inmos

TRANSPUTER DATABOOK

First Edition November 1988

INMOS Databook Series

Transputer Databook
Transputer Support Databook: Development and Sub-systems
Memory Databook
Graphics Databook
Digital Signal Processing Databook

INMOS document number: 72 TRN 203 00

Printed at Bath Press Ltd, Bath

Contents

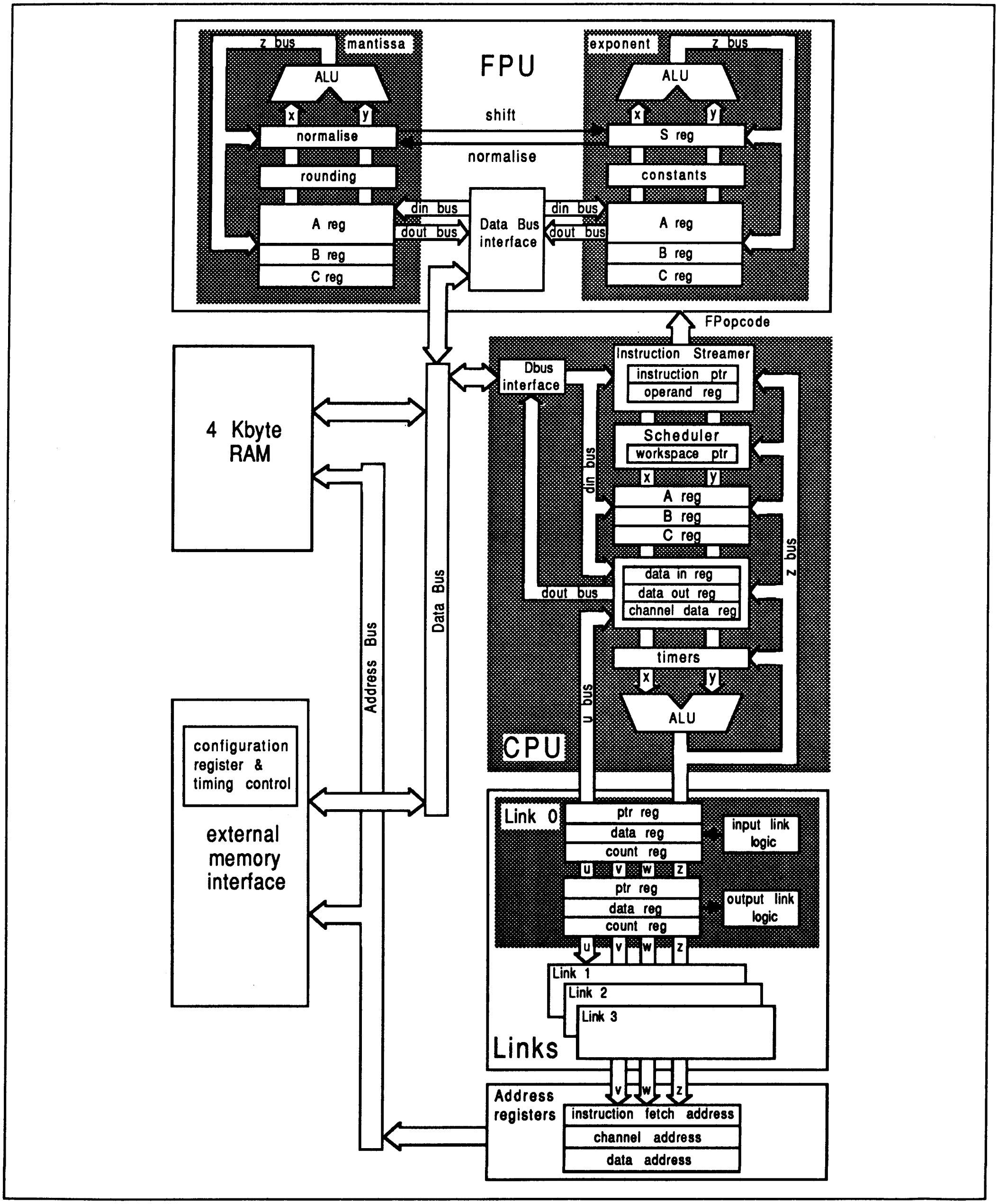


Figure 1 Transputer IMS T800 internal datapaths

Preface

This databook describes the architecture of the transputer family of products and details some of the devices which make up that family. Items described include the 32 bit and 16 bit transputer products IMS T800, IMS T425, IMS T414, IMS T212 and IMS T222; the peripheral controller IMS M212; and the communications devices IMS C004, IMS C011 and IMS C012. Details of the military version of a device are included where relevant. For further information concerning military devices contact INMOS. The military version of a device complies with MIL-STD-833C (refer to appendix A).

The databook first describes the transputer architecture and general features of transputer family devices. It then continues with the various product data sheets, followed by comparative performance details.

A transputer is a single VLSI device with processor, memory and communications links for direct connection to other transputers. An example is the IMS T800 transputer shown opposite. Concurrent systems can be constructed from a collection of transputers operating concurrently and communicating through links. The transputer can be used as a building block for concurrent processing systems, with occam as the associated design formalism.

Current transputer products include the 16 bit IMS T212 and IMS T222, the 32 bit IMS T414 and IMS T425, and the IMS T800, a 32 bit transputer with an integral high speed floating point processor.

The IMS M212 is an intelligent peripheral controller. It contains a 16 bit processor, on-chip memory and communications links. It contains hardware and interface logic to control disk drives and can be used as a programmable disk controller or as a general purpose peripheral interface.

The INMOS serial communication link is a high speed system interconnect which provides full duplex communication between members of the transputer family. It can also be used as a general purpose interconnect even where transputers are not used. The IMS C011 and IMS C012 link adaptors are communications devices enabling the INMOS serial communication link to be connected to parallel data ports and microprocessor buses. Ths IMS C004 is a programmable link switch. It provides a full crossbar switch between 32 link inputs and 32 link outputs.

The transputer development system referred to in this manual comprises an integrated editor, compiler and debugging system which enables transputers to be programmed in occam and in industry standard languages, for example, C, Fortran, Pascal. The *Transputer Development System Manual* is supplied with the transputer development system and is available as a separate publication.

Other information relevant to all transputer products is contained in the occam *Reference Manual*, supplied with INMOS software products and available as a separate publication. If more detail on the machine level operation is required, refer to *The Transputer Instruction Set - A Compiler Writers' Guide*, which is available as a separate publication.

Various application and technical notes are also available from INMOS.

Software and hardware examples given in this manual are outline design studies and are included to illustrate various ways in which transputers can be used. The examples are not intended to provide accurate application designs.

Notation and nomenclature

The nomenclature and notation in general use throughout this manual is described below.

Significance

The bits in a byte are numbered 0 to 7, with bit 0 least significant. The bytes in words are numbered from 0, with byte 0 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.

Similarly, components of arrays are numbered starting from 0 and stored in memory with component 0 at the lowest address.

Transputer memory is byte addressed, with words aligned on four-byte boundaries for 32 bit devices and on two-byte boundaries for 16 bit devices.

Hexadecimal values are prefixed with *#*, as in *#1DF*.

Where a byte is transmitted serially, it is always transmitted least significant bit (0) first. In general, wherever a value is transmitted as a number of component values, the least significant component is transmitted first. Where an array is transmitted serially, component 0 is transmitted first. Consequently, block transfers to and from memory are performed starting with the lowest (most negative) address and ending with the highest (most positive) one.

In diagrams, the least significant component of a value is to the right hand side of the diagram. Component 0 of an array is at the bottom of a diagram, as are the most negative memory locations.

Signal naming conventions

Signal names identifying individual pins of a transputer chip have been chosen to avoid being cryptic, giving as much information as possible. The majority of transputer signals are active high. Those which are active low have names commencing with **not**; names such as **RnotW** imply that the first component of the name refers to its active high state and the second to its active low state. Capitals are used to introduce new components of a name, as in **ProcClockOut**.

All transputer signals described in the text of this manual are printed in **bold**. Registers and flags internal to a device are printed in *italics*, as are instruction operation codes. *Italics* are also used for emphasis. occam program notation is printed in a `fixed space teletype` style.

References

The manual is divided into several *chapters*, each chapter having a number of *sections* and *subsections*. Figures and tables have reference numbers tied to relevant sections of a particular chapter of the manual. Unless otherwise stated, all references refer to those within the current chapter of the manual.

Transputer product numbers

All INMOS products, both memories and transputers, have a part number of the general form

IMS abbbc-xyyz

Field **a** identifies the product group. This is a digit for memory products and a letter for other devices, the particular letter indicating the type of product (table 1). Field **bbb** identifies the product within that group and field **c** is its revision code. Field **x** denotes the package type, whilst field **yy** indicates speed variants etc. The final field **z** indicates to which specification the component is qualified; standard, military etc. Where appropriate some identifiers may be omitted, depending on the device.

A typical product part would be IMS T800C-G20S.

Table 1 INMOS products

IMS 1...	Static RAM products
IMS A...	Digital signal processors
IMS B...	PC boards and modular hardware
IMS C...	Communications adaptors
IMS D...	Development system
IMS G...	Graphics products
IMS L...	Literature
IMS M...	Peripheral control transputers
IMS P...	occam programming system
IMS S...	Software product
IMS T...	Transputers

Chapter 1

transputer architecture

1 Introduction

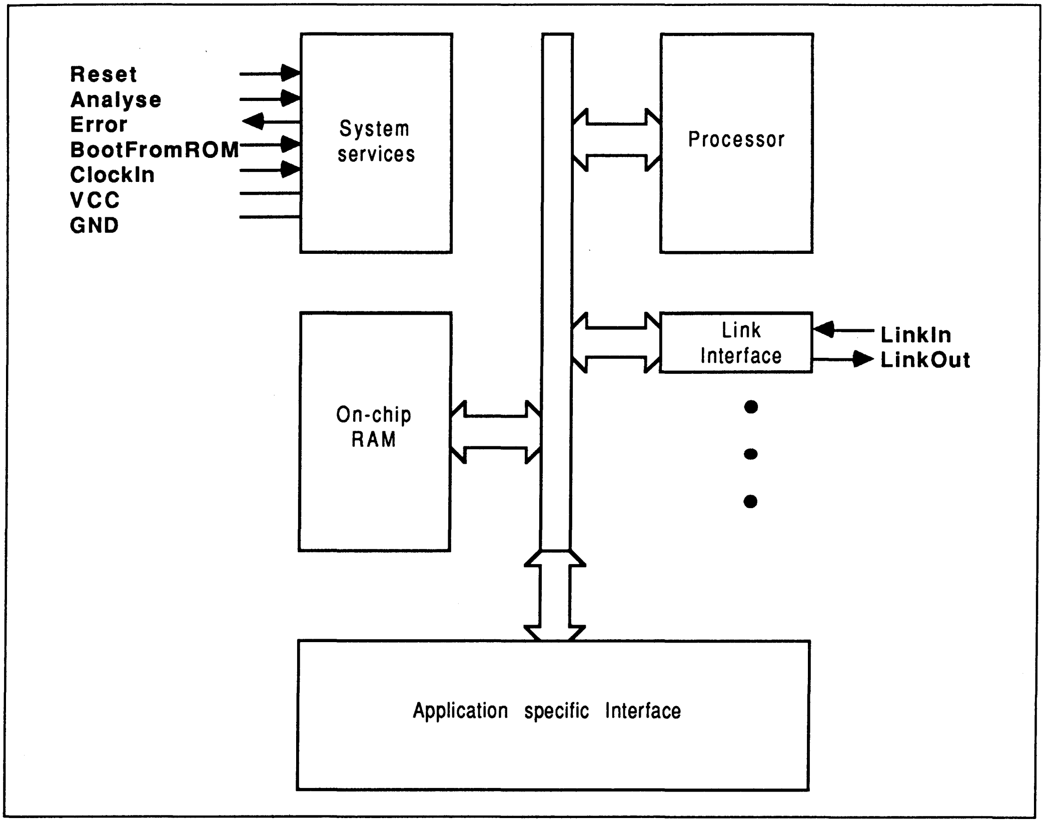


Figure 1.1 Transputer architecture

1.1 Overview

A transputer is a microcomputer with its own local memory and with links for connecting one transputer to another transputer.

The transputer architecture defines a family of programmable VLSI components. The definition of the architecture falls naturally into the *logical* aspects which define how a system of interconnected transputers is designed and programmed, and the *physical* aspects which define how transputers, as VLSI components, are interconnected and controlled.

A typical member of the transputer product family is a single chip containing processor, memory, and communication links which provide point to point connection between transputers. In addition, each transputer product contains special circuitry and interfaces adapting it to a particular use. For example, a peripheral control transputer, such as a graphics or disk controller, has interfaces tailored to the requirements of a specific device.

A transputer can be used in a single processor system or in networks to build high performance concurrent systems. A network of transputers and peripheral controllers is easily constructed using point-to-point communication.

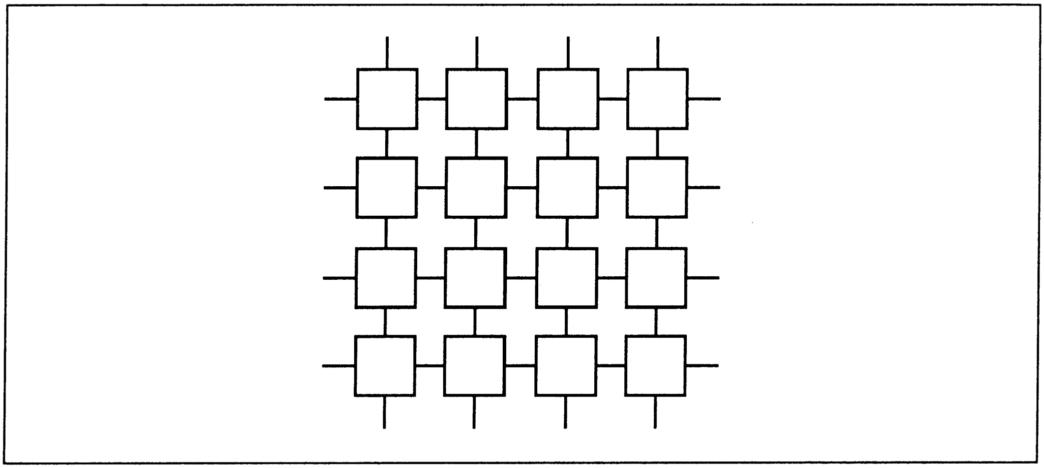


Figure 1.2 Transputer network

Transputers and occam

Transputers can be programmed in most high level languages, and are designed to ensure that compiled programs will be efficient. Where it is required to exploit concurrency, but still to use standard languages, occam can be used as a harness to link modules written in the selected languages.

To gain most benefit from the transputer architecture, the whole system can be programmed in occam (pages 8, 25). This provides all the advantages of a high level language, the maximum program efficiency and the ability to use the special features of the transputer.

occam provides a framework for designing concurrent systems using transputers in just the same way that boolean algebra provides a framework for designing electronic systems from logic gates. The system designer's task is eased because of the architectural relationship between occam and the transputer. A program running in a transputer is formally equivalent to an occam process, so that a network of transputers can be described directly as an occam program.

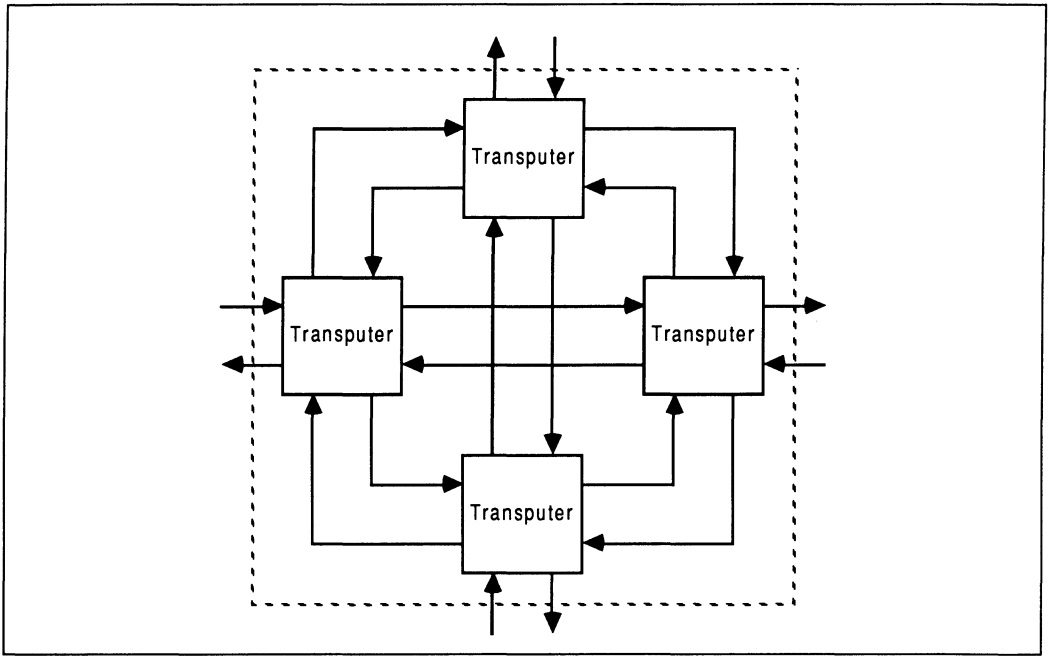


Figure 1.3 A node of four transputers

1.2 System design rationale

The transputer architecture simplifies system design by the use of processes as standard software and hardware building blocks.

An entire system can be designed and programmed in occam, from system configuration down to low level I/O and real time interrupts.

1.2.1 Programming

The software building block is the process. A system is designed in terms of an interconnected set of processes. Each process can be regarded as an independent unit of design. It communicates with other processes along point-to-point channels. Its internal design is hidden, and it is completely specified by the messages it sends and receives. Communication between processes is synchronized, removing the need for any separate synchronisation mechanism.

Internally, each process can be designed as a set of communicating processes. The system design is therefore hierarchically structured. At any level of design, the designer is concerned only with a small and manageable set of processes.

occam is based on these concepts, and provides the definition of the transputer architecture from the logical point of view (pages 8, 25).

1.2.2 Hardware

Processes can be implemented in hardware. A transputer, executing an occam program, is a hardware process. The process can be independently designed and compiled. Its internal structure is hidden and it communicates and synchronizes with other transputers via its links, which implement occam channels.

Other hardware implementations of the process are possible. For example, a transputer with a different instruction set may be used to provide a different cost/performance trade-off. Alternatively, an implementation of the process may be designed in terms of hard-wired logic for enhanced performance.

The ability to specify a hard-wired function as an occam process provides the architectural framework for transputers with specialized capabilities (e.g., graphics). The required function (e.g., a graphics drawing and display engine) is defined as an occam process, and implemented in hardware with a standard occam channel interface. It can be simulated by an occam implementation, which in turn can be used to test the application on a development system.

1.2.3 Programmable components

A transputer can be programmed to perform a specialized function, and be regarded as a 'black box' thereafter. Some processes can be hard-wired for enhanced performance.

A system, perhaps constructed on a single chip, can be built from a combination of software processes, pre-programmed transputers and hardware processes. Such a system can, itself, be regarded as a component in a larger system.

The architecture has been designed to permit a network of programmable components to have any desired topology, limited only by the number of links on each transputer. The architecture minimizes the constraints on the size of such a system, and the hierarchical structuring provided by occam simplifies the task of system design and programming.

The result is to provide new orders of magnitude of performance for any given application, which can now exploit the concurrency provided by a large number of programmable components.

1.3 Systems architecture rationale

1.3.1 Point to point communication links

The transputer architecture simplifies system design by using point to point communication links. Every member of the transputer family has one or more standard links, each of which can be connected to a link of some other component. This allows transputer networks of arbitrary size and topology to be constructed.

Point to point communication links have many advantages over multi-processor buses:

> There is no contention for the communication mechanism, regardless of the number of transputers in the system.
>
> There is no capacitive load penalty as transputers are added to a system.
>
> The communications bandwidth does not saturate as the size of the system increases. Rather, the larger the number of transputers in the system, the higher the total communications bandwidth of the system. However large the system, all the connections between transputers can be short and local.

1.3.2 Local memory

Each transputer in a system uses its own local memory. Overall memory bandwidth is proportional to the number of transputers in the system, in contrast to a large global memory, where the additional processors must share the memory bandwidth.

Because memory interfaces are not shared, and are separate from the communications interfaces, they can be individually optimized on different transputer products to provide high bandwidth with the minimum of external components.

1.4 Communication

To provide synchronised communication, each message must be acknowledged. Consequently, a link requires at least one signal wire in each direction.

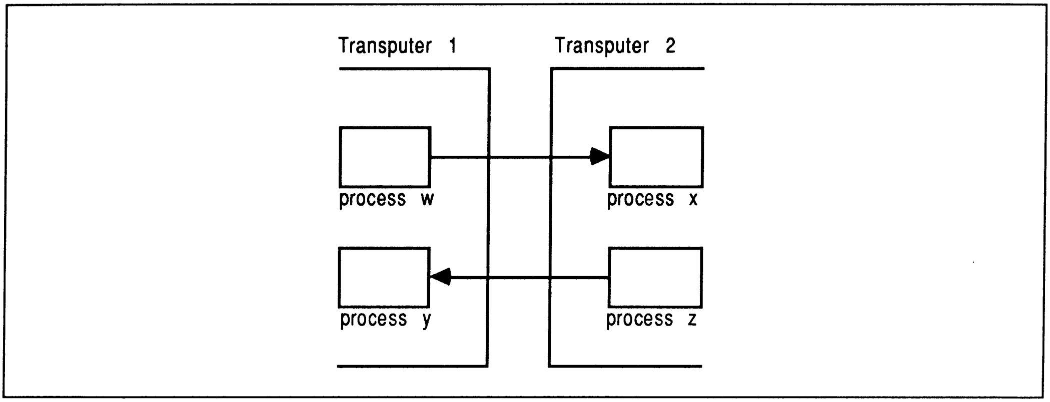


Figure 1.4 Links communicating between processes

A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer by two one-directional signal lines, along which data is transmitted serially.

The two signal wires of the link can be used to provide two occam channels, one in each direction. This requires a simple protocol. Each signal line carries data and control information.

The link protocol provides the synchronized communication of occam. The use of a protocol providing for the transmission of an arbitrary sequence of bytes allows transputers of different word length to be connected.

Each message is transmitted as a sequence of single byte communications, requiring only the presence of a single byte buffer in the receiving transputer to ensure that no information is lost. Each byte is transmitted as a start bit followed by a one bit followed by the eight data bits followed by a stop bit. After transmitting a data byte, the sender waits until an acknowledge is received; this consists of a start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged 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.

Data bytes and acknowledges are multiplexed down each signal line. An acknowledge can be transmitted as soon as reception of a data byte starts (if there is room to buffer another one). Consequently transmission may be continuous, with no delays between data bytes.

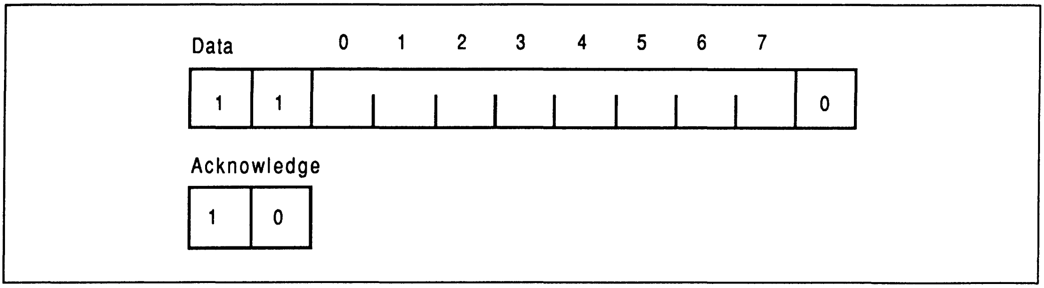


Figure 1.5 Link protocol

The links are designed to make the engineering of transputer systems straightforward. Board layout of two wire connections is easy to design and area efficient. All transputers will support a standard communications frequency of 10 Mbits/sec, regardless of processor performance. Thus transputers of different performance can be directly connected and future transputer systems will directly communicate with those of today.

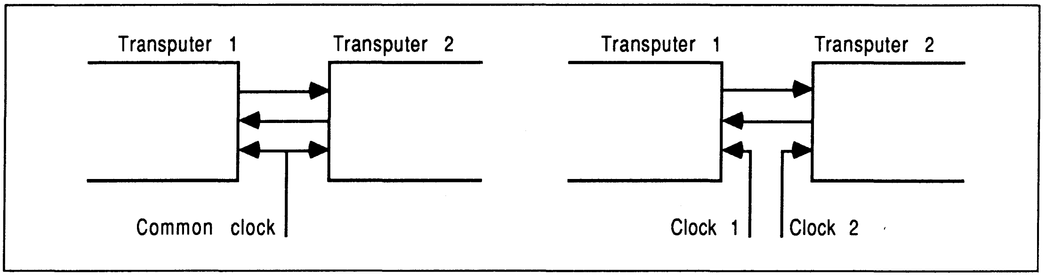


Figure 1.6 Clocking transputers

Link communication is not sensitive to clock phase. Thus, communication can be achieved between independently clocked systems as long as the communications frequency is the same.

The transputer family includes a number of link adaptor devices which provide a means of interfacing transputer links to non-transputer devices.

2 occam model

The programming model for transputers is defined by occam (page 25). The purpose of this section is to describe how to access and control the resources of transputers using occam. A more detailed description is available in the occam programming manual and the transputer development system manual (provided with the development system).

The transputer development system will enable transputers to be programmed in other industry standard languages. Where it is required to exploit concurrency, but still to use standard languages, occam can be used as a harness to link modules written in the selected languages.

2.1 Overview

In occam processes are connected to form concurrent systems. Each process can be regarded as a black box with internal state, which can communicate with other processes using point to point communication channels. Processes can be used to represent the behaviour of many things, for example, a logic gate, a microprocessor, a machine tool or an office.

The processes themselves are finite. Each process starts, performs a number of actions and then terminates. An action may be a set of sequential processes performed one after another, as in a conventional programming language, or a set of parallel processes to be performed at the same time as one another. Since a process is itself composed of processes, some of which may be executed in parallel, a process may contain any amount of internal concurrency, and this may change with time as processes start and terminate.

Ultimately, all processes are constructed from three primitive processes - assignment, input and output. An assignment computes the value of an expression and sets a variable to the value. Input and output are used for communicating between processes. A pair of concurrent processes communicate using a one way channel connecting the two processes. One process outputs a message to the channel and the other process inputs the message from the channel.

The key concept is that communication is synchronized and unbuffered. If a channel is used for input in one process, and output in another, communication takes place when both processes are ready. The value to be output is copied from the outputting process to the inputting process, and the inputting and outputting processes then proceed. Thus communication between processes is like the handshake method of communication used in hardware systems.

Since a process may have internal concurrency, it may have many input channels and output channels performing communication at the same time.

Every transputer implements the occam concepts of concurrency and communication. As a result, occam can be used to program an individual transputer or to program a network of transputers. When occam is used to program an individual transputer, the transputer shares its time between the concurrent processes and channel communication is implemented by moving data within the memory. When occam is used to program a network of transputers, each transputer executes the process allocated to it. Communication between occam processes on different transputers is implemented directly by transputer links. Thus the same occam program can be implemented on a variety of transputer configurations, with one configuration optimized for cost, another for performance, or another for an appropriate balance of cost and performance.

The transputer and occam were designed together. All transputers include special instructions and hardware to provide maximum performance and optimal implementations of the occam model of concurrency and communications.

All transputer instruction sets are designed to enable simple, direct and efficient compilation of occam. Programming of I/O, interrupts and timing is standard on all transputers and conforms to the occam model.

Different transputer variants may have different instruction sets, depending on the desired balance of cost, performance, internal concurrency and special hardware. The occam level interface will, however, remain standard across all products.

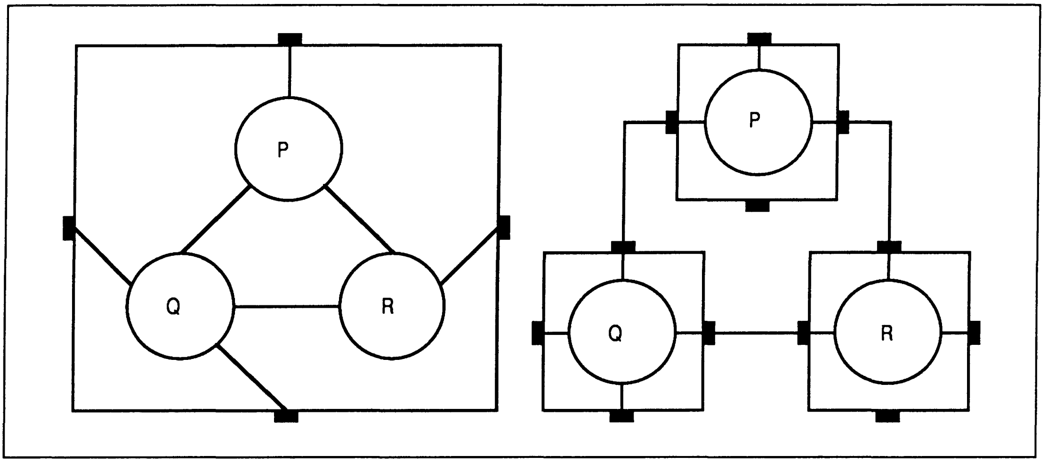


Figure 2.1 Mapping processes onto one or several transputers

2.2 occam overview

2.2.1 Processes

After it starts execution, a process performs a number of actions, and then either stops or terminates. Each action may be an assignment, an input, or an output. An assignment changes the value of a variable, an input receives a value from a channel, and an output sends a value to a channel.

At any time between its start and termination, a process may be ready to communicate on one or more of its channels. Each channel provides a one way connection between two concurrent processes; one of the processes may only output to the channel, and the other may only input from it.

Assignment

An assignment is indicated by the symbol `:=`. The example

```
v := e
```

sets the value of the variable `v` to the value of the expression `e` and then terminates, for example: `x := 0` sets `x` to zero, and `x := x + 1` increases the value of `x` by 1.

Input

An input is indicated by the symbol `?` The example

```
c ? x
```

inputs a value from the channel `c`, assigns it to the variable `x` and then terminates.

Output

An output is indicated by the symbol `!` The example

```
c ! e
```

outputs the value of the expression `e` to the channel `c`.

2.2.2 Constructions

A number of processes can be combined to form a construct. A construct is itself a process and can therefore be used as a component of another construct. Each component process of a construct is written two spaces further from the left hand margin, to indicate that it is part of the construct. There are four classes of constructs namely the sequential, parallel, conditional and the alternative construct.

Sequence

A sequential construct is represented by

```
SEQ
  P1
  P2
  P3
  ...
```

The component processes **P1**, **P2**, **P3** ... are executed one after another. Each component process starts after the previous one terminates and the construct terminates after the last component process terminates. For example

```
SEQ
  c1 ?  x
  x  := x + 1
  c2 !  x
```

inputs a value, adds one to it, and then outputs the result.

Sequential constructs in occam are similar to programs written in conventional programming languages. Note, however, that they provide the performance and efficiency equivalent to that of an assembler for a conventional microprocessor.

Parallel

A parallel construct is represented by

```
PAR
  P1
  P2
  P3
  ...
```

The component processes **P1**, **P2**, **P3** ... are executed together, and are called concurrent processes. The construct terminates after all of the component processes have terminated, for example:

```
PAR
  c1 ? x
  c2 ! y
```

allows the communications on channels **c1** and **c2** to take place together.

The parallel construct is unique to occam. It provides a straightforward way of writing programs which directly reflects the concurrency inherent in real systems. The implementation of parallelism on a single transputer is highly optimized so as to incur minimal process scheduling overhead.

Communication

Concurrent processes communicate only by using channels, and communication is synchronized. If a channel is used for input in one process, and output in another, communication takes place when both the inputting and the outputting processes are ready. The value to be output is copied from the outputting process to the inputting process, and the processes then proceed.

Communication between processes on a single transputer is via memory-to-memory data transfer. Between processes on different transputers it is via standard links. In either case the occam program is identical.

Conditional

A conditional construct

```
IF
  condition1
    P1
  condition2
    P2
  ...
```

means that `P1` is executed if `condition1` is true, otherwise `P2` is executed if `condition2` is true, and so on. Only one of the processes is executed, and then the construct terminates, for example:

```
IF
  x = 0
    y := y + 1
  x <> 0
    SKIP
```

increases `y` only if the value of `x` is 0.

Alternation

An alternative construct

```
ALT
  input1
    P1
  input2
    P2
  input3
    P3
  ...
```

waits until one of `input1`, `input2`, `input3` ... is ready. If `input1` first becomes ready, `input1` is performed, and then process `P1` is executed. Similarly, if `input2` first becomes ready, `input2` is performed, and then process `P2` is executed. Only one of the inputs is performed, then its corresponding process is executed and then the construct terminates, for example:

```
ALT
  count ? signal
    counter := counter + 1
  total ? signal
    SEQ
      out ! counter
      counter := 0
```

either inputs a signal from the channel `count`, and increases the variable `counter` by 1, or alternatively inputs from the channel `total`, outputs the current value of the counter, then resets it to zero.

The **ALT** construct provides a formal language method of handling external and internal events that must be handled by assembly level interrupt programming in conventional microprocessors.

Loop

```
WHILE condition
  P
```

repeatedly executes the process **P** until the value of the condition is false, for example:

```
WHILE (x - 5) > 0
  x := x - 5
```

leaves **x** holding the value of (**x** remainder 5) if **x** were positive.

Selection

A selection construct

```
CASE s
  n
    P1
  m,q
    P2
  ...
```

means that **P1** is executed if **s** has the same value as **n**, otherwise **P2** is executed if **s** has the same value as **m** or **q**, and so on, for example:

```
CASE direction
  up
    x := x + 1
  down
    x := x - 1
```

increases the value of **x** if **direction** is equal to **up**, otherwise if **direction** is equal to **down** the value of **x** is decreased.

Replication

A replicator is used with a **SEQ**, **PAR**, **IF** or **ALT** construction to replicate the component process a number of times. For example, a replicator can be used with **SEQ** to provide a conventional loop.

```
SEQ i = 0 FOR n
  P
```

causes the process **P** to be executed **n** times.

A replicator may be used with **PAR** to construct an array of concurrent processes.

```
PAR i = 0 FOR n
  Pi
```

constructs an array of **n** similar processes **P0**, **P1**, ..., **Pn-1**. The index **i** takes the values 0, 1, ..., n-1, in **P0**, **P1**, ..., **Pn-1** respectively.

2.2.3 Types

Every variable, expression and value has a type, which may be a primitive type, array type, record type or variant type. The type defines the length and interpretation of data.

All implementations provide the primitive types shown in table 2.1.

Table 2.1 Types

`CHAN OF` *protocol*	Each communication channel provides communication between two concurrent processes. Each channel is of a type which allows communication of data according to the specified protocol.
`TIMER`	Each timer provides a clock which can be used by any number of concurrent processes.
`BOOL`	The values of type `BOOL` are true and false.
`BYTE`	The values of type `BYTE` are unsigned numbers $\mathbf{n}$ in the range $0 <= \mathbf{n} < 256$.
`INT`	Signed integers $\mathbf{n}$ in the range $-2^{31} <= \mathbf{n} < 2^{31}$.
`INT16`	Signed integers $\mathbf{n}$ in the range $-2^{15} <= \mathbf{n} < 2^{15}$.
`INT32`	Signed integers $\mathbf{n}$ in the range $-2^{31} <= \mathbf{n} < 2^{31}$.
`INT64`	Signed integers $\mathbf{n}$ in the range $-2^{63} <= \mathbf{n} < 2^{63}$.
`REAL32`	Floating point numbers stored using a sign bit, 8 bit exponent and 23 bit fraction in ANSI/IEEE Standard 754-1985 representation.
`REAL64`	Floating point numbers stored using a sign bit, 11 bit exponent and 52 bit fraction in ANSI/IEEE Standard 754-1985 representation.

2.2.4 Declarations, arrays and subscripts

A declaration `T x` declares `x` as a new channel, variable, timer or array of type `T`, for example:

```
INT x:
P
```

declares `x` as an integer variable for use in process `P`.

Array types are constructed from component types. For example `[ n ] T` is an array type constructed from `n` components of type `T`.

A component of an array may be selected by subscription, for example `v[e]` selects the `e`'th component of `v`.

A set of components of an array may be selected by subscription, for example `[v FROM e FOR c]` selects the `c` components `v[e], v[e + 1], ... v[e + c - 1]`. A set of components of an array may be assigned, input or output.

2.2.5 Procedures

A process may be given a name, for example:

```
PROC square (INT n)
  n := n * n
:
```

defines the procedure **square**. The name may be used as an instance of the process, for example:

```
square (x)
```

is equivalent to

```
n IS x:
n := n * n
```

2.2.6 Functions

A function can be defined in the same way as a procedure. For example:

```
INT FUNCTION factorial (VAL INT n)
  INT product:
  VALOF
    IF
      n >= 0
        SEQ
          product := 1
          SEQ i = 1 FOR n
            product := product * i
    RESULT product
:
```

defines the function **factorial**, which may appear in expressions such as

```
m := factorial (6)
```

2.2.7 Expressions

An expression is constructed from the operators given in table 2.2, from variables, numbers, the truth values **TRUE** and **FALSE**, and the brackets **(** and **)**.

Table 2.2 Operators

Operator	Operand types	Description
+ - * / REM	integer, real	arithmetic operators
PLUS MINUS TIMES AFTER	integer	modulo arithmetic
= <>	any primitive	relational operators
> < >= <=	integer, real	relational operators
AND OR NOT	boolean	boolean operators
/\ \/ >< ~	integers	bitwise operators: and, or, xor, not
<< >>	integer	shift operators

For example, the expression

```
(5 + 7) / 2
```

evaluates to 6, and the expression

```
(#1DF /\ #F0) >> 4
```

evaluates to `#D` (the character `#` introduces a hexadecimal constant).

A string is represented as a sequence of ASCII characters, enclosed in double quotation marks ". If the string has `n` characters, then it is an array of type `[n]BYTE`.

2.2.8 Timer

All transputers incorporate a timer. The implementation directly supports the occam model of time. Each process can have its own independent timer, which can be used for internal measurement or for real time scheduling.

A timer input sets a variable to a value of type `INT` representing the time. The value is derived from a clock, which changes at regular intervals, for example:

```
tim ? v
```

sets the variable `v` to the current value of a free running clock, declared as the timer `tim`.

A delayed input takes the following form

```
tim ? AFTER e
```

A delayed input is unable to proceed until the value of the timer satisfies (*timer* `AFTER` *e*). The comparison performed is a modulo comparison. This provides the effect that, starting at any point in the timer's cycle, the previous half cycle of the timer is considered as being before the current time, and the next half cycle is considered as being after the current time.

2.2.9 Peripheral access

The implementation of occam provides for peripheral access by extending the input and output primitives with a port input/output mechanism. A port is used like an occam channel, but has the effect of transferring information to and from a block of addresses associated with a peripheral.

Ports behave like occam channels in that only one process may input from a port, and only one process may output to a port. Thus ports provide a secure method of accessing external memory mapped status registers etc.

Note that there is no synchronization mechanism associated with port input and output. Any timing constraints which result from the use of asynchronous external hardware will have to be programmed explicitly. For example, a value read by a port input may depend upon the time at which the input was executed, and inputting at an invalid time would produce unusable data.

During applications development it is recommended that the peripheral is modelled by an occam process connected via channels.

2.3 Configuration

occam programs may be configured for execution on one or many transputers. The transputer development system provides the necessary tools for correctly distributing a program configured for many transputers.

Configuration does not affect the logical behaviour of a program (see section four, Program development). However, it does enable the program to be arranged to ensure that performance requirements are met.

`PLACED PAR`

A parallel construct may be configured for a network of transputers by using the **`PLACED PAR`** construct. Each component process (termed a placement) is executed by a separate transputer. The variables and timers used in a placement must be declared within each placement process.

`PRI PAR`

On any individual transputer, the outermost parallel construct may be configured to prioritize its components. Each process is executed at a separate priority. The first process has the highest priority, the last process has the lowest priority. Lower priority components may only proceed when all higher priority components are unable to proceed.

2.3.1 INMOS standard links

Each link provides one channel in each direction between two transputers.

A channel (which must already have been declared) is associated with a link by a channel association, for example:

```
        PLACE Link0Input AT 4 :
```

3 Error handling

Errors in occam programs are either detected by the compiler or can be handled at runtime in one of three ways.

1 Cause the process to **STOP** allowing other processes to continue.

2 Cause the whole system to halt.

3 Have an arbitrary (undefined) effect.

The occam process **STOP** starts but never terminates. In method **1**, an errant process stops and in particular cannot communicate erroneous data to other processes. Other processes will continue to execute until they become dependent on data from the stopped process. It is therefore possible, for example, to write a process which uses a timeout to warn of a stopped process, or to construct a redundant system in which several processes performing the same task are used to enable the system to continue after one of them has failed.

Method **1** is the preferred method of executing a program.

Method **2** is useful for program development and can be used to bring transputers to an immediate halt, preventing execution of further instructions. The transputer **Error** output can be used to inform the transputer development system that such an error has occurred. No variable local to the process can be overwritten with erroneous data, facilitating analysis of the program and data which gave rise to the error.

Method **3** is useful only for optimising programs which are known to be correct!

When a system has stopped or halted as a result of an error, the state of all transputers in the system can be analysed using the transputer development system.

For languages other than occam, the transputer provides facilities for handling individual errors by software.

4 Program development

The development of programs for multiple processor systems can involve experimentation. In some cases, the most effective configuration is not always clear until a substantial amount of work has been done. For this reason, it is desirable that most of the design and programming can be completed before hardware construction is started.

4.1 Logical behaviour

An important property of occam in this context is that it provides a clear notion of 'logical behaviour'; this relates to those aspects of a program not affected by real time effects.

It is guaranteed that the logical behaviour of a program is not altered by the way in which the processes are mapped onto processors, or by the speed of processing and communication. Consequently a program ultimately intended for a network of transputers can be compiled, executed and tested on a single computer used for program development.

Even if the application uses only a single transputer, the program can be designed as a set of concurrent processes which could run on a number of transputers. This design style follows the best traditions of structured programming; the processes operate completely independently on their own variables except where they explicitly interact, via channels. The set of concurrent processes can run on a single transputer or, for a higher performance product, the processes can be partitioned amongst a number of transputers.

It is necessary to ensure, on the development system, that the logical behaviour satisfies the application requirements. The only ways in which one execution of a program can differ from another in functional terms result from dependencies upon input data and the selection of components of an **ALT**. Thus a simple method of ensuring that the application can be distributed to achieve any desired performance is to design the program to behave 'correctly' regardless of input data and **ALT** selection.

4.2 Performance measurement

Performance information is useful to gauge overall throughput of an application, and has to be considered carefully in applications with real time constraints.

Prior to running in the target environment, an occam program should be relatively mature, and indeed should be correct except for interactions which do not obey the occam synchronization rules. These are precisely the external interactions of the program where the world will not wait to communicate with an occam process which is not ready. Thus the set of interactions that need to be tested within the target environment are well identified.

Because, in occam, every program is a process, it is extremely easy to add monitor processes or simulation processes to represent parts of the real time environment, and then to simulate and monitor the anticipated real time interactions. The occam concept of time and its implementation in the transputer is important. Every process can have an independent timer enabling, for example, all the real time interactions to be modelled by separate processes and any time dependent features to be simulated.

4.3 Separate compilation of occam and other languages

A program portion which is separately compiled, and possibly written in a language other than occam, may be executed on a single transputer.

If the program is written in occam, then it takes the form of a single **PROC**, with only channel parameters. If the program is written in a language other than occam, then a run-time system is provided which provides input/output to occam channels.

Such separately compiled program portions are linked together by a framework of channels, termed a harness. The harness is written in occam. It includes all configuration information, and in particular specifies the transputer configuration in which the separately compiled program portion is executed.

Transputers are designed to allow efficient implementations of high level languages, such as C, Pascal and Fortran. Such languages will be available in addition to occam.

At runtime, a program written in such a language is treated as a single occam process. Facilities are provided in the implementations of these languages to allow such a program to communicate on occam channels. It can thus communicate with other such programs, or with programs written in occam. These programs may reside on the same transputer, in which case the channels are implemented in store, or may reside on different transputers, in which case the channels are implemented by transputer links.

It is therefore possible to implement occam processes in conventional high level languages, and arrange for them to communicate. It is possible for different parts of the same application to be implemented in different high level languages.

The standard input and output facilities provided within these languages are implemented by a well-defined protocol of communications on occam channels.

The development system provides facilities for management of separately compiled occam.

4.4 Memory map and placement

The low level memory model is of a signed address space.

Memory is byte addressed, the lowest addressed byte occupying the least significant byte position within the word.

The implementation of occam supports the allocation of the code and data areas of an occam process to specific areas of memory. Such a process must be a separately compiled **PROC**, and must not reference any variables and timers other than those declared within it.

5 Physical architecture

5.1 INMOS serial links

5.1.1 Overview

All transputers have several links. The link protocol and electrical characteristics form a standard for all INMOS transputer and peripheral products.

All transputers support a standard link communications frequency of 10 Mbits/sec. Some devices also support other data rates. Maintaining a standard communications frequency means that devices of mixed performance and type can intercommunicate easily.

Each link consists of two unidirectional signal wires carrying both data and control bits. The link signals are TTL compatible so that their range can be easily extended by inserting buffers.

The INMOS communication links provide for communication between devices on the same printed circuit board or between printed circuit boards via a back plane. They are intended to be used in electrically quiet environments in the same way as logic signals between TTL gates.

The number of links, and any communication speeds in addition to the standard speed of 10 Mbits/sec, are given in the **product data** for each product.

5.1.2 Link electrical specification

The quiescent state of the link signals is low, for a zero. The link input signals and output signals are standard TTL compatible signals.

For correct functioning of the links the specifications for maximum variation in clock frequency between two transputers joined by a link and maximum capacitive load must be met. Each transputer product also has specified the maximum permissible variation in delay in buffering, and minimum permissible edge gradients. Details of these specifications are provided in the product data.

Provided that these specifications are met then any buffering employed may introduce an arbitrary delay into a link signal without affecting its correct operation.

5.2 System services

5.2.1 Powering up and down, running and stopping

At all times the specification of input voltages with respect to the **GND** and **VCC** pins must be met. This includes the times when the **VCC** pins are ramping to 5 V, and also while they are ramping from 5 V down to 0 V.

The system services comprise the clocks, power, and signals used for initialization.

The specification includes minimum times that **VCC** must be within specification, the input clock must be oscillating, and the **Reset** signal must be high before **Reset** goes low. These specifications ensure that internal clocks and logic have settled before the transputer starts.

When the transputer is reset the memory interface is initialised (if present and configurable).

The processor and INMOS serial links start after reset. The transputer obeys a bootstrap program which can either be in off-chip ROM or can be received from one of the links. How to specify where the bootstrap program is taken from depends upon the type of transputer being used. The program will normally load up a larger program either from ROM or from a peripheral such as a disk.

During power down, as during power up, the input and output pins must remain within specification with respect to both **GND** and **VCC**.

A software error, such as arithmetic overflow, array bounds violation or divide by zero, causes an error flag to be set in the transputer processor. The flag is directly connected to the **Error** pin. Both the flag and the pin can be ignored, or the transputer stopped. Stopping the transputer on an error means that the error cannot cause further corruption.

As well as containing the error in this way it is possible to determine the state of the transputer and its memory at the time the error occurred.

5.2.2 Clock distribution

All transputers operate from a standard 5MHz input clock. High speed clocks are derived internally from the low frequency input to avoid the problems of distributing high frequency clocks. Within limits the mark-to-space ratio, the voltage levels and the transition times are immaterial. The limits on these are given in the product data for each product. The asynchronous data reception of the links means that differences in the clock phase between chips is unimportant.

The important characteristic of the transputer's input clock is its stability, such as is provided by a crystal oscillator. An R-C oscillator is inadequate. The edges of the clock should be monotonic (without kinks), and should not undershoot below -0.5 V.

5.3 Bootstrapping from ROM or from a link

The program which is executed after reset can either reside in ROM in the transputer's address space or it can be loaded via any one of the transputer's INMOS serial links.

The transputer bootstraps from ROM by transferring control to the top two bytes in memory, which will invariably contain a backward jump into ROM.

If bootstrapping from a link, the transputer bootstraps from the first link to receive a message. The first byte of the message is the count of the number of bytes of program which follow. The program is loaded into memory starting at a product dependent location *MemStart*, and then control is transferred to this address.

Messages subsequently arriving on other links are not acknowledged until the transputer processor obeys a process which inputs from them. The loading of a network of transputers is controlled by the transputer development system, which ensures that the first message each transputer receives is the bootstrap program.

5.4 Peripheral interfacing

All transputers contain one or more INMOS serial links. Certain transputer products also have other application specific interfaces. The peripheral control transputers contain specialized interfaces to control a specific peripheral or peripheral family.

In general, a transputer based application will comprise a number of transputers which communicate using INMOS links. There are three methods of communicating with peripherals.

The first is by employing peripheral control transputers (eg for graphics or disks), in which the transputer chip connects directly to the peripheral concerned (figure 5.1). The interface to the peripheral is implemented by special purpose hardware within the transputer. The application software in the transputer is implemented as an occam process, and controls the interface via occam channels linking the processor to the special purpose hardware.

The second method is by employing link adaptors (figure 5.2). These devices convert between a link and a specialized interface. The link adaptor is connected to the link of an appropriate transputer, which contains the application designer's peripheral device handler implemented as an occam process.

The third method is by memory mapping the peripheral onto the memory bus of a transputer (figure 5.3). The peripheral is controlled by memory accesses issued as a result of **PORT** inputs and outputs. The application designer's peripheral device handler provides a standard occam channel interface to the rest of the application.

The first transputers implement an event pin which provides a simple means for an external peripheral to request attention from a transputer.

In all three methods, the peripheral driver interfaces to the rest of the application via occam channels. Consequently, a peripheral device can be simulated by an occam process. This enables testing of all aspects of a transputer system before the construction of hardware.

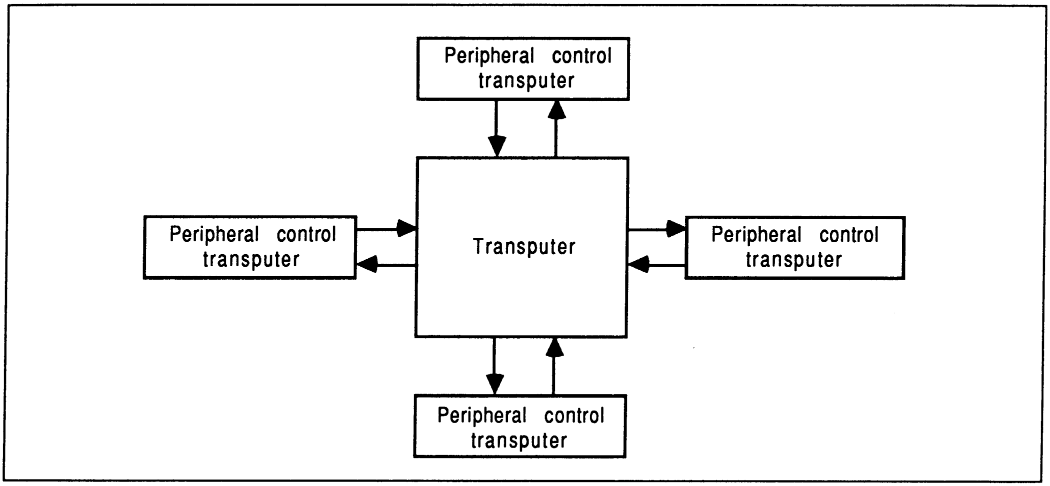


Figure 5.1 Transputer with peripheral control transputers

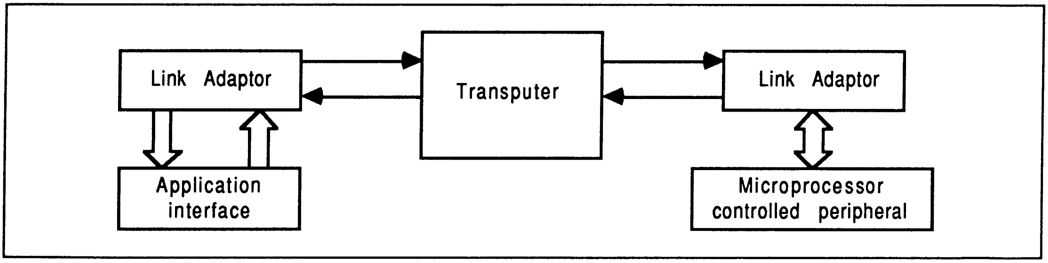


Figure 5.2 Transputer with link adaptors

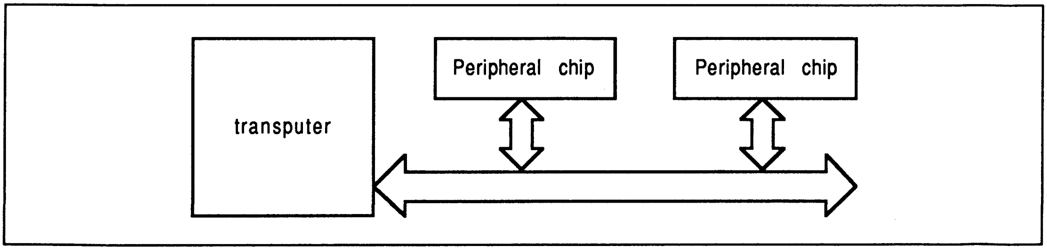


Figure 5.3 Memory mapped peripherals

transputer overview

1 Introduction

The INMOS transputer family is a range of system components each of which combines processing, memory and interconnect in a single VLSI chip. A concurrent system can be constructed from a collection of transputers which operate concurrently and communicate through serial communication links. Such systems can be designed and programmed in occam, a language based on communicating processes, and in other industry standard languages. Transputers have been sucessfully used in application areas ranging from embedded systems to supercomputers.

The first member of the family, the IMS T414 32-bit transputer (Bibliography reference *INMOS '84*), was introduced in September 1985, and has enabled concurrency to be applied in a wide variety of applications such as simulation, robot control, image synthesis, and digital signal processing. Many computationally intensive applications can exploit large arrays of transputers; the system performance depending on the number of transputers, the speed of inter-transputer communication and the performance of each transputer processor.

The power of transputer based systems lies in the smoothly scaleable performance offered by adding more transputers. The transputer embodies the concepts required for effective parallel processing.

Further transputer products are continually being developed which increase the memory, processing performance and communications performance. An important example is the floating point transputer first introduced in 1987.

For publication references used in this chapter, see page 453.

2 The transputer: basic architecture and concepts

2.1 A programmable device

The transputer is a component designed to exploit the potential of VLSI. This technology allows large numbers of *identical* devices to be manufactured cheaply. For this reason, it is attractive to implement a concurrent system using a number of identical components, each of which is customised by an appropriate program. The transputer is, therefore, a VLSI device with a processor, memory to store the program executed by the processor, and communication links for direct connection to other transputers. Transputer systems can be designed and programmed using occam which allows an application to be described as a collection of processes which operate concurrently and communicate through channels. The transputer can therefore be used as a building block for concurrent processing systems, with occam as the associated design formalism.

2.2 occam

occam enables a system to be described as a collection of concurrent processes, which communicate with each other and with peripheral devices through channels. occam programs are built from three primitive processes:

`v := e`	assign expression **`e`** to variable **`v`**
`c ! e`	output expression **`e`** to channel **`c`**
`c ? v`	input from channel **`c`** to variable **`v`**

The primitive processes are combined to form constructs:

`SEQ`uential	components executed one after another
`PAR`allel	components executed together
`ALT`ernative	component first ready is executed

A construct is itself a process, and may be used as a component of another construct.

Conventional sequential programs can be expressed with variables and assignments, combined in sequential constructs. **`IF`** and **`WHILE`** constructs are also provided.

Concurrent programs can be expressed with channels, inputs and outputs, which are combined in parallel and alternative constructs.

Each occam channel provides a communication path between two concurrent processes. Communication is synchronised and takes place when both the inputting process and the outputting process are ready. The data to be output is then copied from the outputting process to the inputting process, and both processes continue.

An alternative process may be ready for input from any one of a number of channels. In this case, the input is taken from the channel which is first used for output by another process.

2.3 VLSI technology

One important property of VLSI technology is that communication between devices is very much slower than communication within a device. In a computer, almost every operation that the processor performs involves the use of memory. For this reason a transputer includes both processor and memory in the same integrated circuit device.

In any system constructed from integrated circuit devices, much of the physical bulk arises from connections between devices. The size of the package for an integrated circuit is determined more by the number of connection pins than by the size of the device itself. In addition, connections between devices provided by paths on a circuit board consume a considerable amount of space.

The speed of communication between electronic devices is optimised by the use of one-directional signal wires, each connecting two devices. If many devices are connected by a shared bus, electrical problems of driving the bus require that the speed is reduced. Also, additional control logic and wiring are required to control sharing of the bus.

To provide maximum speed with minimal wiring, the transputer uses point-to-point serial communication links for direct connection to other transputers. The protocols used on the transputer links are discussed later.

2.4 Simplified processor with micro-coded scheduler

The most effective implementation of simple programs by a programmable computer is provided by a sequential processor. Consequently, the transputer has a fairly conventional microcoded processor. There is a small core of about 32 instructions which are used to implement simple sequential programs. In addition there are other, more specialised groups of instructions which provide facilities such as long arithmetic and process scheduling.

As a process executed by a transputer may itself consist of a number of concurrent processes the transputer has to support the occam programming model internally. The transputer, therefore, has a microcoded scheduler which shares the processor time between the concurrent processes. The scheduler provides two priority levels; any high priority process which can run will do so in preference to any low priority process.

3 Transputer internal architecture

Internally, a transputer consists of a memory, processor and communications system connected via a 32-bit bus. The bus also connects to the external memory interface, enabling additional local memory to be used. The processor, memory and communications system each occupy about 25% of the total silicon area, the remainder being used for power distribution, clock generators and external connections.

The floating point transputers each have an on-chip floating point unit. The small size and high performance of this unit come from a design which takes careful note of silicon economics. This contrasts starkly with conventional co-processors, where the floating point unit typically occupies more area than a complete microprocessor, and requires a second chip.

The block diagram 3.1 indicates the way in which the major blocks of the transputer are interconnected.

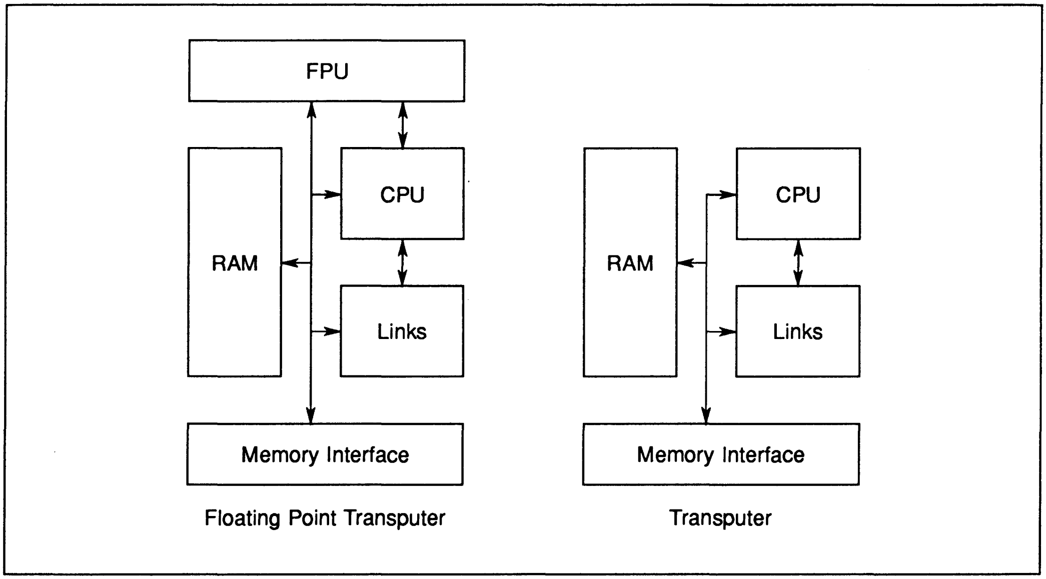


Figure 3.1 Transputer interconnections

The CPU of the transputers contains three registers (A, B and C) used for integer and address arithmetic, which form a hardware stack. 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. Similarly, the FPU includes a three register floating-point evaluation stack, containing the AF, BF, and CF registers. When values are loaded onto, or stored from the stack the AF, BF and CF registers push and pop in the same way as the A, B and C registers.

The addresses of floating point values are formed on the CPU stack, and values are transferred between the addressed memory locations and the FPU stack under the control of the CPU. As the CPU stack is used only to hold the addresses of floating point values, the wordlength of the CPU is independent of that of the FPU. Consequently, it would be possible to use the same FPU together with a 16-bit CPU.

The transputer scheduler provides two priority levels. The FPU register stack is duplicated so that when the floating point transputer switches from low to high priority none of the state in the floating point unit is written to memory. This results in a worst-case interrupt response of about 3 μs. Furthermore, the duplication of the register stack enables floating point arithmetic to be used in an interrupt routine without any performance penalty.

3.1 Sequential processing

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; the CPU contains six registers which 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, and are the 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.

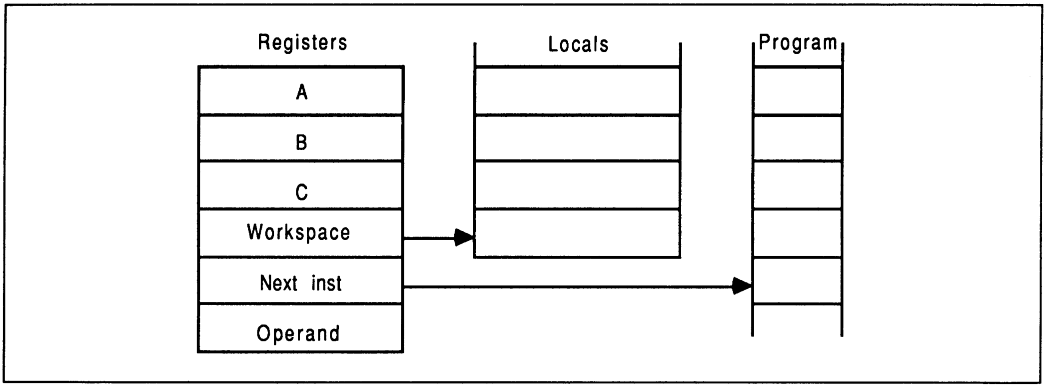


Figure 3.2 Registers

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.

3.2 Instructions

It was a design decision that the transputer should be programmed in a high-level language. The instruction set has, therefore, been designed for simple and efficient compilation. It contains a relatively small number of instructions, all with the same format, chosen to give a compact representation of the operations most frequently occuring in programs. The instruction set is independant of the processor wordlength, allowing the same microcode to be used for transputers with different wordlengths. 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.

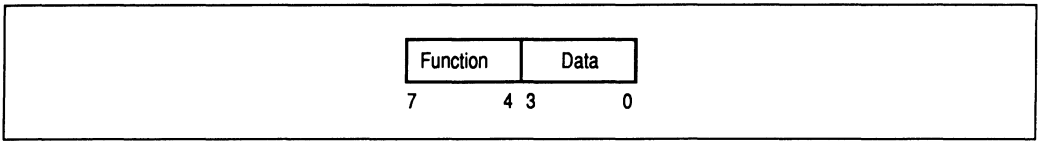


Figure 3.3 Instruction format

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 performed by any computer. These include:

load constant
load local
load non-local
jump
add constant
store local
store non-local
conditional jump
load local pointer
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 block structured programming languages such as occam.

3.2.2 Prefix functions

Two more of the function codes are used to allow the operand of any instruction to be extended in length. These are:

prefix
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 the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

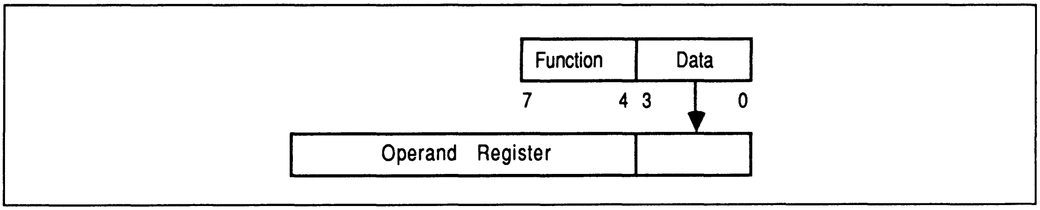


Figure 3.4 Instruction operand register

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.

The 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* *greater than*

Less frequently occuring operations have encodings which require a single prefix operation (the transputer instruction set is not large enough to require more than 512 operations to be encoded!).

The IMS T800 has additional instructions which load into, operate on, and store from, the floating point register stack. It also contains new instructions which support colour graphics, pattern recognition and the implementation of error correcting codes. These instructions have been added whilst retaining the existing IMS T414 instruction set. This has been possible because of the extensible instruction encoding used in transputers.

3.2.4 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte (ie 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 prefetch, which in turn improves processor performance. There is an extra word of prefetch buffer so that 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 Support for concurrency

The processor provides efficient support for the occam model of concurrency and communication. It 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. The processor does not need to support the dynamic allocation of storage as the occam compiler is able to perform the allocation of space to concurrent processes.

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. The active processes waiting to be executed are held on a list. This is a linked list of process workspaces, implemented using two registers, one of which points to the first process on the list, the other to the last. In figure 3.5, S is executing, and P, Q and R are active, awaiting execution.

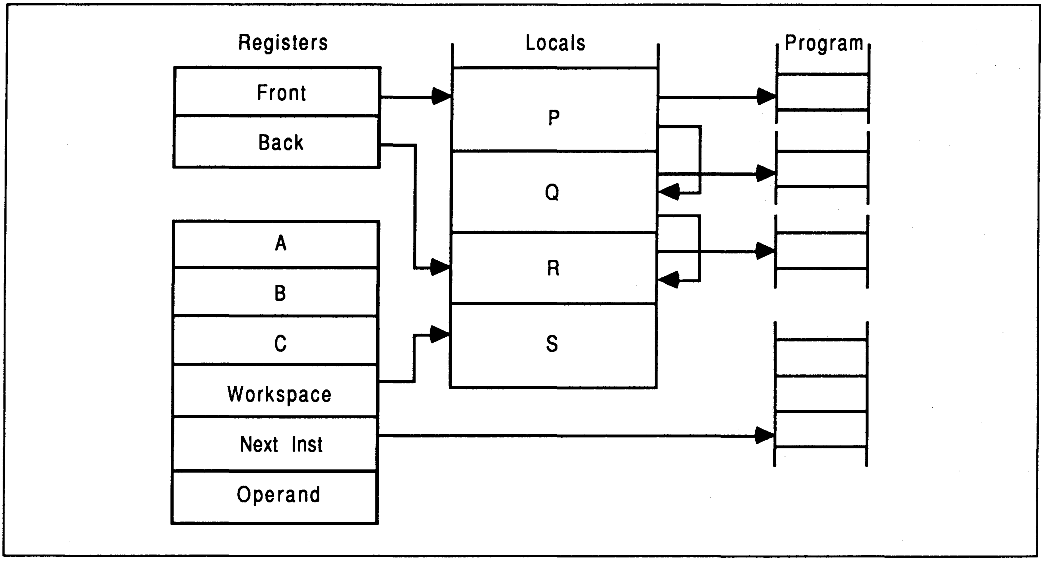


Figure 3.5 Linked process list

A process is executed until it is unable to proceed because it is waiting to input or output, or waiting for the timer. Whenever a process is unable to proceed, its instruction pointer is saved in its workspace and the next process is taken from the list. Actual process switch times are very small as little state needs to be saved; it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model. These include

start process *end process*

When a parallel construct is executed, *start process* instructions are used to create the necessary 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.

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 components of the parallel construct 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 component continues.

3.4 Communications

Communication between processes is achieved by means of channels. occam 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 *output message*

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

As in the occam model, communication takes place when both the inputting and outputting processes are ready. Consequently, 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 an *output message* instruction.

3.4.1 Internal channel communication

At any time, an internal channel (a single word in memory) either holds the identity of a process, or holds the special value *empty*. The channel is initialised to *empty* before it is used.

When a message is passed using the channel, the identity of the first process to become ready is stored in the channel, and the processor starts to execute the next process from the scheduling list. When the second process to use the channel becomes ready, the message is copied, the waiting process is added to the scheduling list, and the channel reset to its initial state. It does not matter whether the inputting or the outputting process becomes ready first.

In figure 3.6, a process P is about to execute an output instruction on an 'empty' channel C. The evaluation stack holds a pointer to a message, the address of channel C, and a count of the number of bytes in the message.

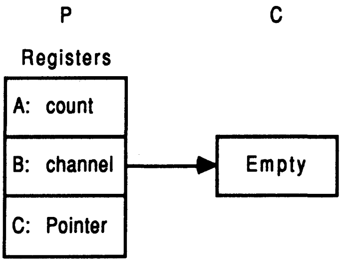


Figure 3.6 Output to empty channel

After executing the output instruction, the channel C holds the address of the workspace of P, and the address of the message to be transferred is stored in the workspace of P. P is descheduled, and the process starts to execute the next process from the scheduling list.

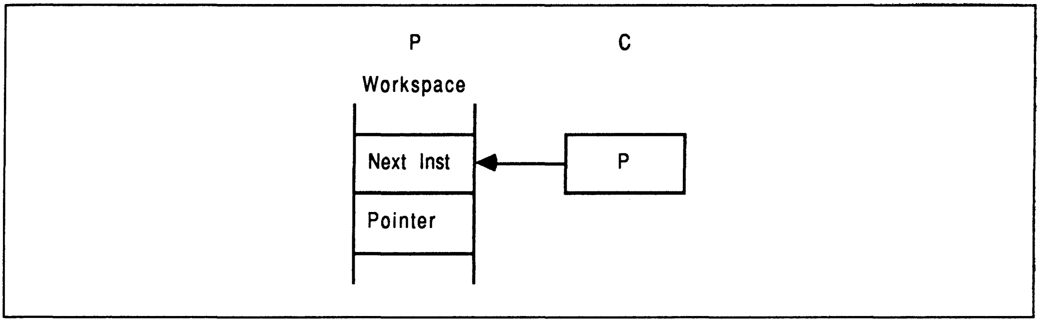


Figure 3.7

The channel C and the process P remain in this state until a second process, Q executes an output instruction on the channel.

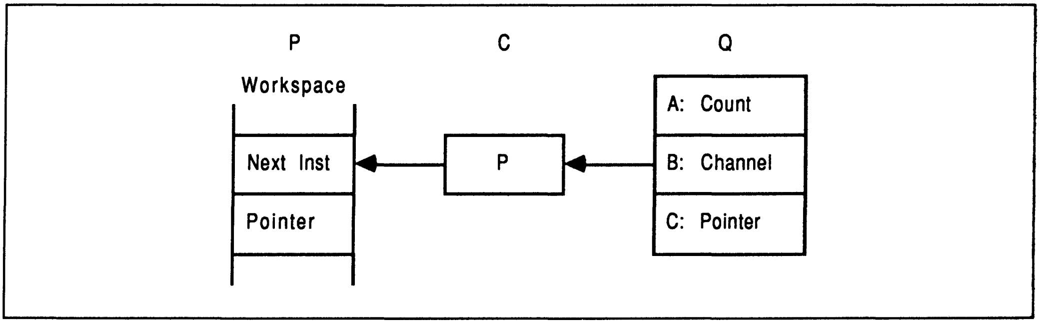


Figure 3.8

The message is copied, the waiting process P is added to the scheduling list, and the channel C is reset to its initial 'empty' state.

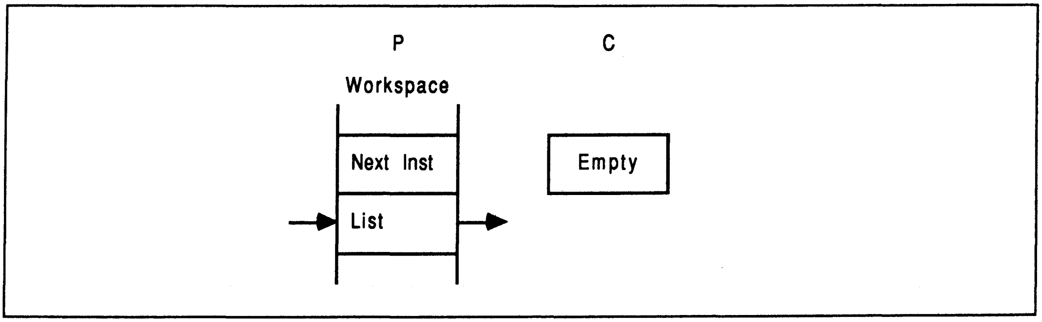


Figure 3.9

3.4.2 External channel communication

When a message is passed via an external channel the processor delegates to an autonomous link interface the job of transferring the message and deschedules the process. When the message has been transferred the link interface causes the processor to reschedule the waiting process. This allows the processor to continue the execution of other processes whilst the external message transfer is taking place.

Each link interface uses three registers:

a pointer to a process workspace
a pointer to a message
a count of bytes in the message

In figure 3.10 processes P and Q executed by different transputers communicate using a channel C implemented by a link connecting two transputers. P outputs, and Q inputs.

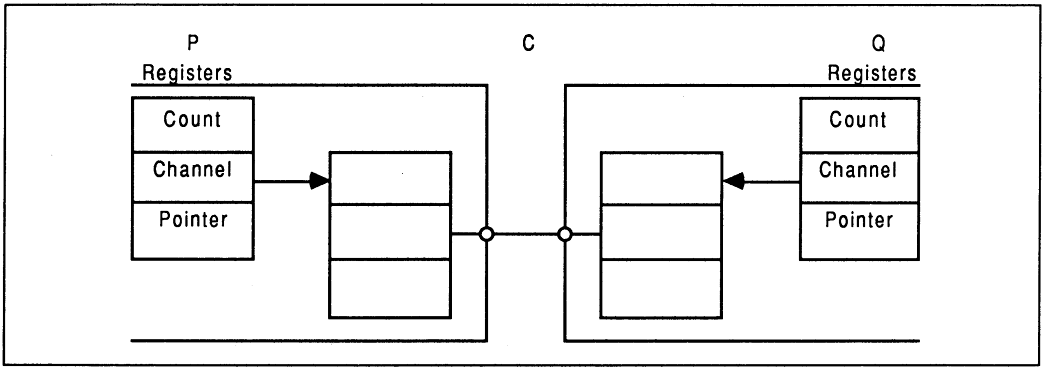


Figure 3.10 Communication between transputers

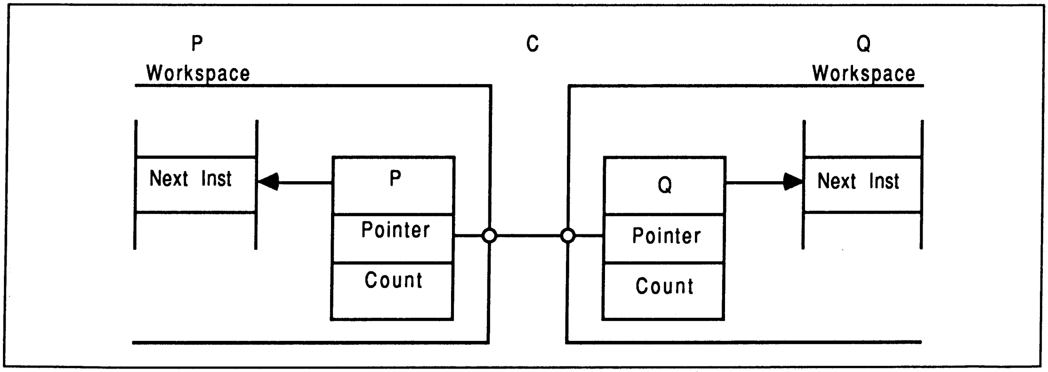


Figure 3.11

When P executes its output instruction, the registers in the link interface of the transputer executing P are initialised, and P is descheduled. Similarly, when Q executes its input instruction, the registers in the link interface of the process executing Q are initialised, and Q is descheduled (figure 3.11).

The message is now copied through the link, after which the workspaces of P and Q are returned to the corresponding scheduling lists (figure 3.12). The protocol used on P and Q ensures that it does not matter which of P and Q first becomes ready.

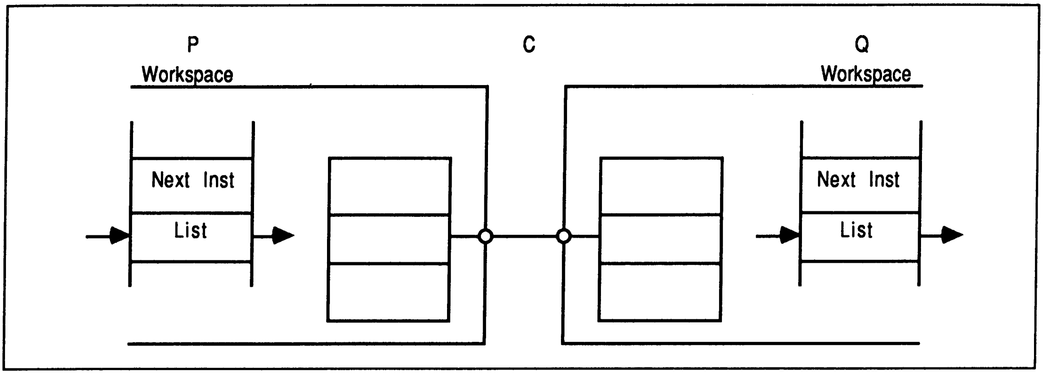


Figure 3.12

3.4.3 Communication links

A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer by two one-directional signal wires, along which data is transmitted serially. The two wires provide two occam channels, one in each direction. This requires a simple protocol to multiplex data and control information. Messages are transmitted as a sequence of bytes, each of which must be acknowledged before the next is transmitted. A byte of data is transmitted as a start bit followed by a one bit followed by eight bits of data followed by a stop bit. An acknowledgement is transmitted as a start bit followed by a stop bit. An acknowledgement indicates both that a process was able to receive the data byte and that it is able to buffer another byte.

The protocol permits an acknowledgement to be generated as soon as the receiver has identified a data packet. In this way the acknowledgement can be received by the transmitter before all of the data packet has been transmitted and the transmitter can transmit the next data packet immediately. Some transputers do not implement this overlapping and achieve a data rate of 0.8 Mbytes/sec using a link to transfer data in one direction. However, by implementing the overlapping and including sufficient buffering in the link hardware, the rate can be more than doubled to achieve 1.8 Mbytes/sec in one direction, and 2.4 Mbytes/sec when the link carries data in both directions. The diagram below shows the signals that would be observed on the two link wires when a data packet is overlapped with an acknowledgement.

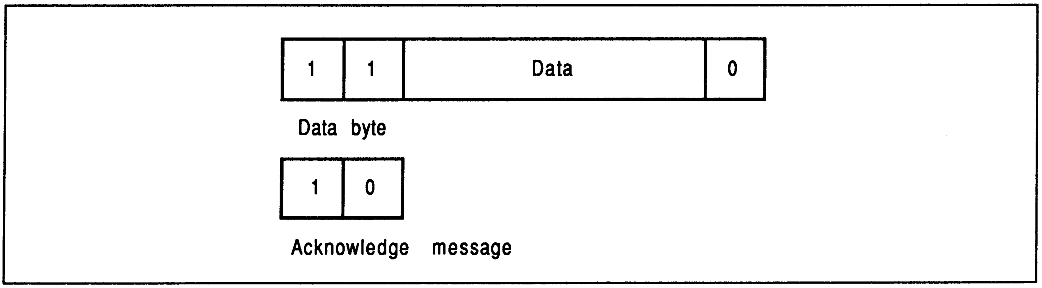


Figure 3.13 Link data and acknowledge formats

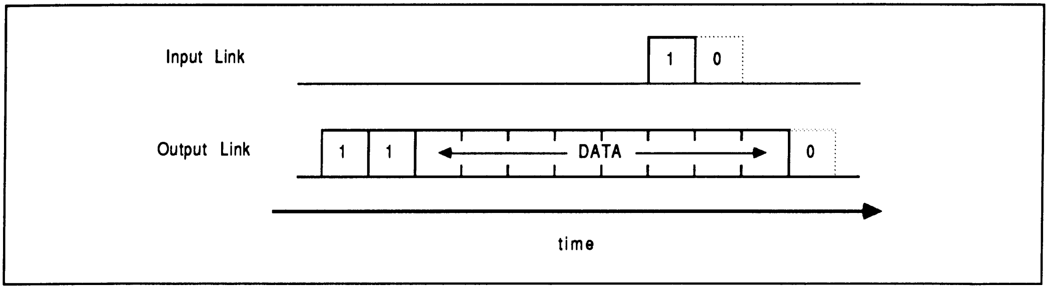


Figure 3.14 Overlapped link acknowledge

3.5 Timer

The transputer has a clock which 'ticks' every microsecond. The current value of the processor clock can be read by executing a *read 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' (i.e. *ClockReg* **AFTER** *SpecifiedTime*) then the instruction has no effect. If the time is in the 'future' (i.e. *SpecifiedTime* **AFTER** *Clockreg* or *SpecifiedTime = ClockReg*) then the process is descheduled. When the specified time is reached the process is scheduled again.

3.6 Alternative

The occam alternative construct enables a process to wait for input from any one of a number of channels, or until a specific time occurs. This requires special instructions, as the normal *input* instruction deschedules a process until a specific channel becomes ready, or until a specific time is reached. The instructions are:

enable channel
enable timer
alternative wait
disable channel
disable timer

The alternative is implemented by 'enabling' the channel input or timer input specified in each of its components. The 'alternative wait' is then used to deschedule the process if none of the channel or timer inputs is ready; the process will be re-scheduled when any one of them becomes ready. The channel and timer inputs are then 'disabled'. The 'disable' instructions are also designed to select the component of the alternative to be executed; the first component found to be ready is executed.

3.7 Floating point instructions

The core of the floating point instruction set was established fairly early in the design of the floating point transputer. This core includes simple load, store and arithmetic instructions. Examination of statistics derived from FORTRAN programs suggested that the addition of some more complex instructions would improve performance and code density. Proposed changes to the instruction set were assesed by examining their effect on a number of numerical programs. For each proposed instruction set, a compiler was constructed, the programs compiled with it, and the resulting code then run on a simulator. The resulting instruction set is now described.

In the floating point transputer operands are transferred between the transputer's memory and the floating point evaluation stack by means of floating point load and store instructions. There are two groups of such instructions, one for single length numbers, one for double length. In the description of the load and store instructions which follow only the double length instructions are described. However, there are single length instructions which correspond with each of the double length instructions.

The address of a floating point operand is computed on the CPU's stack and the operand is then loaded, from the addressed memory location, onto the FPU's stack. Operands in the floating point stack are tagged with their length. The operand's tag will be set when the operand is loaded or is computed. The tags allow the number of instructions needed for floating point operations to be reduced; there is no need, for example, to have both *floating add single* and *floating add double* instructions; a single *floating add* will suffice.

3.7.1 Optimising use of the stack

The depth of the register stacks in the CPU and FPU is carefully chosen. Floating point expressions commonly have embedded address calculations, as the operands of floating point operators are often elements of one dimensional or two dimensional arrays. The CPU stack is deep enough to allow most integer calculations and address calculations to be performed within it. Similarly, the depth of the FPU stack allows most floating point expressions to be evaluated within it, employing the CPU stack to form addresses for the operands.

No hardware is used to deal with stack overflow. A compiler can easily examine expressions and introduce temporary variables in memory to avoid stack overflow. The number of such temporary variables can be minimised by careful choice of the evaluation order; an algorithm to perform this optimisation is given in Bibliography reference *INMOS '88*. The algorithm is used to optimise the use of the integer stack of the transputer CPU.

3.7.2 Concurrent operation of FPU and CPU

In the floating point transputer the FPU operates concurrently with the CPU. This means that it is possible to perform an address calculation in the CPU whilst the FPU performs a floating point calculation. This can lead to significant performance improvements in real applications which access arrays heavily. This aspect of the floating point transputer's performance was carefully assessed, partly through examination of the 'Livermore Loops' (Bibliography reference *McMahon*). These are a collection of small kernels designed to represent the types of calculation performed on super-computers. They are of interest because they contain constructs which occur in real programs which are not represented in such programs as the Whetstone benchmark (see below). In particular, they contain accesses to two and three-dimensional arrays, operations where the concurrency within the floating point transputer is used to good effect. In some cases the compiler is able to choose the order of performing address calculations so as to maximise overlapping; this involves a modification of the algorithm mentioned earlier.

As a simple example of overlapping consider the implementation of Livermore Loop 7. The occam program for loop 7 is as follows:

```
-- LIVERMORE LOOP 7
SEQ k = 0 FOR n
  x[k] :=  u[k] + ((( r*(z[k] + (r*y[k]))) +
           (t*((u[k+3] + (r*(u[k+2] + (r*u[k+1])))))) +
           (t*((u[k+6] + (r*(u[k+5] + (r*u[k+4]))))))
```

The first stage in the computation of this is to load the value `y[k]`. This requires a sequence of four instructions. A further three instructions cause `r` to be loaded and the FPU multiply to be initiated.

Although the floating point multiplication takes several cycles to complete, the CPU is able to continue executing instructions whilst the FPU performs the multiplication. Thus the CPU can execute the next segment of code which computes the address of `z[k]` whilst the FPU perfroms the multiplication.

Finally, the value `z[k]` is pushed onto the floating point stack and added to the previously computed subexpression `r*y[k]`. It is not until value `z[k]` is loaded that the CPU needs to synchronise with the FPU.

The computation of the remainder of the expression proceeds in the same way, and the FPU never has to wait for the CPU to perform an address calculation.

3.8 Floating point unit design

In designing a concurrent systems component such as a transputer, it is important to maximise the performance obtained from a given area of silicon; many components can be used together to deliver more performance. This contrasts with the design of a conventional co-processor where the aim is to maximise the performance of a single processor by the use of a large area of silicon. As a result, in designing the floating point transputer, the performance benefits of silicon hungry devices such as barrel shifters and flash multipliers were carefully examined.

A flash multiplier is too large to fit on chip together with the processor, and would therefore necessitate the use of a separate co-processor chip. The introduction of a co-processor interface to a separate chip slows down the rate at which operands can be transferred to and from the floating point unit. Higher performance can, therefore, be obtained from a slow multiplier on the same chip as the processor than from a fast one on a separate chip. This leads to an important conclusion: *a separate co-processor chip is not appropriate for scalar floating point arithmetic.* A separate co-processor would be effective where a large amount of work can be handed to the co-processor by transferring a small amount of information; for example a vector co-processor would require only the addresses of its vector operands to be transferred via the co-processor interface.

It turns out that a flash multiplier also operates much more quickly than is necessary. Only a pipelined vector processor can deliver operands at a rate consistent with the use of such devices. In fact, any useful floating point calculation involves more operand accesses than operations. As an example consider the assignment `y[i] := y[i] + (t * x[i])` which constitutes the core of the LINPACK floating point benchmark. To perform this it is necessary to load three operands, perform two operations and to store a result. If we assume that it takes twice as long to perform a floating point operation as to load or store a floating point number then the execution time of this example would be evenly split between operand access time and operation time. This means that there would be at most a factor of two available in performance improvement from the use of an infinitely fast floating point unit!

Unlike a flash multiplier, a fast normalising shifter is important for fast floating point operation. When implementing IEEE arithmetic it may be necessary to perform a long shift on every floating point operation and unless a fast shifter is incorporated into the floating point unit the maximum operation time can become very long. Fortunately, unlike a flash multiplier, it is possible to design a fast shifter in a reasonable area of silicon. The shifter used is designed to perform a shift in a single cycle and to normalise in two cycles.

Consequently, the floating point unit contains a fast normalising shifter but not a flash multiplier. However there is a certain amount of logic devoted to multiplication and division. Multiplication is performed three-bits per cycle, and division is performed two-bits per cycle. Figure 3.15 illustrates the physical layout of the floating point unit.

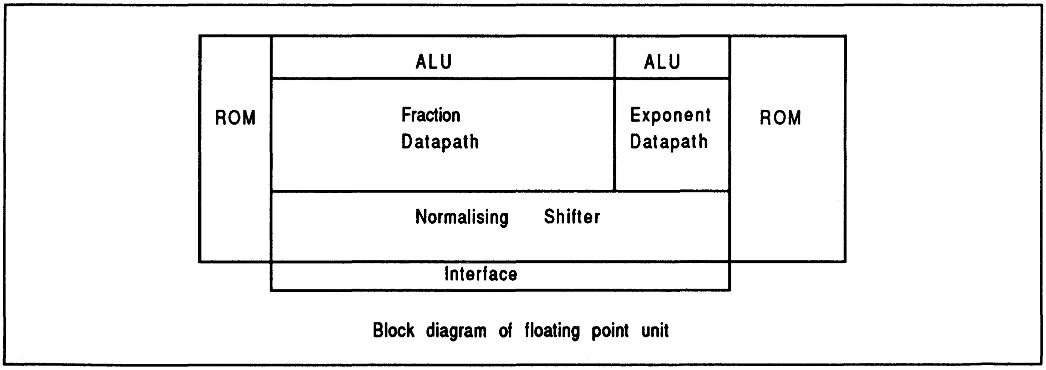


Figure 3.15 Floating point unit block diagram

The datapaths contain registers and shift paths. The fraction datapath is 59 bits wide, and the exponent data path is 13 bits wide. The normalising shifter interfaces to both the fraction data path and the exponent datapath. This is because the data to be shifted will come from the fraction datapath whilst the magnitude of the shift is associated with the exponent datapath. One further interesting aspect of the design is the microcode ROM. Although the diagram shows two ROMs, they are both part of the same logical ROM. This has been split in two so that control signals do not need to be bussed through the datapaths.

3.9 Graphics capability

The fast block move instructions of the transputers make them suitable for use in graphics applications using byte-per-pixel colour displays. The block move on the transputer is designed to saturate the memory bandwidth, moving any number of bytes from any byte boundary in memory to any other byte boundary using the smallest possible number of word read and write operations.

Some transputers extend this capability by incorporation of a two-dimensional version of the block move (`Move2d`) which can move windows around a screen at full memory bandwidth, and conditional versions of the same block move which can be used to place templates and text into windows. One of these operations (`Draw2d`) copies bytes from source to destination, writing only non-zero bytes to the destination. A new object of any shape can therefore be drawn on top of the current image. A further operation (`Clip2d`) copies only zero bytes in the source. All of these instructions achieve the speed of the simple move instruction, enabling a 1 million pixel screen to be drawn many times per second. Unlike the conventional 'bit-blt' instruction, it is never necessary to read the destination data.

3.9.1 Example - drawing coloured text

Drawing proportional spaced text provides a simple example of the use of the two-dimensional move instructions. The font is stored in a two dimensional array `Font`; the height of `Font` is the fixed character height, and the start of each character is defined by an array `start`. The textures of the character and its background are selected from an array of textures; the textures providing a range of colours or even stripes and tartans!

An occam procedure to perform such drawing is given below and the effect of each stage in the drawing process is illustrated by the diagrams on the final page of this document. First, (1) the texture for the character is selected and copied to a temporary area and (2) the character in the font is used to clip this texture to the appropriate shape. Then (3) the background texture is selected and copied to the screen, and (4) the new character drawn on top of it.

```
-- Draw character ch in texture F on background texture B
PROC DrawChar(VAL INT Ch, F, B)
  SEQ
    IF
      (x + width[ch]) > screenwidth
        SEQ
          x := 0
          y := y + height
      (x + width[ch]) <= screenwidth
        SKIP
    [height] [maxwidth] BYTE Temp :
    SEQ
      Move2d(Texture[F],0,0, Temp,0,0, width[ch],height)
      Clip2d(Font[ch],start[ch],0, Temp,0,0, width[ch],height)
      Move2d(Texture[B],0,0, Screen,x,y, width[ch],height))
      Draw2d(Temp,0,0, Screen,x,y, width[ch],height)
      x := x + width[ch]
```

This procedure will fill a 1 million pixel screen with proportionally spaced characters in about 1/6 second. Obviously, a simpler and faster version could be used if the character colour or background colour was restricted. The operation of this procedure is illustrated in figure 3.16.

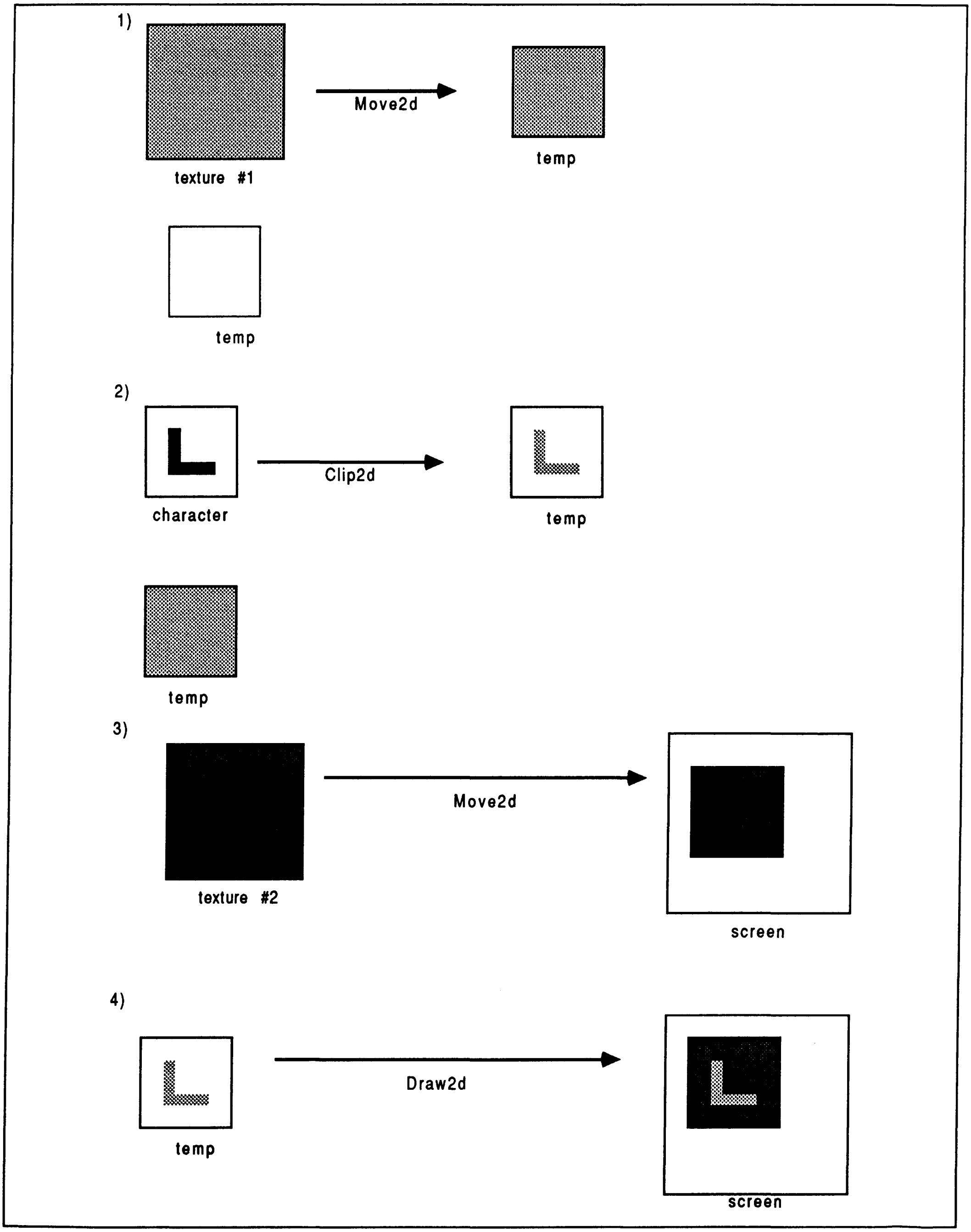


Figure 3.16 Use of enhanced graphics instructions

4 Conclusion

The INMOS transputer family is a range of system components which can be used to construct high performance concurrent systems. As all members of the family incorporate INMOS communications links, a system may be constructed from different members of the family. All transputers provide hardware support for concurrency and offer exceptional performance on process scheduling, inter-process communication and inter-transputer communication.

The design of the transputers takes careful note of silicon economics. The central processor used in the transputer offers a performance comparable with that of other VLSI processors several times larger. The small size of the processor allows a memory and communications system to be integrated on to the same VLSI device. This level of integration allows very fast access to memory and very fast inter-transputer communication. Similarly, the transputer floating point unit is integrated into the same device as the central processor, eliminating the delays inherent in communicating data between devices.

IMS T800
transputer

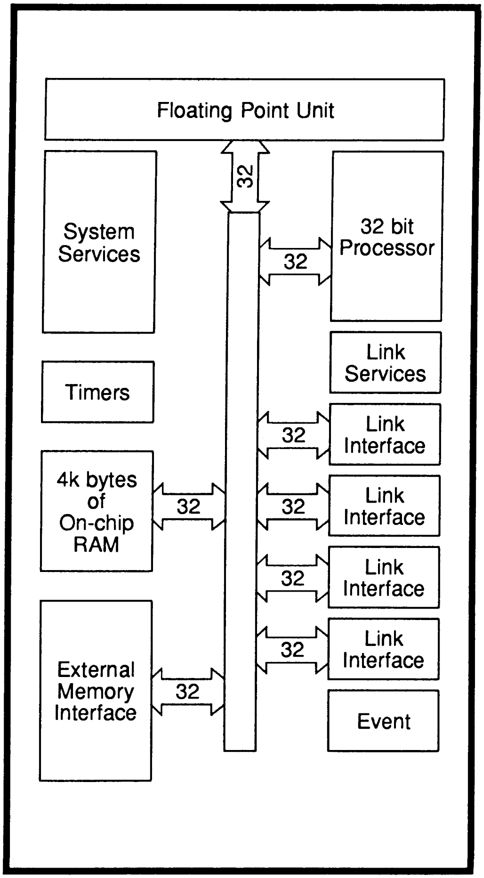
Engineering Data

FEATURES

- 32 bit architecture
- 33 ns internal cycle time
- 30 MIPS (peak) instruction rate
- 4.3 Mflops (peak) instruction rate
- 64 bit on-chip floating point unit which conforms to IEEE 754
- 4 Kbytes on-chip static RAM
- 120 Mbytes/sec sustained data rate to internal memory
- 4 Gbytes directly addressable external memory
- 40 Mbytes/sec sustained data rate to internal memory
- 630 ns response to interrupts
- Four INMOS serial links 5/10/20 Mbits/sec
- Bi-directional data rate of 2.4 Mbytes/sec per link
- High performance graphics support with block move instructions
- Boot from ROM or communication links
- Single 5 MHz clock input
- Single +5V ±5% power supply
- MIL-STD-883C processing will be available

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
- Pattern recognition
- Artificial intelligence



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.

For convenience of description, the IMS T800 operation is split into the basic blocks shown in figure 1.1.

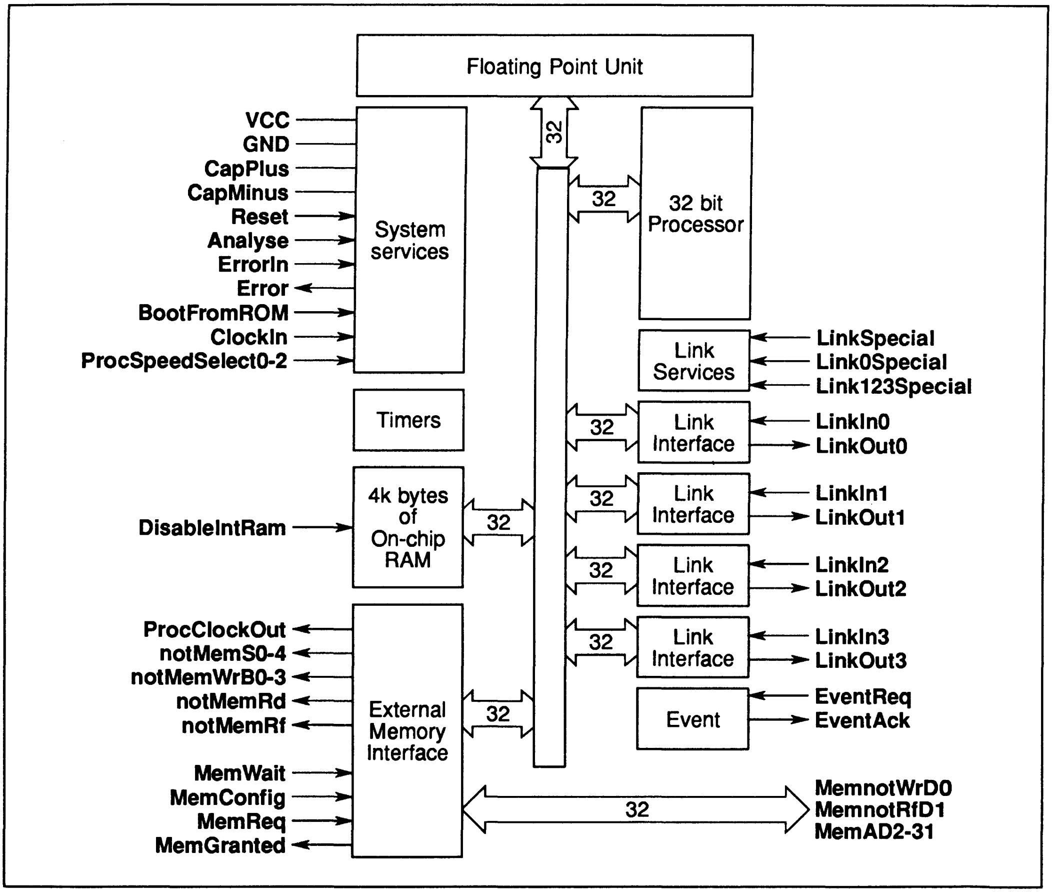


Figure 1.1 IMS T800 block diagram

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 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

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 bootstrap 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/sec, but also operate at 5 or 20 Mbits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

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 Architecture chapter, which details the architecture of the transputer and gives an overview of occam.

2 Pin designations

Table 2.1 IMS T800 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
DisableIntRAM	in	Disable internal RAM
DoNotWire		Must not be wired

Table 2.2 IMS T800 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

Table 2.3 IMS T800 event

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

Table 2.4 IMS T800 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 238.

3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 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 4 Gbytes 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.

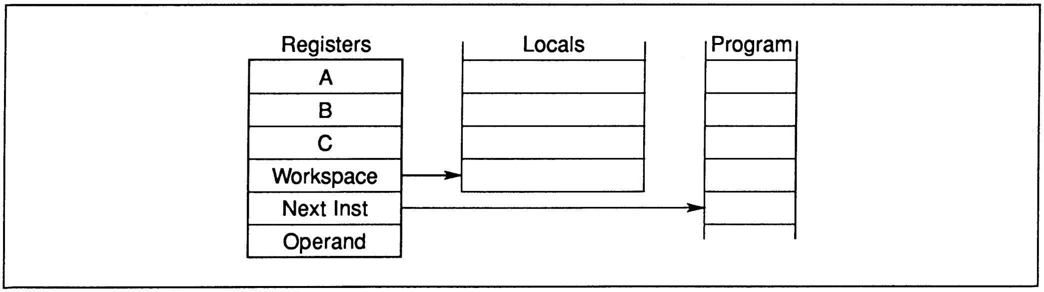


Figure 3.1 Registers

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.

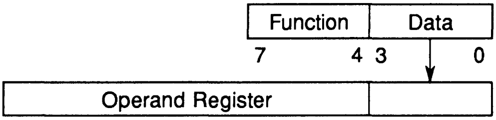


Figure 3.2 Instruction format

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, 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 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 four 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 51).

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 51). 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.

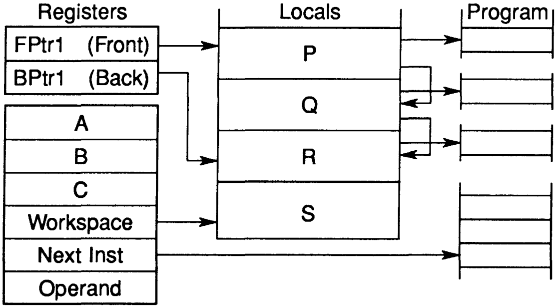


Figure 3.3 Linked process list

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 54). 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 54). 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. 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 54).

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 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 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 4295 seconds. 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 76 hours.

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.

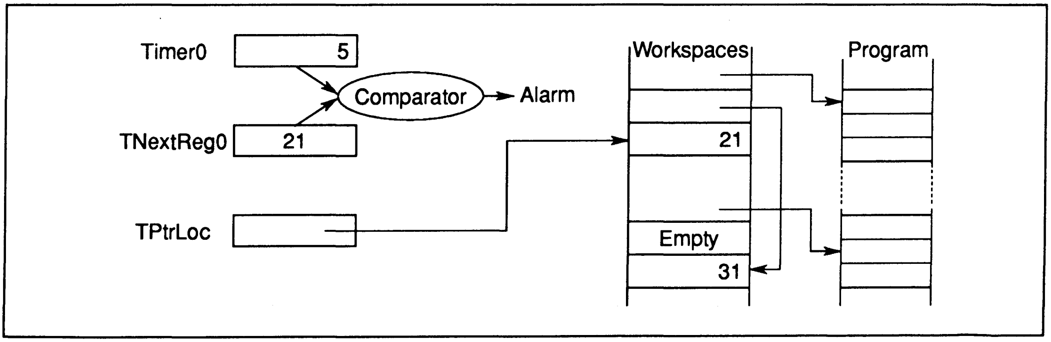


Figure 3.4 Timer registers

4 Instruction set summary

The Function Codes table 4.8. gives the basic function code set (page 48). 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* #FFFFFFE1)		
is coded as			
nfix	#1	#6	#61
ldc	#1	#4	#41

Tables 4.9 to 4.27 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

In the Floating Point Operation Codes tables 4.21 to 4.27, a selector sequence code (page 62) 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 the 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 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.
m	Bit number of the highest bit set in the absolute value of 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.
p	Number of words per row.
r	Number of rows.

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	54
E	The instruction will affect the *Error* flag	55, 69
F	The instruction will affect the *FP_Error* flag	62, 55

4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 50). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 68).

Table 4.5 Descheduling point instructions

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

4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the *Error* flag (page 69) directly. Note, however, that the floating point unit error flag *FP_Error* is set by certain floating point instructions (page 55), and that *Error* can be set from this flag by *fpcheckerror.*

Table 4.6 Error setting instructions

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

4.3 Floating point errors

The instructions in table 4.7 are the only ones which can affect the floating point error flag *FP_Error* (page 62). *Error* is set from this flag by *fpcheckerror* if *FP_Error* is set.

Table 4.7 Floating point error setting instructions

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

Table 4.8 IMS T800 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 4.9 IMS T800 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	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
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*	

Table 4.10 IMS T800 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	33	long multiply	
1A	21FA	ldiv	35	long divide	E
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)	

Table 4.11 IMS T800 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	
5A	25FA	dup	1	duplicate top of stack	

Table 4.12 IMS T800 2D block move operation codes

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

Table 4.13 IMS T800 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
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 word	

Table 4.14 IMS T800 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	
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	4	store byte	
4A	24FA	move	2w+8	move message	

Table 4.15 IMS T800 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 4.16 IMS T800 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 4.17 IMS T800 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 4.18 IMS T800 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 4.19 IMS T800 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 4.20 IMS T800 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	

Table 4.21 IMS T800 floating point load/store operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
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	8/11	fp load non local & add single	F
A6	2AF6	fpldnladddb	9/12	fp load non local & add double	F
AC	2AFC	fpldnlmulsn	13/20	fp load non local & multiply single	F
A8	2AF8	fpldnlmuldb	21/30	fp load non local & multiply double	F
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	

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

Table 4.22 IMS T800 floating point general operation codes

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

Table 4.23 IMS T800 floating point rounding operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
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	

Table 4.24 IMS T800 floating point error operation codes

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

Table 4.25 IMS T800 floating point comparison operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
94	29F4	fpgt	4/6	fp greater than	F
95	29F5	fpeq	3/5	fp equality	F
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	F
0F	**s**	fpuchki64	3/4	check in range of type int64	F

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

Table 4.26 IMS T800 floating point conversion operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	**s**	fpur32tor64	3/4	real32 to real64	F
08	**s**	fpur64tor32	6/9	real64 to real32	F
9D	29FD	fprtoi32	7/9	real to int32	F
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	F

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

Table 4.27 IMS T800 floating point arithmetic operation codes

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

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

5 Floating point unit

The 64 bit 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 (page 47).

As with the CPU stack, the FPU stack is not saved when rescheduling (page 50) 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

Table 5.1 *fpentry* coding

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 (page 55). *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 (page 55). 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.

Table 5.2 Typical floating point operation times for IMS T800

	T800-20		**T800-30**	
Operation	**Single length**	**Double length**	**Single length**	**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.

6 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.

6.1 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.

6.2 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 50 mm. The connections must not touch power supplies or other noise sources.

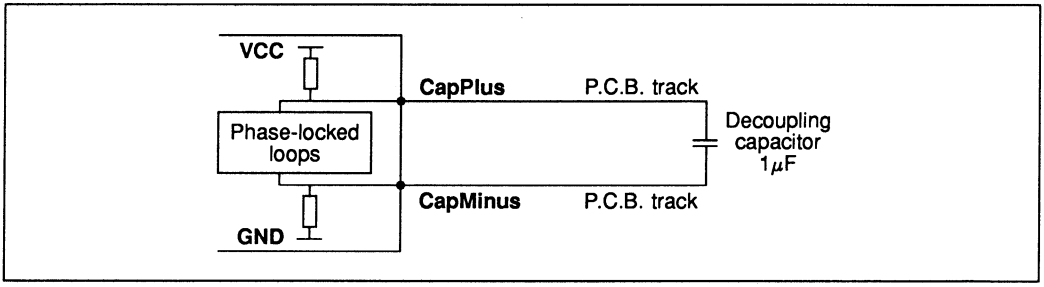


Figure 6.1 Recommended PLL decoupling

6.3 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 6.1 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 11.3).

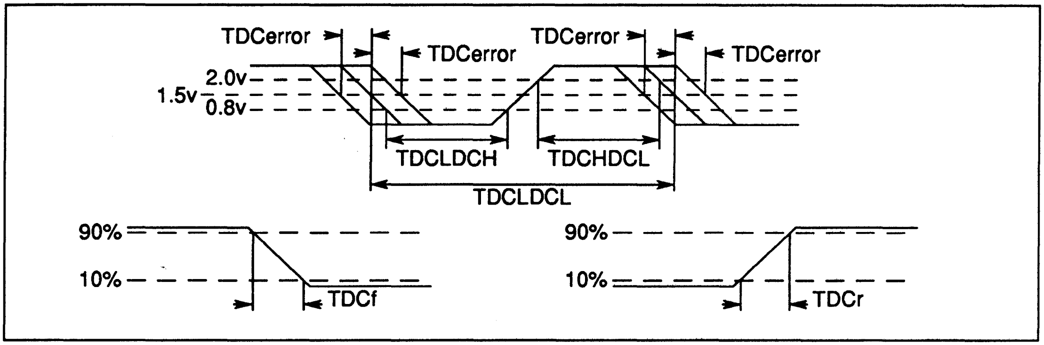


Figure 6.2 ClockIn timing

6.4 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 20 MHz 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.

Table 6.2 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.

6.5 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 bootstrap (page 91).

After the end of **Reset** there will be a delay of 144 periods of **ClockIn** (figure 6.3). Following this, the **MemWrD0**, **MemRfD1** and **MemAD2-31** pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 81). 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 (page 82), 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.

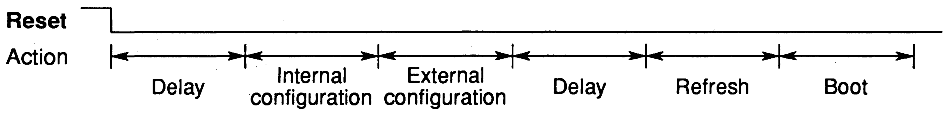


Figure 6.3 IMS T800 post-reset sequence

6.6 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 #7FFFFFFE. 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 70).

Table 6.3 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.

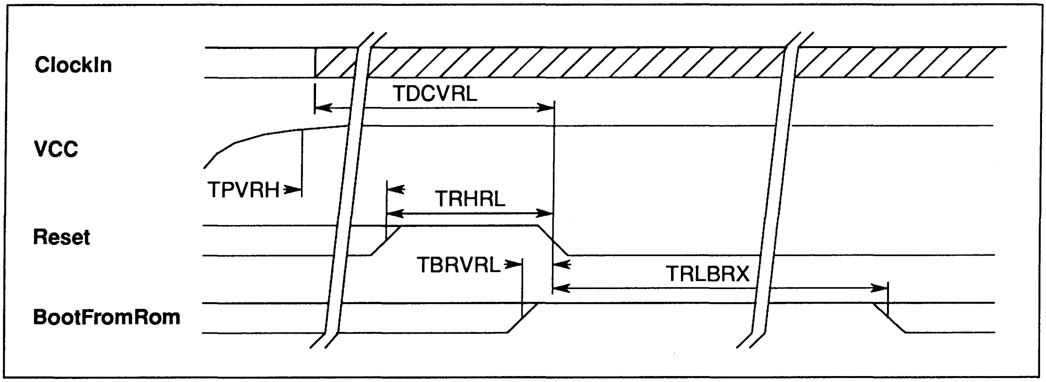


Figure 6.4 Transputer reset timing with Analyse low

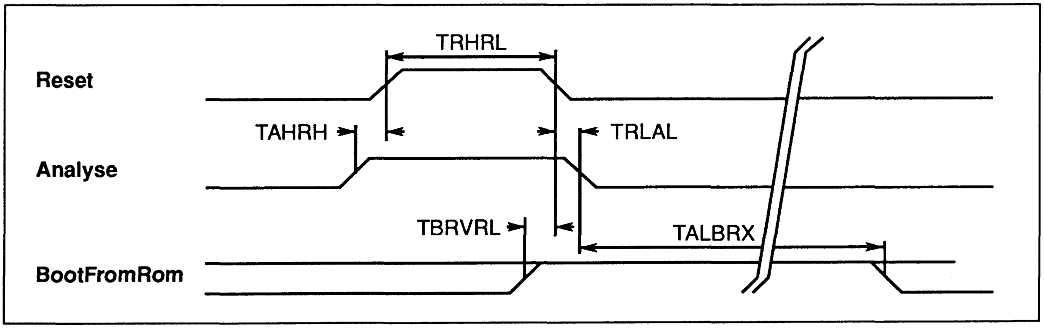


Figure 6.5 Transputer reset and analyse timing

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.

6.7 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 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 bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

6.8 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 54). 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. 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 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 6.4.

Table 6.4 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.

6.9 Error, ErrorIn

The **Error** pin carries the OR'ed output 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 (page 55). It can also be set from the floating point unit under certain circumstances (page 55, 62). 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 68).

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. **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 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 '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. 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 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 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*.

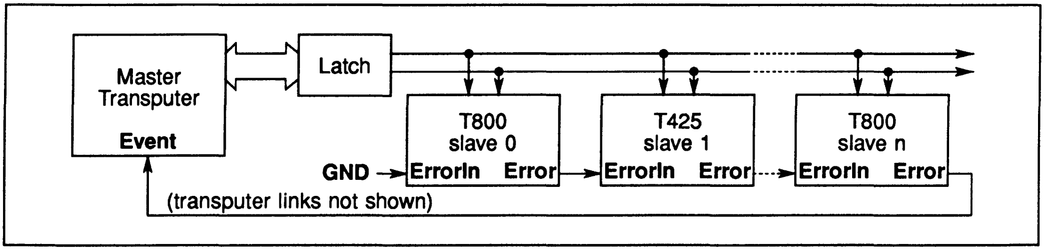


Figure 6.6 Error handling in a multi-transputer system

7 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** (page 72). The transputer can also access 4 Gbytes 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 T800 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.

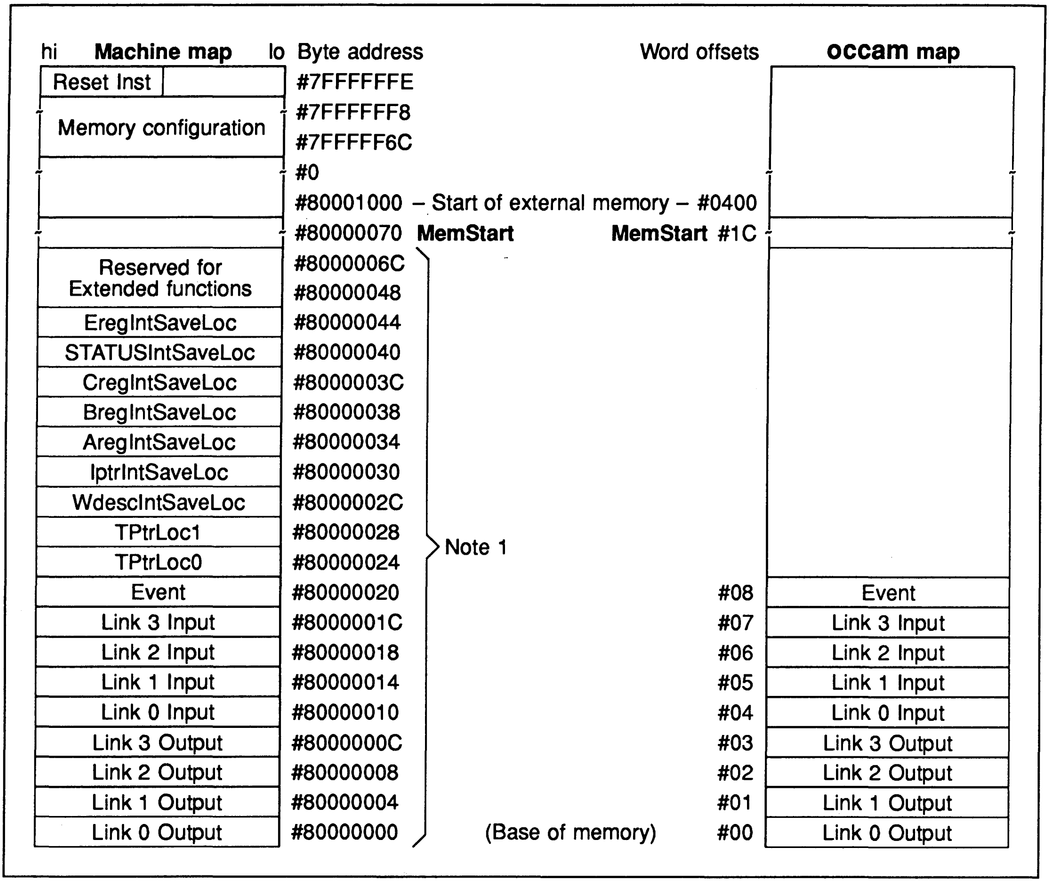
Internal memory starts at the most negative address #80000000 and extends to #80000FFF. User memory begins at #80000070; this location is given the name *MemStart*.

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 #80000048	Note 1		
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

Figure 7.1 IMS T800 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 68). For details see The Transputer Instruction Set - A Compiler Writers' Guide.

8 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 (page 81). If none are suitable the user can configure the interface to specific requirements, as shown in page 82.

8.1 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).

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 8.4.

The timing parameters in the following tables are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

8.2 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 (figure 8.11).

Table 8.1 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
Tm	ProcClockOut half cycle	**b**-0.5	**b**	**b**+0.5	ns	2
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.

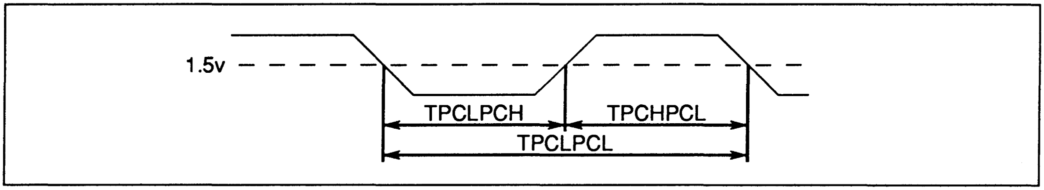


Figure 8.1 IMS T800 ProcClockOut timing

8.3 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 (page 68).

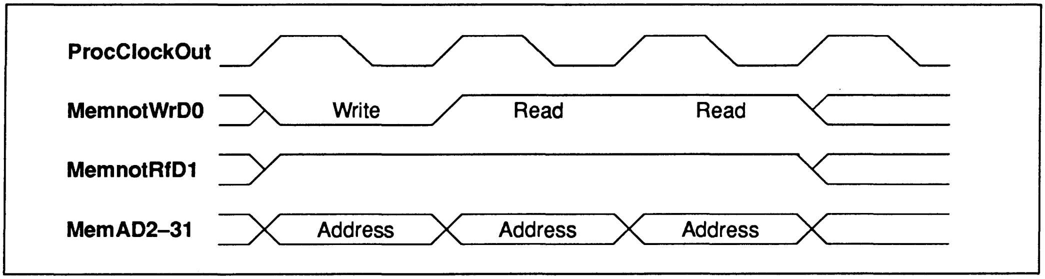


Figure 8.2 IMS T800 bus activity for internal memory cycle

8.4 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 transputer 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 transputer 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** (page 74) to start on a rising edge of **ProcClockOut**, data will be valid during this time also.

8.5 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**.

8.6 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**.

8.7 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**.

8.8 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 (page 89). 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 low. Figure 8.5 shows the effect of **Wait** on strobes in more detail; each division on the scale is one period **Tm**.

Table 8.2 Read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

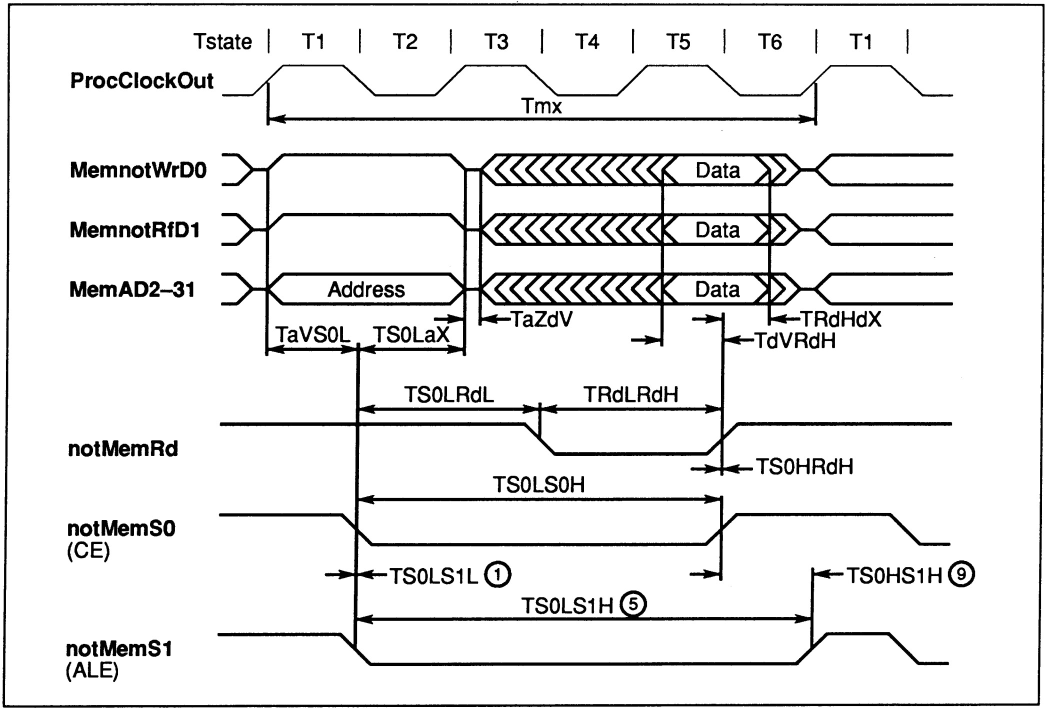


Figure 8.3 IMS T800 external read cycle: static memory

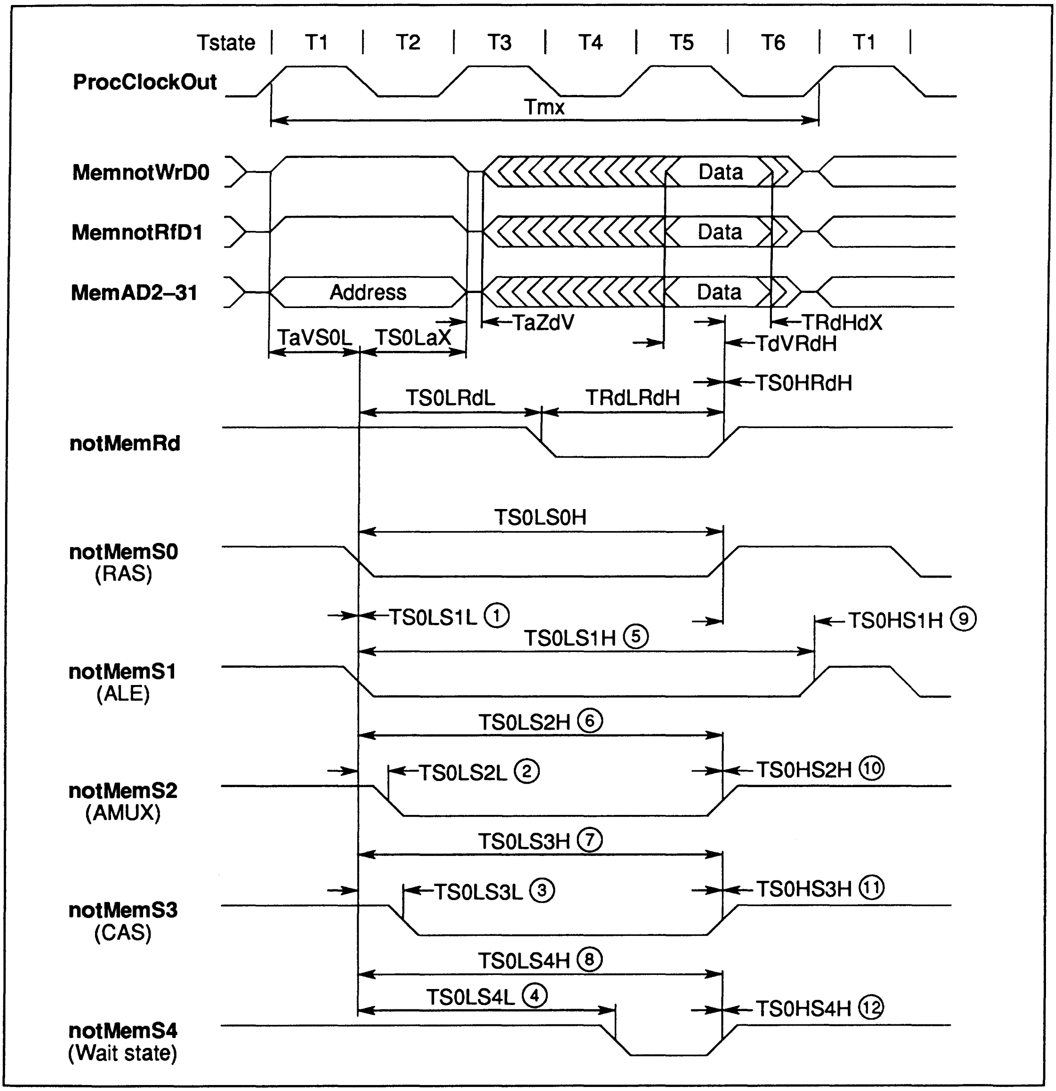


Figure 8.4 IMS T800 external read cycle: dynamic memory

Table 8.3 IMS T800 strobe timing

SYMBOL	ⓝ	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

Figure 8.5 IMS T800 effect of wait states on strobes

Table 8.4 Strobe S0 to ProcClockOut skew

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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	

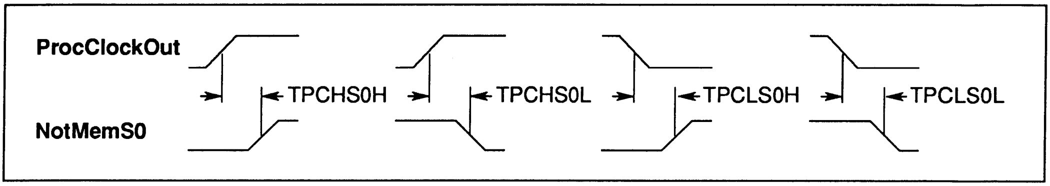


Figure 8.6 IMS T800 skew of notMemS0 to ProcClockOut

8.9 notMemWrB0-3

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

The transputer 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.

Table 8.5 Write

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

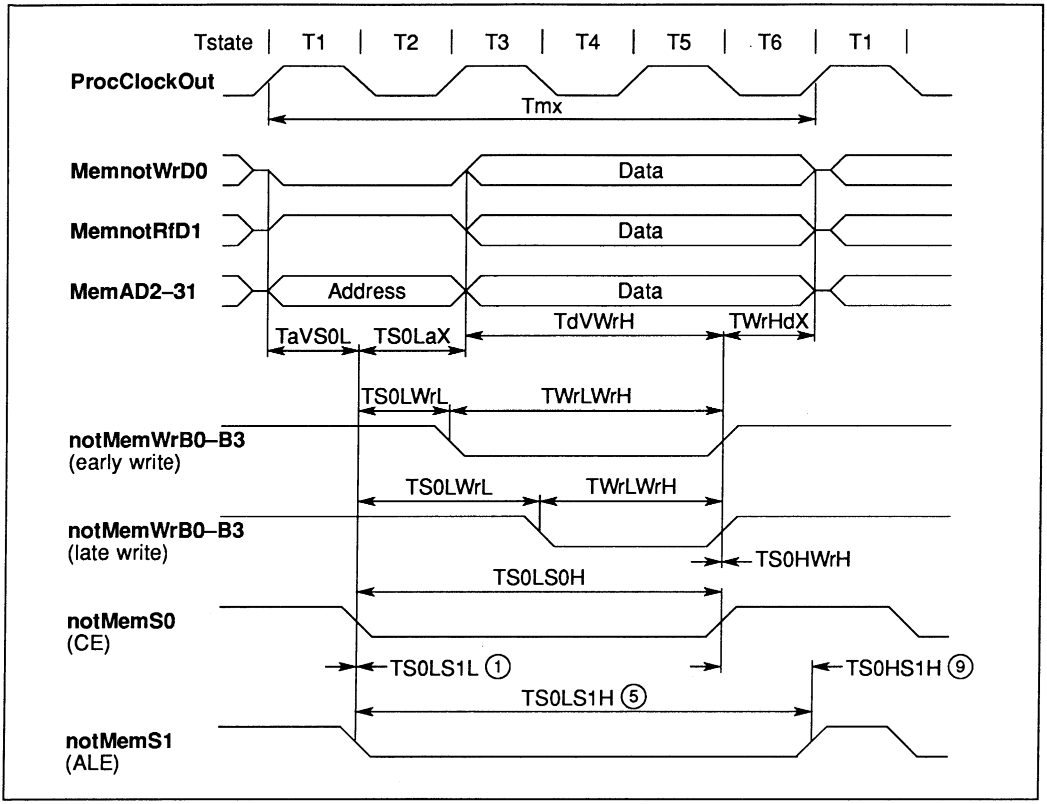


Figure 8.7 IMS T800 external write cycle

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.

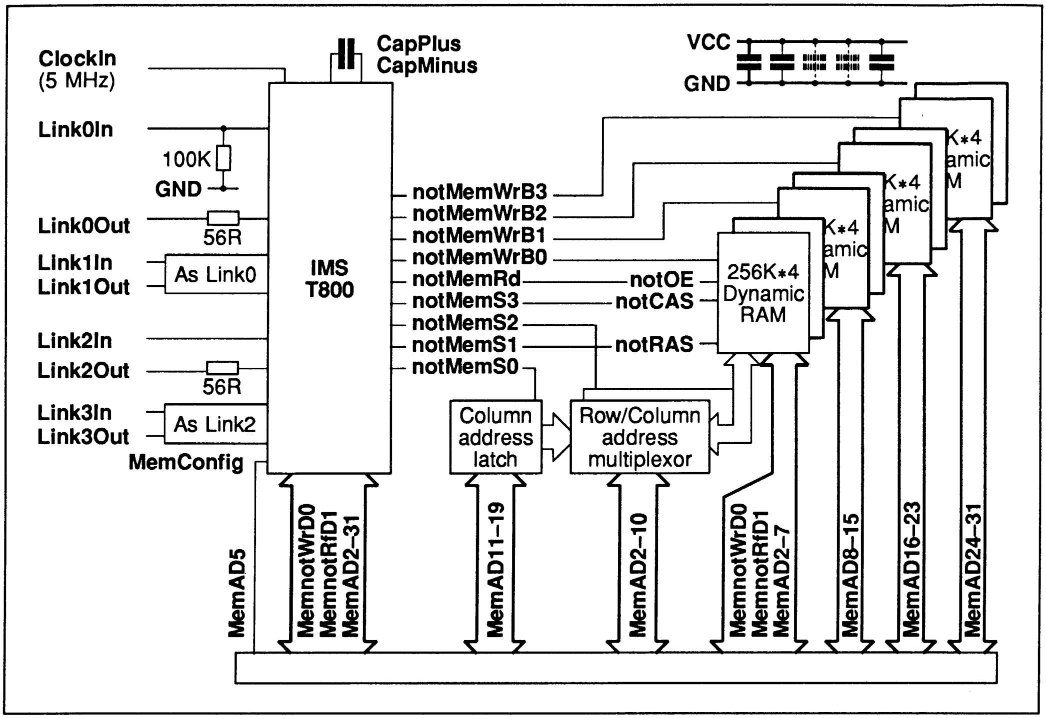


Figure 8.8 IMS T800 dynamic RAM application

8.10 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.

8.10.1 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 17 of the possible configurations are valid, all others remain at the default configuration.

Table 8.6 IMS T800 internal configuration coding

	Duration of each Tstate periods Tm						Strobe coefficient				Write cycle	Refresh interval	Cycle time
Pin	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**s1**	**s2**	**s3**	**s4**	**type**	**ClockIn cycles**	**Proc cycles**
MemnotWrD0	1	1	1	1	1	1	30	1	3	5	late	72	3
MemnotRfD1	1	2	1	1	1	2	30	1	2	7	late	72	4
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6
MemAD8	1	1	1	1	1	1	30	1	2	3	early	†	3
MemAD9	1	1	2	1	2	1	30	2	5	9	early	†	4
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9
MemAD12	1	1	2	1	2	1	4	1	2	3	early	72	4
MemAD13	2	1	2	1	2	2	5	1	2	3	early	72	5
MemAD14	2	2	2	1	3	2	6	1	3	4	early	72	6
MemAD15	2	1	2	3	3	3	8	1	2	3	early	72	7
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12

† Provided for static RAM only.

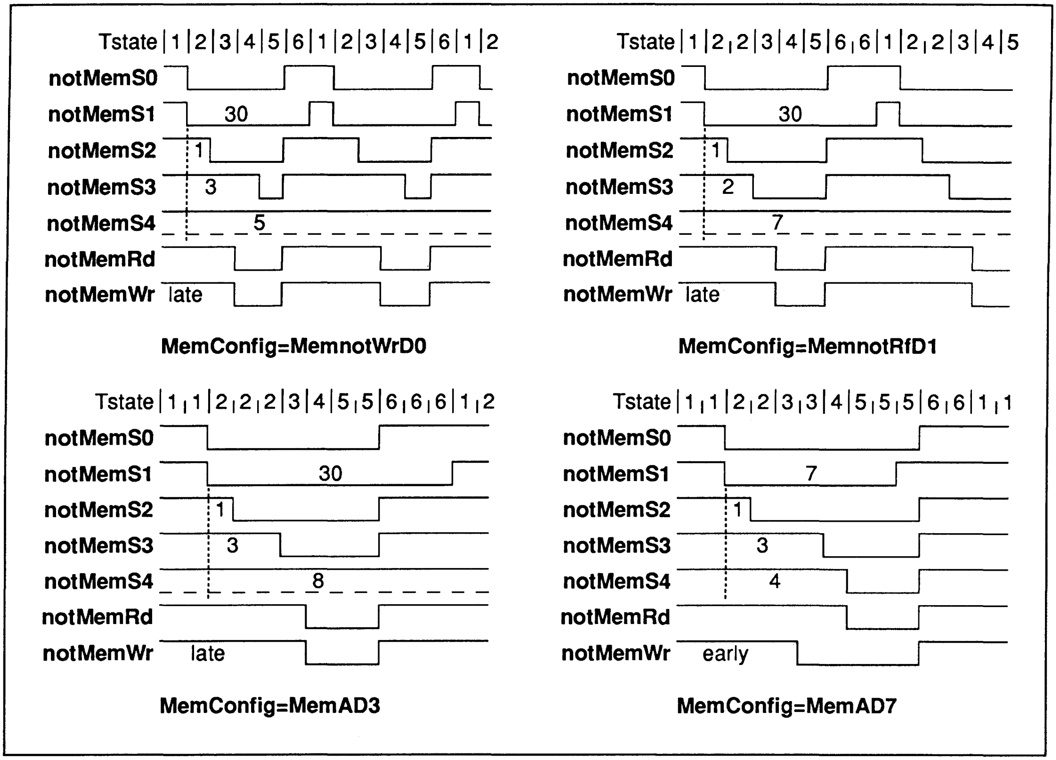


Figure 8.9 IMS T800 internal configuration

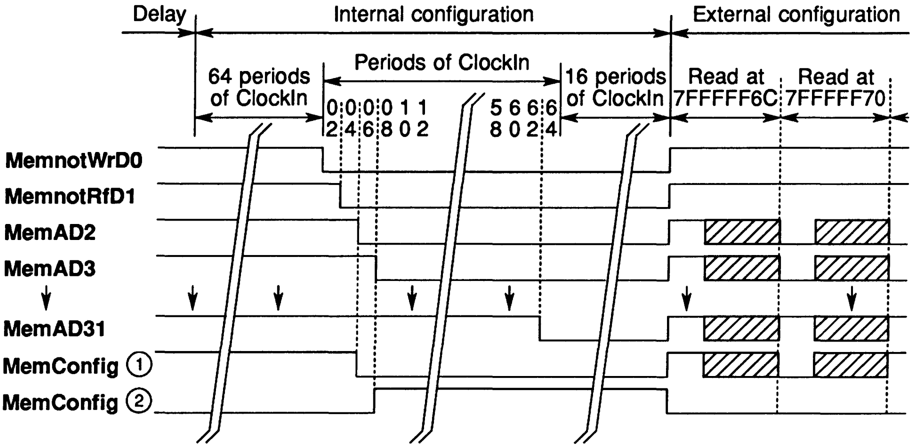
8.10.2 External configuration

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 table 8.7, 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; late write.

The external memory configuration table 8.7 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.



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

Figure 8.10 IMS T800 internal configuration scan

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 page 74.

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** (page 72).

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. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

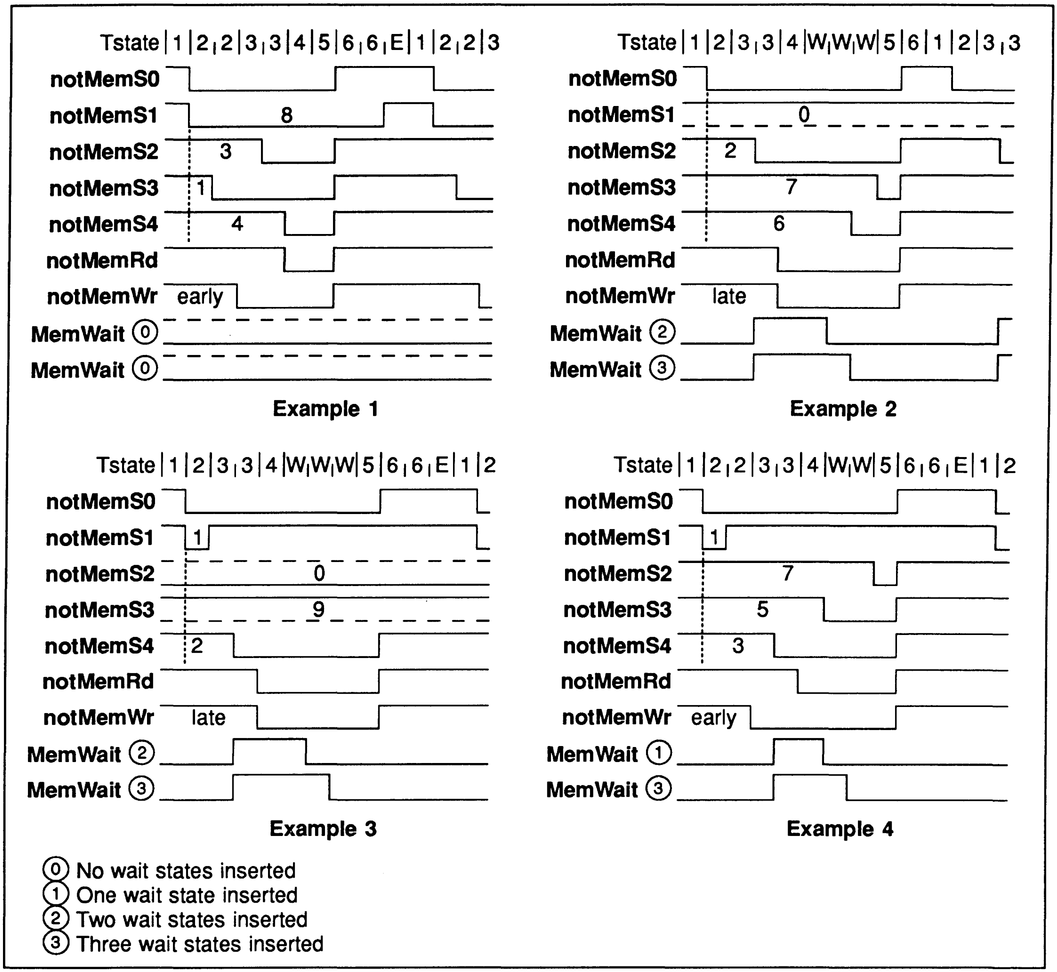
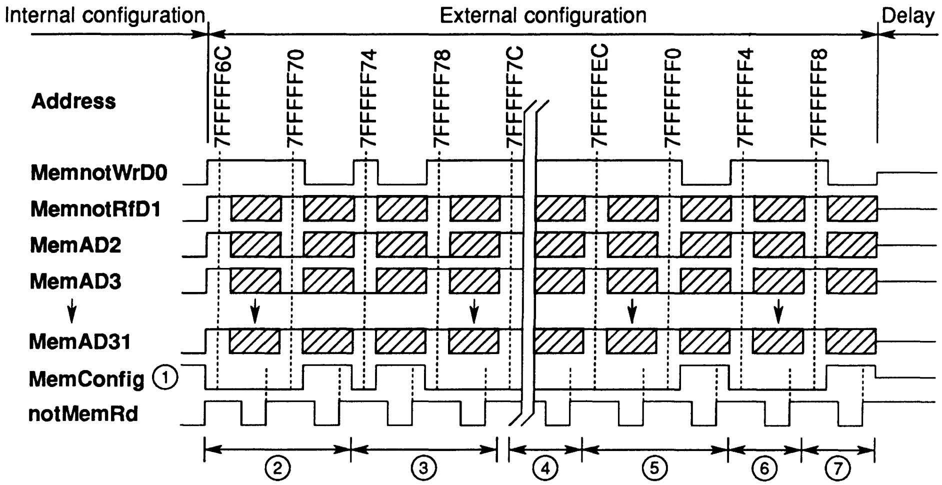


Figure 8.11 IMS T800 external configuration



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

Figure 8.12 IMS T800 external configuration scan

Table 8.7 IMS T800 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

Table 8.8 IMS T800 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 5 MHz:

$$\text{Interval} = 18 * 200 = 3600 \text{ ns}$$

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

Table 8.9 Memory configuration

SYMBOL	**PARAMETER**	**MIN**	**NOM**	**MAX**	**UNITS**	**NOTE**
TMCVRdH	Memory configuration data setup	30			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**.

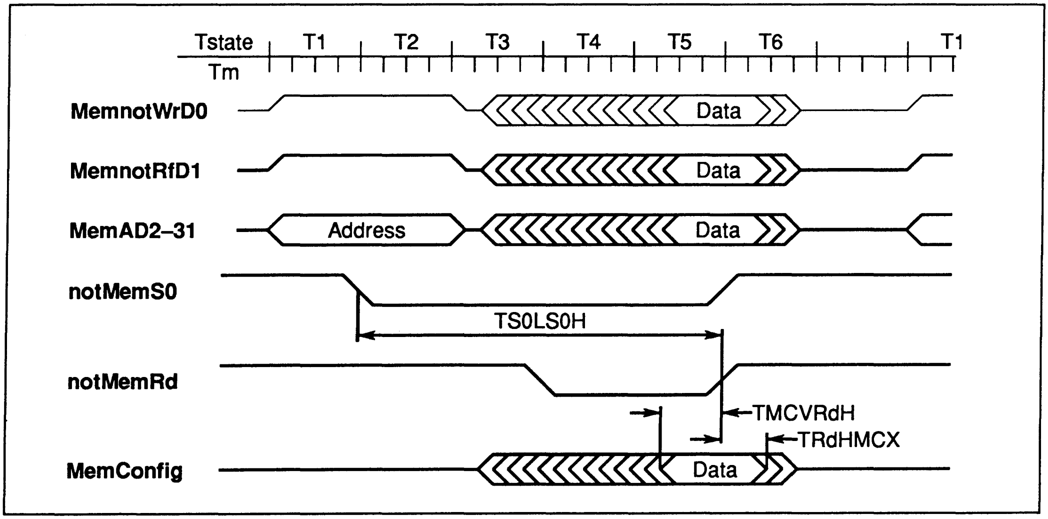


Figure 8.13 IMS T800 external configuration read cycle timing

8.11 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. **MemWait** operates normally during refresh cycles.

Table 8.10 Memory refresh

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRfLRfH	Refresh pulse width low	a		a+6	ns	1
TRaVSOL	Refresh address setup before notMemS0		b		ns	2
TRfLSOL	Refresh indicator setup before notMemS0		b		ns	2

Notes

1 **a** is total **Tmx+Tm**.

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

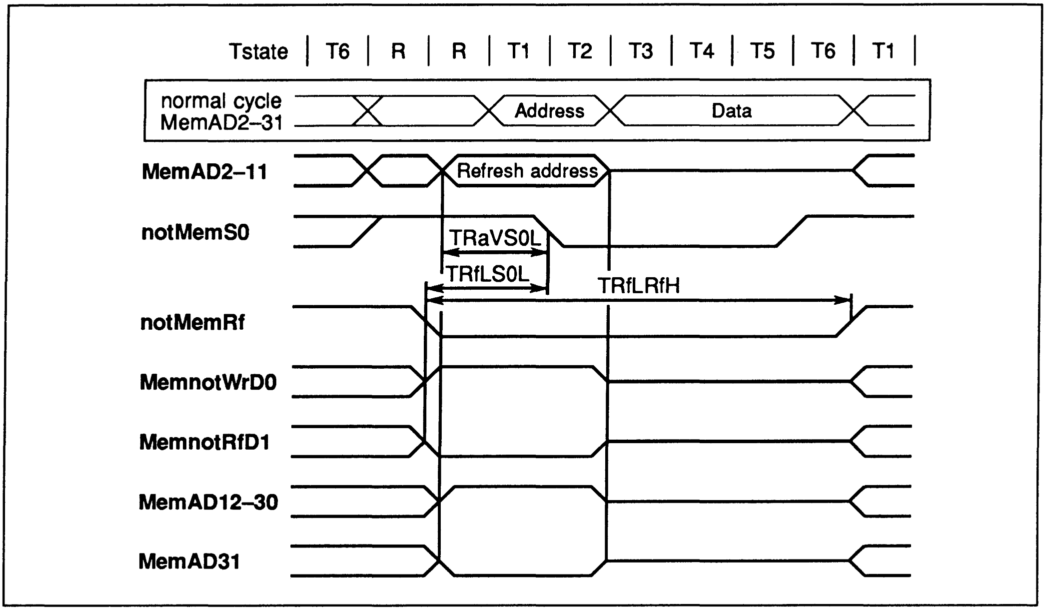


Figure 8.14 IMS T800 refresh cycle timing

8.12 MemWait

Taking **MemWait** high with the timing shown will extend the duration of **T4**. **MemWait** is sampled relative to the falling edge of **ProcClockOut** during a **T3** period, 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. **MemWait** operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

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.

Table 8.11 Memory wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHWtH	Wait setup	0.5**Tm**+3			ns	1,2
TPCHWtL	Wait hold	0.5**Tm**+3			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	2**Tm**			ns	

Notes

1 **ProcClockOut** load should not exceed 50pf.

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

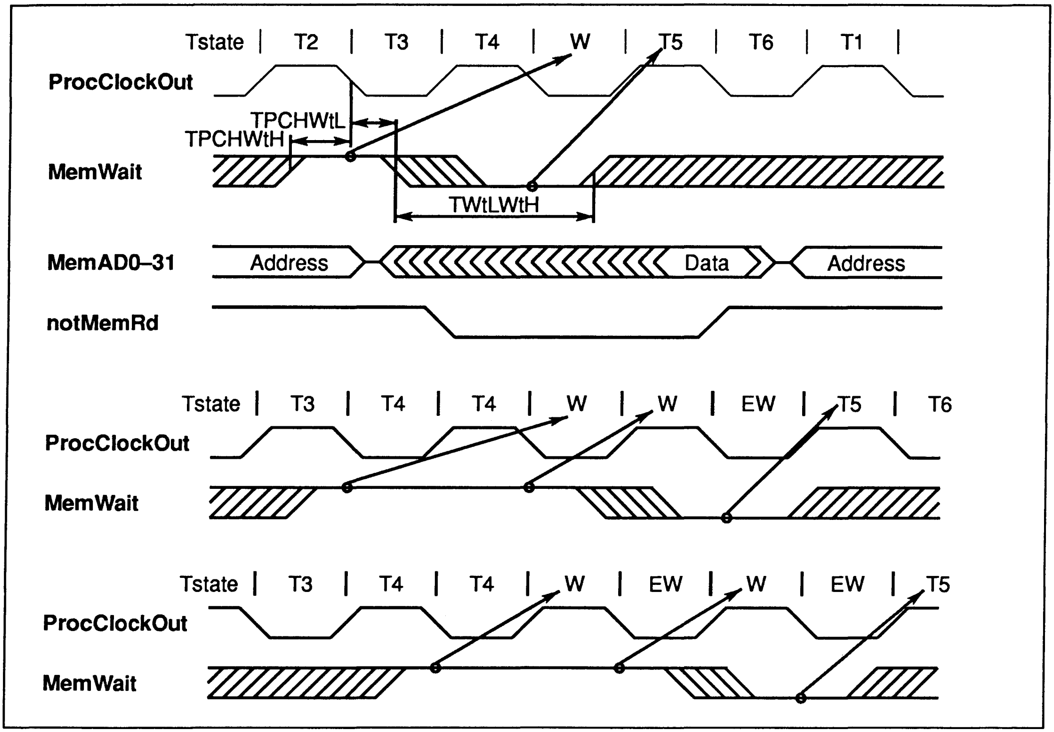


Figure 8.15 IMS T800 memory wait timing

8.13 MemReq, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer 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 8.8), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

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 8.12 Memory request

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4		7	Tm	1
TMRLMGL	Memory request end response time	2		5	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**).

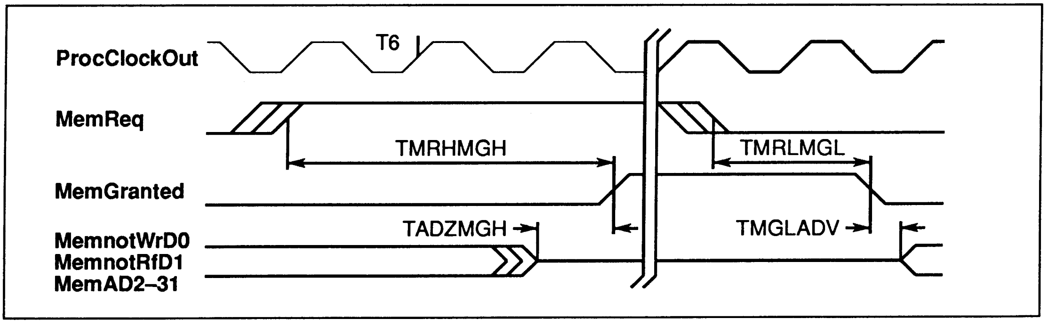
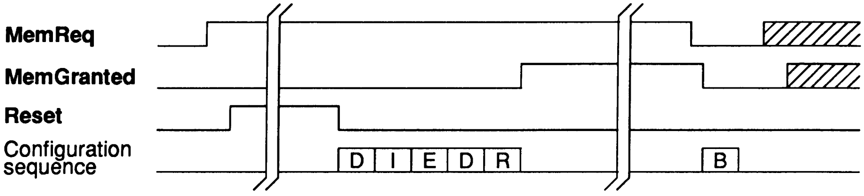


Figure 8.16 IMS T800 memory request timing



D Pre- and post-configuration delays (figure 6.3)
I Internal configuration sequence
E External configuration sequence
R Initial refresh sequence
B Bootstrap sequence

Figure 8.17 IMS T800 DMA sequence at reset

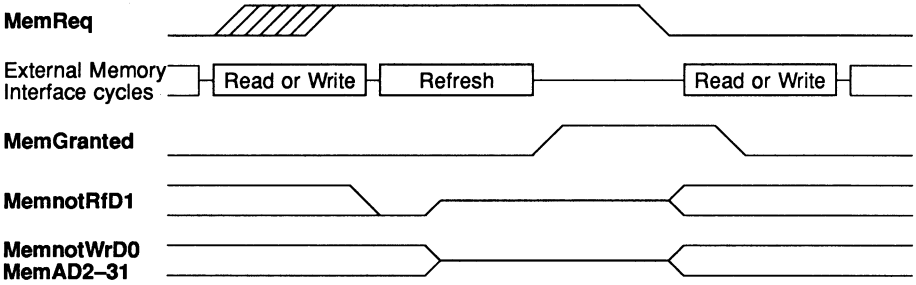


Figure 8.18 IMS T800 operation of MemReq, MemGranted with external, refresh memory cycles

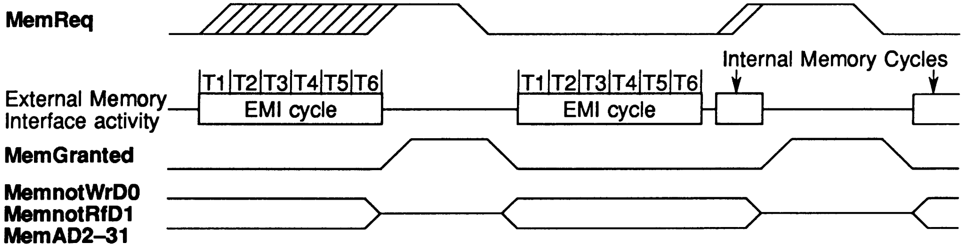


Figure 8.19 IMS T800 operation of MemReq, MemGranted with external, internal memory cycles

9 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 51. Setting a high priority task to wait for an event input is a way of interrupting a transputer program.

Table 9.1 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**.

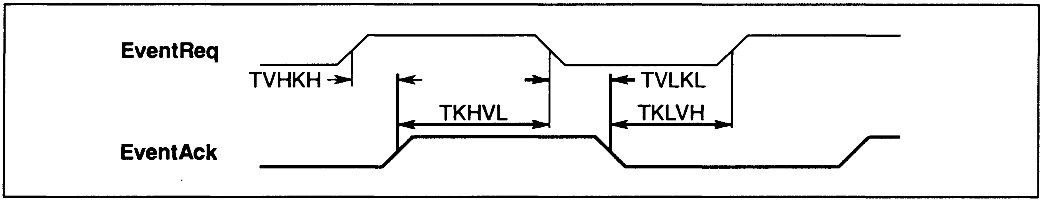


Figure 9.1 IMS T800 event timing

10 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 T800 links allow 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/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 10.1 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 10.1 Speed Settings for Transputer 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

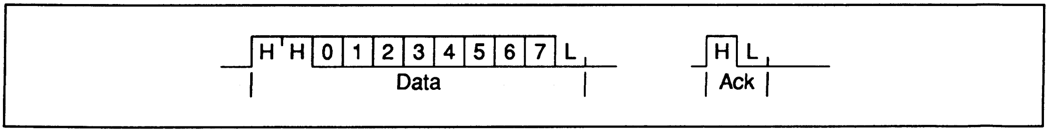


Figure 10.1 IMS T800 link data and acknowledge packets

Table 10.2 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.

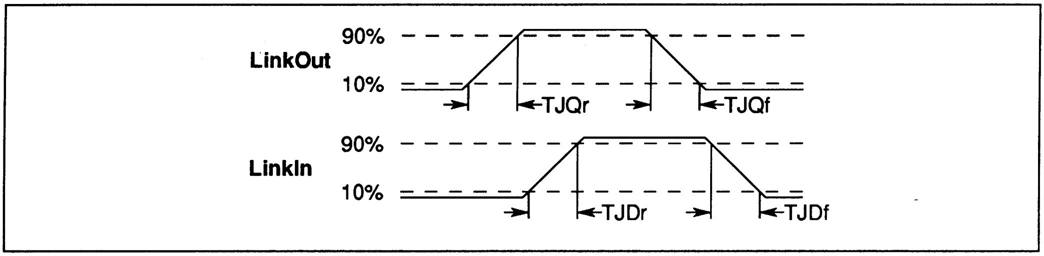


Figure 10.2 IMS T800 link timing

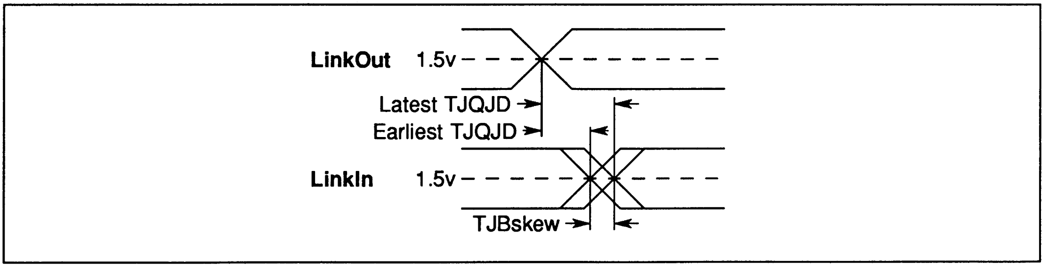


Figure 10.3 IMS T800 buffered link timing

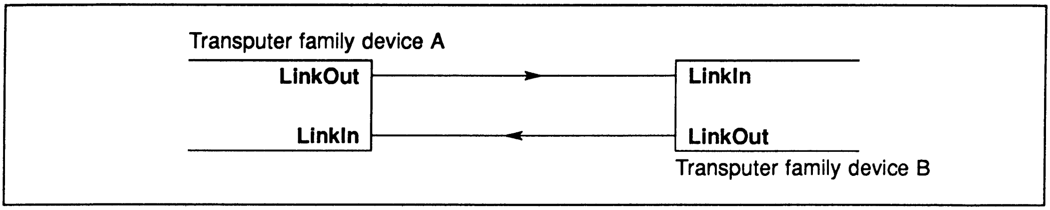


Figure 10.4 IMS T800 Links directly connected

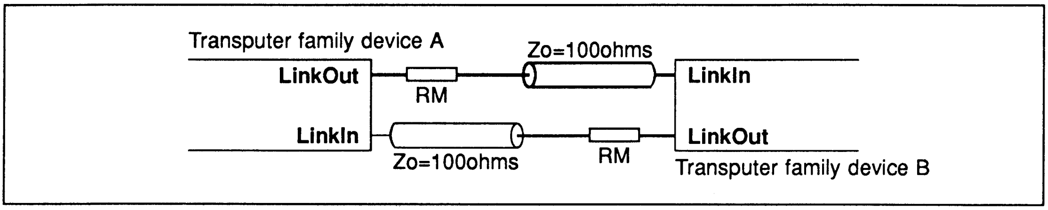


Figure 10.5 IMS T800 Links connected by transmission line

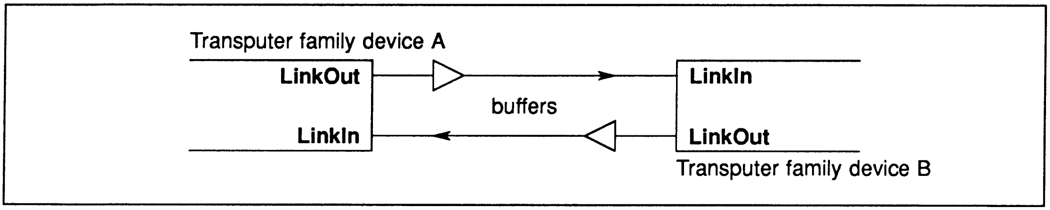


Figure 10.6 IMS T800 Links connected by buffers

11 Electrical specifications

11.1 DC electrical characteristics

Table 11.1 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 11.2 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 IMS T800-S	0	70	°C	3
TA	Operating temperature range IMS T800-M	-55	125	°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 11.3 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	1,2,3,7
		65	100	mA	1,2,4,7
		25	90	mA	1,2,3,8
		50	130	mA	1,2,4,8
IOZ	Tristate output current @ GND<VO<VCC		±10	μA	1,2
PD	Power dissipation		1.2	W	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 T800-S measured at 4.75V<**VCC**<5.25V and 0°C<**TA**<70°C.
Parameters for IMS T800-M measured at 4.75V<**VCC**<5.25V and -55°C<**TA**<125°C.
Input clock frequency = 5MHz.

3 Current sourced from non-link outputs.

4 Current sourced from link outputs.

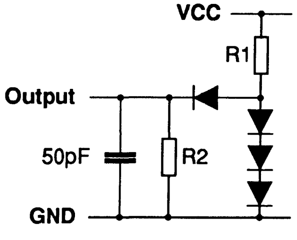
5 Power dissipation varies with output loading and program execution.
Power dissipation for processor operating at 20MHz.

6 This parameter is sampled and not 100% tested.

7 Parameter for IMS T800-S.

8 Parameter for IMS T800-M.

11.2 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 11.1 Load circuit for AC measurements

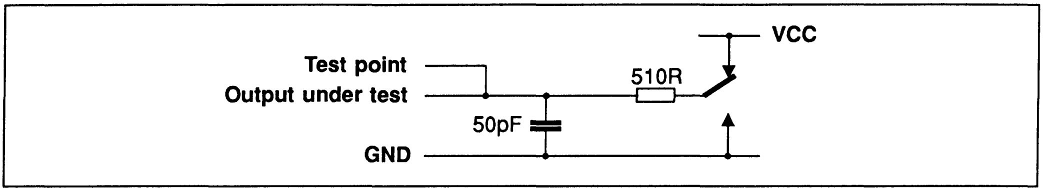


Figure 11.2 Tristate load circuit for AC measurements

11.3 AC timing characteristics

Table 11.4 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
TS0LaHZ	Address high to tristate	**a**	**a**+6	ns	3
TS0LaLZ	Address low to tristate	**a**	**a**+6	ns	3

Notes

1 Non-link pins; see section on links.

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

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

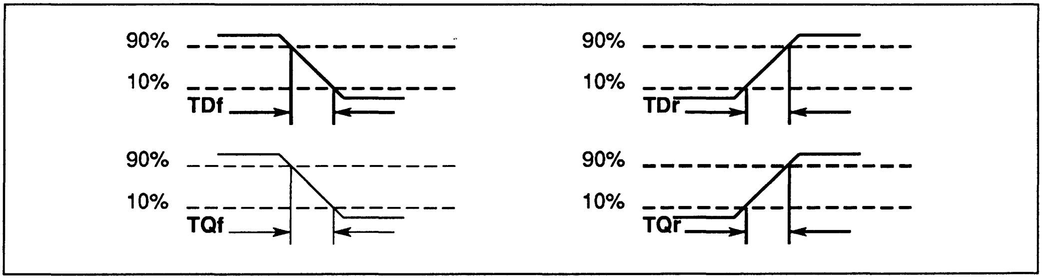


Figure 11.3 IMS T800 input and output edge timing

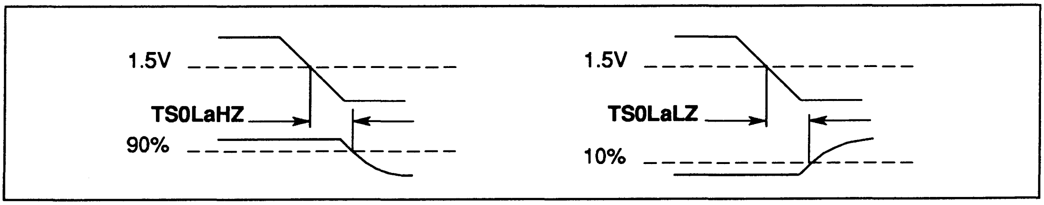


Figure 11.4 IMS T800 tristate timing relative to notMemS0

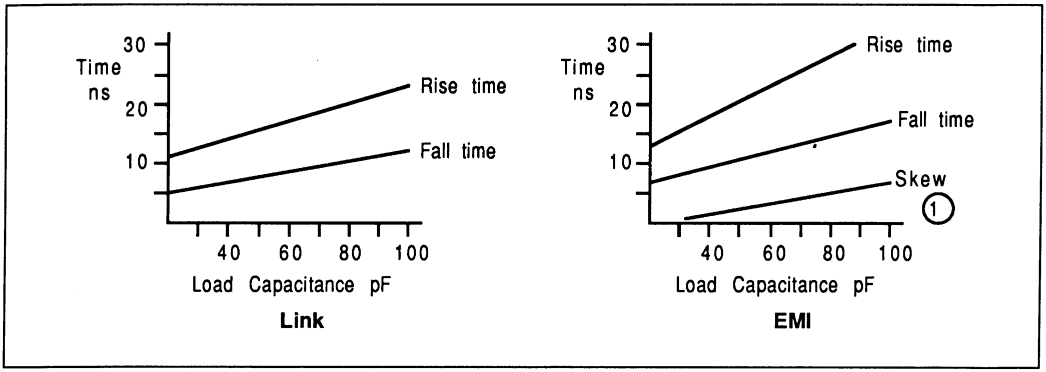


Figure 11.5 Typical rise/fall times

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.

11.4 Power rating

Internal power dissipation P_{INT} of transputer and peripheral chips depends on **VCC**, as shown in figure 11.6. 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.

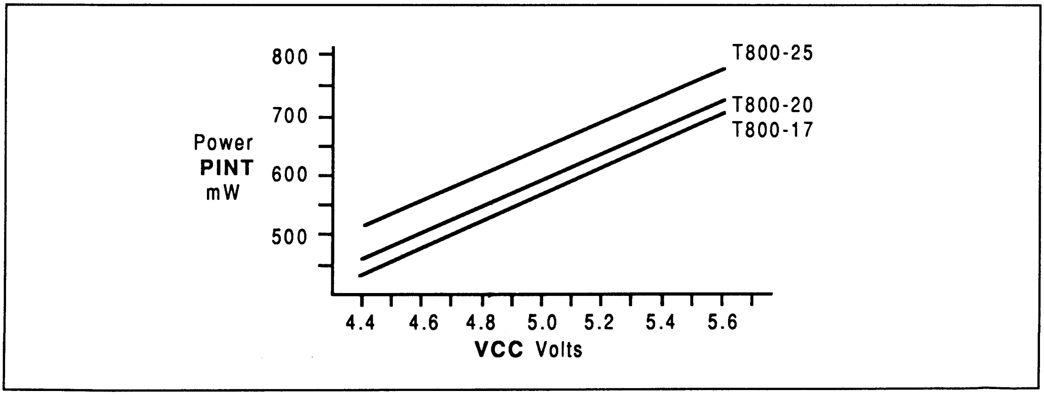


Figure 11.6 IMS T800 internal power dissipation vs VCC

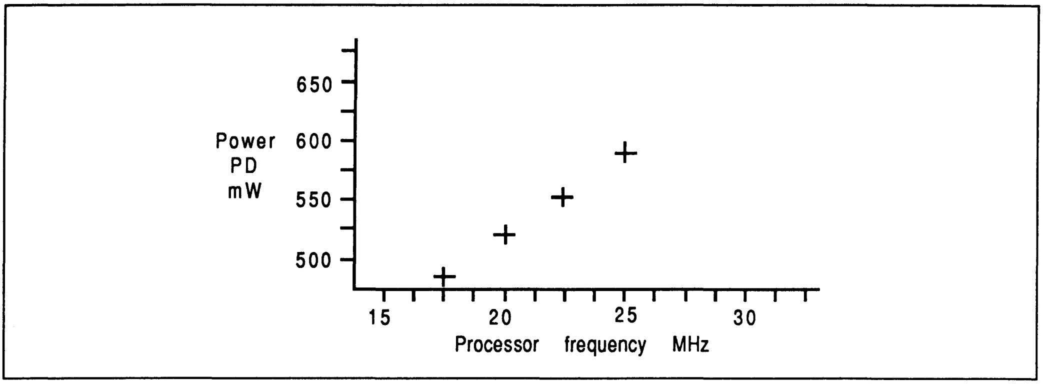


Figure 11.7 IMS T800 typical power dissipation with processor speed

12 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.

With transputers incorporating an FPU, this type of performance calculation is straight forward when considering only integer data types. However, when floating point calculations using the **REAL32** and **REAL64** data types are present in the program, complications arise due to the concurrency inherent in the transputer's design whereby integer calculations can be overlapped with floating point calculations. A more comprehensive guide to the impact of this concurrency on transputer performance can be found in The Transputer Instruction Set - A Compiler Writers' Guide.

12.1 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 12.1 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 12.1 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 12.2 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	39
/	2	40
REM	2	38
>> <<	2	3+**p**
Modulo Arithmetic operators		
PLUS	2	2
MINUS	1	1
TIMES (fast multiply, positive operand)	1	4+**Tbp**
TIMES (fast multiply, negative operand)	1	5+**Tbc**
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 12.3 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**+2**Et**	34+2**Eg**+2**Et**
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}

12.2 Fast multiply, TIMES

The IMS T800 has a fast integer multiplication instruction *product*. For a positive multiplier its execution time is 4+**Tbp** cycles, and for a negative multiplier 5+**Tbc** cycles (table 12.1). The time taken for a multiplication by zero is 3 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.

12.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic. In the IMS T800, floating point arithmetic is taken care of by the FPU. In table 12.4 **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 12.4 Arithmetic performance

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		34	8
LONGDIV		36	8
SHIFTRIGHT	(**n**<32)	4+**n**	8
	(**n**>=32)	**n**-27	
SHIFTLEFT	(**n**<32)	4+**n**	8
	(**n**>=32)	**n**-27	
NORMALISE	(**n**<32)	**n**+6	7
	(**n**>=32)	**n**-25	
	(**n**=64)	4	
ASHIFTRIGHT		**SHIFTRIGHT**+2	5
ASHIFTLEFT		**SHIFTLEFT**+4	5
ROTATERIGHT		**SHIFTRIGHT**	7
ROTATELEFT		**SHIFTLEFT**	7
FRACMUL		**LONGPROD**+4	5

† Assuming local variables.

12.4 IMS T800 floating point operations

All references to **REAL32** or **REAL64** operands within programs compiled for the IMS T800 normally produce the following performance figures.

Table 12.5 Floating point performance

	Size (bytes)	**REAL32** Time (cycles)	**REAL64** Time (cycles)
Names			
variables			
in expression	3.1	3	5
assigned to or input to	3.1	3	5
in **PROC** or **FUNCTION** call, corresponding to a **REAL** parameter	1.1+r	1.1+r	1.1+r
Arithmetic operators			
+ -	2	7	7
*	2	11	20
/	2	17	32
REM	11	19	34
Comparison operators			
=	2	4	4
<>	3	6	6
> <	2	5	5
>= <=	3	7	7
Conversions			
REAL32 to -	2		3
REAL64 to -	2	6	
To **INT32** from -	5	9	9
To **INT64** from -	18	32	32
INT32 to -	3	7	7
INT64 to -	14	24	22

12.4.1 IMS T800 floating point functions

These functions are provided by the development system. They are compiled directly into special purpose instructions designed to support the efficient implementation of some of the common mathematical functions of other languages. The functions provide **ABS** and **SQRT** for both **REAL32** and **REAL64** operand types.

Table 12.6 IMS T800 floating point arithmetic performance

Function	Cycles	+ cycles for parameter access †	
		REAL32	**REAL64**
ABS	2	8	
SQRT	118	8	
DABS	2		12
DSQRT	244		12

† Assuming local variables.

12.4.2 IMS T800 special purpose functions and procedures

The functions and procedures given in tables 12.8 and 12.9 are provided by the development system to give access to the special instructions available on the IMS T800. Table 12.7 shows the key to the table.

Table 12.7 Key to special performance table

Tb	most significant bit set in the word counting from zero
n	number of words per row (consecutive memory locations)
r	number of rows in the two dimensional move
nr	number of bits to reverse

Table 12.8 Special purpose functions performance

Function	Cycles	+ cycles for parameter access †
BITCOUNT	2+Tb	2
CRCBYTE	11	8
CRCWORD	35	8
BITREVNBIT	5+nr	4
BITREVWORD	36	2

† Assuming local variables.

Table 12.9 Special purpose procedures performance

Procedure	Cycles	+ cycles for parameter access †
MOVE2D	8+(2n+23)*r	8
DRAW2D	8+(2n+23)*r	8
CLIP2D	8+(2n+23)*r	8

† Assuming local variables.

12.5 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**. For the IMS T800, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

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 (**e**-3)/4. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 12.10 **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 12.10 External memory performance

	IMS T800	
	Program off chip	**Data off chip**
Boolean expressions	e-2	0
IF	3en-8	en
Replicated **IF**	(6e-4)n+7	(5e-2)n+8
Replicated **SEQ**	(3e-3)n+2	(4e-2)n
PAR	(3e-1)n+8	3en+4
Replicated **PAR**	(10e-8)n+8	16en-12
ALT	(2e-4)n+6e	(2e-2)n+10e-8
Array assignment and communication in one transputer	0	max (2e, e(b/2))

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 12.11 IMS T800 external memory performance

	Program	**e=2**	**e=3**	**e=4**	**e=5**	**On chip**
Program off chip	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
Data off chip	1	1.5	1.8	2.1	2.3	1
	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

12.6 Interrupt latency

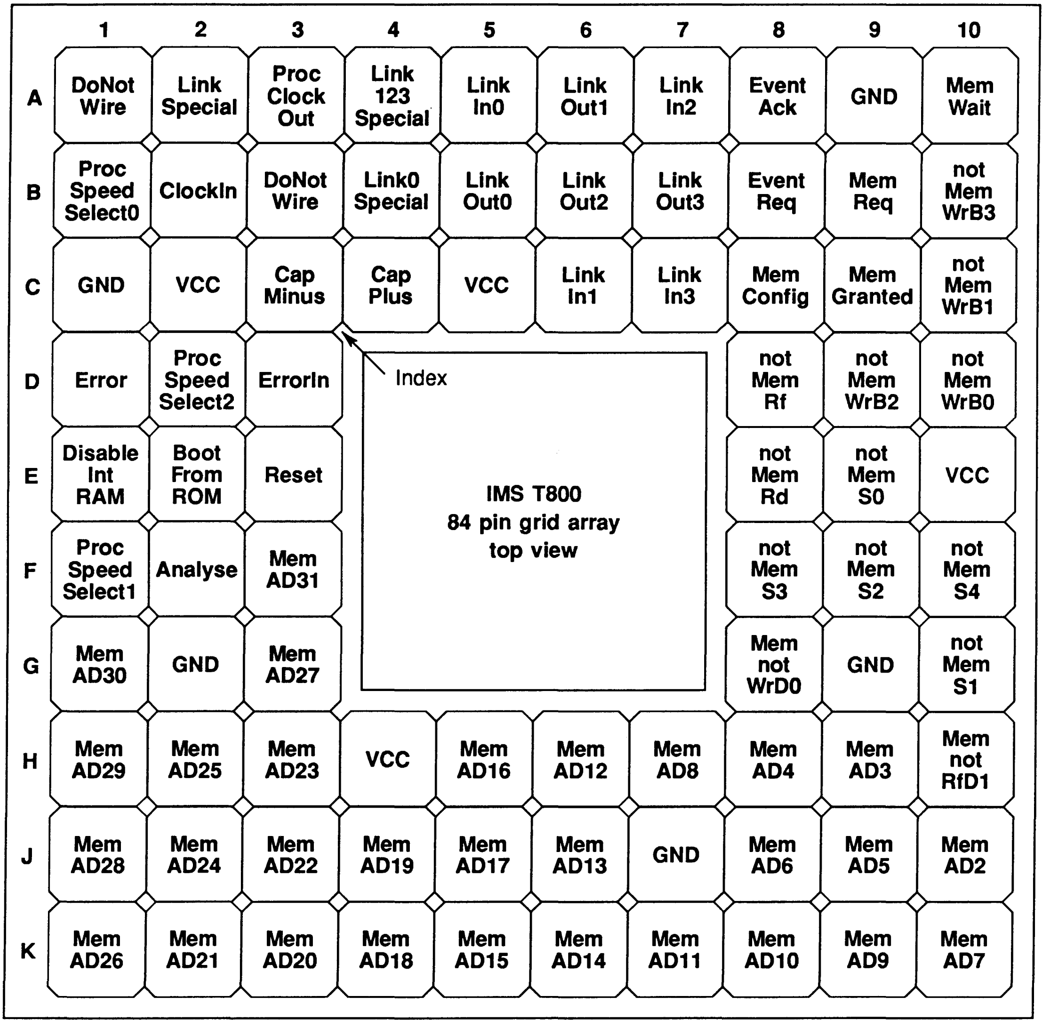
If the process is a high priority one and no other high priority process is running, the latency is as described in table 12.12. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

Table 12.12 Interrupt latency

	Typical		Maximum	
	TPCLPCL	**Tm**	**TPCLPCL**	**Tm**
IMS T800 with FPU in use	19	38	78	156
IMS T800 with FPU not in use	19	38	58	116

13 Package specifications

13.1 84 pin grid array package



	1	2	3	4	5	6	7	8	9	10
A	DoNot Wire	Link Special	Proc Clock Out	Link 123 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	not Mem WrB3
C	GND	VCC	Cap Minus	Cap Plus	VCC	Link In1	Link In3	Mem Config	Mem Granted	not Mem WrB1
D	Error	Proc Speed Select2	ErrorIn					not Mem Rf	not Mem WrB2	not Mem WrB0
E	Disable Int RAM	Boot From ROM	Reset					not Mem Rd	not Mem S0	VCC
F	Proc Speed Select1	Analyse	Mem AD31					not Mem S3	not Mem S2	not Mem S4
G	Mem AD30	GND	Mem AD27					Mem not WrD0	GND	not Mem S1
H	Mem AD29	Mem AD25	Mem AD23	VCC	Mem AD16	Mem AD12	Mem AD8	Mem AD4	Mem AD3	Mem not 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

Figure 13.1 IMS T800 84 pin grid array package pinout

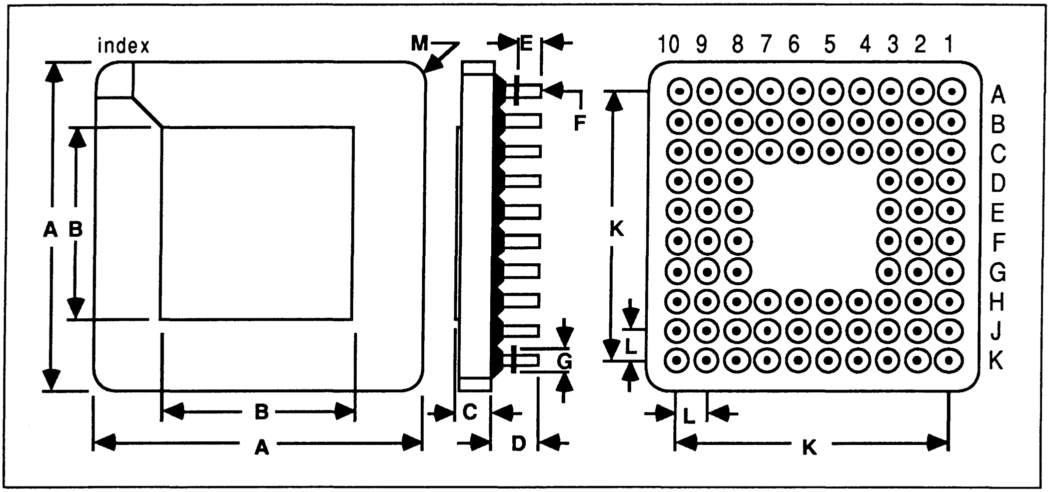


Figure 13.2 84 pin grid array package dimensions

Table 13.1 84 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	±0.005	
C	2.456	±0.278	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	±0.025	0.018	±0.001	Pin diameter
G	1.143	±0.127	0.045	±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 7.2 grams

Table 13.2 84 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	

14 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 14.1 IMS T800 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T800-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T800-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T800-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T800-G30S	30.0 MHz	33 ns	6.0	Ceramic Pin Grid
IMS T800-G17M	17.5 MHz	57 ns	3.5	Ceramic Pin Grid MIL Spec
IMS T800-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid MIL Spec

The timing parameters in this datasheet are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

inmos

IMS T425 transputer

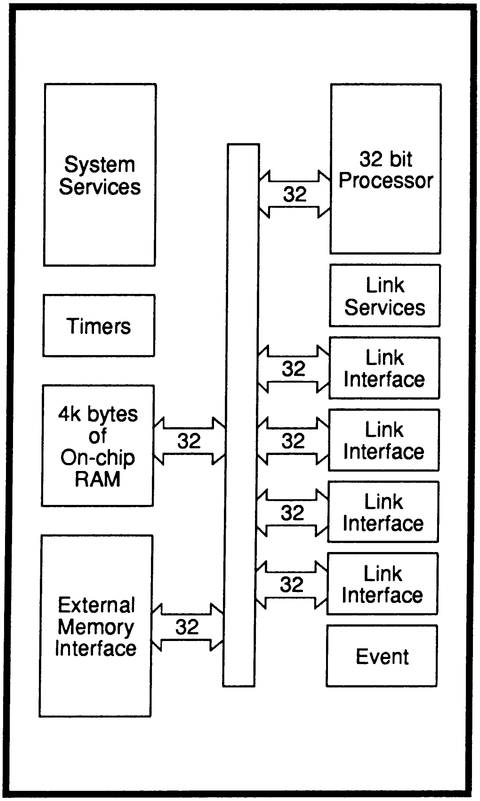
Advance Data

FEATURES

32 bit architecture
33 ns internal cycle time
30 MIPS (peak) instruction rate
Pin compatible with IMS T800 and IMS T425
4 Kbytes on-chip static RAM
120 Mbytes/sec sustained data rate to internal memory
4 Gbytes directly addressable external memory
40 Mbytes/sec sustained data rate to external memory
630 ns response to interrupts
Four INMOS serial links 5/10/20 Mbits/sec
High performance graphics support with block move instructions
Boot from ROM or communication links
Single 5 MHz clock input
Single +5V ±5% power supply
MIL-STD-883C processing will be available

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
Pattern recognition
Artificial intelligence



1 Introduction

The IMS T425 transputer is a 32 bit CMOS microcomputer with 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.

For convenience of description, the IMS T425 operation is split into the basic blocks shown in figure 1.1.

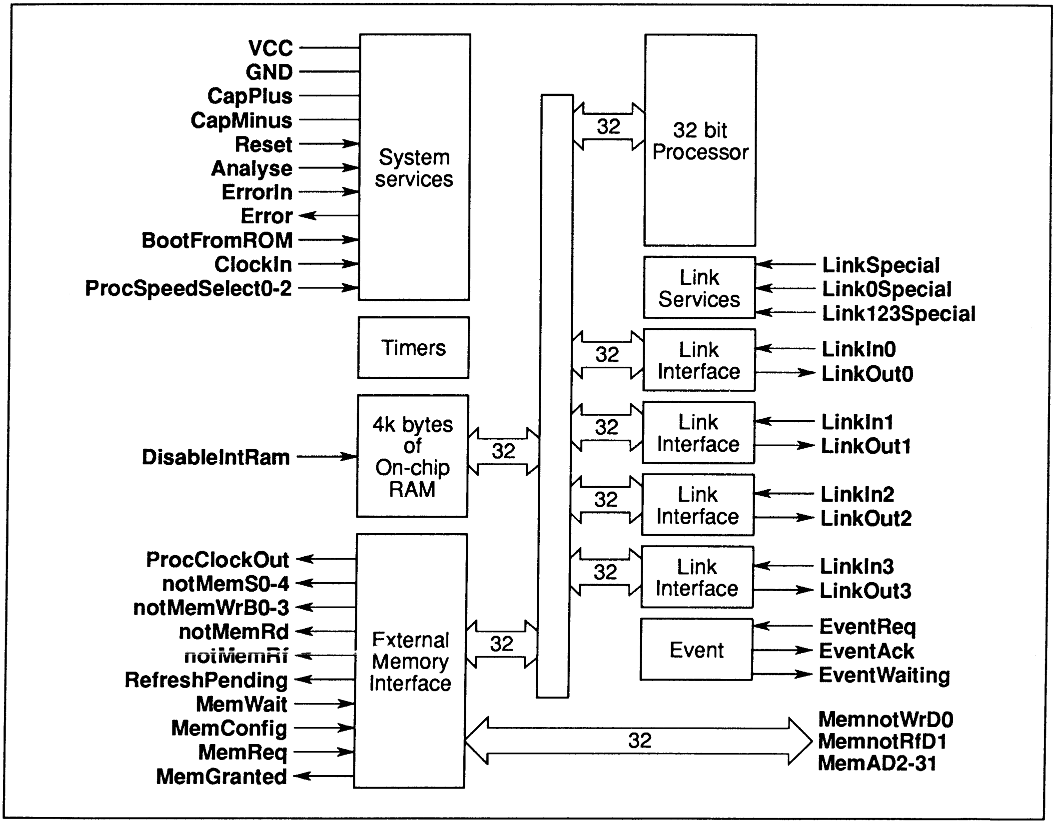


Figure 1.1 IMS T425 block diagram

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 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

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 T425, useful for pattern recognition, is the facility to count bits set in a word.

The IMS T425 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 bootstrap 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 T425 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

The IMS T425 is pin compatible with the IMS T800 and can be plugged directly into a circuit designed for that device. It has a number of additions to improve hardware interfacing and to facilitate software initialising and debugging. The improvements have been made in an upwards-compatible manner. Software should be recompiled, although no changes to the source code are necessary.

The IMS T425-20 is also pin compatible with the IMS T414-20, as the extra inputs used are all held to ground on the IMS T414. The IMS T425-20 can thus be plugged directly into a circuit designed for a 20 MHz version of the IMS T414.

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. The instruction set of the IMS T425 is the same as that of the IMS T800, except that the IMS T800 floating point instructions are replaced by the IMS T414 floating point support instructions.

This data sheet supplies hardware implementation and characterisation details for the IMS T425. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of occam.

2 Pin designations

Table 2.1 IMS T425 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
DisableIntRAM	in	Disable internal RAM

Table 2.2 IMS T425 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
RefreshPending	out	Dynamic refresh is pending
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemConfig	in	Memory configuration data input

Table 2.3 IMS T425 event

Pin	In/Out	Function
EventReq	in	Event request
EventAck	out	Event request acknowledge
EventWaiting	out	Event input requested by software

Table 2.4 IMS T425 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 174.

3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 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 4 Gbytes 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.

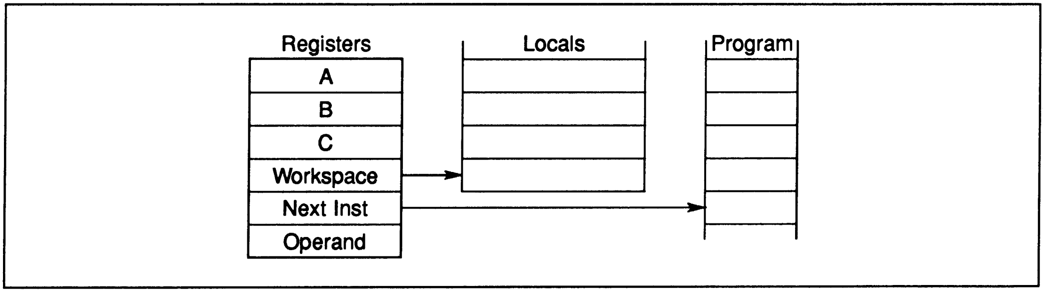


Figure 3.1 Registers

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.

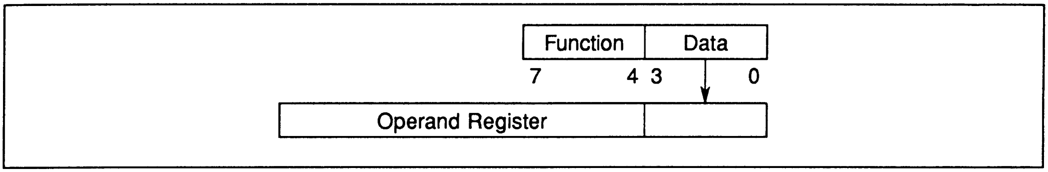


Figure 3.2 Instruction format

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, 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 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 four 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 121).

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 121). 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.

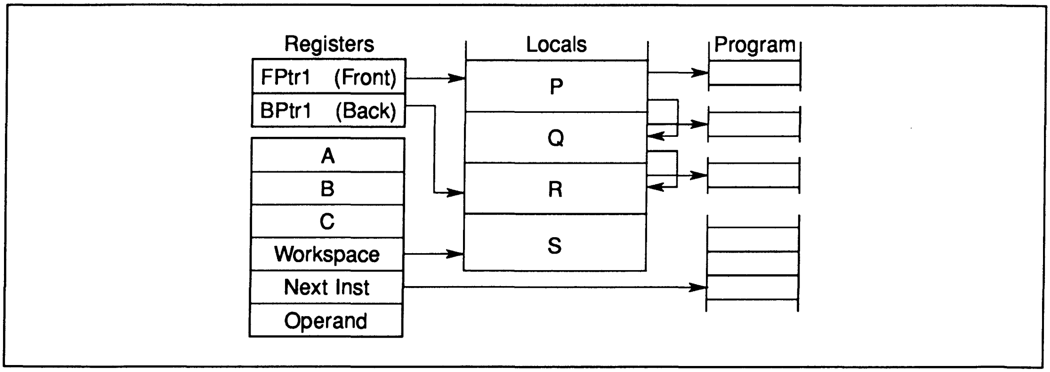


Figure 3.3 Linked process list

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 124). 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 124). 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 T425 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 124).

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 58 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 32 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 4295 seconds. 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 76 hours.

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.

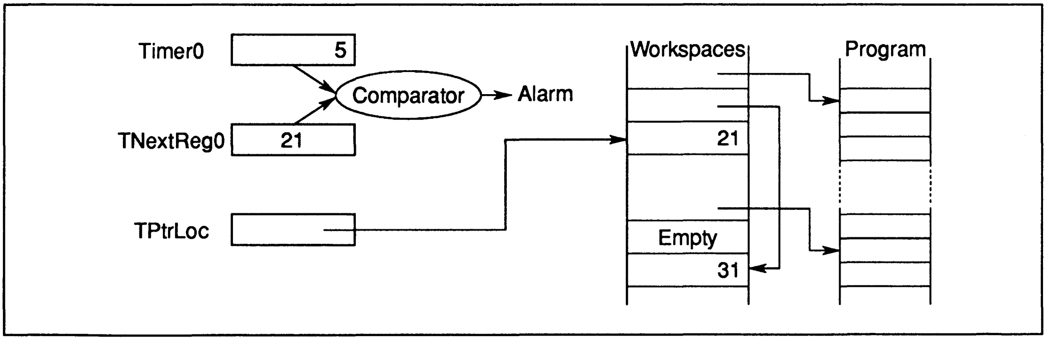


Figure 3.4 Timer registers

4 Instruction set summary

The Function Codes table 4.7. gives the basic function code set (page 118). 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* #FFFFFFE1)		
is coded as			
nfix	#1	#6	#61
ldc	#1	#4	#41

Tables 4.8 to 4.21 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.
m	Bit number of the highest bit set in the absolute value of 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.
p	Number of words per row.
r	Number of rows.

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	124
E	The instruction will affect the *Error* flag	124, 135

4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 120). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 134).

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 135) directly.

Table 4.6 Error setting instructions

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

Table 4.7 IMS T425 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 T425 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	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
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*	

Table 4.9 IMS T425 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	33	long multiply	
1A	21FA	ldiv	35	long divide	E
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)	

Table 4.10 IMS T425 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	
5A	25FA	dup	1	duplicate top of stack	
79	27F9	pop	1	pop processor stack	

Table 4.11 IMS T425 floating point support operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
73	27F3	cflerr	3	check floating point error	E
9C	29FC	fptesterr	1	load value true (FPU not present)	
63	26F3	unpacksn	15	unpack single length fp number	
6D	26FD	roundsn	12/15	round single length fp number	
6C	26FC	postnormsn	5/30	post-normalise correction of single length fp number	
71	27F1	ldinf	1	load single length infinity	

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

Table 4.12 IMS T425 2D block move operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
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	

Table 4.13 IMS T425 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
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 word	

Table 4.14 IMS T425 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	
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	4	store byte	
4A	24FA	move	2**w**+8	move message	

Table 4.15 IMS T425 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 4.16 IMS T425 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 4.17 IMS T425 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 4.18 IMS T425 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 4.19 IMS T425 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 4.20 IMS T425 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	
17C	2127FC	lddevid	1	load device identity	
7E	27FE	ldmemstartval	1	load value of memstart address	

Table 4.21 IMS T425 debugger support codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	00	jump 0	3	jump 0 (break not enabled)	D
			11	jump 0 (break enabled, high priority)	
			13	jump 0 (break enabled, low priority)	
B1	2BF1	break	9	break (high priority)	
			11	break (low priority)	
B2	2BF2	clrj0break	1	clear jump 0 break enable flag	
B3	2BF3	setj0break	1	set jump 0 break enable flag	
B4	2BF4	testj0break	2	test jump 0 break enable flag set	
7A	27FA	timerdisableh	1	disable high priority timer interrupt	
7B	27FB	timerdisablel	1	disable low priority timer interrupt	
7C	27FC	timerenableh	6	enable high priority timer interrupt	
7D	27FD	timerenablel	6	enable low priority timer interrupt	

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 10 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.

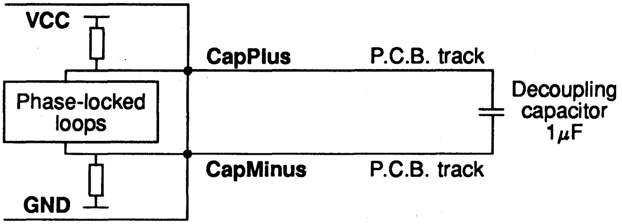


Figure 5.1 Recommended PLL decoupling

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).

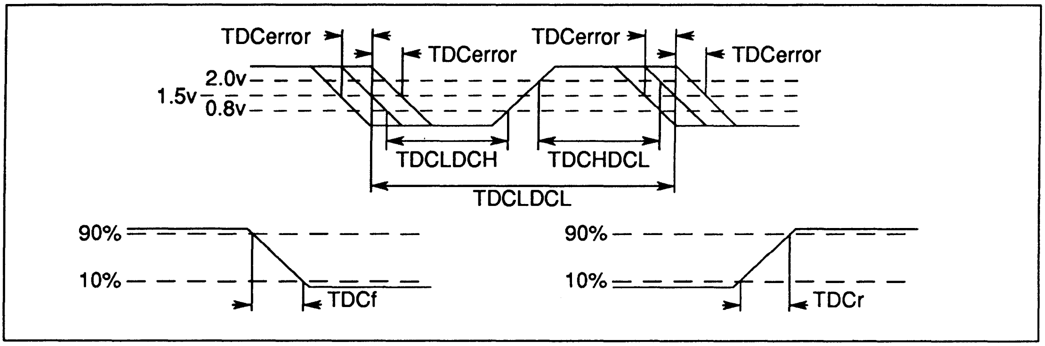


Figure 5.2 ClockIn timing

5.4 ProcSpeedSelect0-2

Processor speed of the IMS T425 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 T425 can be plugged directly into a board designed for a 20 MHz 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.

Table 5.2 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.

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 bootstrap (page 157).

After the end of **Reset** there will be a delay of 144 periods of **ClockIn** (figure 5.3). Following this, the **MemWrD0**, **MemRfD1** and **MemAD2-31** pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 147). 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 (page 148), 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.

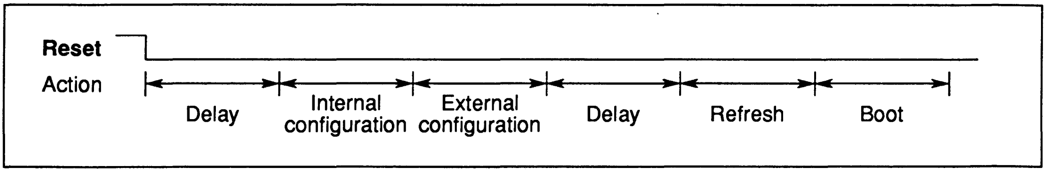


Figure 5.3 IMS T425 post-reset sequence

5.6 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 #7FFFFFFE. 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 136).

Table 5.3 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.

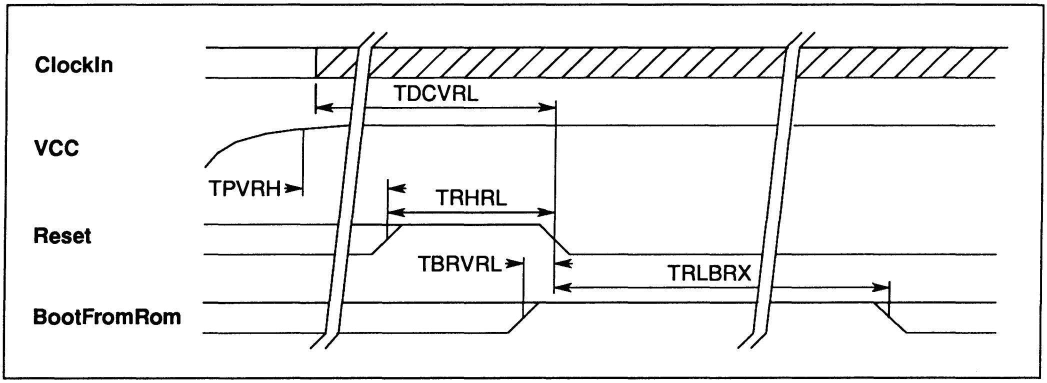


Figure 5.4 Transputer reset timing with Analyse low

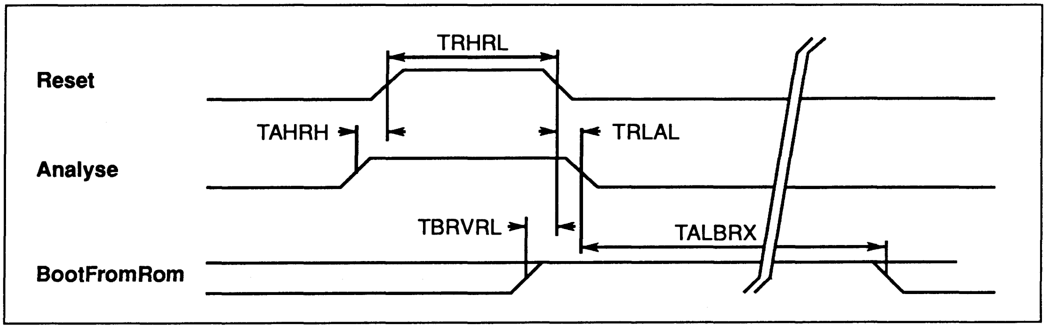


Figure 5.5 Transputer reset and analyse timing

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.

5.7 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 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 bootstrap program. 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 (page 124). 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, HaltOnError** and **EnableJ0Break** are normally cleared at reset on the IMS T425; however, if **Analyse** is asserted the flags are not altered. 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 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 5.4.

Table 5.4 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.

5.9 Error, ErrorIn

The **Error** pin carries the OR'ed output 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 (page 124). 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 134).

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. **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 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 '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. 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 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 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*.

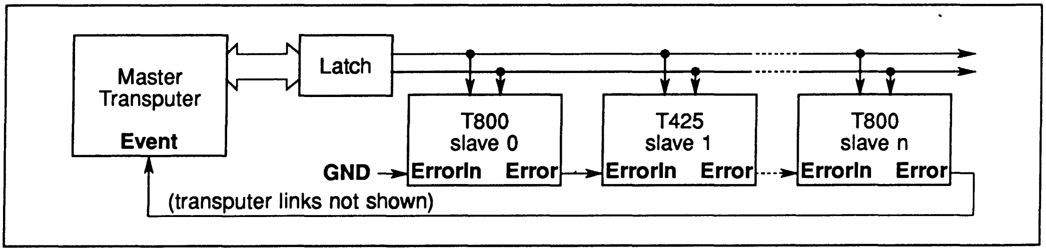


Figure 5.6 Error handling in a multi-transputer system

6 Memory

The IMS T425 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 138). The transputer can also access 4 Gbytes 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 T425 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*. An instruction *ldmemstartval* is provided to obtain the value of **MemStart**.

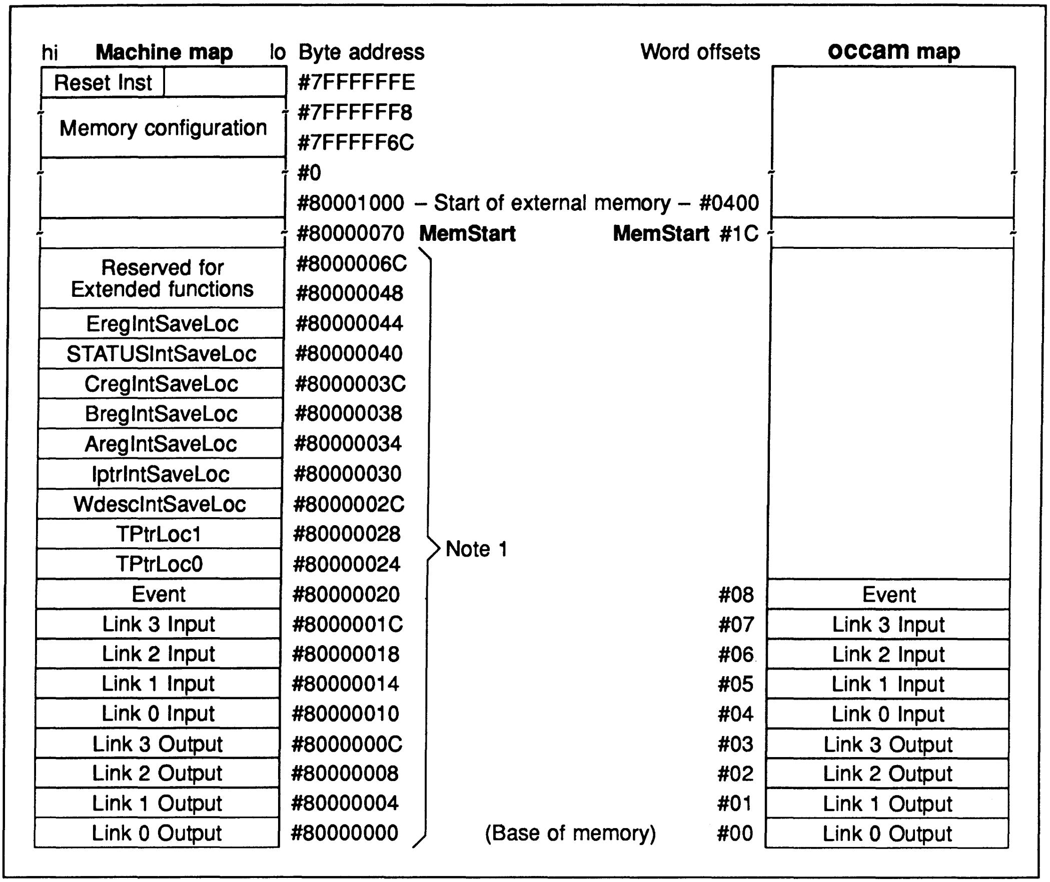
The context of a process in the transputer model involves a workspace descriptor (**WPtr**) and an instruction pointer (**IPtr**). **WPtr** is a word address pointer to a workspace in memory. **IPtr** points to the next instruction to be executed for the process which is the currently executing process. The context switch performed by the breakpoint instruction swaps the **WPtr** and **IPtr** of the currently executing process with the **WPtr** and **IPtr** held above **MemStart**. Two contexts are held above **MemStart**, one for high priority and one for low priority; this allows processes at both levels to have breakpoints. Note that on bootstrapping from a link, these contexts are overwritten by the loaded code. If this is not acceptable, the values should be peeked from memory before bootstrapping from a link.

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.

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			
	#80000048			
EregIntSaveLoc	#80000044			
STATUSIntSaveLoc	#80000040			
CregIntSaveLoc	#8000003C			
BregIntSaveLoc	#80000038			
AregIntSaveLoc	#80000034			
IptrIntSaveLoc	#80000030			
WdescIntSaveLoc	#8000002C			
TPtrLoc1	#80000028	Note 1		
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

Figure 6.1 IMS T425 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 134). 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 (page 147). If none are suitable the user can configure the interface to specific requirements, as shown in page 148.

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).

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 7.4.

The timing parameters in the following tables are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

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 (figure 7.11).

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
Tm	ProcClockOut half cycle	**b**-0.5	**b**	**b**+0.5	ns	2
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.

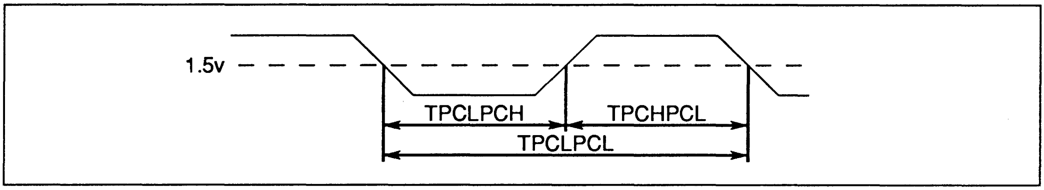


Figure 7.1 IMS T425 ProcClockOut timing

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 (page 134).

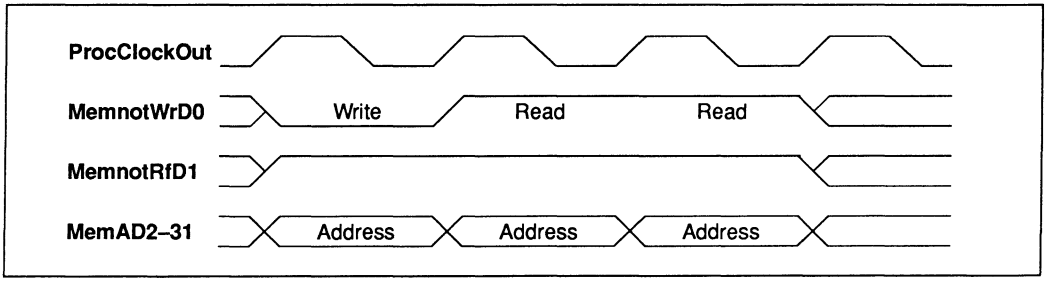


Figure 7.2 IMS T425 bus activity for internal memory cycle

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 transputer 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 transputer 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** (page 140) 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**.

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 (page 155). 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 low. Figure 7.5 shows the effect of **Wait** on strobes in more detail; each division on the scale is one period **Tm**.

Table 7.2 Read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

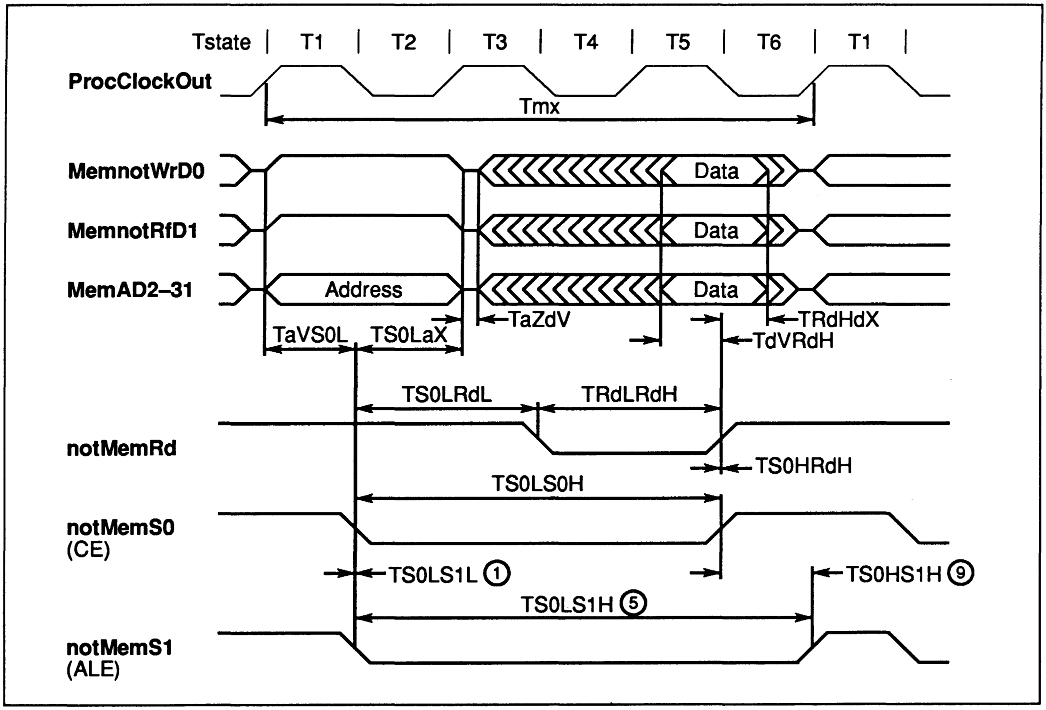


Figure 7.3 IMS T425 external read cycle: static memory

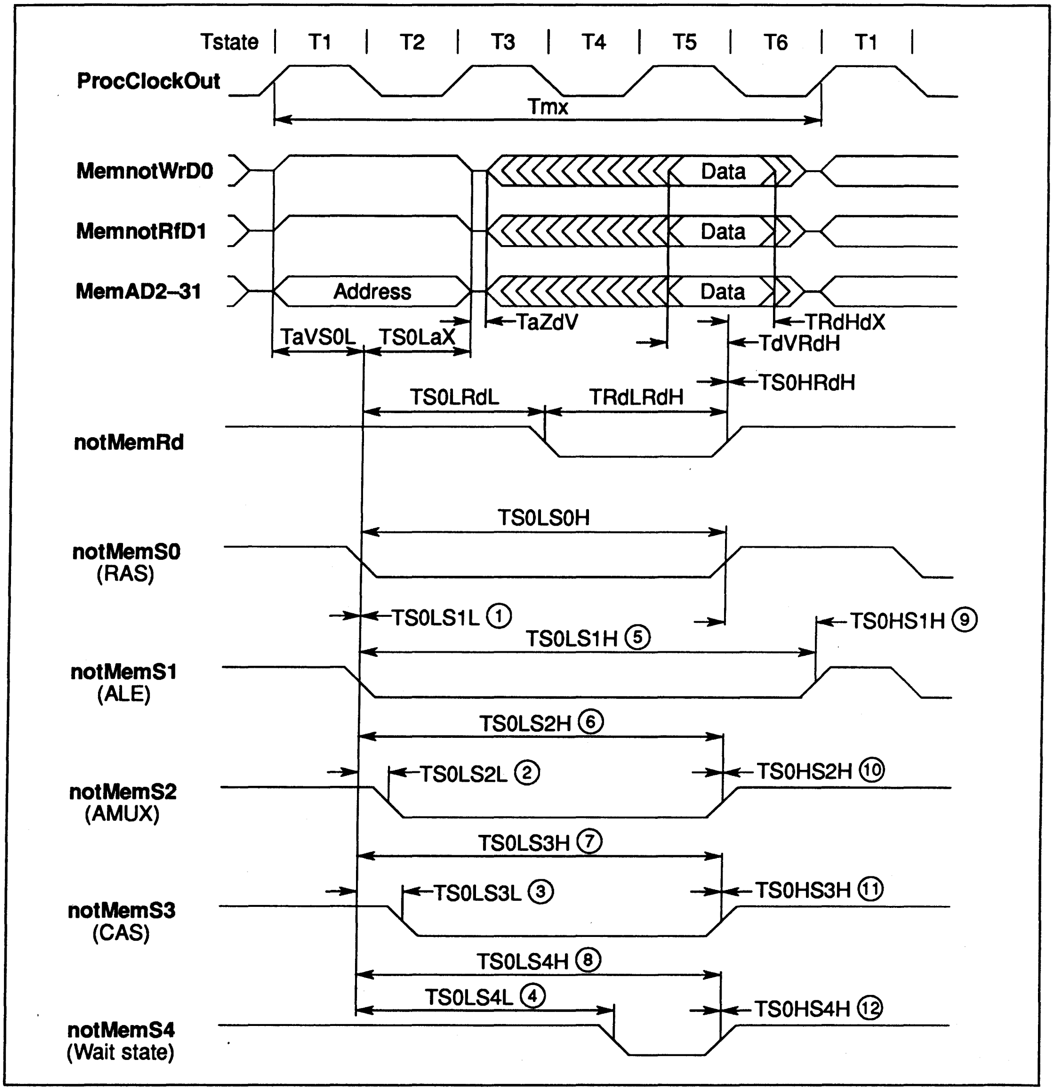


Figure 7.4 IMS T425 external read cycle: dynamic memory

Table 7.3 IMS T425 strobe timing

SYMBOL	ⓝ	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

Figure 7.5 IMS T425 effect of wait states on strobes

Table 7.4 Strobe S0 to ProcClockOut skew

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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	

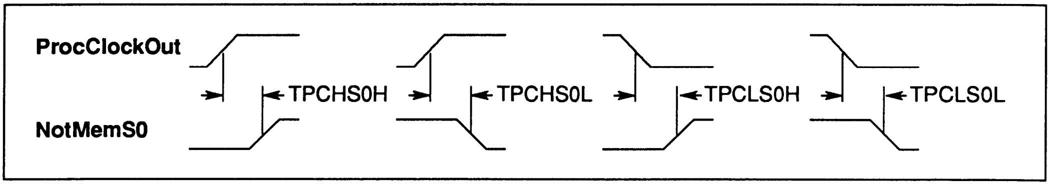


Figure 7.6 IMS T425 skew of notMemS0 to ProcClockOut

7.9 notMemWrB0-3

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

The transputer 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.

Table 7.5 Write

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

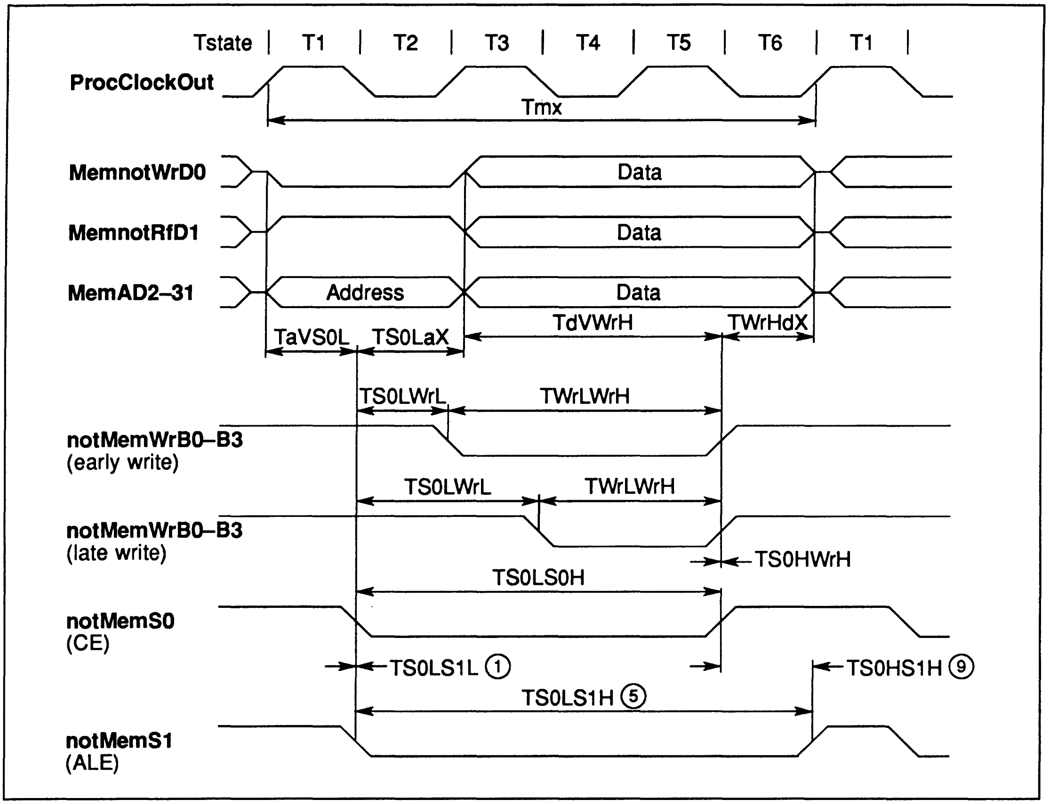


Figure 7.7 IMS T425 external write cycle

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.

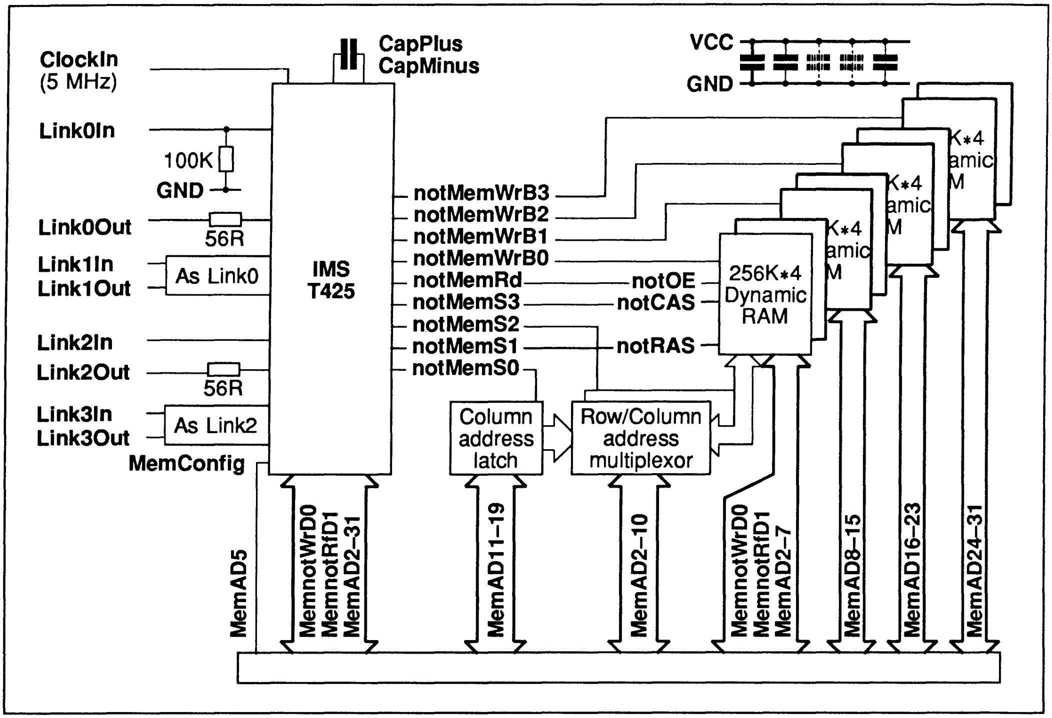


Figure 7.8 IMS T425 dynamic RAM application

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 17 of the possible configurations are valid, all others remain at the default configuration.

Table 7.6 IMS T425 internal configuration coding

	Duration of each Tstate periods Tm						**Strobe coefficient**				**Write cycle**	**Refresh interval**	**Cycle time**
Pin	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**s1**	**s2**	**s3**	**s4**	**type**	**ClockIn cycles**	**Proc cycles**
MemnotWrD0	1	1	1	1	1	1	30	1	3	5	late	72	3
MemnotRfD1	1	2	1	1	1	2	30	1	2	7	late	72	4
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6
MemAD8	1	1	1	1	1	1	30	1	2	3	early	†	3
MemAD9	1	1	2	1	2	1	30	2	5	9	early	†	4
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9
MemAD12	1	1	2	1	2	1	4	1	2	3	early	72	4
MemAD13	2	1	2	1	2	2	5	1	2	3	early	72	5
MemAD14	2	2	2	1	3	2	6	1	3	4	early	72	6
MemAD15	2	1	2	3	3	3	8	1	2	3	early	72	7
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12

† Provided for static RAM only.

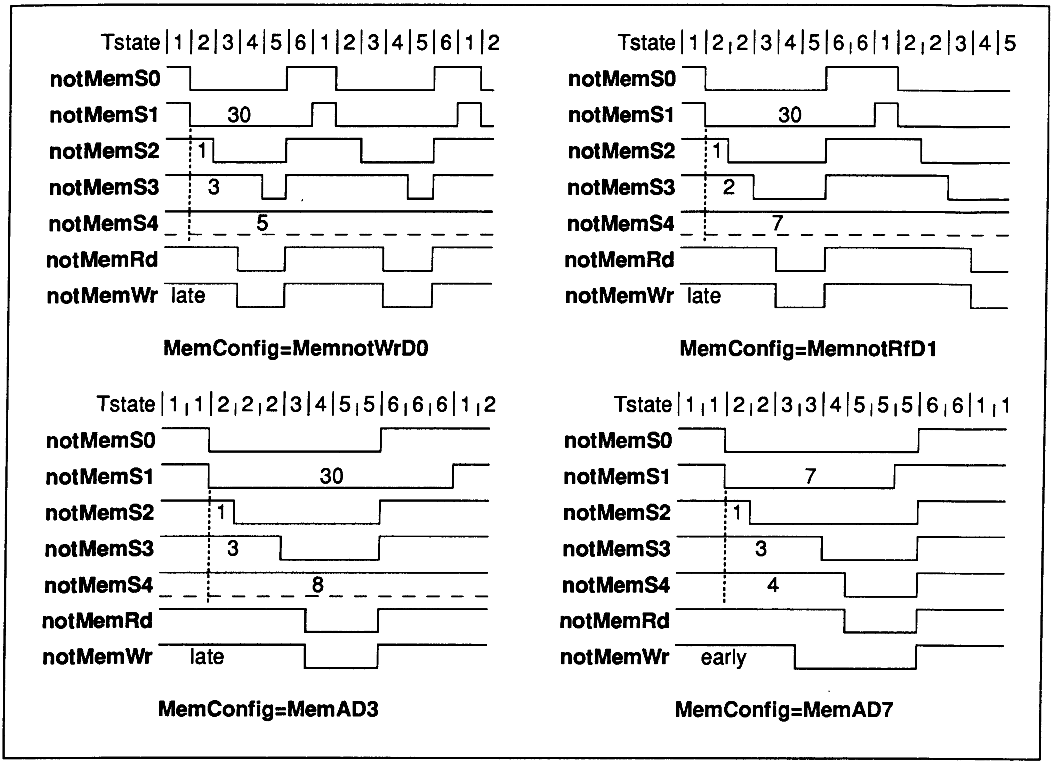


Figure 7.9 IMS T425 internal configuration

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 table 7.7, 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; late write.

The external memory configuration table 7.7 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.

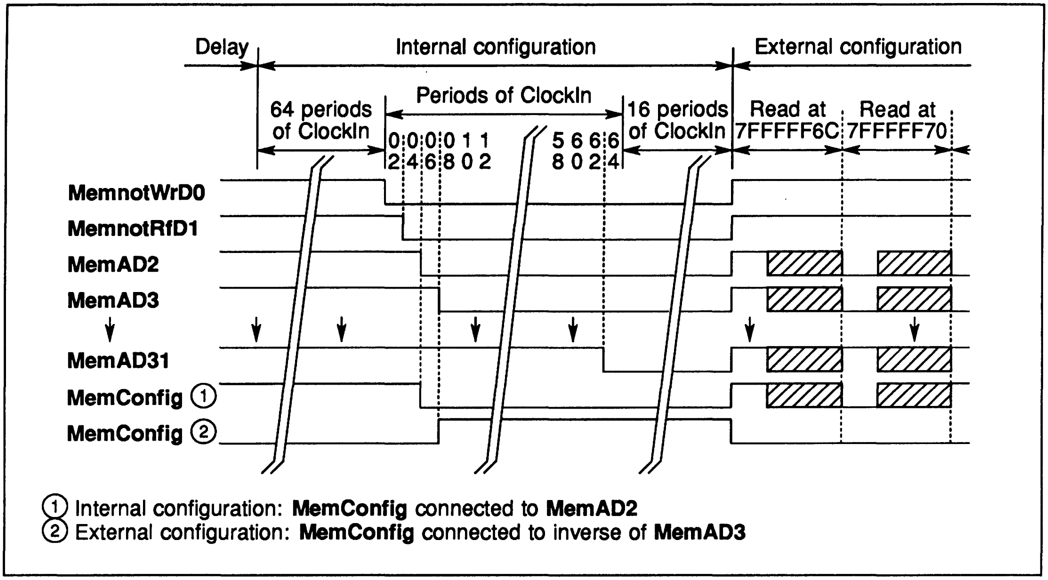


Figure 7.10 IMS T425 internal configuration scan

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 page 140.

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** (page 138).

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. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

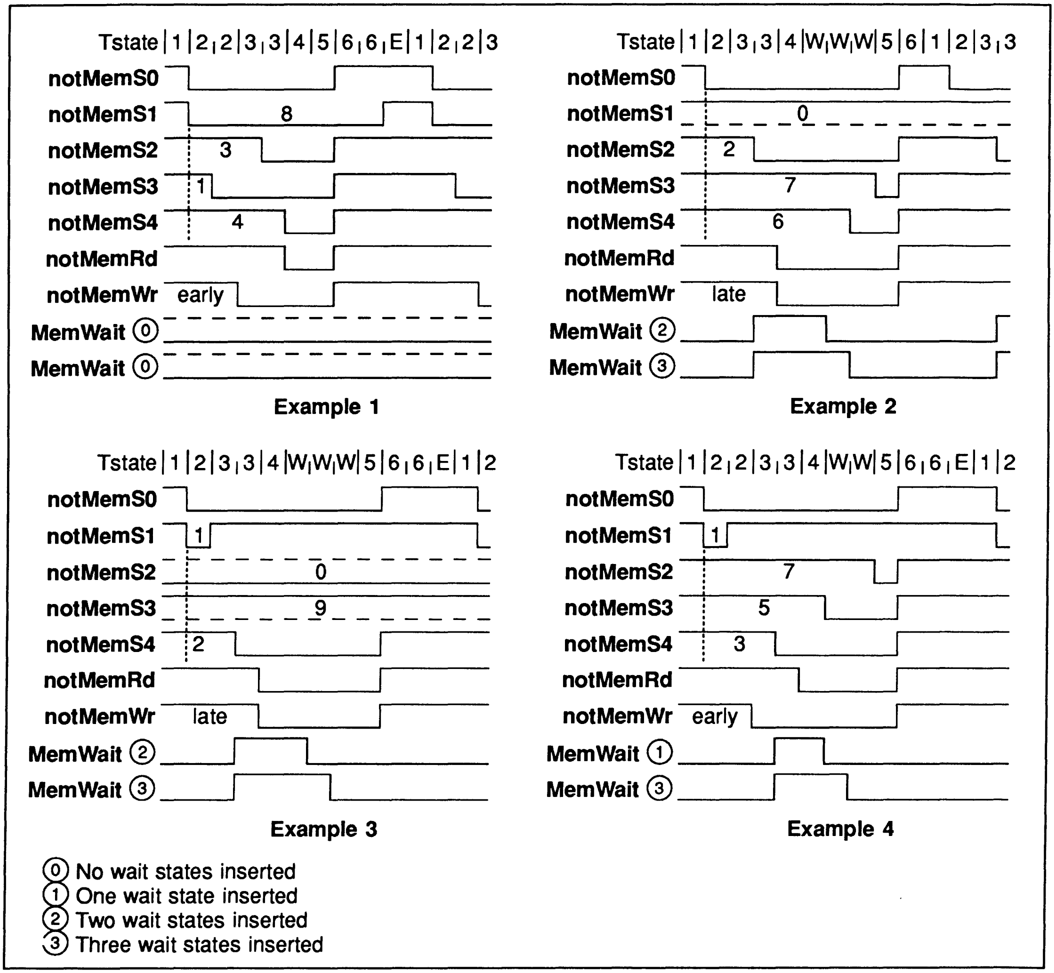
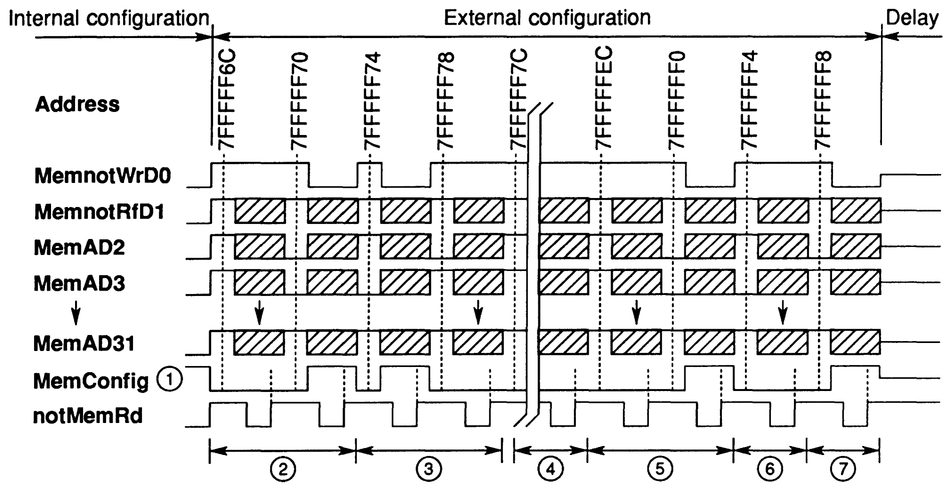


Figure 7.11 IMS T425 external configuration



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

Figure 7.12 IMS T425 external configuration scan

Table 7.7 IMS T425 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

Table 7.8 IMS T425 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 5 MHz:

$$\text{Interval} = 18 * 200 = 3600 \text{ ns}$$

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

Table 7.9 Memory configuration

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMCVRdH	Memory configuration data setup	30			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**.

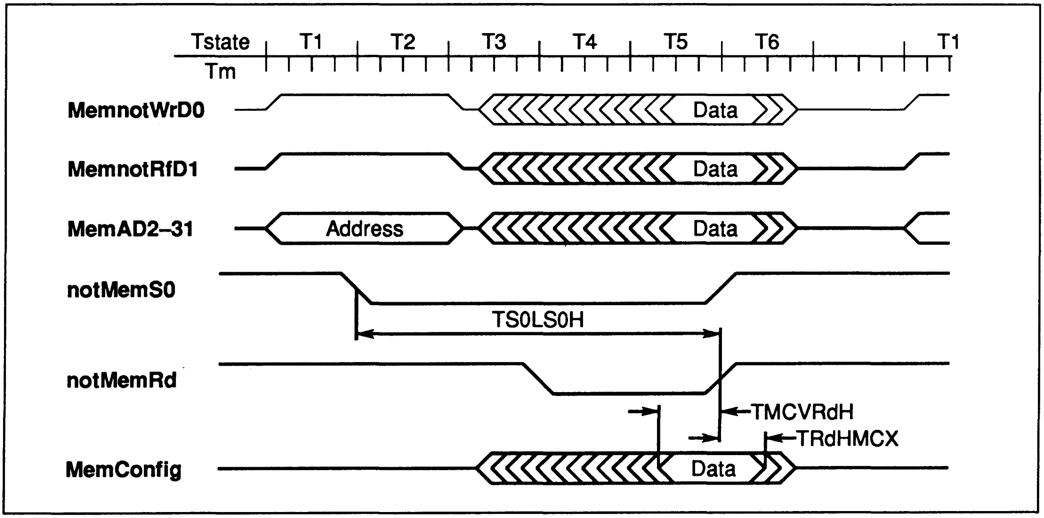


Figure 7.13 IMS T425 external configuration read cycle timing

7.11 RefreshPending

When high, this pin signals that a refresh cycle is pending. If no DMA is active then refresh will be performed following the end of the current internal or external memory cycle. If DMA is active the transputer will wait for DMA to terminate before commencing the refresh cycle. Unlike **MemnotRfD1**, **RefreshPending** is never tristated and can thus be interrogated by the DMA device; the DMA cycle can then be suspended, at the discretion of the DMA device, to allow refresh to take place.

The simple circuit of Figure 7.14 will suspend DMA requests from the external logic when **RefreshPending** is asserted, so that a memory refresh cycle can be performed. DMA is restored on completion of the refresh cycle. The transputer will not perform an external memory cycle other than a refresh cycle, using this method, until the requesting device removes its DMA request.

RefreshPending can change state only on a rising edge of **ProcClockOut**. It remains high during the refresh period. **MemReq** may be re-asserted any time after the commencement of the refresh cycle. This function is not available on the IMS T414 and IMS T800.

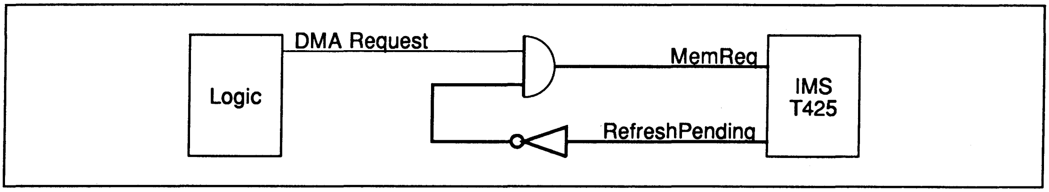


Figure 7.14 IMS T425 refresh with DMA

7.12 notMemRf

The IMS T425 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. **MemWait** operates normally during refresh cycles.

Table 7.10 Memory refresh

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

Notes

1 **a** is total **Tmx+Tm**.

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

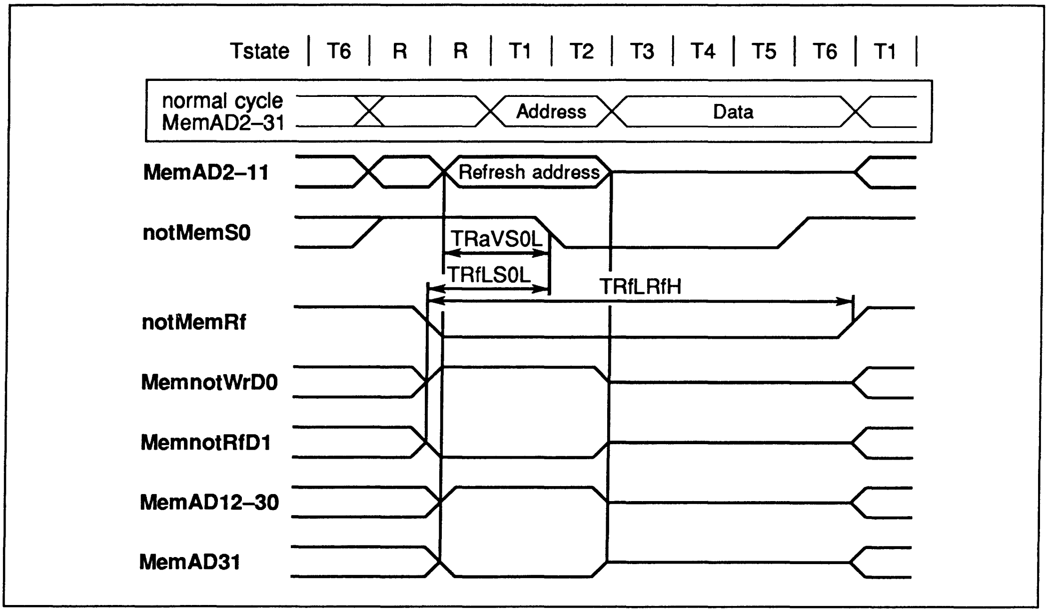


Figure 7.15 IMS T425 refresh cycle timing

7.13 MemWait

Taking **MemWait** high with the timing shown will extend the duration of **T4**. **MemWait** is sampled relative to the falling edge of **ProcClockOut** during a **T3** period, 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. **MemWait** operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

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.

Table 7.11 Memory wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHWtH	Wait setup	0.5**Tm**+3			ns	1,2
TPCHWtL	Wait hold	0.5**Tm**+3			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	2**Tm**			ns	

Notes

1 **ProcClockOut** load should not exceed 50pf.

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

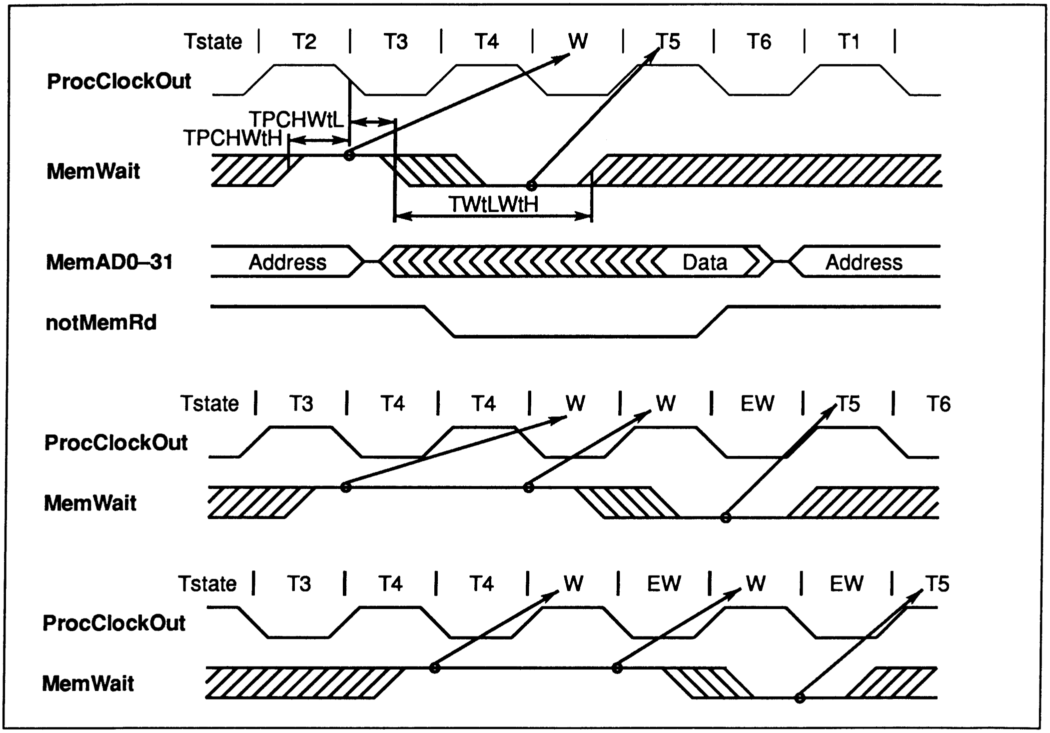


Figure 7.16 IMS T425 memory wait timing

7.14 MemReq, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer 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 7.8), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

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 7.12 Memory request

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4		7	Tm	1
TMRLMGL	Memory request end response time	2		5	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**).

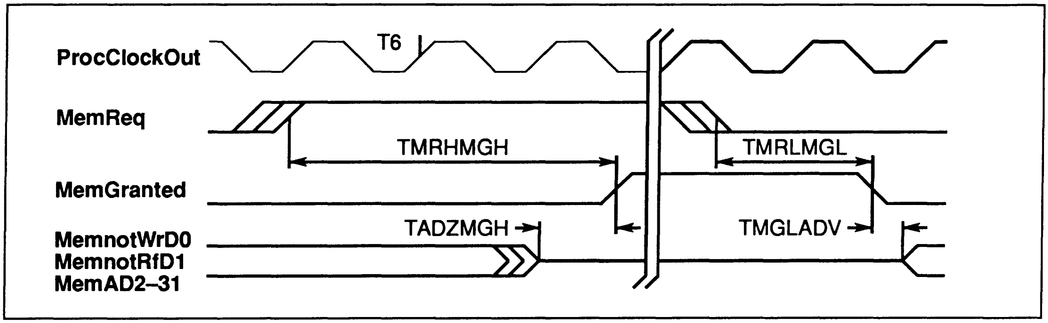


Figure 7.17 IMS T425 memory request timing

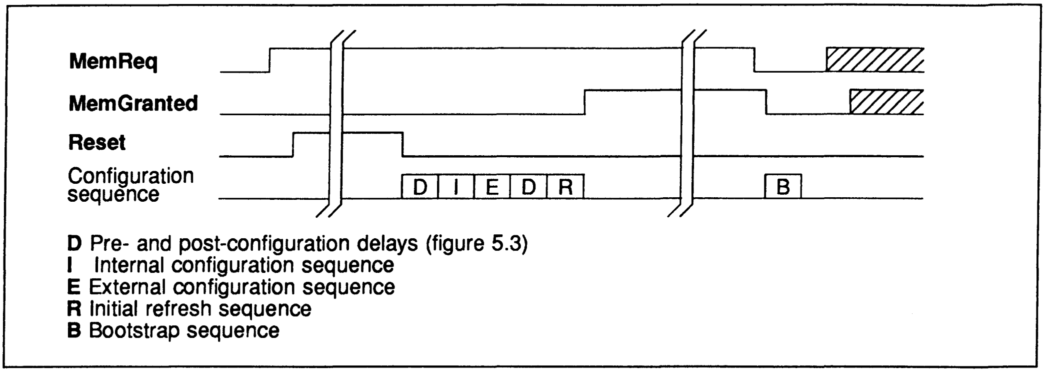


Figure 7.18 IMS T425 DMA sequence at reset

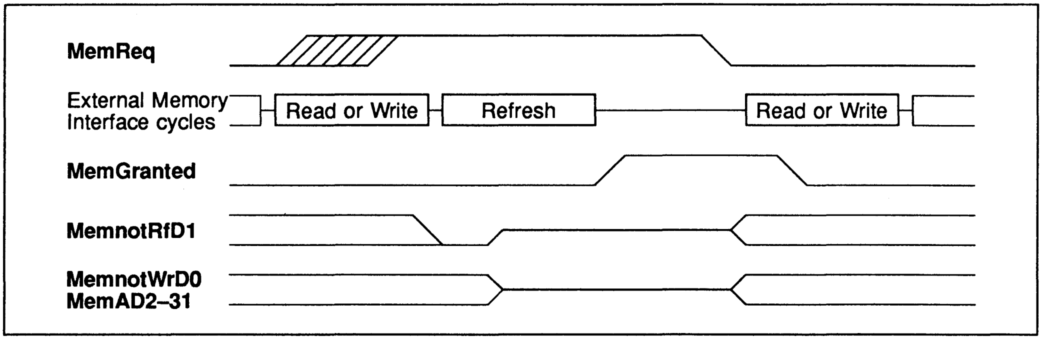


Figure 7.19 IMS T425 operation of MemReq, MemGranted with external, refresh memory cycles

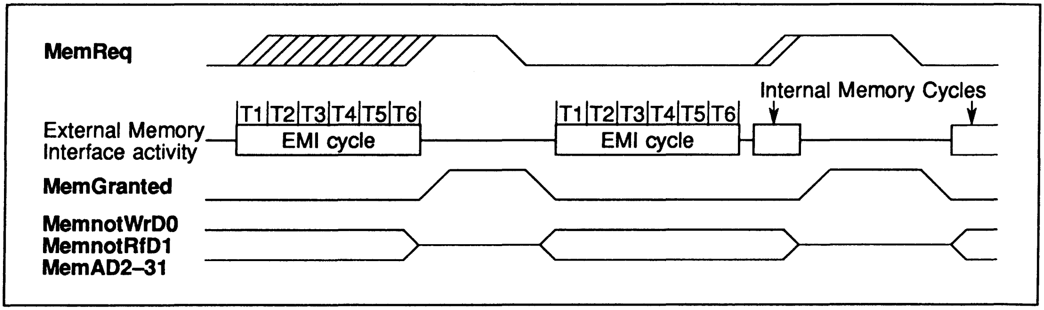


Figure 7.20 IMS T425 operation of MemReq, MemGranted with external, internal memory cycles

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.

EventWaiting is asserted by the transputer when a process executes an input on the event channel; typically with the occam `EVENT ? ANY` instruction. It remains high whilst the transputer is waiting for or servicing **EventReq** and is returned low when **EventAck** goes high. **EventWaiting** allows a process to control external logic; for example, to clock a number of inputs into a memory mapped data latch so that the event request type can be determined. This function is not available on the IMS T414 and IMS T800.

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 121. 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**.

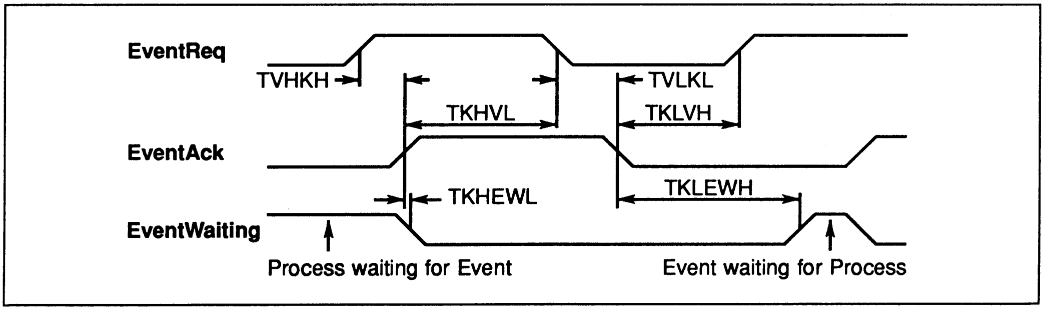


Figure 8.1 IMS T425 event timing

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 T425 links allow 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 T425 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	**Linkn**		**Kbytes/sec**	
Special	**Special**	**Mbits/sec**	**Uni**	**Bi**
0	0	10	910	1250
0	1	5	450	670
1	0	10	910	1250
1	1	20	1740	2350

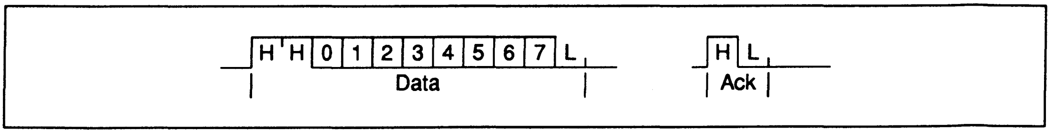


Figure 9.1 IMS T425 link data and acknowledge packets

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.

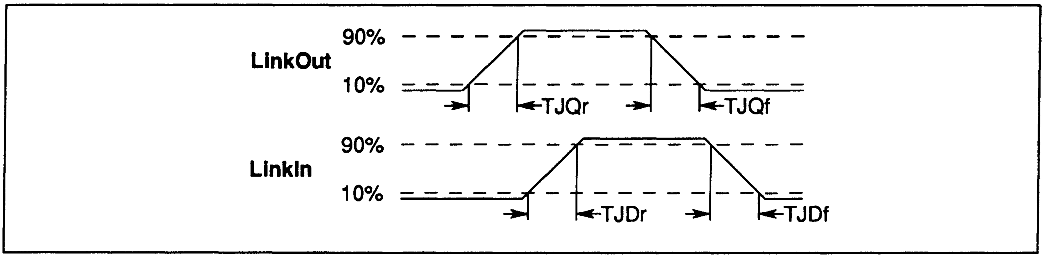


Figure 9.2 IMS T425 link timing

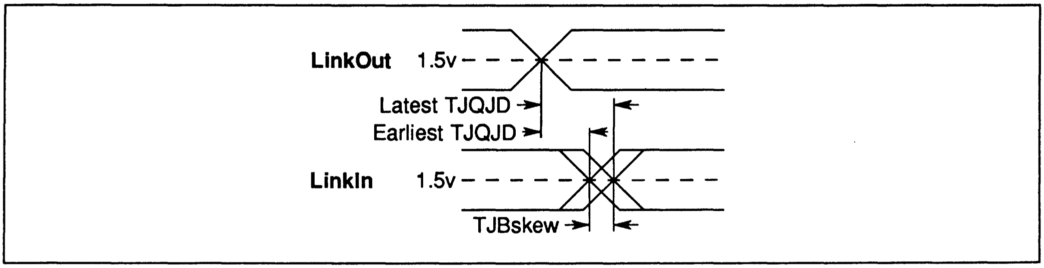


Figure 9.3 IMS T425 buffered link timing

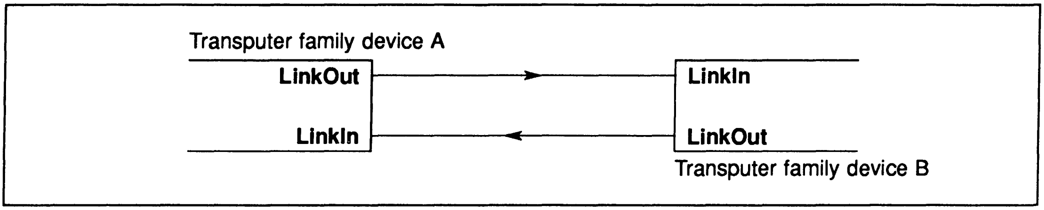


Figure 9.4 IMS T425 Links directly connected

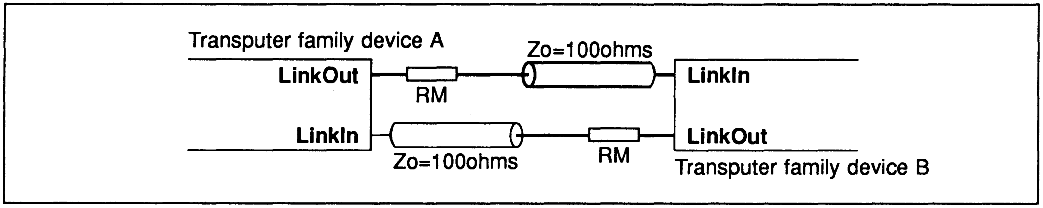


Figure 9.5 IMS T425 Links connected by transmission line

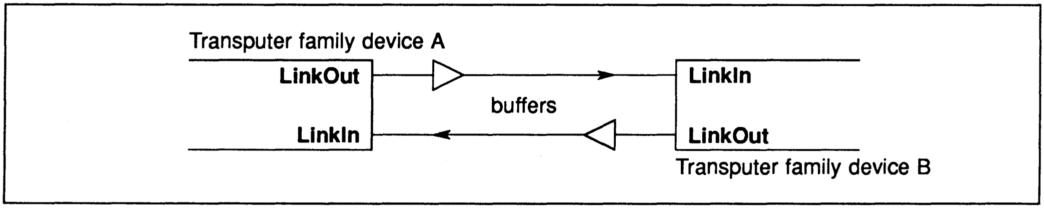


Figure 9.6 IMS T425 Links connected by buffers

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 IMS T425-S	0	70	°C	3
TA	Operating temperature range IMS T425-M	-55	125	°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	1,2,3,7
			65	100	mA	1,2,4,7
			25	90	mA	1,2,3,8
			50	130	mA	1,2,4,8
IOZ	Tristate output current	@ GND<VO<VCC		±10	μA	1,2
PD	Power dissipation			1.0	W	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 T425-S measured at 4.75V<**VCC**<5.25V and 0°C<**TA**<70°C.
Parameters for IMS T425-M measured at 4.75V<**VCC**<5.25V and -55°C<**TA**<125°C.
Input clock frequency = 5MHz.

3 Current sourced from non-link outputs.

4 Current sourced from link outputs.

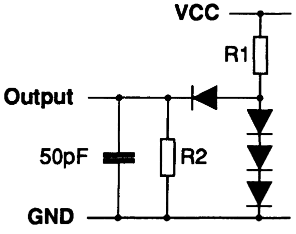
5 Power dissipation varies with output loading and program execution.
Power dissipation for processor operating at 20MHz.

6 This parameter is sampled and not 100% tested.

7 Parameter for IMS T425-S.

8 Parameter for IMS T425-M.

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

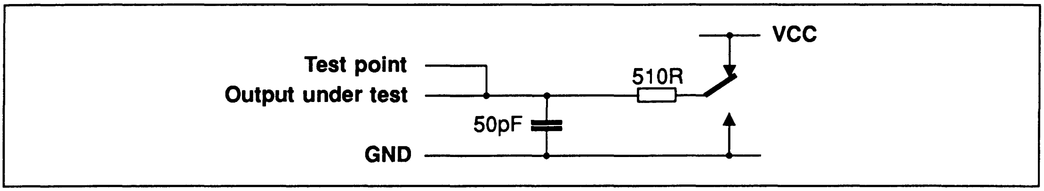


Figure 10.2 Tristate load circuit for AC measurements

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
TS0LaHZ	Address high to tristate	a	a+6	ns	3
TS0LaLZ	Address low to tristate	a	a+6	ns	3

Notes

1 Non-link pins; see section on links.

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

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

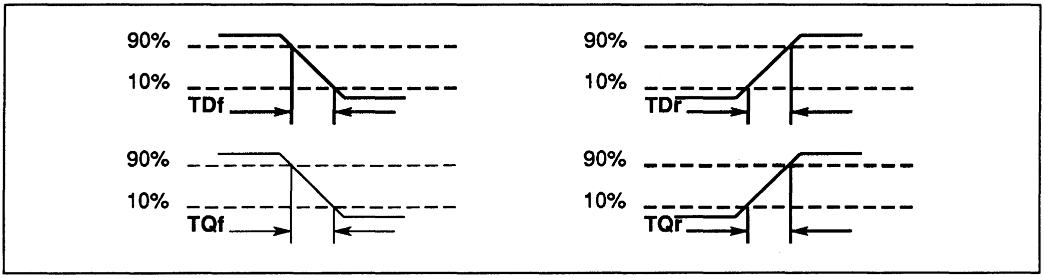


Figure 10.3 IMS T425 input and output edge timing

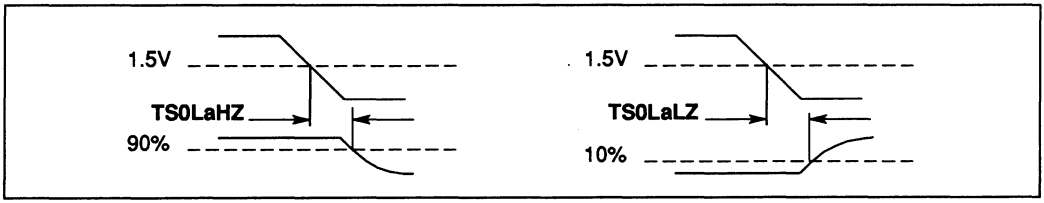


Figure 10.4 IMS T425 tristate timing relative to notMemS0

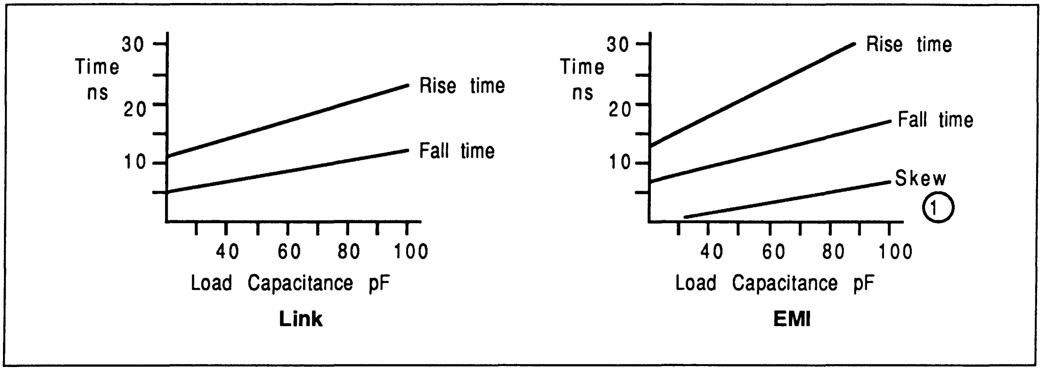


Figure 10.5 Typical rise/fall times

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

Internal power dissipation P_{INT} of transputer and peripheral chips depends on **VCC**, as shown in figure 10.6. 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.

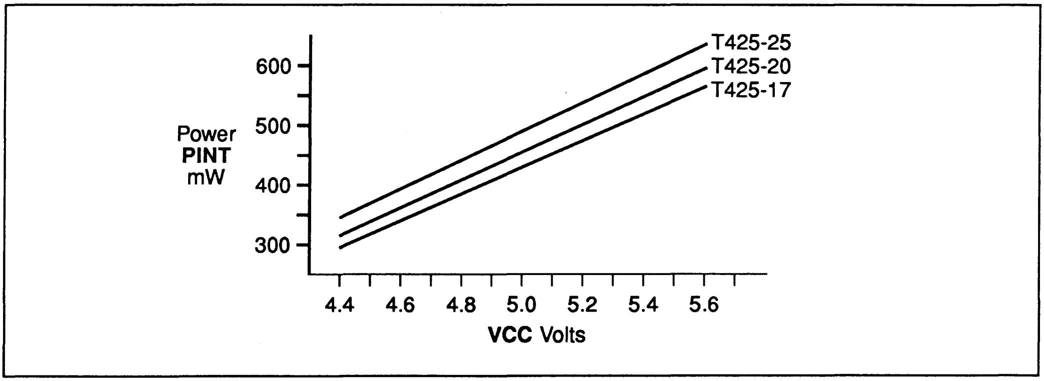


Figure 10.6 IMS T425 internal power dissipation vs VCC

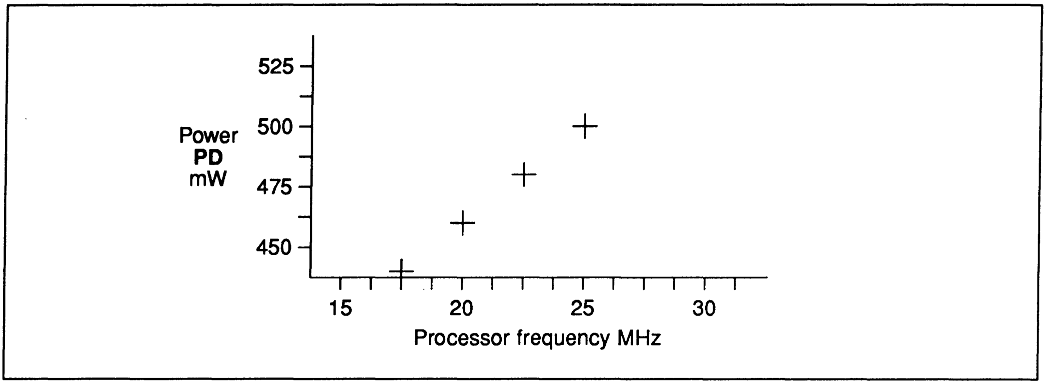


Figure 10.7 IMS T425 typical power dissipation with processor speed

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	39
/	2	40
REM	2	38
>> <<	2	3+**p**
Modulo Arithmetic operators		
PLUS	2	2
MINUS	1	1
TIMES (fast multiply, positive operand)	1	4+**Tbp**
TIMES (fast multiply, negative operand)	1	5+**Tbc**
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**+2**Et**	34+2**Eg**+2**Et**
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 T425 has a fast integer multiplication instruction *product*. For a positive multiplier its execution time is 4+**Tbp** cycles, and for a negative multiplier 5+**Tbc** cycles (table 11.1). The time taken for a multiplication by zero is 3 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. 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		34	8
LONGDIV		36	8
SHIFTRIGHT	(n<32)	4+n	8
	(n>=32)	n-27	
SHIFTLEFT	(n<32)	4+n	8
	(n>=32)	n-27	
NORMALISE	(n<32)	n+6	7
	(n>=32)	n-25	
	(n=64)	4	
ASHIFTRIGHT		**SHIFTRIGHT**+2	5
ASHIFTLEFT		**SHIFTLEFT**+4	5
ROTATERIGHT		**SHIFTRIGHT**	7
ROTATELEFT		**SHIFTLEFT**	7
FRACMUL		**LONGPROD**+4	5

† Assuming local variables.

11.4 IMS T425 floating point operations

Floating point operations for the IMS T425 are provided by a run-time package. This requires approximately 400 bytes of memory for the single length arithmetic operations, and 2500 bytes for the double length arithmetic operations. Table 11.5 summarizes the estimated performance of the package.

Table 11.5 IMS T425 floating point operations performance

		Processor cycles	
		IMS T425	
		Typical	Worst
REAL32	+ -	230	300
	*	200	240
	/	245	280
	< > = >= <= <>	60	60
REAL64	+ -	565	700
	*	760	940
	/	1115	1420
	< > = >= <= <>	60	60

11.4.1 IMS T425 special purpose functions and procedures

The functions and procedures given in tables 11.7 and 11.8 are provided by the development system to give access to the special instructions available on the IMS T425. Table 11.6 shows the key to the table.

Table 11.6 Key to special performance table

Tb	most significant bit set in the word counting from zero
n	number of words per row (consecutive memory locations)
r	number of rows in the two dimensional move
nr	number of bits to reverse

Table 11.7 Special purpose functions performance

Function	Cycles	+ cycles for parameter access †
BITCOUNT	2+Tb	2
CRCBYTE	11	8
CRCWORD	35	8
BITREVNBIT	5+nr	4
BITREVWORD	36	2

† Assuming local variables.

Table 11.8 Special purpose procedures performance

Procedure	Cycles	+ cycles for parameter access †
MOVE2D	8+(2n+23)∗r	8
DRAW2D	8+(2n+23)∗r	8
CLIP2D	8+(2n+23)∗r	8

† Assuming local variables.

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**. For the IMS T425, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

If 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 (**e**-3)/4. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 11.9 **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 11.9 External memory performance

	IMS T425	
	Program off chip	**Data off chip**
Boolean expressions	**e**-2	0
IF	3**en**-8	**en**
Replicated **IF**	(6**e**-4)**n**+7	(5**e**-2)**n**+8
Replicated **SEQ**	(3**e**-3)**n**+2	(4**e**-2)**n**
PAR	(3**e**-1)**n**+8	3**en**+4
Replicated **PAR**	(10**e**-8)**n**+8	16**en**-12
ALT	(2**e**-4)**n**+6**e**	(2**e**-2)**n**+10**e**-8
Array assignment and communication in one transputer	0	max (2**e**, **e**(**b**/2))

For the IMS T425 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 11.10 IMS T425 external memory performance

	Program	**e=2**	**e=3**	**e=4**	**e=5**	**On chip**
Program off chip	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
Data off chip	1	1.5	1.8	2.1	2.3	1
	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

11.6 Interrupt latency

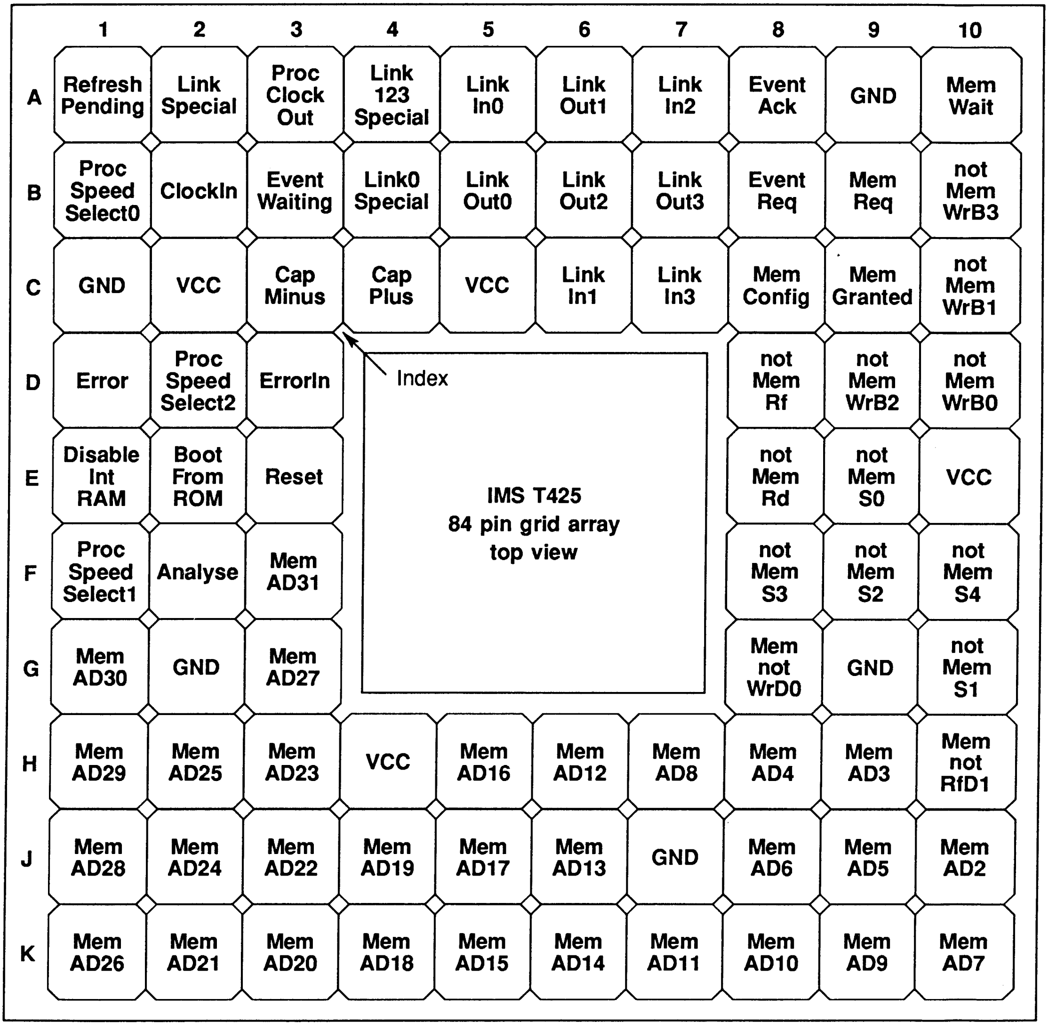
If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.11. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

Table 11.11 Interrupt latency

	Typical		Maximum	
	TPCLPCL	Tm	TPCLPCL	Tm
IMS T425	19	38	53	116

12 Package specifications

12.1 84 pin grid array package



	1	2	3	4	5	6	7	8	9	10
A	Refresh Pending	Link Special	Proc Clock Out	Link 123 Special	Link In0	Link Out1	Link In2	Event Ack	GND	Mem Wait
B	Proc Speed Select0	ClockIn	Event Waiting	Link0 Special	Link Out0	Link Out2	Link Out3	Event Req	Mem Req	not Mem WrB3
C	GND	VCC	Cap Minus	Cap Plus	VCC	Link In1	Link In3	Mem Config	Mem Granted	not Mem WrB1
D	Error	Proc Speed Select2	ErrorIn					not Mem Rf	not Mem WrB2	not Mem WrB0
E	Disable Int RAM	Boot From ROM	Reset					not Mem Rd	not Mem S0	VCC
F	Proc Speed Select1	Analyse	Mem AD31					not Mem S3	not Mem S2	not Mem S4
G	Mem AD30	GND	Mem AD27					Mem not WrD0	GND	not Mem S1
H	Mem AD29	Mem AD25	Mem AD23	VCC	Mem AD16	Mem AD12	Mem AD8	Mem AD4	Mem AD3	Mem not 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

Figure 12.1 IMS T425 84 pin grid array package pinout

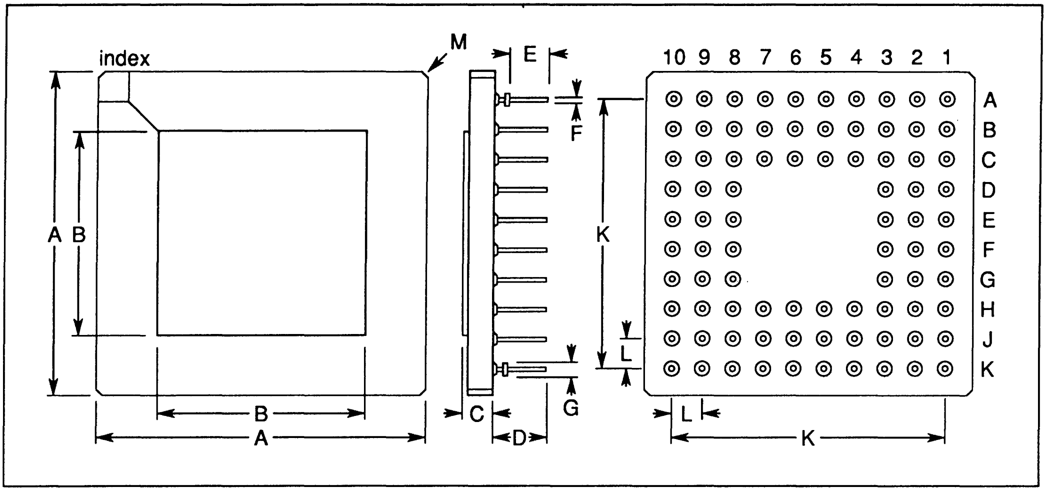


Figure 12.2 84 pin grid array package dimensions

Table 12.1 84 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	±0.005	
C	2.456	±0.278	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	±0.025	0.018	±0.001	Pin diameter
G	1.143	±0.127	0.045	±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 7.2 grams

Table 12.2 84 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 84 PLCC J-bend package

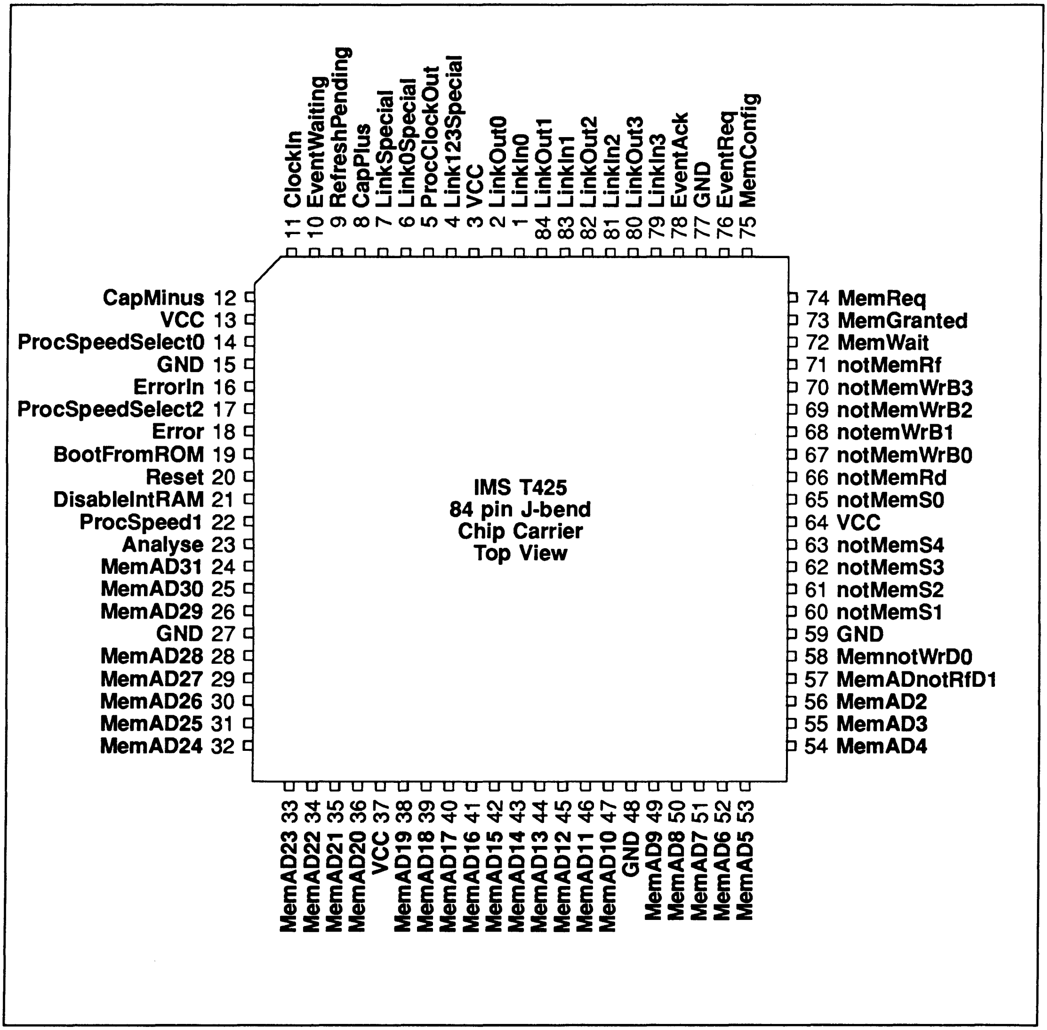


Figure 12.3 IMS T425 84 PLCC J-bend package pinout

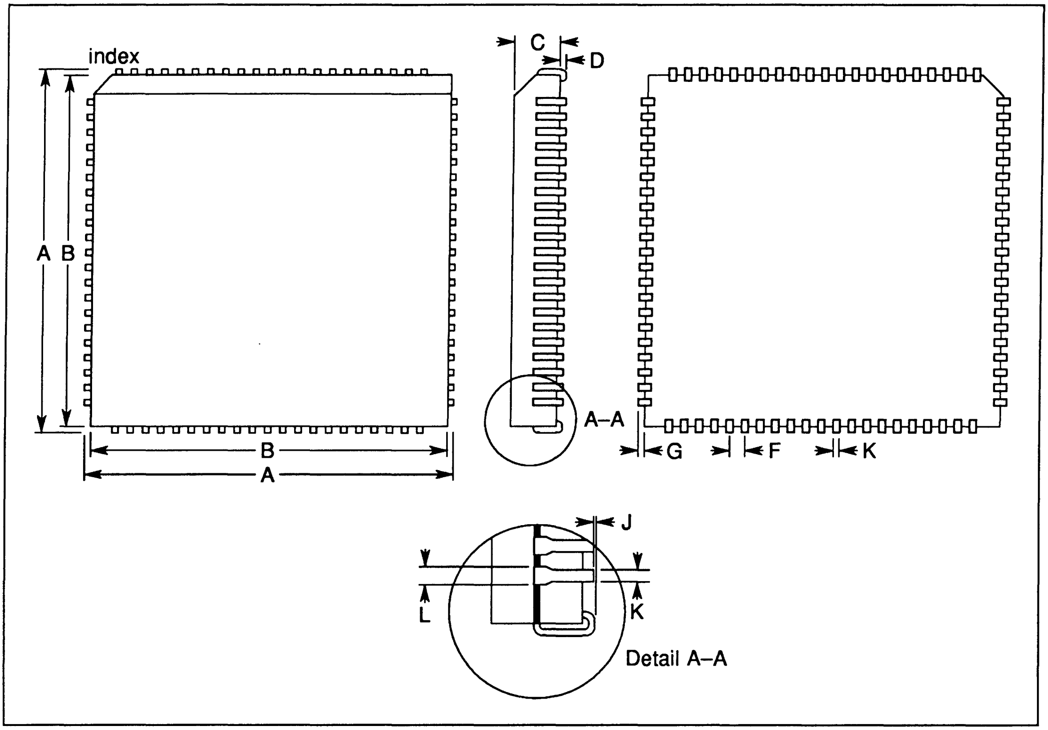


Figure 12.4 84 PLCC J-bend package dimensions

Table 12.3 84 PLCC J-bend package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	30.226	±0.127	1.190	±0.005	
B	29.312	±0.127	1.154	±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 7.0 grams

Table 12.4 84 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	

13 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 13.1 IMS T425 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T425-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T425-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T425-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T425-J17S	17.5 MHz	57 ns	3.5	Plastic PLCC J-Bend
IMS T425-J20S	20.0 MHz	50 ns	4.0	Plastic PLCC J-Bend
IMS T425-G17M	17.5 MHz	57 ns	3.5	Ceramic Pin Grid MIL Spec
IMS T425-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid MIL Spec

The timing parameters in this datasheet are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

IMS T414
transputer

The IMS T425 is recommended for new designs

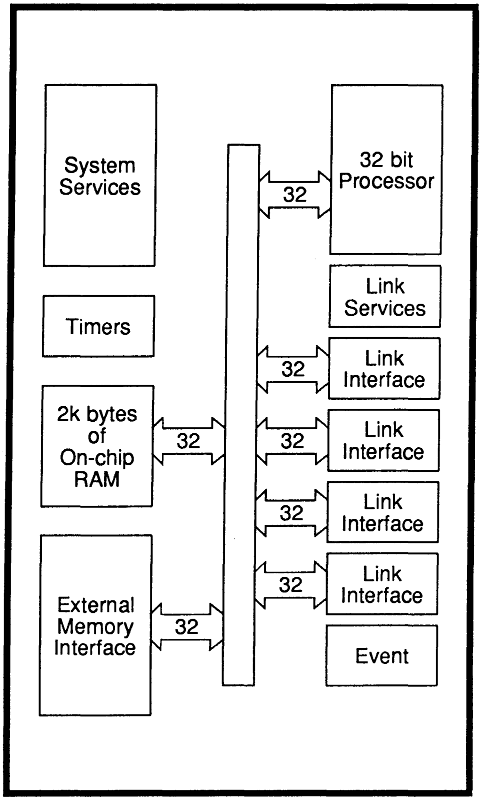
Engineering Data

FEATURES

32 bit architecture
50 ns internal cycle time
20 MIPS (peak) instruction rate
Pin compatible with IMS T800 and IMS T425
2 Kbytes on-chip static RAM
80 Mbytes/sec sustained data rate to internal memory
4 Gbytes directly addressable external memory
26 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

APPLICATIONS

High speed multi processor systems
Real time processing
Microprocessor applications
Workstations and workstation clusters
Image processing
Graphics processing
Accelerator processors
Distributed databases
Supercomputers
System simulation
Digital signal processing
Telecommunications
Robotics
Fault tolerant systems
Medical instrumentation
Pattern recognition
Artificial intelligence



1 Introduction

The IMS T414 transputer is a 32 bit CMOS microcomputer with 2 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 IMS T414 provides high performance arithmetic and microcode support for floating point operations. A device running at 20 MHz achieves an instruction throughput of 10 MIPS.

For convenience of description, the IMS T414 operation is split into the basic blocks shown in figure 1.1.

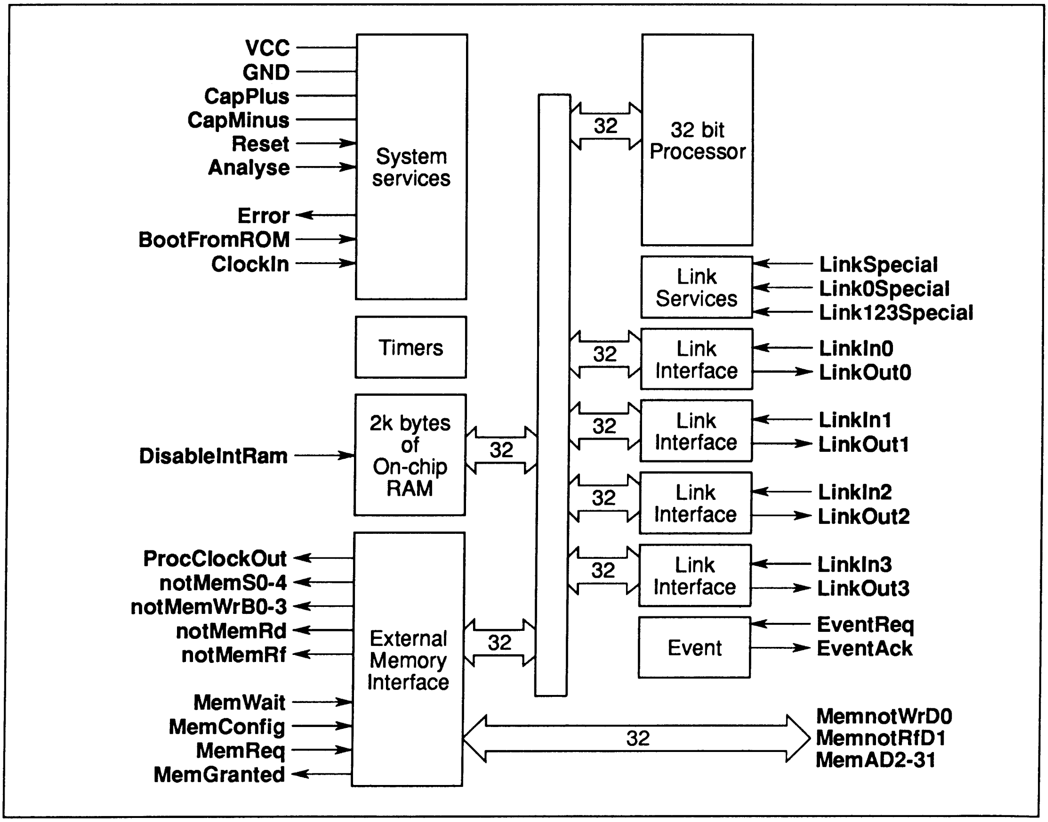


Figure 1.1 IMS T414 block diagram

The IMS T414 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 150 nanoseconds (26.6 Mbytes/sec) for a 20 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 bootstrap control, together with facilities for error analysis.

The INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T414 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec.

The IMS T414 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 T414. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of occam.

2 Pin designations

Table 2.1 IMS T414 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	Bootstrap from external ROM or from link
DisableIntRAM	in	Disable internal RAM
HoldToGND		Must be connected to **GND**
DoNotWire		Must not be wired

Table 2.2 IMS T414 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

Table 2.3 IMS T414 event

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

Table 2.4 IMS T414 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 238.

3 Processor

The 32 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 4 Gbytes 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.

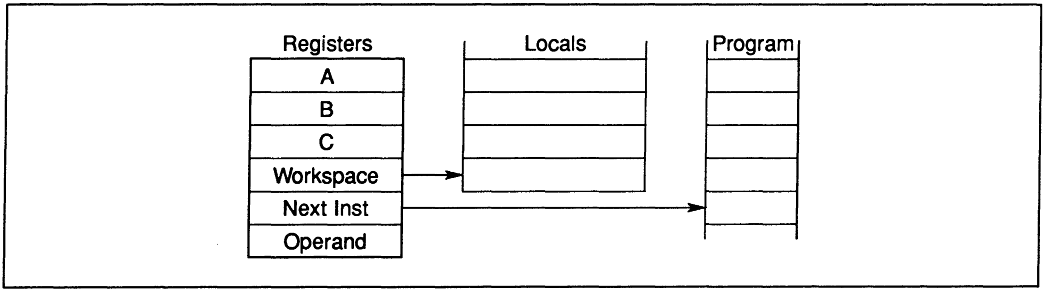


Figure 3.1 Registers

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.

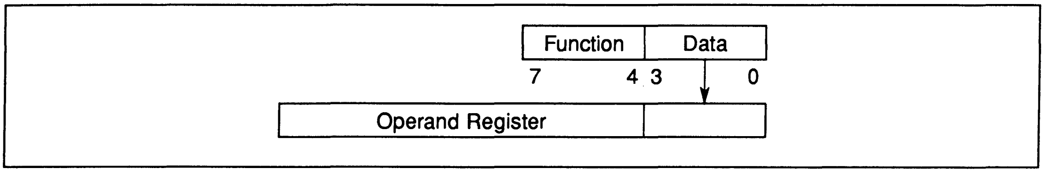


Figure 3.2 Instruction format

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, 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 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 four 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 187).

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 187). 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.

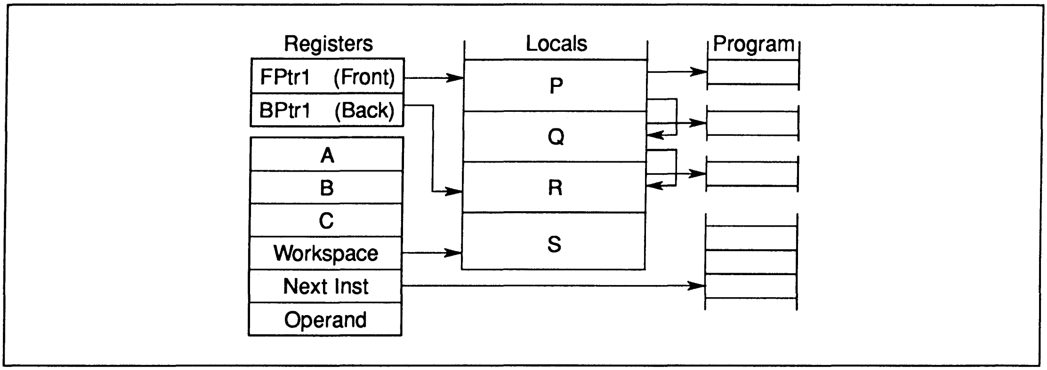


Figure 3.3 Linked process list

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 190). 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 190). 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 T414 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 190).

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 58 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 32 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 4295 seconds. 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 76 hours.

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.

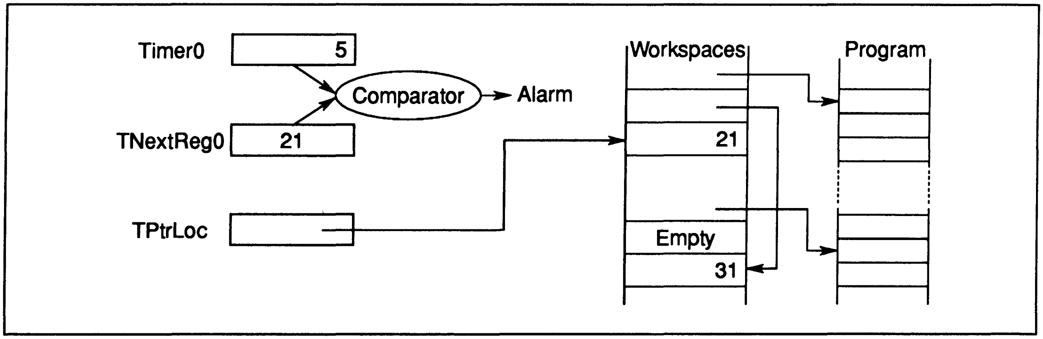


Figure 3.4 Timer registers

4 Instruction set summary

The Function Codes table 4.7. gives the basic function code set (page 184). 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* #FFFFFFE1)		
is coded as			
nfix	#1	#6	#61
ldc	#1	#4	#41

Tables 4.8 to 4.18 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	190
E	The instruction will affect the *Error* flag	190, 200

4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 186). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 199).

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 200) directly.

Table 4.6 Error setting instructions

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

Table 4.7 IMS T414 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 T414 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	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	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 T414 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	33	long multiply	
1A	21FA	ldiv	35	long divide	E
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)	

Table 4.10 IMS T414 floating point support operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
73	27F3	cflerr	3	check floating point error	E
63	26F3	unpacksn	15	unpack single length fp number	
6D	26FD	roundsn	12/15	round single length fp number	
6C	26FC	postnormsn	5/30	post-normalise correction of single length fp number	
71	27F1	ldinf	1	load single length infinity	

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

Table 4.11 IMS T414 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 4.12 IMS T414 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	5	word count	
01	F1	lb	5	load byte	
3B	23FB	sb	4	store byte	
4A	24FA	move	2w+8	move message	

Table 4.13 IMS T414 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 4.14 IMS T414 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 4.15 IMS T414 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 4.16 IMS T414 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 4.17 IMS T414 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 4.18 IMS T414 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.

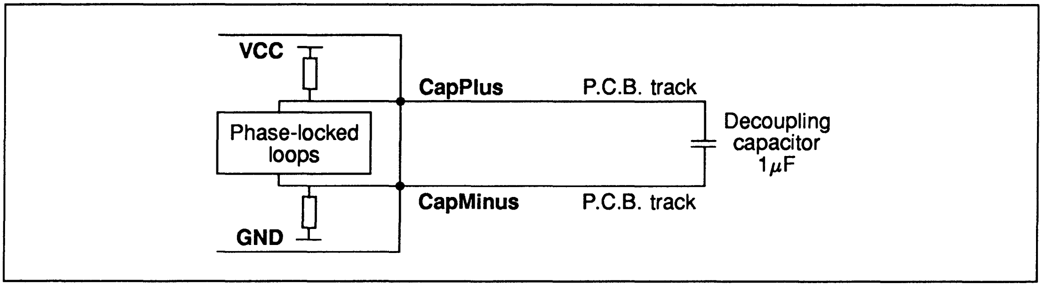


Figure 5.1 Recommended PLL decoupling

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).

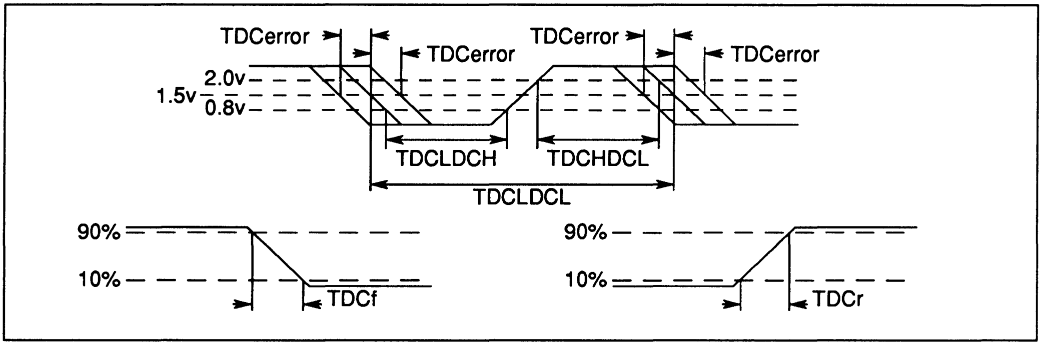


Figure 5.2 ClockIn timing

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, 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 bootstrap (page 222).

After the end of **Reset** there will be a delay of 144 periods of **ClockIn** (figure 5.3). Following this, the **MemWrD0**, **MemRfD1** and **MemAD2-31** pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 212). 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 (page 213), 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.

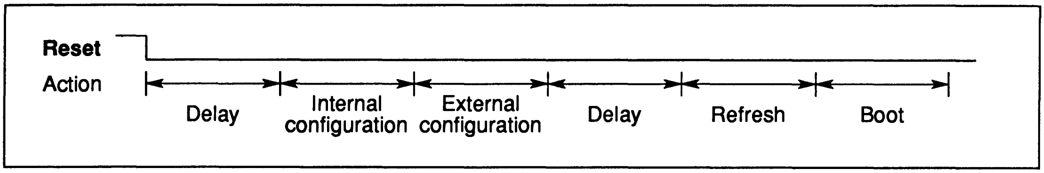


Figure 5.3 IMS T414 post-reset sequence

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 #7FFFFFFE. 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 201).

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.

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.

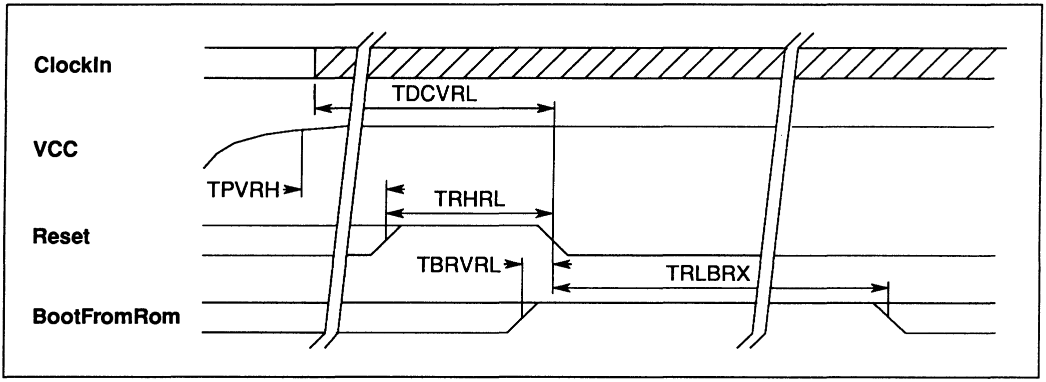


Figure 5.4 Transputer reset timing with Analyse low

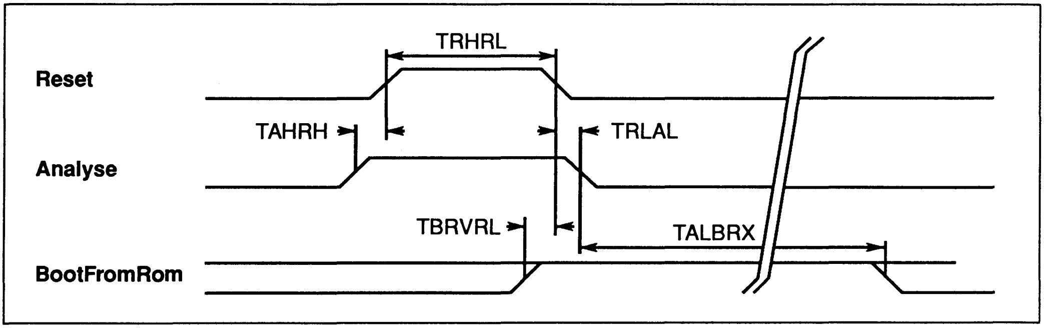


Figure 5.5 Transputer reset and analyse timing

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 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 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 190). 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. 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 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 190). 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 199).

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. Memory refresh will continue to take place.

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*.

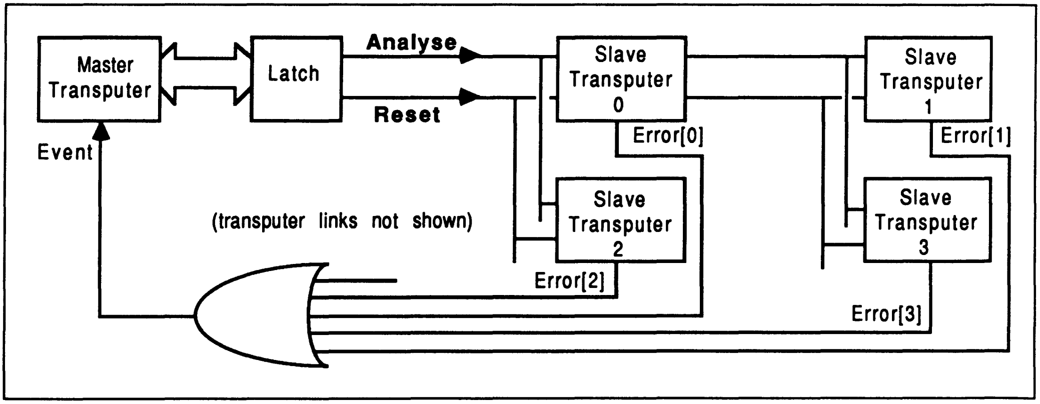


Figure 5.6 Error handling in a multi-transputer system

6 Memory

The IMS T414 has 2 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 203). The transputer can also access 4 Gbytes 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 T414 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 #800007FF. User memory begins at #80000048; 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 #80000800 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.

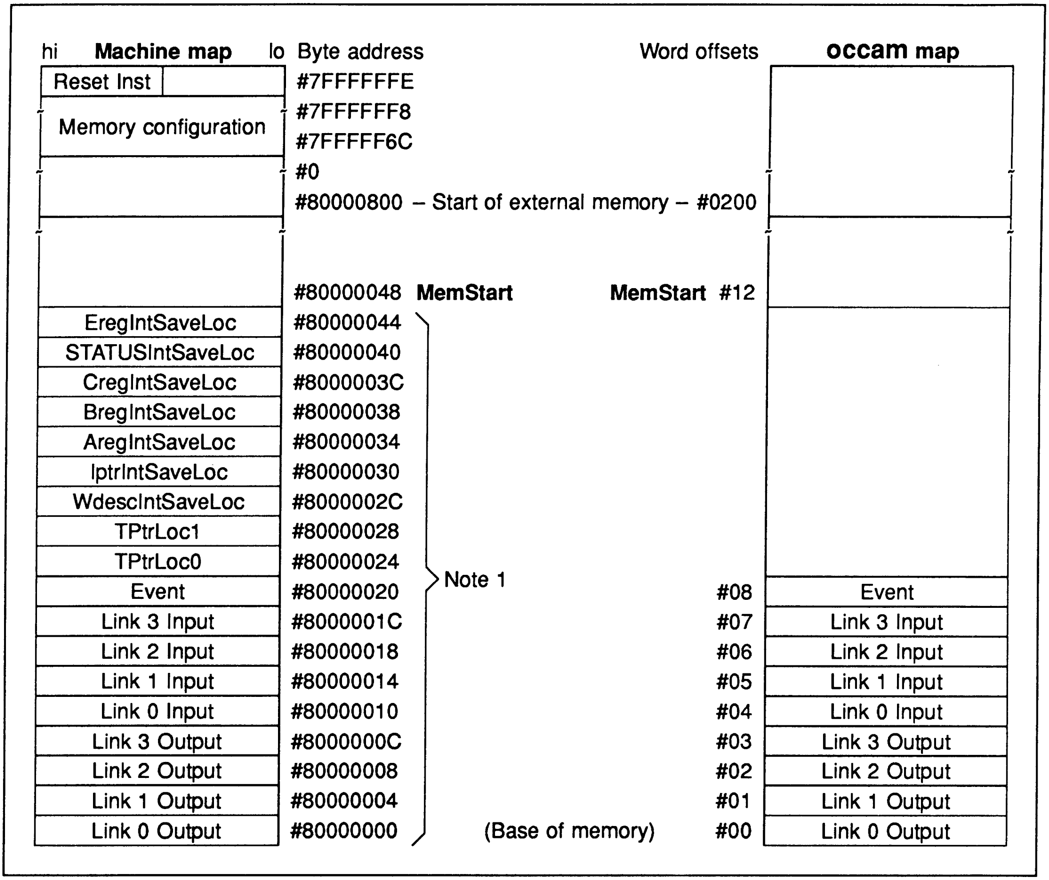


Figure 6.1 IMS T414 memory map

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 199). 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 (page 212). If none are suitable the user can configure the interface to specific requirements, as shown in page 213.

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).

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 7.4.

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 (figure 7.11).

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
Tm	ProcClockOut half cycle	**b**-0.5	**b**	**b**+0.5	ns	2
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.

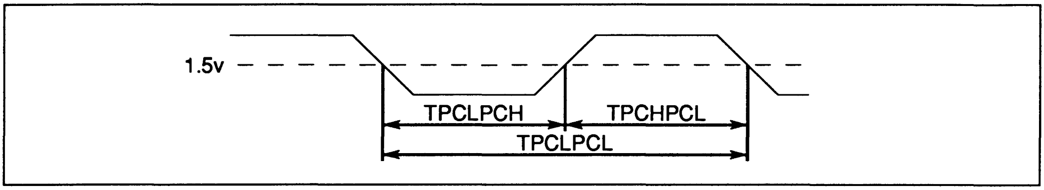


Figure 7.1 IMS T414 ProcClockOut timing

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 (page 199).

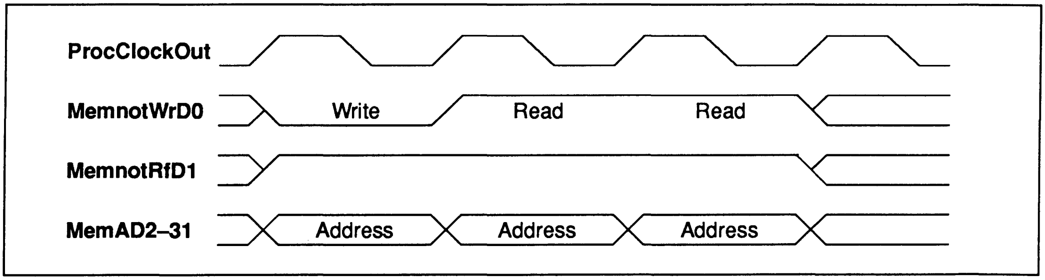


Figure 7.2 IMS T414 bus activity for internal memory cycle

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 transputer 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 transputer 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** (page 205) 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**.

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 (page 220). 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 low. Figure 7.5 shows the effect of **Wait** on strobes in more detail; each division on the scale is one period **Tm**.

Table 7.2 Read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

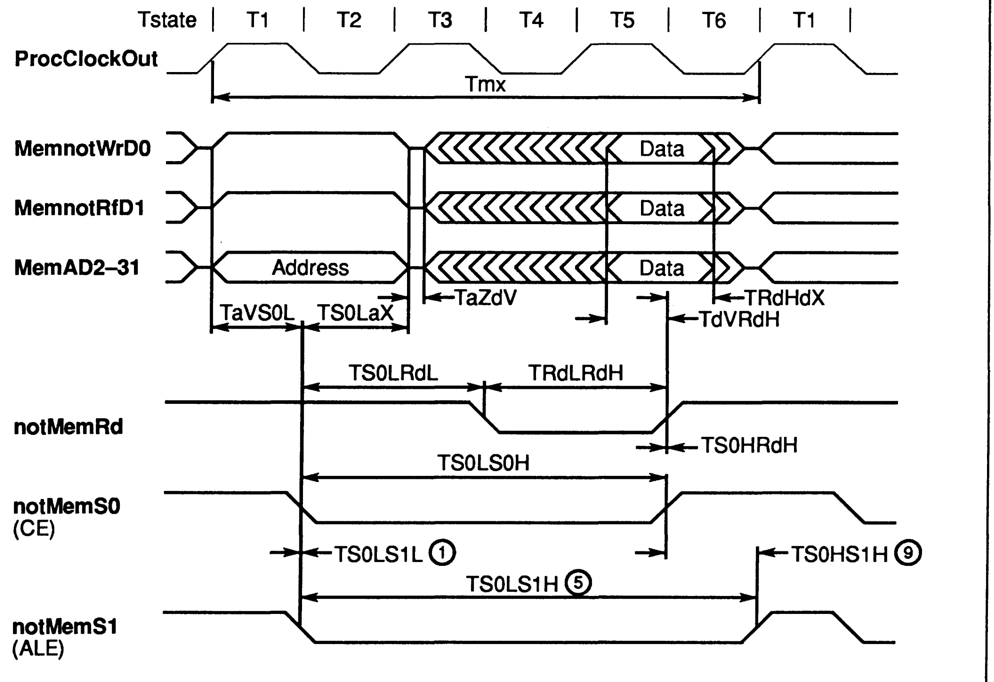


Figure 7.3 IMS T414 external read cycle: static memory

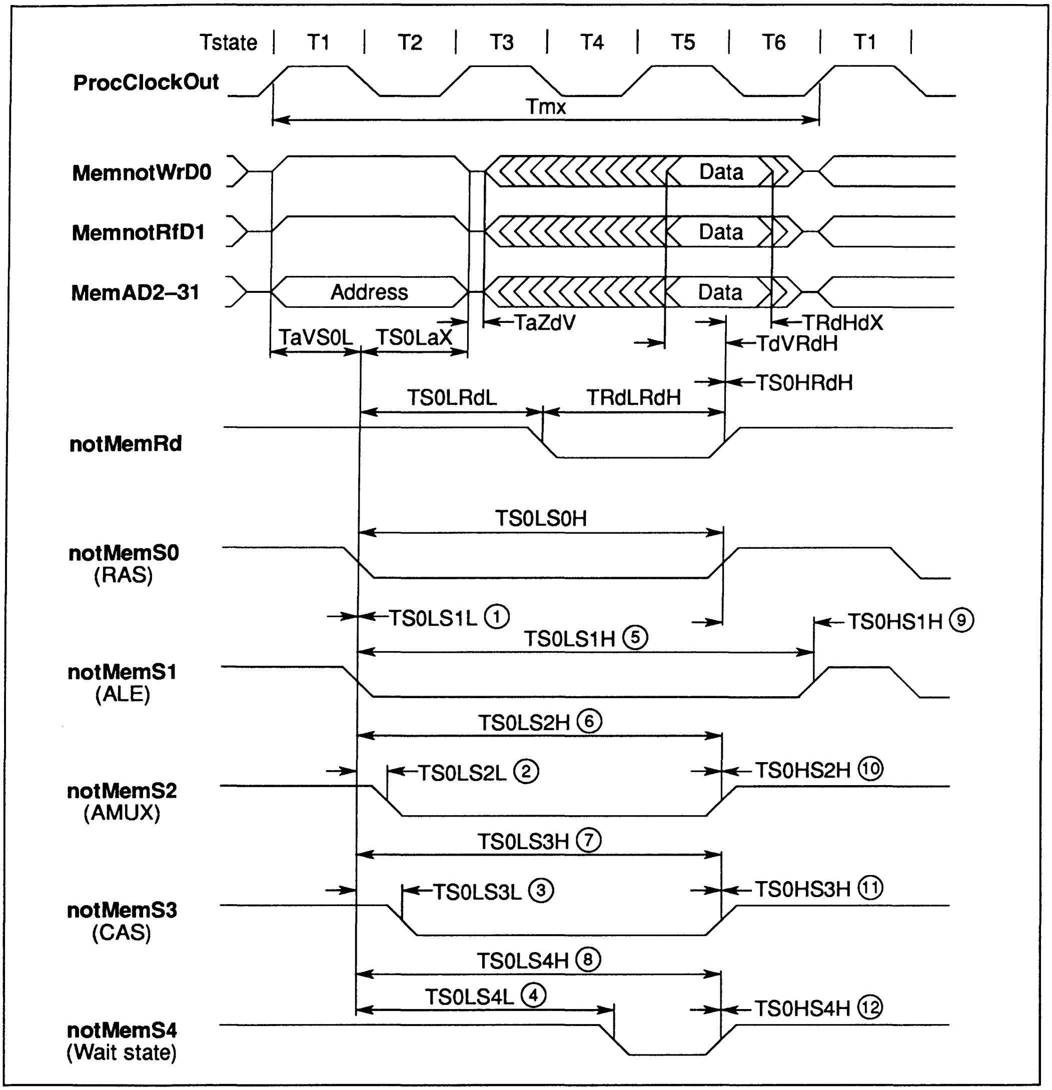


Figure 7.4 IMS T414 external read cycle: dynamic memory

Table 7.3 IMS T414 strobe timing

SYMBOL	ⓝ	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

Figure 7.5 IMS T414 effect of wait states on strobes

Table 7.4 Strobe S0 to ProcClockOut skew

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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	

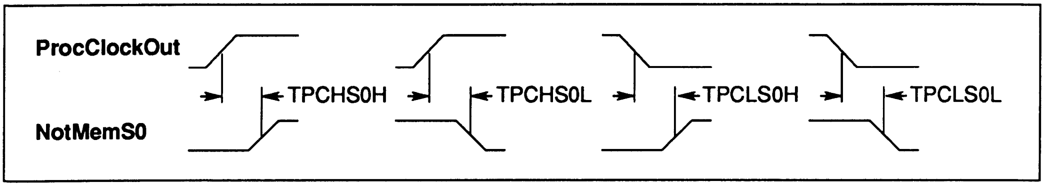


Figure 7.6 IMS T414 skew of notMemS0 to ProcClockOut

7.9 notMemWrB0-3

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

The transputer 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.

Table 7.5 Write

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
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.

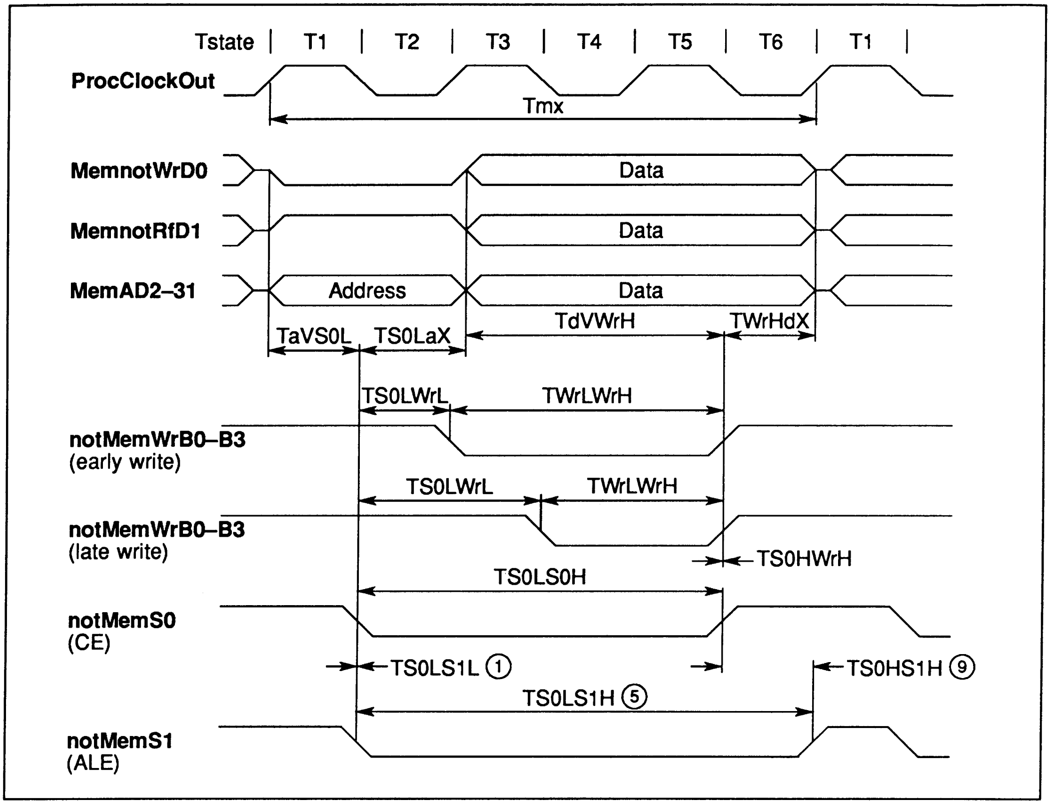


Figure 7.7 IMS T414 external write cycle

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.

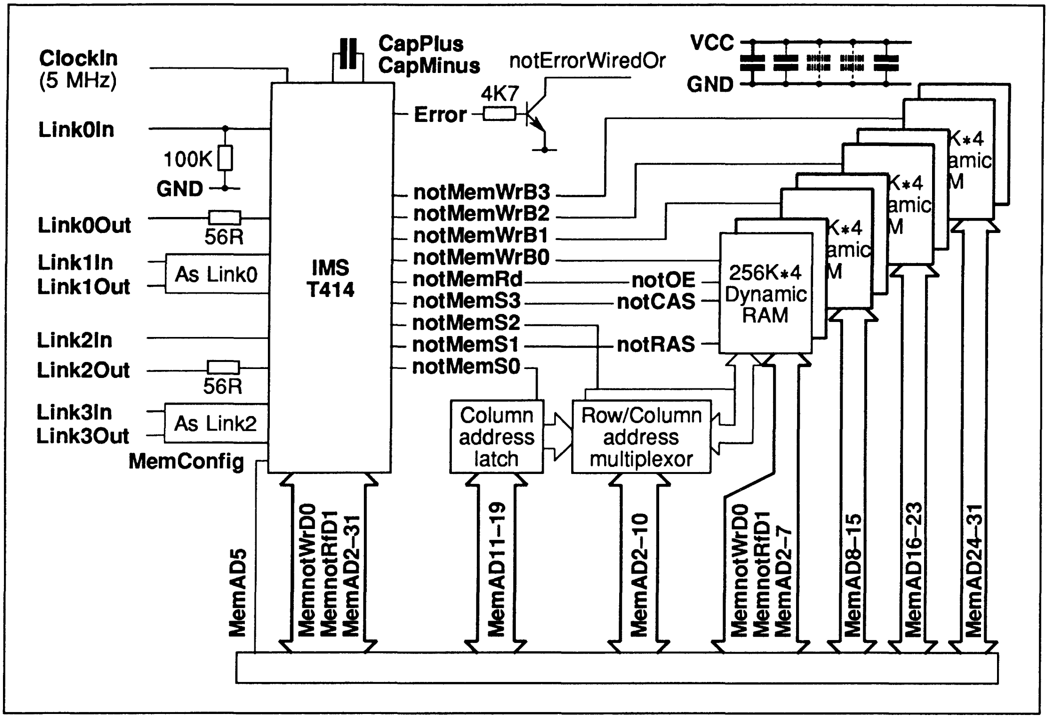


Figure 7.8 IMS T414 dynamic RAM application

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, all others remain at the default configuration.

Table 7.6 IMS T414 internal configuration coding

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

† **Provided for static RAM only.**

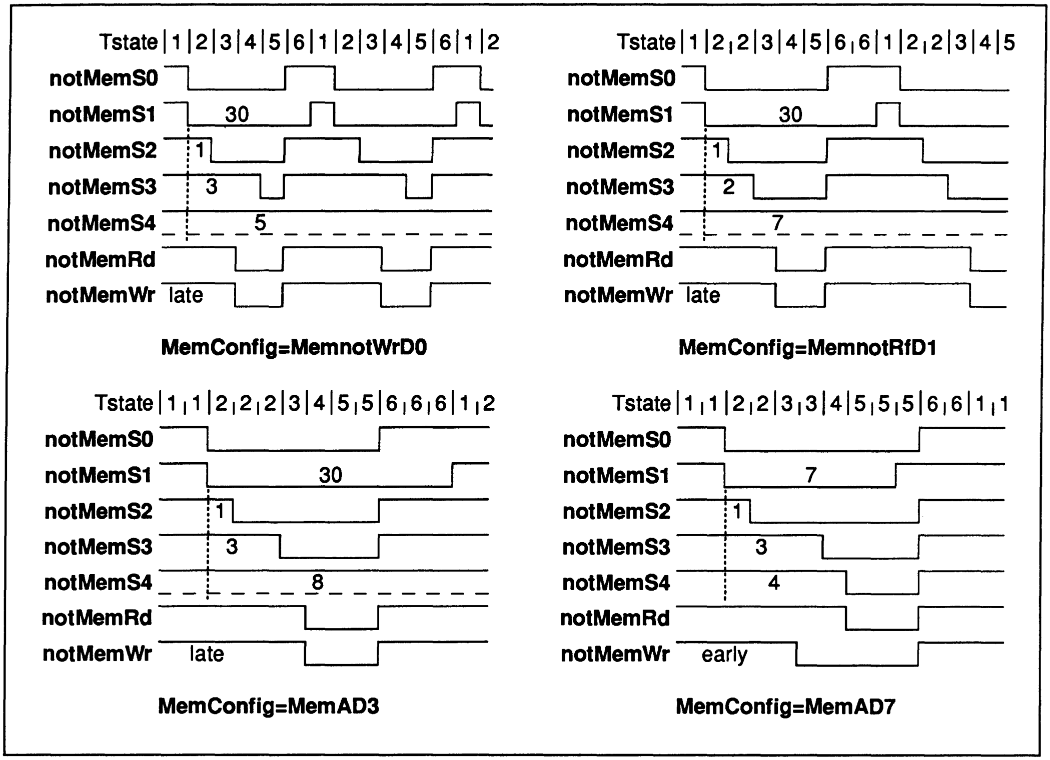


Figure 7.9 IMS T414 internal configuration

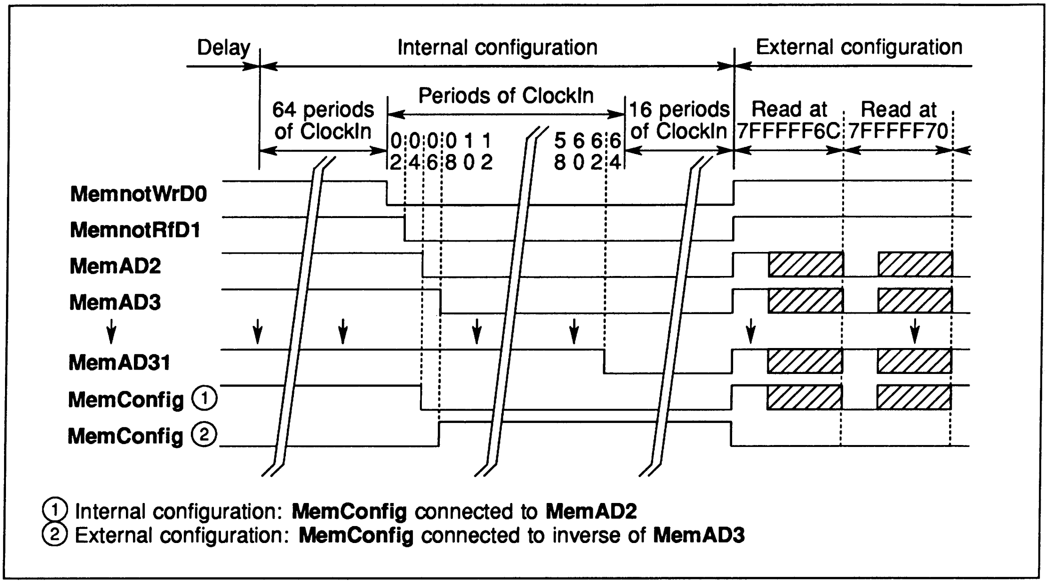
7.10.2 External configuration

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 table 7.7, 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; late write.

The external memory configuration table 7.7 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.



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

Figure 7.10 IMS T414 internal configuration scan

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 page 205.

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** (page 203).

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. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

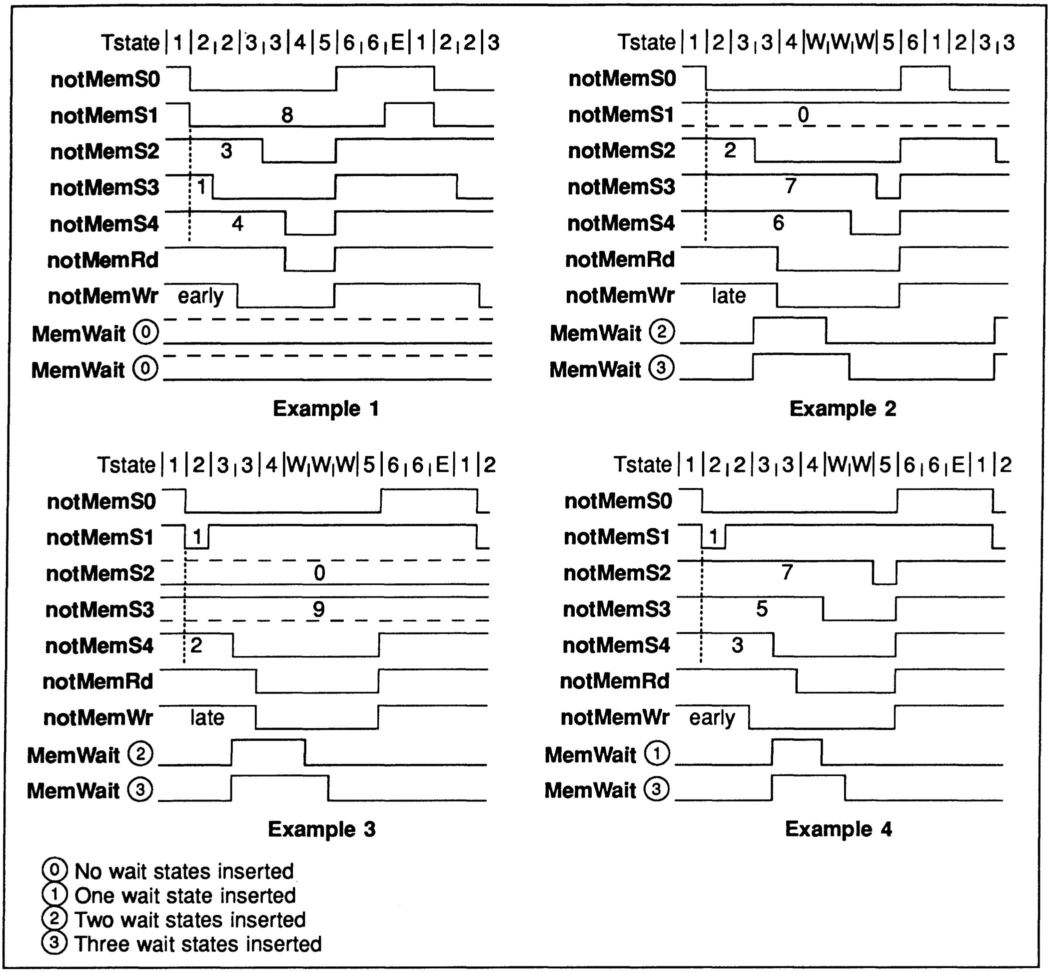
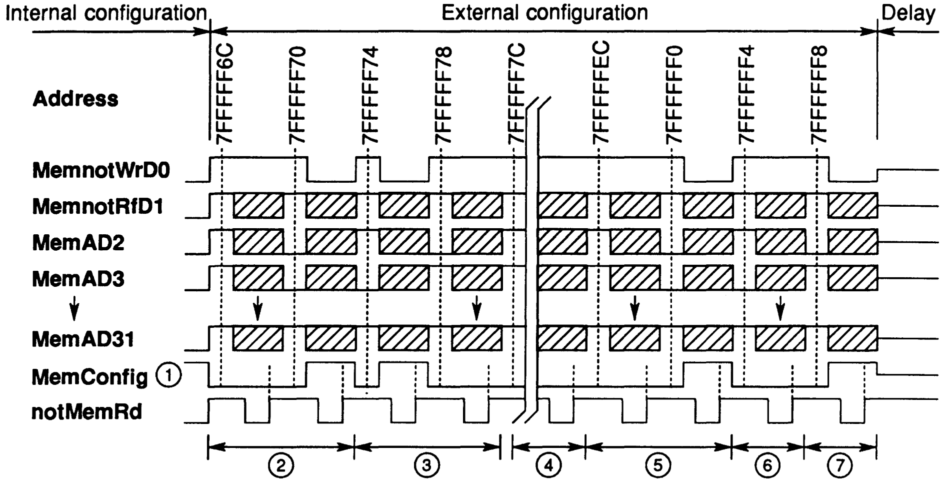


Figure 7.11 IMS T414 external configuration



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

Figure 7.12 IMS T414 external configuration scan

Table 7.7 IMS T414 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

Table 7.8 IMS T414 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 5 MHz:

$$\text{Interval} = 18 * 200 = 3600 \text{ ns}$$

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

Table 7.9 Memory configuration

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMCVRdH	Memory configuration data setup	30			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**.

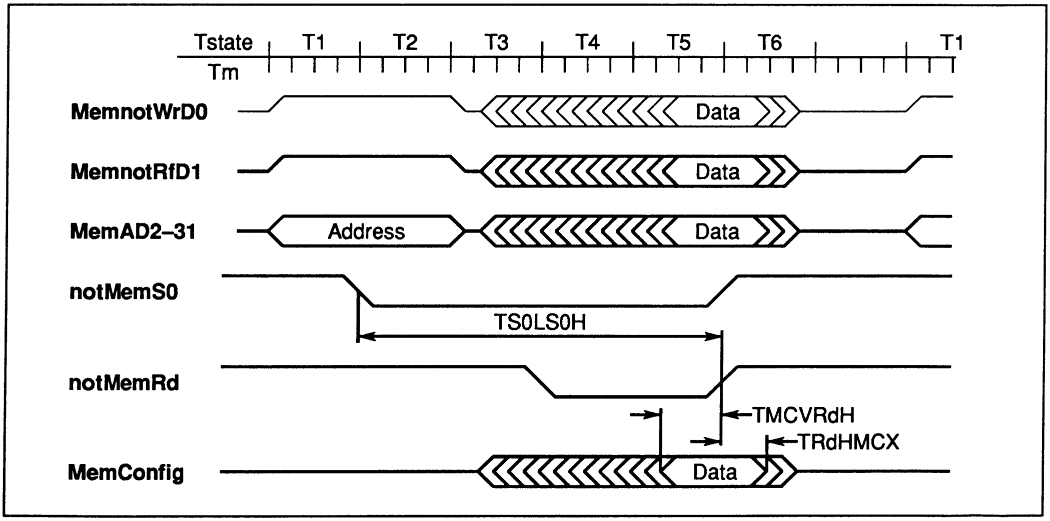


Figure 7.13 IMS T414 external configuration read cycle timing

7.11 notMemRf

The IMS T414 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. **MemWait** operates normally during refresh cycles.

Table 7.10 Memory refresh

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

Notes

1 **a** is total **Tmx+Tm**.

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

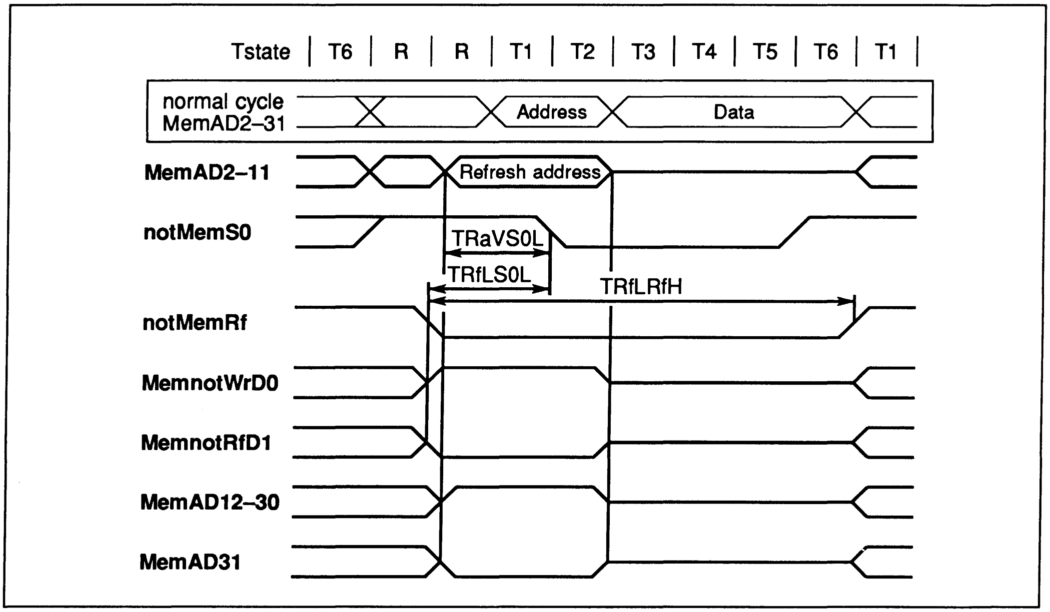


Figure 7.14 IMS T414 refresh cycle timing

7.12 MemWait

Taking **MemWait** high with the timing shown will extend the duration of **T4**. **MemWait** is sampled relative to the falling edge of **ProcClockOut** during a **T3** period, 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. **MemWait** operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

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.

Table 7.11 Memory wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHWtH	Wait setup	0.5**Tm**+3			ns	1,2
TPCHWtL	Wait hold	0.5**Tm**+3			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	2**Tm**			ns	

Notes

1 **ProcClockOut** load should not exceed 50pf.

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

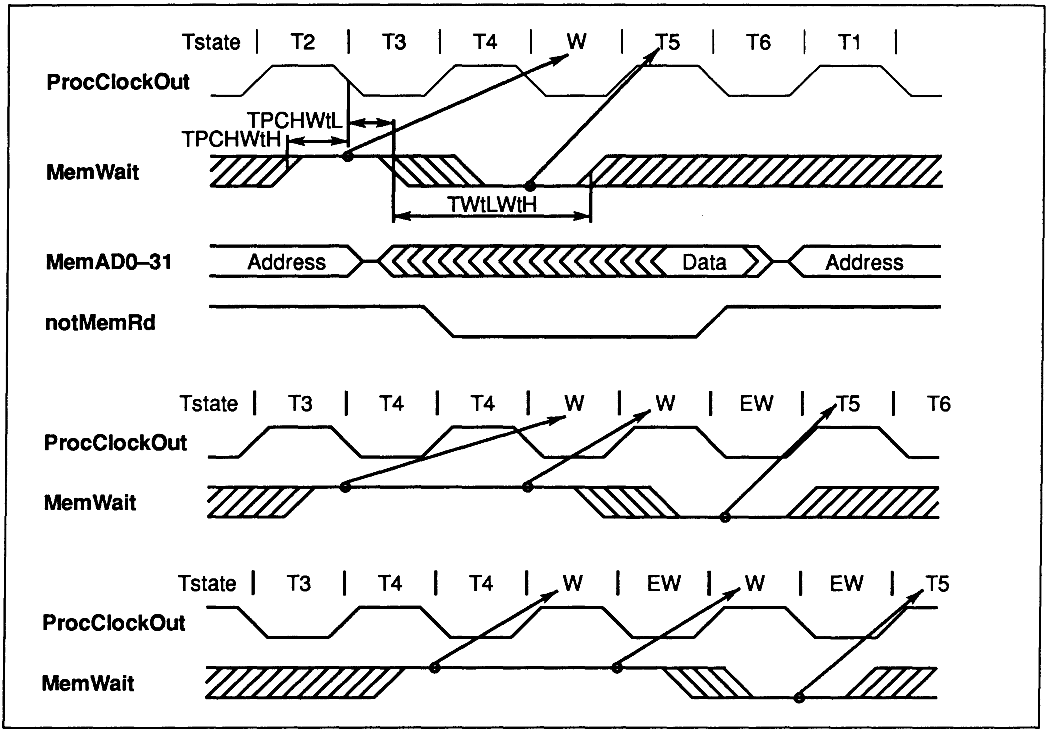


Figure 7.15 IMS T414 memory wait timing

7.13 MemReq, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer 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 7.8), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

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 7.12 Memory request

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4		7	Tm	1
TMRLMGL	Memory request end response time	2		5	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**).

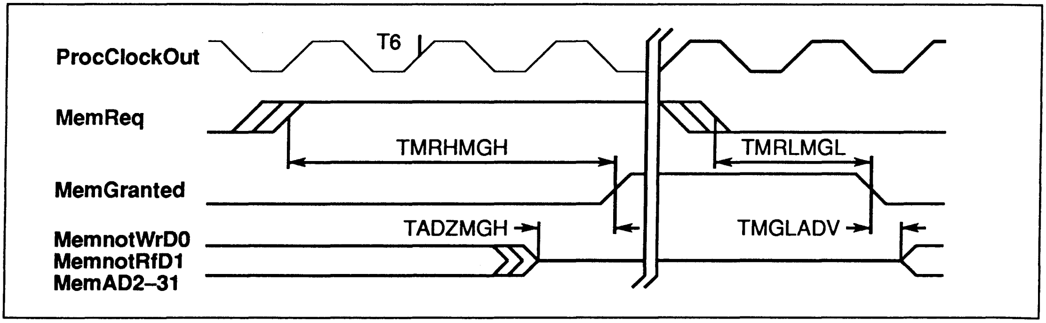
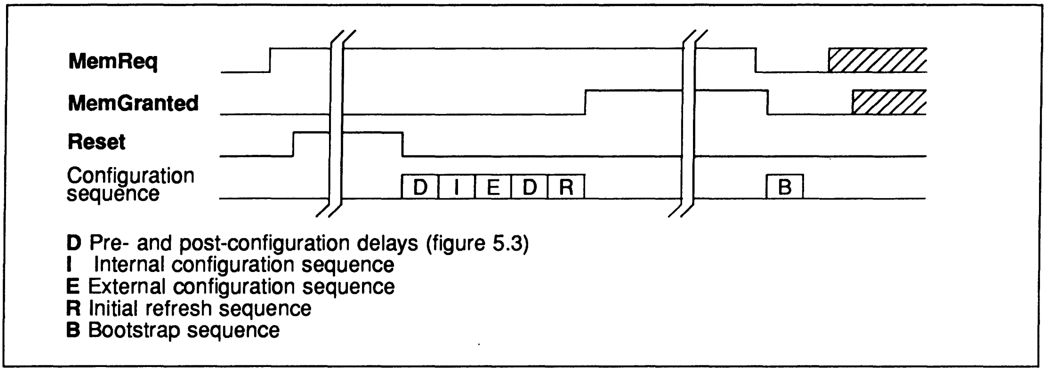


Figure 7.16 IMS T414 memory request timing



D Pre- and post-configuration delays (figure 5.3)
I Internal configuration sequence
E External configuration sequence
R Initial refresh sequence
B Bootstrap sequence

Figure 7.17 IMS T414 DMA sequence at reset

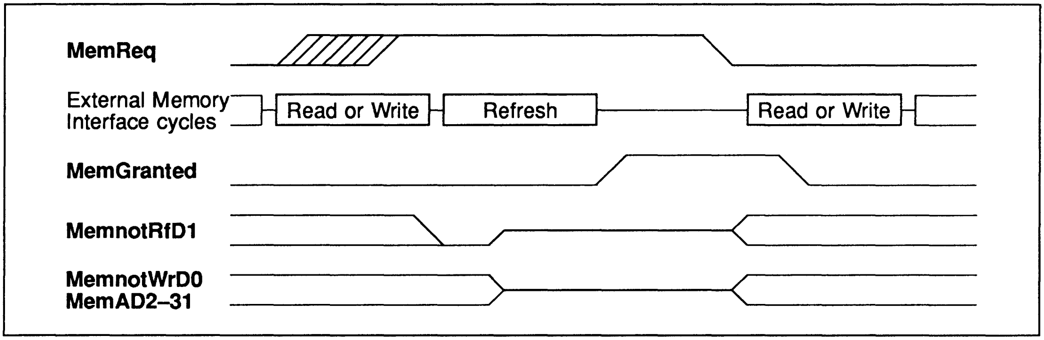


Figure 7.18 IMS T414 operation of MemReq, MemGranted with external, refresh memory cycles

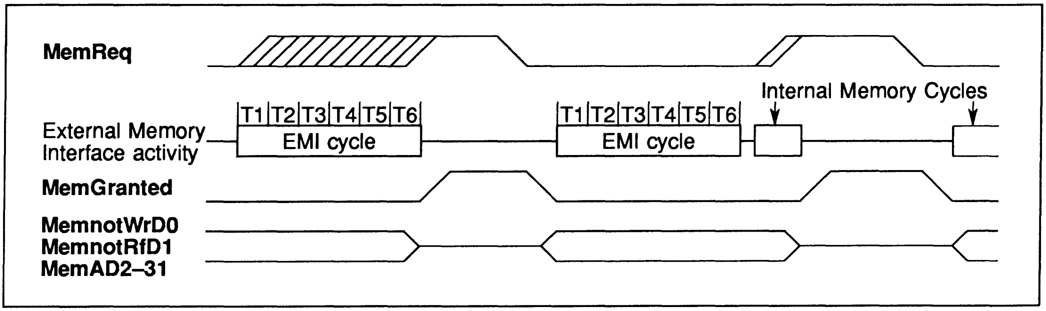


Figure 7.19 IMS T414 operation of MemReq, MemGranted with external, internal memory cycles

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 latency is as described on page 187. 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**.

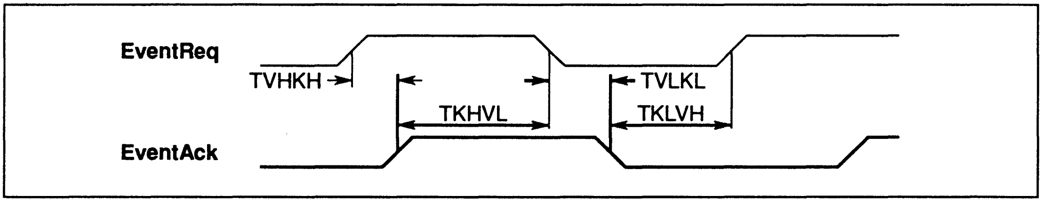


Figure 8.1 IMS T414 event timing

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 T414 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

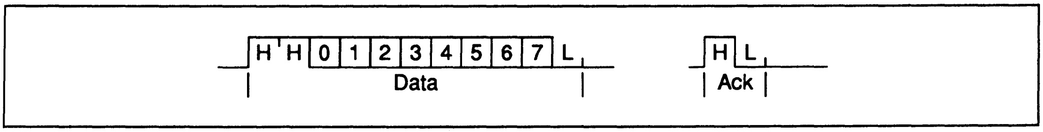


Figure 9.1 IMS T414 link data and acknowledge packets

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.

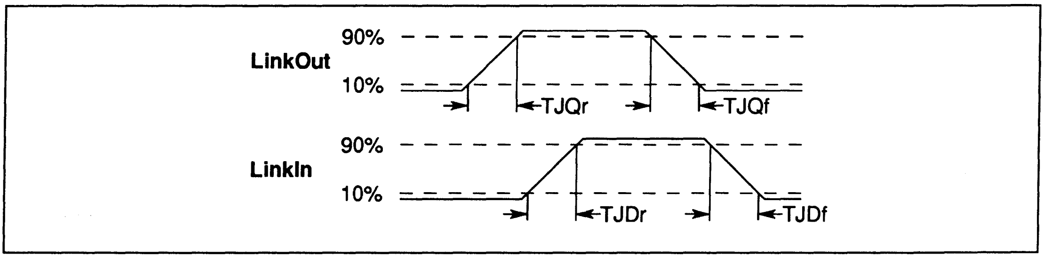


Figure 9.2 IMS T414 link timing

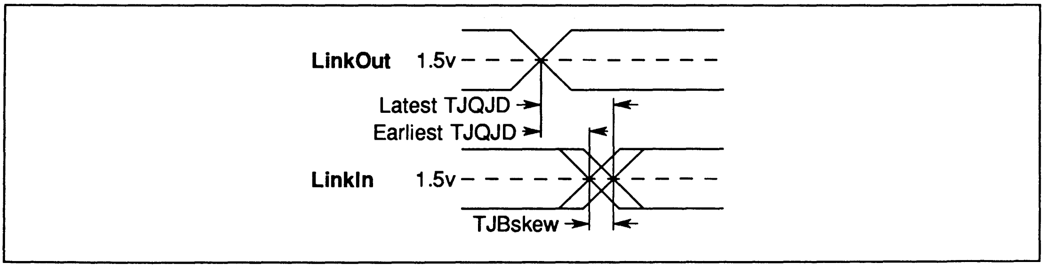


Figure 9.3 IMS T414 buffered link timing

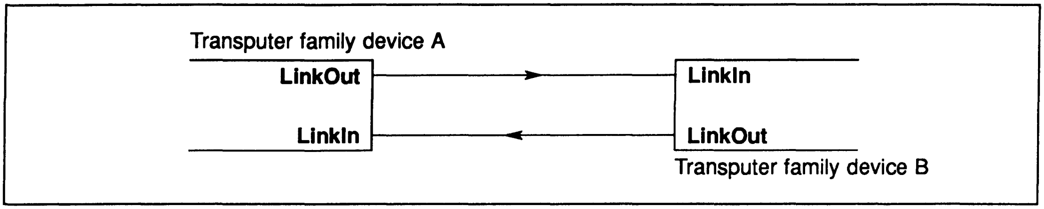


Figure 9.4 IMS T414 Links directly connected

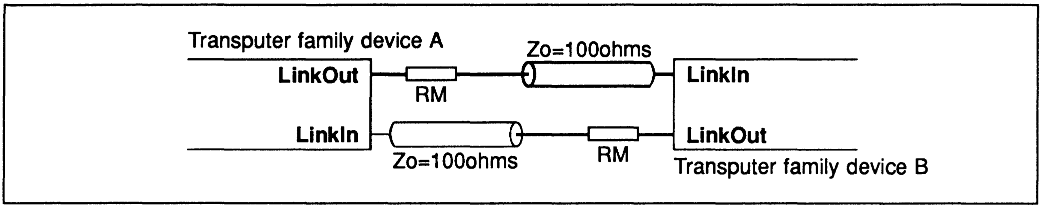


Figure 9.5 IMS T414 Links connected by transmission line

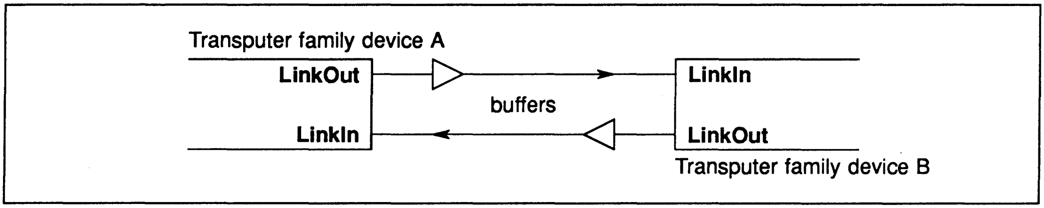


Figure 9.6 IMS T414 Links connected by buffers

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	1,2,7
				±50	μA	1,2,8
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			900	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 T414-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.

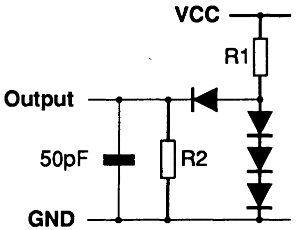
5 Power dissipation varies with output loading and program execution.

6 This parameter is sampled and not 100% tested.

7 For inputs other than those in Note 8.

8 For **MemReq, MemWait, MemConfig, Analyse, Reset, ClockIn, EventReq, LinkIn0-3, LinkSpecial, Link0Special, Link123Special, BootFromRom, HoldToGND**.

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

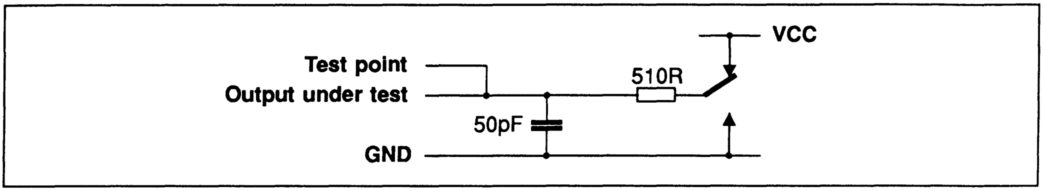


Figure 10.2 Tristate load circuit for AC measurements

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
TS0LaHZ	Address high to tristate	**a**	**a**+6	ns	3
TS0LaLZ	Address low to tristate	**a**	**a**+6	ns	3

Notes

1 Non-link pins; see section on links.

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

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

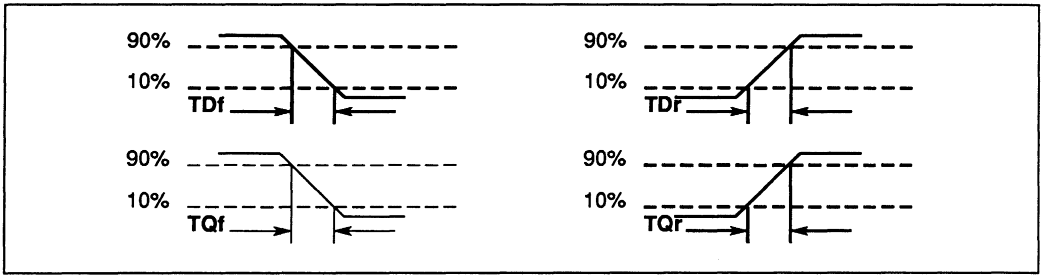


Figure 10.3 IMS T414 input and output edge timing

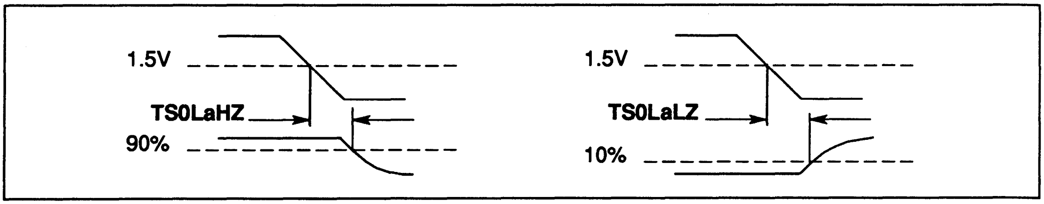


Figure 10.4 IMS T414 tristate timing relative to notMemS0

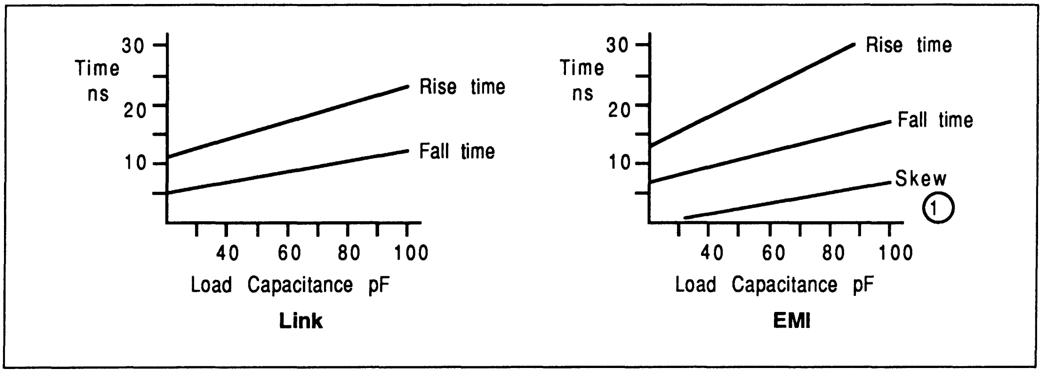


Figure 10.5 Typical rise/fall times

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

Internal power dissipation P_{INT} of transputer and peripheral chips depends on **VCC**, as shown in figure 10.6. 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.

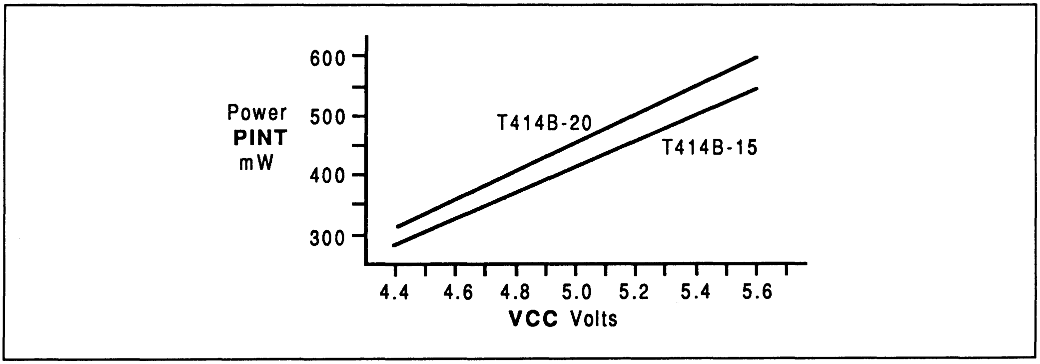


Figure 10.6 IMS T414 internal power dissipation vs VCC

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	39
`/`	2	40
`REM`	2	38
`>>` `<<`	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+2Eg+2Et	34+2Eg+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+2Eg	20+2Eg
skip alt guard	8+2Eg	10+2Eg
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 T414 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 35 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. 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		34	8
LONGDIV		36	8
SHIFTRIGHT	(n<32)	4+n	8
	(n>=32)	n-27	
SHIFTLEFT	(n<32)	4+n	8
	(n>=32)	n-27	
NORMALISE	(n<32)	n+6	7
	(n>=32)	n-25	
	(n=64)	4	
ASHIFTRIGHT		**SHIFTRIGHT**+2	5
ASHIFTLEFT		**SHIFTLEFT**+4	5
ROTATERIGHT		**SHIFTRIGHT**	7
ROTATELEFT		**SHIFTLEFT**	7
FRACMUL		**LONGPROD**+4	5

† Assuming local variables.

11.4 IMS T414 floating point operations

Floating point operations for the IMS T414 are provided by a run-time package. This requires approximately 400 bytes of memory for the single length arithmetic operations, and 2500 bytes for the double length arithmetic operations. Table 11.5 summarizes the estimated performance of the package.

Table 11.5 IMS T414 floating point operations performance

		Processor cycles	
		IMS T414	
		Typical	Worst
REAL32	+ -	230	300
	*	200	240
	/	245	280
	< > = >= <= <>	60	60
REAL64	+ -	565	700
	*	760	940
	/	1115	1420
	< > = >= <= <>	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**. For the IMS T414, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

If 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 (**e**-3)/4. 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 T414	
	Program off chip	**Data off chip**
Boolean expressions	**e**-2	0
`IF`	3**en**-8	**en**
Replicated `IF`	(6**e**-4)**n**+7	(5**e**-2)**n**+8
Replicated `SEQ`	(3**e**-3)**n**+2	(4**e**-2)**n**
`PAR`	(3**e**-1)**n**+8	3**en**+4
Replicated `PAR`	(10**e**-8)**n**+8	16**en**-12
`ALT`	(2**e**-4)**n**+6**e**	(2**e**-2)**n**+10**e**-8
Array assignment and communication in one transputer	0	max (2**e**, **e**(**b**/2))

For the IMS T414 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 T414 external memory performance

	Program	e=2	e=3	e=4	e=5	On chip
Program off chip	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
Data off chip	1	1.5	1.8	2.1	2.3	1
	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

11.6 Interrupt latency

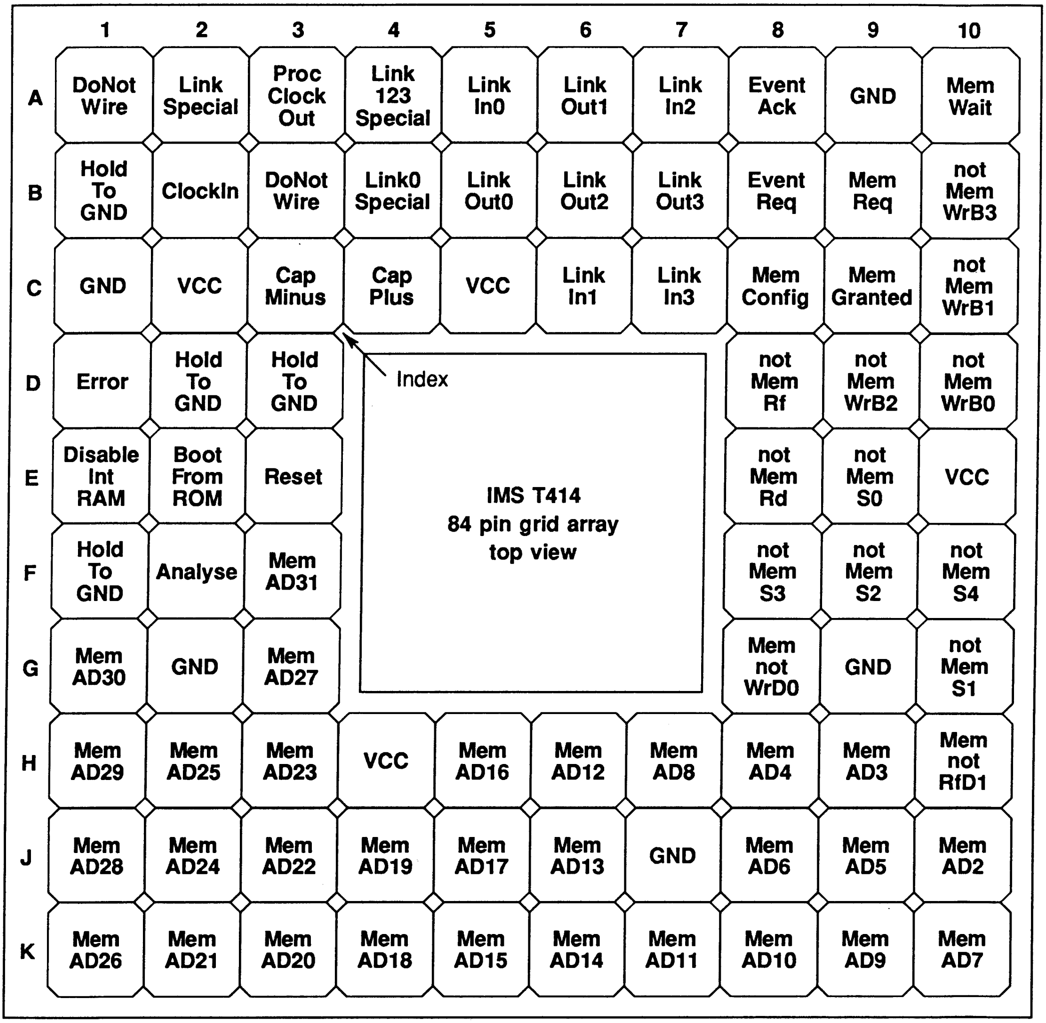
If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.8. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal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

Table 11.8 Interrupt latency

	Typical		Maximum	
	TPCLPCL	Tm	TPCLPCL	Tm
IMS T414	19	38	53	116

12 Package specifications

12.1 84 pin grid array package



	1	2	3	4	5	6	7	8	9	10
A	DoNot Wire	Link Special	Proc Clock Out	Link 123 Special	Link In0	Link Out1	Link In2	Event Ack	GND	Mem Wait
B	Hold To GND	ClockIn	DoNot Wire	Link0 Special	Link Out0	Link Out2	Link Out3	Event Req	Mem Req	not Mem WrB3
C	GND	VCC	Cap Minus	Cap Plus	VCC	Link In1	Link In3	Mem Config	Mem Granted	not Mem WrB1
D	Error	Hold To GND	Hold To GND					not Mem Rf	not Mem WrB2	not Mem WrB0
E	Disable Int RAM	Boot From ROM	Reset					not Mem Rd	not Mem S0	VCC
F	Hold To GND	Analyse	Mem AD31					not Mem S3	not Mem S2	not Mem S4
G	Mem AD30	GND	Mem AD27					Mem not WrD0	GND	not Mem S1
H	Mem AD29	Mem AD25	Mem AD23	VCC	Mem AD16	Mem AD12	Mem AD8	Mem AD4	Mem AD3	Mem not 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

Figure 12.1 IMS T414 84 pin grid array package pinout

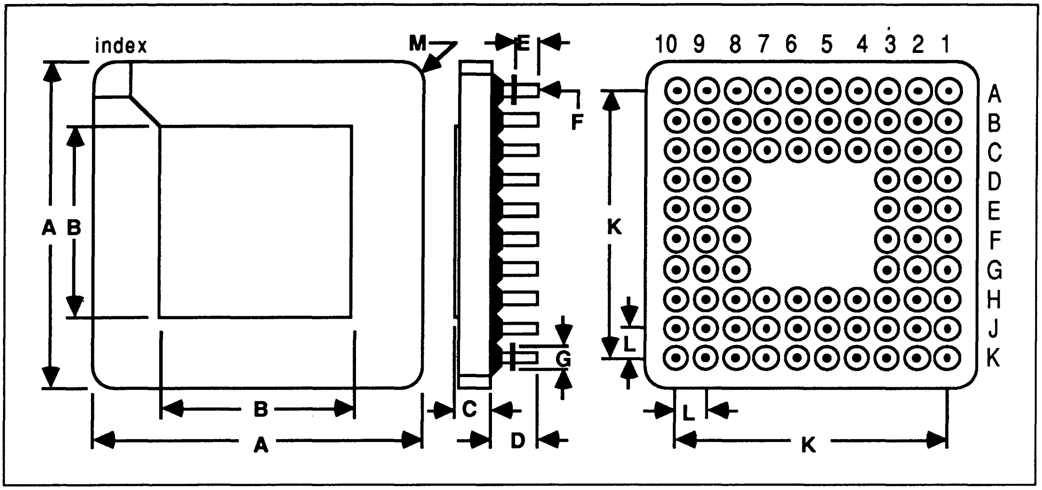


Figure 12.2 84 pin grid array package dimensions

Table 12.1 84 pin grid array package dimensions

DIM	Millimetres NOM	Millimetres TOL	Inches NOM	Inches TOL	Notes
A	26.924	±0.254	1.060	±0.010	
B	17.019	±0.127	0.670	±0.005	
C	2.456	±0.278	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	±0.025	0.018	±0.001	Pin diameter
G	1.143	±0.127	0.045	±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 7.2 grams

Table 12.2 84 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	

12.1.1 84 pin PLCC J-bend package

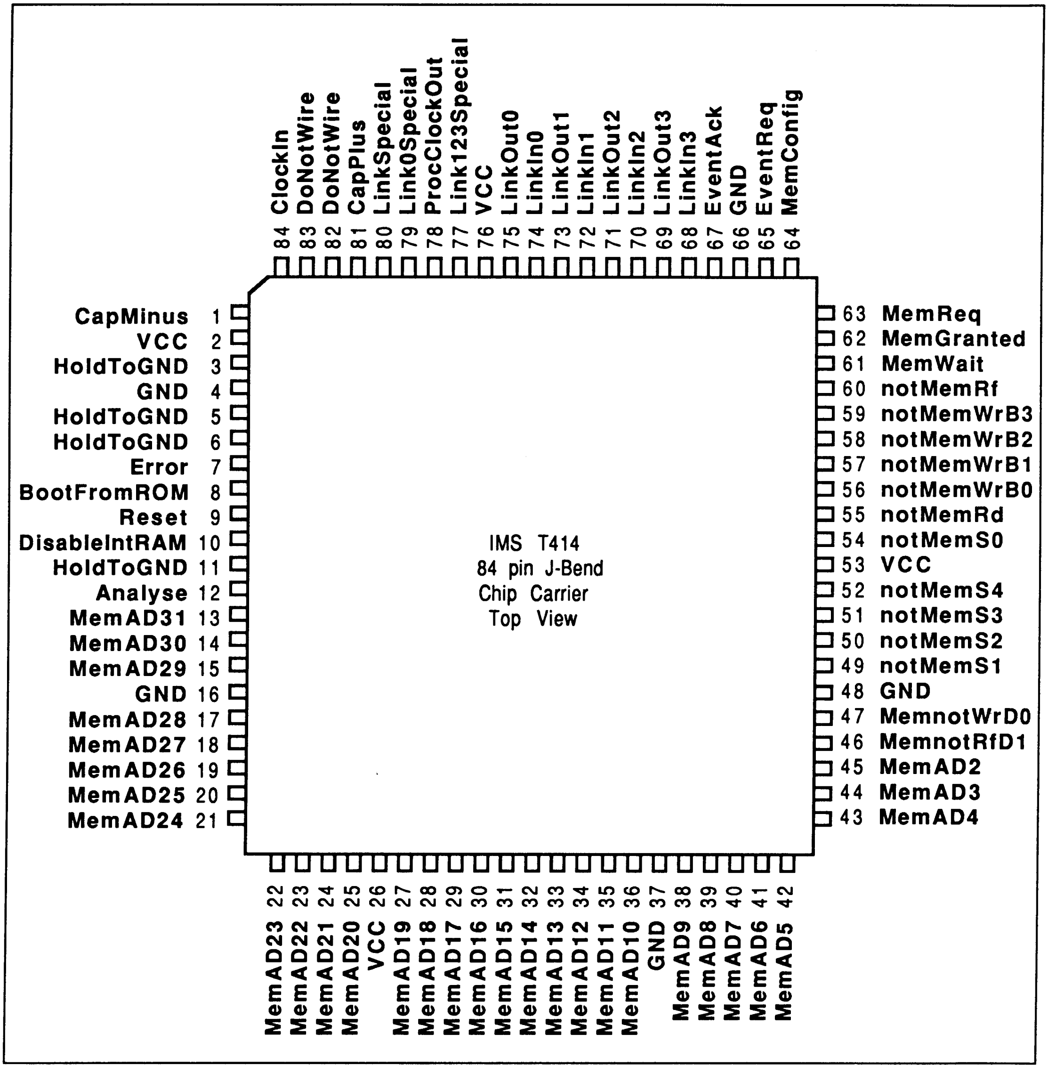


Figure 12.3 IMS T414 84 pin PLCC J-bend package pinout

Notes

1 Since the manufacture of the IMS T414, the pin numbers for the 84 pin J-bend chip carrier have been changed, however the pin functions remain the same.

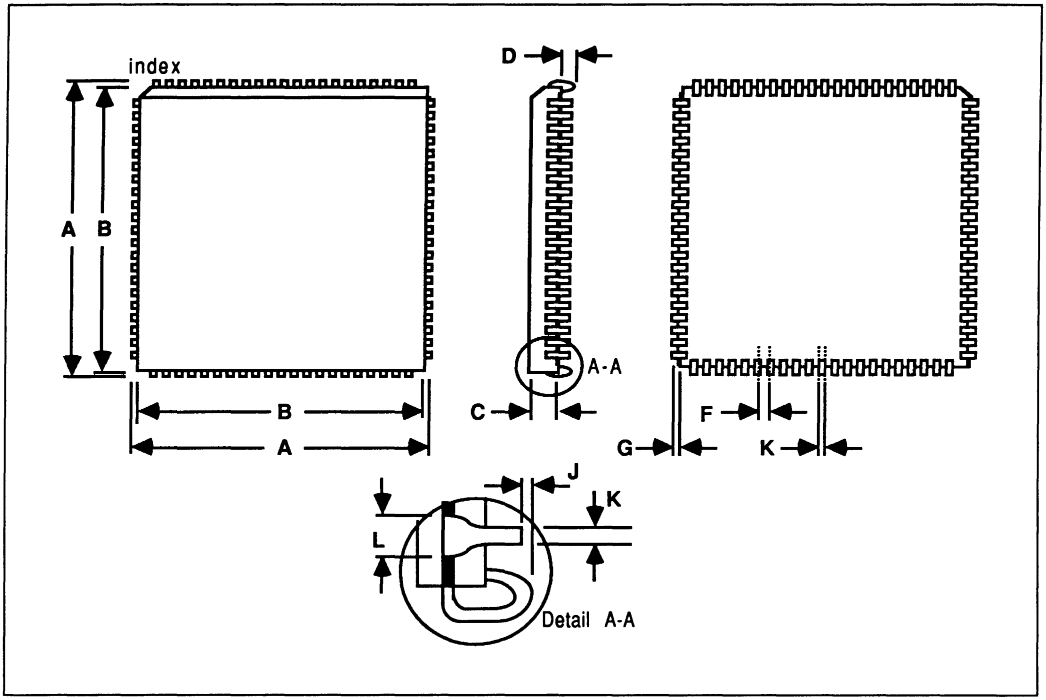


Figure 12.4 84 pin PLCC J-bend package dimensions

Table 12.3 84 pin PLCC J-bend package dimensions

	Millimetres		Inches		
DIM	NOM	TOL	NOM	TOL	Notes
A	30.226	±0.127	1.190	±0.005	
B	29.312	±0.127	1.154	±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 7.0 grams

Table 12.4 84 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	

13 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 13.1 IMS T414 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T414-G15S	15 MHz	67 ns	3.0	Ceramic Pin Grid
IMS T414-G20S	20 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T414-J15S	15 MHz	67 ns	3.0	Plastic PLCC J-Bend
IMS T414-J20S	20 MHz	50 ns	4.0	Plastic PLCC J-Bend

IMS T222
transputer

inmos

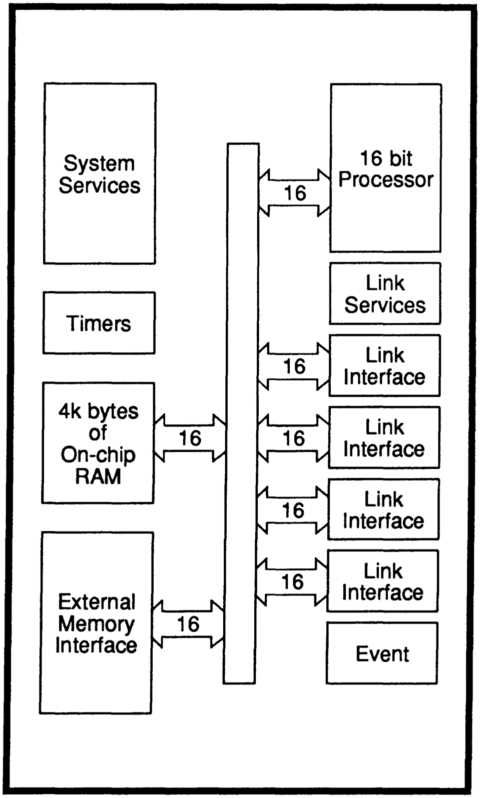
Engineering Data

FEATURES

16 bit architecture
50 ns internal cycle time
20 MIPS (peak) instruction rate
Pin compatible with IMS T212
4 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 2.4 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
MIL-STD-883C processing will be available

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



October 1988

1 Introduction

The IMS T222 transputer is a 16 bit CMOS microcomputer with 4 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 20 MIPS peak. The extended temperature version of the device complies with MIL-STD-883C.

For convenience of description, the IMS T222 operation is split into the basic blocks shown in figure 1.1.

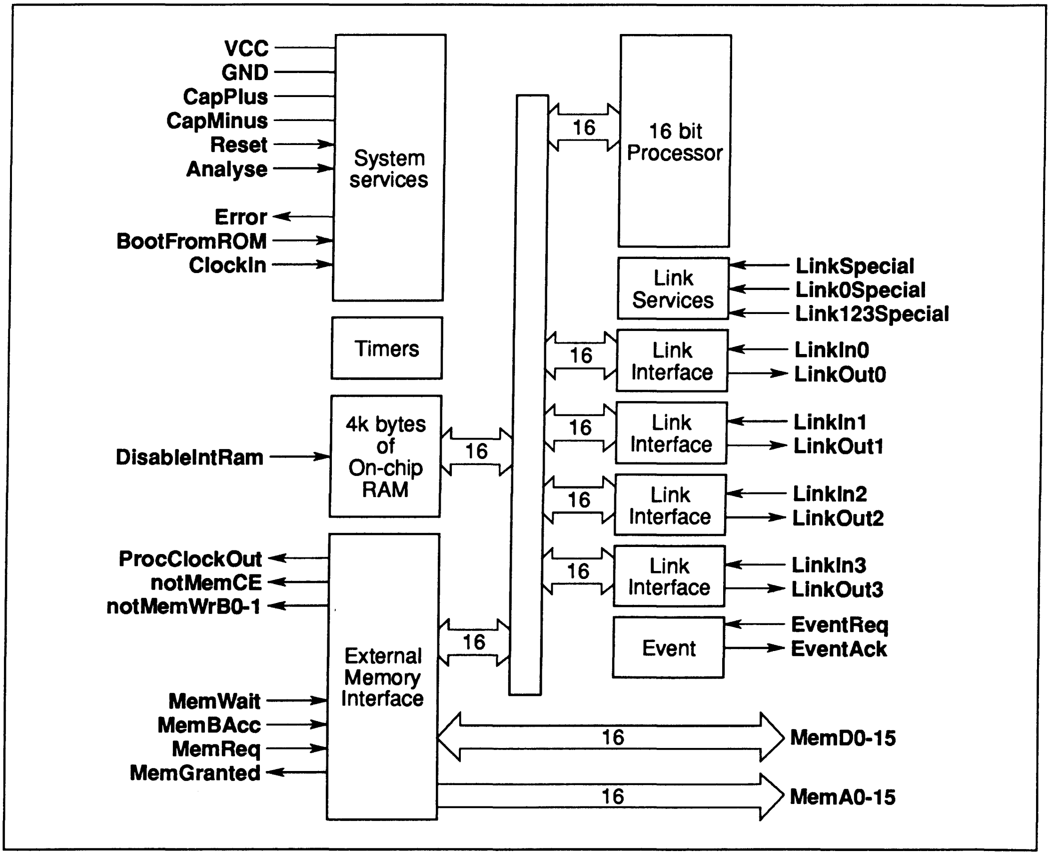


Figure 1.1 IMS T222 block diagram

The IMS T222 is functionally equivalent to the IMS T212 but has a more easily connected memory interface and improved links. The devices are pin-compatible and the IMS T222 is a direct replacement in many applications. The IMS T222 can directly access a linear address space of 64 Kbytes. The 16 bit wide non-multiplexed external memory interface provides a data rate of up to 2 bytes every 100 nanoseconds (20 Mbytes/sec) for a 20 MHz device.

System Services include processor reset and bootstrap control, together with facilities for error analysis.

The INMOS communication links allow networks of transputers to be constructed by direct point to point connections with no external logic. The links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec. The links have been improved over those of the IMS T212 and support overlapped acknowledge; each IMS T222 link can transfer data bi-directionally at up to 2.05 Mbytes/sec.

The IMS T222 is designed to implement the occam language, detailed in the occam Reference Manual, but also efficiently supports other languages such as C and Pascal. 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 T222. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of occam.

2 Pin designations

Table 2.1 IMS T222 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 T222 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 T222 event

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

Table 2.4 IMS T222 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 291.

3 Processor

The 16 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 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.

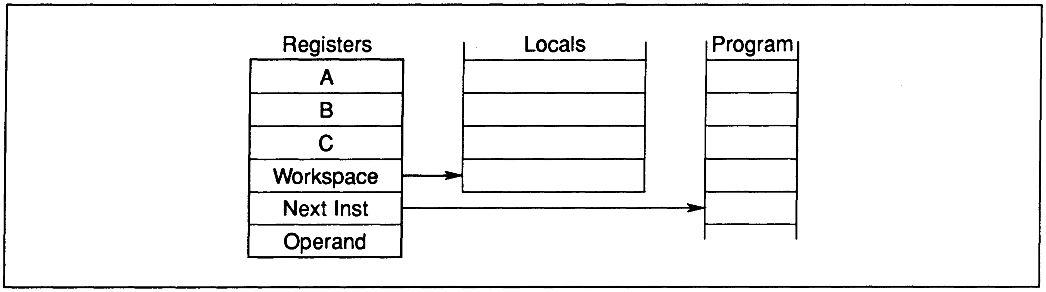


Figure 3.1 Registers

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.

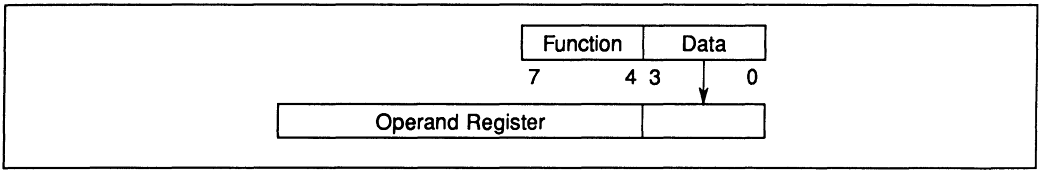


Figure 3.2 Instruction format

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 251).

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 251). 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.

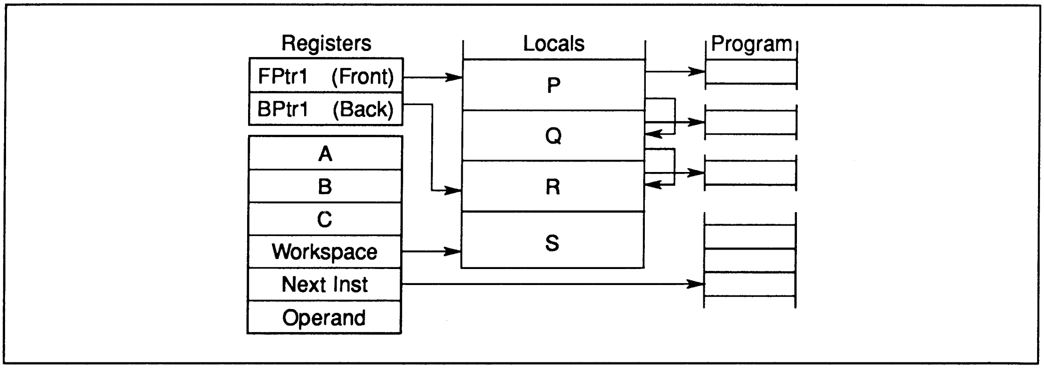


Figure 3.3 Linked process list

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 254). 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 254). 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 T222 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 254).

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.

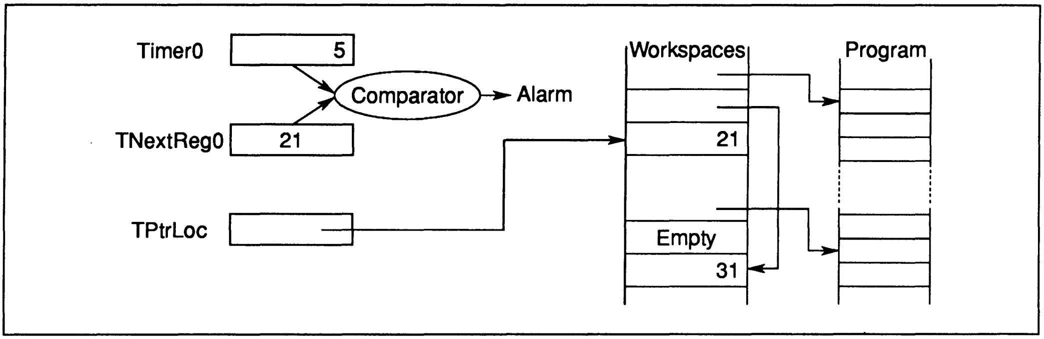


Figure 3.4 Timer registers

4 Instruction set summary

The Function Codes table 4.7. gives the basic function code set (page 248). 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	254
E	The instruction will affect the *Error* flag	254, 263

4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 250). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 262).

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 263) 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 T222 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 T222 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 T222 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 T222 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 T222 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	2w+8	move message	

Table 4.12 IMS T222 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 T222 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 T222 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 T222 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 T222 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 T222 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.

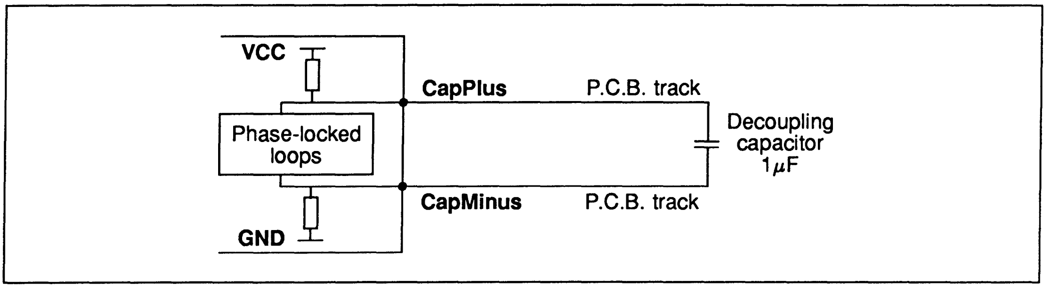


Figure 5.1 Recommended PLL decoupling

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).

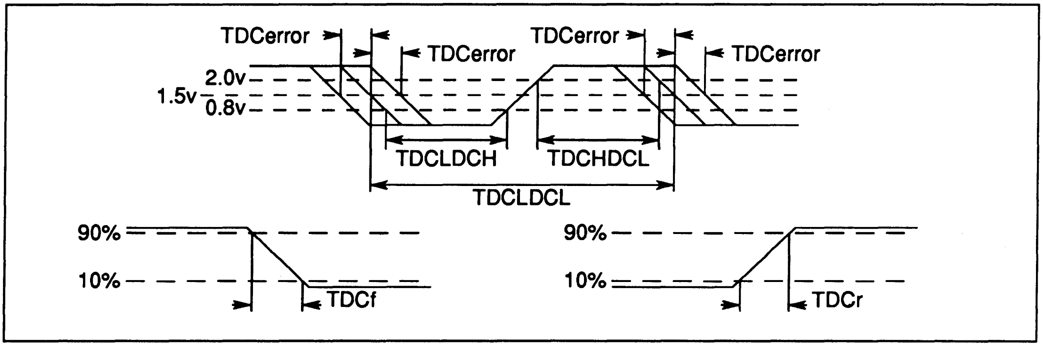


Figure 5.2 ClockIn timing

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 264).

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.

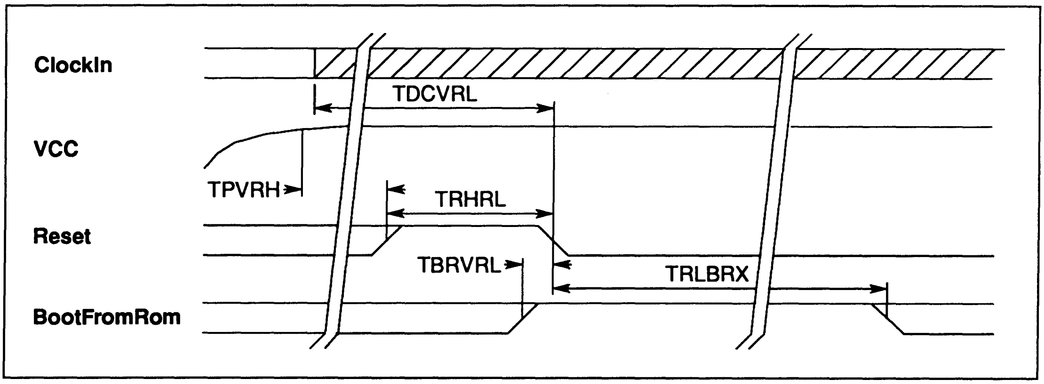


Figure 5.3 Transputer reset timing with Analyse low

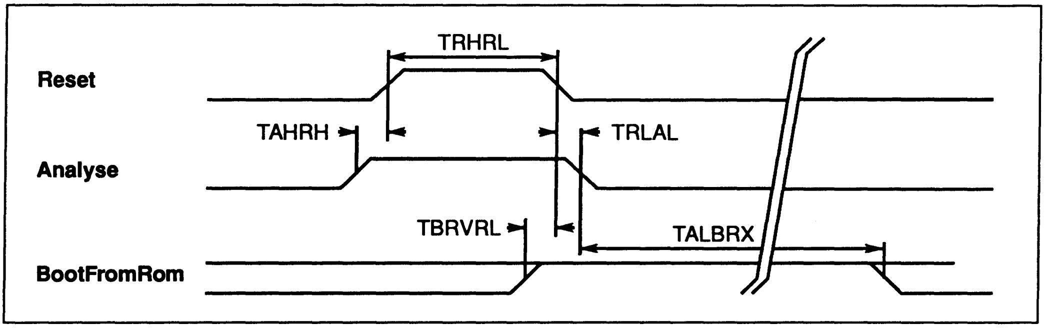


Figure 5.4 Transputer reset and analyse timing

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 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 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 254). 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 254). 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 262).

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*.

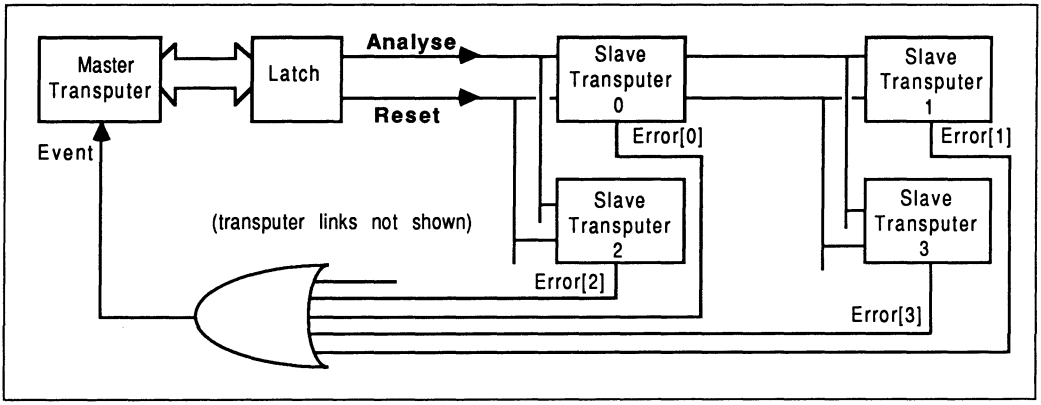


Figure 5.5 Error handling in a multi-transputer system

6 Memory

The IMS T222 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 266). The transputer can also access an additional 60 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 T222 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 #8FFF. 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 #9000 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.

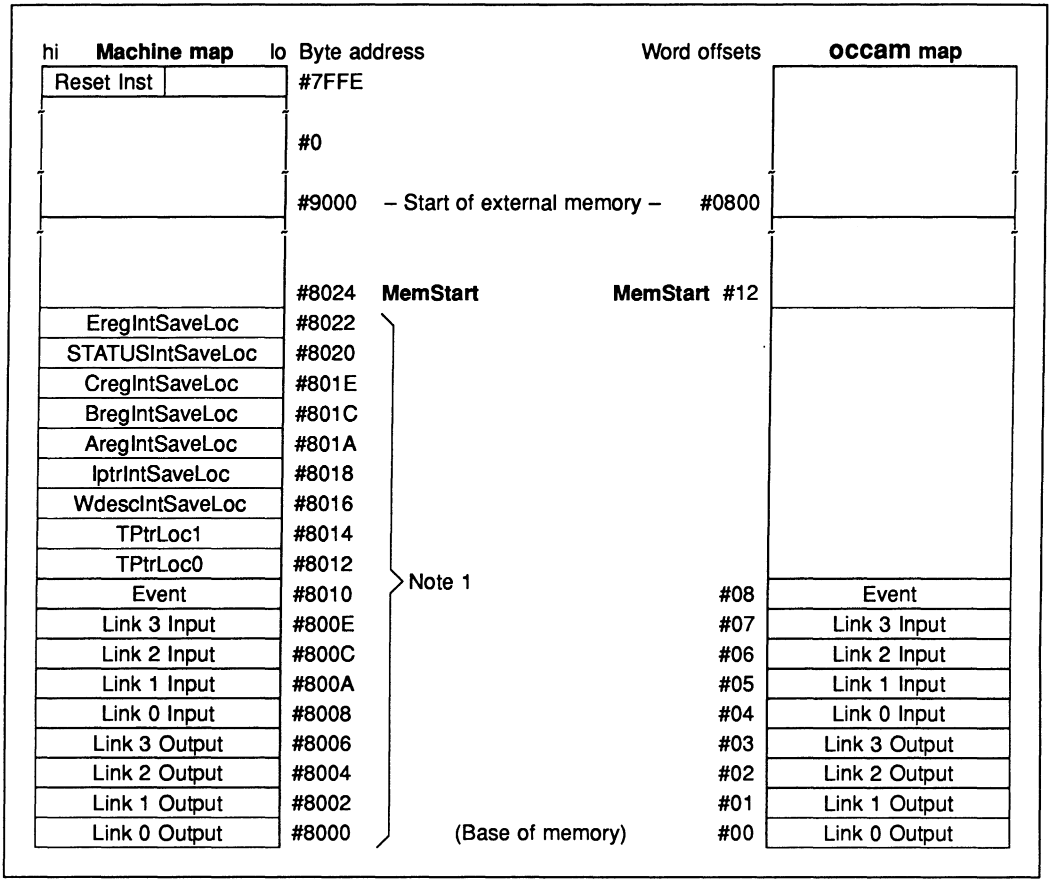


Figure 6.1 IMS T222 memory map

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 262). For details see The Transputer Instruction Set - A Compiler Writers' Guide.

7 External memory interface

The IMS T222 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 272).

The IMS T222 has the ability to handle slower access static RAMs than the IMS T212 and can operate with a simpler wait state generator. In addition, the write strobes are no longer active during internal memory access in order to simplify external memory interfacing.

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**. **ProcClockOut** has a different phase relationship with the strobes when compared to the IMS T212.

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.

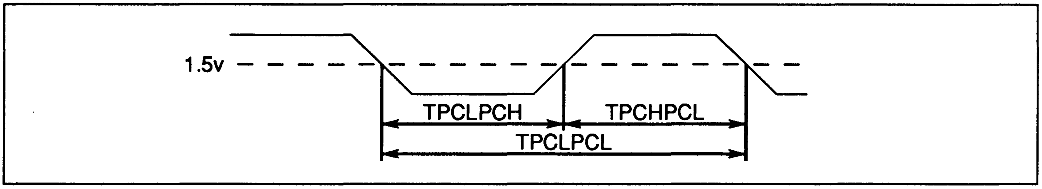


Figure 7.1 IMS T222 ProcClockOut timing

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. An external memory cycle is always a complete number of cycles **TPCLPCL** in length and the start of **T1** always coincides with a rising edge of edge of **ProcClockOut**. **T2** can be extended indefinitely by adding externally generated wait states of one complete processor cycle each.

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** and **notMemCE** are 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 262).

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 T222 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 T222 reads it during **T4**. Data can be removed any time after the rising edge of **notMemCE**, but must be off the bus no later than the end of **T4**.

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 272).

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 273).

Table 7.2 Read

SYMBOL	PARAMETER	T222-20		T222-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TAVEL	Address valid before chip enable low	10		12		ns	1
TELEH	Chip enable low	72	78	83	88	ns	1
TEHEL	Delay before chip enable re-assertion	20		24		ns	1,2
TEHAX	Address hold after chip enable high	10		12		ns	1
TELDrV	Data valid from chip enable low	47		53		ns	
TAVDrV	Data valid from address valid	57		65		ns	
TDrVEH	Data setup before chip enable high	25		30		ns	
TEHDrZ	Data hold after chip enable high	0	20	0	24	ns	
TWEHEL	Write enable setup before chip enable low	20		24		ns	3
TPCHEL	ProcClockOut high to chip enable low	10		12		ns	1

Notes

1 This parameter is common to read and write cycles and to byte-wide memory accesses.

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

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

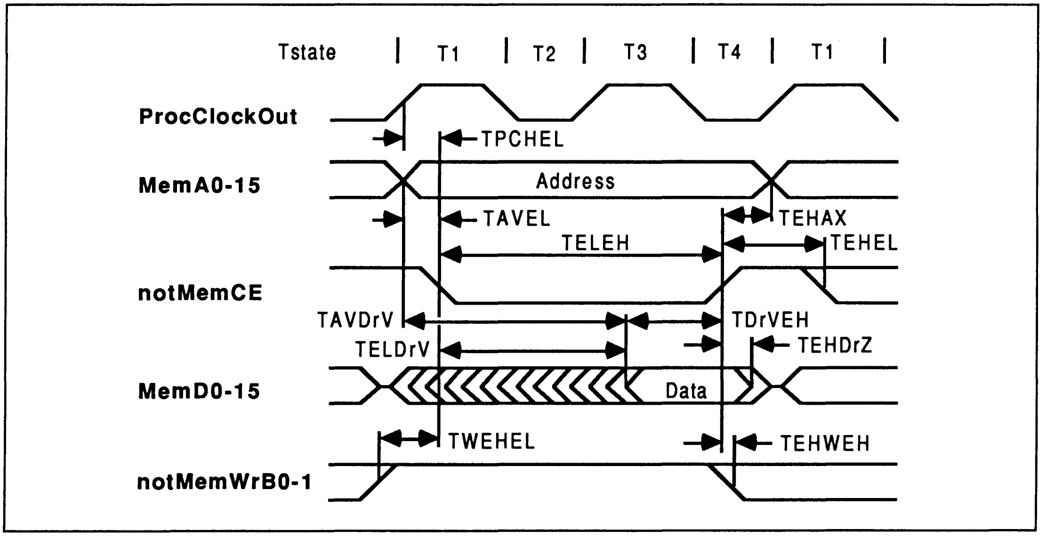


Figure 7.2 IMS T222 external read cycle

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 synchronised with the chip enable signal **notMemCE**, allowing them to be used without **notMemCE** for simple designs.

Data may be strobed into memory using **notMemWrB0-1** without the use of **notMemCE**, as the write enables go high between consecutive external memory write cycles. The write enables are placed in a high impedance state during DMA, and are inactive during internal memory access.

Table 7.3 Write

SYMBOL	PARAMETER	T222-20 MIN	T222-20 MAX	T222-17 MIN	T222-17 MAX	UNITS	NOTE
TDwVEH	Data setup before chip enable high	50		57		ns	
TEHDwZ	Data hold after write	10	15	12	17	ns	
TDwZEL	Write data invalid to next chip enable	10		12		ns	
TWELEL	Write enable setup before chip enable low	-3	0	-4	0	ns	1
TEHWEH	Write enable hold after chip enable high	0	3	0	4	ns	1

Notes

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

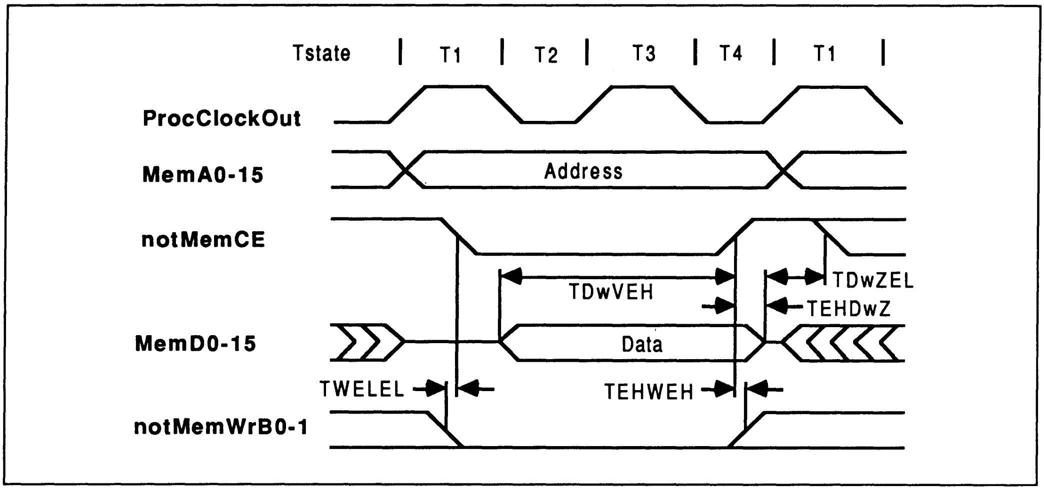


Figure 7.3 IMS T222 external write cycle

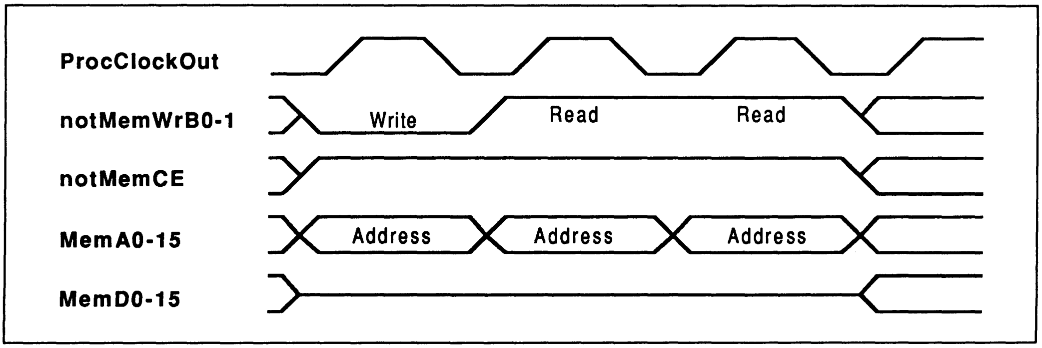


Figure 7.4 IMS T222 typical bus activity for internal memory cycles

7.7 notMemCE

The active low signal **notMemCE** is used to enable external memory on both read and write cycles.

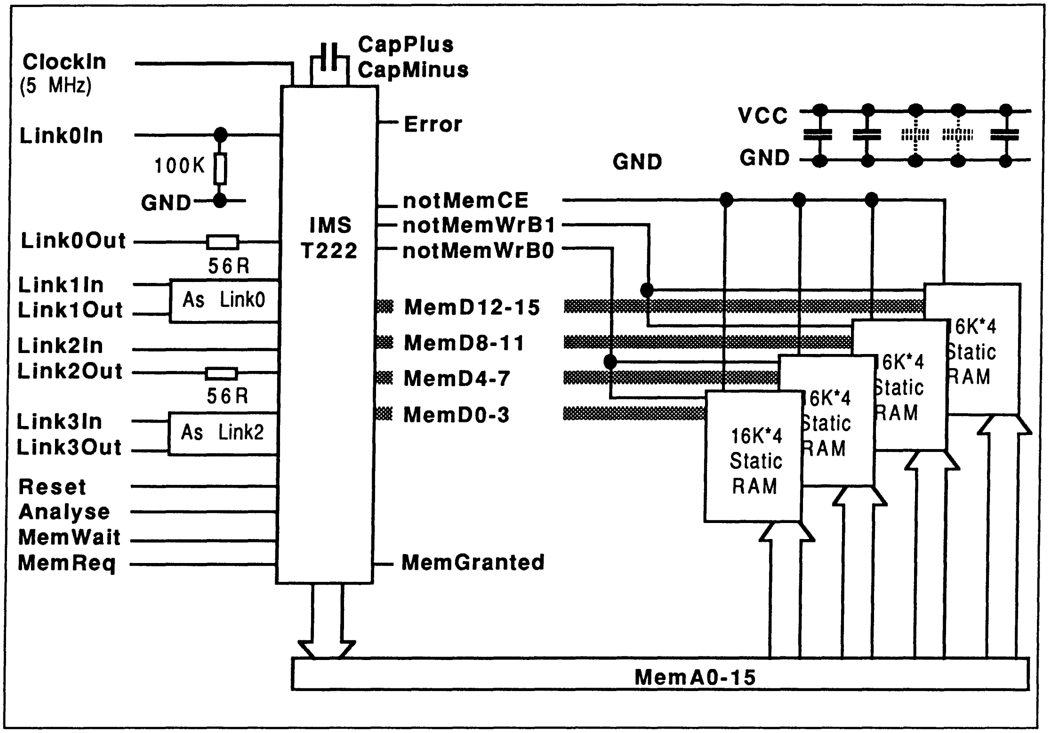


Figure 7.5 IMS T222 application

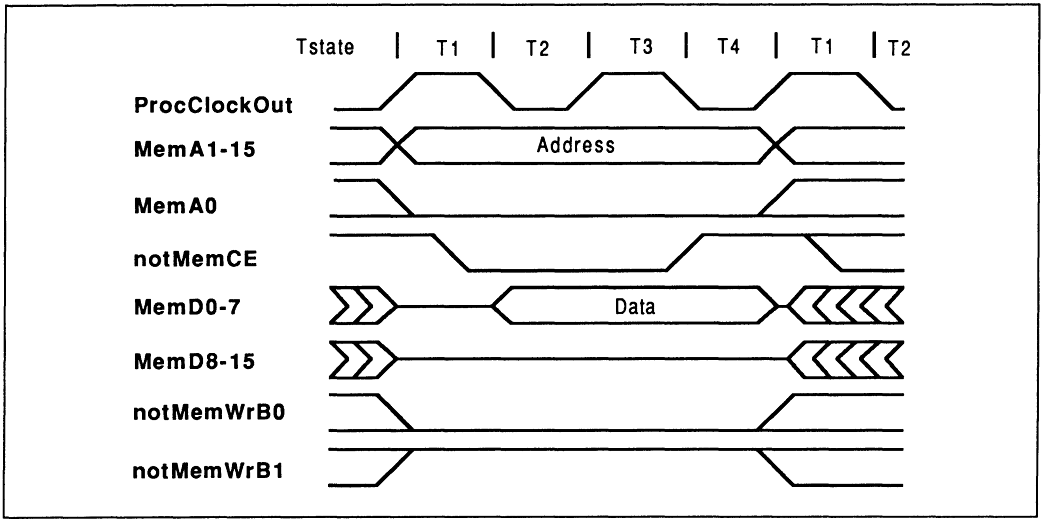


Figure 7.6 IMS T222 Least significant byte write in word access mode

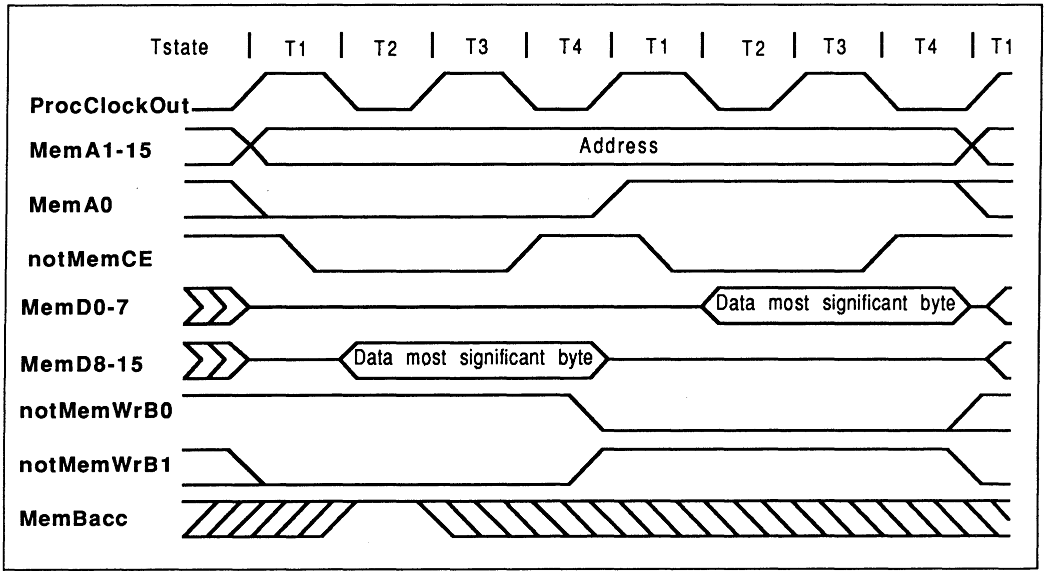


Figure 7.7 IMS T222 Most significant byte write to byte-wide memory

7.8 MemBAcc

The IMS T222 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 7.4 Byte-wide memory access

SYMBOL	PARAMETER	T222-20		T222-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TELBAH	MemBAcc high from chip enable		12		15	ns	
TELBAL	MemBAcc low from chip enable	26		29		ns	

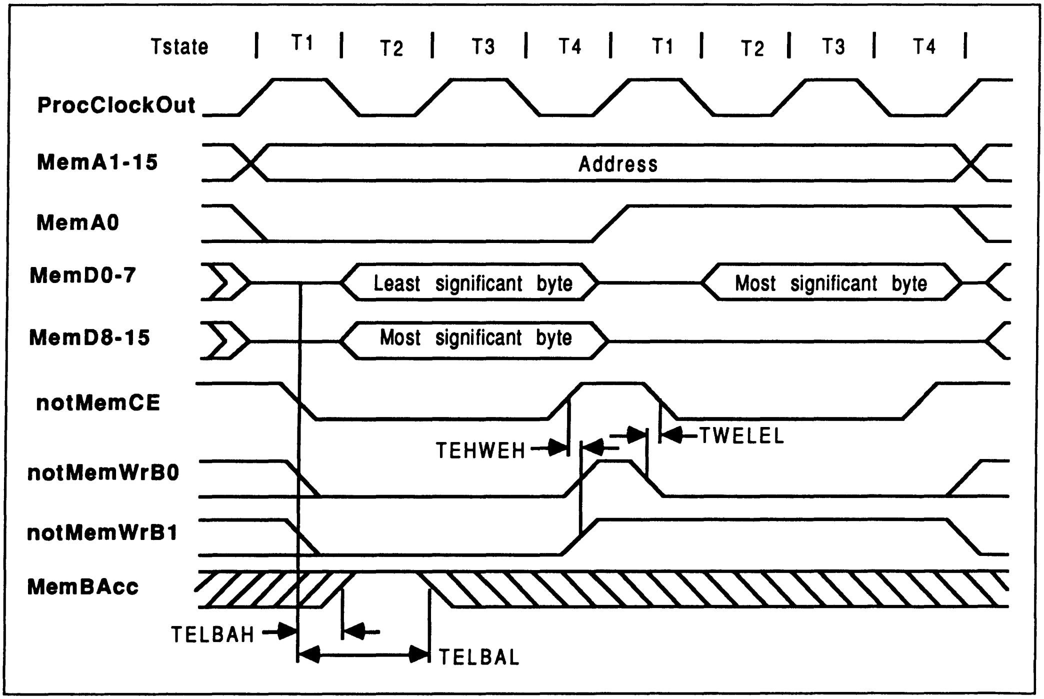


Figure 7.8 IMS T222 word write to byte-wide memory

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 IMS T222 has modified internal circuitry, compared with the IMS T212, to reduce the quantity of external components required. The **Single Wait State Generator** circuit in figure 7.10 can be extended to provide two or more wait states, as shown in figure 7.11.

Table 7.5 Memory wait

SYMBOL	PARAMETER	T222-20		T222-17		UNITS	NOTE
		MIN	MAX	MIN	MAX		
TPCHWtH	MemWait asserted after ProcClockOut high		25		27	ns	
TPCHWtL	Wait low after ProcClockOut high	35		39		ns	

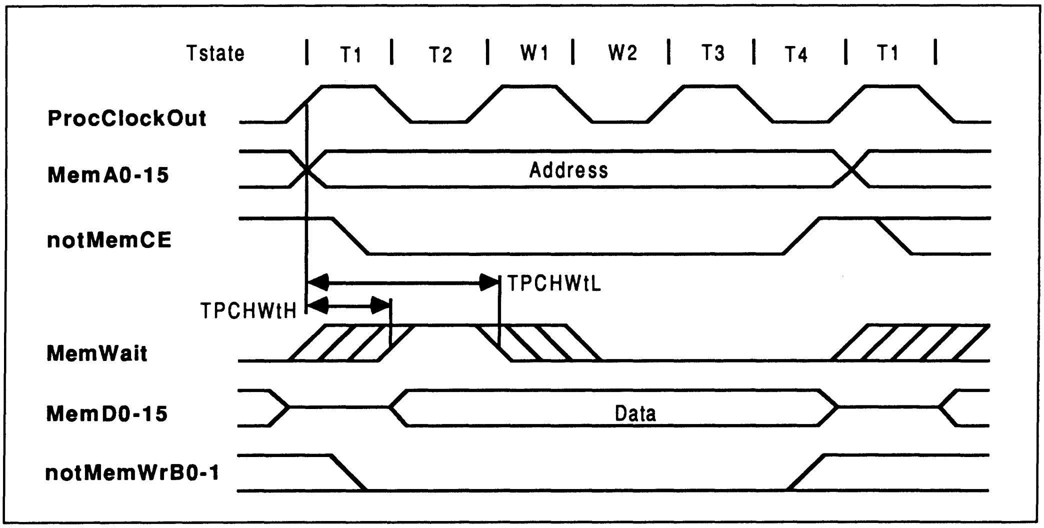


Figure 7.9 IMS T222 memory wait timing

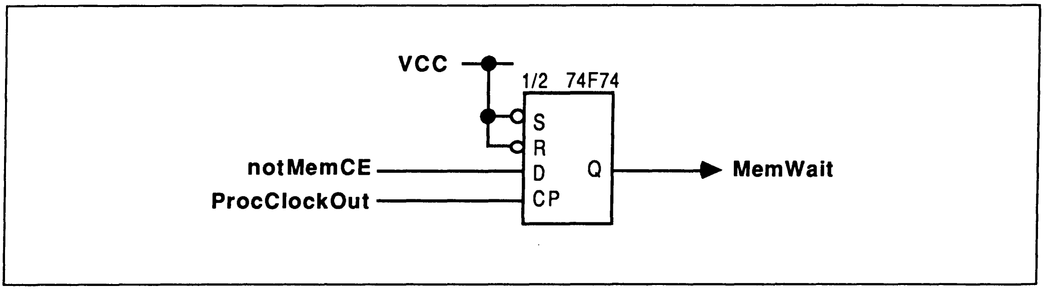


Figure 7.10 Single wait state generator

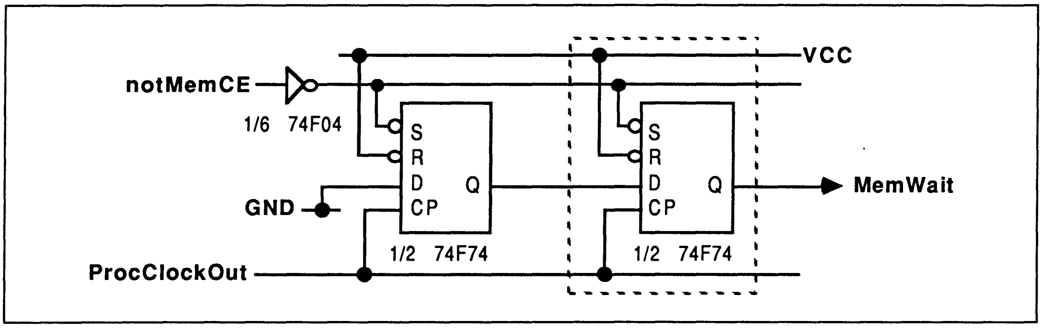


Figure 7.11 Extendable wait state generator

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 T222 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 rising edge of **ProcClockOut**. **MemA0-15**, **MemD0-15**, **notMemWrB0-1** and **notMemCE** are tristated before **MemGranted** is asserted.

Removal of **MemReq** is sampled at each rising edge 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.

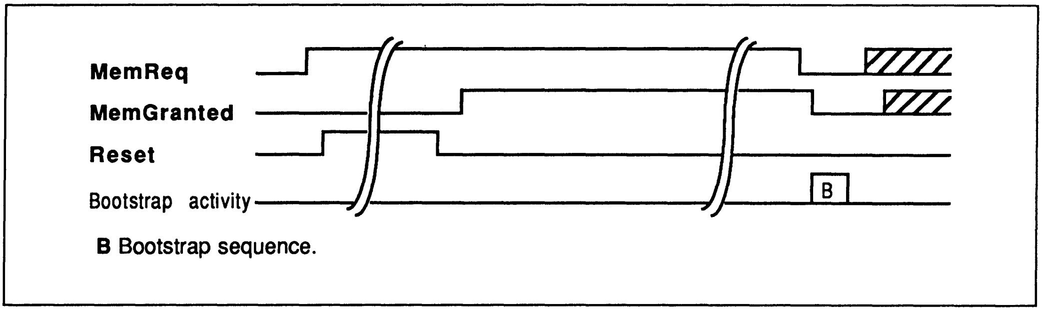


Figure 7.12 IMS T222 DMA sequence at reset

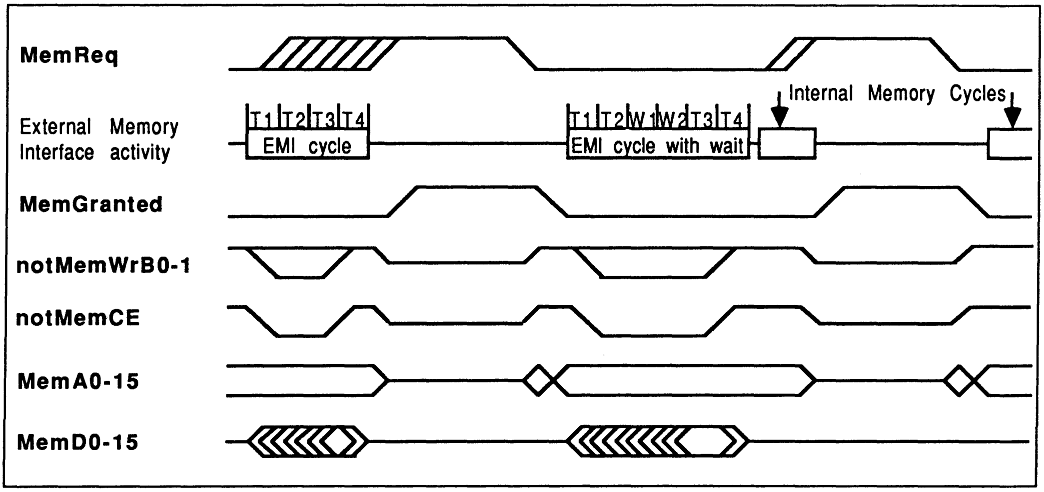


Figure 7.13 IMS T222 operation of **MemReq** and **MemGranted** with external and internal memory cycles

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 7.6 Memory request

SYMBOL	PARAMETER	T222-20		T222-17		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	Chip enable tristate before MemGranted	0		0		ns	2
TEVMGL	Chip enable active after MemGranted end	0		0		ns	
TWEZMGH	Write enable tristate before MemGranted	0		0		ns	2
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.
2. When using DMA, **notMemCE** and **notMemWrB0-1** should be pulled up with a resistor (typically 1.2k). Capacitance should be limited to a maximum of 50pF.

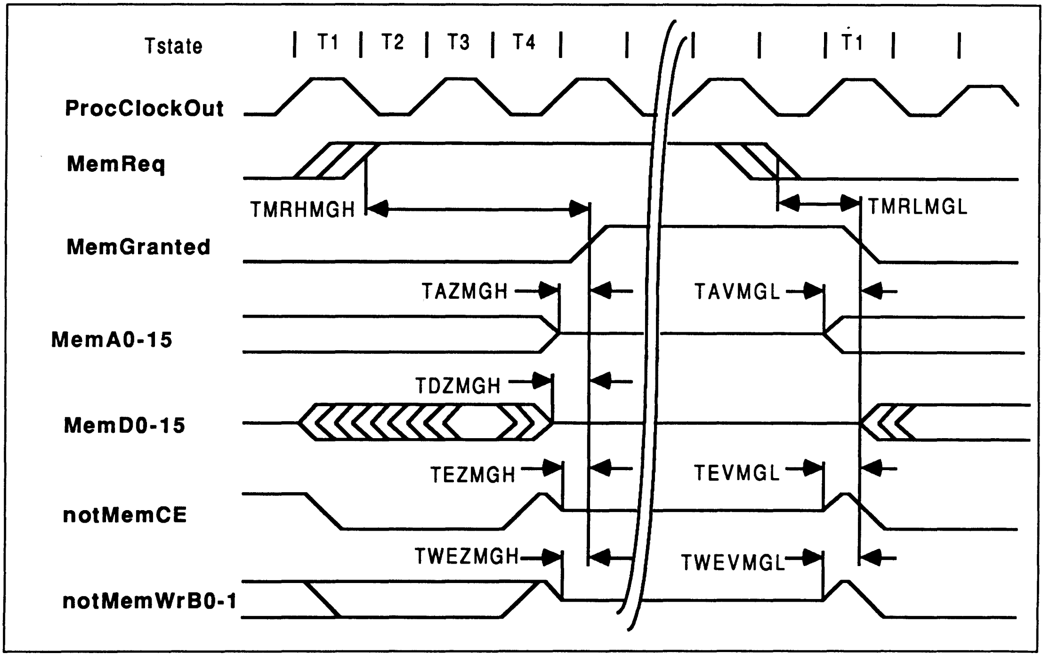


Figure 7.14 IMS T222 memory request timing

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 251. 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**.

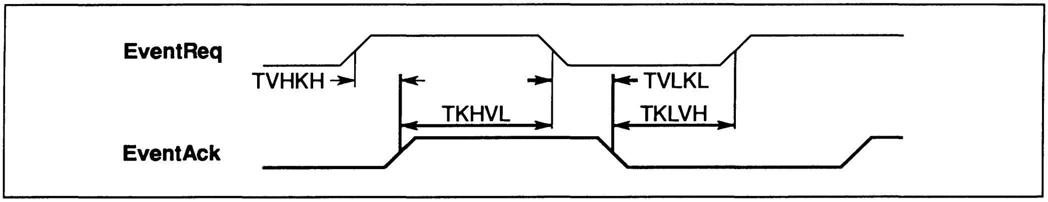


Figure 8.1 IMS T222 event timing

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 T222 links allow 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 hard output channels are not double buffered. There is thus a pause between transmission of the last byte of a word of the message and the first byte of the next word. This pause time is related to memory speed. Hard input channels have one byte of double buffering and are unlikely to affect the data rate. The dominant factor affecting link bandwidth is therefore the memory bandwidth of the transmitting transputer, as shown in table 9.1. Internal memory access time is similar to zero wait state external access time. Times are for two interconnected IMS T222's with 20 Mbits/sec link speed.

Table 9.1 Memory/Link speed relationship

Memory Speed (20MHz device)	**Byte Output Time nS**	**Word Memory Read nS**	**Unidirectioal Data Rate Mbytes/sec**
1 cycle (0 wait)	575	200	1.48
2 cycle (1 wait)	575	250	1.42
3 cycle (2 wait)	575	300	1.38

The IMS T222 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 9.2 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 9.2 Speed Settings for Transputer Links

Link Special	Linkn Special	Mbits/sec	Kbytes/sec Uni	Kbytes/sec Bi
0	0	10	800	1130
0	1	5	430	590
1	0	10	800	1130
1	1	20	1480	2050

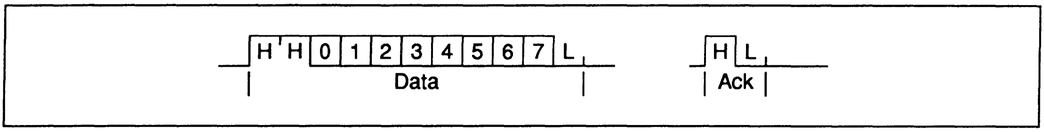


Figure 9.1 IMS T222 link data and acknowledge packets

Table 9.3 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.

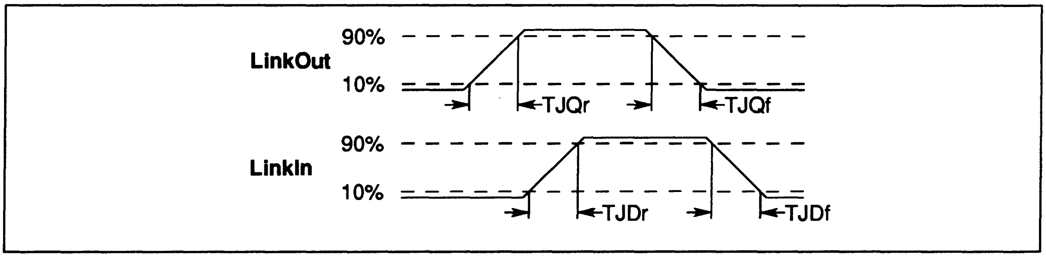


Figure 9.2 IMS T222 link timing

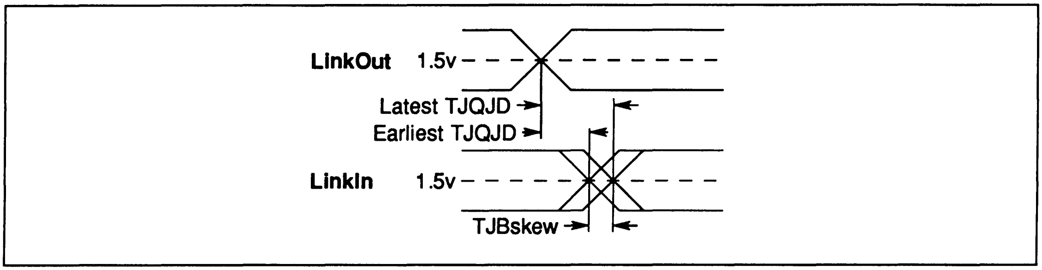


Figure 9.3 IMS T222 buffered link timing

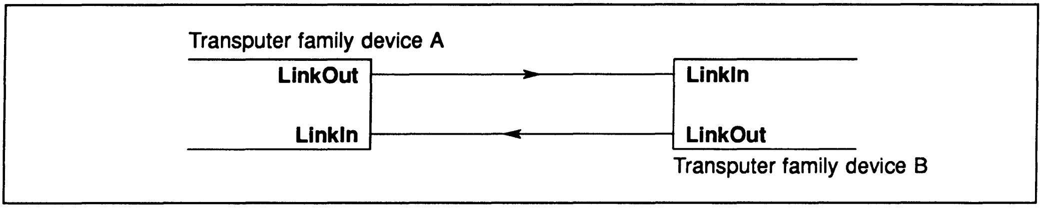


Figure 9.4 IMS T222 Links directly connected

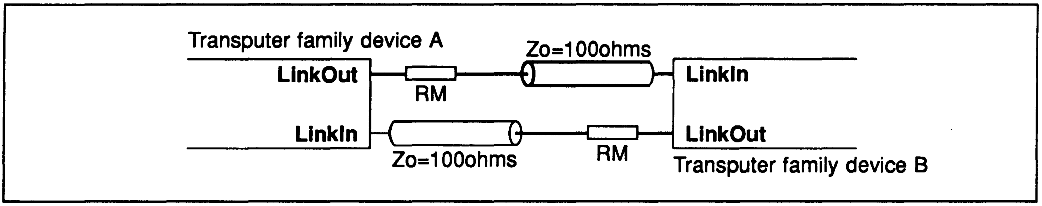


Figure 9.5 IMS T222 Links connected by transmission line

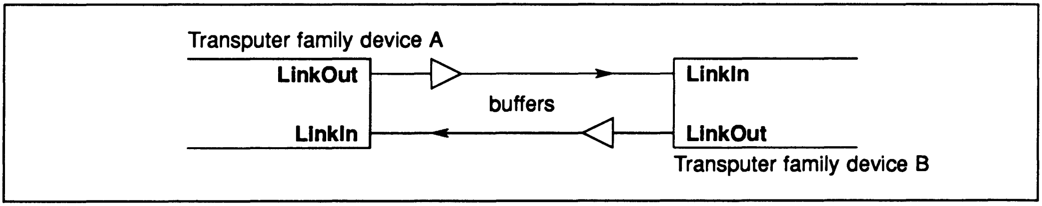


Figure 9.6 IMS T222 Links connected by buffers

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 IMS T222-S	0	70	°C	3
TA	Operating temperature range IMS T222-M	-55	125	°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	1,2,3,7
			65	100	mA	1,2,4,7
			25	90	mA	1,2,3,8
			50	130	mA	1,2,4,8
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 T222-S measured at 4.75V<**VCC**<5.25V and 0°C<**TA**<70°C.
Parameters for IMS T222-M measured at 4.75V<**VCC**<5.25V and -55°C<**TA**<125°C.
Input clock frequency = 5MHz.

3 Current sourced from non-link outputs.

4 Current sourced from link outputs.

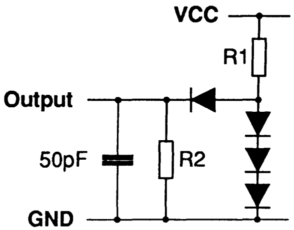
5 Power dissipation varies with output loading and program execution.

6 This parameter is sampled and not 100% tested.

7 Parameter for IMS T222-S.

8 Parameter for IMS T222-M.

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

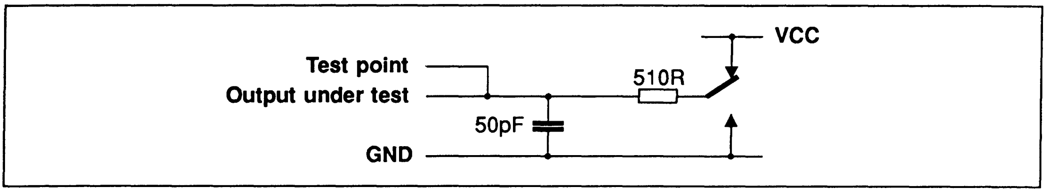


Figure 10.2 Tristate load circuit for AC measurements

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**.

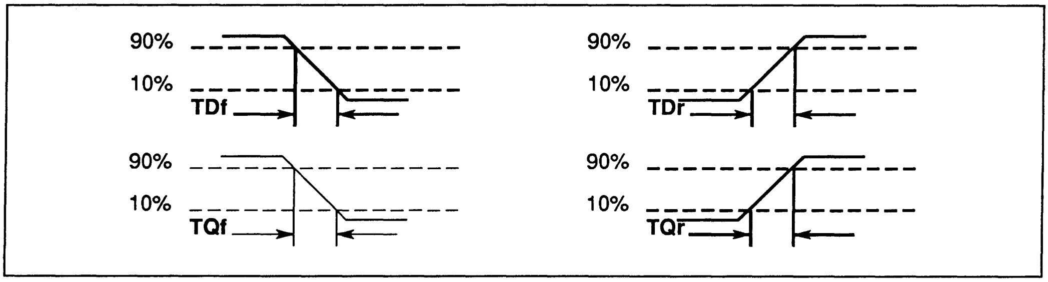


Figure 10.3 IMS T222 input and output edge timing

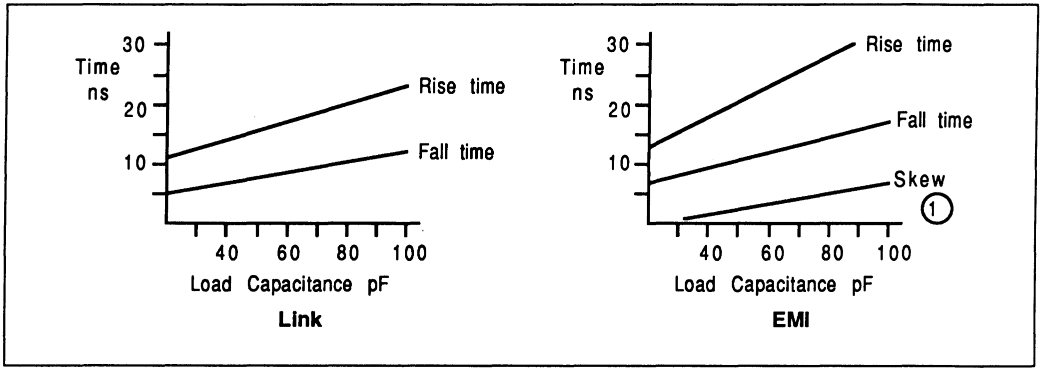


Figure 10.4 Typical rise/fall times

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.

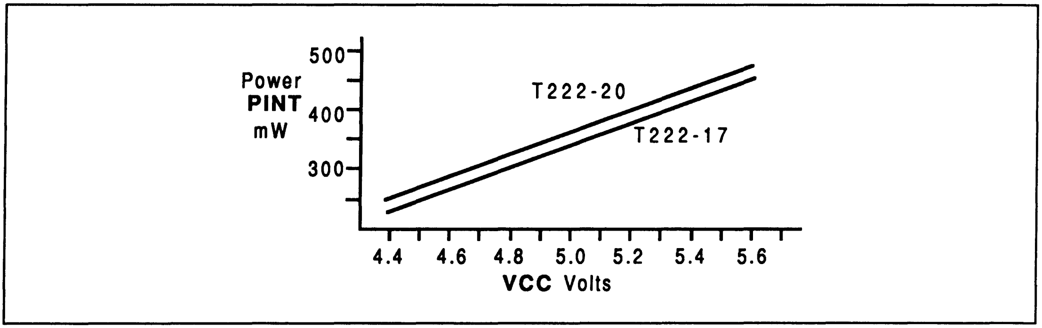


Figure 10.5 IMS T222 internal power dissipation vs VCC

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**+2**Et**	34+2**Eg**+2**Et**
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 T222 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 T222 floating point operations

Floating point operations for the IMS T222 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 T222 floating point operations performance

		Processor cycles	
		IMS T222	
		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 T222 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 T222	
	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 T222 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 T222 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

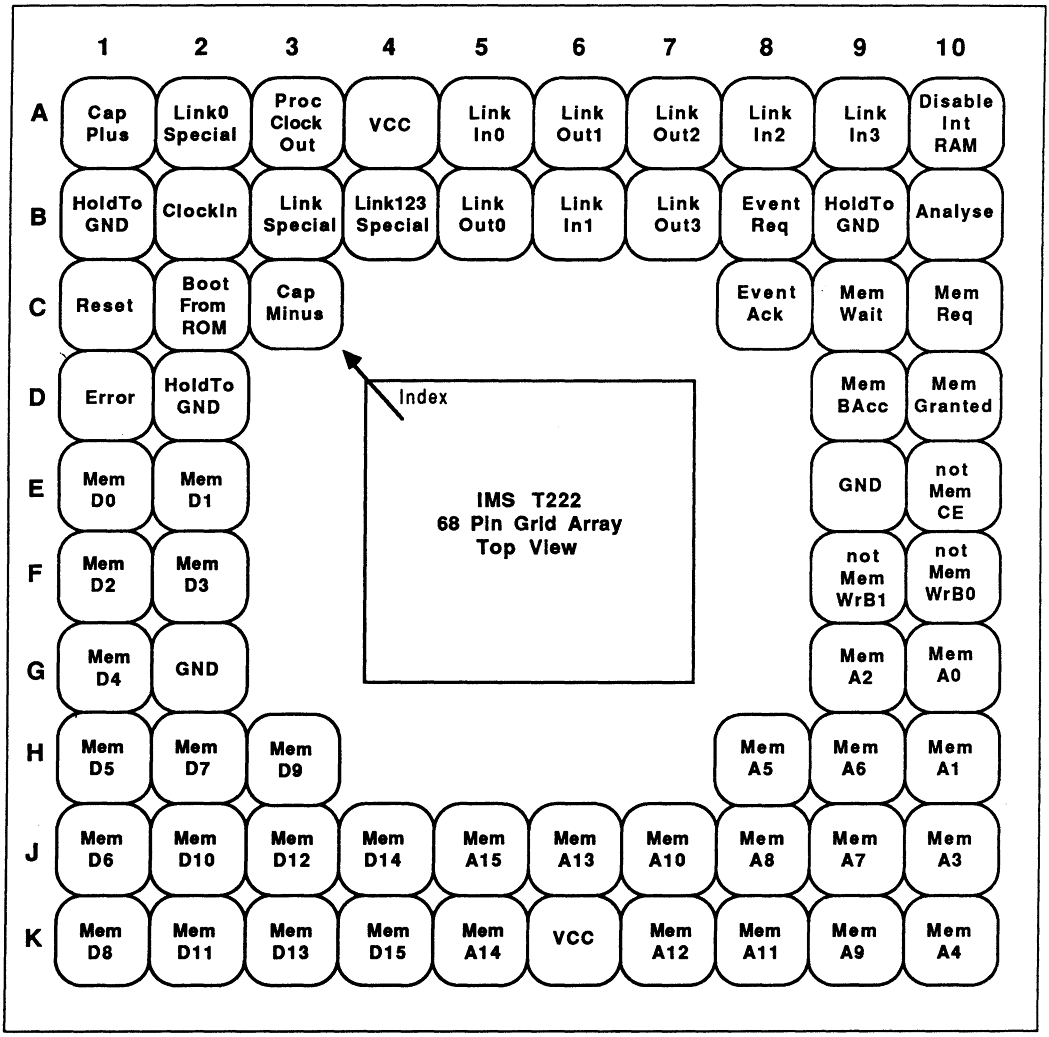
If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.8. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal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

Table 11.8 Interrupt latency

	Typical		Maximum	
	TPCLPCL	Tm	TPCLPCL	Tm
IMS T222	19		53	

12 Package specifications

12.1 68 pin grid array package



	1	2	3	4	5	6	7	8	9	10
A	Cap Plus	Link0 Special	Proc Clock Out	VCC	Link In0	Link Out1	Link Out2	Link In2	Link In3	Disable Int RAM
B	HoldTo GND	ClockIn	Link Special	Link123 Special	Link Out0	Link In1	Link Out3	Event Req	HoldTo GND	Analyse
C	Reset	Boot From ROM	Cap Minus					Event Ack	Mem Wait	Mem Req
D	Error	HoldTo GND							Mem BAcc	Mem Granted
E	Mem D0	Mem D1							GND	not Mem CE
F	Mem D2	Mem D3							not Mem WrB1	not Mem WrB0
G	Mem D4	GND							Mem A2	Mem A0
H	Mem D5	Mem D7	Mem D9					Mem A5	Mem A6	Mem A1
J	Mem D6	Mem D10	Mem D12	Mem D14	Mem A15	Mem A13	Mem A10	Mem A8	Mem A7	Mem A3
K	Mem D8	Mem D11	Mem D13	Mem D15	Mem A14	VCC	Mem A12	Mem A11	Mem A9	Mem A4

Figure 12.1 IMS T222 68 pin grid array package pinout

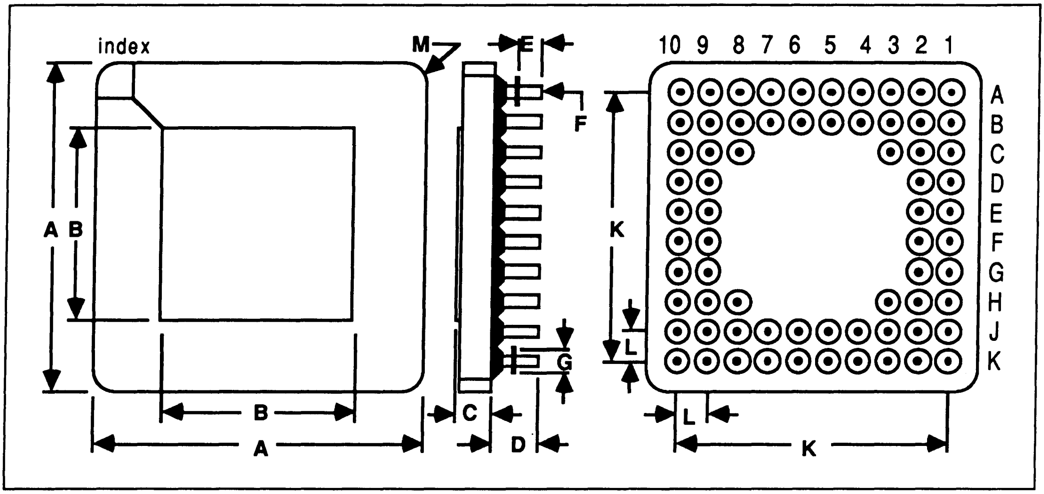


Figure 12.2 68 pin grid array package dimensions

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

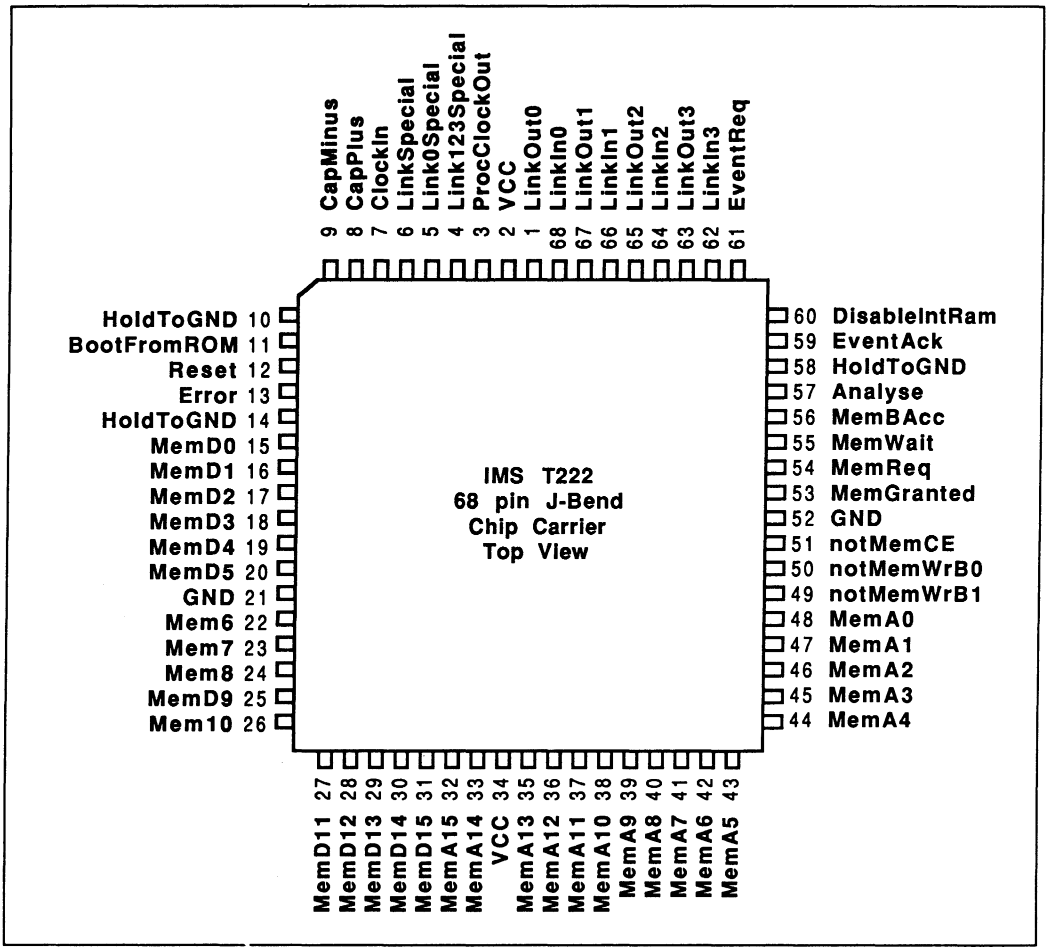


Figure 12.3 IMS T222 68 pin PLCC J-bend package pinout

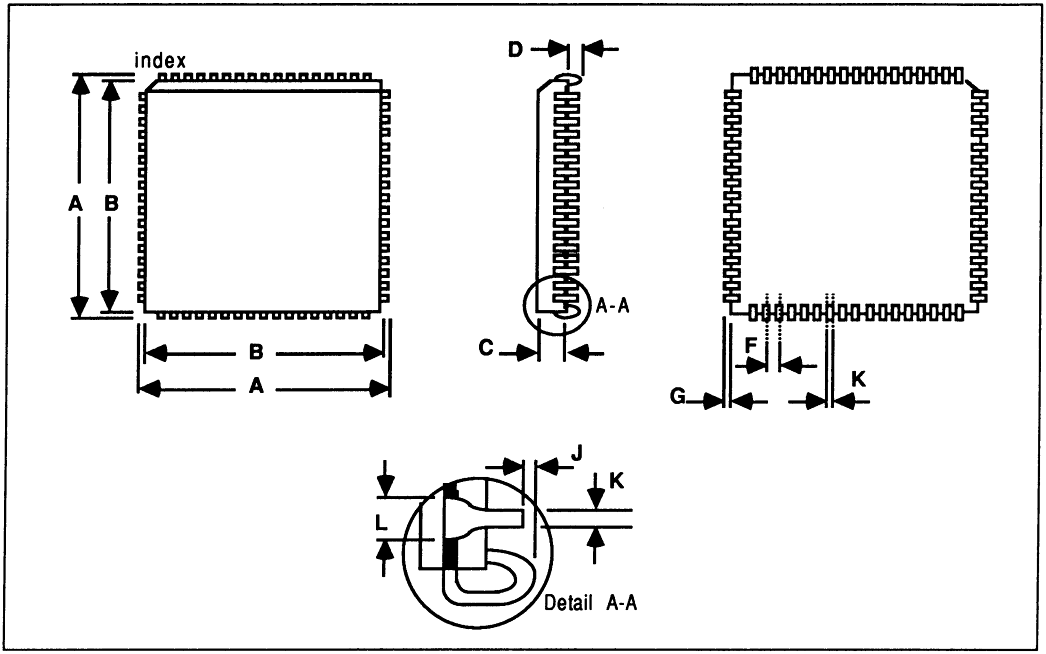


Figure 12.4 68 pin PLCC J-bend package dimensions

Table 12.3 68 pin PLCC J-bend package dimensions

DIM	Millimetres NOM	Millimetres TOL	Inches NOM	Inches TOL	Notes
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	

13 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 13.1 IMS T222 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T222-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T222-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T222-J17S	17.5 MHz	57 ns	3.5	Plastic J-Bend
IMS T222-J20S	20.0 MHz	50 ns	4.0	Plastic J-Bend
IMS T222-G17M	17.5 MHz	57 ns	3.5	Ceramic Pin Grid MIL Spec
IMS T222-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid MIL Spec

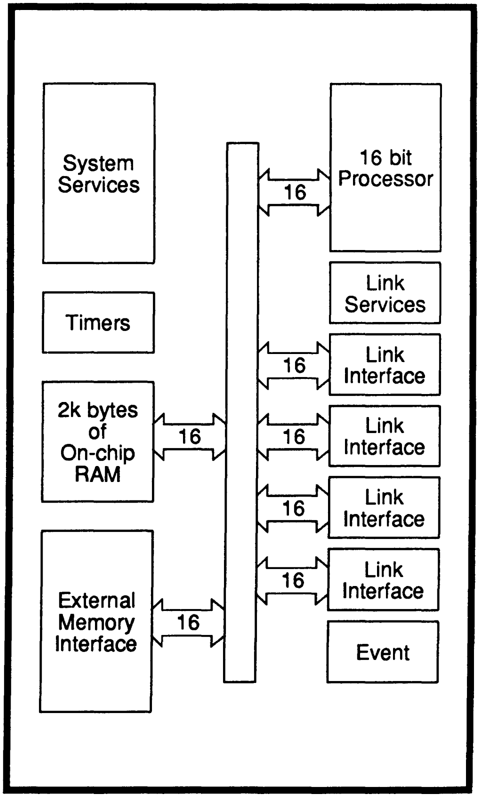
IMS T212
transputer

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.

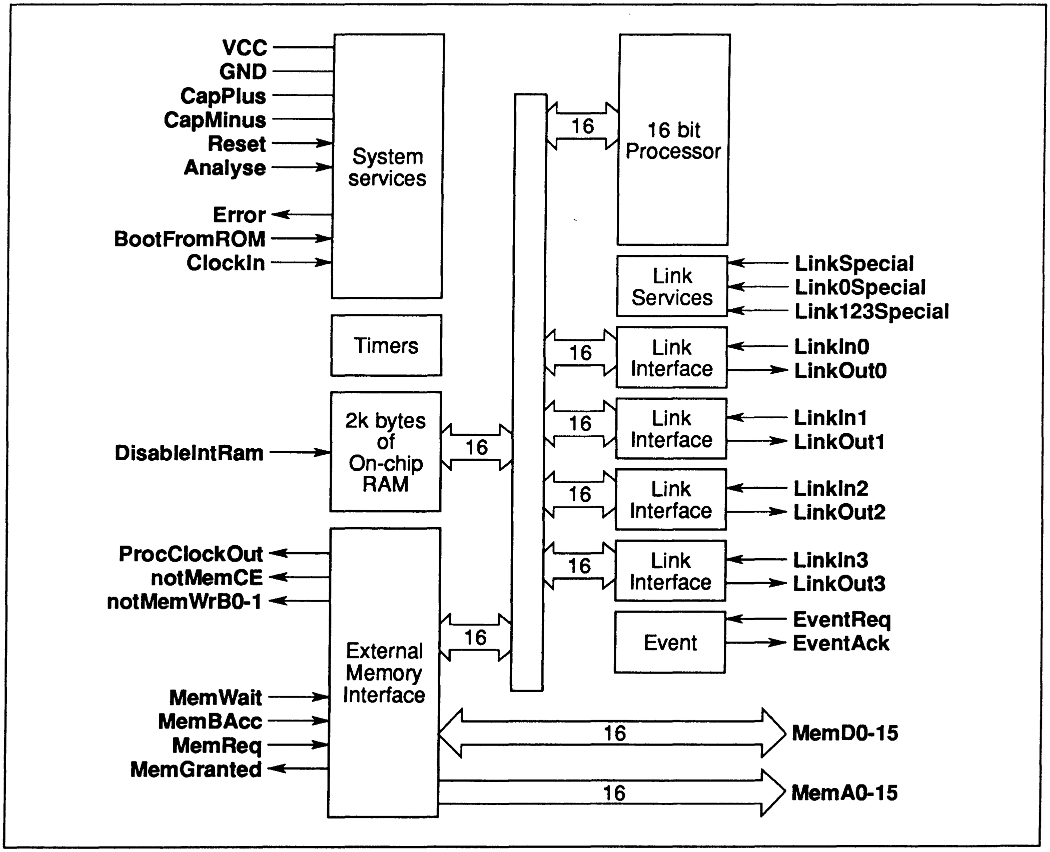


Figure 1.1 IMS T212 block diagram

The IMS T212 can directly access a linear address space of 64 Kbytes. The 16 bit wide non-multiplexed external memory interface provides a data rate of up to 2 bytes every 100 nanoseconds (20 Mbytes/sec) for a 20 MHz device. System Services include processor reset and bootstrap control, together with facilities for error analysis.

The INMOS communication links allow networks of transputers to be constructed by direct point to point connections with no external logic. The links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec.

The IMS T212 is designed to implement the occam language, detailed in the occam Reference Manual, but also efficiently supports other languages such as C and Pascal. 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 T212. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of occam.

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.

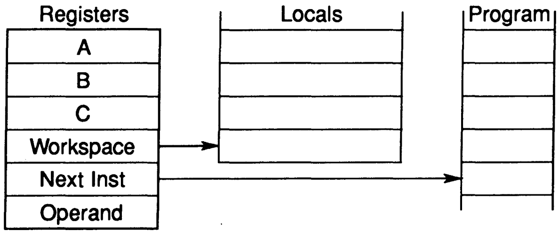


Figure 3.1 Registers

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.

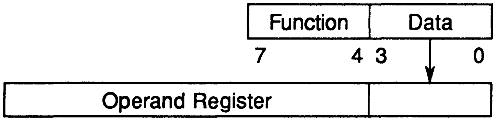


Figure 3.2 Instruction format

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.

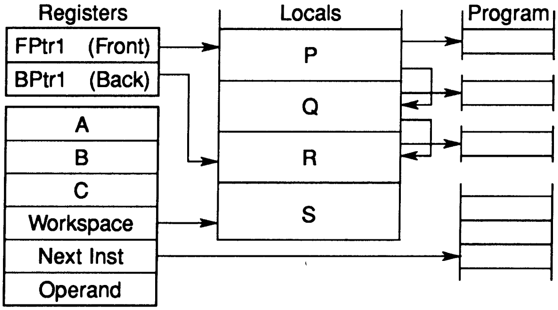


Figure 3.3 Linked process list

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.

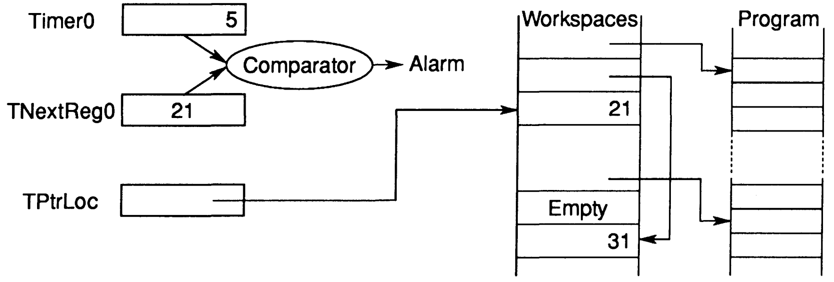


Figure 3.4 Timer registers

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.

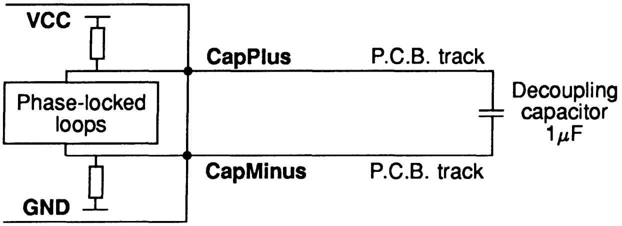


Figure 5.1 Recommended PLL decoupling

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).

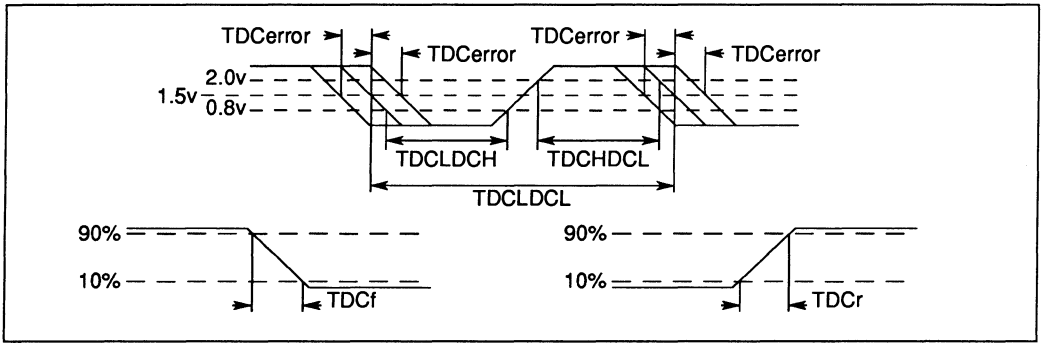


Figure 5.2 ClockIn timing

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.

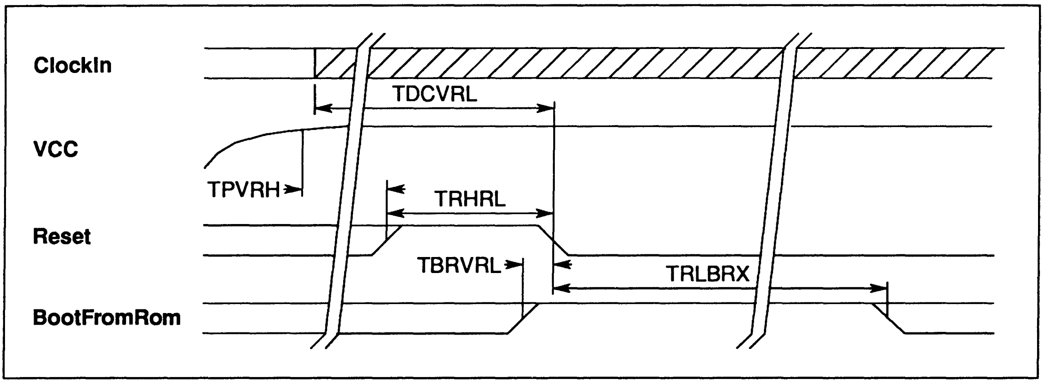


Figure 5.3 Transputer reset timing with Analyse low

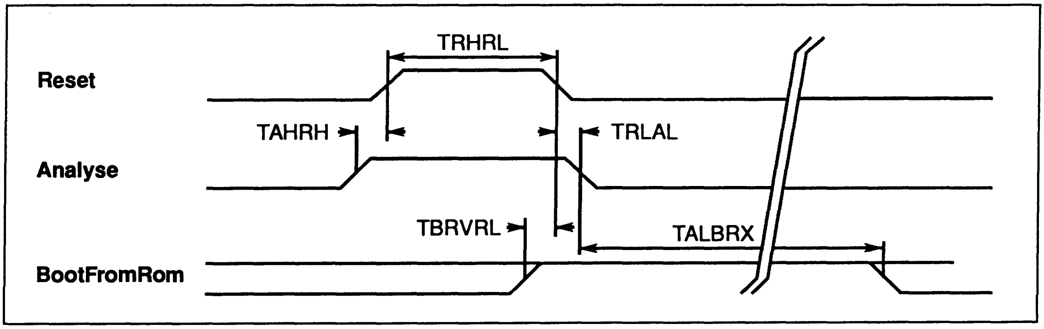


Figure 5.4 Transputer reset and analyse timing

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*.

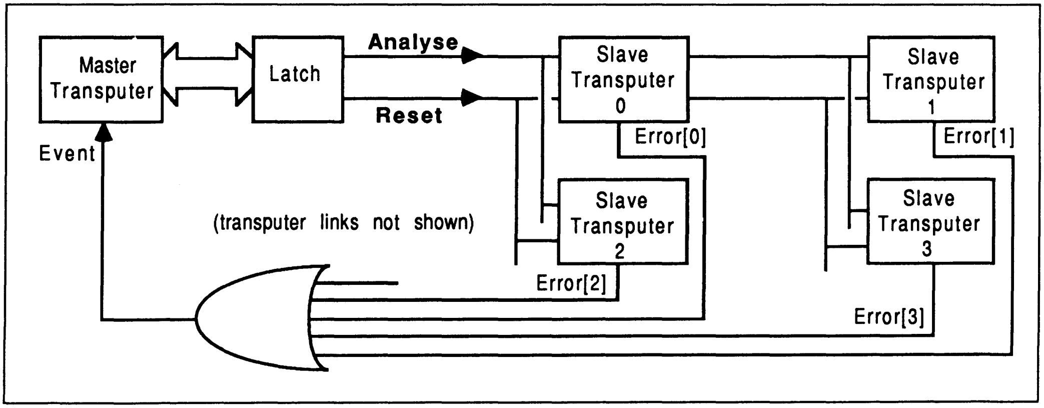


Figure 5.5 Error handling in a multi-transputer system

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.

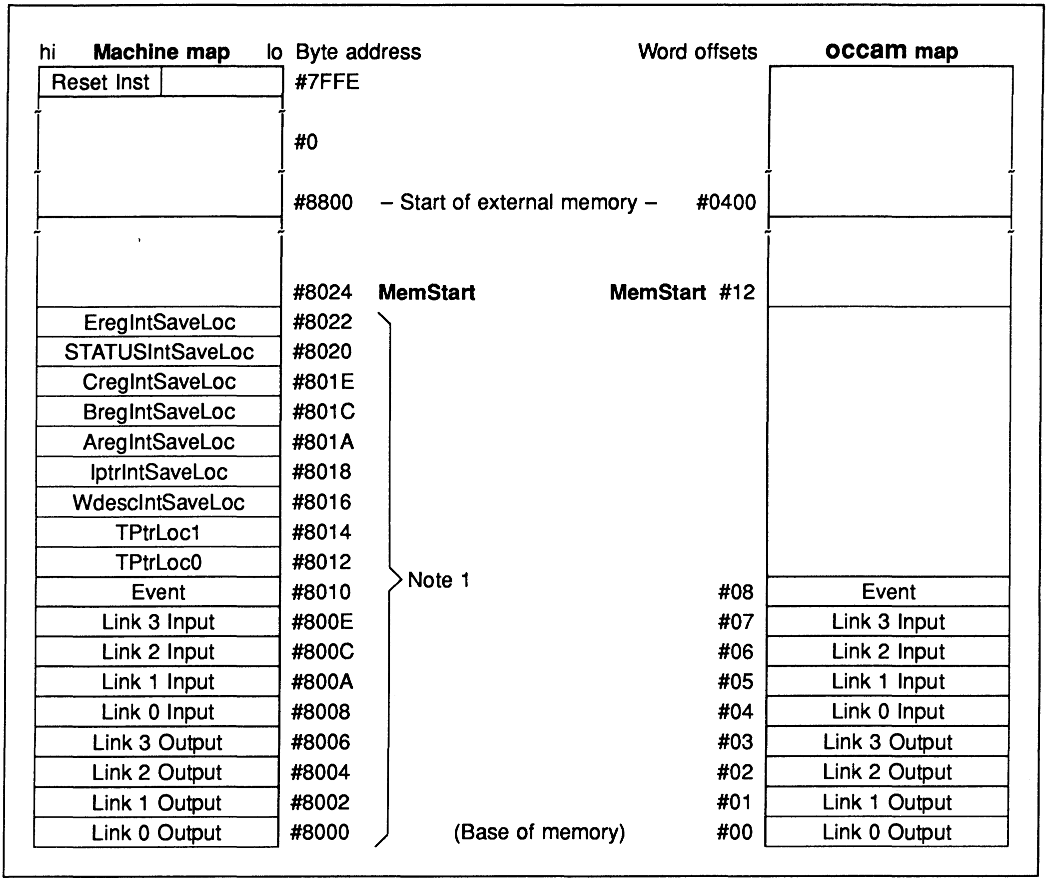


Figure 6.1 IMS T212 memory map

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.

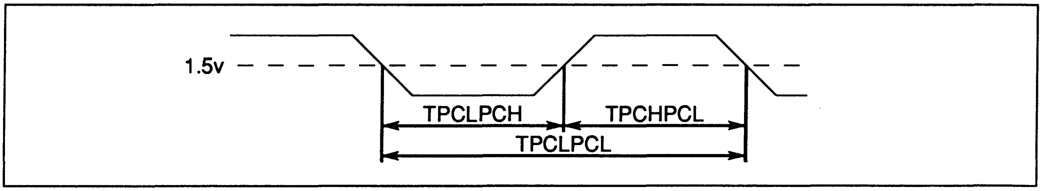


Figure 7.1 IMS T212 ProcClockOut timing

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 MIN	T212-20 MAX	T212-17 MIN	T212-17 MAX	UNITS	NOTE
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**.

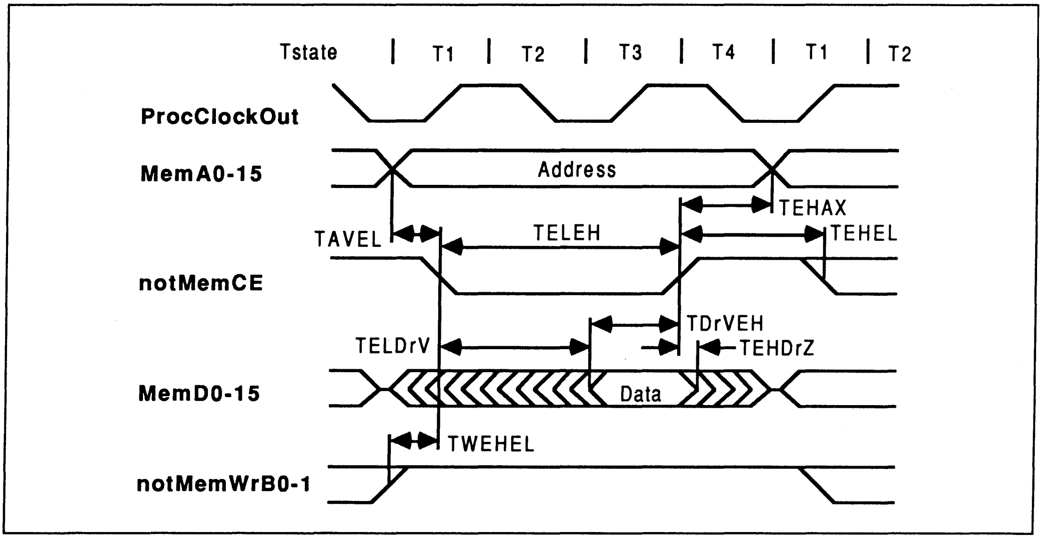


Figure 7.2 IMS T212 external read cycle

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**.

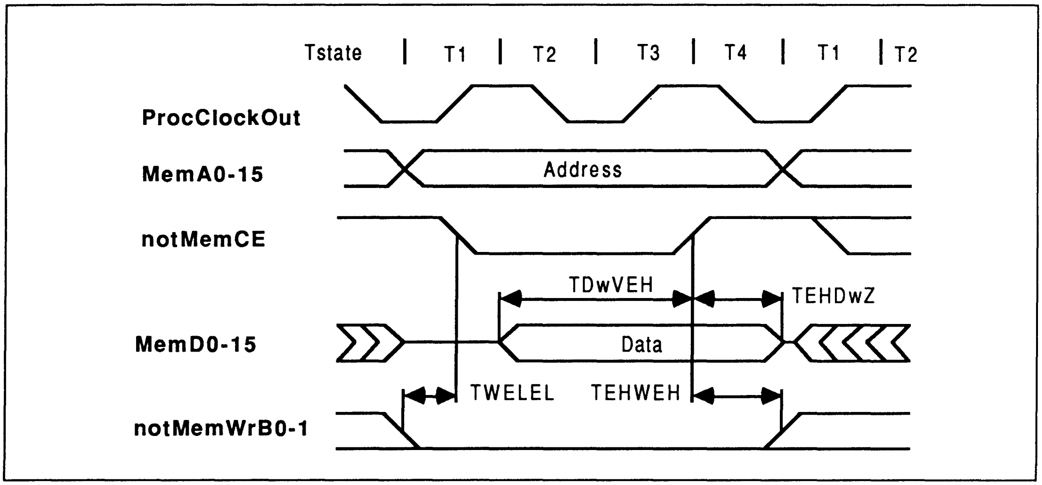


Figure 7.3 IMS T212 external write cycle

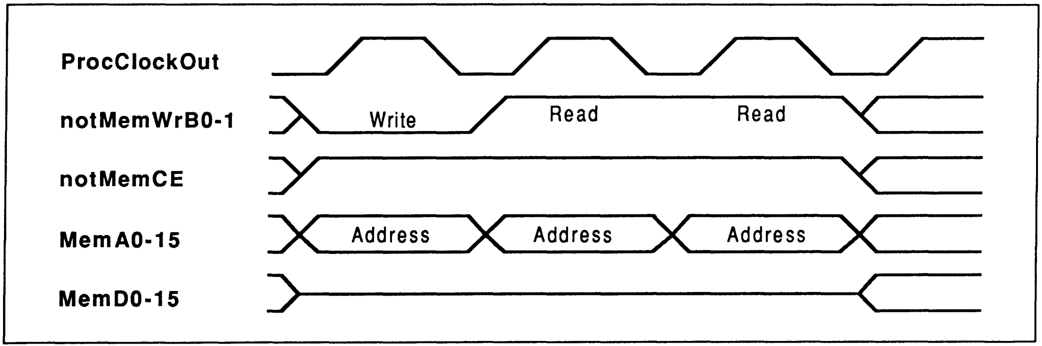


Figure 7.4 IMS T212 typical bus activity for internal memory cycles

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

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

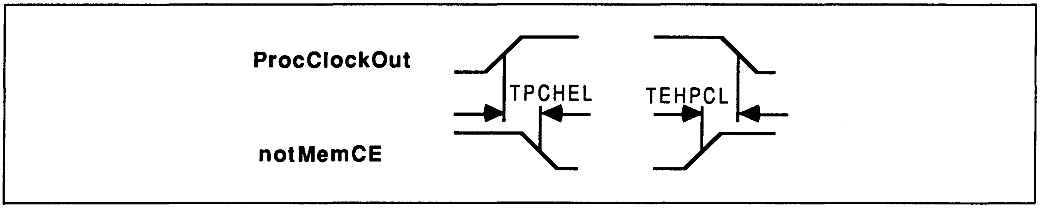


Figure 7.5 IMS T212 skew of **notMemCE** to **ProcClockOut**

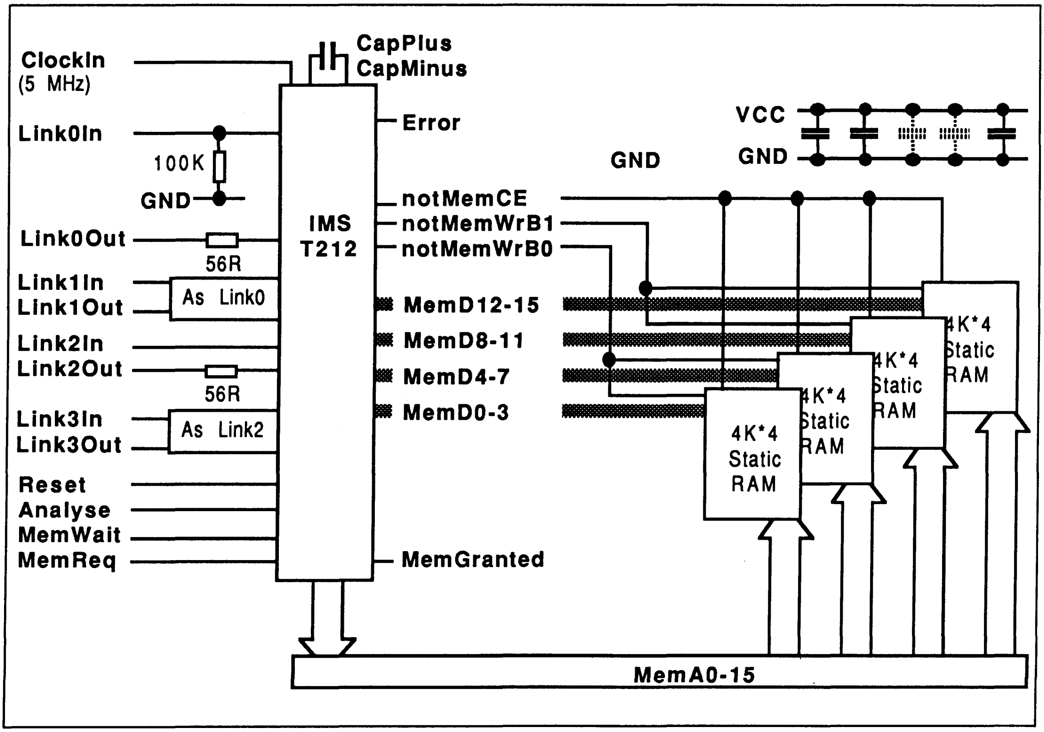


Figure 7.6 IMS T212 application

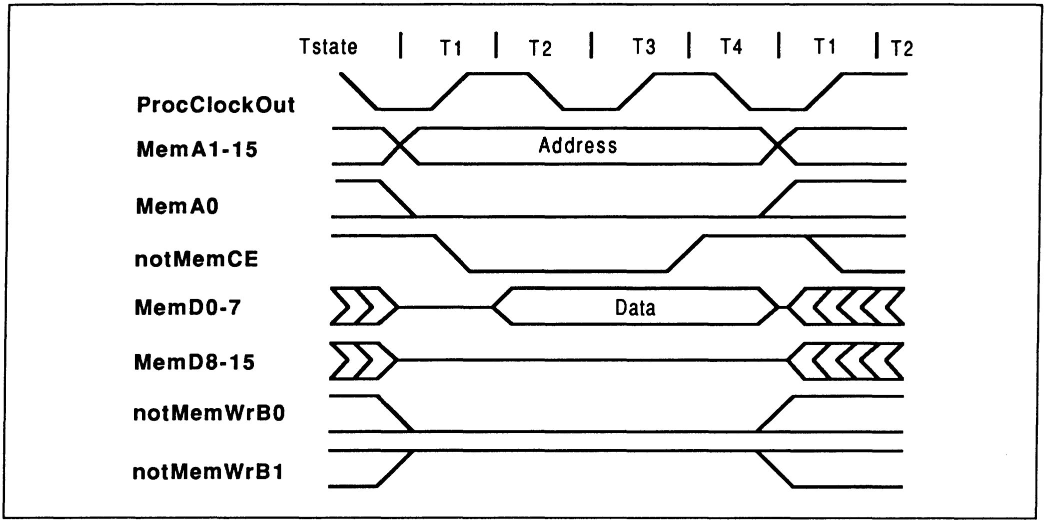


Figure 7.7 IMS T212 Least significant byte write in word access mode

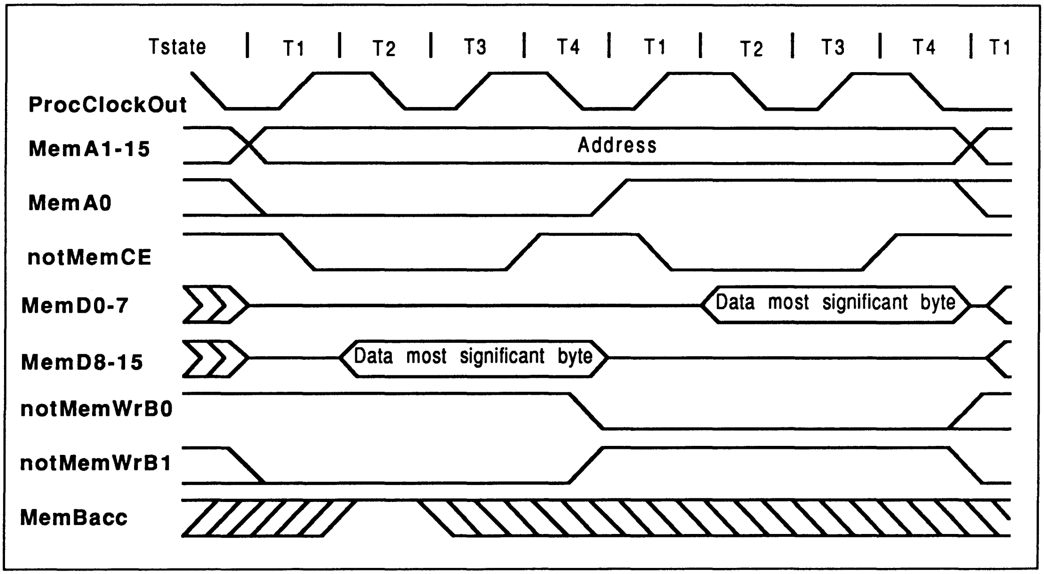


Figure 7.8 IMS T212 Most significant byte write to byte-wide memory

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	

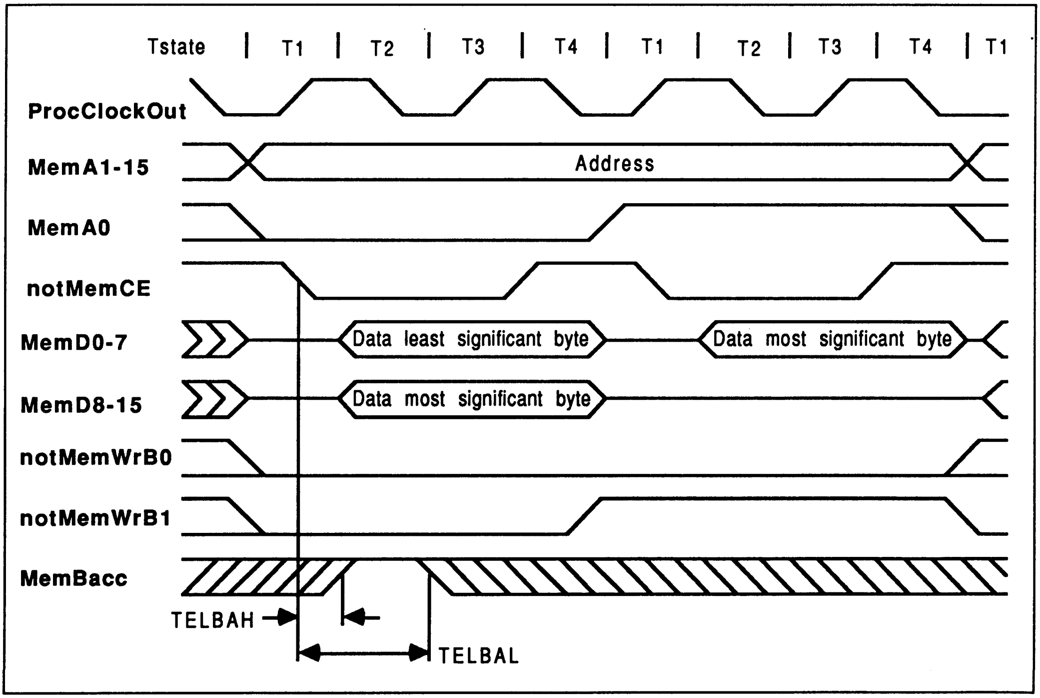


Figure 7.9 IMS T212 word write to byte-wide memory

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.

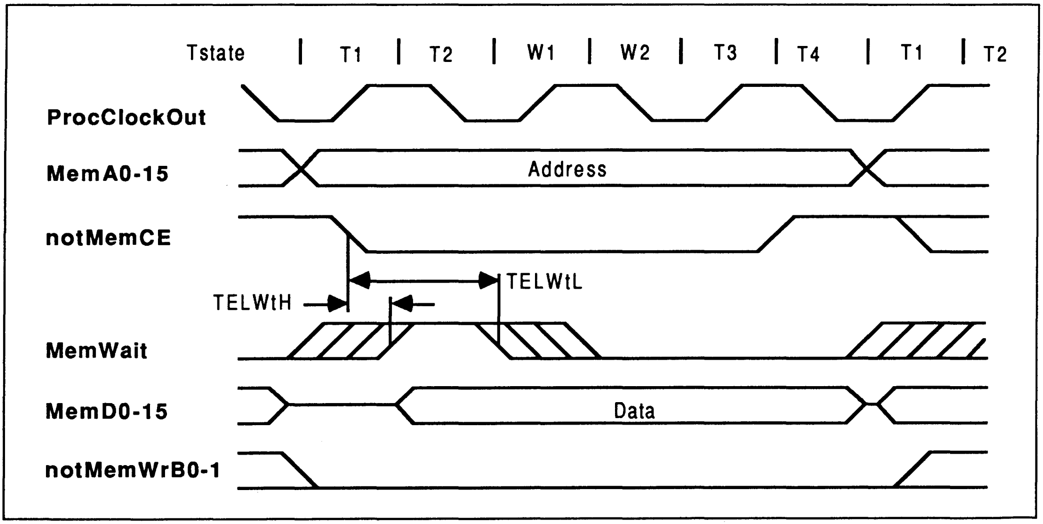


Figure 7.10 IMS T212 memory wait timing

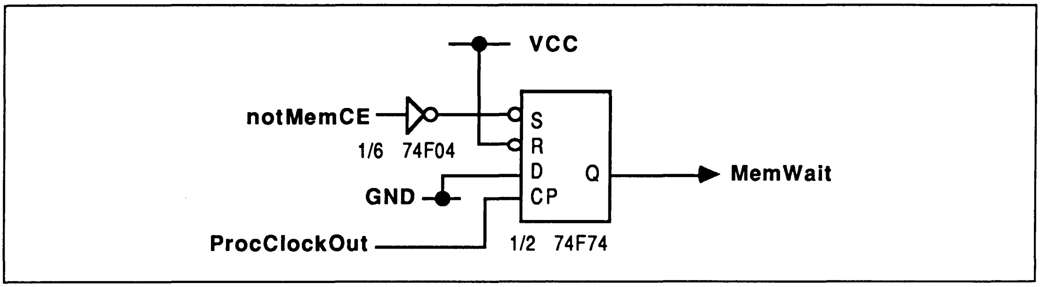


Figure 7.11 Single wait state generator

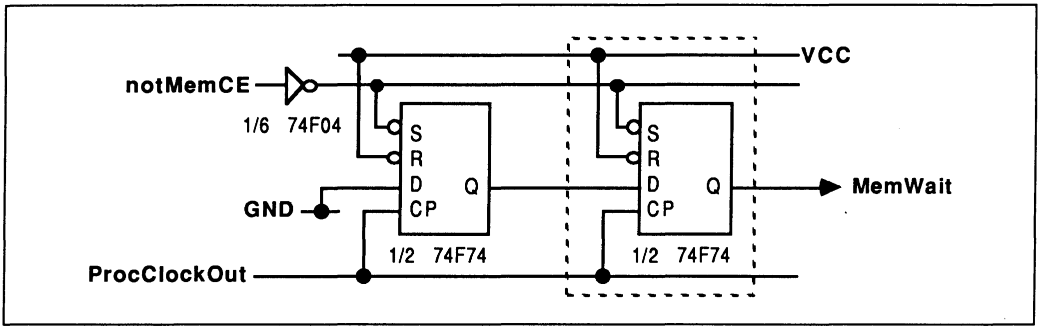


Figure 7.12 Extendable wait state generator

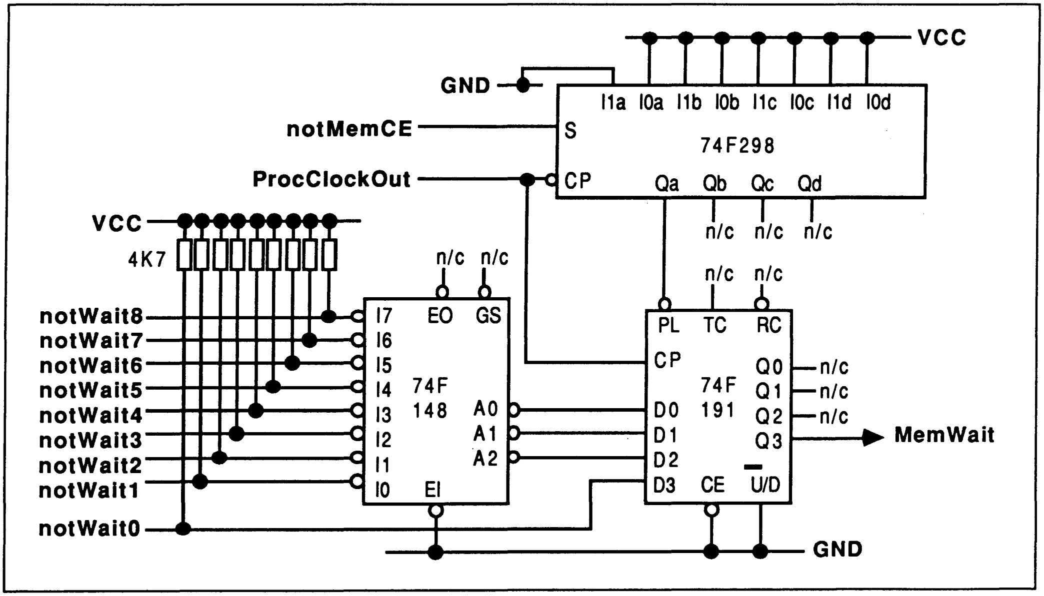


Figure 7.13 Programmable wait state generator

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.

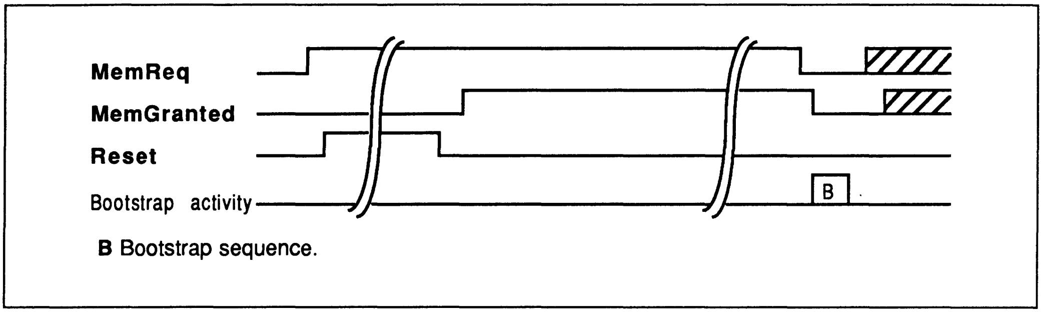


Figure 7.14 IMS T212 DMA sequence at reset

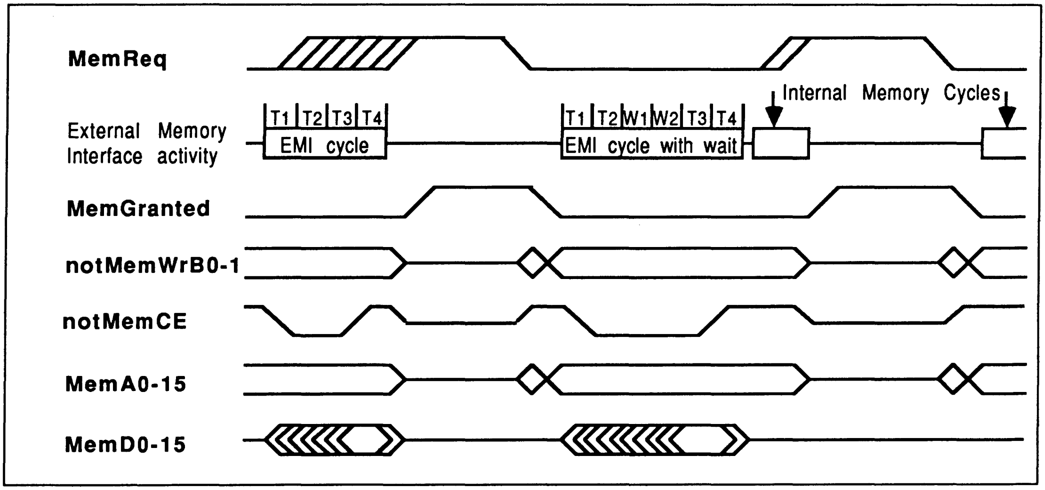


Figure 7.15 IMS T212 operation of **MemReq** and **MemGranted** with external and internal memory cycles

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

		T212-20		T212-17			
SYMBOL	**PARAMETER**	**MIN**	**MAX**	**MIN**	**MAX**	**UNITS**	**NOTE**
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.

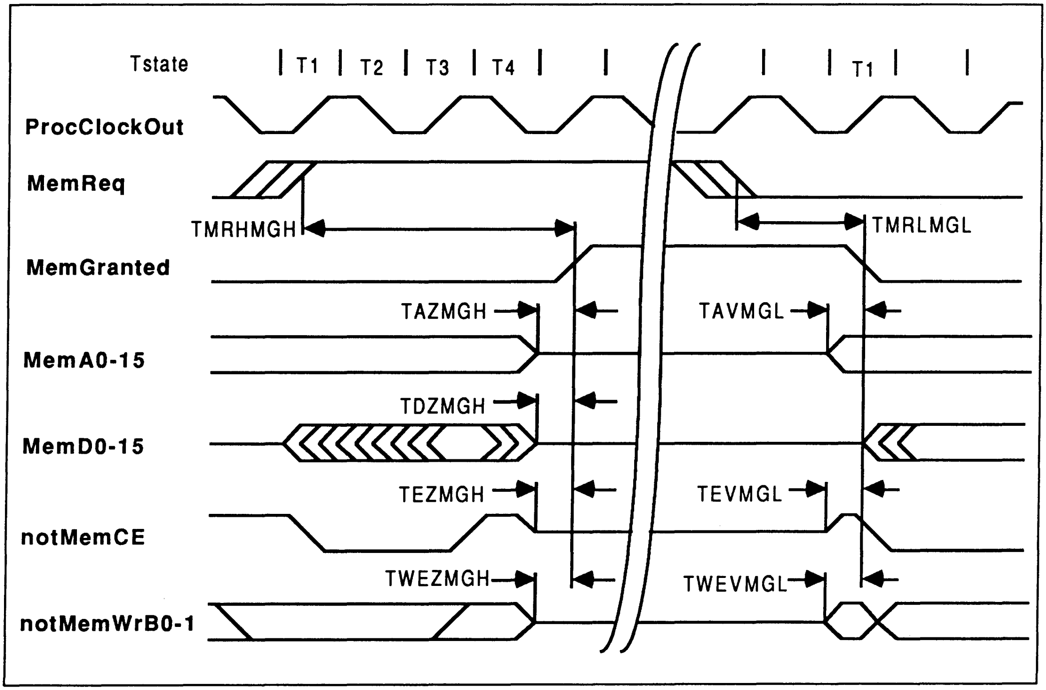


Figure 7.16 IMS T212 memory request timing

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**.

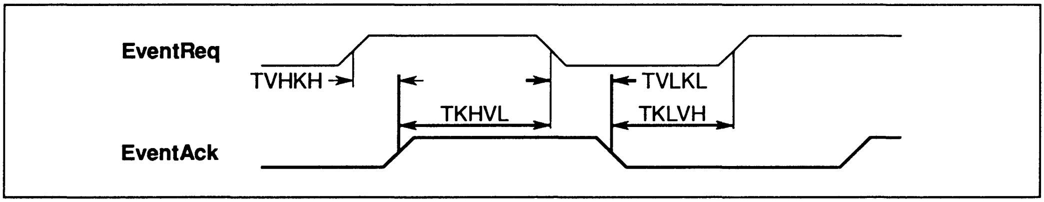


Figure 8.1 IMS T212 event timing

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

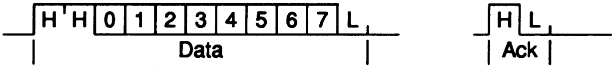


Figure 9.1 IMS T212 link data and acknowledge packets

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.

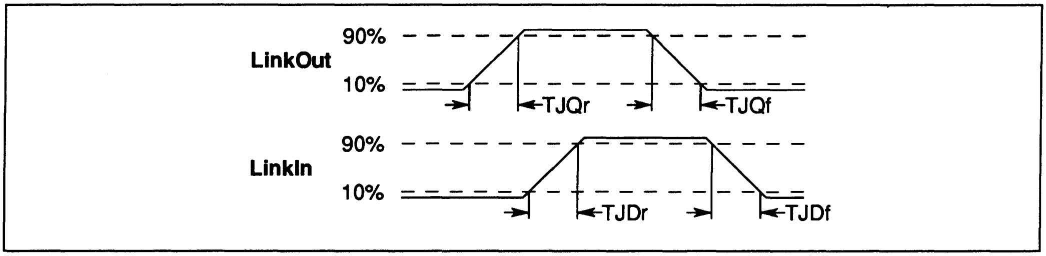


Figure 9.2 IMS T212 link timing

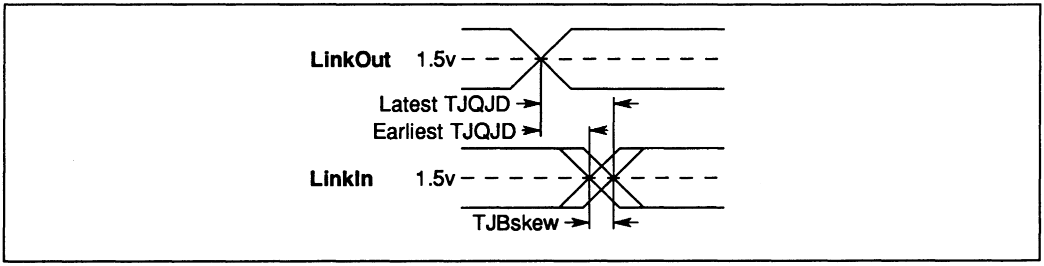


Figure 9.3 IMS T212 buffered link timing

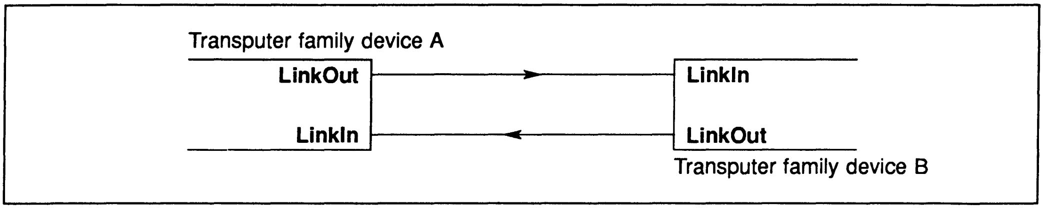


Figure 9.4 IMS T212 Links directly connected

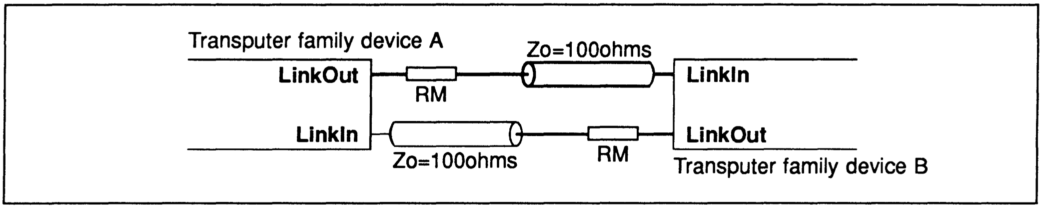


Figure 9.5 IMS T212 Links connected by transmission line

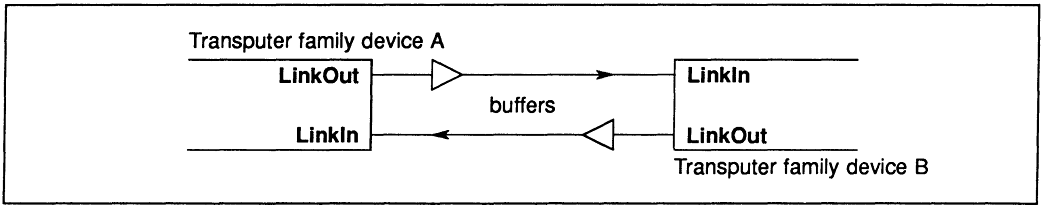


Figure 9.6 IMS T212 Links connected by buffers

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.

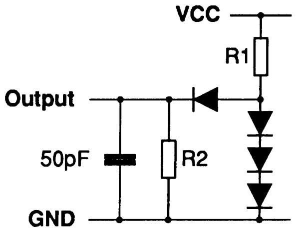
Table 10.3 DC characteristics

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

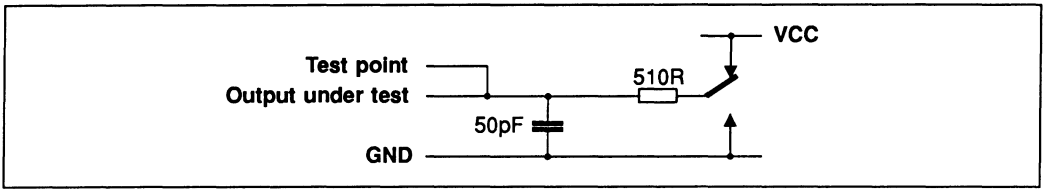


Figure 10.2 Tristate load circuit for AC measurements

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**.

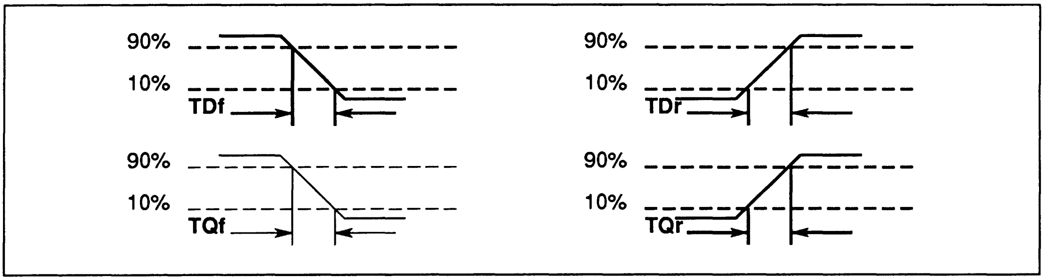


Figure 10.3 IMS T212 input and output edge timing

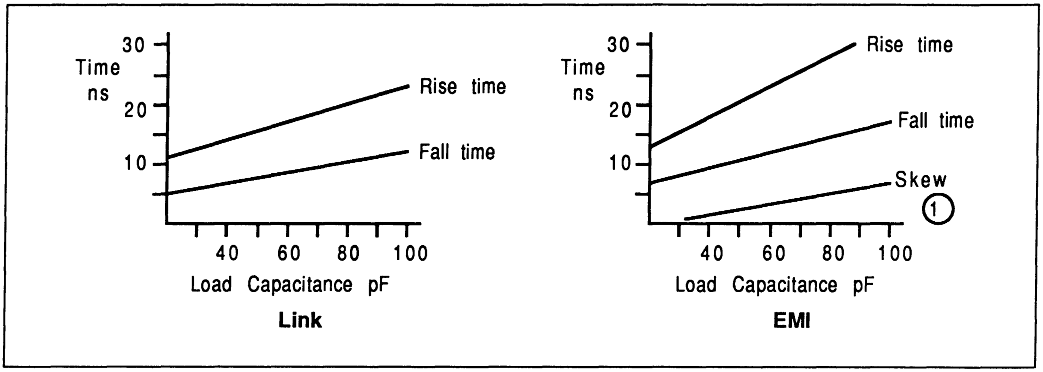


Figure 10.4 Typical rise/fall times

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.

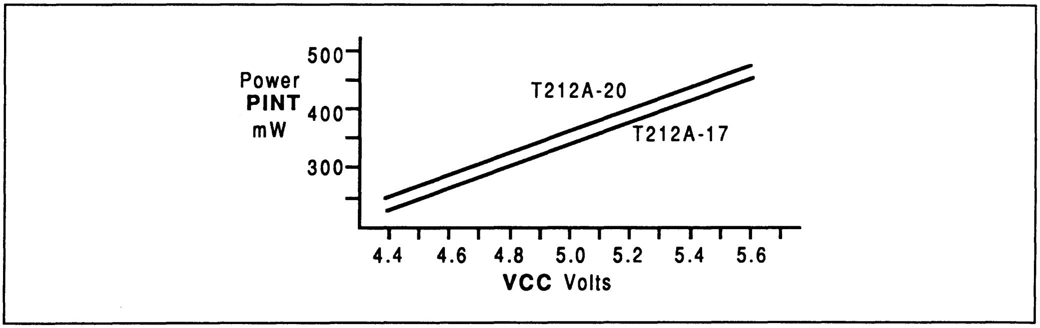


Figure 10.5 IMS T212 internal power dissipation vs VCC

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**+2**Et**	34+2**Eg**+2**Et**
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**+9**e**-12	(5e-2)**n**+6
Replicated `SEQ`	(4**e**-3)**n**+3e	(4**e**-2)**n**+3-e
`PAR`	4**en**	3**en**
Replicated `PAR`	(17e-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

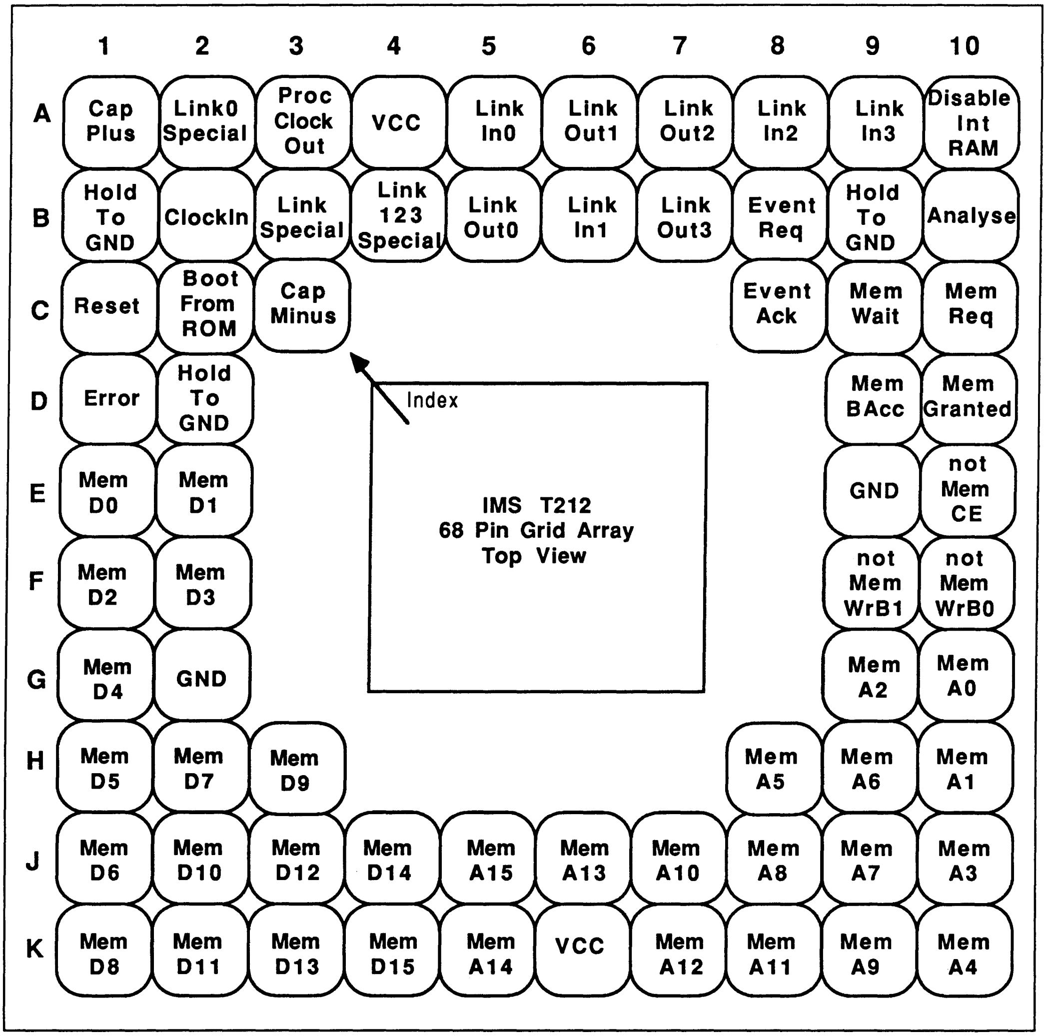
If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.8. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal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



	1	2	3	4	5	6	7	8	9	10
A	Cap Plus	Link0 Special	Proc Clock Out	VCC	Link In0	Link Out1	Link Out2	Link In2	Link In3	Disable Int RAM
B	Hold To GND	ClockIn	Link Special	Link 123 Special	Link Out0	Link In1	Link Out3	Event Req	Hold To GND	Analyse
C	Reset	Boot From ROM	Cap Minus					Event Ack	Mem Wait	Mem Req
D	Error	Hold To GND							Mem BAcc	Mem Granted
E	Mem D0	Mem D1							GND	not Mem CE
F	Mem D2	Mem D3							not Mem WrB1	not Mem WrB0
G	Mem D4	GND							Mem A2	Mem A0
H	Mem D5	Mem D7	Mem D9					Mem A5	Mem A6	Mem A1
J	Mem D6	Mem D10	Mem D12	Mem D14	Mem A15	Mem A13	Mem A10	Mem A8	Mem A7	Mem A3
K	Mem D8	Mem D11	Mem D13	Mem D15	Mem A14	VCC	Mem A12	Mem A11	Mem A9	Mem A4

Figure 12.1 IMS T212 68 pin grid array package pinout

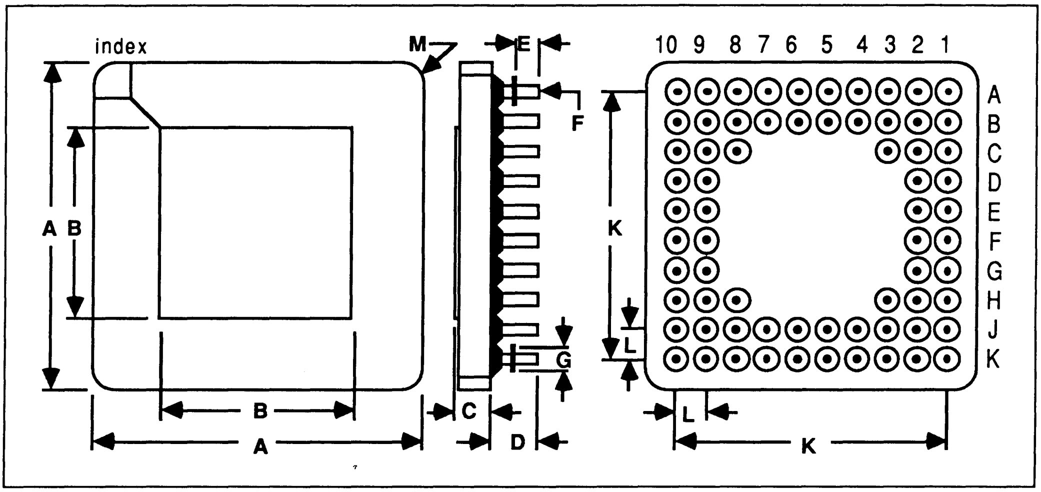


Figure 12.2 68 pin grid array package dimensions

Table 12.1 68 pin grid array package dimensions

DIM	Millimetres NOM	Millimetres TOL	Inches NOM	Inches TOL	Notes
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

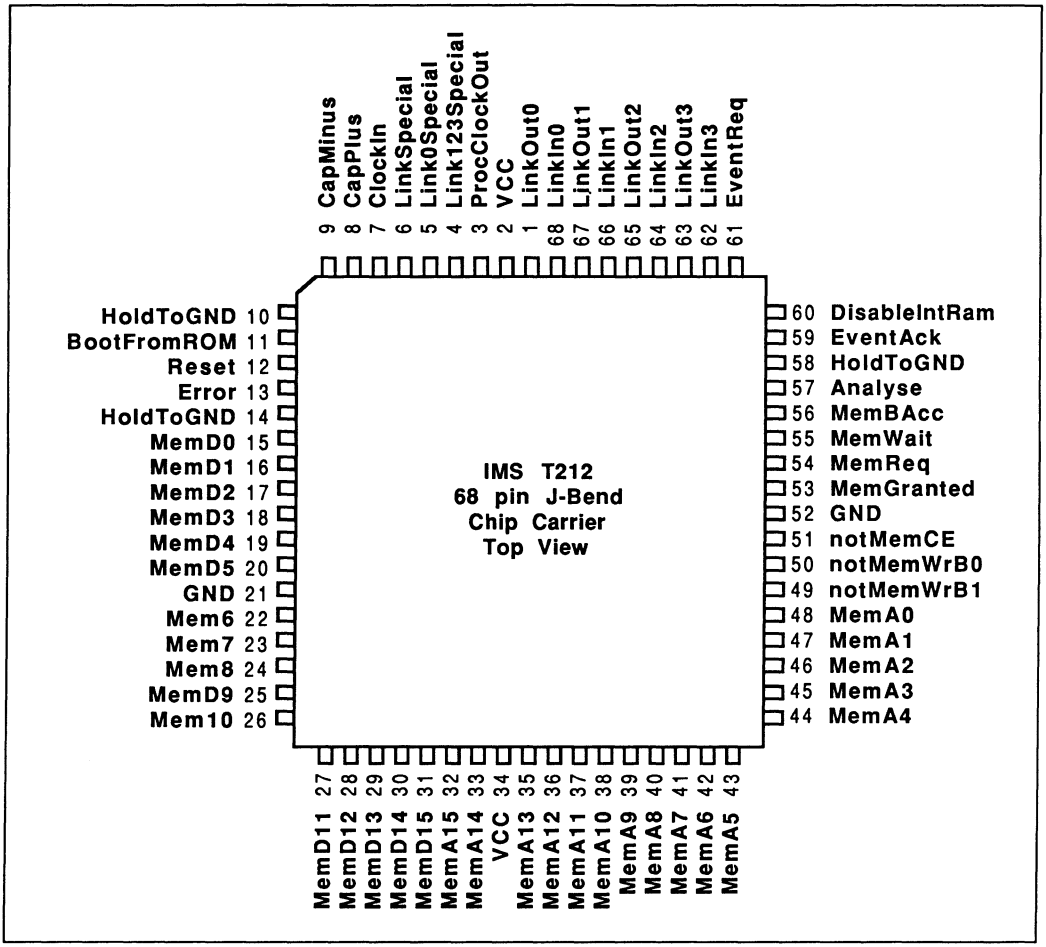


Figure 12.3 IMS T212 68 pin PLCC J-bend package pinout

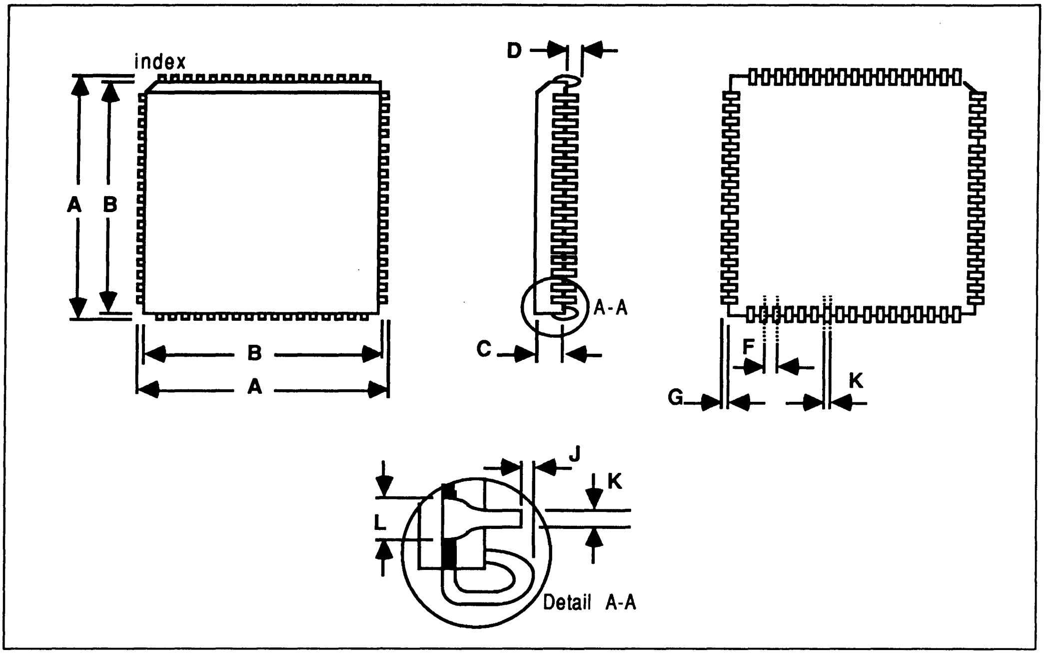


Figure 12.4 68 pin PLCC J-bend package dimensions

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	

13 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 13.1 IMS T212 ordering details

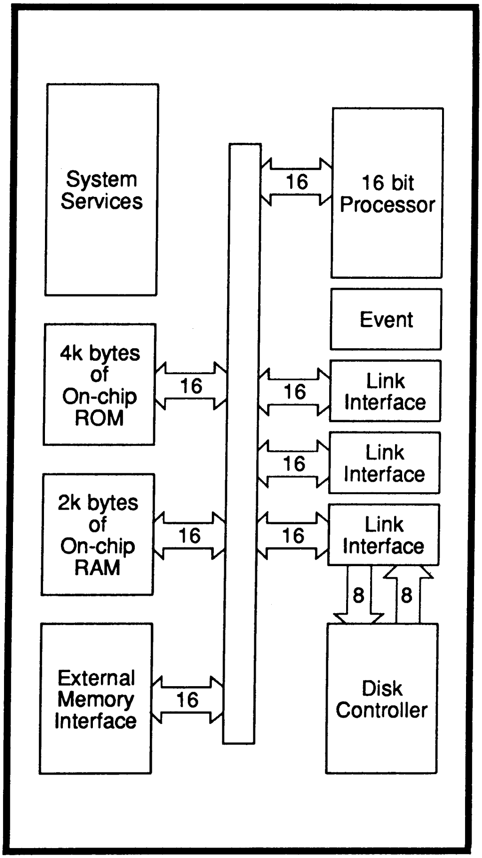
INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T212-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T212-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T212-J17S	17.5 MHz	57 ns	3.5	Plastic J-Bend
IMS T212-J20S	20.0 MHz	50 ns	4.0	Plastic J-Bend

IMS M212
disk processor

Product Preview

FEATURES

ST506/ST412, SA400/450 compatible interface
Full disk interface logic on chip
Minimum of external components required
On-chip 16 bit processor
2 Kbytes on-chip RAM
4 Kbytes on-chip ROM disk control software
External memory interface
Hardware CRC/ECC generator
Two bi-directional 8 bit data ports
Two 10/20 Mbits/sec INMOS serial links
External event interrupt
Variable wait states for slow memory
Internal timers
Support for run-time error diagnostics
Bootstraps from ROM, link or disk
Single 5 MHz processor clock input
Power dissipation less than 1 Watt



1 Introduction

The IMS M212 peripheral processor is an intelligent peripheral controller of the INMOS transputer family, configured for connection to soft sectored winchester and floppy disk drives. It satisfies the demand for increasing intelligence in peripheral controllers and maintains a high degree of flexibility, allowing designers to modify the controller function without altering the hardware.

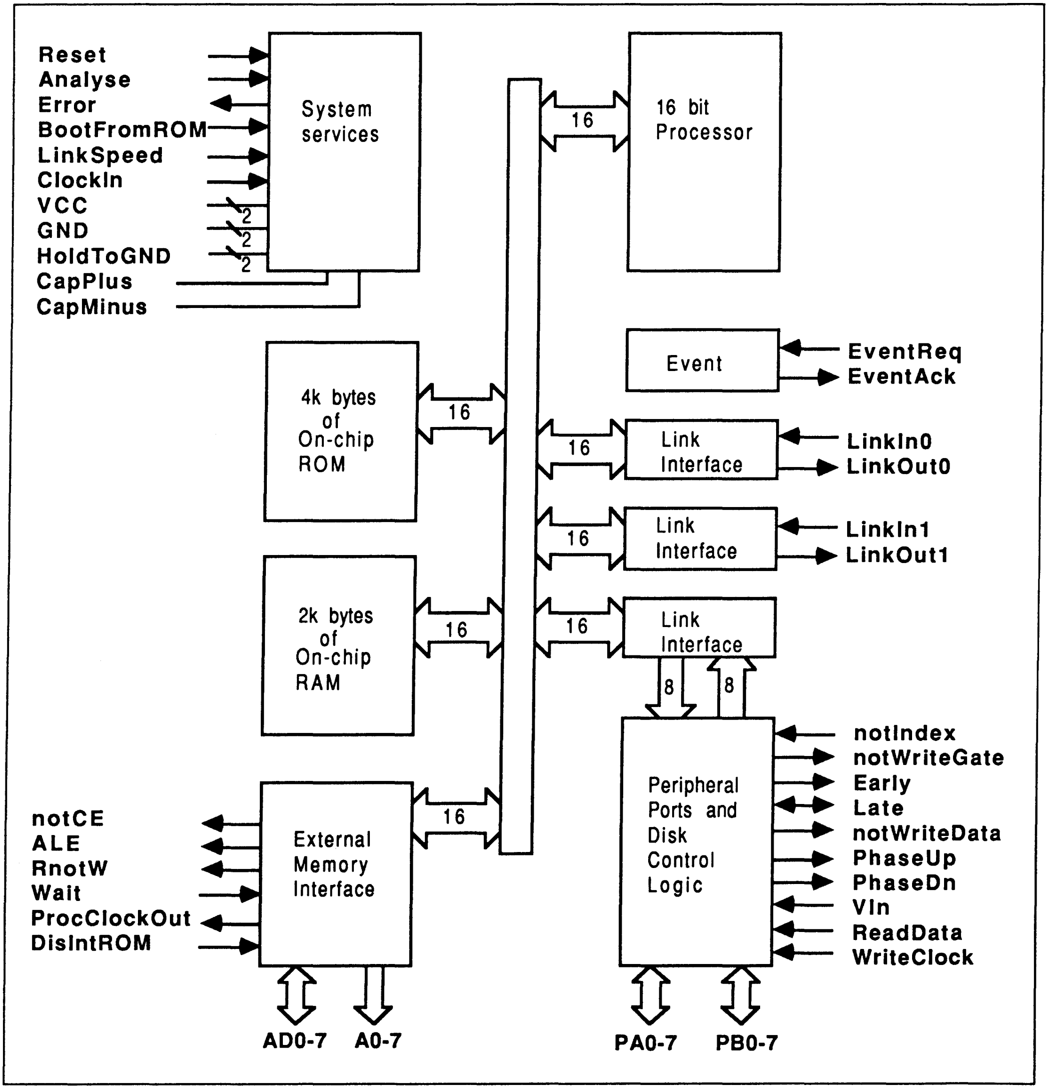


Figure 1.1 IMS M212 block diagram

The disk control function has been designed to provide easy connection, with minimal external hardware, to a standard winchester and/or floppy disk interface. Two byte-wide programmable bidirectional ports are provided to control and monitor disk functions such as head position, drive selection and disk status. A dedicated port is provided for serial data interfaces and critical timing signals.

The IMS M212 is programmed as a normal transputer, permitting extremely powerful peripheral control facilities to be built into the device and thus reducing the load on the traditional central processor of a computer. Full details are given in the IMS M212 Disk Processor Product Data manual.

1.1 IMS M212 peripheral processor

1.1.1 Central processor

At the heart of the IMS M212 is a 16 bit processor which is compatible with the transputer family. Its design achieves compact programs, efficient high level language implementation and provides direct support for the occam model of concurrency. The processor shares its time between any number of concurrent processes. A process waiting for communication or a timer does not consume any processor time. Two levels of process priority enable fast interrupt response to be achieved.

The IMS M212 has been designed so that the on-chip processor performs as many functions as possible, providing flexible operation and minimising on-chip disk-specific hardware.

1.1.2 Peripheral interface

The two 8 bit data ports **PA0-7** and **PB0-7** are controlled by the processor via a pair of channels. This allows the programmer to modify the function of these ports in order to implement a wide variety of applications.

The peripheral interface includes data output registers and TTL compatible input ports, as well as facilities for defining the direction of the pins on a bit-selectable basis. The interface contains logic to detect a change of state on the input pins and to store this change for interrogation by the program.

In addition to this, the external memory interface can support memory mapped peripherals on its byte-wide data bus. An event pin is also provided, so that peripherals can request attention.

1.1.3 Disk controller

The disk interface provides a simple interconnection to ST506/ST412 and SA400/SA450 compatible disk drives via ten dedicated disk control lines and the two general purpose 8 bit bidirectional data ports **PA0-7** and **PB0-7**. Although the on-chip disk control hardware handles much of the specialised data conversion, as many disk operations as possible are controlled by the processor, using sequences of control and data information.

The processor can program and interrogate all the registers controlling the disk functions and data ports, and thereby control the external interface lines. As a result of this versatility, the IMS M212 can also be used in applications other than disk control ones.

A versatile hardware 32 bit Error Correcting Codes (ECC) and 16 bit Cyclic Redundancy Codes (CRC) generator is included to check data integrity. ECC's allow certain classes of errors to be corrected as well as detected, whilst CRC's only allow detection.

When writing data to the disk the hardware serialises the data and encodes it into a Frequency Modulated (FM) or Modified Frequency Modulated (MFM) data stream. Any necessary precompensation is performed internally before outputting the data together with the necessary control signals. Any necessary modification of the data, for instance writing the Address Marks (AM) or inserting the CRC/ECC bytes, is automatically performed by the hardware.

When reading data from disk the raw read data is input and the function known as data separation is performed internally. The hardware examines the data stream for an Address Mark to achieve byte synchronisation and then searches for the desired sector information. When the required data is located it is decoded and a serial to parallel conversion is performed before the data is transferred to the processor.

1.1.4 Links

The IMS M212 uses a DMA block transfer mechanism to transfer messages between memory and another transputer product via the INMOS links. The link interfaces and the processor all operate concurrently, allowing processing to continue while data is being transferred on all of the links.

The host interface of the IMS M212 is via two INMOS standard links, providing simple connection to any transputer based system or, via a link adaptor, to a conventional microprocessor system. Link speeds of 10 Mbits/sec and 20 Mbits/sec are available, making the device compatible with all other INMOS transputer products.

The on-chip disk control logic is controlled by the processor, using simple command sequences, via two channels which appear to the processor as a normal pair of hardware channels.

1.1.5 Memory system

The 2 Kbytes of on-chip static RAM can be used for program or data storage, as a sector buffer or to store parameter and format information. It can be extended off chip, via the external memory interface, to provide a total of 64 Kbytes. Internal and external memory appear as a single contiguous address space.

Software contained in 4 Kbytes of internal ROM enables the IMS M212 to be used as a stand alone disk processor. The ROM can be disabled to free the address space for external memory.

1.1.6 Error handling

High level language execution is made secure with array bounds checking, arithmetic overflow detection etc. A flag is set when an error is detected, and the error can be handled internally by software or externally by sensing the error pin. System state is preserved for subsequent analysis.

2 Operation

The IMS M212 can be used in two modes: Mode 1, which uses the software in the internal ROM, and Mode 2, which relies upon custom designed software.

2.1 Mode 1

Mode 1 operation uses code in the on-chip ROM to control the disk controller hardware, and little knowledge of the hardware is required to implement winchester and floppy disk drivers. The programming interface to all drive types is identical, and there is sufficient flexibility to allow a wide variety of formats and drive types to be used.

Both ST506/412 compatible winchester and SA400/450 compatible floppy drives are supported in standard double density formats; this includes common 5.25 and 3.5 inch drives. Up to 4096 cylinders are allowed. Floppy drives can have up to 8 heads and winchesters up to 16 heads. There can be between 1 and 256 sectors per track, with sector sizes of 128 to 16384 bytes in powers of 2. Drives with or without 'seek complete' and 'ready' lines are supported, and step rates can be from 64 μs to 16 ms. A range of non-standard formats can also be set up for user-specific requirements.

As with transputers, the IMS M212 can be bootstrapped from ROM or via a link. In addition, the Mode 1 monitor process also provides a facility whereby the disk processor can bootstrap itself with code read from a disk; this code runs instead of the Mode 1 process. Another option sends a standard bootstrap message, read from a disk, out of link 0; the Mode 1 process then continues as normal. It is also possible in Mode 1 to send a command, at any time, to bootstrap from code in the sector buffer.

General workspace for Mode 1 is contained in on-chip RAM, which also provides 1280 bytes of sector buffer. Contiguous external RAM immediately past the internal RAM will automatically be used to extend the size of the sector buffer. As many sectors as will fit into the sector buffer can be stored in it at the same time.

In Mode 1 a separate data area, in on-chip RAM, contains all the required control information (parameters) for each of the four possible drives. Parameters may be read from or written to via the links, and contain such information as the capacity of the disk, current position of the heads, desired sector for reading or writing, drive type, timing details etc.

Command and data bytes are accepted down either of the IMS M212 links; an interlock system prevents conflict between commands received on both links simultaneously. Any results are returned on the link which received the command. Available commands are

EndOfSequence
ReadBuffer
Restore
PollDrives
Initialise
WriteBuffer
Seek
FormatTrack
ReadParameter
ReadSector
SelectHead
Boot
WriteParameter
WriteSector
SelectDrive

Disk access commands implicitly select the drive, perform a seek and select the head. If an ECC or CRC error is found when reading a sector, a programmable number of automatic retries are performed and a subsequent correction attempted if possible. Mode 1 supports two of the four IMS M212 ECC/CRC modes - ECC and CRC. Either CRC or ECC can be specified in either of the ID or Data fields, making it possible to have floppies with correctable Data fields.

All appropriate parameters are checked to ensure that, for example, an attempt is not made to access a non-existent sector, relieving the host processor of such checking. Another feature which reduces the load on the host processor is the logical sector mode, in which all the sectors are specified as a single linear address space rather than physical cylinder/head/sector.

The logical address can also be auto-incremented if desired, as can the sector buffer. This allows a number of consecutive sectors to be read from or written to the disk with little overhead. As a *sticky status* checking technique is used, the status only has to be checked once at the end of a stream of commands; if an error occurred then reading and writing is inhibited, so that the logical address can be inspected to find where the error occurred.

2.2 Mode 2

In Mode 2 operation the internal ROM is bypassed, allowing the device to utilise user-defined software. This software can be held in external ROM, bootstrapped from a floppy or winchester disk, or loaded from the host processor via a link into internal or external RAM.

In this mode the user services the disk control hardware via a pair of on-chip high bandwidth channels. Using these channels the processor has access to the 49 registers which control the operation of the disk controller. Sequences of control codes and data bytes are sent by the processor to the disk controller logic via one of the hardware channels and data returned to the IMS M212 processor via the other. Each control code is a single byte, and may be followed by one or more data bytes.

In Mode 2 the designer can define new commands which are more complex than otherwise available. Examples include a *Format Disk* command as an extension to the *Format Track*; an application-specific directory structure; a software interface to optimise a particular file structure. Mode 2 also allows the user to optimise data transfer; thus, data could be read from a disk with no interleave, or data transfers could be re-ordered to minimise head movement. Disk searches can be arranged such that data transfer back to the host is minimised, as data comparisons can be performed by the on-chip processor.

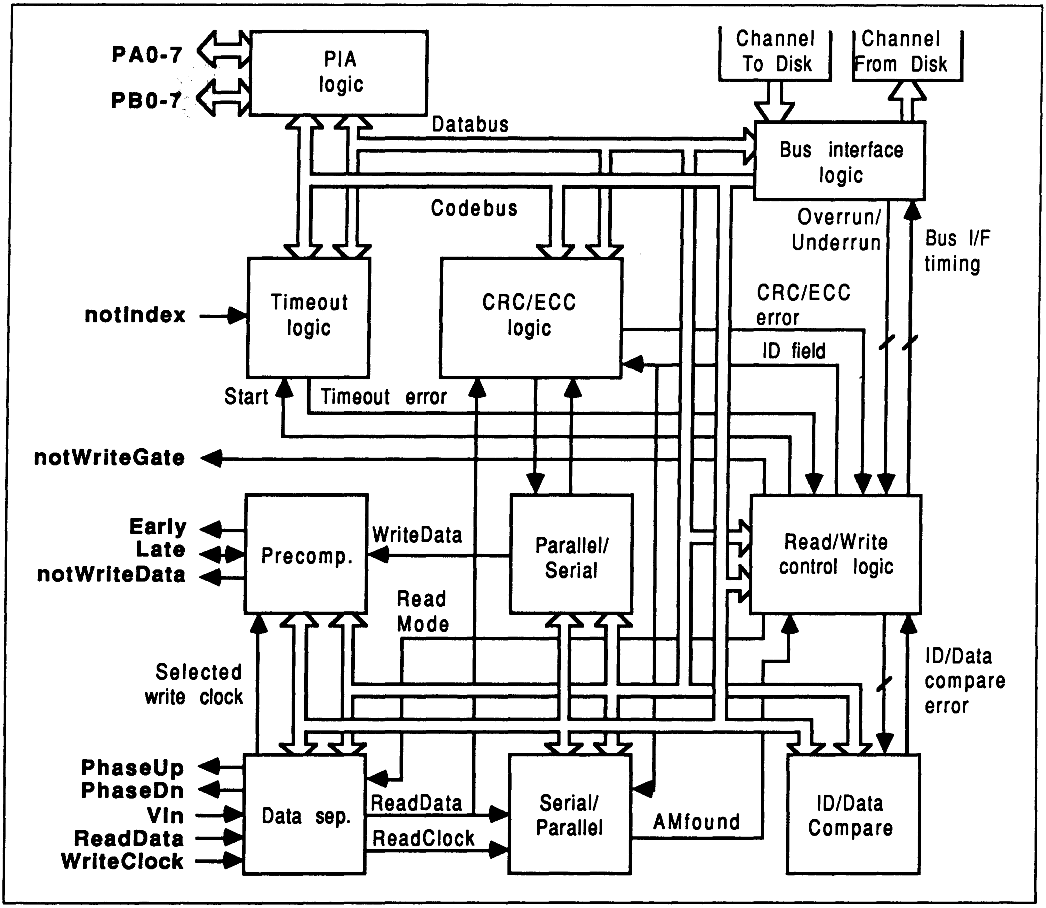


Figure 2.1 Disk controller interface

3 Applications

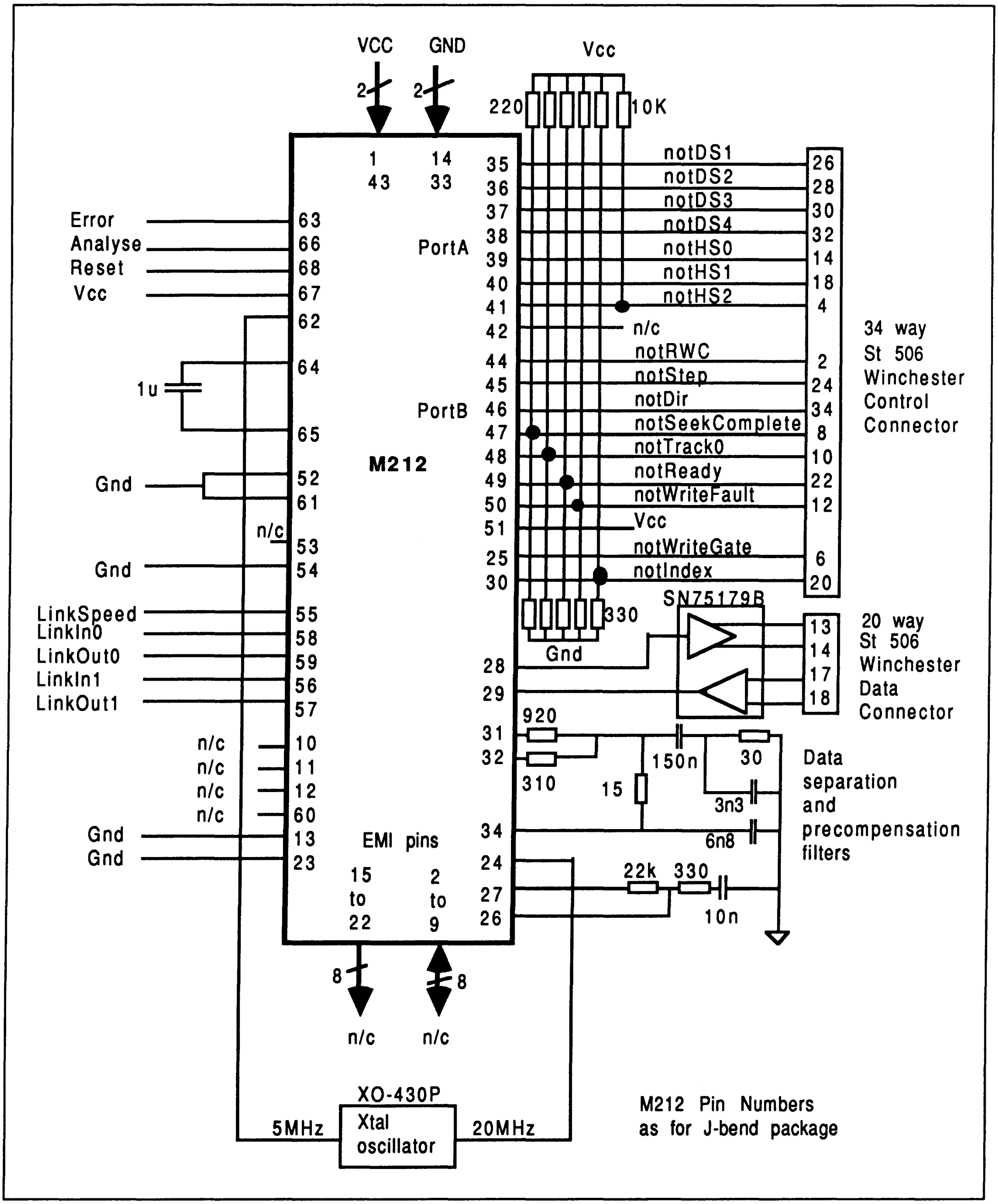


Figure 3.1 Winchester disk controller

The IMS M212 can interface to a floppy or winchester disk with very little external circuitry when used in Mode 1 or if a program is bootstrapped from a link. A typical arrangement is shown in figure 3.1. Note the absence of any control port buffers; this is possible provided the drive characteristics are not infringed.

Additional external memory can easily be added to the IMS M212. In both Modes 1 and 2, external RAM can be added for extra sector storage, whilst in Mode 2 extra RAM or ROM can be provided for program storage.

With the addition of control buffers and suitable clocks, a single IMS M212 can interface to both floppy and winchester drives. Link adaptors provide a means of interfacing to conventional microprocessors.

The IMS B005 evaluation board is an example of an application with control for both types of drive. The board also has a fully populated memory interface.

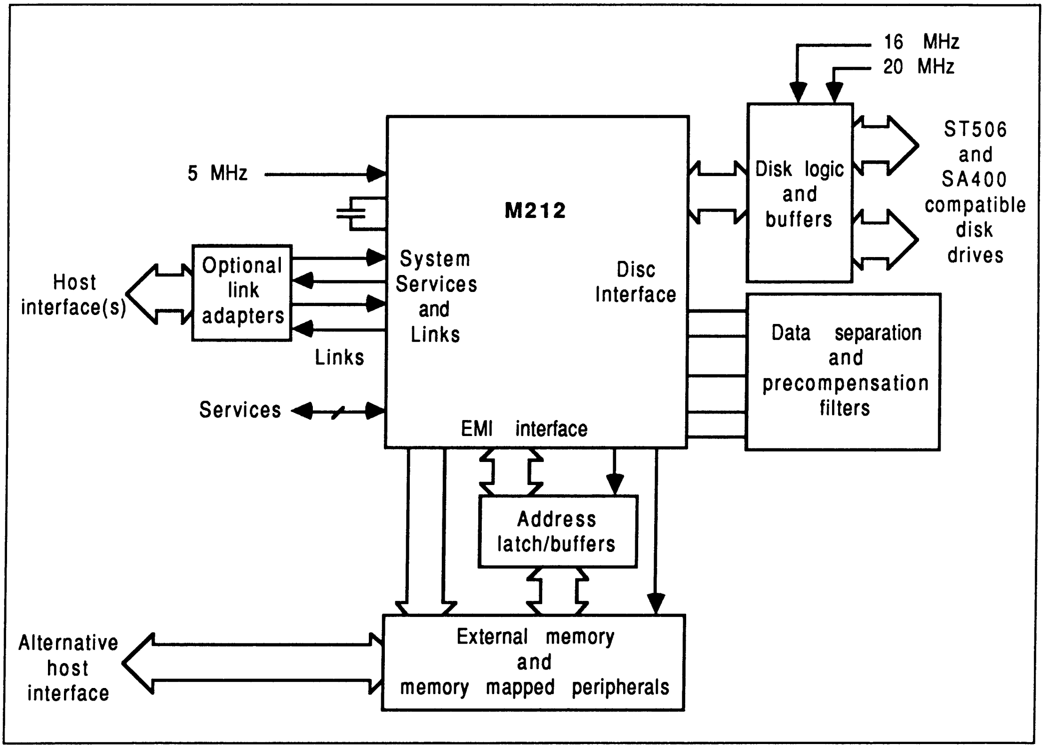


Figure 3.2 Enhanced disk controller interface

The IMS M212 can interface with both floppy and winchester disk drives, and the data rate to and from the disk can be selected by software. As a result the device is suitable for interfacing to the new generation of floppy disk drives which use vertical recording. These drives have an increased data rate of 1 Mbit/sec, and quadruple the capacity of existing floppy disk drives to 4 Mbytes. A single IMS M212 can be used to control a mixture of standard floppy drives, winchester drives and the new high speed high capacity drives. This eases compatibility and portability problems, and provides a simple upgrade path from standard floppies to high capacity floppies to winchesters.

The IMS M212 provides a very simple and compact disk controller solution, making it very easy to replace a single large disk drive with an array of IMS M212's, each controlling a single smaller disk drive. This has several advantages: cheaper drives can be used; overall available disk bandwidth is increased; local processing is provided by a high performance processor at each disk node; fault tolerant operation. The latter can be achieved by holding duplicated data on several drives. This prevents the whole system from stopping, as would be the case if the single large drive failed.

These advantages are particularly applicable when transputers are connected in arrays to provide high performance concurrent systems (figure 3.3). The IMS M212's can be directly connected to the array via INMOS links and the spare link used to communicate with the adjacent IMS M212 to provide the fault tolerant operation.

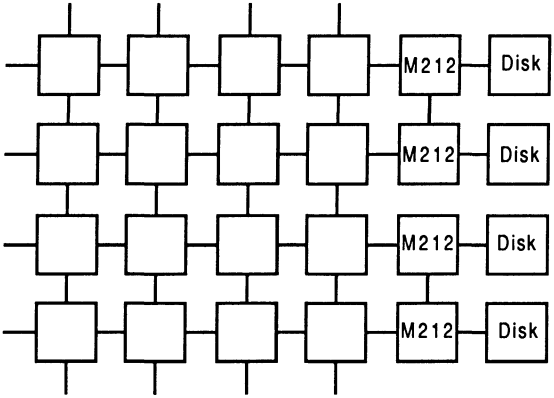


Figure 3.3 Transputer network with disk processors

A high performance processor allows many operations to be performed locally to the disk. This not only frees the host processor for other work but also removes the need for large amounts of data to be needlessly transferred to the host. Operations which can be performed by the IMS M212 include: file management with directory management and pre-reading; data manipulation such as compression/de-compression and encryption/de-cryption; data search such as database key searching; performance optimisation such as head scheduling and cacheing.

The IMS M212 external memory interface can be used to connect to memory mapped peripherals. One application of this is interfacing to a SCSI bus controller, permitting direct connection to the SCSI bus in a low part count system. The processor is used to control the SCSI bus controller and implements the required command interface, as well as controlling the disk or other peripheral.

This arrangement allows floppy and winchester disks to be simply connected to a SCSI bus. Because the command interface is controlled by a process running in the IMS M212, any future command upgrades can easily be incorporated.

The design can be used both as a target and an initiator interface, again controlled by the process running in the IMS M212. It provides a means of implementing a link to SCSI interface, as well as a SCSI controlled disk.

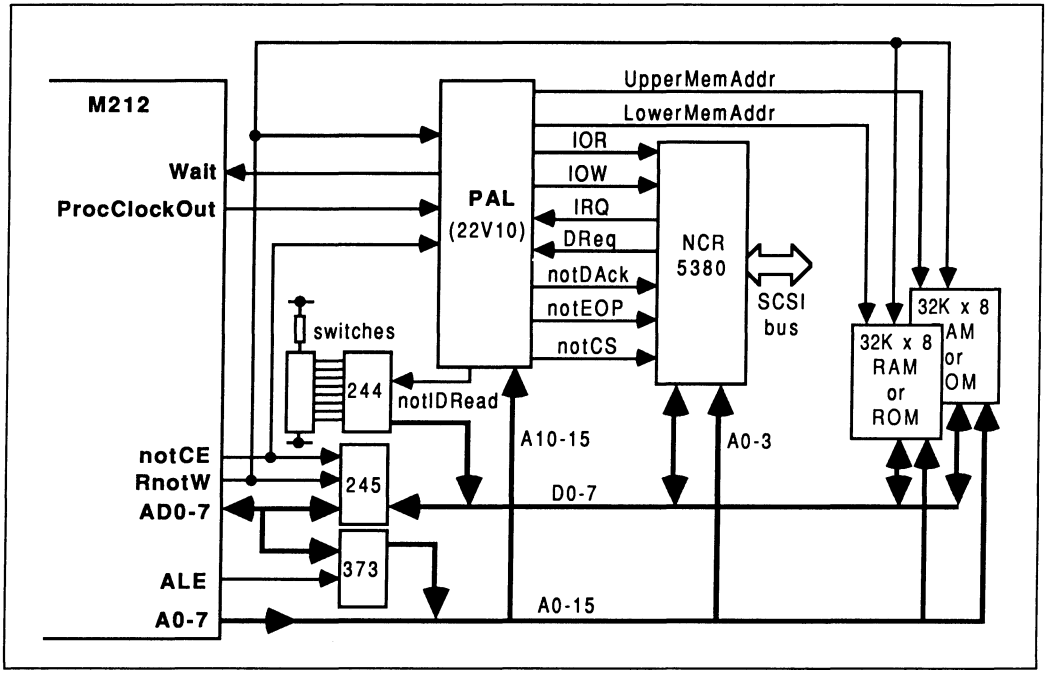
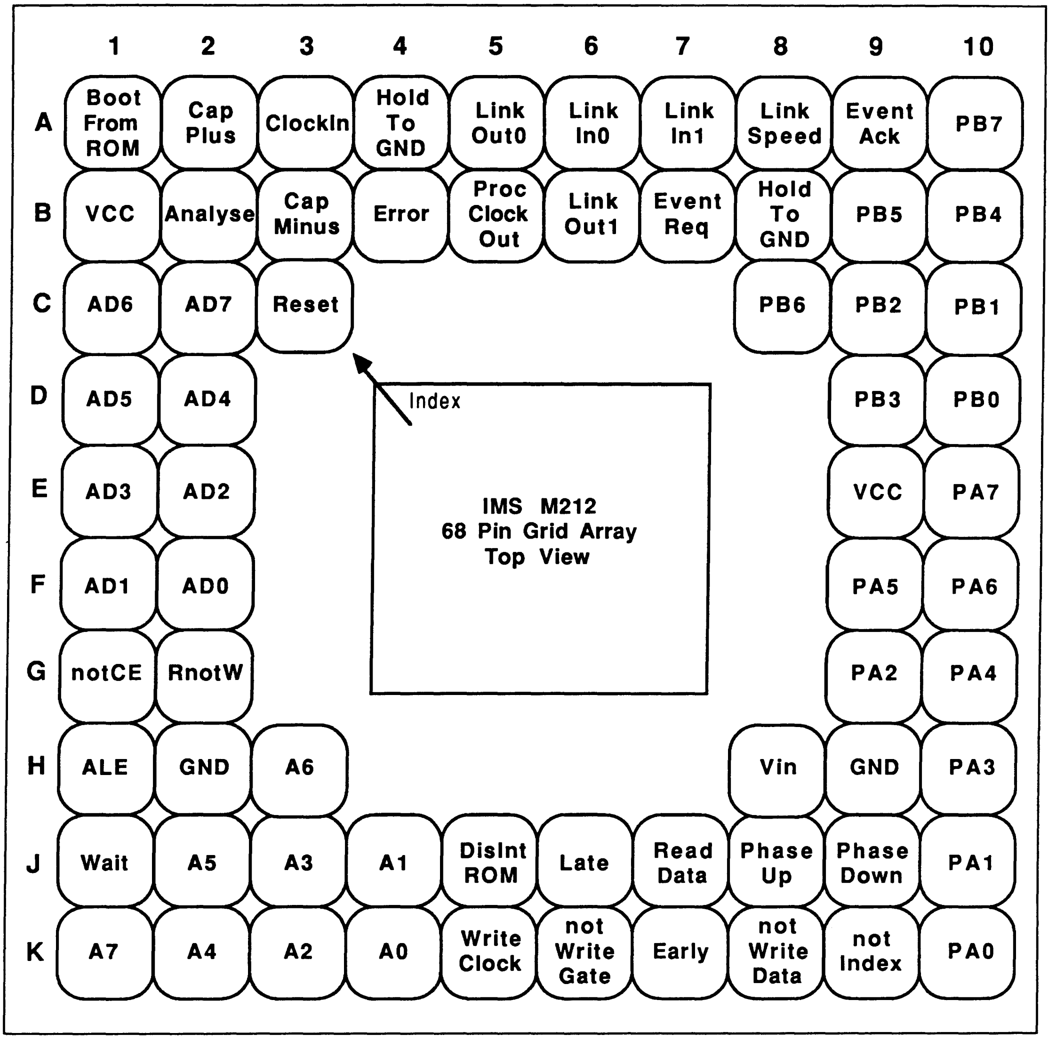


Figure 3.4 SCSI interface

4 Package specifications

4.1 68 pin grid array package



	1	2	3	4	5	6	7	8	9	10
A	Boot From ROM	Cap Plus	ClockIn	Hold To GND	Link Out0	Link In0	Link In1	Link Speed	Event Ack	PB7
B	VCC	Analyse	Cap Minus	Error	Proc Clock Out	Link Out1	Event Req	Hold To GND	PB5	PB4
C	AD6	AD7	Reset					PB6	PB2	PB1
D	AD5	AD4							PB3	PB0
E	AD3	AD2							VCC	PA7
F	AD1	AD0							PA5	PA6
G	notCE	RnotW							PA2	PA4
H	ALE	GND	A6					Vin	GND	PA3
J	Wait	A5	A3	A1	DisInt ROM	Late	Read Data	Phase Up	Phase Down	PA1
K	A7	A4	A2	A0	Write Clock	not Write Gate	Early	not Write Data	not Index	PA0

Figure 4.1 IMS M212 68 pin grid array package pinout

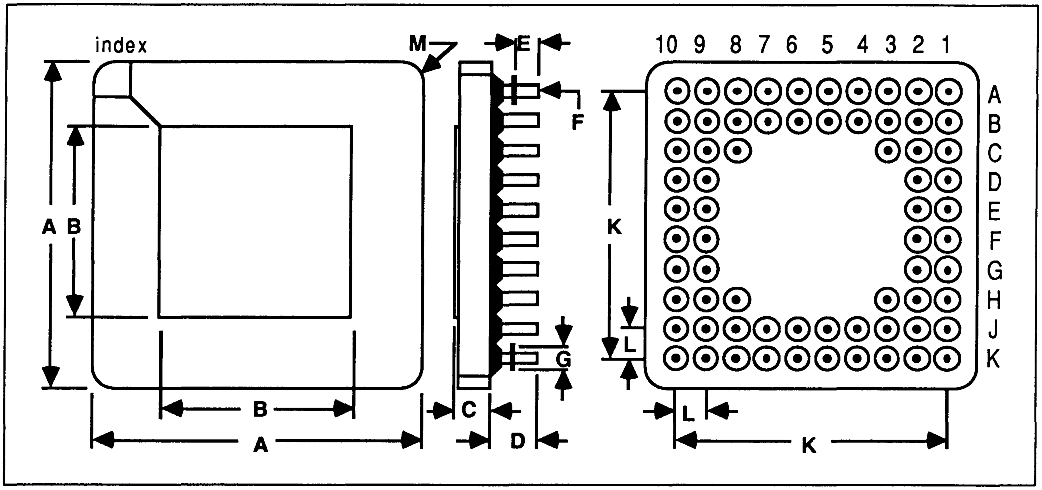


Figure 4.2 68 pin grid array package dimensions

Table 4.1 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 4.2 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	

4.2 68 pin PLCC J-bend package

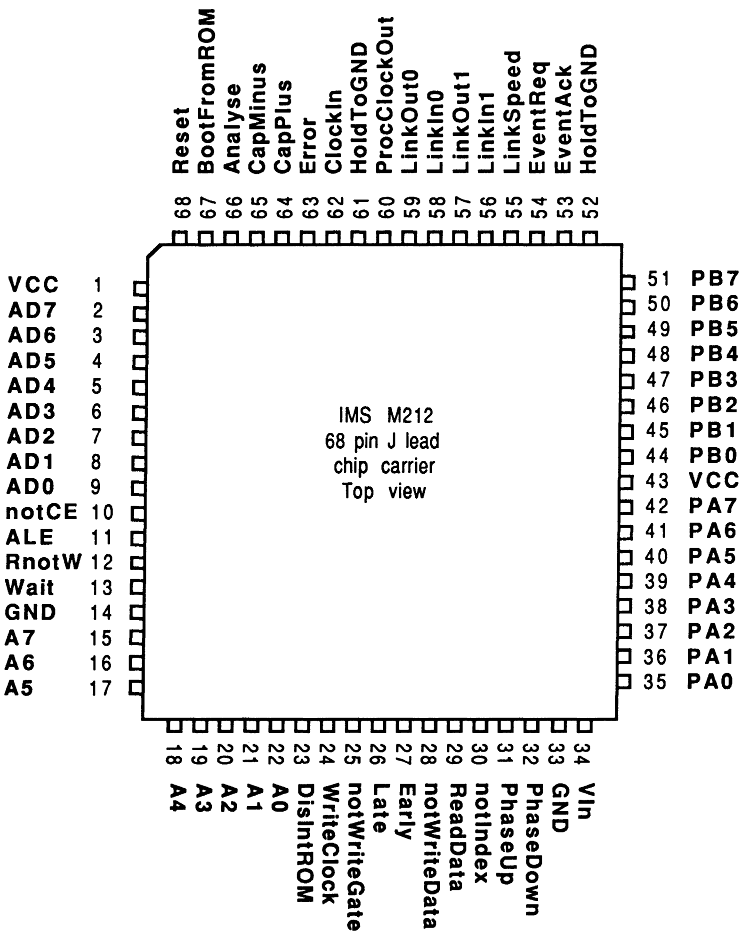


Figure 4.3 IMS M212 68 pin PLCC J-bend package pinout

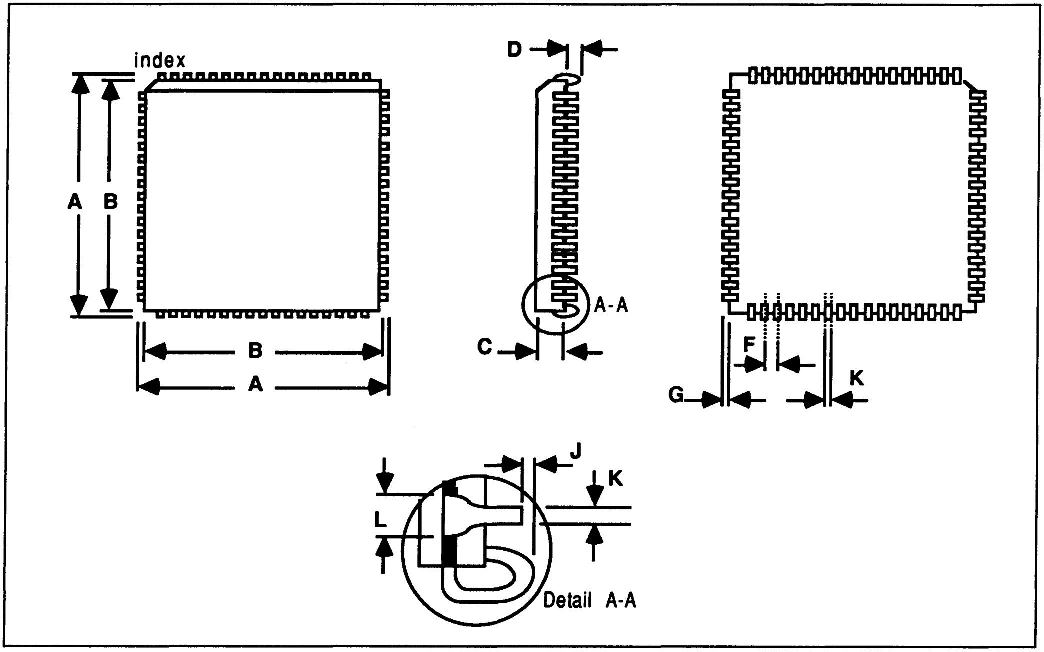


Figure 4.4 68 pin PLCC J-bend package dimensions

Table 4.3 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 4.4 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	

5 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 5.1 IMS M212 ordering details

INMOS designation	**Processor clock speed**	**Processor cycle time**	**PLLx**	**Package**
IMS M212-G15S	15 MHz	67 ns	3.0	Ceramic Pin Grid
IMS M212-G20S	20 MHz	50 ns	4.0	Ceramic Pin Grid
IMS M212-J15S	15 MHz	67 ns	3.0	Plastic PLCC J-Bend
IMS M212-J20S	20 MHz	50 ns	4.0	Plastic PLCC J-Bend

inmos

IMS C004
programmable link switch

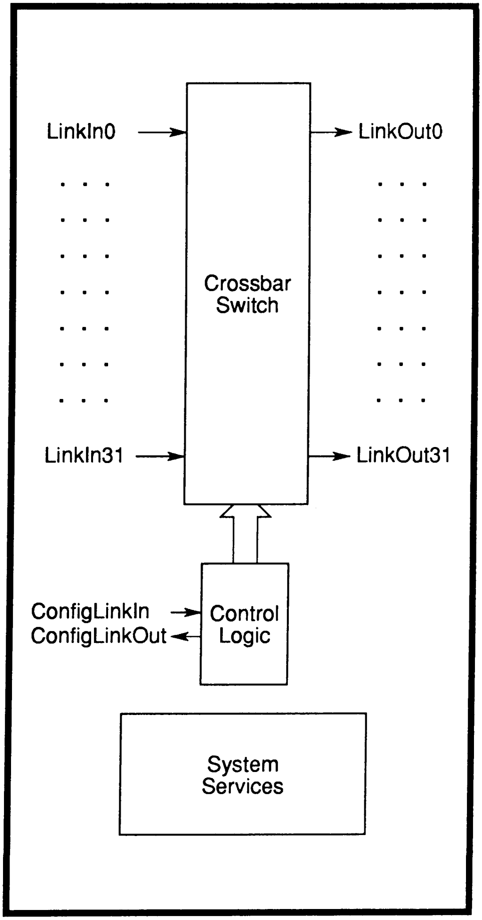
Engineering Data

FEATURES

- Standard INMOS serial links
- 32 way crossbar switch
- Regenerates input signal
- Cascadable to any depth
- No loss of signal integrity
- 10 or 20 Mbits/sec operating speed
- Separate INMOS configuration link
- Single +5V ±5% power supply
- TTL and CMOS compatibility
- 1W power dissipation
- Standard 84 pin ceramic PGA
- MIL-STD-883C device will be available

APPLICATIONS

- Programmable crossbar switch
- Component of larger switch
- Reconfigurable supercomputers
- Message routing system
- High speed multiprocessor systems
- Telecommunications
- Robotics
- Fault tolerant systems
- Additional links for transputers



42 1409 01

October 1988

1 Introduction

The INMOS communication link is a high speed system interconnect which provides full duplex communication between members of the INMOS transputer family, according to the INMOS serial link protocol. The IMS C004, a member of this family, is a transparent programmable link switch designed to provide a full crossbar switch between 32 link inputs and 32 link outputs.

The IMS C004 will switch links running at either the standard speed of 10 Mbits/sec or at the higher speed of 20 Mbits/sec. It introduces, on average, only a 1.75 bit time delay on the signal. Link switches can be cascaded to any depth without loss of signal integrity and can be used to construct reconfigurable networks of arbitrary size. The switch is programmed via a separate serial link called the *configuration link*.

All INMOS products which use communication links, regardless of device type, support a standard communications frequency of 10 Mbits/sec; most products also support 20 Mbits/sec. Products of different type or performance can, therefore, be interconnected directly and future systems will be able to communicate directly with those of today.

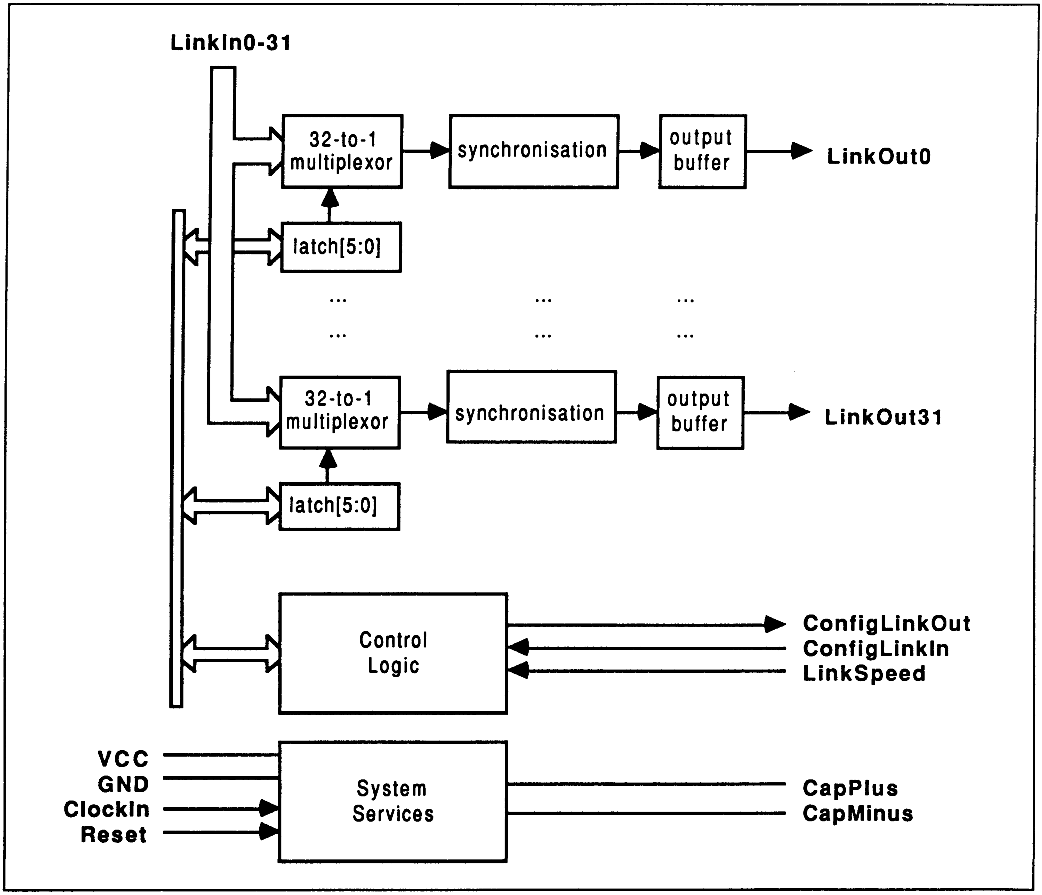


Figure 1.1 IMS C004 block diagram

2 Pin designations

Table 2.1 IMS C004 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
DoNotWire		Must not be wired

Table 2.2 IMS C004 configuration

Pin	In/Out	Function
ConfigLinkIn	in	INMOS configuration link input
ConfigLinkOut	out	INMOS configuration link output

Table 2.3 IMS C004 link

Pin	In/Out	Function
LinkIn0-31	in	INMOS link inputs to the switch
LinkOut0-31	out	INMOS link outputs from the switch
LinkSpeed	in	Link speed selection

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 385.

3 System services

System services include all the necessary logic to start up and maintain the IMS C004.

3.1 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.

3.2 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 50 mm. The connections must not touch power supplies or other noise sources.

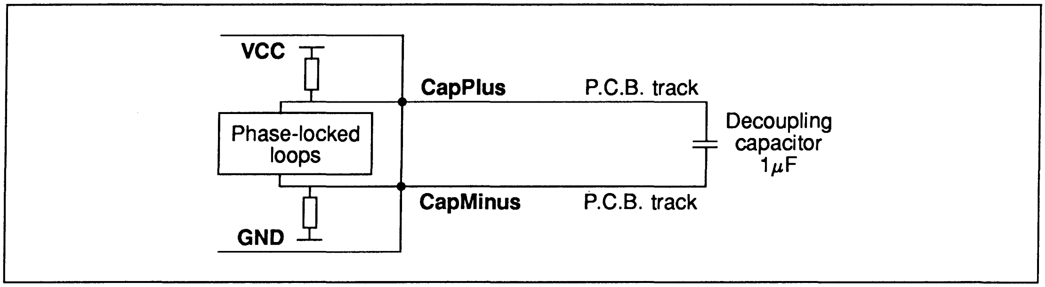


Figure 3.1 Recommended PLL decoupling

3.3 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 family 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 3.1 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 7.3).

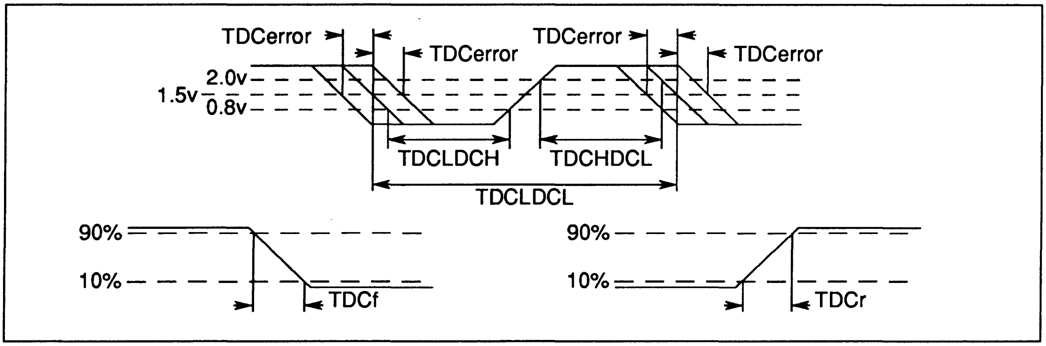


Figure 3.2 ClockIn timing

3.4 Reset

The **Reset** pin 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**.

Reset initialises the IMS C004 to a state where all link outputs from the switch are disconnected and held low; the control link is then ready to receive a configuration message.

Table 3.2 Reset

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

Notes

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

2 At power-on reset.

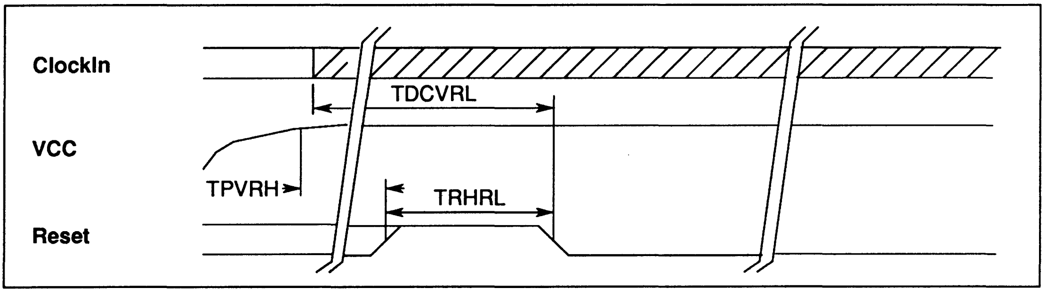


Figure 3.3 Reset Timing

4 Links

INMOS bi-directional serial links provide synchronized communication between INMOS products and with the outside world. Each link comprises an input channel and output channel. A link between two devices is implemented by connecting a link interface on one device to a link interface on the other device. 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.

A receiver can transmit an acknowledge as soon as it starts to receive a data byte. In this way the transmission of an acknowledge can be overlapped with receipt of a data byte to provide continuous transmission of data. This technique is fully compatible with all other INMOS transputer family links.

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 that the receiving link is able to receive another byte.

Links are not synchronised with **ClockIn** and are insensitive to its phase. 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.

The IMS C004 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 20 Mbits/sec. When the **LinkSpeed** pin is low, all links operate at the standard 10 Mbits/sec; when high they operate at 20 Mbits/sec.

A single IMS C004 inserted between two transputers which fully implement overlapped acknowledges will cause some reduction in data bandwidth, see table 4.2 and figure 4.7.

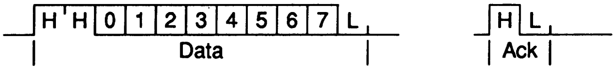


Figure 4.1 IMS C004 link data and acknowledge packets

Table 4.1 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
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.

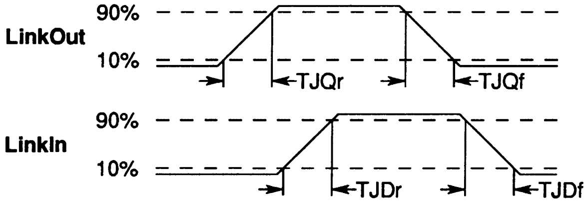


Figure 4.2 IMS C004 link timing

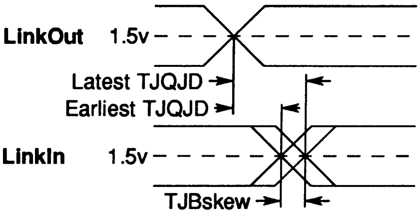


Figure 4.3 IMS C004 buffered link timing

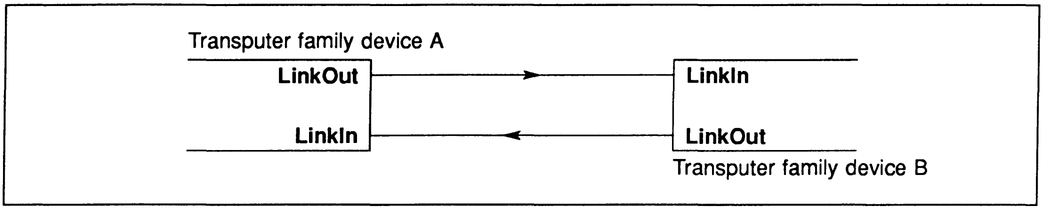


Figure 4.4 IMS C004 Links directly connected

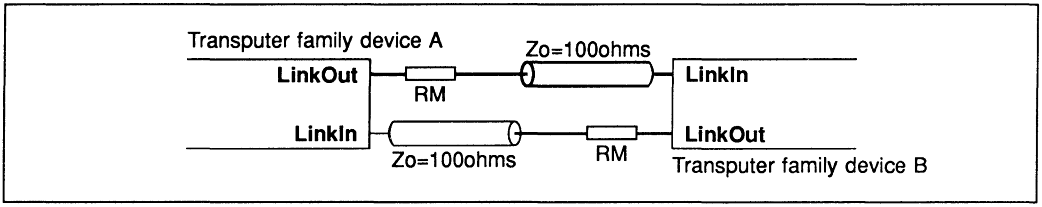


Figure 4.5 IMS C004 Links connected by transmission line

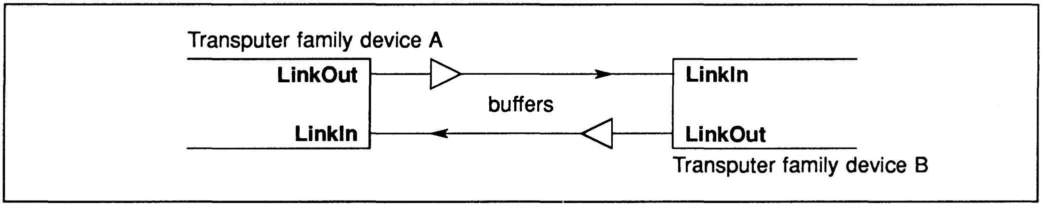


Figure 4.6 IMS C004 Links connected by buffers

Table 4.2 T800 links data transfer rate at 20 Mbits/sec

	Without C004	With C004	Degradation
Unidirectional	1.7	1.3	25%
Bidirectional	2.3	2.1	10%

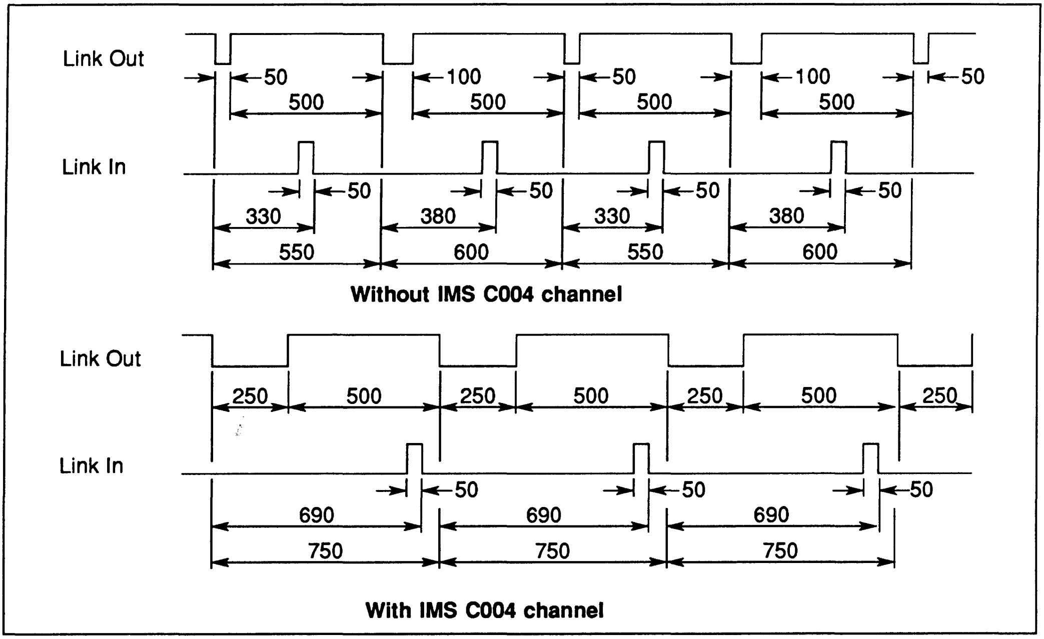


Figure 4.7 IMS C004 link timing

Notes

1 All values are in ns.

2 Timing values shown are for links at 20 Mbits/sec.

5 Switch implementation

The IMS C004 is internally organised as a set of thirtytwo 32-to-1 multiplexors. Each multiplexor has associated with it a six bit latch, five bits of which select one input as the source of data for the corresponding output. The sixth bit is used to connect and disconnect the output. These latches can be read and written by messages sent on the configuration link via **ConfigLinkIn** and **ConfigLinkOut**.

The output of each multiplexor is synchronised with an internal high speed clock and regenerated at the output pad. This synchronisation introduces, on average, a 1.75 bit time delay on the signal. As the signal is not electrically degraded in passing through the switch, it is possible to form links through an arbitrary number of link switches.

Each input and output is identified by a number in the range 0 to 31. A configuration message consisting of one, two or three bytes is transmitted on the configuration link. The configuration messages sent to the switch on this link are shown in table 5.1. If an unspecified configuration message is used, the effect of it is undefined.

Table 5.1 IMS C004 configuration messages

Configuration Message	Function
[0] [input] [output]	Connects **input** to **output**.
[1] [link1] [link2]	Connects **link1** to **link2** by connecting the input of **link1** to the output of **link2** and the input of **link2** to the output of **link1**.
[2] [output]	Enquires which input the **output** is connected to. The IMS C004 responds with the input. The most signifigant bit of this byte indicates whether the output is connected (bit set high) or disconnected (bit set low).
[3]	This command byte must be sent at the end of every configuration sequence which sets up a connection. The IMS C004 is then ready to accept data on the connected inputs.
[4]	Resets the switch. All outputs are disconnected and held low. This also happens when **Reset** is applied to the IMS C004.
[5] [output]	Output **output** is disconnected and held low.
[6] [link1] [link2]	Disconnects the output of **link1** and the output of **link2**.

6 Applications

6.1 Link switching

The IMS C004 provides full switching capabilities between 32 INMOS links. It can also be used as a component of a larger link switch. For example, three IMS C004's can be connected together to produce a 48 way switch, as shown in figure 6.1. This technique can be extended to the switch shown in figure 6.2.

A fully connected network of 32 INMOS transputers (one in which all four links are used on every transputer) can be completely configured using just four IMS C004's. Figure 6.5 shows the connected transputer network.

In these diagrams each link line shown represents a unidirectional link; i.e. one output to one input. Where a number is also given, that denotes the number of lines.

6.2 Multiple IMS C004 control

Many systems require a number of IMS C004's, each configured via its own configuration link. A simple method of implementing this uses a master IMS C004, as shown in figure 6.3. One of the transputer links is used to configure the master link switch, whilst another transputer link is multiplexed via the master to send configuration messages to any of the other 31 IMS C004 links.

6.3 Bidirectional exchange

Use of the IMS C004 is not restricted to computer configuration applications. The ability to change the switch setting dynamically enables it to be used as a general purpose message router. This may, of course, also find applications in computing with the emergence of the new generation of supercomputers, but a more widespread use may be found as a communication exchange.

In the application shown in figure 6.4, a message into the exchange must be preceded by a destination token *dest*. When this message is passed, the destination token is replaced with a source token so that the receiver knows where the message has come from. The **in.out** device in the diagram and the controller can be implemented easily with a transputer, and the link protocol for establishing communication with these devices can be interfaced with INMOS link adaptors. All messages from **rx[i]** are preceded by the destination output *dest*. On receipt of such a message the **in.out** device requests the controller to connect a bidirectional link path to *dest*. The controller determines what is currently connected to each end of the proposed link. When both ends are free it sets up the IMS C004 and informs both ends of the new link. Note that in this network two channels are placed on each IMS C004 link, one for each direction.

6.4 Bus systems

The IMS C004 can be used in conjunction with the INMOS IMS C011/C012 link adaptors to provide a flexible means of connecting conventional bus based microprocessor systems.

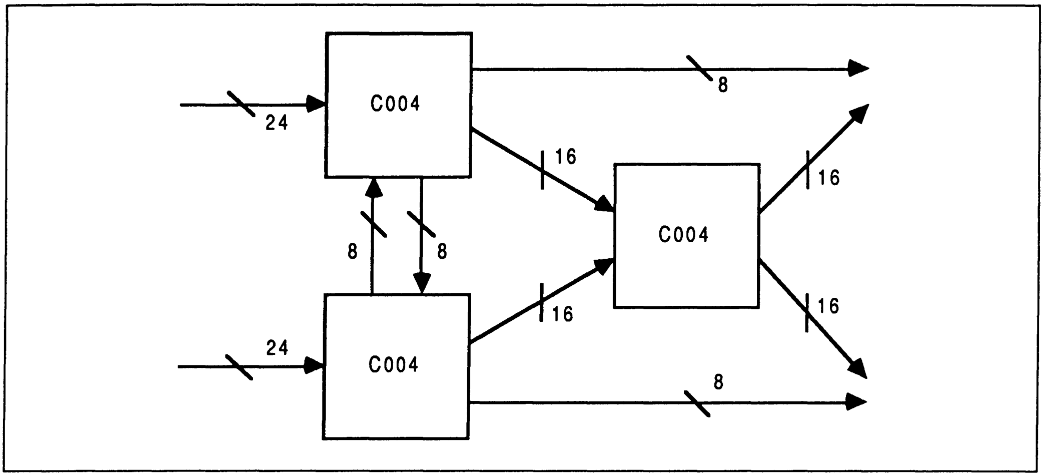


Figure 6.1 48 way link switch

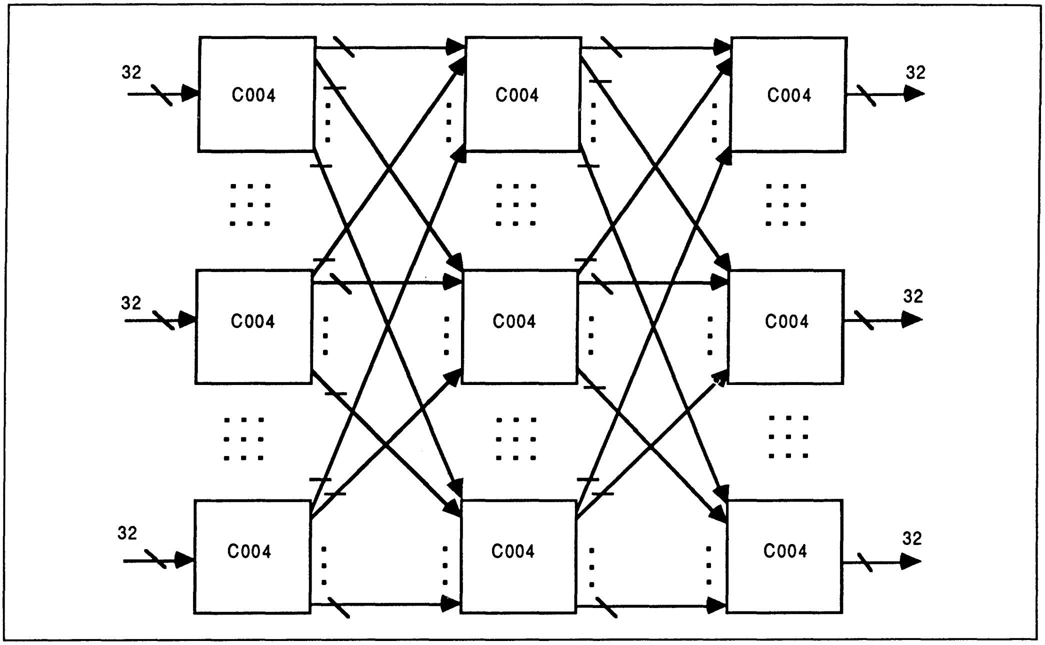


Figure 6.2 Generalised link switch

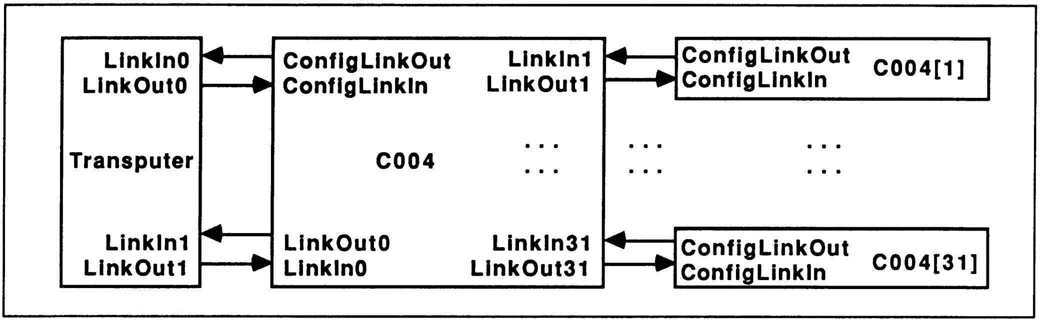


Figure 6.3 Multiple IMS C004 controller

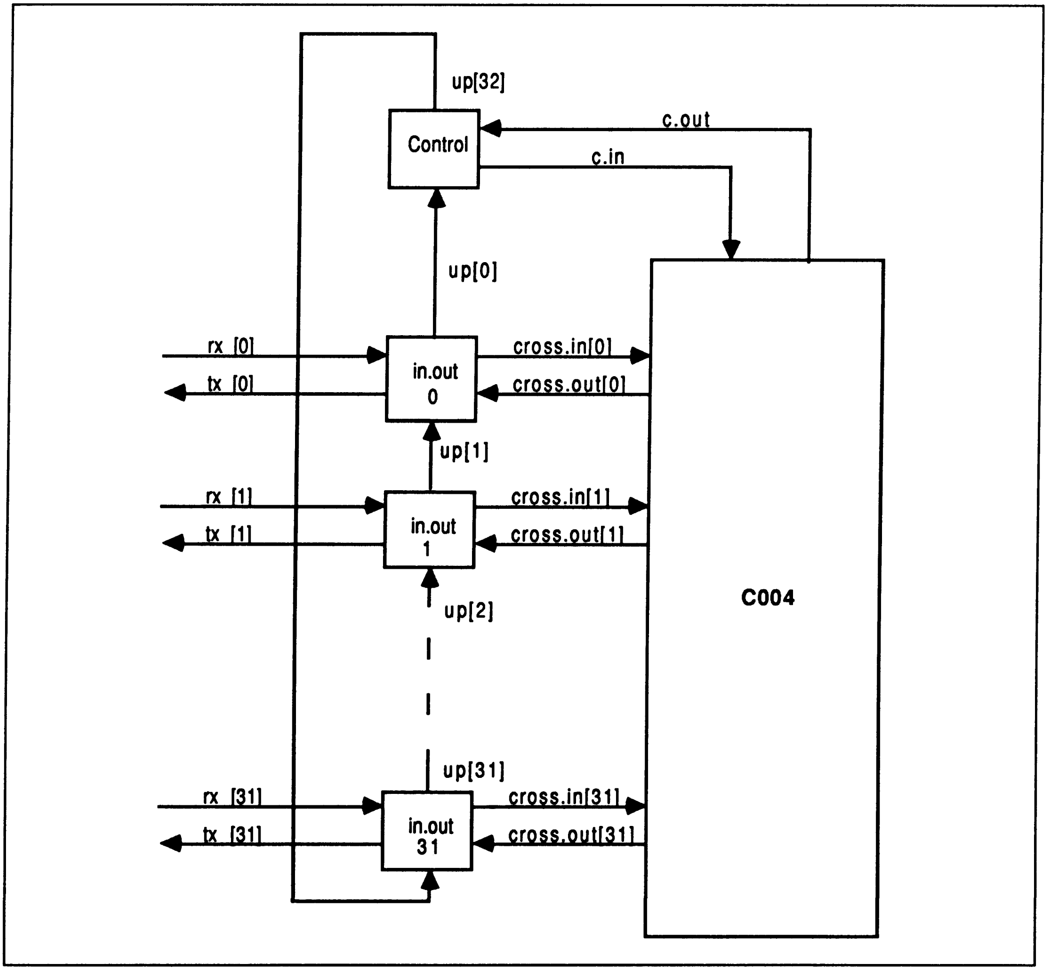


Figure 6.4 32 way bidirectional exchange

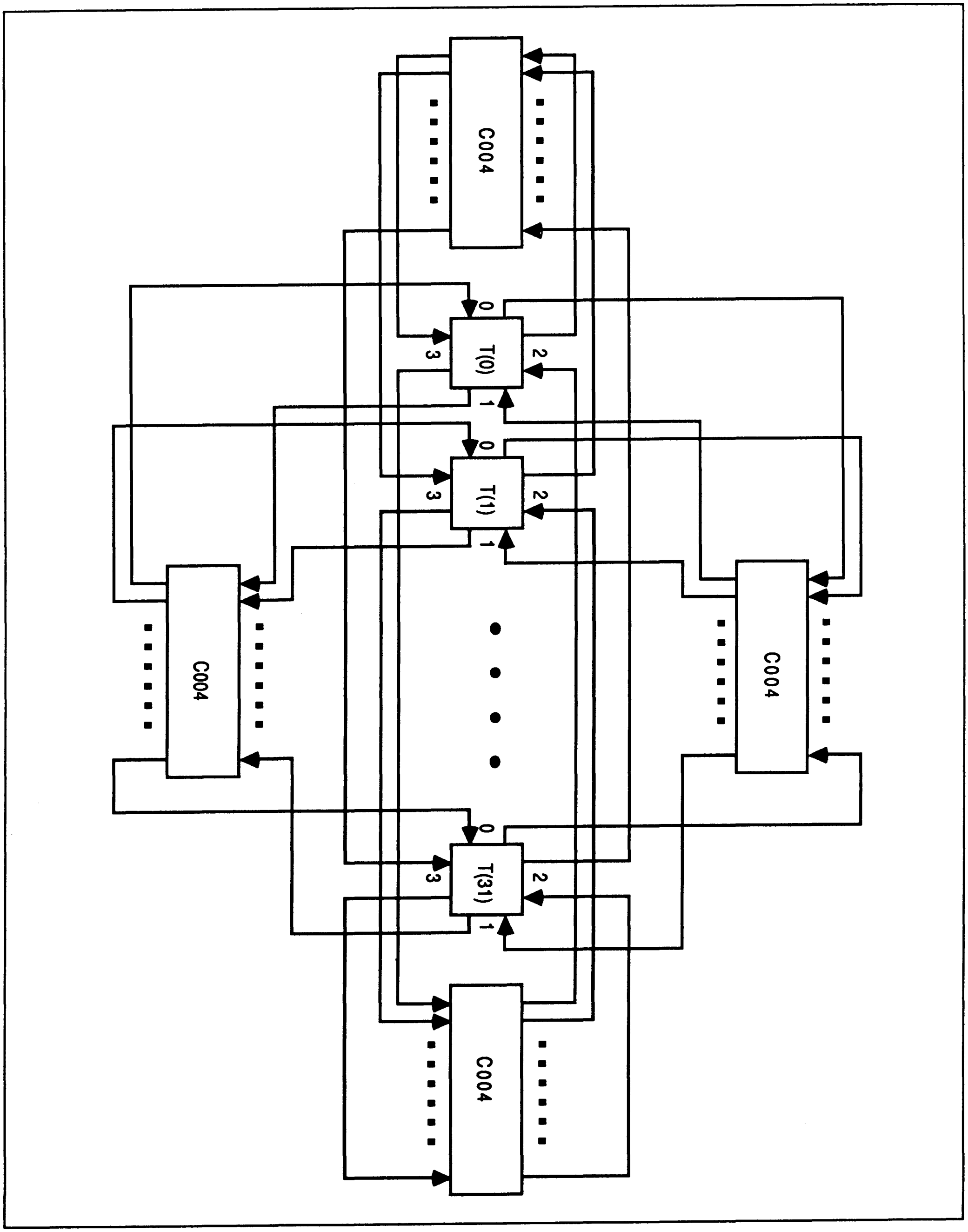


Figure 6.5 Complete connectivity of a transputer network using four IMS C004's

7 Electrical specifications

7.1 DC electrical characteristics

Table 7.1 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 7.2 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 IMS C004-S	0	70	°C	3
TA	Operating temperature range IMS C004-M	-55	125	°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 7.3 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	25	90	mA	1,2,3,7
			50	130	mA	1,2,4,7
			36	65	mA	1,2,3,8
			65	100	mA	1,2,4,8
PD	Power dissipation			1.5	W	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 C004-S measured at 4.75V<**VCC**<5.25V and 0°C<**TA**<70°C.
Parameters for IMS C004-M measured at 4.75V<**VCC**<5.25V and -55°C<**TA**<125°C.
Input clock frequency = 5MHz.

3 Current sourced from non-link outputs.

4 Current sourced from link outputs.

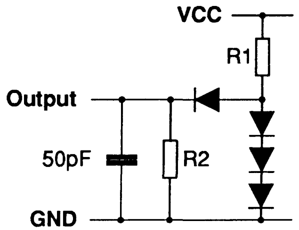
5 Power dissipation varies with output loading and with the number of links active.

6 This parameter is sampled and not 100% tested.

7 Parameter for IMS C004-S.

8 Parameter for IMS C004-M.

7.2 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 7.1 Load circuit for AC measurements

7.3 AC timing characteristics

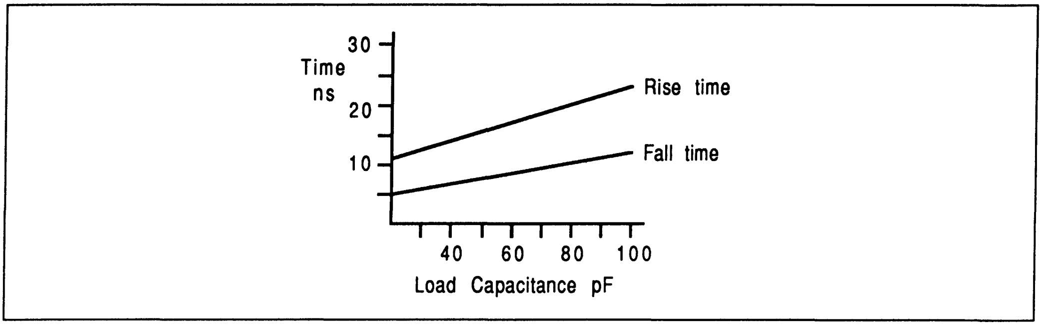


Figure 7.2 Typical link rise/fall times

7.4 Power rating

Internal power dissipation P_{INT} of transputer and peripheral chips depends on **VCC**, as shown in figure 7.3. 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.

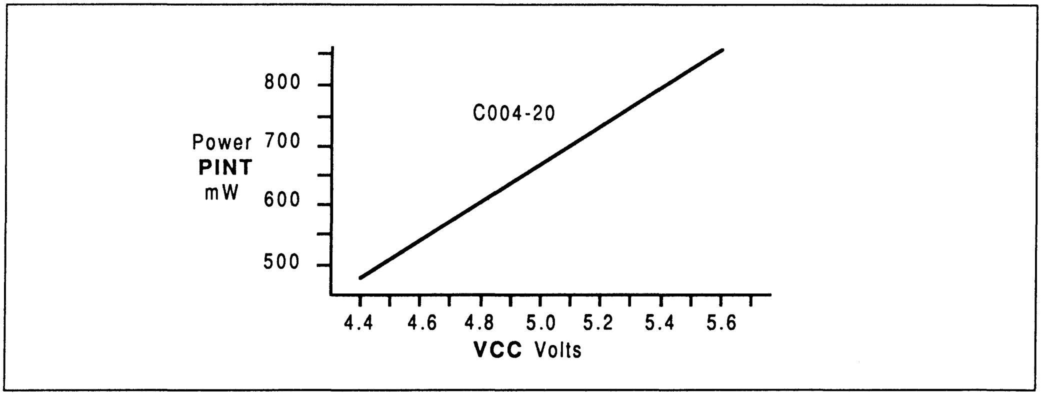
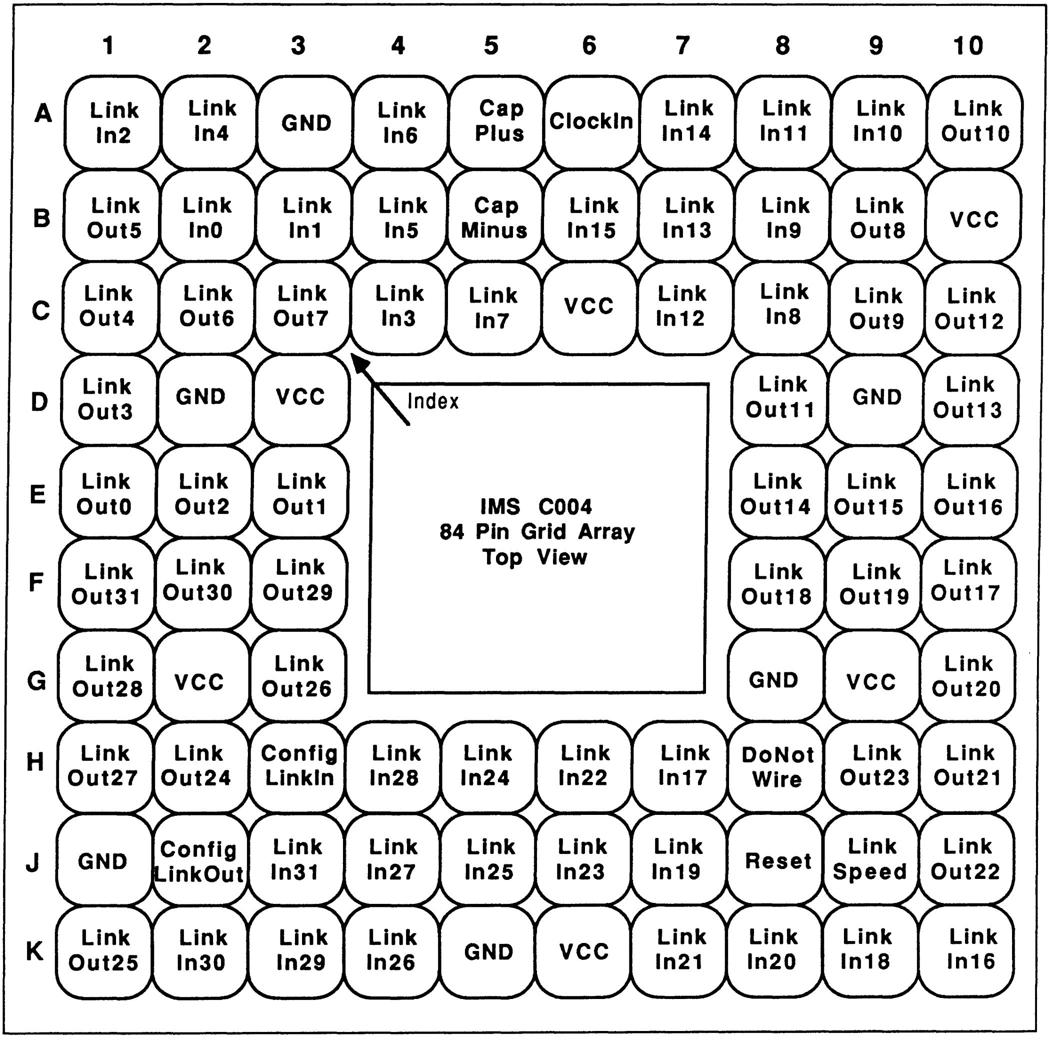


Figure 7.3 IMS C004 internal power dissipation vs VCC

8 Package specifications

8.1 84 pin grid array package



	1	2	3	4	5	6	7	8	9	10
A	Link In2	Link In4	GND	Link In6	Cap Plus	ClockIn	Link In14	Link In11	Link In10	Link Out10
B	Link Out5	Link In0	Link In1	Link In5	Cap Minus	Link In15	Link In13	Link In9	Link Out8	VCC
C	Link Out4	Link Out6	Link Out7	Link In3	Link In7	VCC	Link In12	Link In8	Link Out9	Link Out12
D	Link Out3	GND	VCC					Link Out11	GND	Link Out13
E	Link Out0	Link Out2	Link Out1					Link Out14	Link Out15	Link Out16
F	Link Out31	Link Out30	Link Out29					Link Out18	Link Out19	Link Out17
G	Link Out28	VCC	Link Out26					GND	VCC	Link Out20
H	Link Out27	Link Out24	Config LinkIn	Link In28	Link In24	Link In22	Link In17	DoNot Wire	Link Out23	Link Out21
J	GND	Config LinkOut	Link In31	Link In27	Link In25	Link In23	Link In19	Reset	Link Speed	Link Out22
K	Link Out25	Link In30	Link In29	Link In26	GND	VCC	Link In21	Link In20	Link In18	Link In16

Figure 8.1 IMS C004 84 pin grid array package pinout

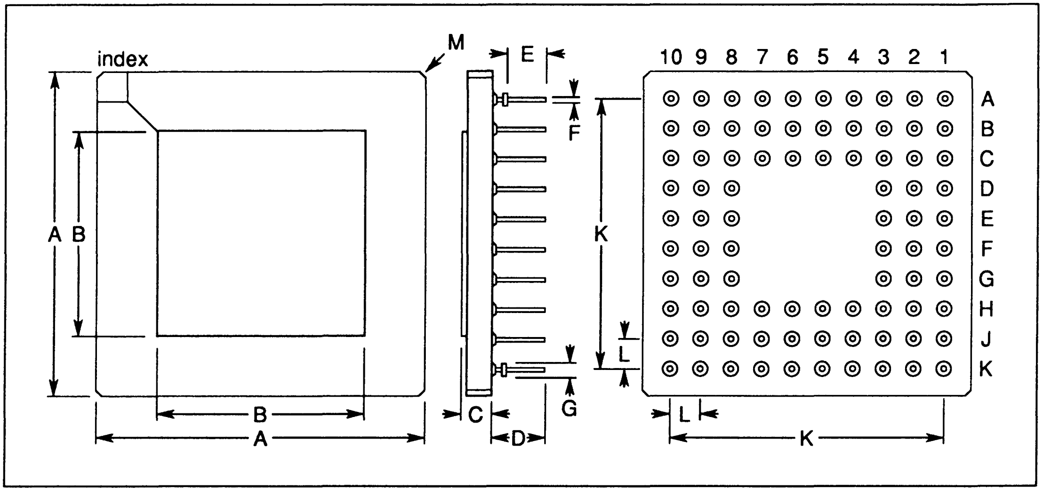


Figure 8.2 84 pin grid array package dimensions

Table 8.1 84 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	±0.005	
C	2.456	±0.278	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	±0.025	0.018	±0.001	Pin diameter
G	1.143	±0.127	0.045	±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 7.2 grams

Table 8.2 84 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	

9 Ordering

This section indicates the designation of package selections for the IMS C004. Speed of **ClockIn** is 5 MHz for all parts.

For availability contact local INMOS sales office or authorised distributor.

Table 9.1 IMS C004 ordering details

INMOS designation	Package
IMS C004-G20S	Ceramic Pin Grid Array
IMS C004-G20M	Ceramic Pin Grid Array MIL Spec

IMS C011
link adaptor

inmos®

Engineering Data

FEATURES

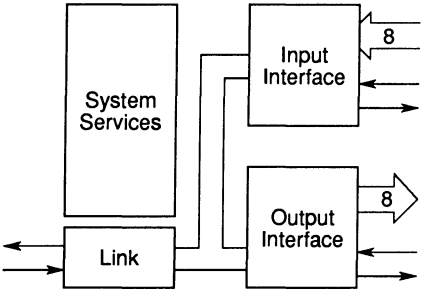
Standard INMOS link protocol
10 or 20 Mbits/sec operating speed
Communicates with INMOS transputers
Converts between serial link and parallel bus
Converts between serial link and parallel device

Two modes of parallel operation:
- **Mode 1: Peripheral interface**
  - Eight bit parallel input interface
  - Eight bit parallel output interface
  - Full handshake on input and output
- **Mode 2: Bus interface**
  - Tristate bidirectional bus interface
  - Memory mapped registers
  - Interrupt capability

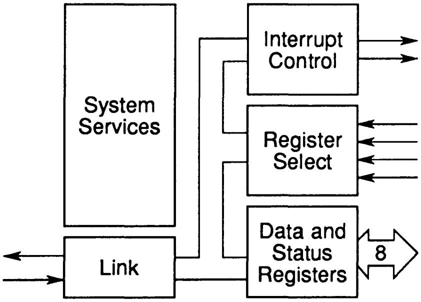
Single +5V ±5% power supply
TTL and CMOS compatibility
120mW power dissipation
Standard 28 pin 0.6" plastic package
MIL-STD-883C device will be available

APPLICATIONS

Programmable I/O pins for transputer
Connecting microprocessors to transputers
High speed links between microprocessors
Inter-family microprocessor interfacing
Interconnecting different speed links



Mode 1



Mode 2

1 Introduction

The INMOS communication link is a high speed system interconnect which provides full duplex communication between members of the INMOS transputer family, according to the INMOS serial link protocol. The IMS C011, a member of this family, provides for full duplex transputer link communication with standard microprocessor and sub-system architectures, by converting bi-directional serial link data into parallel data streams. The extended temperature version of the device complies with MIL-STD-883C.

All INMOS products which use communication links, regardless of device type, support a standard communications frequency of 10 Mbits/sec; most products also support 20 Mbits/sec. Products of different type or performance can, therefore, be interconnected directly and future systems will be able to communicate directly with those of today. The IMS C011 link will run at either the standard speed of 10 Mbits/sec or at the higher speed of 20 Mbits/sec. Data reception is asynchronous, allowing communication to be independent of clock phase.

The link adaptor can be operated in one of two modes. In Mode 1 the IMS C011 converts between a link and two independent fully handshaken byte-wide interfaces, one input and one output. It can be used by a peripheral device to communicate with a transputer, an INMOS peripheral processor or another link adaptor, or it can provide programmable input and output pins for a transputer. Two IMS C011 devices in this mode can be connected back to back via the parallel ports and used as a frequency changer between different speed links.

In Mode 2 the IMS C011 provides an interface between an INMOS serial link and a microprocessor system bus. Status and data registers for both input and output ports can be accessed across the byte-wide bi-directional interface. Two interrupt outputs are provided, one to indicate input data available and one for output buffer empty.

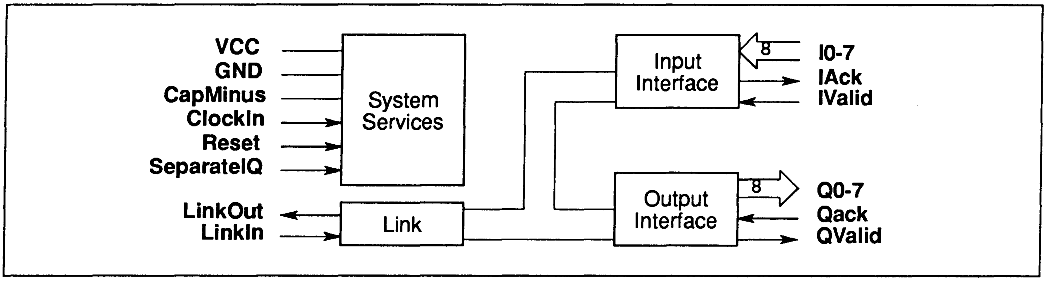


Figure 1.1 IMS C011 Mode 1 block diagram

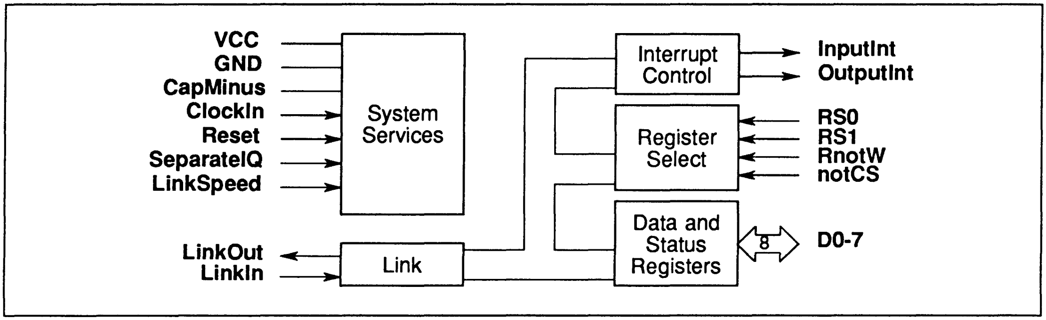


Figure 1.2 IMS C011 Mode 2 block diagram

2 Pin designations

Table 2.1 IMS C011 services and link

Pin	In/Out	Function
VCC, GND		Power supply and return
CapMinus		External capacitor for internal clock power supply
ClockIn	in	Input clock
Reset	in	System reset
SeparateIQ	in	Select mode and Mode 1 link speed
LinkIn	in	Serial data input channel
LinkOut	out	Serial data output channel

Table 2.2 IMS C011 Mode 1 parallel interface

Pin	In/Out	Function
I0-7	in	Parallel input bus
IValid	in	Data on **I0-7** is valid
IAck	out	Acknowledge **I0-7** data received by other link
Q0-7	out	Parallel output bus
QValid	out	Data on **Q0-7** is valid
QAck	in	Acknowledge from device: data **Q0-7** was read

Table 2.3 IMS C011 Mode 2 parallel interface

Pin	In/Out	Function
D0-7	in/out	Bi-directional data bus
notCS	in	Chip select
RS0-1	in	Register select
RnotW	in	Read/write control signal
InputInt	out	Interrupt on link receive buffer full
OutputInt	out	Interrupt on link transmit buffer empty
LinkSpeed	in	Select link speed as 10 or 20 Mbits/sec
HoldToGND		Must be connected to **GND**
DoNotWire		Must not be wired

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 409.

3 System services

System services include all the necessary logic to start up and maintain the IMS C011.

3.1 Power

Power is supplied to the device via the **VCC** and **GND** pins. 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.

AC noise between **VCC** and **GND** must be kept below 200 mV peak to peak at all frequencies above 100 KHz. AC noise between **VCC** and the ground reference of load capacitances must be kept below 200 mV peak to peak at all frequencies above 30 MHz. 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.

3.2 CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance 1μF capacitor to be connected between **VCC** 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 50 mm. The positive connection of the capacitor must be connected directly to **VCC**. Connections must not otherwise touch power supplies or other noise sources.

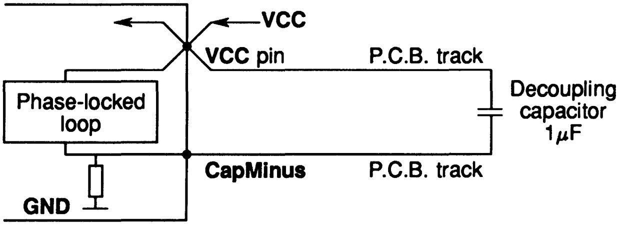


Figure 3.1 Recommended PLL decoupling

3.3 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 family 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 3.1 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	400	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 7.3).

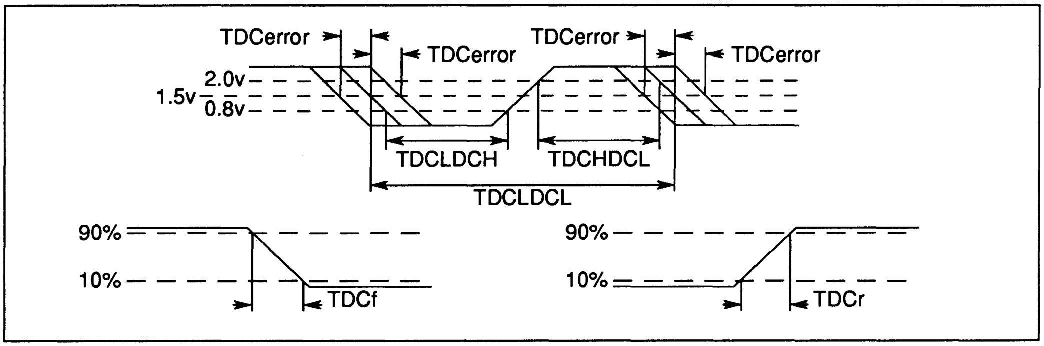


Figure 3.2 ClockIn timing

3.4 SeparateIQ

The IMS C011 link adaptor has two different modes of operation. Mode 1 is basically a link to peripheral adaptor, whilst Mode 2 interfaces between a link and a microprocessor bus system.

Mode 1 can be selected for one of two link speeds by connecting **SeparateIQ** to **VCC** (10 Mbits/sec) or to **ClockIn** (20 Mbits/sec).

Mode 2 is selected by connecting **SeparateIQ** to **GND**; in this mode 10 Mbits/sec or 20 Mbits/sec is selected by **LinkSpeed**. Link speeds are specified for a **ClockIn** frequency of 5 MHz.

In order to select the link speed, **SeparateIQ** may be changed dynamically providing the link is in a quiescent state and no input or output is required. **Reset** must be applied subsequent to the selection to initialise the device. If **ClockIn** is gated to achieve this, its skew must be limited to the value **TDCHSIQH** shown in table 3.3. The mode of operation (Mode 1, Mode 2) must not be changed dynamically.

Table 3.2 SeparateIQ mode selection

SeparateIQ	Mode	Link Speed Mbits/sec
VCC	1	10
ClockIn	1	20
GND	2	10 or 20

Table 3.3 SeparateIQ

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCHSIQH	Skew from ClockIn to ClockIn			20	ns	1

Notes

1 Skew between **ClockIn** arriving on the **ClockIn** pin and on the **SeparateIQ** pin.

3.5 Reset

The **Reset** pin 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**. **LinkIn** must be held low during **Reset**.

Reset initialises the IMS C011 to the following state: **LinkOut** is held low; the control outputs (**IAck** and **QValid** in Mode 1, **InputInt** and **OutputInt** in Mode 2) are held low; interrupts (Mode 2) are disabled; the states of **Q0-7** in Mode 1 are unspecified; **D0-7** in Mode 2 are high impedance.

Table 3.4 Reset

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

Notes

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

2 At power-on reset.

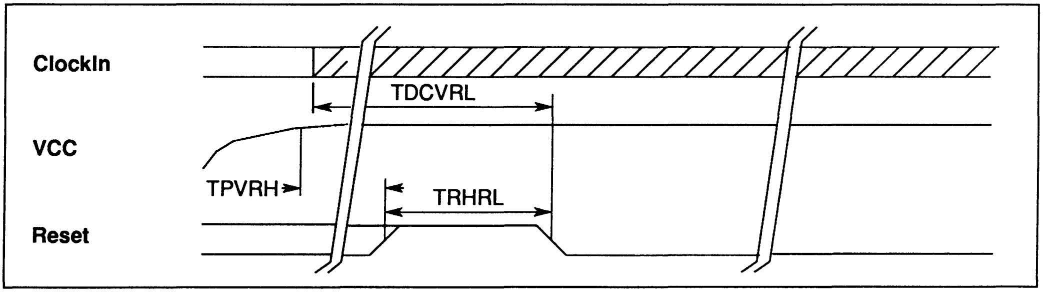


Figure 3.3 Reset Timing

4 Links

INMOS bi-directional serial links provide synchronized communication between INMOS products and with the outside world. Each link comprises an input channel and output channel. A link between two devices is implemented by connecting a link interface on one device to a link interface on the other device. 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.

Links are not synchronised with **ClockIn** and are insensitive to its phase. 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.

The IMS C011 link supports the standard INMOS communication speed of 10 Mbits/sec. In addition it can be used at 20 Mbits/sec. Link speed can be selected in one of two ways. In Mode 1 it is altered by **SeparateIQ** (page 393). In Mode 2 it is selected by **LinkSpeed**; when the **LinkSpeed** pin is low, the link operates at the standard 10 Mbits/sec; when high it operates at 20 Mbits/sec.

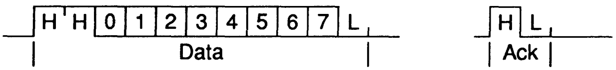


Figure 4.1 IMS C011 link data and acknowledge packets

Table 4.1 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
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.

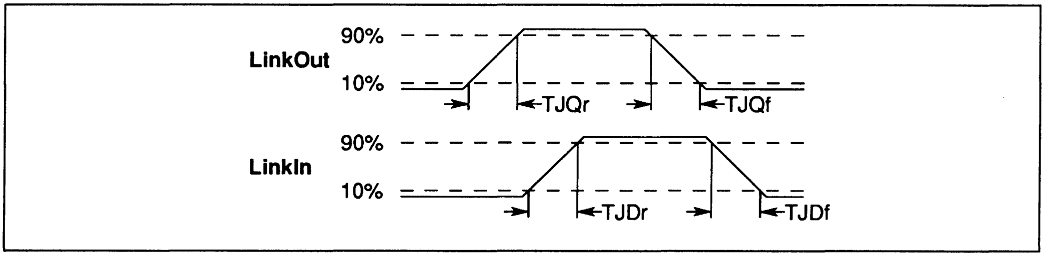


Figure 4.2 IMS C011 link timing

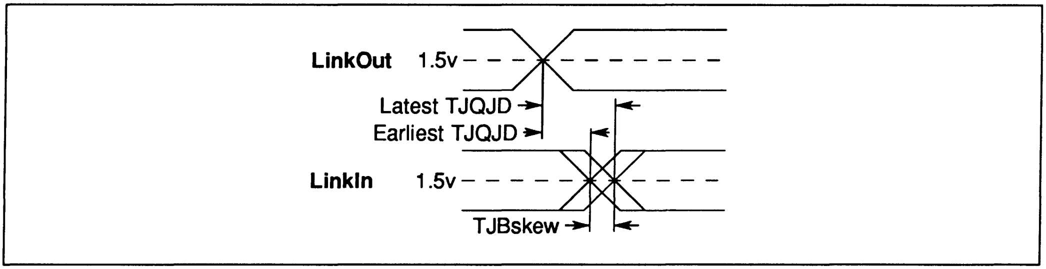


Figure 4.3 IMS C011 buffered link timing

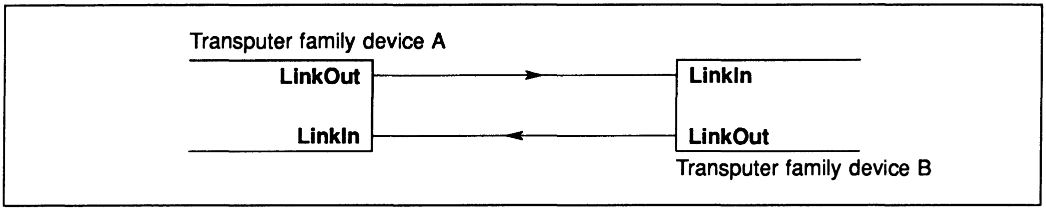


Figure 4.4 IMS C011 Links directly connected

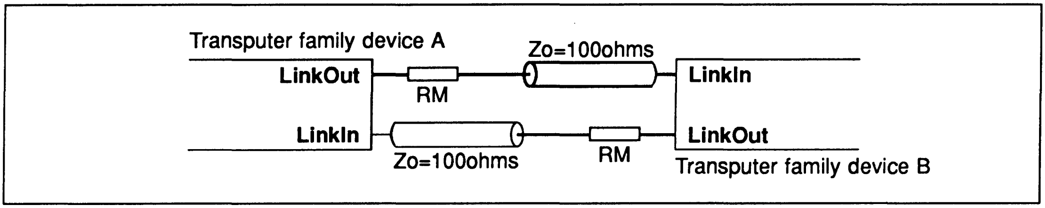


Figure 4.5 IMS C011 Links connected by transmission line

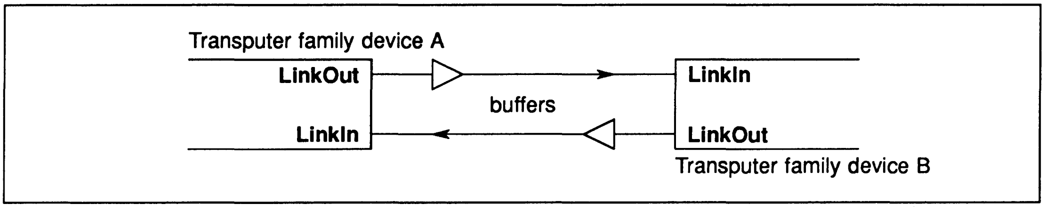


Figure 4.6 IMS C011 Links connected by buffers

5 Mode 1 parallel interface

In Mode 1 the IMS C011 link adaptor is configured as a parallel peripheral interface with handshake lines. Communication with a transputer family device is via the serial link. The parallel interface comprises an input port and an output port, both with handshake.

5.1 Input port

The eight bit parallel input port **I0-7** can be read by a transputer family device via the serial link. **IValid** and **IAck** provide a simple two-wire handshake for this port. When data is valid on **I0-7, IValid** is taken high by the peripheral device to commence the handshake. The link adaptor transmits data presented on **I0-7** out through the serial link. When the acknowledge packet is received on the input link, the IMS C011 sets **IAck** high. To complete the handshake, the peripheral device must return **IValid** low. The link adaptor will then set **IAck** low. New data should not be put onto **I0-7** until **IAck** is returned low.

Table 5.1 Mode 1 parallel data input

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TIdVIvH	Data setup	5			ns	
TIvHLdV	IValid high to link data output	0.8		2	bits	1,2
TLaVIaH	Link acknowledge start to IAck high			3	bits	1
TIaHIdX	Data hold after IAck high	0			ns	
TIaHIvL	IValid hold after IAck high	0			ns	
TIvLIaL	IAck hold after IValid low	1		4	bits	1
TIaLIvH	Delay before next IValid high	0			ns	

Notes

1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 nS.

2 Maximum time assumes there is no acknowledge packet already on the link. Maximum time with acknowledge on the link is extended by 2 bits.

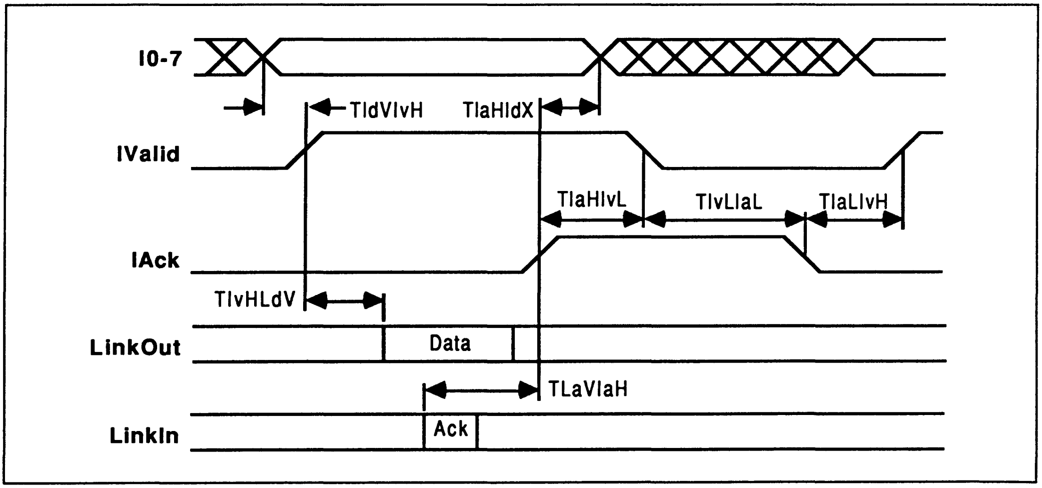


Figure 5.1 IMS C011 Mode 1 parallel data input to link adaptor

5.2 Output port

The eight bit parallel output port **Q0-7** can be controlled by a transputer family device via the serial link. **QValid** and **QAck** provide a simple two-wire handshake for this port.

A data packet received on the input link is presented on **Q0-7**; the link adaptor then takes **QValid** high to initiate the handshake. After reading data from **Q0-7**, the peripheral device sets **QAck** high. The IMS C011 will then send an acknowledge packet out of the serial link to indicate a completed transaction and set **QValid** low to complete the handshake.

Table 5.2 Mode 1 parallel data output

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TLdVQvH	Start of link data to QValid	11.5			bits	1
TQdVQvH	Data setup	15			ns	2
TQvHQaH	QAck setup time from QValid high	0			ns	
TQaHQvL	QAck high to QValid low	1.8			bits	1
TQaHLaV	QAck high to Ack on link	0.8		2	bits	1,3
TQvLQaL	QAck hold after QValid low	0			ns	
TQvLQdX	Data hold	11			bits	1,4

Notes

1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 nS.

2 Where an existing data output bit is re-written with the same level there will be no glitch in the output level.

3 Maximum time assumes there is no data packet already on the link. Maximum time with data on the link is extended by 11 bits.

4 Data output remains valid until overwritten by new data.

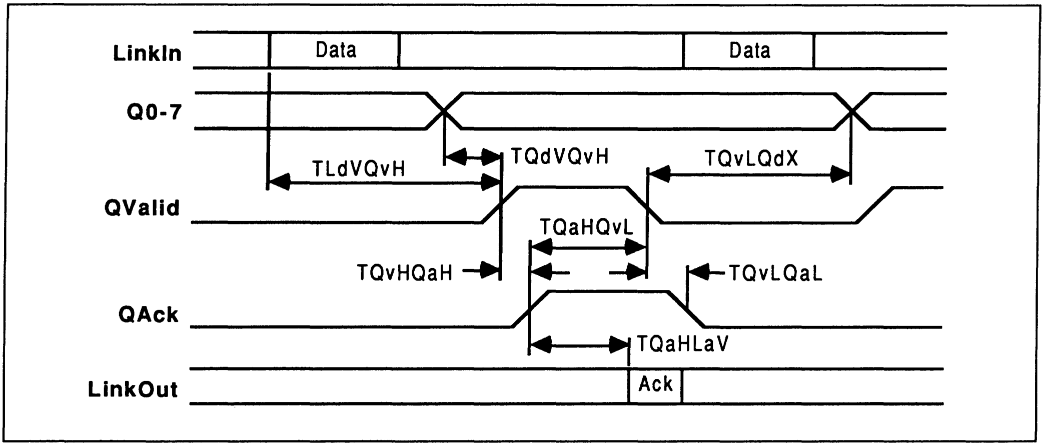


Figure 5.2 IMS C011 Mode 1 parallel data output from link adaptor

6 Mode 2 Parallel interface

The IMS C011 provides an interface between a link and a microprocessor style bus. Operation of the link adaptor is controlled through the parallel interface bus lines **D0-7** by reading and writing various registers in the link adaptor. Registers are selected by **RS0-1** and **RnotW**, and the chip enabled with **notCS**.

For convenience of description, the device connected to the parallel side of the link adaptor is presumed to be a microprocessor, although this will not always be the case.

6.1 D0-7

Data is communicated between a microprocessor bus and the link adaptor via the bidirectional bus lines **D0-7**. The bus is high impedance unless the link adaptor chip is selected and the **RnotW** line is high. The bus is used by the microprocessor to access status and data registers.

6.2 notCS

The link adaptor chip is selected when **notCS** is low. Register selectors **RS0-1** and **RnotW** must be valid before **notCS** goes low; **D0-7** must also be valid if writing to the chip (**RnotW** low). Data is read by the link adaptor on the rising edge of **notCS**.

6.3 RnotW

RnotW, in conjunction with **notCS**, selects the link adaptor registers for read or write mode. When **RnotW** is high, the contents of an addressed register appear on the data bus **D0-7**; when **RnotW** is low the data on **D0-7** is written into the addressed register. The state of **RnotW** is latched into the link adaptor by **notCS** going low; it may be changed before **notCS** returns high, within the timing restrictions given.

6.4 RS0-1

One of four registers is selected by **RS0-1**. A register is addressed by setting up **RS0-1** and then taking **notCS** low; the state of **RnotW** when **notCS** goes low determines whether the register will be read or written. The state of **RS0-1** is latched into the link adaptor by **notCS** going low; it may be changed before **notCS** returns high, within the timing restrictions given. The register set comprises a read-only data input register, a write-only data output register and a read/write status register for each.

Table 6.1 IMS C011 Mode 2 register selection

RS1	RS0	RnotW	Register
0	0	1	Read data
0	0	0	Invalid
0	1	1	Invalid
0	1	0	Write data
1	0	1	Read input status
1	0	0	Write input status
1	1	1	Read output status
1	1	0	Write output status

6.4.1 Input Data Register

This register holds the last data packet received from the serial link. It never contains acknowledge packets. It contains valid data only whilst the *data present* flag is set in the input status register. It cannot be assumed to contain valid data after it has been read; a double read may or may not return valid data on the second read. If *data present* is valid on a subsequent read it indicates new data is in the buffer. Writing to this register will have no effect.

Table 6.2 IMS C011 Mode 2 parallel interface control

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRSVCSL	Register select setup	5			ns	
TCSLRSX	Register select hold	5			ns	
TRWVCSL	Read/write strobe setup	5			ns	
TCSLRWX	Read/write strobe hold	5			ns	
TCSLCSH	Chip select active	50			ns	
TCSHCSL	Delay before re-assertion of chip select	50			ns	

Table 6.3 IMS C011Mode 2 parallel interface read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TLdVIIH	Start of link data to InputInt high			13	bits	1
TCSLIIL	Chip select to InputInt low			30	ns	
TCSLDrX	Chip select to bus active	5			ns	
TCSLDrV	Chip select to data valid			40	ns	
TCSHDrZ	Chip select high to bus tristate			25	ns	
TCSHDrX	Data hold after chip select high	5			ns	
TCSHLaV	Chip de-select to start of Ack	0.8		2	bits	1,2

Notes

1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 nS.

2 Maximum time assumes there is no data packet already on the link. Maximum time with data on the link is extended by 11 bits.

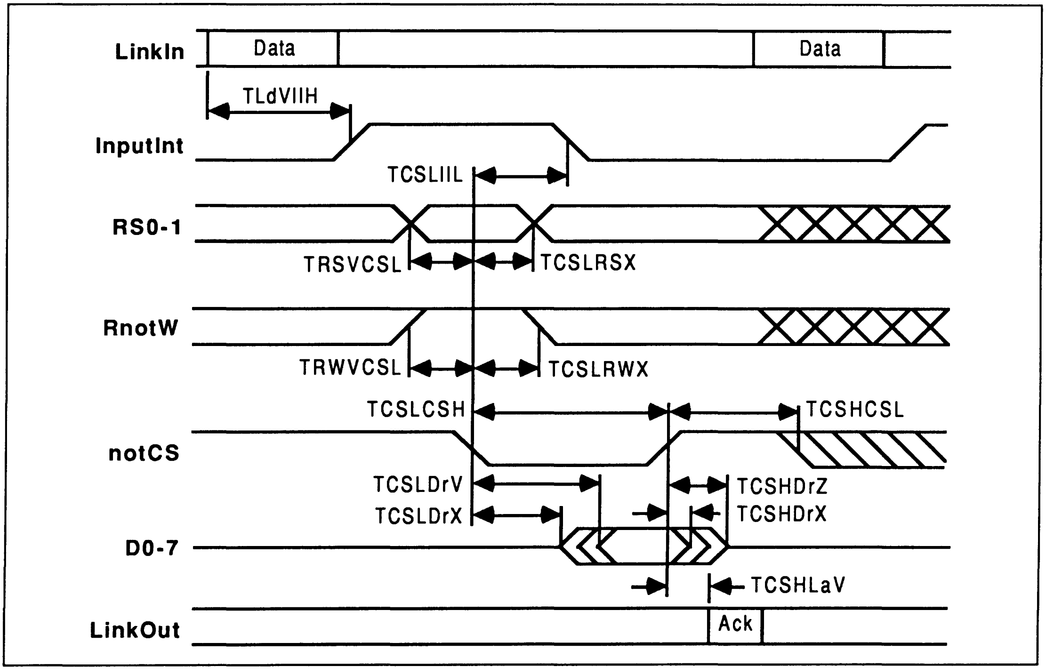


Figure 6.1 IMS C011 Mode 2 read parallel data from link adaptor

Table 6.4 IMS C011 Mode 2 parallel interface write

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TCSHDwV	Data setup	15			ns	
TCSHDwX	Data hold	5			ns	
TCSLOIL	Chip select to OutputInt low			30	ns	
TCSHLdV	Chip select high to start of link data	0.8		2	bits	1,2
TLaVOIH	Start of link Ack to OutputInt high			3	bits	1,3
TLdVOIH	Start of link data to OutputInt high			13	bits	1,3

Notes

1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 nS.

2 Maximum time assumes there is no acknowledge packet already on the link. Maximum time with acknowledge on the link is extended by 2 bits.

3 Both data transmission and the returned acknowledge must be completed before **OutputInt** can go high.

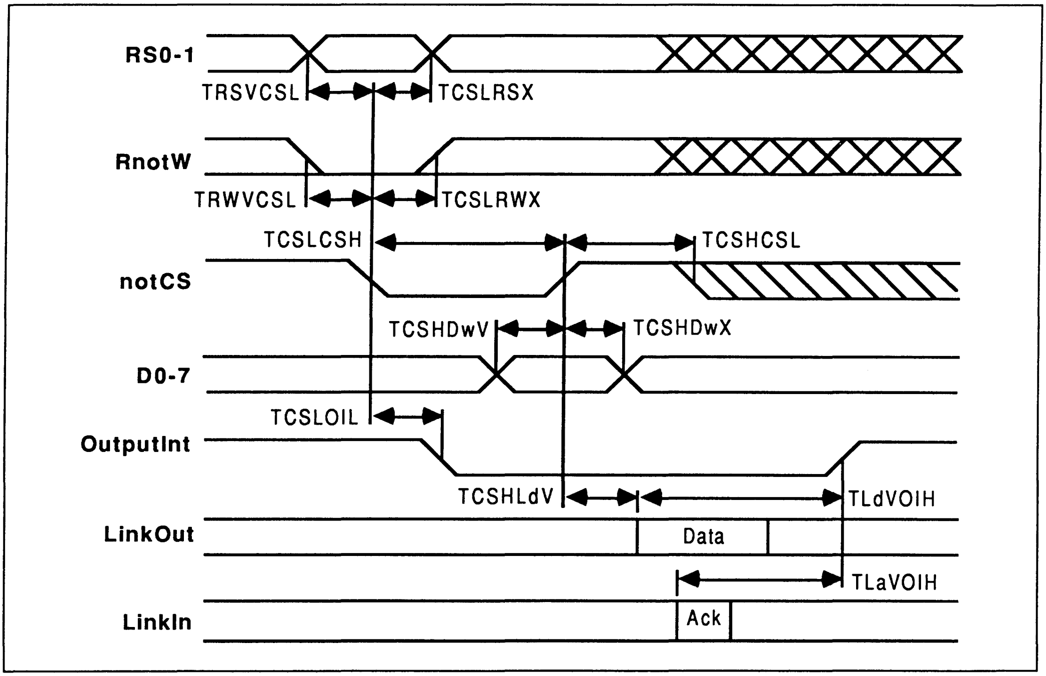


Figure 6.2 IMS C011 Mode 2 write parallel data to link adaptor

6.4.2 Input Status Register

This register contains the *data present* flag and the *interrupt enable* control bit for **InputInt**. The *data present* flag is set to indicate that data in the data input buffer is valid. It is reset low only when the data input buffer is read, or by **Reset**. When writing to this register, the *data present* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *data present* flags are both high, the **InputInt** output will be high (page 403). Resetting *interrupt enable* will take **InputInt** low; setting it again before reading the data input register will set **InputInt** high again. The *interrupt enable* bit can be read to determine its status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

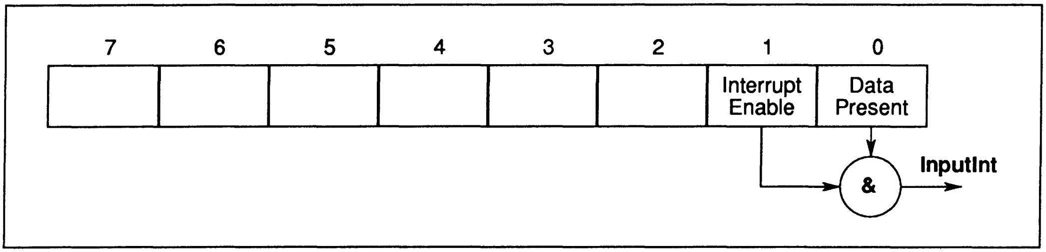


Figure 6.3 IMS C011 input status register

6.5 InputInt

The **InputInt** output is set high to indicate that a data packet has been received from the serial link. It is inhibited from going high when the *interrupt enable* bit in the input status register is low (page 403). **InputInt** is reset low when data is read from the input data register (page 400) and by **Reset** (page 394).

6.5.1 Output Data Register

Data written to this link adaptor register is transmitted out of the serial link as a data packet. Data should only be written to this register when the *output ready* bit in the output status register is high, otherwise data already being transmitted may be corrupted. Reading this register will result in undefined data being read.

6.5.2 Output Status Register

This register contains the *output ready* flag and the *interrupt enable* control bit for **OutputInt**. The *output ready* flag is set to indicate that the data output buffer is empty. It is reset low only when data is written to the data output buffer; it is set high by **Reset**. When writing to this register, the *output ready* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *output ready* flags are both high, the **OutputInt** output will be high (page 404). Resetting *interrupt enable* will take **OutputInt** low; setting it again whilst the data output register is empty will set **OutputInt** high again. The *interrupt enable* bit can be read to determine it's status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

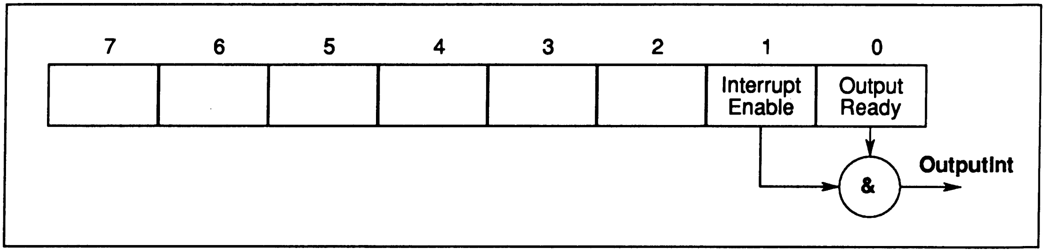


Figure 6.4 IMS C011 output status register

6.6 OutputInt

The **OutputInt** output is set high to indicate that the link is free to receive data from the microprocessor for transmission as a data packet out of the serial link. It is inhibited from going high when the *interrupt enable* bit in the output status register is low (page 403). **OutputInt** is reset low when data is written to the data output register (page 403); it is set high by **Reset** (page 394).

6.7 Data read

A data packet received on the input link sets the *data present* flag in the input status register. If the *interrupt enable* bit in the status register is set, the **InputInt** output pin will be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the input status register until the *data present* bit is high.

When data is available from the link, the microprocessor reads the data packet from the data input register. This will reset the *data present* flag and cause the link adaptor to transmit an acknowledge packet out of the serial link output. **InputInt** is automatically reset by reading the data input register; it is not necessary to read or write the input status register.

6.8 Data write

When the data output buffer is empty the *output ready* flag in the output status register is set high. If the *interrupt enable* bit in the status register is set, the **OutputInt** output pin will also be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the output status register until the *output ready* bit is high.

When the *output ready* flag is high, the microprocessor can write data to the data output buffer. This will result in the link adaptor resetting the *output ready* flag and commencing transmission of the data packet out of the serial link. The *output ready* status bit will remain low until an acknowledge packet is received by the input link. This will set the *output ready* flag high; if the *interrupt enable* bit is set, **OutputInt** will also be set high.

7 Electrical specifications

7.1 DC electrical characteristics

Table 7.1 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		600	mW	

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 7.2 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.

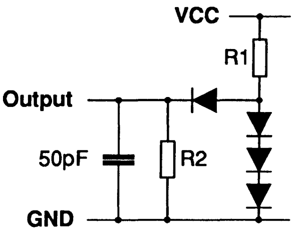
Table 7.3 DC characteristics

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	1,2,7
			±200	µA	1,2,8
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		120	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 C011-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.
6. This parameter is sampled and not 100% tested.
7. For inputs other than those in Note 8.
8. For pins 2, 3, 5, 6, 7, 9, 11, 13, 15, 16, 25.

7.2 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 7.1 Load circuit for AC measurements

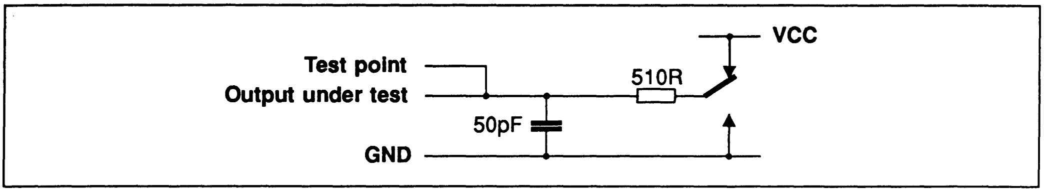


Figure 7.2 Tristate load circuit for AC measurements

7.3 AC timing characteristics

Table 7.4 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
CSLaHZ	Chip select high to tristate		25	ns	
CSLaLZ	Chip select low to tristate		25	ns	

Notes

1 Non-link pins; see section on links.

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

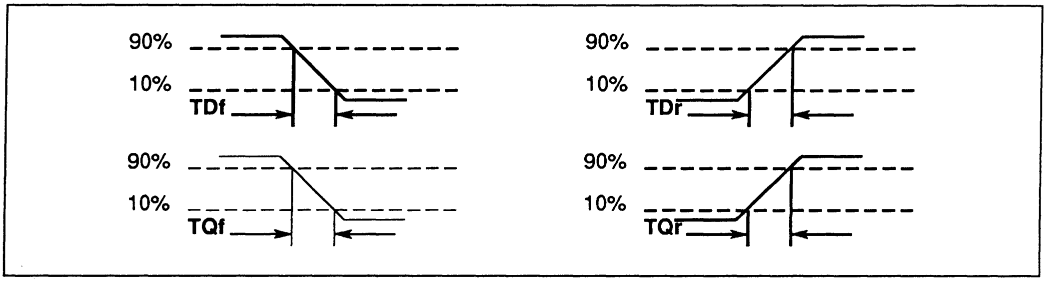


Figure 7.3 IMS C011 input and output edge timing

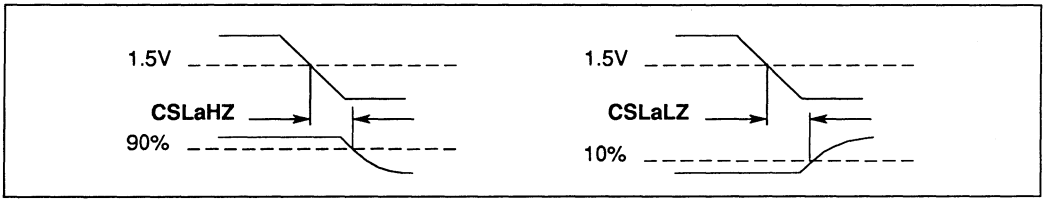


Figure 7.4 IMS C011 tristate timing relative to notCS

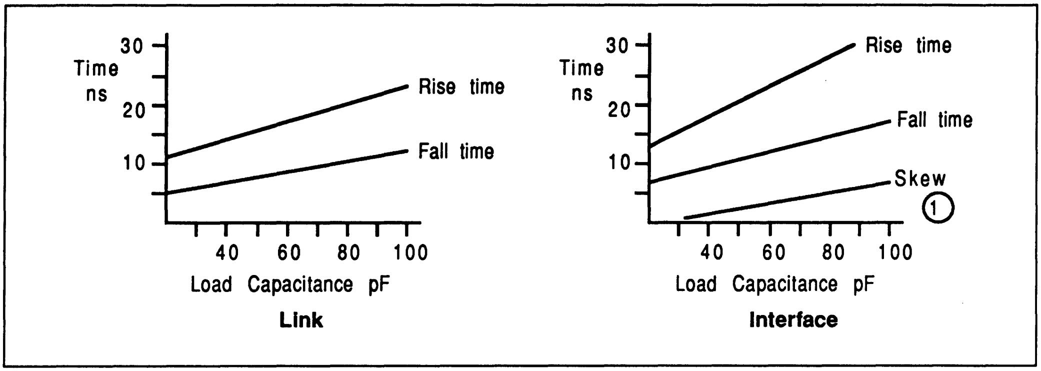


Figure 7.5 Typical rise/fall times

Notes

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

7.4 Power rating

Internal power dissipation P_{INT} of transputer and peripheral chips depends on **VCC**, as shown in figure 7.6. 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.

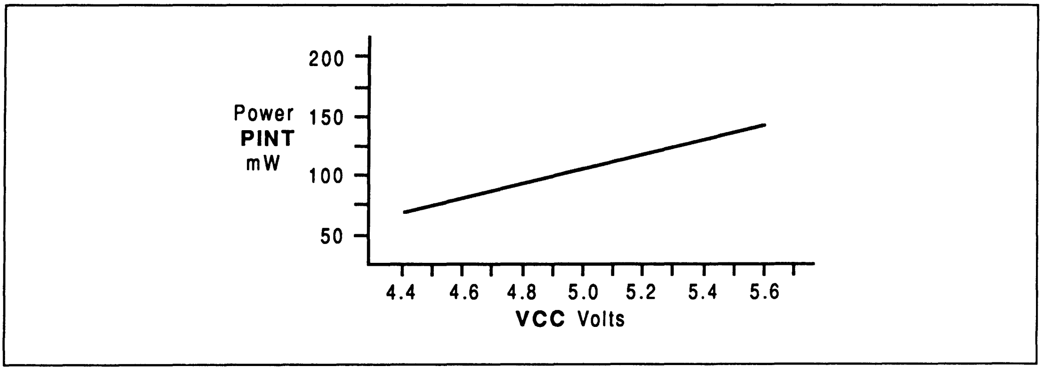


Figure 7.6 IMS C011 internal power dissipation vs VCC

8 Package specifications

8.1 28 pin plastic dual-in-line package

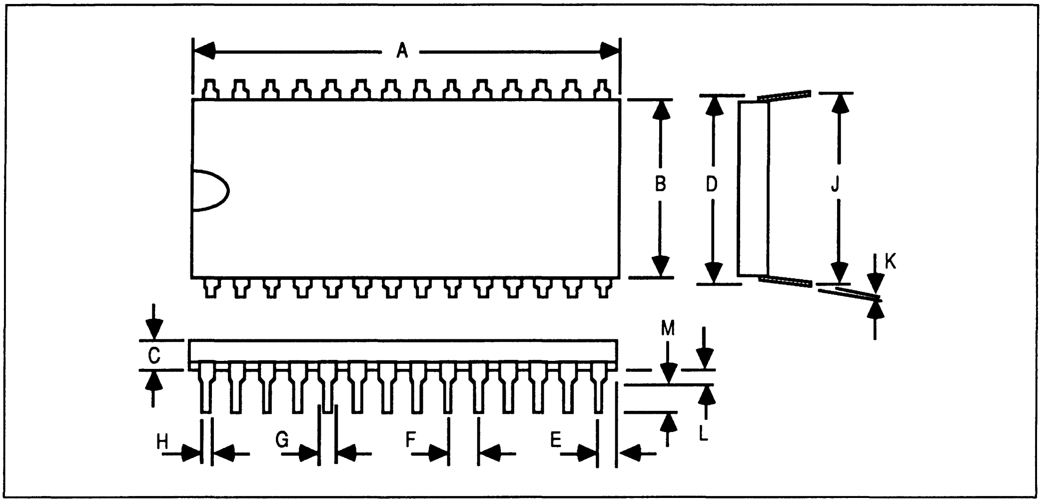


Figure 8.1 28 pin plastic dual-in-line package dimensions

Table 8.1 28 pin plastic dual-in-line package dimensions

	Millimetres		Inches		
DIM	NOM	TOL	NOM	TOL	Notes
A	36.830	+0.508 -0.254	1.450	+0.020 -0.010	
B	13.716	±0.051	0.540	±0.002	
C	3.810	±0.254	0.150	±0.010	
D	15.240	±0.076	0.600	±0.003	
E	1.905	±0.051	0.075	±0.002	
F	2.540	±0.051	0.100	±0.002	
G	1.524	±0.051	0.060	±0.002	
H	0.457	±0.051	0.018	±0.002	
J	16.256	0.508	0.640	0.020	
K	0.254	±0.025	0.010	±0.001	
L	0.58		0.020		Minimum
M	3.429		0.135		Maximum

Package weight is approximately 4 grams

Table 8.2 28 pin plastic dual-in-line package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow		110		°C/W	

8.2 28 pin ceramic dual-in-line package

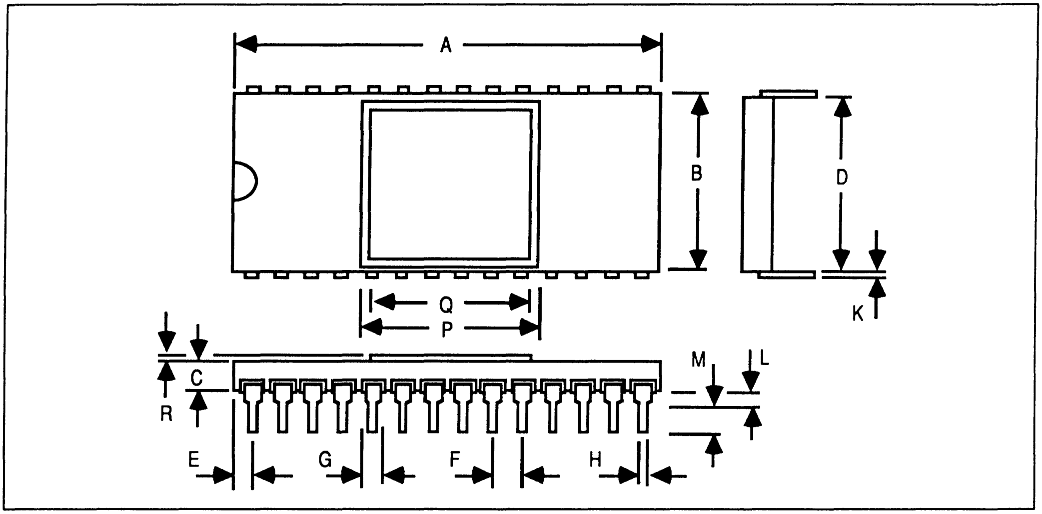


Figure 8.2 28 pin ceramic dual-in-line package dimensions

Table 8.3 28 pin ceramic dual-in-line package dimensions

	Millimetres		Inches		
DIM	NOM	TOL	NOM	TOL	Notes
A	35.560	±0.406	1.400	±0.016	
B	15.113	±0.254	0.595	±0.010	
C	2.159	±0.254	0.085	±0.010	
D	15.113	+0.762 -0.000	0.595	+0.030 -0.000	
E	2.032		0.080		Maximum
F	2.540	±0.127	0.100	±0.005	
G	1.270	±0.152	0.050	±0.006	
H	0.457	+0.102 -0.051	0.018	+0.004 -0.002	
K	0.254	+0.076 -0.025	0.010	+0.003 -0.001	
L	1.270	±0.381	0.050	±0.015	
M	3.683	±0.508	0.145	±0.020	
P	13.208	±0.229	0.520	±0.009	Square
Q	12.827	±0.127	0.505	±0.005	Square
R	0.3175	±0.0635	0.0125	±0.0025	

Package weight is approximately 5 grams

Table 8.4 28 pin ceramic dual-in-line package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow		60		°C/W	

8.3 Pinout

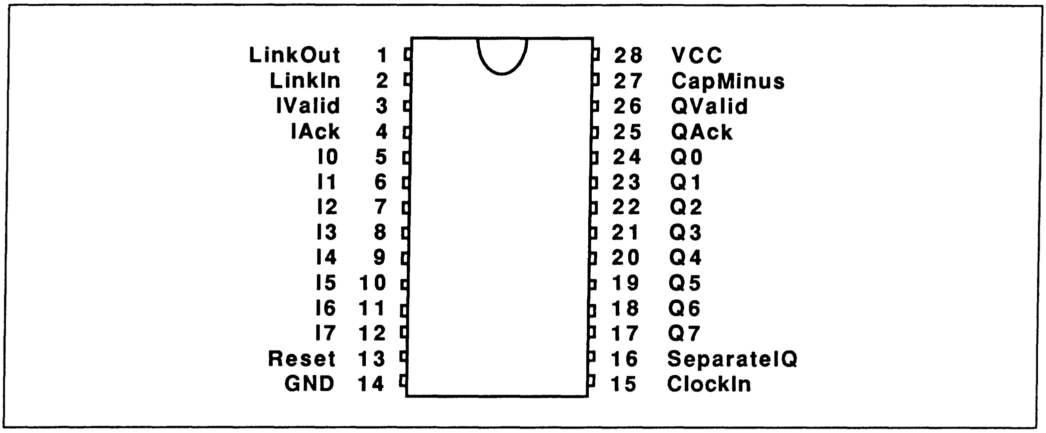


Figure 8.3 IMS C011 Mode 1 pinout

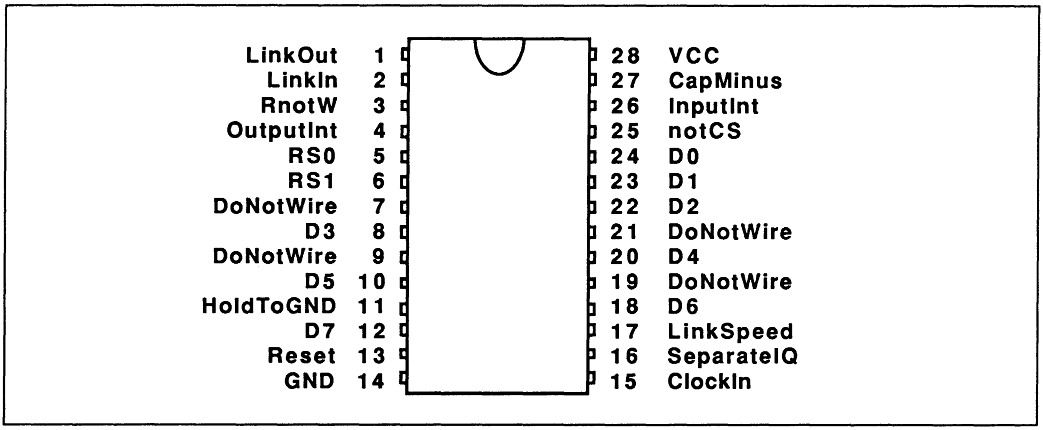


Figure 8.4 IMS C011 Mode 2 pinout

9 Ordering

This section indicates the designation of package selections for the IMS C011. Speed of **ClockIn** is 5 MHz for all parts.

For availability contact local INMOS sales office or authorised distributor.

Table 9.1 IMS C011 ordering details

INMOS designation	Package
IMS C011-P20S	28 pin plastic dual-in-line
IMS C011-S20S	28 pin ceramic sidebraze
IMS C011-S20M	28 pin ceramic sidebraze MIL Spec

IMS C012

link adaptor

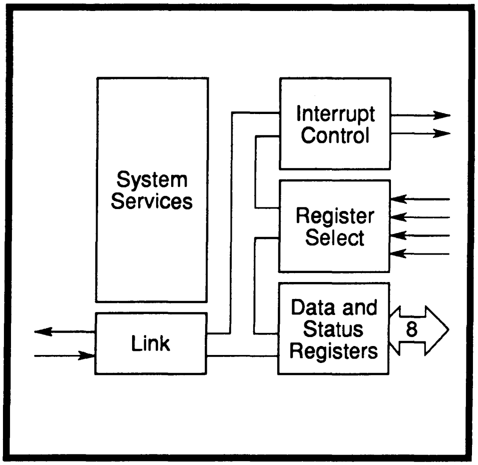
Engineering Data

FEATURES

Standard INMOS link protocol
10 or 20 Mbits/sec operating speed
Communicates with INMOS transputers
Converts between serial link and parallel bus
Tristate bidirectional bus interface
Memory mapped registers
Interrupt capability
Single +5V ±5% power supply
TTL and CMOS compatibility
120mW power dissipation
Standard 24 pin 0.3" plastic package

APPLICATIONS

Connecting microprocessors to transputers
High speed links between microprocessors
Inter-family microprocessor interfacing



1 Introduction

The INMOS communication link is a high speed system interconnect which provides full duplex communication between members of the INMOS transputer family, according to the INMOS serial link protocol. The IMS C012, a member of this family, provides for full duplex transputer link communication with standard microprocessor and sub-system architectures, by converting bi-directional serial link data into parallel data streams.

All INMOS products which use communication links, regardless of device type, support a standard communications frequency of 10 Mbits/sec; most products also support 20 Mbits/sec. Products of different type or performance can, therefore, be interconnected directly and future systems will be able to communicate directly with those of today. The IMS C012 link will run at either the standard speed of 10 Mbits/sec or at the higher speed of 20 Mbits/sec. Data reception is asynchronous, allowing communication to be independent of clock phase.

The IMS C012 provides an interface between an INMOS serial link and a microprocessor system bus. Status and data registers for both input and output ports can be accessed across the byte-wide bi-directional interface. Two interrupt outputs are provided, one to indicate input data available and one for output buffer empty.

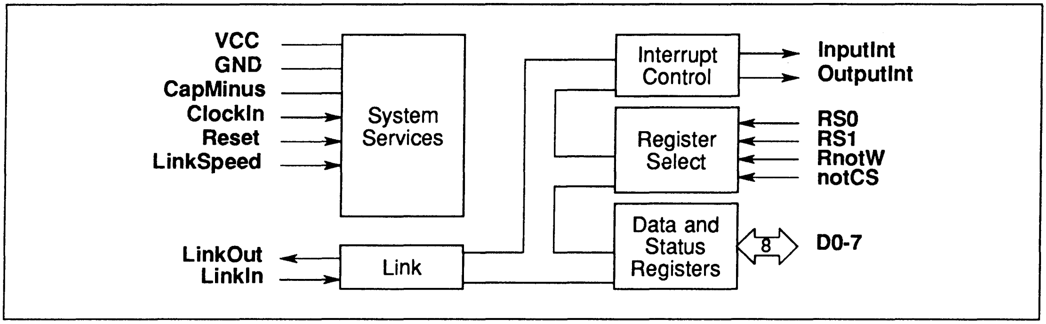


Figure 1.1 IMS C012 block diagram

2 Pin designations

Table 2.1 IMS C012 services and link

Pin	In/Out	Function
VCC, GND		Power supply and return
CapMinus		External capacitor for internal clock power supply
ClockIn	in	Input clock
Reset	in	System reset
LinkIn	in	Serial data input channel
LinkOut	out	Serial data output channel

Table 2.2 IMS C012 parallel interface

Pin	In/Out	Function
D0-7	in/out	Bi-directional data bus
notCS	in	Chip select
RS0-1	in	Register select
RnotW	in	Read/write control signal
InputInt	out	Interrupt on link receive buffer full
OutputInt	out	Interrupt on link transmit buffer empty
LinkSpeed	in	Select link speed as 10 or 20 Mbits/sec
HoldToGND		Must be connected to **GND**

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 431.

3 System services

System services include all the necessary logic to start up and maintain the IMS C012.

3.1 Power

Power is supplied to the device via the **VCC** and **GND** pins. 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.

AC noise between **VCC** and **GND** must be kept below 200 mV peak to peak at all frequencies above 100 KHz. AC noise between **VCC** and the ground reference of load capacitances must be kept below 200 mV peak to peak at all frequencies above 30 MHz. 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.

3.2 CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance 1μF capacitor to be connected between **VCC** 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 50 mm. The positive connection of the capacitor must be connected directly to **VCC**. Connections must not otherwise touch power supplies or other noise sources.

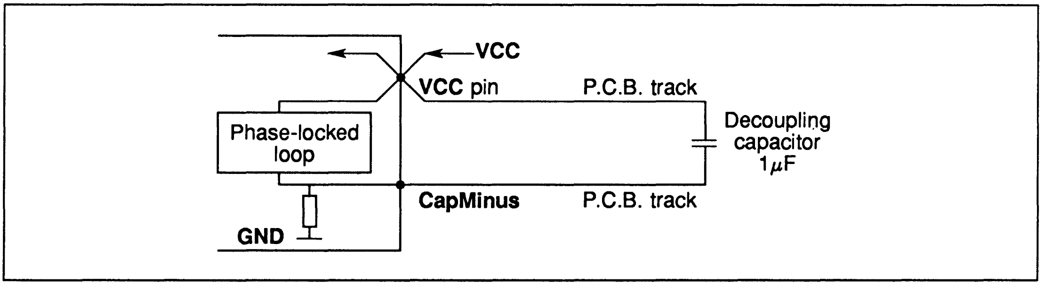


Figure 3.1 Recommended PLL decoupling

3.3 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 family 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 3.1 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	400	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 6.3).

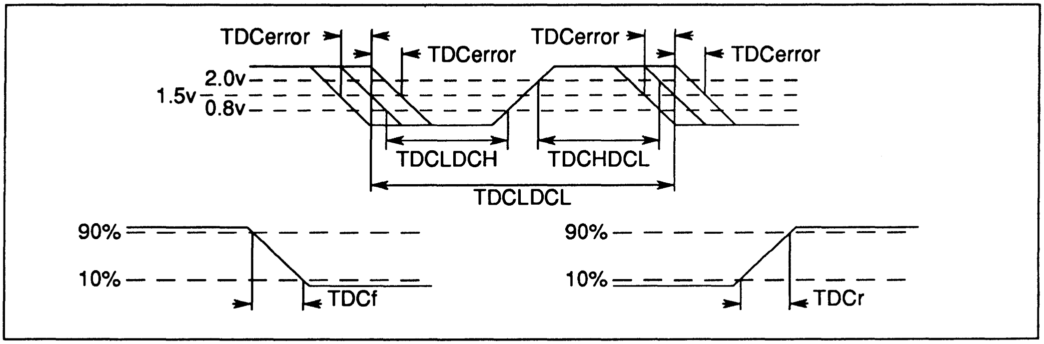


Figure 3.2 ClockIn timing

3.4 Reset

The **Reset** pin 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**. **LinkIn** must be held low during **Reset**.

Reset initialises the IMS C012 to the following state: **LinkOut** is held low; the interrupt outputs **InputInt** and **OutputInt** are held low; interrupts are disabled; **D0-7** are high impedance.

Table 3.2 Reset

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

Notes

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

2 At power-on reset.

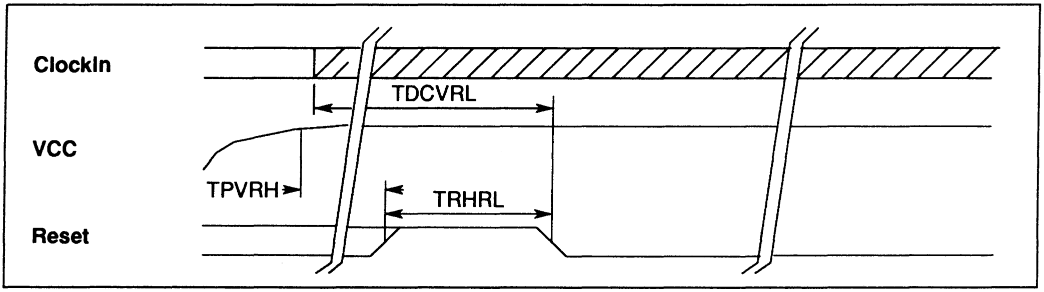


Figure 3.3 Reset Timing

4 Links

INMOS bi-directional serial links provide synchronized communication between INMOS products and with the outside world. Each link comprises an input channel and output channel. A link between two devices is implemented by connecting a link interface on one device to a link interface on the other device. 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.

Links are not synchronised with **ClockIn** and are insensitive to its phase. 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.

The IMS C012 link supports the standard INMOS communication speed of 10 Mbits/sec. In addition it can be used at 20 Mbits/sec. Link speed is selected by **LinkSpeed**; when the **LinkSpeed** pin is low, the link operates at the standard 10 Mbits/sec; when high it operates at 20 Mbits/sec.

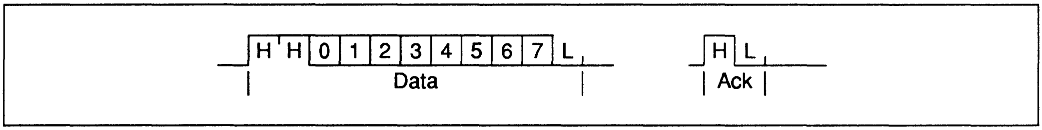


Figure 4.1 IMS C012 link data and acknowledge packets

Table 4.1 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
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.

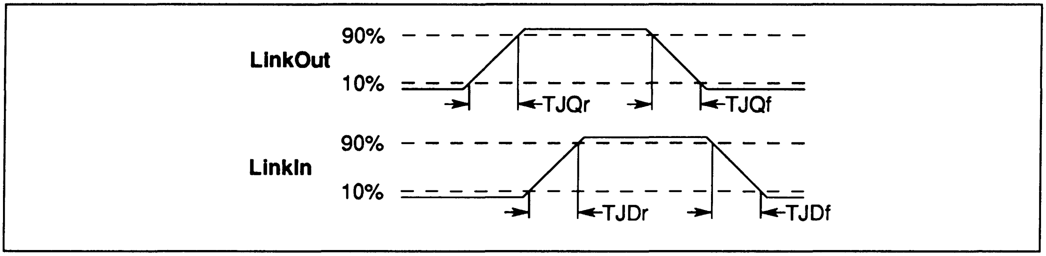


Figure 4.2 IMS C012 link timing

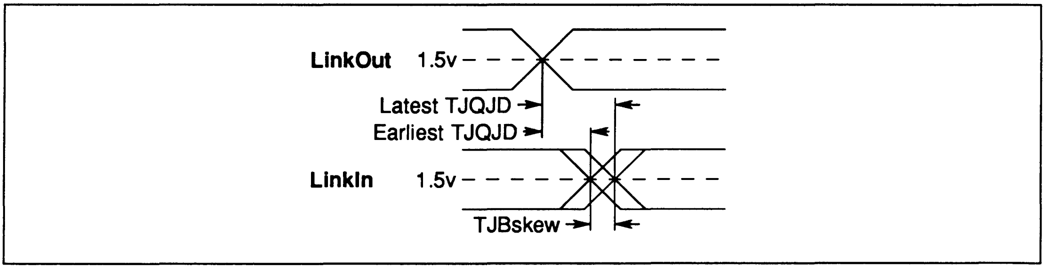


Figure 4.3 IMS C012 buffered link timing

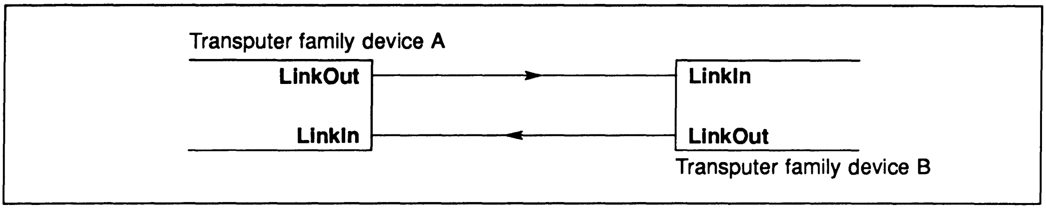


Figure 4.4 IMS C012 Links directly connected

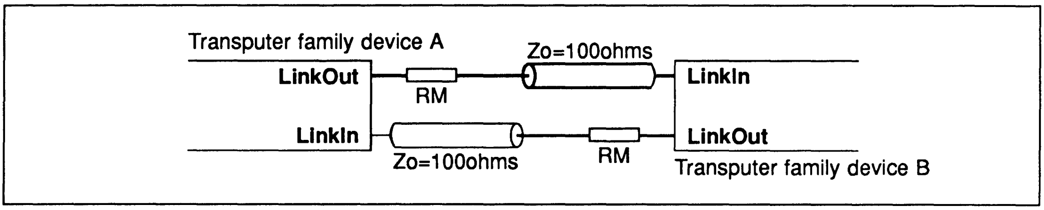


Figure 4.5 IMS C012 Links connected by transmission line

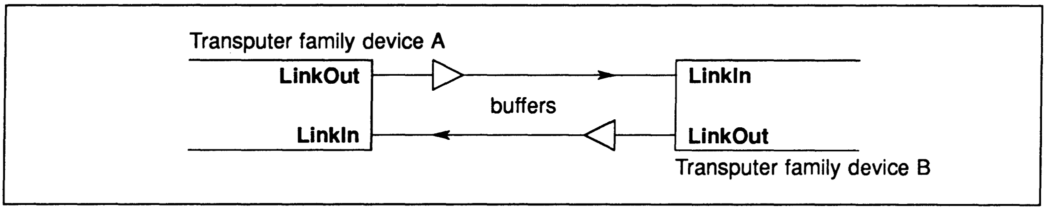


Figure 4.6 IMS C012 Links connected by buffers

5 Parallel interface

The IMS C012 provides an interface between a link and a microprocessor style bus. Operation of the link adaptor is controlled through the parallel interface bus lines **D0-7** by reading and writing various registers in the link adaptor. Registers are selected by **RS0-1** and **RnotW**, and the chip enabled with **notCS**.

For convenience of description, the device connected to the parallel side of the link adaptor is presumed to be a microprocessor, although this will not always be the case.

5.1 D0-7

Data is communicated between a microprocessor bus and the link adaptor via the bidirectional bus lines **D0-7**. The bus is high impedance unless the link adaptor chip is selected and the **RnotW** line is high. The bus is used by the microprocessor to access status and data registers.

5.2 notCS

The link adaptor chip is selected when **notCS** is low. Register selectors **RS0-1** and **RnotW** must be valid before **notCS** goes low; **D0-7** must also be valid if writing to the chip (**RnotW** low). Data is read by the link adaptor on the rising edge of **notCS**.

5.3 RnotW

RnotW, in conjunction with **notCS**, selects the link adaptor registers for read or write mode. When **RnotW** is high, the contents of an addressed register appear on the data bus **D0-7**; when **RnotW** is low the data on **D0-7** is written into the addressed register. The state of **RnotW** is latched into the link adaptor by **notCS** going low; it may be changed before **notCS** returns high, within the timing restrictions given.

5.4 RS0-1

One of four registers is selected by **RS0-1**. A register is addressed by setting up **RS0-1** and then taking **notCS** low; the state of **RnotW** when **notCS** goes low determines whether the register will be read or written. The state of **RS0-1** is latched into the link adaptor by **notCS** going low; it may be changed before **notCS** returns high, within the timing restrictions given. The register set comprises a read-only data input register, a write-only data output register and a read/write status register for each.

Table 5.1 IMS C012 register selection

RS1	RS0	RnotW	Register
0	0	1	Read data
0	0	0	Invalid
0	1	1	Invalid
0	1	0	Write data
1	0	1	Read input status
1	0	0	Write input status
1	1	1	Read output status
1	1	0	Write output status

5.4.1 Input Data Register

This register holds the last data packet received from the serial link. It never contains acknowledge packets. It contains valid data only whilst the *data present* flag is set in the input status register. It cannot be assumed to contain valid data after it has been read; a double read may or may not return valid data on the second read. If *data present* is valid on a subsequent read it indicates new data is in the buffer. Writing to this register will have no effect.

Table 5.2 IMS C012 parallel interface control

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRSVCSL	Register select setup	5			ns	
TCSLRSX	Register select hold	5			ns	
TRWVCSL	Read/write strobe setup	5			ns	
TCSLRWX	Read/write strobe hold	5			ns	
TCSLCSH	Chip select active	50			ns	
TCSHCSL	Delay before re-assertion of chip select	50			ns	

Table 5.3 IMS C012 parallel interface read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TLdVIIH	Start of link data to InputInt high			13	bits	1
TCSLIIL	Chip select to InputInt low			30	ns	
TCSLDrX	Chip select to bus active	5			ns	
TCSLDrV	Chip select to data valid			40	ns	
TCSHDrZ	Chip select high to bus tristate			25	ns	
TCSHDrX	Data hold after chip select high	5			ns	
TCSHLaV	Chip de-select to start of Ack	0.8		2	bits	1,2

Notes

1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 nS.

2 Maximum time assumes there is no data packet already on the link. Maximum time with data on the link is extended by 11 bits.

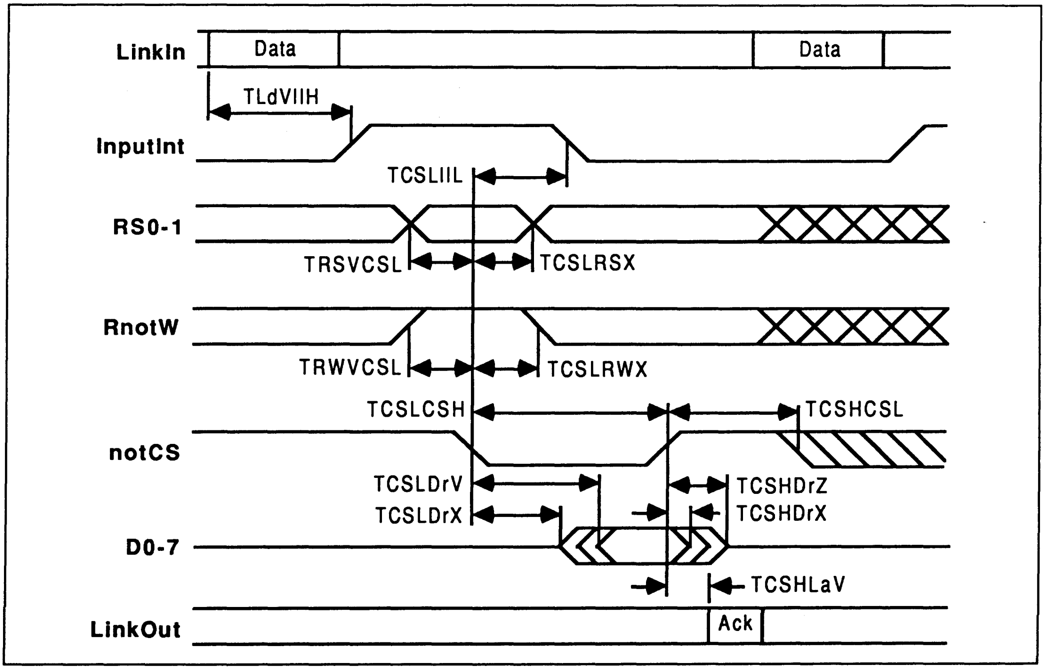


Figure 5.1 IMS C012 read parallel data from link adaptor

Table 5.4 IMS C012 parallel interface write

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TCSHDwV	Data setup	15			ns	
TCSHDwX	Data hold	5			ns	
TCSLOIL	Chip select to OutputInt low			30	ns	
TCSHLdV	Chip select high to start of link data	0.8		2	bits	1,2
TLaVOIH	Start of link Ack to OutputInt high			3	bits	1,3
TLdVOIH	Start of link data to OutputInt high			13	bits	1,3

Notes

1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 nS.

2 Maximum time assumes there is no acknowledge packet already on the link. Maximum time with acknowledge on the link is extended by 2 bits.

3 Both data transmission and the returned acknowledge must be completed before **OutputInt** can go high.

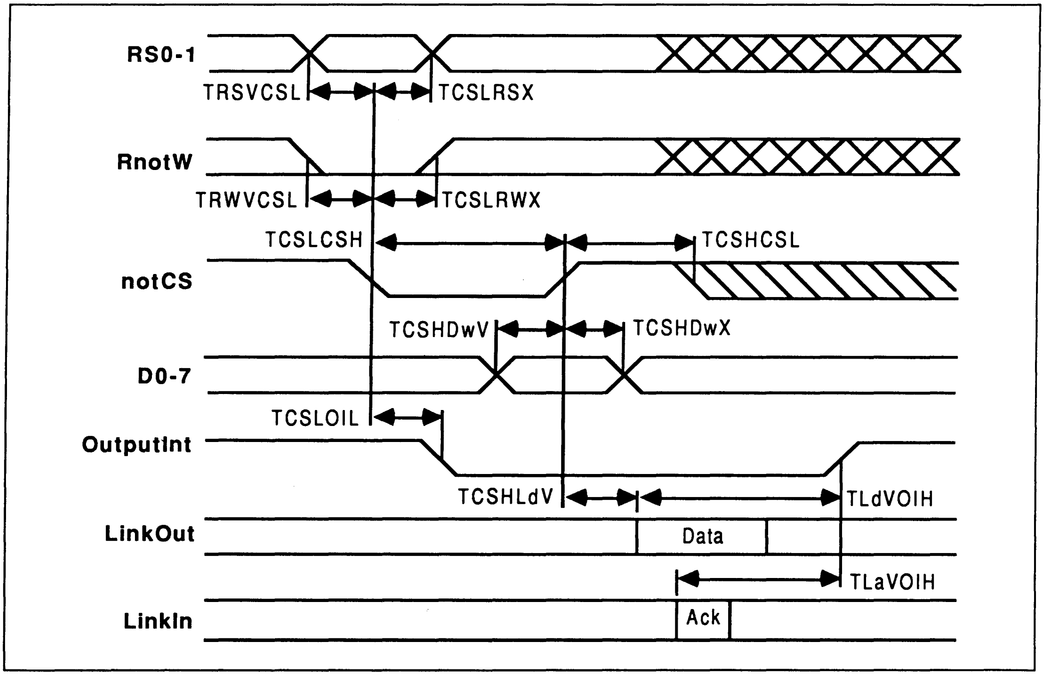


Figure 5.2 IMS C012 write parallel data to link adaptor

5.4.2 Input Status Register

This register contains the *data present* flag and the *interrupt enable* control bit for **InputInt**. The *data present* flag is set to indicate that data in the data input buffer is valid. It is reset low only when the data input buffer is read, or by **Reset**. When writing to this register, the *data present* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *data present* flags are both high, the **InputInt** output will be high (page 425). Resetting *interrupt enable* will take **InputInt** low; setting it again before reading the data input register will set **InputInt** high again. The *interrupt enable* bit can be read to determine its status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

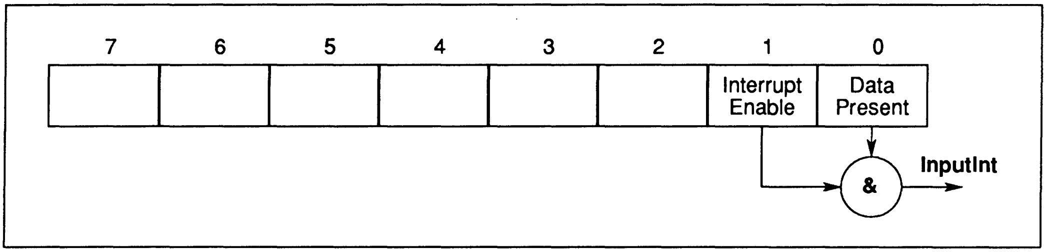


Figure 5.3 IMS C012 input status register

5.5 InputInt

The **InputInt** output is set high to indicate that a data packet has been received from the serial link. It is inhibited from going high when the *interrupt enable* bit in the input status register is low (page 425). **InputInt** is reset low when data is read from the input data register (page 422) and by **Reset** (page 418).

5.5.1 Output Data Register

Data written to this link adaptor register is transmitted out of the serial link as a data packet. Data should only be written to this register when the *output ready* bit in the output status register is high, otherwise data already being transmitted may be corrupted. Reading this register will result in undefined data being read.

5.5.2 Output Status Register

This register contains the *output ready* flag and the *interrupt enable* control bit for **OutputInt**. The *output ready* flag is set to indicate that the data output buffer is empty. It is reset low only when data is written to the data output buffer; it is set high by **Reset**. When writing to this register, the *output ready* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *output ready* flags are both high, the **OutputInt** output will be high (page 426). Resetting *interrupt enable* will take **OutputInt** low; setting it again whilst the data output register is empty will set **OutputInt** high again. The *interrupt enable* bit can be read to determine it's status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

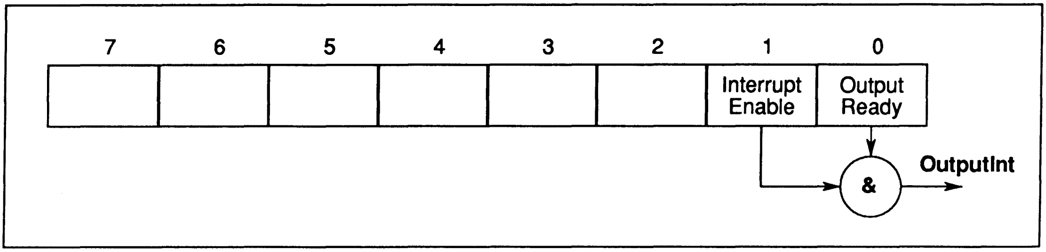


Figure 5.4 IMS C012 output status register

5.6 OutputInt

The **OutputInt** output is set high to indicate that the link is free to receive data from the microprocessor for transmission as a data packet out of the serial link. It is inhibited from going high when the *interrupt enable* bit in the output status register is low (page 425). **OutputInt** is reset low when data is written to the data output register (page 425); it is set high by **Reset** (page 418).

5.7 Data read

A data packet received on the input link sets the *data present* flag in the input status register. If the *interrupt enable* bit in the status register is set, the **InputInt** output pin will be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the input status register until the *data present* bit is high.

When data is available from the link, the microprocessor reads the data packet from the data input register. This will reset the *data present* flag and cause the link adaptor to transmit an acknowledge packet out of the serial link output. **InputInt** is automatically reset by reading the data input register; it is not necessary to read or write the input status register.

5.8 Data write

When the data output buffer is empty the *output ready* flag in the output status register is set high. If the *interrupt enable* bit in the status register is set, the **OutputInt** output pin will also be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the output status register until the *output ready* bit is high.

When the *output ready* flag is high, the microprocessor can write data to the data output buffer. This will result in the link adaptor resetting the *output ready* flag and commencing transmission of the data packet out of the serial link. The *output ready* status bit will remain low until an acknowledge packet is received by the input link. This will set the *output ready* flag high; if the *interrupt enable* bit is set, **OutputInt** will also be set high.

6 Electrical specifications

6.1 DC electrical characteristics

Table 6.1 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		600	mW	

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 6.2 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 6.3 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	1,2,7
			±200	μA	1,2,8
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		120	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 C012-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.

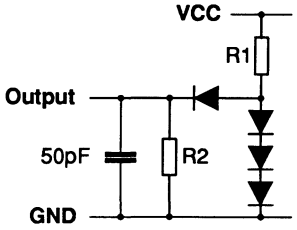
5 Power dissipation varies with output loading.

6 This parameter is sampled and not 100% tested.

7 For inputs other than those in Note 8.

8 For pins 2, 3, 5, 6, 7, 9, 11, 13, 14, 21.

6.2 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 6.1 Load circuit for AC measurements

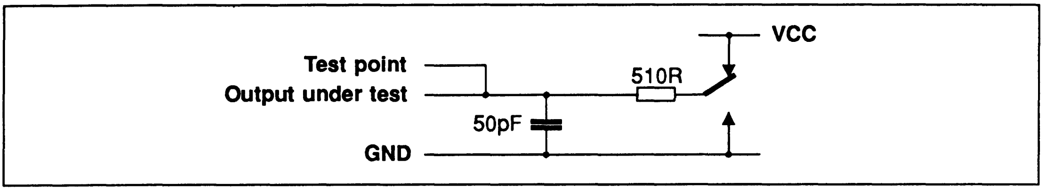


Figure 6.2 Tristate load circuit for AC measurements

6.3 AC timing characteristics

Table 6.4 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
CSLaHZ	Chip select high to tristate		25	ns	
CSLaLZ	Chip select low to tristate		25	ns	

Notes

1 Non-link pins; see section on links.

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

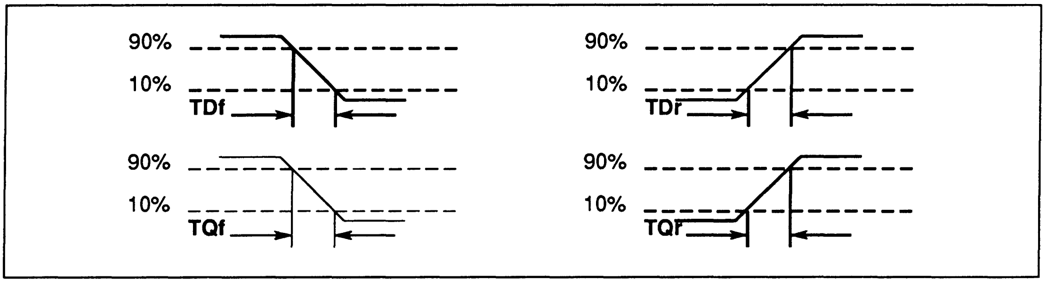


Figure 6.3 IMS C012 input and output edge timing

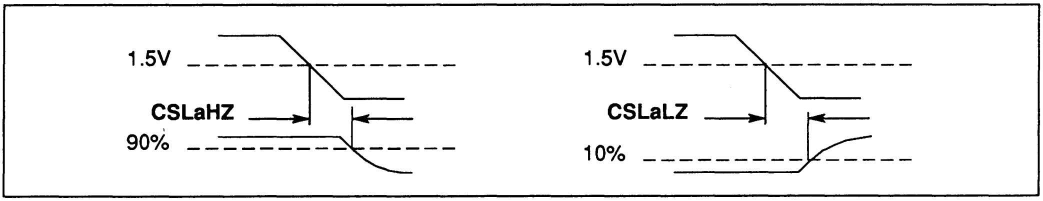


Figure 6.4 IMS C012 tristate timing relative to notCS

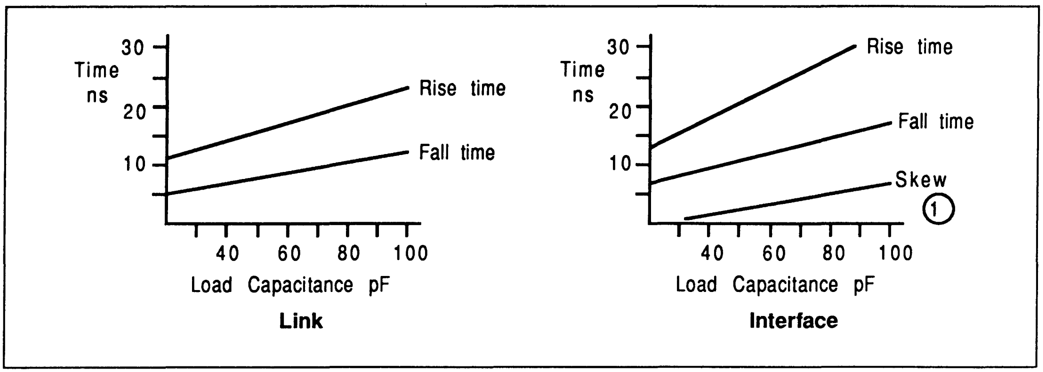


Figure 6.5 Typical rise/fall times

Notes

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

6.4 Power rating

Internal power dissipation P_{INT} of transputer and peripheral chips depends on **VCC**, as shown in figure 6.6. 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.

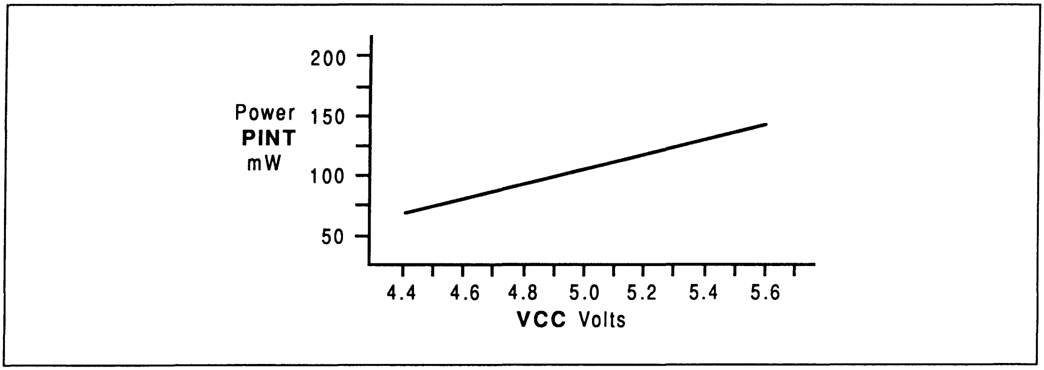


Figure 6.6 IMS C012 internal power dissipation vs VCC

7 Package specifications

7.1 24 pin plastic dual-in-line package

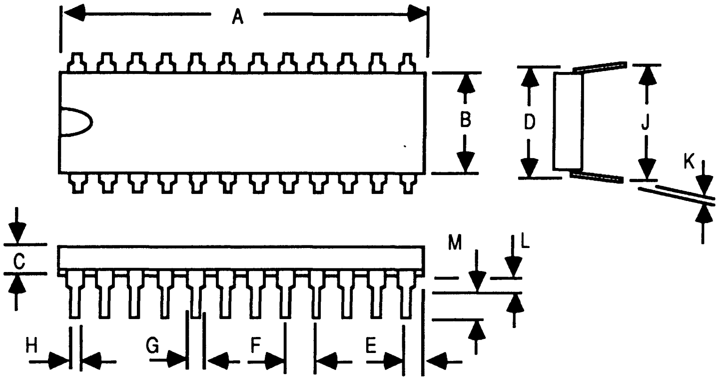


Figure 7.1 24 pin plastic dual-in-line package dimensions

Table 7.1 24 pin plastic dual-in-line package dimensions

	Millimetres		Inches		
DIM	**NOM**	**TOL**	**NOM**	**TOL**	**Notes**
A	31.242	+0.508 -0.254	1.230	+0.020 -0.010	
B	6.604	±0.127	0.260	±0.005	
C	3.302	±0.381	0.130	±0.015	
D	7.620	±0.127	0.300	±0.005	
E	1.651	±0.127	0.065	±0.005	
F	2.540	±0.127	0.100	±0.005	
G	1.524	±0.127	0.060	±0.005	
H	0.457	±0.127	0.018	±0.005	
J	8.382	±0.508	0.330	±0.020	
K	0.254	±0.025	0.010	±0.001	
L	0.508	±0.127	0.020	±0.005	
M	3.048		0.120		Minimum

Package weight is approximately 2 grams

Table 7.2 24 pin plastic dual-in-line package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow		115		°C/W	

7.2 Pinout

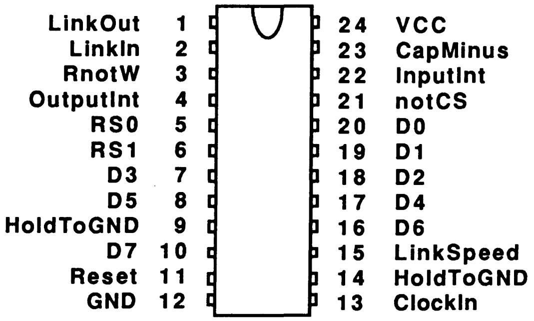


Figure 7.2 IMS C012 pinout

8 Ordering

This section indicates the designation of package selections for the IMS C012. Speed of **ClockIn** is 5 MHz for all parts.

For availability contact local INMOS sales office or authorised distributor.

Table 8.1 IMS C012 ordering details

INMOS designation	Package
IMS C012-P20S	24 pin plastic dual-in-line

MIL specification

A Military standard program

For full details see INMOS document 49-9047 “Military General Processing Specification”.

The INMOS military program is designed to provide class B microcircuits in accordance with 1.2.1 of MIL-STD-883, “Provisions for the use of MIL-STD-883 in conjunction with compliant non-JAN devices”. Devices are processed for general applications where component quality and reliability must conform to the guidelines and objectives of military procurement. Suitability for use in specific applications should be determined using the guidelines of MIL-STD-454.

Screening procedures are compliant with Method 5004 and the provisions of paragraph 3.3 therein. Quality conformance procedures are compliant with Method 5005 using the alternate Group B provisions of paragraph 3.5.2. All electrical testing is performed to guarantee operation at -55°C, +25°C and +125°C.

All INMOS military grade components are provided in hermetically sealed ceramic packages.

By specifying an INMOS military product, the user can be assured of receiving a product manufactured, tested and inspected in compliance with MIL-STD-883 and one with superior performance for those applications where quality and reliability are of the essence.

For more details on the military version of the devices, including characterisation details, contact your local INMOS sales outlet.

Table A.1 Expression evaluation

100 Percent Process Step	MIL-STD-883C Method	Test Condition	Comment
Internal visual	2010	B	
Stabilization bake	1008	C	
Temperature cycle	1010	C	
Constant acceleration	2001	D	Y-1 axis
Seal test	1014	B	
Seal test	1014	C	
Visual inspection			INMOS 89-1001
Pre burn-in electrical			+25°C data sheet
Burn-in	1015	D	
Post burn-in electrical			+25°C data sheet
PDA			5% max
Final electrical			+125°C data sheet
Final electrical			-55°C data sheet
External visual	2009		
Group A	5005	3.5.1	A1-A11
Group B	5005	3.5.2	
Group C	5005		MIL-STD-883C 1.2.1.b.17
Group D	5005		MIL-STD-883C 1.2.1.b.17

84 lead quad cerpack

B 84 lead quad cerpack

Product under development - for availability contact INMOS.

In the future some transputers will be available in a surface mount quad cerpack package. The leads will be unformed to allow the user to form them to specific requirements.

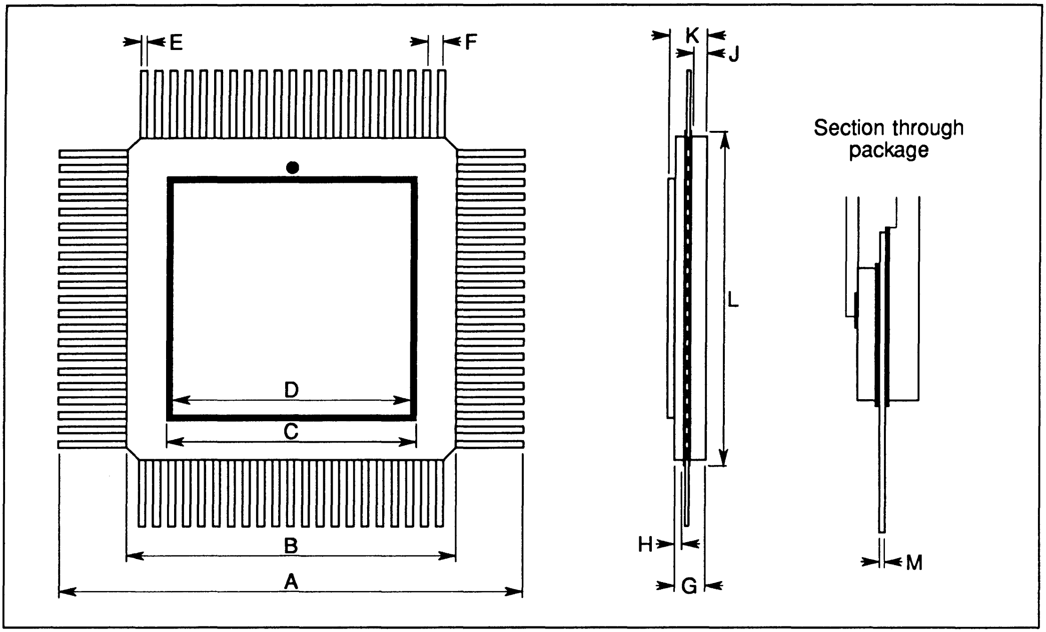


Figure B.1 84 lead quad cerpack package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	38.100	±0.508	1.500	±0.020	
B	26.924	±0.305	1.060	±0.012	
C	20.574	±0.203	0.810	±0.008	
D	19.558	±0.254	0.770	±0.010	
E	0.508		0.020		
F	1.270	±0.051	0.050	±0.002	
G	2.489	±0.305	0.098	±0.012	
H	0.635	±0.076	0.025	±0.003	
J	1.143	±0.102	0.045	±0.004	
K	3.099		0.122		Max.
L	27.940		1.100		Max.
M	0.178	±0.025	0.007	±0.001	

Table B.1 84 lead quad cerpack package dimensions

quality and reliability

C Quality and Reliability

The INMOS Quality Programme is attentive to every phase of the semiconductor product life cycle. This includes specific programmes in each of the following areas:

- Total Quality Control (TQC)
- Quality and Reliability in Design
- Document Control
- New Product Qualification
- Product Monitoring Programme
- Production Testing and Quality Monitoring Procedure

C.1 Total Quality Control (TQC) and Reliability Programme

Our objective to continuously build improved quality and reliability into every INMOS part has resulted in a comprehensive Quality/Reliability Programme that we are proud of. This programme demonstrates INMOS' serious commitment to supporting the quality and reliability needs of the electronics marketplace.

INMOS is systematically shifting away from a traditional screening approach to quality control and towards one of building in Experimental Design quality through Statistical Process Control (SPC). This new direction was initiated with a vigorous programme of education and scientific method training.

In the first year of the programme approximately 80 INMOS employees worldwide received thorough SPC training. In 1987 this training was extended to cover advanced SPC and experimental design. Some of the courses taught are listed below:

- Experimental Design Techniques
- Statistical Process Control Methods
- Quality Concepts
- Problem Solving Techniques
- Statistical Software Analysis Techniques

Today INMOS utilizes experimental design techniques and process control/monitoring throughout its development and manufacturing cycles. The following TQC tools are currently supported by extensive databases and analysis software.

1. Pareto charts
2. Cause/Effect Diagrams
3. Process Flow Charts
4. Run Charts
5. Histograms
6. Correlation Plots
7. Control Charts
8. Experimental Design
9. Process Capability Studies

C.2 Quality and Reliability in Design

The INMOS quality programme begins with the design of new INMOS products. The following procedures are examples from the INMOS programme to design quality and reliability into every product.

Innovative design techniques are employed to achieve product performance using, whenever possible, state of the art techniques. For example, INMOS uses 300 nanometre gate oxides on its high performance static SRAM and MICRO products to obtain the reliability inherent in the thicker gate oxide. In addition, circuit design engineers work hand in hand with process engineers to optimise the design for the process and the process for the product family. The result is a highly reliable design implemented in a process technology achievable within manufacturing.

INMOS products are designed to have parametric margins beyond the product target specifications. The design performance is verified using simulations of circuit performance over voltage and temperature values beyond those of specified product operation, including verification beyond the military performance range. In addition, the device models are chosen to ensure tolerance to wide variations in process parameters beyond those expected in manufacture.

The design process includes consideration of quality issues such as signal levels available for sensing, reduction of internal noise levels, stored data integrity and testability of all device functions. Electro-static damage protection techniques are included in the design with input protection goals of 2K volts for MIL-STD-883 testing methods. Specific customer requirements can be met by matching their detailed specifications against INMOS designed in margins.

The completion of the design includes the use of INMOS computer aided design software to fully check and verify the design and layout. This improves quality as well as ensuring the timely introduction of new products.

C.3 Document Control

The Document Control Department maintains control over all manufacturing specifications, lot travellers, procurement specifications and drawings, reticle tapes and test programmes. New specifications and changes are subject to approval by the Engineering and Manufacturing managers or their delegates. Change is rigorously controlled through an Engineering Change Notice procedure, and QA department managers screen and approve all such changes.

An extensive archiving system ensures that the history of any Change Notice is readily available.

Document Control also has responsibility for controlling in-line documentation in all manufacturing areas which includes distribution of specifications, control of changes and liaison with production control and manufacturing in introducing changed procedures into the line.

Extensive use is made of computer systems to control documentation on an international basis.

C.4 New Product Qualification

INMOS performs a thorough internal product qualification prior to the delivery of any new product, other than engineering samples of prototypes to customers.

Care is taken to select a representative sample from the final prototype material. This typically consists of three different production lots. Testing is then done to assure the initial product reliability levels are achieved. Product qualifications are done in accordance with MIL-STD-883, methods 5004 and 5005, or CECC/BS9000.

The initial INMOS qualification data, and the ongoing monitor data can be very useful in the user qualification decision process. INMOS also has a very successful history of performing customer qualification testing in-house and performing joint qualification programmes with customers. INMOS remains committed to joint customer/vendor programmes.

C.5 Product Monitoring Programme

At the levels of quality and reliability performance required today (low PPM and FIT levels), it is essential that a large statistically significant, current product database be maintained. One of the programmes that INMOS uses to accomplish this is the Product Monitoring Programme (PMP).

The PMP is comprehensive ongoing programme of reliability testing. A small sample is pulled from production lots of a particular part type. This population is then used to create the specific samples to put on the various operating and environmental tests. Tests run in this programme include extended temperature operating life, THB and temperature cycle. Efforts are continuing to identify and correlate more accelerated tests to be used in the PMP.

C.6 Production Testing and Quality Monitoring Procedure

C.6.1 Reliability Testing

INMOS' primary reliability test method is to bias devices at their maximum rated operating power supply level in a 140° C ambient temperature. A scheme of time varying input signals is used to simulate the complete functional operation of the device. The failure rate is then computed from the results of the operating life test using Arrhenius modelling for each specific failure mechanism known. The failure rate is reported at a temperature that is a typical worst case application environment and is expressed in units of FITs where 1 FIT = 1 Fail in 10E9 device hours, (100 FIT = 0.01%/1000 Hrs). The current database enables the failure rate to be valid over various environmental conditions.

The failure rate goal for INMOS products is 100 FITs or less at product introduction with a 50 FIT level to be attained within one year.

For plastic packaged product, additional testing methods and reliability indices become important. Humidity testing is used to evaluate the relative hermeticity of the package, and thermal cycling tests are used principally to evaluate the durability of the assembly (e.g. die/bond attach).

The Humidity Test comprises of temperature, humidity bias (THB) at 85°C, 85% Relative Humidity, and a 5V static bias configuration selected to maintain the component in a state of minimum power dissipation and enhance the formation of galvanic corrosion. INMOS reliability goals have always been to meet or better the current 'industry standards' and a target of less than 1% failures through 1000 hours of THB at 90% confidence has been set.

The Thermal Cycling tests are performed from −65°C to + 150 °C for 500 - 1000 cycles, with no bias applied. Thermal Shock tests using a liquid to liquid (Freon) method are cycled between −55°C and + 125 °C. The INMOS Reliability qualification and monitoring goal for the above tests is less than 1at 90

C.6.2 Production Testing

Electrical testing at INMOS begins while the devices are still in wafer form before being divided into individual die. While in this form, two different types of electrical test are performed.

The Parametric Probe test is to verify that the individual component parameters are within their design limits. This is accomplished by testing special components on the wafer. The results of these tests provide feedback to our wafer fab manufacturing facilities which allows them to ensure that the components used in the actual devices perform within their design limits. This testing is performed on all lots which are processed, and any substandard wafers being discarded. These components are placed in the scribe streets of the wafer so they are destroyed in the dicing operation when they are not of any further use. By placing them there, valuable chip real estate is saved, thereby holding down cost while still providing the necessary data.

The Electrical Probe test performed on all wafers is the test of each individual circuit or chip on every wafer. The defective die are identified so they may be later discarded after the wafer has been separated into individual die. This test fully exercises the circuits for all AC and DC datasheet parameters in addition to verifying functionality.

After the die have been assembled into packages they are again tested in our Final Test operation. In a mature product the typical flow is:

- Preburn-in test
- Burn-in @ 140°C
- Final test

- PDA (Percent Defect Allowed)
- Device Symbolisation
- QA Final Acceptance

The temperature setting used for hot testing is selected so that the junction temperature is the same as it would be after thermal stabilisation occurred in the specified environment. This is calculated using the hot temperature power dissipation along with the thermal resistance of the package used. All INMOS product is electrically tested and burned-in prior to shipment. Historically, the industry has selected burn-in times using the MIL Standards as a guide (when the market would support the cost) or on a 'best guess' basis dominated by cost considerations. Whereas INMOS invoke a burn-in reduction exercise to ensure the reduced time has no reliability impact.

C.6.3 Quality Monitoring Procedure

In the Outgoing Quality Monitoring programme, random samples are pulled from lots, that have been successfully tested to data sheet criteria. Rejected lots are 100% retested and more importantly, failures are analysed and corrective actions identified to prevent the recurence of specific problems.

The extensive series of electrical tests with the associated Burn-in PDA limits and Quality Assurance tests ensure we will be able to continue to improve our high quality and reliability standards.

bibliography

D Bibliography

This appendix contains a list of some transputer-related publications which may be of interest to the reader. The References section details publications referred to in this manual, other than the standard INMOS documents detailed below.

INMOS publish manuals and data sheets pertaining to transputer based products and to occam. Apart from items detailed below, INMOS produce an Engineering Data Sheet for each product, as well as Product Information Guides detailing the INMOS range of products. There are also a number of Technical Notes and Application Notes available from INMOS, covering a wide range of topics in both the hardware and software areas.

D.1 INMOS publications

INMOS
occam *Programming Manual*
72 OCC 040

INMOS
occam *2 Language Definition*
72 OCC 044

INMOS
IMS M212 Product Data manual
72 TRN 103

INMOS
Transputer Development System Manual
Prentice Hall
ISBN 0-13-928995-X

INMOS
The Transputer Instruction Set - A Compiler Writers' Guide
Prentice Hall
ISBN 0-13-929100-8

INMOS
Communicating Process Architecture
Prentice Hall
ISBN 0-13-629320-4

INMOS
Transputer Technical Notes
Prentice Hall
ISBN 0-13-929126-1

INMOS
occam *2 Reference Manual*
Prentice Hall
ISBN 0-13-629312-3

INMOS
occam
Keigaku Shuppan Publishing Company
ISBN 4-7665-0133-0
(In Japanese)

D.2 INMOS technical notes

A wide range of technical and application notes are published by INMOS. Some of these are listed in this section. Please note that the Prentice Hall publications *Communicating Process Architecture* and *Transputer Technical Notes* are compilations of suitable technical notes listed below.

Number	Title
72 TCH 000	A transputer based radio navigation system
72 TCH 001	Extraordinary use or transputer links
72 TCH 002	A Transputer based navigation system - Testing embedded systems
72 TCH 005	Program design for concurrent systems
72 TCH 006	IMS T800 Architecture
72 TCH 007	Exploiting concurrency; a ray tracing example
72 TCH 009	Designing with the T414 and IMS T800 memory interface
72 TCH 010	IMS B003 Design of a multi-transputer board
72 TCH 017	Performance maximisation
72 TCH 018	Connecting INMOS links
72 TCH 019	Designs and applications for the IMS C004 link switch
72 TCH 020	Communicating processes and occam
72 TCH 021	The transputer implementation of occam
72 TCH 022	Communicating process computers
72 TCH 023	Compiling occam into silicon
72 TCH 024	Exploring multiple transputer arrays
72 TCH 026	Graphics support and performance improvements on the IMS T800
72 TCH 027	Lies, damned lies, and benchmarks
72 TCH 029	TRAM module description
72 TCH 032	Security aspects of occam 2
72 TCH 036	The INMOS flight simulator
72 TCH 037	High performance graphics with the IMS T800
72 TCH 046	Transputer based graphics display
72 TCH 047	The role of occam in the design of the IMS T800
72 TCH 049	Module motherboard architecture
72 TCH 051	Simple real time programming with the transputer

D.3 References

This section details publications referred to in the TRANSPUTER OVERVIEW chapter of this document.

Harp
Phase 1 of the development and application of a low cost, high performance multiprocessor machine
J G Harp et al
ESPRIT '86: Results and Achievements, Elsevier Science Publishers B.V.
pp 551-562

IEEE
IEEE Standard for Binary Floating-Point Arithmetic
ANSI/IEEE Std 754-1985.

INMOS '84
IMS T414 reference manual
INMOS Limited 1984

INMOS '88
Transputer Instruction Set - A Compiler Writers' Guide
Prentice Hall
ISBN 0-13-929200-8

McMahon
The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range
F H McMahon
Lawrence Livermore National Laboratory
UCRL-53745

A number of papers have been written by external authors, and a bibliography of these is included in the occam *users group* newsletter. For details on joining this group contact your local INMOS outlet.

index

E Index