress and whether byte access is active. Maximum time is (2 processor cycles) + (number of wait state cycles) for word access; in byte access mode this time is doubled.

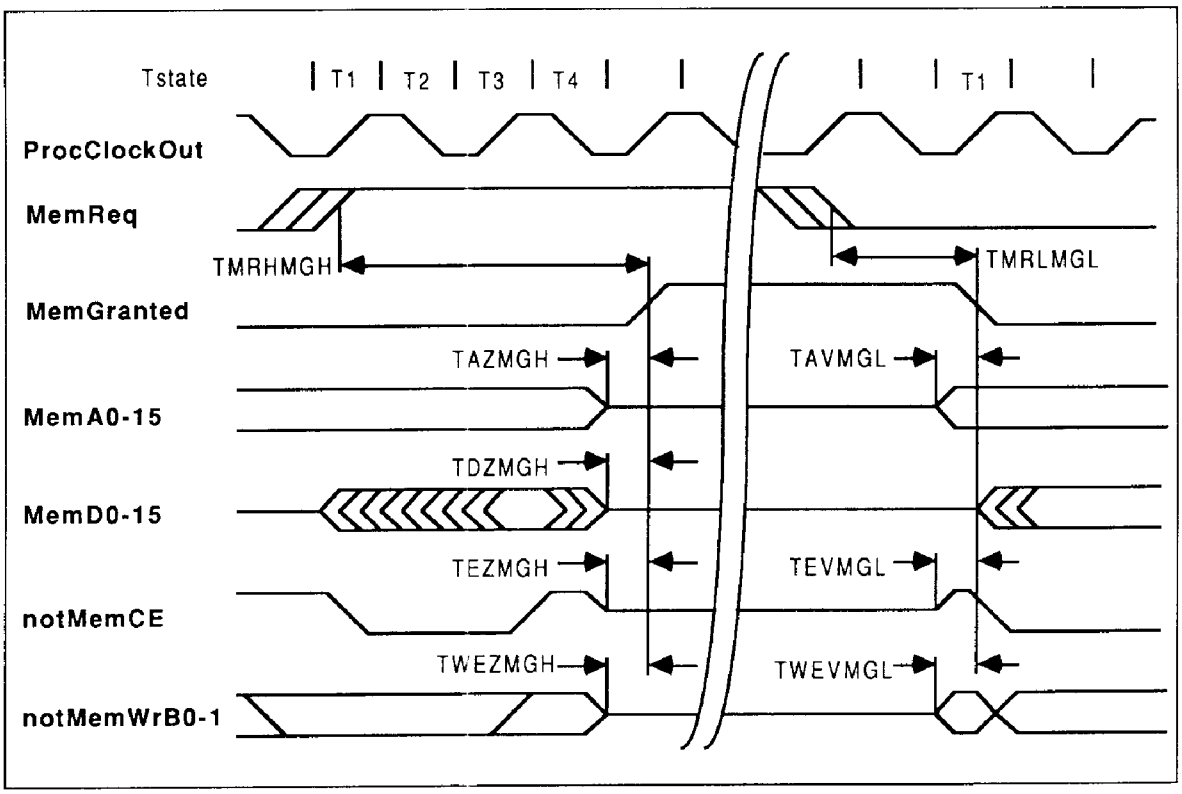


Figure 7.16 IMS T212 memory request timing

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 latency is as described on page 305. Setting a high priority task to wait for an event input is a way of interrupting a transputer program.

Table 8.1 Event

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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	
TKHEWL	Event acknowledge to end of event waiting	0			ns	
TKLEWH	End of event acknowledge to event waiting	0			ns	

Notes

1 **a** is 3 processor cycles **TPCLPCL**.

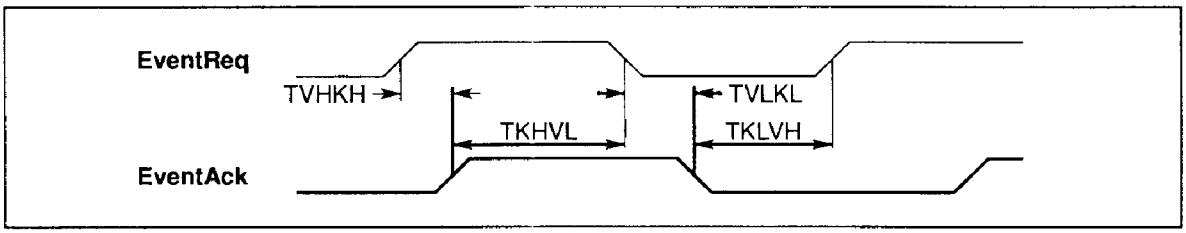


Figure 8.1 IMS T212 event timing

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.

The IMS T212 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 5 or 20 Mbits/sec. 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. Table 9.1 shows uni-directional and bi-directional data rates in Kbytes/sec 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.

Table 9.1 Speed Settings for Transputer Links

Link Special	Linkn Special	Mbits/sec	Kbytes/sec	
			Uni	Bi
0	0	10	400	800
0	1	5	200	400
1	0	10	400	800
1	1	20	800	1600

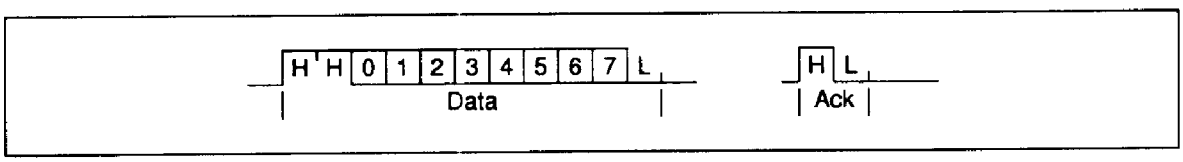


Figure 9.1 IMS T212 link data and acknowledge packets

Table 9.2 Link

SYMBOL	PARAMETER		MIN	NOM	MAX	UNITS	NOTE
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	20 Mbits/s			3	ns	1
		10 Mbits/s			10	ns	1
		5 Mbits/s			30	ns	1
CLIZ	LinkIn capacitance	@ f=1MHz			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.

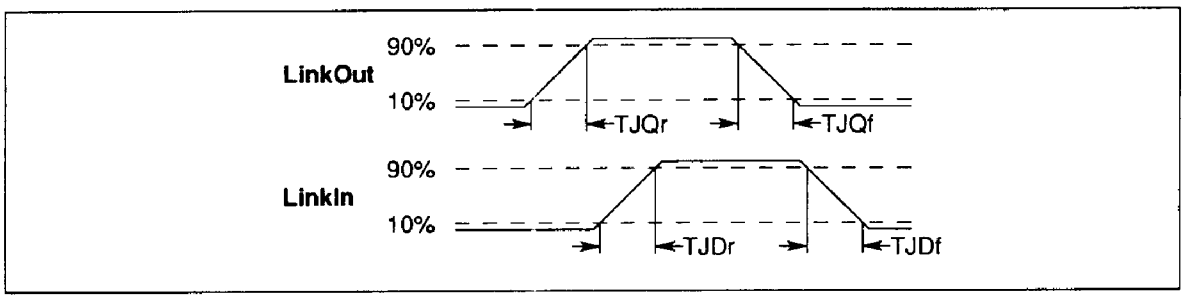


Figure 9.2 IMS T212 link timing

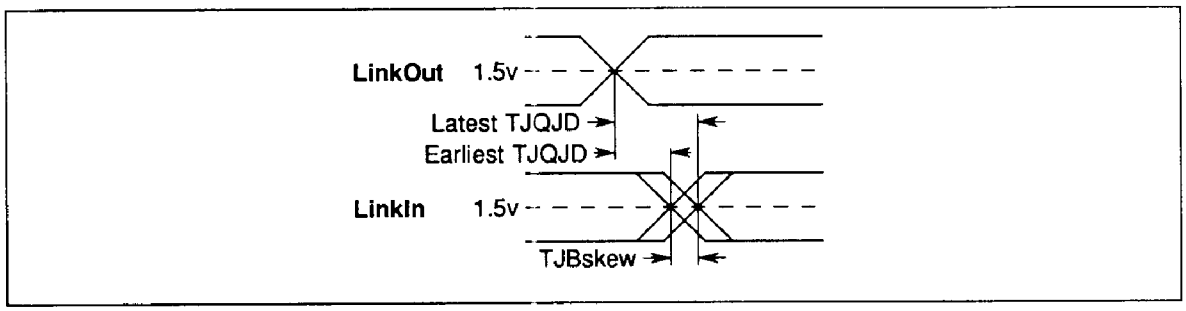


Figure 9.3 IMS T212 buffered link timing

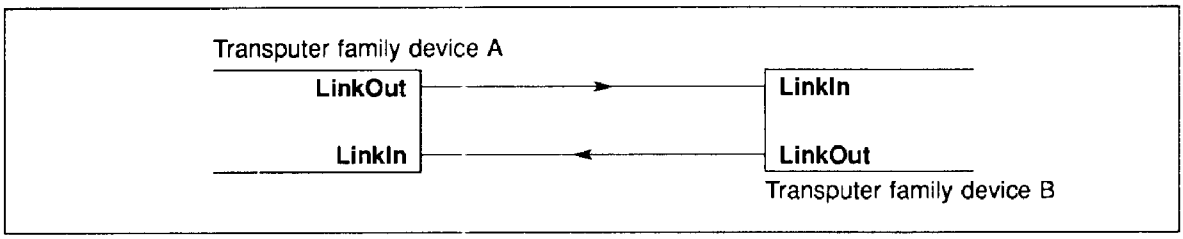


Figure 9.4 IMS T212 Links directly connected

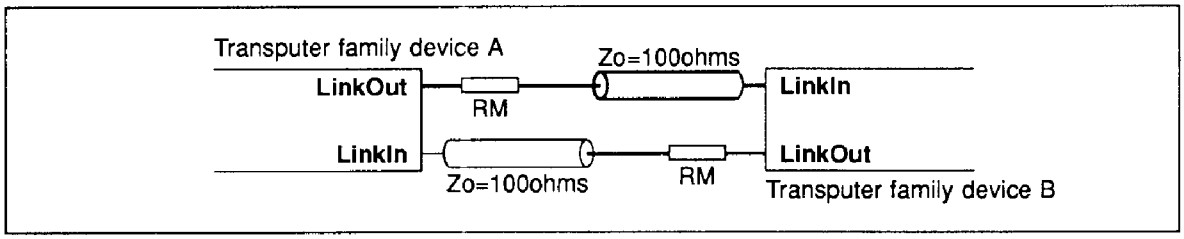


Figure 9.5 IMS T212 Links connected by transmission line

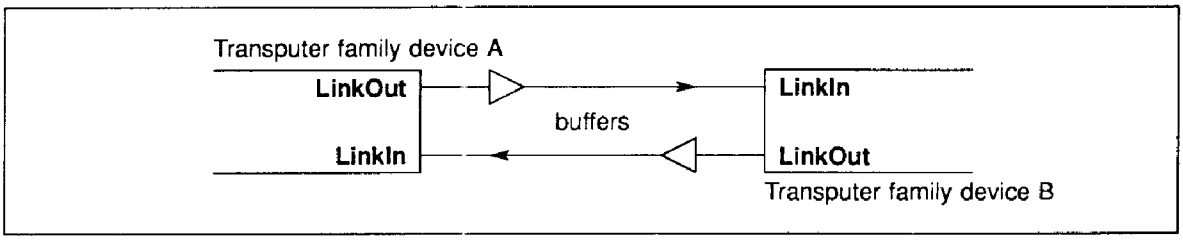


Figure 9.6 IMS T212 Links connected by buffers

10 Electrical specifications

10.1 DC electrical characteristics

Table 10.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
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
II	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. Unused inputs should be 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**.

Table 10.2 Operating conditions

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
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		60	pF	
TA	Operating temperature range	0	70	°C	3

Notes

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

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

3 Air flow rate 400 linear ft/min transverse air flow.

Table 10.3 DC characteristics

SYMBOL	PARAMETER		MIN	MAX	UNITS	NOTE
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	36	65	mA	1,2,3
			65	100	mA	1,2,4
IOZ	Tristate output current	@ GND<VO<VCC		±10	μA	1,2
PD	Power dissipation			700	mW	2,5
CIN	Input capacitance	@ f=1MHz		7	pF	6
COZ	Output capacitance	@ f=1MHz		10	pF	6

Notes

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

2 Parameters for IMS T212-S measured at 4.75V<**VCC**<5.25V and 0°C<**TA**<70°C. Input clock frequency = 5MHz.

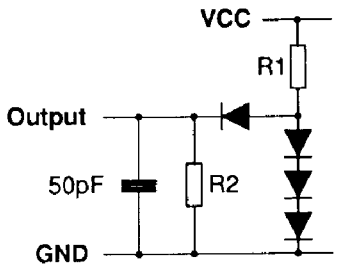
3 Current sourced from non-link outputs.

4 Current sourced from link outputs.

5 Power dissipation varies with output loading and program execution.

6 This parameter is sampled and not 100% tested.

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

Figure 10.1 Load circuit for AC measurements

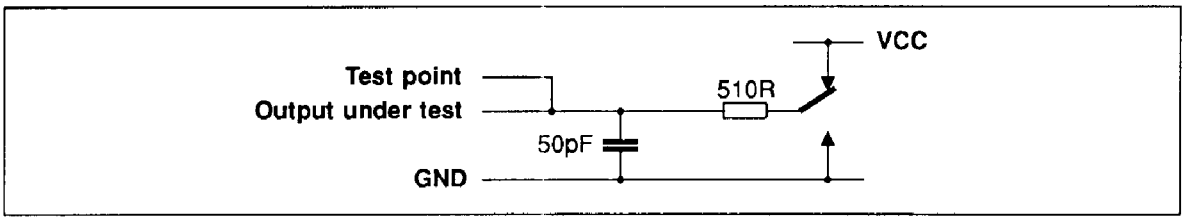


Figure 10.2 Tristate load circuit for AC measurements

10.3 AC timing characteristics

Table 10.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges		15	ns	1

Notes

1 Non-link pins; see section on links.

2 All inputs except **ClockIn**; see section on **ClockIn**.

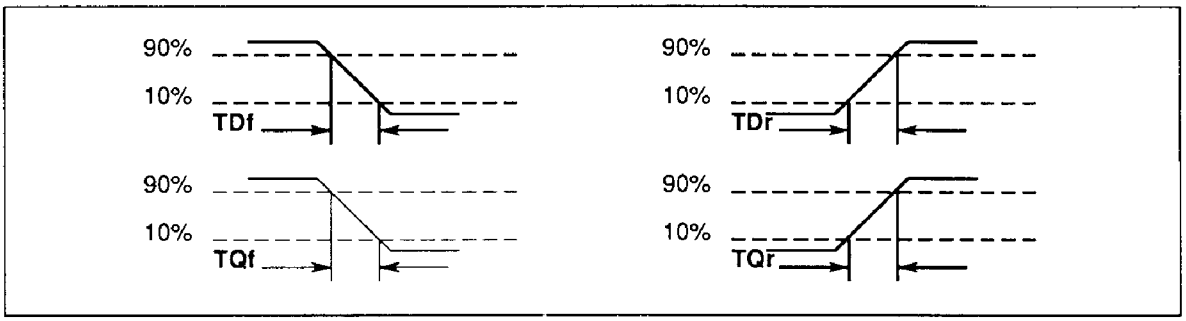


Figure 10.3 IMS T212 input and output edge timing

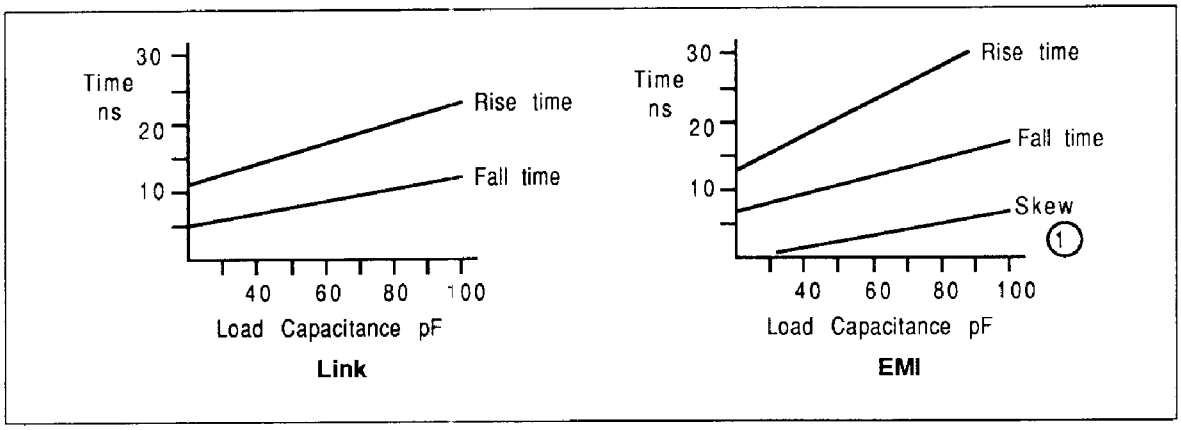


Figure 10.4 Typical rise/fall times

Notes

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

10.4 Power rating

Internal power dissipation P_{INT} of transputer and peripheral chips depends on **VCC**, as shown in figure 10.5. P_{INT} is substantially independent of temperature.

Total power dissipation P_D of the chip is

$$P_D = P_{INT} + P_{IO}$$

where P_{IO} is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature T_J of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where T_A is the external ambient temperature in °C and θJ_A is the junction-to-ambient thermal resistance in °C/W. θJ_A for each package is given in the Packaging Specifications section.

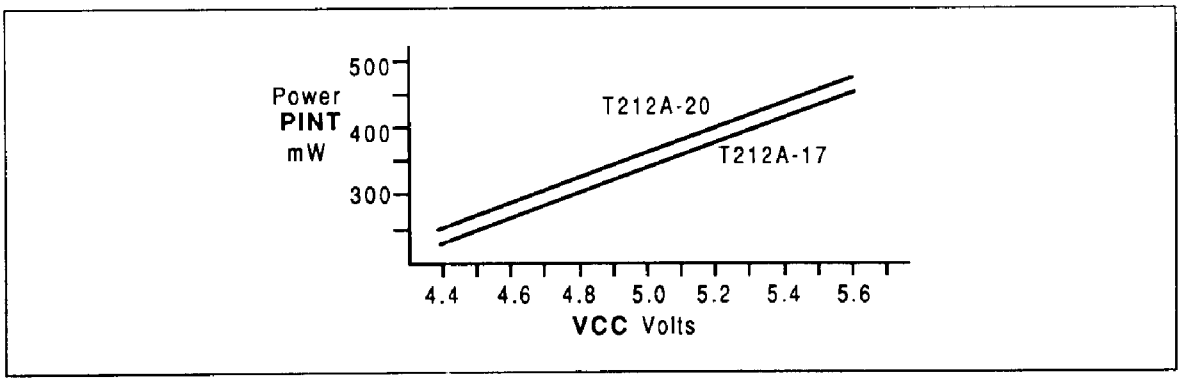


Figure 10.5 IMS T212 internal power dissipation vs VCC

11 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to occam programs. For the same function, other languages should achieve approximately the same performance as occam.

11.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 11.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 11.1 Key to performance table

np	number of component processes
ne	number of processes earlier in queue
r	1 if **INT** parameter or array parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
p	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard
Tb	most significant bit set of multiplier ((-1) if the multiplier is 0)
Tbp	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
Tbc	most significant bit set in the two's complement of a negative multiplier
nsp	Number of scalar parameters in a procedure
nap	Number of array parameters in a procedure

Table 11.2 Performance

	Size (bytes)	Time (cycles)
Names		
variables		
in expression	1.1+**r**	2.1+2(**r**)
assigned to or input to	1.1+**r**	1.1+(**r**)
in **PROC** or **FUNCTION** call, corresponding to an **INT** parameter	1.1+**r**	1.1+(**r**)
channels	1.1	2.1
Array Variables (for single dimension arrays)		
constant subscript	0	0
variable subscript	5.3	7.3
expression subscript	5.3	7.3
Declarations		
CHAN OF *protocol*	3.1	3.1
[size]CHAN OF *protocol*	9.4	2.2 + 20.2***size**
PROC	body+2	0
Primitives		
assignment	0	0
input	4	26.5
output	1	26
STOP	2	25
SKIP	0	0
Arithmetic operators		
+ -	1	1
*	2	23
/	2	24
REM	2	22
>> <<	2	3+**p**
Modulo Arithmetic operators		
PLUS	2	2
MINUS	1	1
TIMES (fast multiply)	1	4+**Tb**
Boolean operators		
OR	4	8
AND NOT	1	2
Comparison operators		
= constant	0	1
= variable	2	3
<> constant	1	3
<> variable	3	5
> <	1	2
>= <=	2	4
Bit operators		
/\ \/ >< ~	2	2
Expressions		
constant in expression	**w**	**w**
check if error	4	6

Table 11.3 Performance

	Size (bytes)	Time (cycles)
Timers		
timer input	2	3
timer **AFTER**		
if past time	2	4
with empty timer queue	2	31
non-empty timer queue	2	38+**ne***9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+**ne***9
timer alt guard	8+2**Eg**+2Et	34+2**Eg**+2Et
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	6	26
alt channel guard	10.2+2**Eg**	20+2**Eg**
skip alt guard	8+2**Eg**	10+2**Eg**
PAR	11.5+(**np**-1)*7.5	19.5+(**np**-1)*30.5
WHILE	4	12
Procedure or function call	3.5+(nsp-2)*1.1 +nap*2.3	16.5+(nsp-2)*1.1 +nap*2.3
Replicators		
replicated **SEQ**	7.3{+5.1}	(-3.8)+15.1*count{+7.1}
replicated **IF**	12.3{+5.1}	(-2.6)+19.4*count{+7.1}
replicated **ALT**	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer **ALT**	24.8{+10.2}	62.4+33.4*count{+14.2}
replicated **PAR**	39.1{+5.1}	(-6.4)+70.9*count{+7.1}

11.2 Fast multiply, TIMES

The IMS T212 has a fast integer multiplication instruction *product*. The time taken for a fast multiply is 4+**Tb**. The time taken for a multiplication by zero is 3 cycles. For example, if the multiplier is 1 the time taken is 4 cycles, if the multiplier is -1 (all bits set) the time taken is 19 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the occam modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

11.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic where relevant. In table 11.4 **n** gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the occam reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

Table 11.4 Arithmetic performance

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		18	8
LONGDIV		20	8
SHIFTRIGHT	(n<16)	4+n	8
	(n>=16)	n-11	8
SHIFTLEFT	(n<16)	4+n	8
	(n>=16)	n-11	8
NORMALISE	(n<16)	n+6	7
	(n>=16)	n-9	7
	(n=32)	4	7
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7

† Assuming local variables.

11.4 IMS T212 floating point operations

Floating point operations for the IMS T212 are provided by a run-time package. This requires approximately 2000 bytes of memory for the double length arithmetic operations, and 2500 bytes for the quadruple length arithmetic operations. Table 11.5 summarizes the estimated performance of the package.

Table 11.5 IMS T212 floating point operations performance

		Processor cycles	
		IMS T212	
		Typical	Worst
REAL32	+ -	530	705
	*	650	705
	/	1000	1410
	< > = >= <= <>	60	60
REAL64	+ -	875	1190
	*	1490	1950
	/	2355	3255
	< > = >= <= <>	60	60

11.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. The value of **e** for the IMS T212 with no wait states is 1.

If a program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at (2**e**-1)/4 per byte of program. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 11.6 **n** denotes the number of components in a construct. In the case of **IF**, the **n**'th conditional is the first to evaluate to **TRUE**, and the costs include the costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by **b**.

Table 11.6 External memory performance

	IMS T212	
	Program off chip	**Data off chip**
Boolean expressions	**e**-1	0
IF	3**en**-1	**en**
Replicated **IF**	6**en**+9e-12	(5**e**-2)**n**+6
Replicated **SEQ**	(4**e**-3)**n**+3e	(4**e**-2)**n**+3-e
PAR	4**en**	3**en**
Replicated **PAR**	(17**e**-12)**n**+9	16**en**
ALT	(4**e**-1)**n**+9**e**-4	(4**e**-1)**n**+9**e**-3
Array assignment and communication in one transputer	0	max (2**e**, **eb**)

For the IMS T212 the effective rate of INMOS links is slowed down on output from external memory by **e** cycles per word output, and on input to external memory at 10 Mbits/sec by **e**-6 cycles per word if **e**≥6.

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

Table 11.7 IMS T212 external memory performance

	Program	e=1	e=2	e=3	e=4	On chip
Program off chip	1	1.2	1.4	1.8	2.1	1
	2	1.1	1.2	1.4	1.6	1
Data off chip	1	1.2	1.5	1.8	2.1	1
	2	1.1	1.3	1.4	1.6	1
Program and data off chip	1	1.4	1.9	2.5	3.0	1
	2	1.2	1.5	1.8	2.1	1

11.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.8. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

Table 11.8 Interrupt latency

	Typical		Maximum	
	TPCLPCL	Tm	TPCLPCL	Tm
IMS T212	19		53	

12 Package specifications

12.1 68 pin grid array package

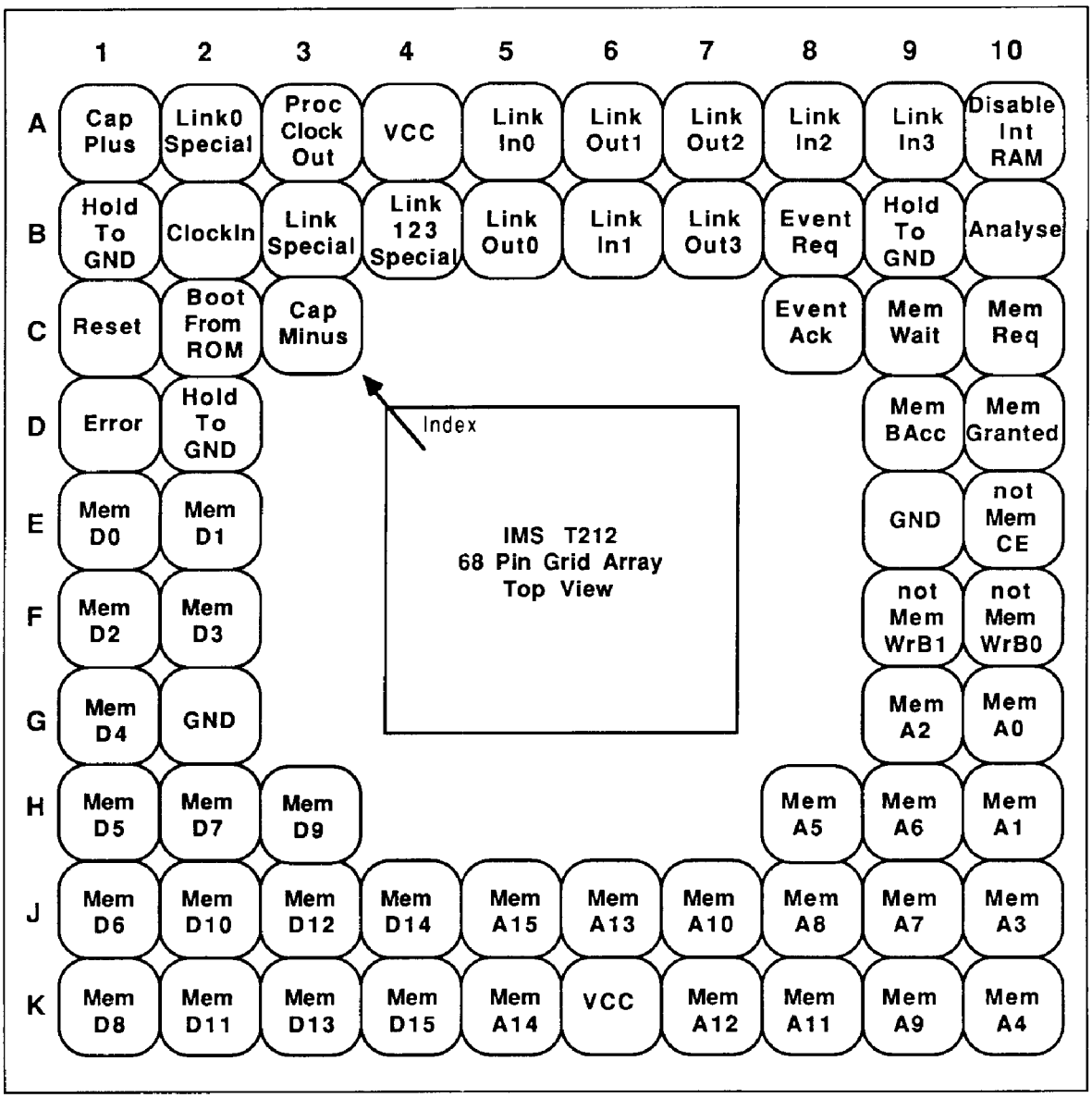


Figure 12.1 IMS T212 68 pin grid array package pinout

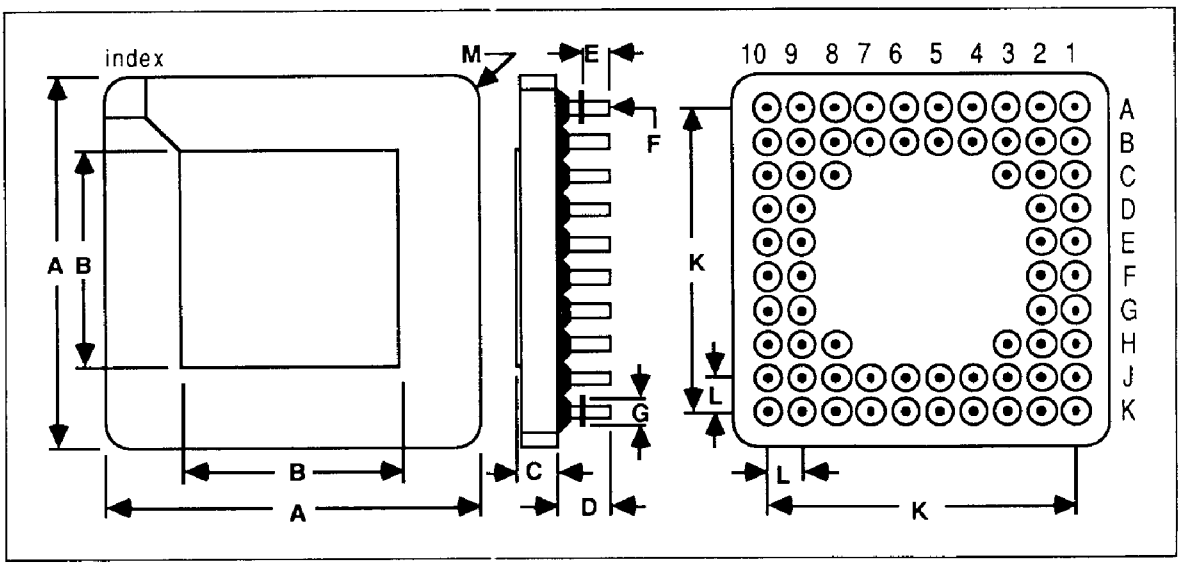


Figure 12.2 68 pin grid array package dimensions

Table 12.1 68 pin grid array package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	26.924	±0.254	1.060	±0.010	
B	17.019	±0.127	0.670	±0.008	
C	2.466	±0.279	0.097	±0.011	
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	1.270	±0.127	0.050	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
M	0.508		0.020		Chamfer

Package weight is approximately 6.8 grams

Table 12.2 68 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow			35	°C/W	

12.2 68 pin PLCC J-bend package

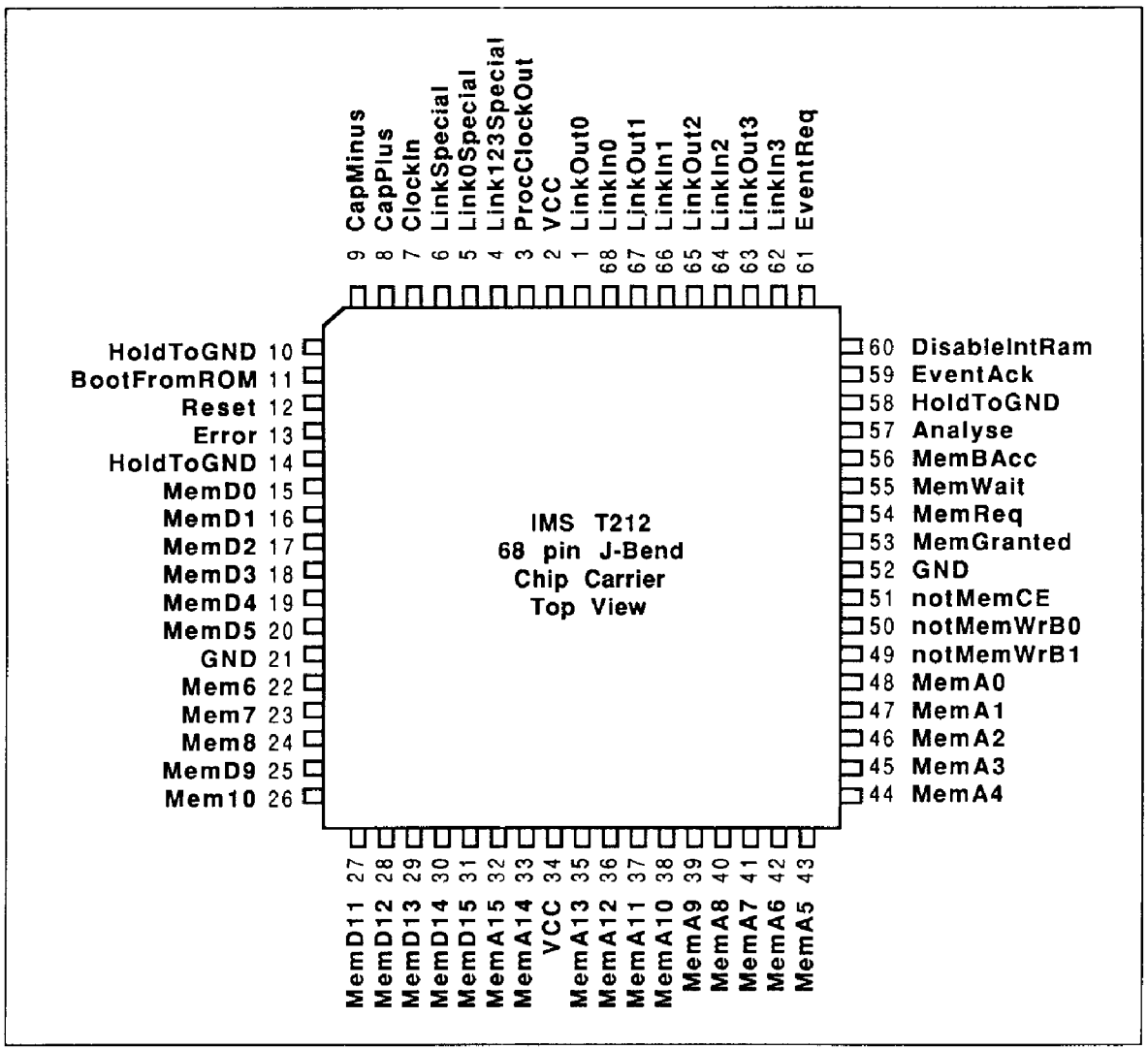


Figure 12.3 IMS T212 68 pin PLCC J-bend package pinout

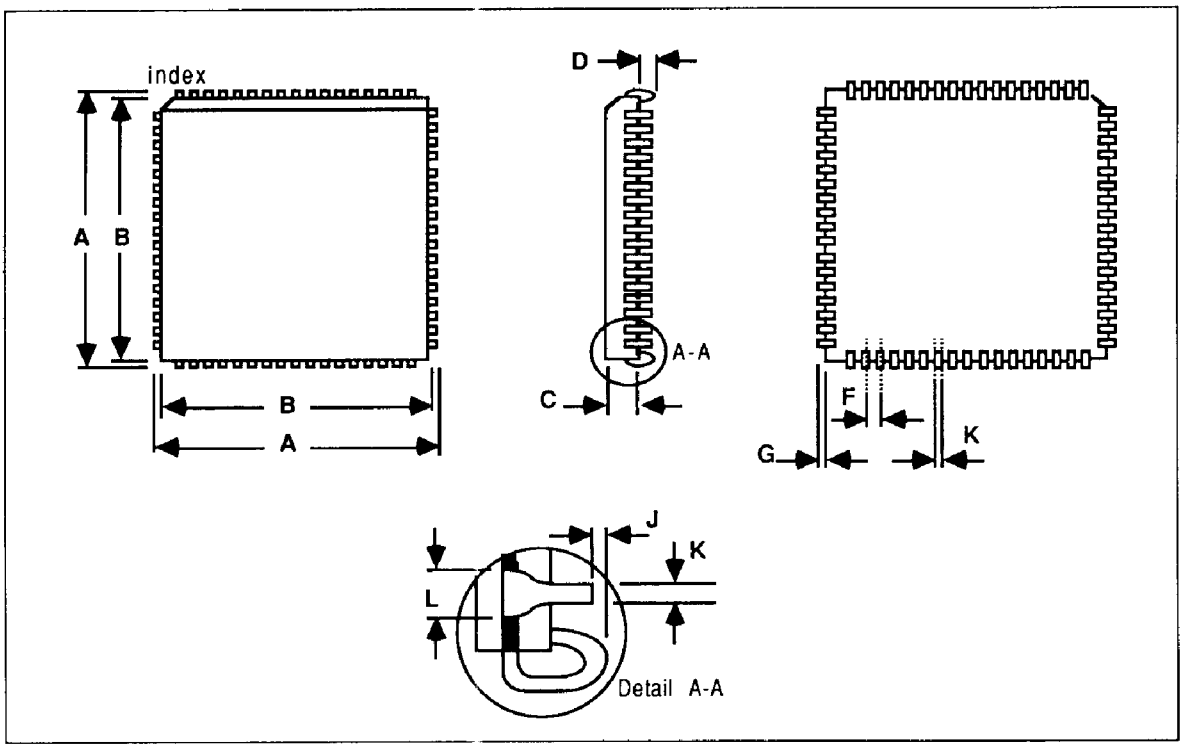


Figure 12.4 68 pin PLCC J-bend package dimensions

Table 12.3 68 pin PLCC J-bend package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	25.146	±0.127	0.990	±0.005	
B	24.232	±0.127	0.954	±0.005	
C	3.810	±0.127	0.150	±0.005	
D	0.508	±0.127	0.020	±0.005	
F	1.270	±0.127	0.050	±0.005	
G	0.457	±0.127	0.018	±0.005	
J	0.000	±0.051	0.000	±0.002	
K	0.457	±0.127	0.018	±0.005	
L	0.762	±0.127	0.030	±0.005	

Package weight is approximately 5.0 grams

Table 12.4 68 pin PLCC J-bend package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow		35		°C/W	