inmos®

Reference manual

transputer

January 1987

Contents

transputer architecture

This manual describes the architecture of the transputer family of products. The first section gives a brief summary of the major features of the transputer architecture and a rationale. The second section gives an overview of the programming model. The third section describes the aspects of the transputer which are common to all members of the transputer family.

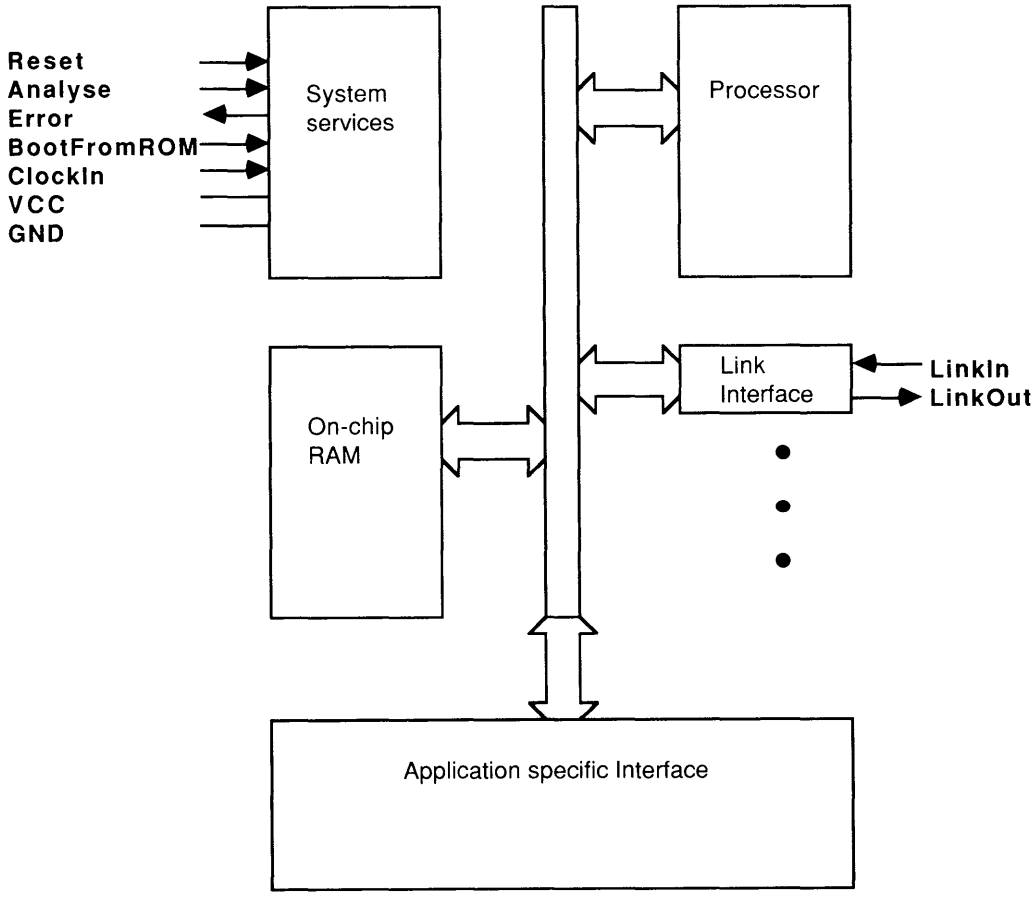
Product data manuals, describing the individual products, are combined with this manual.

Other information relevant to all transputer products is contained in the occam programming manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

The 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.

This edition of the manual is dated October 27, 1986.

Transputer Architecture



1.1 Overview

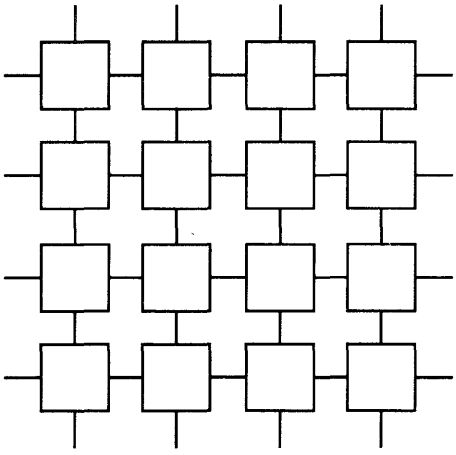
A transputer is a microcomputer with its own local memory and with links for connecting one transputer to another transputer.

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 is easily constructed using point-to-point communication



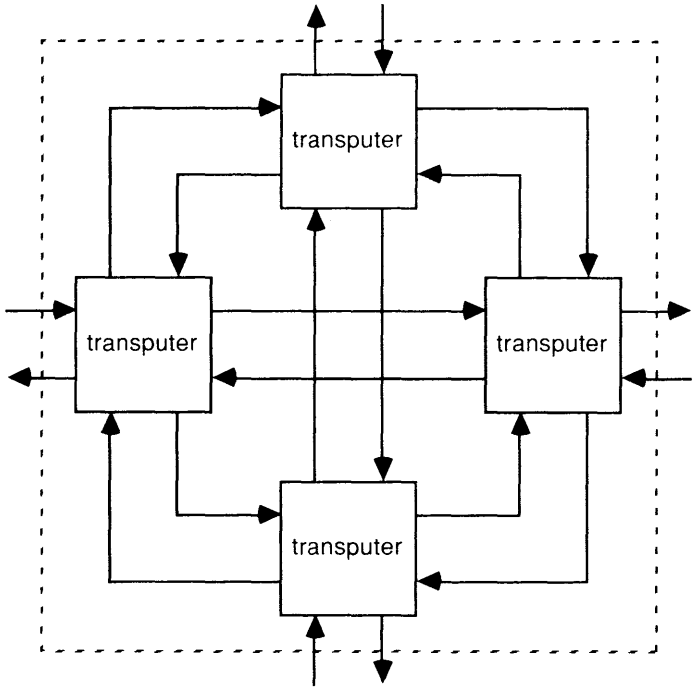
Transputers and occam

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. 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.

A node of four transputers



1.2 Rationale

1.2.1 System design

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.

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 (see section 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.

Architecture

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.

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, preprogrammed 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.

1.2.2 Systems architecture

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.

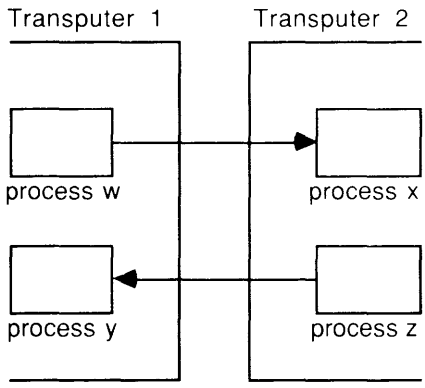
Local memory

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.

1.2.3 Communication

INMOS links provide direct communication between processes on individual transputers



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

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.

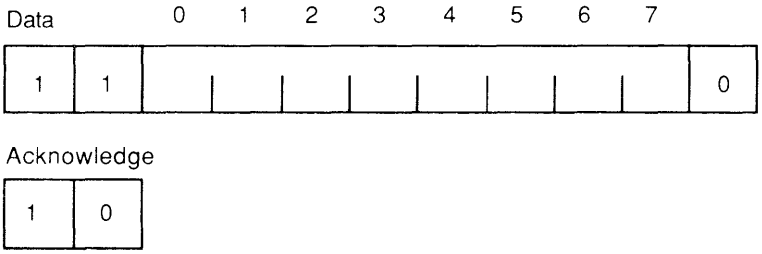
The two signal wires of the link can be used to provide two occam channels, one in each direction. This requires a simple protocol. Each signal line carries data and control information.

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.

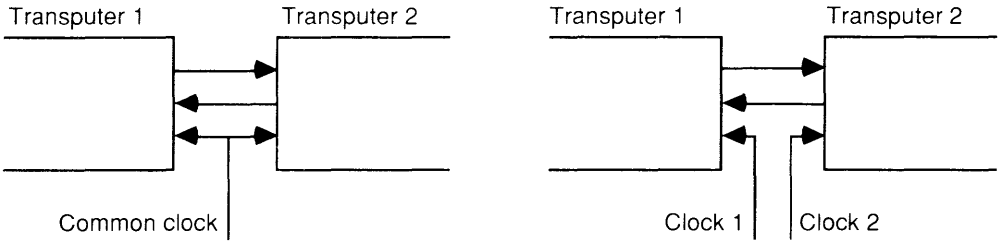
Data bytes and acknowledges are multiplexed down each signal line. An acknowledge can be transmitted as soon as reception of a data byte starts (if there is room to buffer another one). Consequently transmission may be continuous, with no delays between data bytes.

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.

Clocking of 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.

The programming model for transputers is defined by occam.

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.

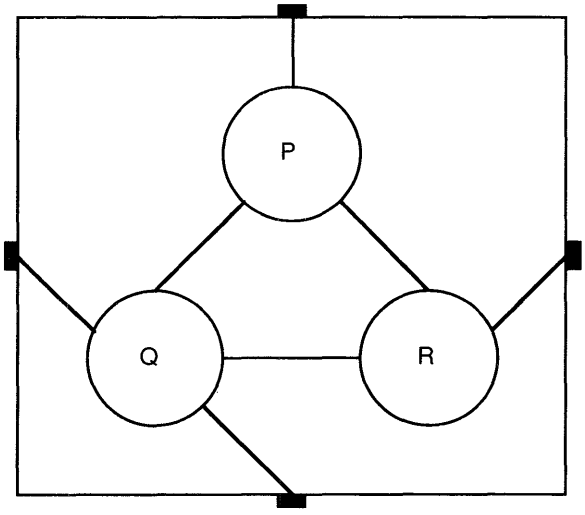
Ultimately, all processes are constructed from three primitive processes - assignment, input and output. An assignment computes the value of an expression and sets a variable to the value. Input and output are used for communicating between processes. A pair of concurrent processes communicate using 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.

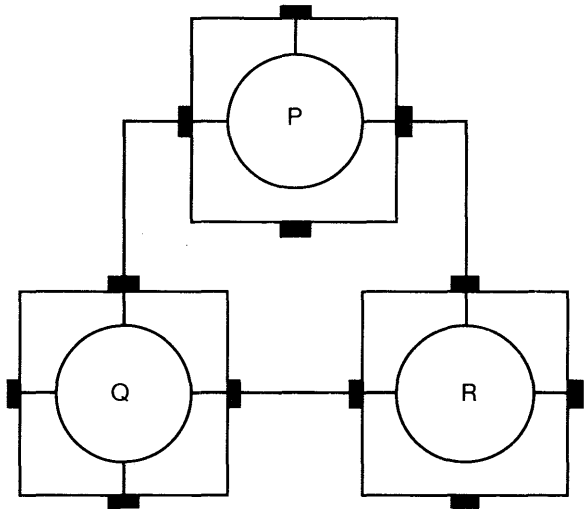
Since a process may have internal concurrency, it may have many input channels and output channels performing communication at the same time.

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.

Mapping processes onto one or several transputers



A program on a single transputer



The same program on three transputers

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.

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 Constructs

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.

SEQ

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.

PAR

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.

IF

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.

ALT

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.

2.2.3 Repetition

```
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.

2.2.4 Replication

A replicator is used with a constructor 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.

2.2.5 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.

Primitive types

All implementations provide the primitive types.

CHAN OF *type* Each communication channel enables values of the specified type to be communicated between two concurrent processes.

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 nonnegative numbers less than 256.

INT **INT** is the type of signed integer most efficiently provided by the implementation.

Implementations may also support the following

INT16 Signed integers **n** in the range $-32768 <= n < 32768$

INT32 Signed integers **n** in the range $-2^{31} <= n < 2^{31}$

INT64 Signed integers **n** in the range $-2^{63} <= n < 2^{63}$

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

2.2.6 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.

2.2.7 Procedures

A process may be given a name. For example

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

2.2.8 Expressions

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

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
/\ (bitwise and), \/ (bitwise or), >< (bitwise xor), ~ (bitwise not)	integers	bit operators
<<, >>	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**.

2.2.9 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.

2.2.10 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.

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 :
```

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.

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.

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.

4.1 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.

4.2 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.

4.3 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.

5.1 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 megabits per second. 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 Electrical specification of links

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.

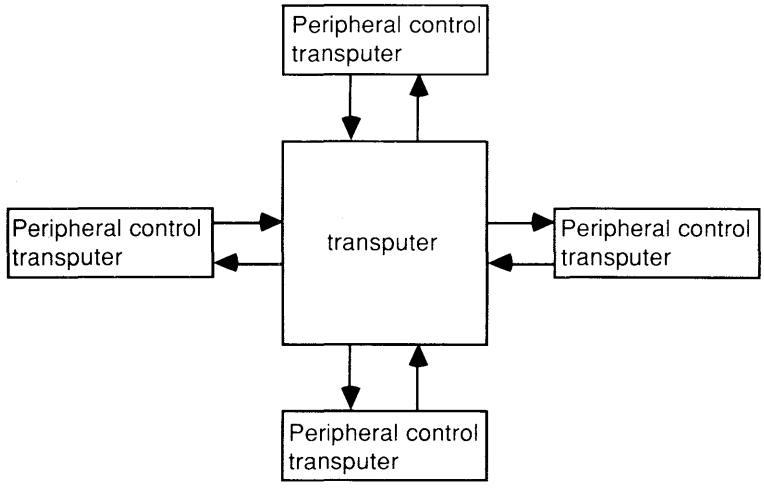
5.4 Peripheral interfacing

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.

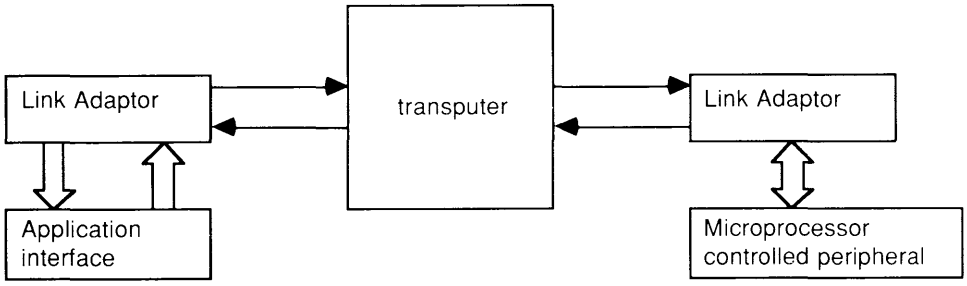
The first is by employing peripheral control transputers (eg for graphics or disks), in which the transputer chip connects directly to the peripheral concerned. The interface to the peripheral is implemented by special purpose hardware within the transputer. The application software in the transputer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

Transputer with peripheral control transputers



The second method is by employing link adaptors. 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.

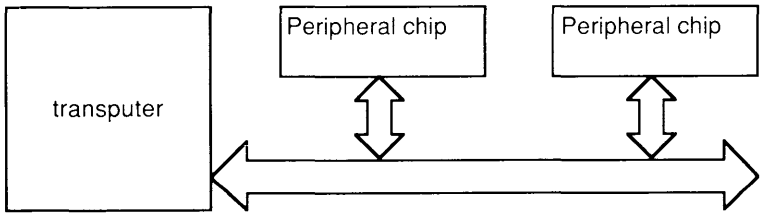
Transputer with link adaptors



The third method is by memory mapping the peripheral onto the memory bus of a transputer. 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.

Memory mapped peripherals



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.

The bits in a byte are numbered 0 to 7, with bit 0 the 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.

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 the diagram and memory locations with more negative addresses are also to the bottom of the diagram.

6.1 Signal naming conventions

The signal names, identifying the individual pins on a transputer chip, have been chosen to avoid being cryptic, giving as much information as possible.

All transputer signals described in the text of this manual are printed in **bold**.

The majority of transputer signals are active high. Those which are active low have names commencing with **not**.

All INMOS products - both memories and transputers - have a number of the form

IMS xxxx-xx

The main field identifies the product, and the field after the hyphen is used for speed variants, etc. Extra letters are sometimes introduced, eg for military quality products.

The initial character of the main field is a digit for memory products, a letter for transputer products. The particular letter indicates the type of transputer product:

IMS Cxxx	Communications adaptors
IMS Gxxx	Graphics transputers
IMS Mxxx	Mass storage transputers
IMS Txxx	Transputers

Support products are numbered as follows:

IMS Bxxx	Module level product
IMS Dxxx	Development system
IMS Lxxx	Literature
IMS Pxxx	Occam programming system
IMS Sxxx	Software product

inmos®

Product data

IMS T414 transputer

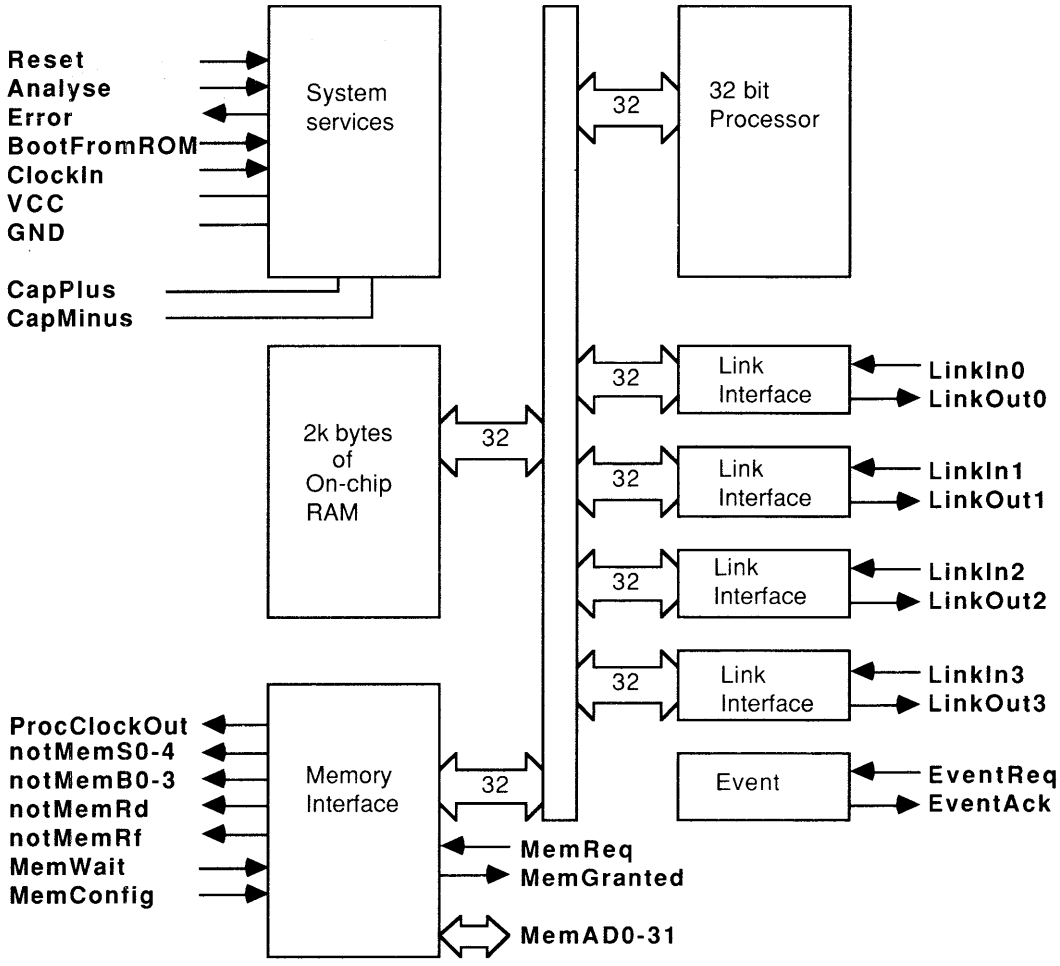
The IMS T414 is the first in a family of transputers, all of which are consistent with the INMOS transputer architecture, described in the transputer architecture manual.

This manual details the product specific aspects of the IMS T414 and contains data relevant to the engineering and programming of the device.

Other information relevant to transputer products is contained in the occam programing manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

The 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.

This edition of the manual is dated October 27, 1986.


Reset
Analyse
Error
BootFromROM
ClockIn
VCC
GND
CapPlus
CapMinus
System services
32
32 bit Processor
2k bytes of On-chip RAM
32
32
Link Interface
LinkIn0
LinkOut0
32
Link Interface
LinkIn1
LinkOut1
32
Link Interface
LinkIn2
LinkOut2
32
Link Interface
LinkIn3
LinkOut3
Event
EventReq
EventAck
ProcClockOut
notMemS0-4
notMemB0-3
notMemRd
notMemRf
MemWait
MemConfig
Memory Interface
32
MemReq
MemGranted
MemAD0-31

IMS T414

The IMS T414 integrates a 32-bit microprocessor, four standard transputer communications links, 2K bytes of on-chip RAM, a memory interface and peripheral interfacing on a single chip, using a 1.5 micron CMOS process.

Processor

The design achieves compact programs, efficient high level language implementation and provides direct support for the occam model of concurrency. Procedure calls, process switching and interrupt latency are all sub-microsecond. The T414 provides high performance arithmetic and microcode support for floating point operations.

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.

Links

The T414 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.

Memory

The 2K bytes of static RAM provide a maximum data rate of 80 MBytes/sec with access for both the processor and links.

Memory interface

The T414 can directly access a linear address space up to 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 nano seconds (26.6 MBytes/sec). A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

Peripheral interface

The memory controller supports memory mapped peripherals, which may use DMA. Links may be interfaced to peripherals via an INMOS link adaptor. A peripheral can request attention via the event pin.

Time

The processor includes timers for both high and low priority processes.

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. The error can be handled internally by software or externally by sensing the error pin. System state is preserved for subsequent analysis.

The optimal method of programming the T414 processor is to use occam. See the occam programming manual for full details. Several standard languages are also supported. These include C, Fortran and Pascal.

The processor provides direct support for the occam model of concurrency and communication. It has a scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. The number of registers which hold the process context is small and this, combined with fast on-chip RAM, provides a sub-microsecond process switch time.

Process communication is implemented by memory to memory block move operations. These fully utilize the bandwidth available from the on-chip RAM.

2.1 T414 types

The implementation of occam for the T414 transputer supports the following types:

CHAN OF *type* Each communication channel provides communication between two concurrent processes. Each channel allows the communication of data of the specified type.

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 **n** in the range $0 <= \mathbf{n} < 256$

INT Signed integers **n** in the range $-2^{31} <= \mathbf{n} < 2^{31}$

INT16 Signed integers **n** in the range $-32768 <= \mathbf{n} < 32768$

INT32 Signed integers **n** in the range $-2^{31} <= \mathbf{n} < 2^{31}$

INT64 Signed integers **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 T414 process multiplexing

The T414 supports two levels of priority - in occam notation, a **PRI PAR** (priority parallel) process may have two components. The priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

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.

Low priority processes

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 its 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.

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

To ensure that each low priority process proceeds, high priority processes should never occupy the processor continuously for a period of time equal to a timeslice period. A good guideline is to ensure that, if there are a total of **n** high priority processes, then each limits its activity to much less than 1/**n**'th of any 1 millisecond period.

Interrupt latency

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, maximum 58 cycles (assuming use of on-chip RAM).

2.3 T414 Error flag

Expressions which cause arithmetic overflow are invalid, and processes which cause array bound violations are invalid. If the compiler is unable to check that a given construct contains only valid expressions and processes, then extra instructions are compiled to perform the necessary checks at runtime. If the result of the check indicates that an invalid expression or invalid process has occurred, then the processor's **Error** flag is set.

In the implementation of occam, the offending process stops when the **Error** flag is set.

The T414 can be initialised so that the processor halts when the **Error** flag is set. The appropriate initialisation sequence is provided by the development system.

If the processor has been halted as the result of an error, the links continue with any outstanding transfers, the memory interface continues to provide refresh cycles, and the transputer may be analysed. See section 3.2.

When a high priority process pre-empts a low priority process, the current value of the **Error** flag is saved; the **Error** flag itself is maintained. When, finally, there are no high priority processes able to run, the current state of the **Error** flag is lost, and the preserved state is restored as part of commencing to execute the pre-empted low priority process.

This ensures that the state of the **Error** flag is preserved during the evaluation of an expression or a sequence of assignments.

2.4 T414 memory map

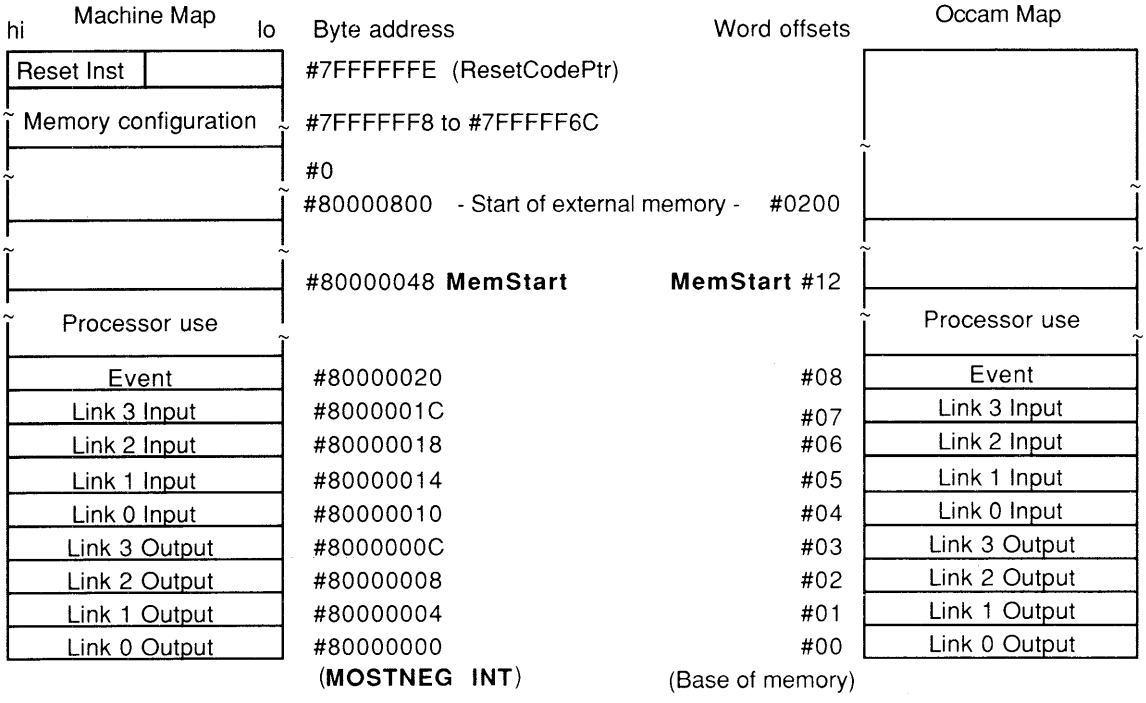
The address space of the T414 is signed and byte addressed. Words are aligned on four-byte boundaries. Addresses in the range #80000000 to #800007FF reference on-chip memory.

The first 18 words of the address space are used for system purposes. The next available location is, by convention, referred to as **MemStart**.

A suitable declaration of **MemStart**, for example is

```
VAL MemStart IS #12 :
```

The programmer can access locations in memory by using the occam mechanism of placement. This allows variables of any type to be placed at a location specified as a word offset from the base of memory. The placement of a byte array allows access to the byte components of a word. For example, a byte array could be placed in internal memory to optimize access, or a similar array could be placed in external memory to address one or more memory mapped devices.



This schema for calculating addresses is used to provide word length independent code, as though memory were an array of type **INT** (**[]INT**). The link addresses will always be found within the lower bounds of the memory array space.

The top of address space is used, by convention, for ROM based code. If the transputer is configured to bootstrap from ROM, then the processor commences execution from address #7FFFFFFE. If the transputer is configured to use an externally defined memory interface configuration, then this is stored at locations #7FFFFF6C to #7FFFFFF8. Tools are supplied with the compiler and development system to enable the generation and placement of ROM images.

2.5 T414 timer

At the standard **ClockIn** cycle rate of 5MHz, the high priority timer has a resolution of one microsecond and cycles approximately every 4.295 seconds (approximately 71 minutes). When comparing timer values, it should be noted that each half cycle is approximately 35 minutes.

The low priority timer has a resolution of 64 microseconds. One second is exactly equal to 15625 ticks. Each half cycle is approximately 37 hours.

2.6 T414 event pins

An attention-seeking peripheral may signal the T414 via the **EventReq** pin, which the T414 handshakes using **EventAck**. The T414 implements a hardware channel to allow a low to high transition on the **EventReq** pin to be communicated to a process as a synchronizing message.

An occam channel (which must already have been declared) may be associated with **EventReq** pin by a channel placement. The conventional name and the value used for this channel are given by

```
PLACE Event AT 8 :
```

Event behaves like an ordinary occam channel, and a process may synchronize with a low to high transition on the **EventReq** pin by using the occam construct:

```
Event ? signal
```

The process waits until the channel **Event** is ready. The channel is made ready by the transition on the **EventReq** pin (this may occur before the process attempts to input).

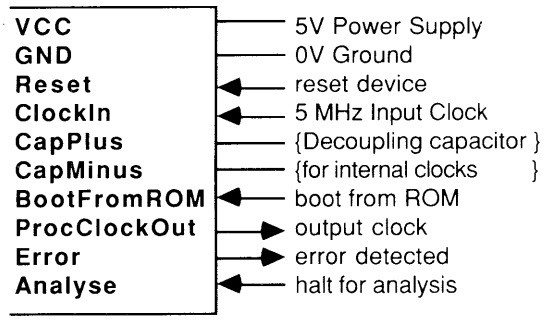
When the process is able to proceed, and if it executes at high priority, then it will take priority over any low priority process which may be executing when the transition occurs on the **EventReq** pin.

2.7 T414 link placement

The link addresses will always be found within the lower bounds of the memory array. The conventional names and the values used for these channels are

```
PLACE Link0Output AT 0 :
PLACE Link1Output AT 1 :
PLACE Link2Output AT 2 :
PLACE Link3Output AT 3 :
PLACE Link0Input  AT 4 :
PLACE Link1Input  AT 5 :
PLACE Link2Input  AT 6 :
PLACE Link3Input  AT 7 :
```

The system services comprise the clocks, power and initialisation used by the whole of the transputer. The **Reset** and **Analyse** inputs enable the T414 to be initialised, for example on power up, or halted in a way which preserves its state for subsequent analysis. Whilst the T414 is running both **Reset** and **Analyse** are held low. The **Error** signal is directly connected to the processor's **Error** flag. See section 2.3.



3.1 Reset

The T414 is initialised by pulsing **Reset** high whilst holding **Analyse** low. Operation ceases immediately and all state information is lost. The memory interface configuration is set up when the transputer is initialised.

The processor and links start operating after the memory interface configuration cycle is complete and eight refresh cycles have been executed to initialise any dynamic RAM.

The processor then bootstraps. If the **BootFromROM** input is high it will start to execute code starting from address #7FFFFFFE. If **BootFromROM** is low it will bootstrap from a link, that is, it will load a program from a link and then execute it.

When initialising following power-on, a time is specified during which **VCC** must be within specification, **Reset** must be high, and the input on **ClockIn** must be oscillating. **Reset** is taken low after the specified time has elapsed.

During power on reset all link inputs must be held low. (N.B. all link outputs are made low by reset.)

3.2 Analyse

A system built from transputers may be brought to a halt in a consistent state which is preserved for subsequent analysis. This analysis is performed in a manner similar to bootstrapping. The transputer development system includes appropriate bootstrap and analysis software.

A signal may be applied to the **Analyse** inputs of all the transputers in the system. The system is analysed by first taking **Analyse** high. This causes each transputer to halt, after a short period of time, in a consistent internal state. When the system has halted, **Reset** is taken high for the specified hold time, after which it is taken low. **Analyse** is then taken low, at which time each transputer bootstraps.

When **Analyse** is taken high, the processor will halt within three timeslice periods (approximately three milliseconds), plus the time taken for any high priority process to cease processing. Any outputting links continue until they complete the remainder of the current word. Input links will continue to receive data. Provided that there are no delays in sending acknowledgements, the links in a system will therefore cease activity within a few microseconds. Sufficient time must be allowed to allow the processor to halt and link traffic to cease before **Reset** is asserted.

The system must be designed so that links connected to the external world are quiet during reset.

The memory interface is not affected by **Analyse**, or by **Reset** while **Analyse** is held high. If refresh cycles are enabled, it continues to refresh external dynamic RAM.

3.3 Bootstrapping and analysis of a "failed" system

The transputer has two methods of bootstrapping. Firstly, the conventional bootstrap which occurs after **Reset** and secondly, an analysis bootstrap which occurs after **Reset** plus **Analyse**. This second method allows the state of a system, which could well be a complex network of transputers, to be examined. For example, memory continues to be refreshed so that data is not lost. In both cases the mechanism used is very similar and the bootstrap or analysis code may be provided from either the external memory of the transputer, often in ROM, or if the transputer is a part of a network, via its communication links.

3.3.1 Bootstrapping

It is possible to bootstrap the T414 either by executing code held in ROM or by executing code received on a link. In addition, prior to bootstrapping from a link, it is possible to read or write to any memory location in a transputer's memory map via a link.

3.3.2 Bootstrapping from ROM

To bootstrap from ROM, the **BootFromROM** input is wired to **VCC**.

The T414 bootstraps from ROM by executing a process at low priority. Control is transferred to the top two bytes in memory (at #7FFFFFFE), which will invariably contain a backward jump into ROM. **`Memstart`** (#80000048) is used as the location of the process workspace.

3.3.3 Bootstrapping from a link

To bootstrap from a link, the **BootFromROM** input is wired to **GND**.

The T414 bootstraps from a link by waiting for the first byte (the control byte) to arrive on any of the four link inputs.

If the value of the control byte is two, or greater, it is considered to be a count of the number of bytes to be input. The following bytes are then placed into memory at **`MemStart`** (#80000048) and the T414 begins to execute the input code as a process at low priority by transferring control to **`MemStart`**. The memory space immediately above the loaded code is used as the process workspace.

The mechanism of bootstrapping from any link allows a network of transputers to be bootstrapped without the need for ROM or any external memory.

3.3.4 Peeking and Poking

A unique feature of the transputer allows any location of transputer memory, be it internal or external, to be read or written to from a link. This allows, with appropriate software, an external memory system to be debugged without the need to run a program on the system under development.

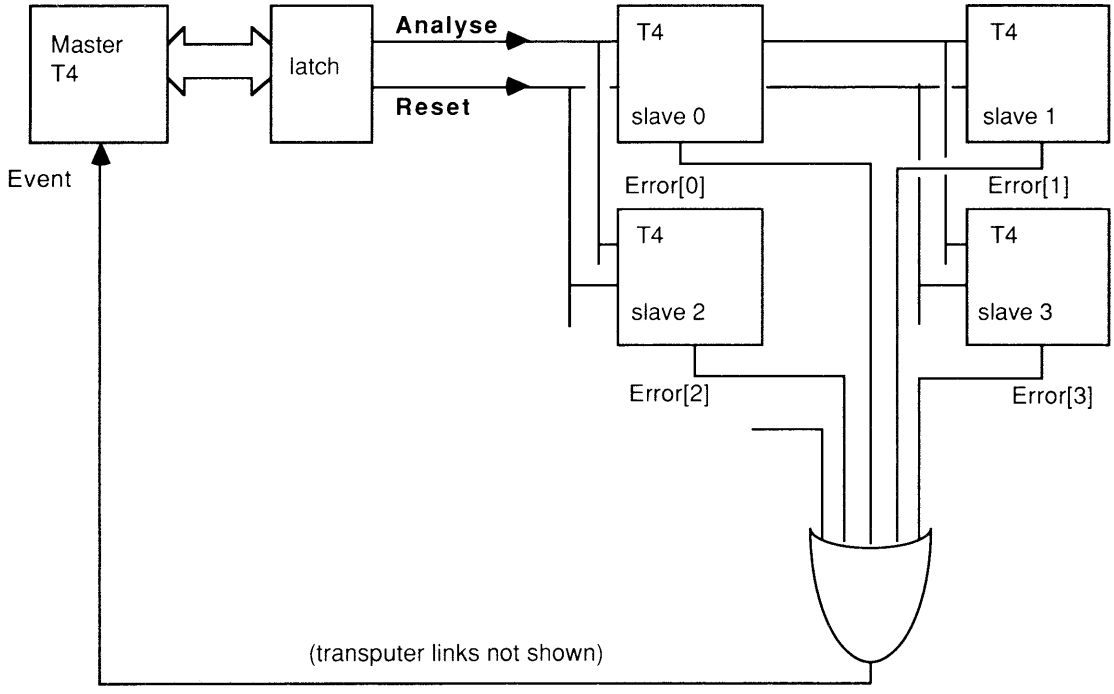
If, whilst the transputer is waiting to boot from a link, it receives a zero byte then a word address is input, followed by a word of data which is written to that address. The transputer then returns to the state of awaiting a message from any link.

If the first byte received is one, then a word address is input, a word of data is read from that address and is output down the corresponding output link. The transputer will then return to the state of awaiting a message from any link.

3.4 Using Error and Analyse

Analyse may be used in conjunction with the **Error** output to isolate errors in a multi-transputer system. A transputer which has signalled an error may be halted and subsequently analysed under the control of a 'master' transputer, using the methods described above. Or this could be achieved by connecting the 'OR' of all the **Error** pins to the **EventReq** pin on the master transputer, and connecting the **Analyse** and **Reset** pins to the master transputer as a 'peripheral'.

Error treatment in multi-transputer system

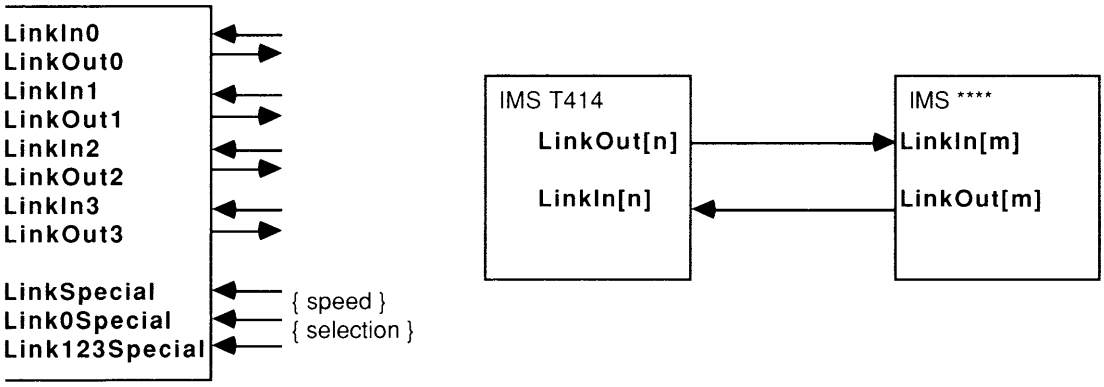


4.1 Standard transputer links

The T414 provides four standard links, each providing two uni-directional point-to-point occam channels.

The links implement the standard inter transputer communications protocol. The links are connected by wiring a **LinkOut[n]** to a **LinkIn[m]** and **LinkIn[n]** to a **LinkOut[m]**. Unused link inputs should be held to ground, **GND**.

Link connection



The T414 links wait until each full byte has been received before outputting the corresponding acknowledge packet. An acknowledge packet can be received at any time following the transmission of a data packet start bit.

At a link speed of 10Mbits/sec, data is transmitted at about 400Kbytes/sec in each direction, and the combined (bidirectional) data rate when the link carries data in both directions at once is 800Kbytes/sec.

4.1.1 Link speed selection

Link speed selection depends on the settings of three pins, **Link0Special**, **Link123Special** and **LinkSpecial**. These are shown in the tables below, a 0 indicating the signal should be held to **GND**, and a 1 indicating the signal should be held to **VCC**.

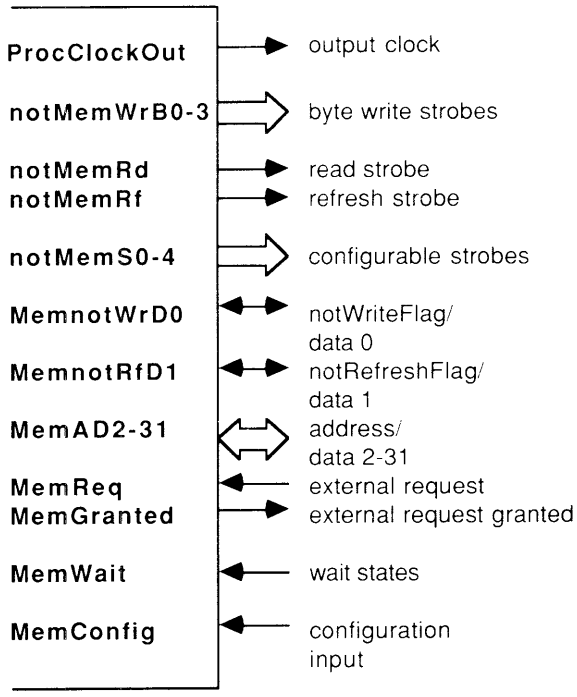
Table 1. Speed settings for Link 0.

LinkSpecial	**Link0Special**	**Speed at 5MHz** Mbits/sec	**Unidirectional data rate** Kbytes/sec	**Bidirectional data rate** Kbytes/sec
0	0	10	400	800
0	1	5	200	400
1	0	10	400	800
1	1	20	800	1600

Table 2. Speed settings for Links 1,2 and 3.

LinkSpecial	**Link123Special**	**Speed** Mbits/sec	**Unidirectional data rate** Kbytes/sec	**Bidirectional data rate** Kbytes/sec
0	0	10	400	800
0	1	5	200	400
1	0	10	400	800
1	1	20	800	1600

The data rates are reduced when performing transfers using slow external memory.



The memory interface uses a 32-bit wide multiplexed address/data bus and its controller may be configured, on reset, to suit a wide variety of different memory systems. One of 13 preset configurations may be selected or the configuration may be supplied externally.

The T414 memory interface cycle has six states, referred to as "Tstates". The duration of each Tstate is chosen to suit the particular memory system.

Tstate	
T1	address setup time before address valid strobe
T2	address hold time after address valid strobe
T3	read cycle tristate/write cycle data setup
T4	extended for wait states
T5	read or write data
T6	end tristate/data hold

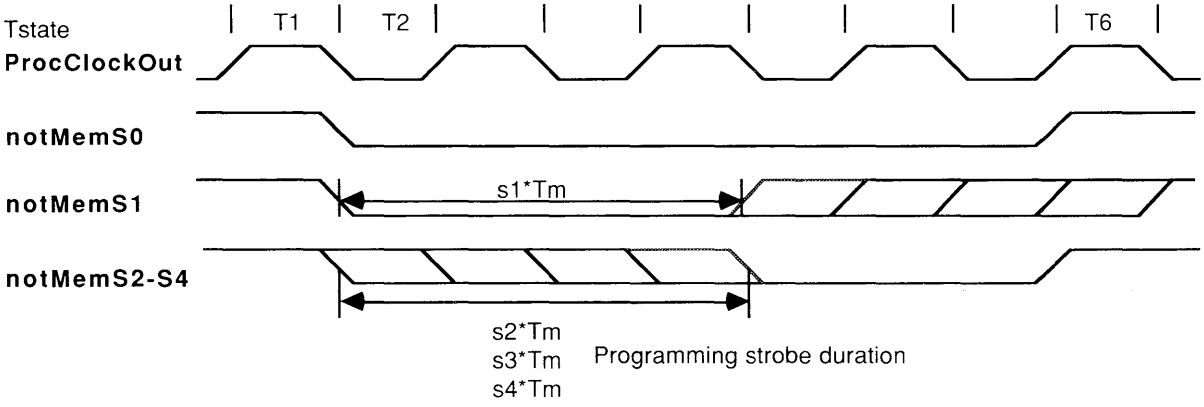
Each Tstate is configured to have a duration of one to four periods **Tm**. In addition, **T4** may be extended by wait states. One period **Tm** is half the processor cycle time. In common with all transputer devices, the T414 operates from a single 5MHz clock input, regardless of the speed selection of the particular device. A signal, **ProcClockOut**, is provided which has a period equal to the internal processor microcode cycle time. Thus for a T414-20 **ProcClockOut** has a period of 50ns (20MHz), for a T414-15 **ProcClockOut** has a period of 67ns (15MHz).

The period of a Tstate **Tm** is half the period of **ProcClockOut**.

5.1 Control signals

The Memory interface provides several control signals. They include separate write strobes provided for each byte **notMemWrB0-B3**. A read strobe, **notMemRd** is provided, and should be used to ensure the **AD**bus is tristated before read data is enabled. Five general purpose strobes are provided **notMemS0-S4**. Four of these strobes, **notMemS1-S4** are programmable. The duration of each strobe can be calculated using the table below and the configuration table. It should be noted that all external events on the memory interface occur, with some skew, synchronously with a rising or falling edge of **ProcClockOut** depending on the configuration.

The use of the strobes **notMemS0-S4** will depend upon the memory system, for example they may be used directly to control dynamic RAM. The durations of **notMemS1-S4** can be calculated using the strobe coefficients **s1** to **s4** respectively, and are independently configurable to be between 0 and 31 periods **Tm**. If **notMemS1** is configured to zero periods then it will remain high continuously. If **notMemS2-S4** are configured to zero then the pins will remain low continuously.

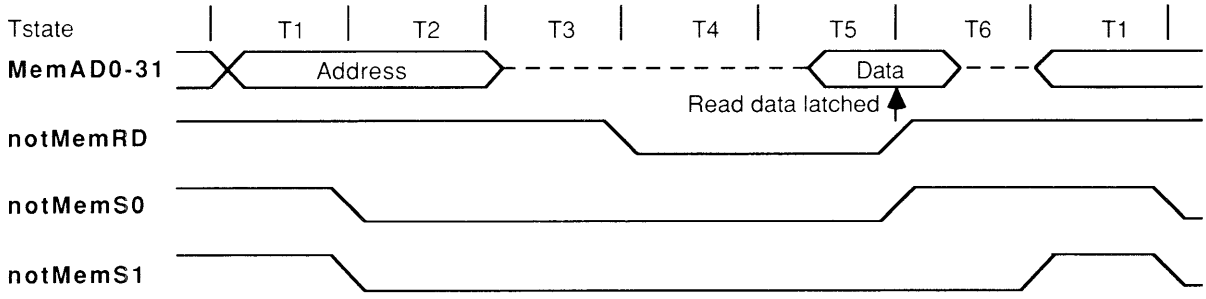


Signal	Starts beginning	Ends beginning	Max. strobe duration Tms (excluding wait states)	Min. strobe duration Tms
notMemRd	**T4**	**T6**	8	2
notMemWr B0-B3	**T3**	**T6**	12	3
	T4	**T6**	8	2
notMemS0	**T2**	**T6**	16	4
notMemS1	**T2**	**T2**+(s1*__Tm__) or end of **T6** (whichever is first)	20	0
notMemS2	**T2**+(**Tm***s2)	**T6**	16	0
notMemS3	**T2**+(**Tm***s3)	**T6**	16	0
notMemS4	**T2**+(**Tm***s4)	**T6**	16	0

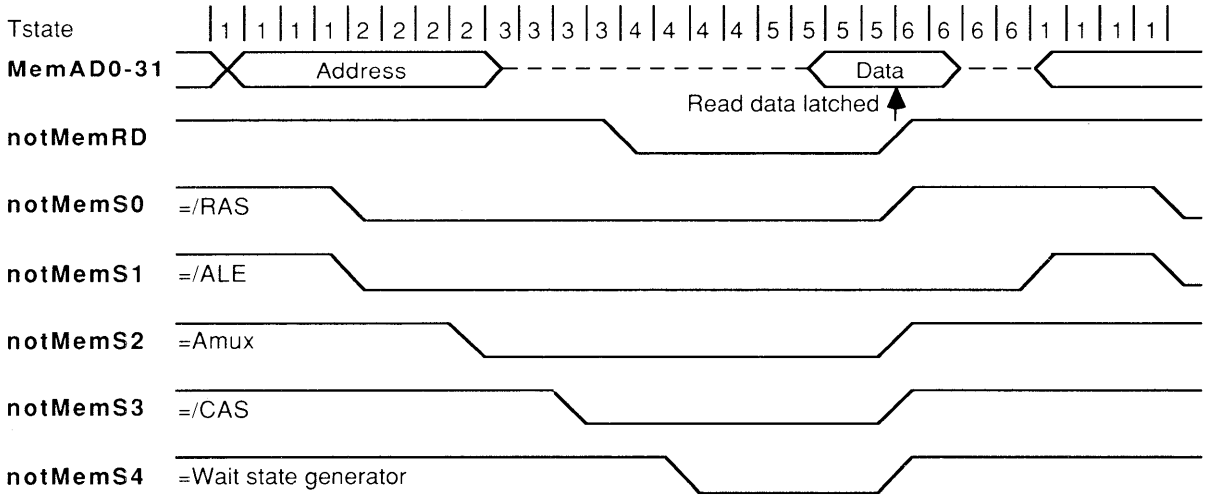
5.2 Read cycles

Read cycles generate **notMemS0-S4** and the **notMemRd** gate data signal. Input data is latched into the memory interface at the end of **T5**. Byte reads are performed as word reads, with byte selection performed by the processor. The byte address bits are not output on the **MemAD** signals.

Read cycle with each Tstate lasting one period Tm



Read cycle with each Tstate lasting four periods Tm



5.3 Write cycles

Write cycles generate **notMemS0-S4** and the **notMemWrB0-B3** write data strobe signals. A separate **notMemWrB** signal is provided for each byte. A word write uses all the **notMemWrB** signals. If a particular byte is not to be written, then the corresponding data outputs are tristated.

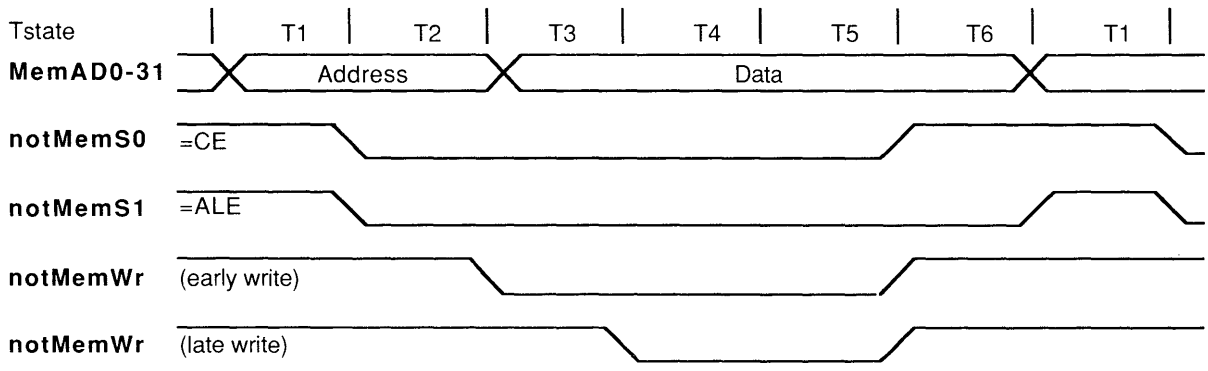
Early indication of a write cycle is provided by a low signal on **MemnotWrD0** during **T1** and **T2**, with the same timing as the address signals.

Write cycles may be configured to be either early or late. In late write cycles, write data is valid before the leading edge of **notMemWrB**, and held valid until after the trailing edge of **notMemWrB**.

Early write cycles provide a wide **notMemWrB** pulse, with data valid at the trailing edge of **notMemWrB**.

The **notMemRd** signal is held high during write cycles.

Write cycles, early and late



5.4 Use of the MemWait input

An external memory interface cycle may be extended by holding the **MemWait** input high. A wait period **Tm** is inserted at the end of **T4** if **MemWait** is high when sampled. This can be achieved by connecting **MemWait** to one of the configurable strobes **notMemS2-S4**, or to an appropriate external signal.

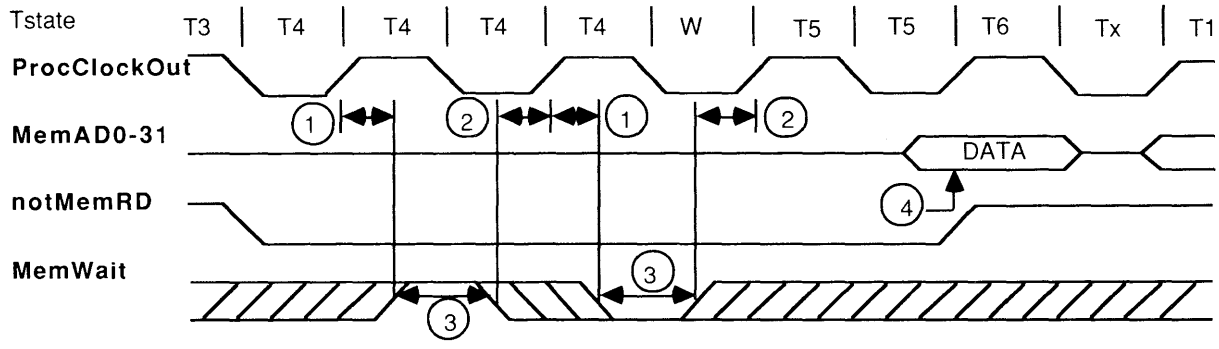
T5 must always begin on a rising edge of **ProcClockOut** when wait states are inserted, but may otherwise commence on either a rising or falling edge. If the number of wait states inserted means **T5** would commence on a falling edge of **ProcClockOut** one further wait period **Tm** is inserted.

MemWait is sampled during two periods **Tm** before the end of **T4** or wait period **Tm**.

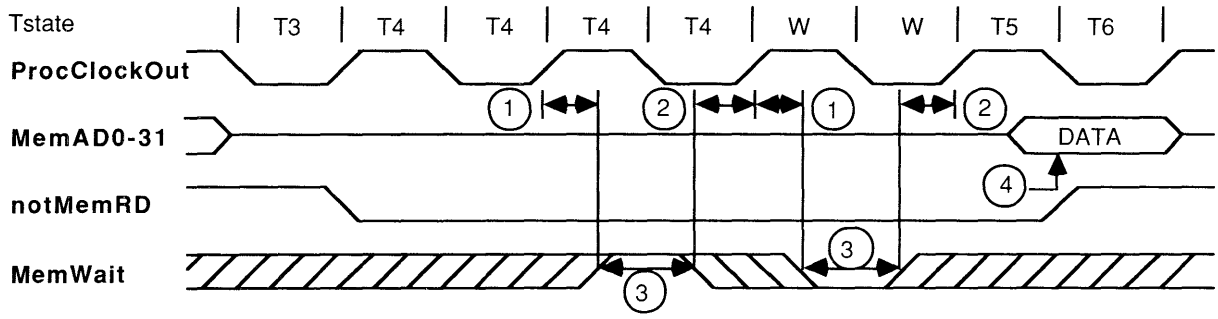
An external memory interface cycle should consist of an even number of periods **Tm** and therefore if an odd number of periods **Tm** are programmed an extra period **Tm** will be inserted at the end of **T6**.

Note that if **MemWait** is used to extend cycles for longer than the interval between refresh cycles, refresh cycles will be lost.

Cycle configured for long T4, with MemWait extending the cycle by one period.



Cycle configured for long T4, with MemWait extending the cycle by two periods.



1 **MemWait** setup from rising edge of **ProcClockOut**. ((0.5*Tm)+3)ns

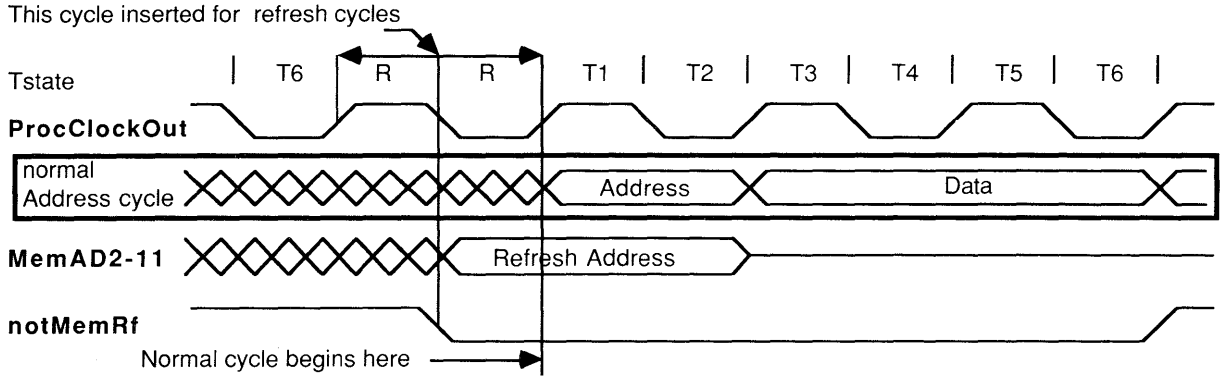
2 **MemWait** hold to rising edge of **ProcClockOut**. ((0.5*Tm)+3)ns

3 Note that **MemWait** is sampled during this period and should therefore not change state.

4 Read data latched.

5.5 Refresh

Memory Refresh



Refresh can be configured to be enabled or disabled. If Refresh is enabled, the interval between refresh cycles can be configured to be 18, 36, 54 or 72 periods of the input clock signal **ClockIn**.

Refresh interval in periods of ClockIn	**Interval with ClockIn frequency of 5.0 MHz** (in microseconds)	**Configuration encoding of Refresh**
18	3.6	00
36	7.2	01
54	10.8	10
72	14.4	11

Refresh cycles generate all the strobes **notMemS0-S4**, but neither **notMemRd** nor the **notMemWrB** signals. **notMemRf** goes low a period **Tm** before **T1** and remains low until the end of **T6**. Refresh is also indicated by **MemnotRfD1** going low a **Tm** before the start of **T1** with the same timing as refresh address signals.

A refresh cycle outputs a 10 bit refresh address, on **MemAD2-11**. It is incremented after each refresh cycle. The remaining address bits are held high except for **MemAD31**, **MemnotRfD1** which remain low during **T1** and **T2**.

5.5.1 Internal Memory access

During access of internal memory, the appropriate address appears for a single processor cycle on **MemAD2-31**, but no strobes are active.

5.6 Memory interface configuration

When **Reset** is high, the T414 is held in a reset state. When **Reset** goes low with **Analyse** held low, the memory controller executes the configuration sequence, selects the memory interface configuration after which the T414 bootstraps.

During the configuration sequence, the memory interface controller performs some memory cycles. It also carries out eight refresh cycles, to initialise any dynamic RAM that requires it.

The **MemReq** signal is ignored until the configuration is complete.

The configuration sequence is described below.

The sequence is:

- The **Reset** signal goes low.
- A delay of 144 periods of the input clock **ClockIn**.
- The **MemAD** pins are scanned to see which (if any) of the preprogrammed interface configurations is selected. The scan lasts for 144 clock periods. During this period **notMemRd, notMemWrB0-B3** and **notMemS0-S4** are all held high.
- The memory interface performs 36 read cycles using the default internal configuration as defined by **MemAD31** (see Configuration table) to access the configuration (if any) in external ROM.
- A delay dependent on the selected configuration.
- Eight refresh cycles to initialise dynamic RAM requiring it. If the configuration chosen disables refresh then no EMI cycles will be performed during this period. A delay equivalent to the period of eight refresh cycles will be inserted.
- The T414 bootstraps.

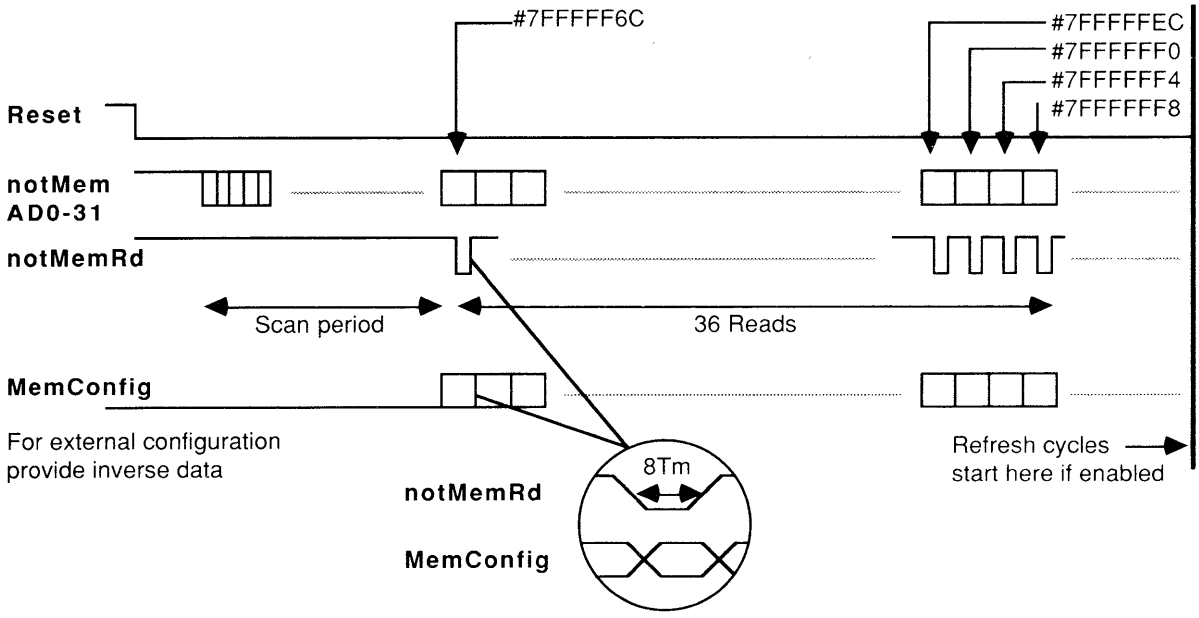
5.6.1 Interface configuration selection

MemConfig determines the configuration to be selected. To select one of the 13 predefined configurations **MemConfig** is connected directly to one of the **MemAD** signals. The particular **MemAD** signal used determines the preset configuration.

If the configuration is to be supplied externally from ROM then **MemConfig** is driven via an inverter from one of the **MemAD** signals. When the interface performs the 36 configuration read cycles to access the external configuration, the data on that particular **MemAD** signal will be read as the configuration data. These read cycles behave as standard read cycles. The configuration diagram shows only the value of the address during this read cycle. The address bus **notMemAD0-31** tristates in the normal way for the data period.

Alternatively, the configuration can be supplied externally by special purpose logic (eg a PAL or small PROM). In this case the logic must hold **MemConfig** low during the address scan and must supply a 36 bit serial word to the **MemConfig** input during the 36 configuration read cycles. The data supplied must be inverse data.

Configuration Diagram



The following table gives, for each preset configuration, the **MemConfig** connection, the duration of each Tstate, the width of **notMemS1**, and the delay from the start of **T2** to the falling edge of **notMemS2-S4**. These times are all in periods **Tm**. The table also gives the number of periods of the input clock between refresh cycles, e.g. 72 periods of 200ns give a refresh interval of 14.4 microseconds.

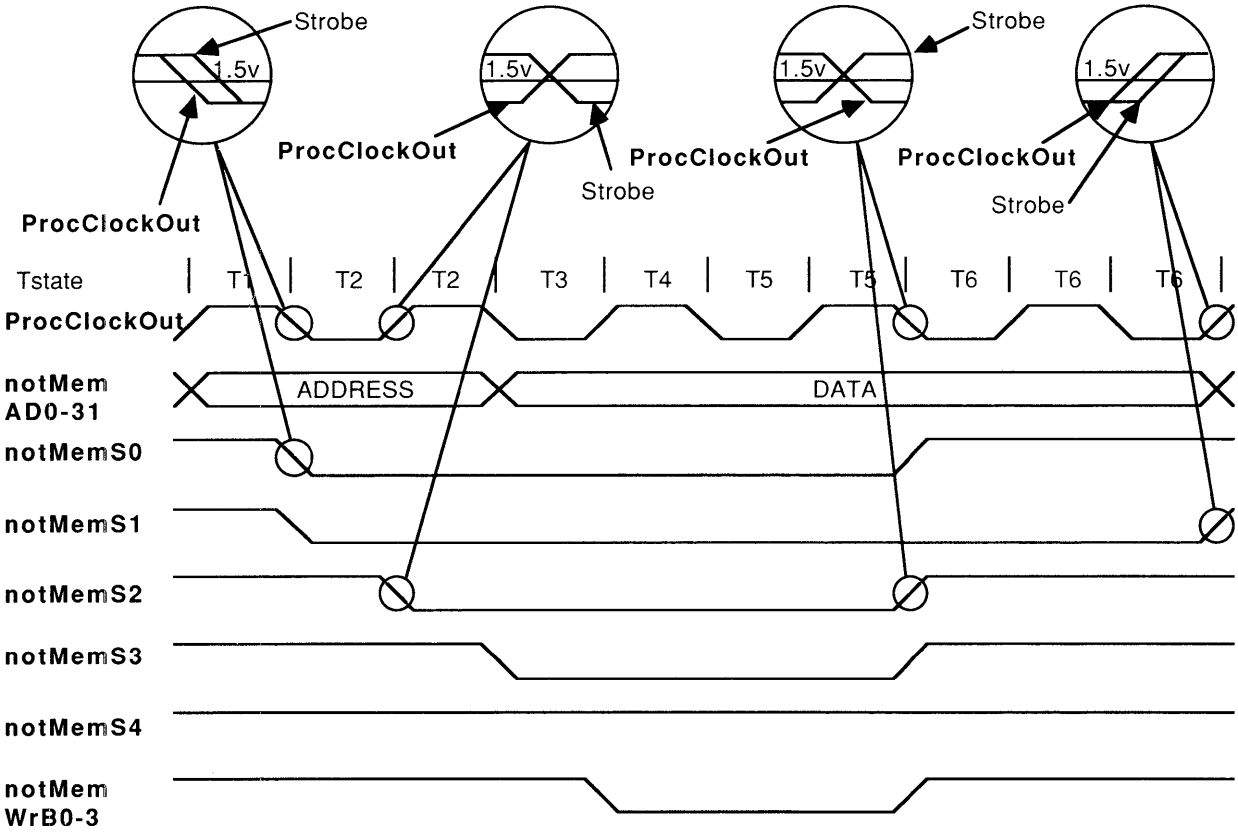
For each configuration the table gives the required memory access and cycle times (in multiples of processor cycles) plus the value of **e** to be used when estimating performance (see section 7.2.2 on T414 performance)

Configuration table

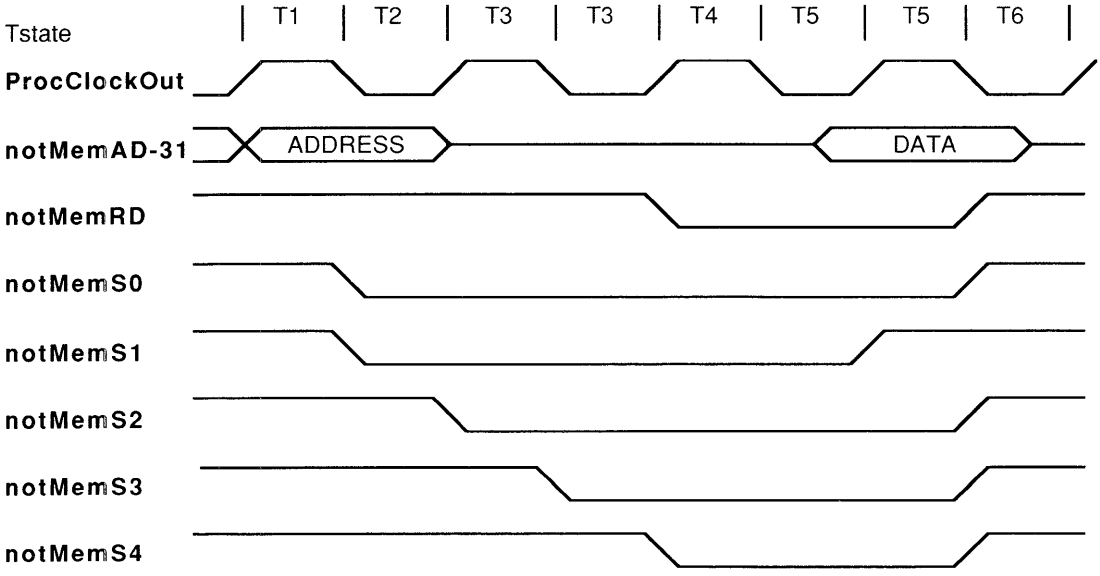
Pin	Duration of each Tstate periods **Tm**						Strobe coefficient				Write cycle type	Refresh interval input clocks	Cycle time proc cycles	Extra cycles e
	T1	**T2**	**T3**	**T4**	**T5**	**T6**	**s1**	**s2**	**s3**	**s4**				
MemAD0	1	1	1	1	1	1	30	1	3	5	late	72	3	2
MemAD1	1	2	1	1	1	2	30	1	2	7	late	72	4	3
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5	4
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6	5
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3	2
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4	3
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5	4
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6	5
MemAD8	1	1	1	1	1	1	30	1	2	3	early	-	3	2
MemAD9	1	1	2	1	2	1	30	2	5	9	early	-	4	3
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7	6
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9	8
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12	11

To determine which configuration setting to use, the particular setup times, hold times, access times, and cycle times that the memory system requires must be calculated. A program is provided with the transputer development system for assisting in this task. The AC characteristics of all output signals are defined in terms of the appropriate number of periods **Tm**, plus or minus a maximum skew from the corresponding edge of **ProcClockOut**. Examples of the internal configurations **AD2** and **AD5** are provided as a guide to using the above table. The example configuration **AD2** also demonstrates skew as the difference between the rising and falling edges of **ProcClockOut**, and the corresponding rising or falling edge of any signal on the memory interface. For simplicity, only four of the many combinations are shown.

T414 write cycle AD2 configuration showing strobe skew relative to ProcClockOut



T414 Read cycle AD5 configuration



5.6.2 Externally supplied configuration

When supplying the configuration externally a sequence of 36 one bit values, as defined in the following table, must be presented on the **MemConfig** input. The resulting 36 bit word is effectively composed of 13 fields. The value presented must be inverse data.

Fields 1 to 6 are 2 bit binary values which define the number of periods **Tm** that each Tstate is longer than its minimum time of one period **Tm**.

Field 7 is a 5 bit binary value defining the value of both the strobe coefficient **s1** and the width of **notMemS1** in periods **Tm**.

Fields 8 to 10 are 5 bit binary values defining, respectively, the delay to the leading edges of **notMemS2-S4** in periods **Tm**, and therefore the values of strobe coefficients **s2-s4** respectively.

Field 11 is a 2 bit value which defines the refresh interval as ((Field11 + 1) * 18) periods of the input clock **ClockIn**, i.e., selecting one of the values 18, 36, 54, or 72.

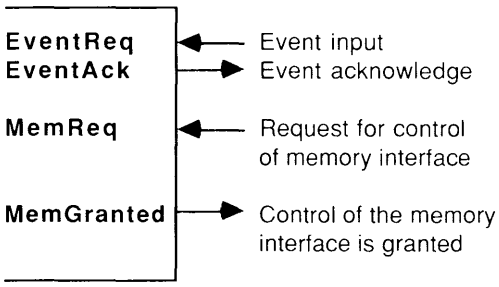
If Field 12 = 1, refresh is enabled; otherwise no refresh cycles are generated.

If Field 13 = 1, late write cycles are generated; otherwise early write cycles are generated.

The default setting of the configuration register used for initialization is the internal setting that would be selected if **MemConfig** were directly connected to **MemAD31**. This setting has each bit of the configuration register set to one. Thus, the slowest possible memory cycles are used (4 periods **Tm** for each of the Tstates **T1** to **T6**).

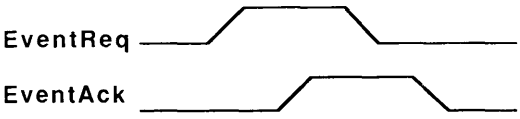
Value of MemAD during address phases of configuration interface cycles	Field	Function
#7FFFFF6C	1	**T1** lsb
#7FFFFF70	1	**T1** msb
#7FFFFF74	2	**T2** lsb
#7FFFFF78	2	**T2** msb
#7FFFFF7C	3	**T3** lsb
#7FFFFF80	3	**T3** msb
#7FFFFF84	4	**T4** lsb
#7FFFFF88	4	**T4** msb
#7FFFFF8C	5	**T5** lsb
#7FFFFF90	5	**T5** msb
#7FFFFF94	6	**T6** lsb
#7FFFFF98	6	**T6** msb
#7FFFFF9C	7	**notMemS1** lsb
#7FFFFFA0	7	**notMemS1**
#7FFFFFA4	7	**notMemS1**
#7FFFFFA8	7	**notMemS1**
#7FFFFFAC	7	**notMemS1** msb
#7FFFFFB0	8	**notMemS2** lsb
#7FFFFFB4	8	**notMemS2**
#7FFFFFB8	8	**notMemS2**
#7FFFFFBC	8	**notMemS2**
#7FFFFFC0	8	**notMemS2** msb
#7FFFFFC4	9	**notMemS3** lsb
#7FFFFFC8	9	**notMemS3**
#7FFFFFCC	9	**notMemS3**
#7FFFFFD0	9	**notMemS3**
#7FFFFFD4	9	**notMemS3** msb
#7FFFFFD8	10	**notMemS4** lsb
#7FFFFFDC	10	**notMemS4**
#7FFFFFE0	10	**notMemS4**
#7FFFFFE4	10	**notMemS4**
#7FFFFFE8	10	**notMemS4** msb
#7FFFFFEC	11	**RefreshInterval** lsb
#7FFFFFF0	11	**RefreshInterval** msb
#7FFFFFF4	12	**RefreshEnable**
#7FFFFFF8	13	**LateWrite**

Event Request block diagram



The two event signals, **EventReq** and **EventAck**, together provide a handshaken interface with an occam process executing in the processor.

Event request signal protocol



External logic takes **EventReq** high when the logic wishes to communicate with a process in the transputer. The rising edge of **EventReq** makes an external channel ready to communicate with the process. (This channel is additional to the external channels of the links.) When both the channel is ready and a process is ready to input from the channel, then the processor takes **EventAck** high and the process is scheduled. At any time after this point the external logic may take **EventReq** low, following which the processor will set **EventAck** low. After **EventAck** goes low, **EventReq** may go high to indicate the next event. Any further communication or synchronization necessary (for example, to tell external logic that the process has acted in response to the event) must be programmed explicitly.

If the process has high priority, and there is no other high priority process already running, then the maximum latency is 58 processor cycles, assuming that all memory accesses are to on-chip RAM. The typical latency is 19 processor cycles.

EventReq should be held low on reset.

DMA via the memory interface

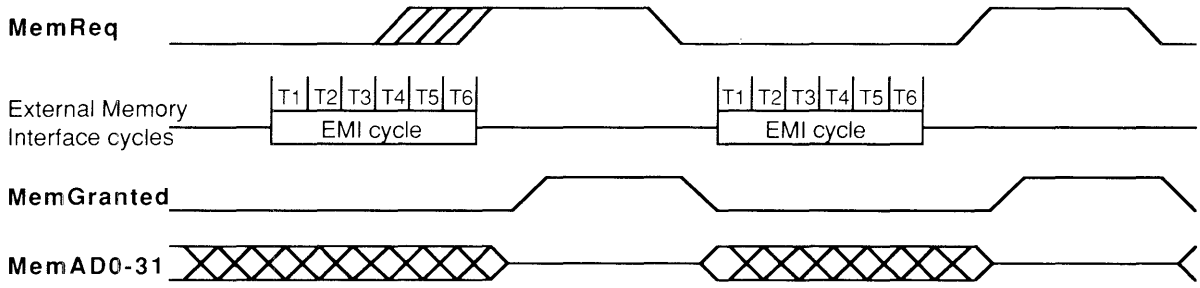
Peripheral controllers may access the whole of the external memory address space for DMA transfers using the asynchronous signals **MemReq** and **MemGranted**, which provide a handshake protocol for requesting and acknowledging control of the external address and data buses.

MemReq is sampled during the last period **Tm** of **T6** during all transputer external memory cycles. These cycles will be either Read, Write or Refresh of the external memory. **MemReq** is also sampled every alternate period **Tm** while the T414 does internal memory cycles.

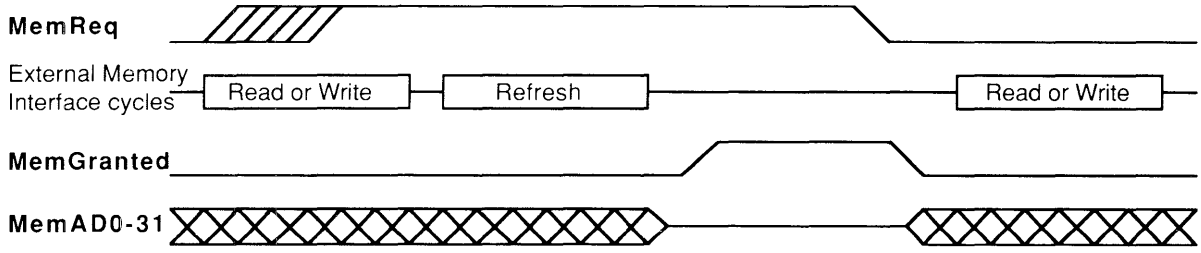
To ensure the transputer recognises a request to control the external bus, **MemReq** must be taken high one full processor clock cycle (2 periods **Tm**) before the end of **T6**, being a rising edge of **ProcClockOut**.

After the transputer has recognised an external bus request, and assuming there is no memory refresh cycle outstanding, the **MemAD** pins will be taken high impedance within 2 periods **Tm** of the end of the cycle after **T6**. **MemGranted** is taken high 1 period **Tm** later. If a refresh cycle is outstanding, then it is completed. At the end of the refresh cycle the bus will become tristated within 2 periods **Tm** of the end of the refresh cycle and **MemGranted** is taken high 1 period **Tm** later.

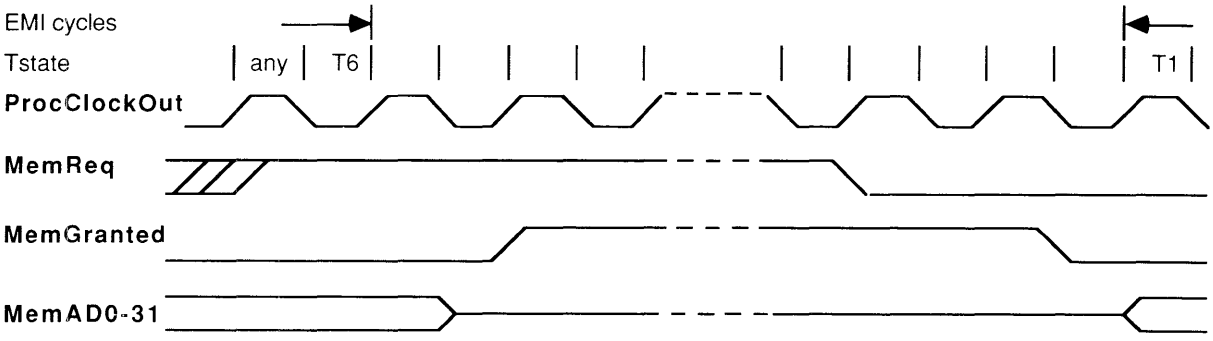
Functional operation of MemReq and MemGranted with External Memory cycle.



Functional operation of MemReq and MemGranted with Refresh cycle outstanding.



Timing diagram of MemReq and MemGranted



Note : MemReq is an asynchronous input signal and it is sampled internally. This sampling is effected by the current state of the external memory interface such that **MemGranted** will only be asserted at the end of any outstanding external memory cycle, including refresh.

Internal memory access

During processor cycles where the external memory bus is not accessed via the memory interface, the internal address on the transputer is normally output on the **MemAD** pins. This does not take place during DMA access.

Strobe signals

The memory interface controls all the strobes for controlling external memory. These are active low. During DMA, these signals will be held high.

DMA completion

At the end of the DMA, **MemReq** will be taken low. **MemReq** is sampled every 2 periods **Tm**. **MemGranted** will be taken low 2 **Tm** periods after **MemReq** is sampled and detected low. The next transputer memory cycle may start 1 **Tm** period later if an external cycle is outstanding, with addresses placed on the **MemAD** pins.

If **MemReq** is taken high asynchronously with the transputers external memory interface there is a worst case time before **MemGranted** will be taken high. This time is the sum of (Memory cycle time) + (Memory refresh time) + (5 periods **Tm**). Also note, if memory request time is longer than the external memory refresh period then refresh cycles will be lost. **MemGranted** will go low a maximum of two processor cycles after **MemReq** is taken low.

The processor and links can still continue to operate to internal memory while DMA proceeds to external memory, provided the processor or links do not try to use the external memory during DMA. The acknowledge signal **MemGranted** will function regardless of the state of **Reset** and **Analyse**.

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.

7.1 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 following abbreviations are used to represent the quantities indicated.

np number of component processes
ne number of processes earlier in queue
r 1 if INT 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 multiplier is 0)

Performance table

	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** 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 *type*	3.1	3.1
[size]CHAN OF *type*	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**

	Size (bytes)	**Time** (cycles)
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
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 call		
	3.5+(nparams-2)*1.1 +nvecparams*2.3	16.5+(nparams-2)*1.1 +nvecparams*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}

Figures in curly brackets 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.

7.1.1 Fast multiply, TIMES

The T414 has a fast multiplication instruction ('product'). If **Tb** is the position of the most significant bit set in the multiplier, then 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 occam on the transputer take advantage of this instruction. The modulo operator **TIMES** is mapped onto the instruction and is treated as non-commutative, the expression on the right of the expression being the multiplier.

The fast muliplication instruction is also used in occam implementations for the multiplication implicit in multi-dimensional array access.

7.1.2 T414 arithmetic procedures

A set of predefined procedures (**PROC**s) are provided to support the efficient implementation of multiple length and floating point arithmetic. In the following table, **n** gives the number of places shifted and all parameters are assumed to be local. Full details of these procedures are provided in the occam reference manual supplied as part of the development system and available as a separate publication.

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

PROC		Cycles	+Cycles for Parameter Access (Assuming local variables)
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

7.1.3 Floating point operations

Floating point operations are provided by a run-time package, which requires approximately 400 bytes of memory for the single length arithmetic operations, and 2500 bytes for the double length arithmetic operations. The following table summarizes the estimated performance of the package.

		Processor cycles (typical)	Processor cycles (worst)
REAL32	+, -	230	300
	*	200	240
	/	245	280
	<, >, =, >=, <=, <>	60	60
REAL64	+, -	565	700
	*	760	940
	/	1115	1420
	<, >, =, >=, <=, <>	60	60

7.1.4 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** is 2 for the fastest external memory: a typical value for a large external memory is 5.

The number of additional cycles required to access data in external memory is **e**. 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 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 the following table, **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**.

	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))

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**$\geqslant$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.

	e	2	3	4	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

7.2 T414 speed selections

The following table illustrates the designation of the T414 speed selections.

Designation	Instruction throughput	Processor clock speed	Processor cycle time	Input clock frequency
IMS T414-15	7.5 MIPS	15 MHz	67 ns	5 MHz
IMS T414-20	10 MIPS	20 MHz	50 ns	5 MHz

Parameters given in this section will be revised as a result of fuller characterization.

8.1 Absolute maximum ratings

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	0	7.0	V	1, 2, 3
VI,VO	Input or output voltage on any pin	-0.5	**VCC**+0.5	V	1, 2, 3
OSCT	Output short circuit time (one pin)		1	S	1
TS	Storage temperature	-65	150	°C	1
TA	Ambient temperature under bias	-55	125	°C	1
PD	Power dissipation rating		1	W	

Notes

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

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

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

8.2 Recommended operating conditions

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	4.5	5.5	V	1
VI,VO	Input or output voltage	0	**VCC**	V	1,2
II	Input current		±25	mA	3
CL	Load capacitance on any **MemAD** pin		50	pF	
TA	Operating temperature range	0	70	°C	

Notes

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

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

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

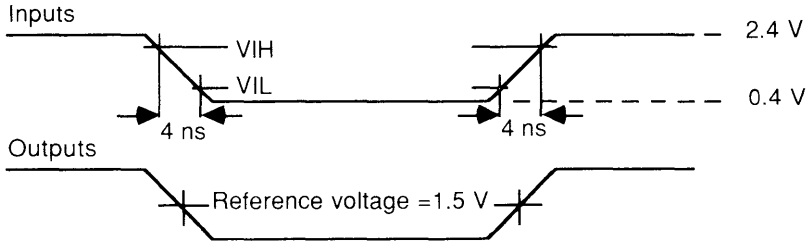
8.3 DC characteristics

Parameter		Conditions	Min	Max	Unit
VIH	High level input voltage		2.0	**VCC**+0.5	V
VIL	Low level input voltage		-0.5	0.8	V
II	Input current	**GND**<**VI**<**VCC**		±10	μA
VOH	Output high voltage	**IOH**=2mA	**VCC**-1		V
VOL	Output low voltage	**IOL**=4mA		0.4	V
IOS	Output short circuit current	**GND**<**VO**<**VCC**		50	mA
IOZ	Tristate output current	**GND**<**VI**<**VCC**		±10	μA
PD	Power dissipation			0.9	W
CIN	Input capacitance	f=1 MHz		7	pF
COZ	Output capacitance tristate	f=1 MHz		10	pF

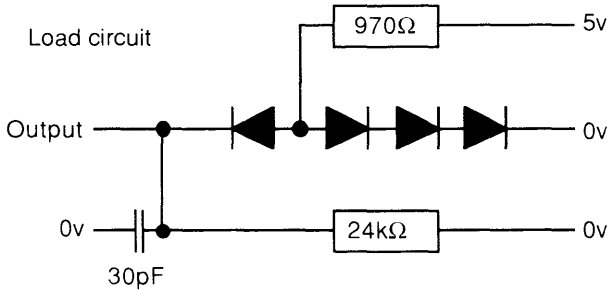
Note :
4.5 V < **VCC** < 5.5 V
0 °C < **TA** < 70 °C
input clock frequency = 5 MHz
All voltages are with respect to **GND**

8.4 Measurement of AC characteristics

Reference points for AC measurements



Load circuit for AC measurements



The load circuit approximates to two Schottky TTL loads, with total capacitance of 30 pF.

8.5 Connection of INMOS serial links

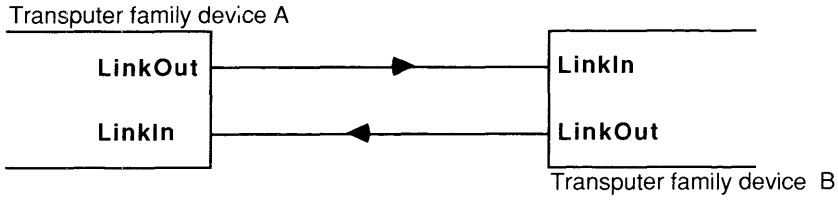
INMOS serial links can be connected in 3 different ways depending on their environment:

1 Directly connected

2 Connected via a series matching resistor

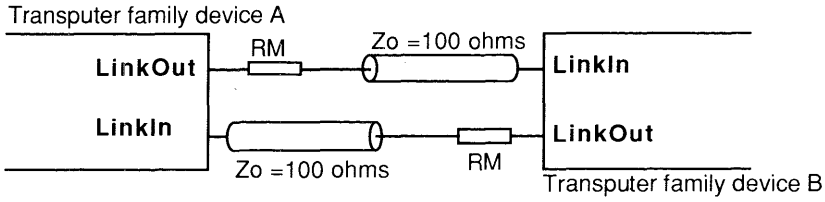
3 Connected via buffers

Direct connection



Direct connection is suitable for short distances on a printed circuit board.

Matched line

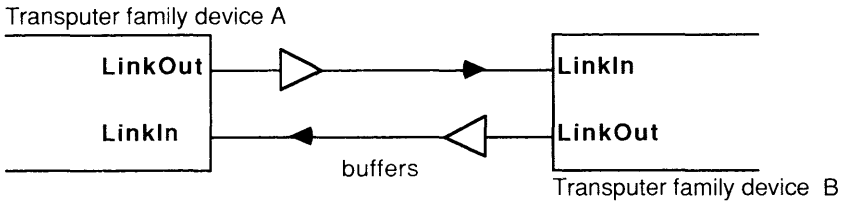


For long wires, approximately >30 cm, then a 100ohm transmission line should be used with series matching resistors.

Parameter		Nom	Max	Unit
RM	Series matching resistor for 100 ohm line.	47		ohm
TD	Delay down line		0.4	bit time

Note that if two connected devices have different values for **TD**, the lower value should be used. With series termination at **LinkOut** the transmission line reflection must return within 1 Bit time. Otherwise line buffers should be used.

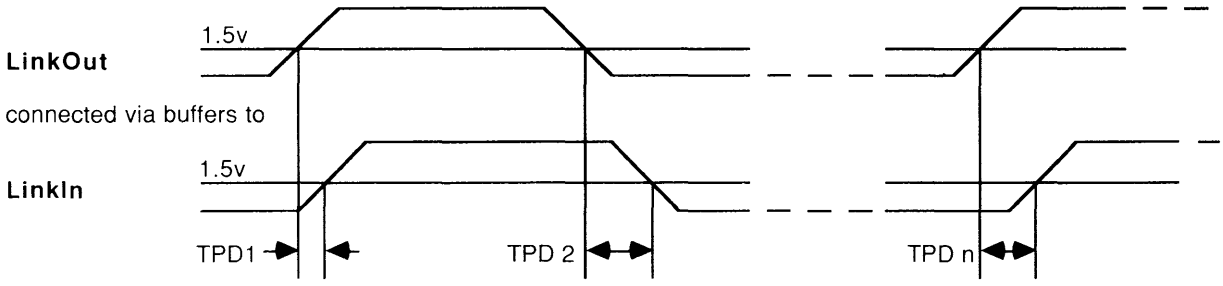
Buffered links



If buffers are used their overall propagation delay, TPD, should be stable within the skew tolerance.

Parameter	Max	Unit
Skew in buffering at 5 Mbits/sec	30	ns
Skew in buffering at 10 Mbits/sec	10	ns
Skew in buffering at 20 Mbits/sec		ns
Rise and fall time of **LinkIn** (10% to 90%)	20	ns

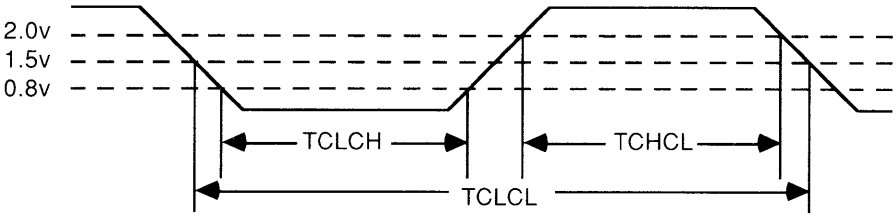
The above figures indicate that buffered links can be realised at 5Mbits/sec and 10Mbits/sec. For the case of 20 Mbits/sec, the maximum value can only be specified as a result of further characterisation.



The absolute value of TPD is immaterial because data reception is asynchronous. However, TPD will vary from moment to moment because of ground noise, variation in the power supplies of buffers and the difference in the delay for rising and falling edges. This will vary the length of data bits reaching **LinkIn**. *Skew* is the difference between the maximum and minimum instantaneous values of TPD.

8.6 AC characteristics of system services

ClockIn waveform



Parameter		Min	Nom	Max	Unit	Note
TCHCL	Clock pulse width high	40			ns	1
TCLCH	Clock pulse width low	40			ns	1
	Rise and fall time of **ClockIn** (10% to 90%)			10	ns	1
TCLCL	Clock period		200		ns	2
	Difference in frequencies of **ClockIn** for two devices connected by a link			400	ppm	6
Cap	Decoupling capacitor between **CapPlus** and **CapMinus**	1			μF	3
TRHRL	**Reset** pulse width high	8			**ClockIn** periods	4
	Time **VCC** must be valid and **Clockin** running before **Reset** goes low	10			ms	5
	BootFromROM setup time before **Reset** or **Analyse** goes low	0				
	BootFromROM hold time after **Reset** goes low	50			ms	
	Analyse pulse width	8			**Clockin** periods	

Notes

1 The clock transitions must be monotonic within the range between **VIH** and **VIL**.

2 The **TCLCL** parameter is measured between corresponding points on consecutive falling edges.

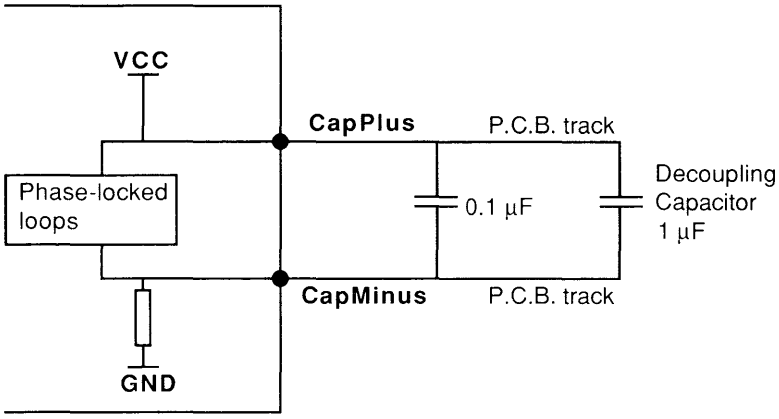
3 A 1 μF tantalum or ceramic low inductance, low leakage capacitor must be connected between **CapPlus** and **CapMinus** to decouple the supply to the on-chip clock generator. An additional good quality 0.1 μF ceramic capacitor should be connected in parallel with this. PCB track lengths to the capacitor should be minimised. No power supply should flow in these tracks.

4 Link inputs must be held low during reset. Reset forces link outputs low.

5 At power on reset.

6 This value allows the use of low cost 200ppm crystal oscillators.

Recommended PLL Decoupling.



8.7 Memory interface AC characteristics

The AC characteristics of the memory interface are dependent upon the speed of the transputer, the configuration of the memory interface, and the use of wait states.

The design of the memory interface enables all the AC characteristics in a particular instance to be derived by a simple calculation involving the number of periods **Tm** that the signal is high or low, plus factors for skew. A program, provided with the development system, performs this function.

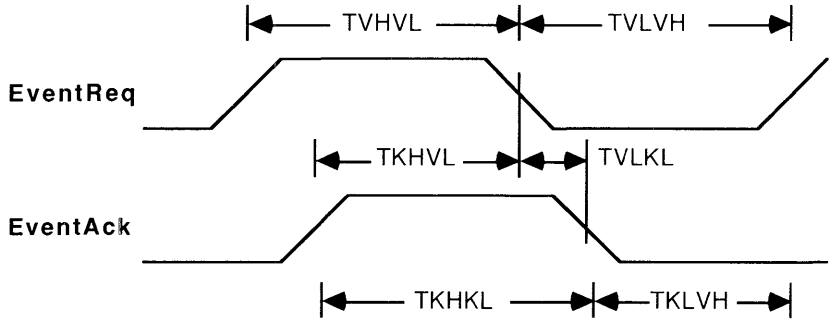
The start of a memory interface cycle always coincides with a rising edge of **ProcClockOut**, and **ProcClockOut** changes state at every period **Tm**.

Parameter	min	max	unit
Skew of any signal change to corresponding edge of **ProcClockOut**	-3	+7	ns
MemWait setup from rising edge of **ProcClockOut**		0.5**Tm**+3	ns
MemWait hold to rising edge of next **ProcClockOut**		0.5**Tm**+3	ns
Read data setup	20		ns
Read data hold time	0		

Skew should assume that the pads are equally loaded. Max occurs when **ProcClockOut** falls and a strobe rises. Min occurs when **ProcClockOut** rises and a strobe falls.

8.8 Peripheral interfacing AC characteristics

Event signals waveforms



Parameter		Min	Max	Unit
TVHVL	**EventReq** pulse width high	2		processor cycles
TVLVH	**EventReq** pulse width low	2		processor cycles
TVLKL	Falling edge delay from **EventReq** to **EventAck**	0	2	processor cycles
TKHKL	**EventAck** pulse width high	2		processor cycles
TKHVL	Delay from **EventAck** to falling edge of **EventReq**	0		
TKLVH	Delay from falling edge of **EventAck** to next **EventReq**	0		

Full details of each signal are provided in the referenced section.

Analyse Signal used to investigate the state of a T414. The signal brings the processor, links and clocks to a halt within approx three timeslice periods (approx 2.5 milliseconds). **Reset** may then be applied, which will not destroy state information nor reinitialise the memory interface. After **Reset** has been taken low, **Analyse** may be taken low after which the processor will execute its bootstrap routine. (**2.3**, **3.4**)
Input

BootFromROM If this is held to **VCC**, then the boot program is taken from ROM. Otherwise the T414 awaits a bootstrap message from a link. (**3.3.1**)
Input

CapMinus The negative terminal of an internal power supply used for the internal clocks. This must be connected via a 1 microfarad capacitor to **CapPlus**. If a tantalum capacitor is used **CapMinus** should be connected to the negative terminal. (**8.6**)

CapPlus The positive terminal of an internal power supply used for the internal clocks. This must be connected via a 1 microfarad capacitor to **CapMinus**. If a tantalum capacitor is used **CapPlus** should be connected to the positive terminal.

ClockIn Input clock from which all internal clocks are generated. The nominal input clock frequency for all transputers, of whatever wordlength and speed, is 5 MHz. (**8.6**)
Input

DoNotWire These are output pins reserved for INMOS use. They should be left unconnected.
Output

Error The processor has detected an error and remains high until **Reset**. Errors result from arithmetic overflow, by array bounds violations or by divide by zero. The **Error** flag can also be set by an instruction, to allow other forms of software detected error. (**2.3**)
Output

EventAck The **EventReq** and **EventAck** are a pair of handshake signals for external events. External logic takes **EventReq** high when the logic wishes to communicate with a process in the T414. The rising edge of **EventReq** causes a process to be scheduled. The processor takes **EventAck** high when the process is scheduled. At any time after this point the external logic may take **EventReq** low, following which the processor will set **EventAck** low. (**6**, **8.8**)
Output

EventReq Request external event. See description of **EventAck** above.
Input

GND Power supply return and logic reference, 0 V. There are several **GND** pins to minimize inductance within the package.

HoldToGND These are input pins reserved for INMOS use. They should be held to ground either directly or through a resistor of less than 10K ohms.
Input

LinkIn0 - 3 Input pins of the standard links. A **LinkIn** pin receives a serial stream of bits including protocol bits and data bits. Link inputs must be connected to another Link output or tied to **GND**. The Link input must not be tied high or left floating.
Input

LinkOut0 - 3 Output pins of each of the standard links. A **LinkOut** pin may be left floating or may be connected to one (and only one) **LinkIn** pin. As long as the skew specification is met, the connection may be via buffers. (**8.5**)
Output

LinkSpecial — **LinkSpecial** selects the non-standard speed to be either 20MBits/sec when high or 5MBits/sec when low. (**4.1.1**)
Input

Link0Special — Link 0 operates at the non-standard speed selected by **LinkSpecial** if this pin is wired high. (**4.1.1**)
Input

Link123Special — Links 1, 2 and 3 operate at the non-standard speed selected by **LinkSpecial** if this pin is wired high. (**4.1.1**)
Input

MemAD — 32-bit wide multiplexed external memory address and data bus. (**5**)
Input/Output

MemnotRfD1 — This signal is a notRefresh flag to indicate a write cycle. Otherwise this pin is data bit 1.

MemnotWrD0 — This signal is a notWrite flag to indicate a write cycle. Otherwise this pin is data bit 0.

MemConfig — The **MemConfig** input determines how the external memory interface is to be configured. It may be connected to any of the **MemAd** pins directly, or to the output of an inverter which takes its input from any of the **MemAD** pins in order to select the required memory configuration. (**5.6**)
Input

MemReq — This input is used by external devices to access the memory interface **MemAD**. When **MemReq** is sampled high, and if there is no refresh cycle outstanding, the **MemAD** signals are taken high impedance. **MemGranted** is taken high one period **Tm** later. If a refresh cycle is outstanding when **MemReq** is sampled, the refresh cycle is completed before access to the memory interface is granted.
Input

MemGranted — See description below of **MemReq**.
Output

While **MemGranted** is high, **MemReq** is sampled every alternate clock period **Tm**; if it is found to be low, **MemGranted** is taken low two periods **Tm** later, and any pending transputer interface cycle can start. (**6**)

MemWait — Wait input for the memory interface. The **MemWait** signal is sampled synchronously just before the end of **T4**. If, at the time **MemWait** is sampled, the input is low, the interface cycle proceeds. Otherwise, the interface is held in **T4** until the input is sampled and found to be low; one period **Tm** later the interface cycle proceeds to **T5**. (**5.4**)
Input

notMemRd — Used for Read Data, Read Enable, Output Enable for external memory. The **notMemRd** signal is taken low during read cycles, including those which occur during configuration, at the start of **T4**, and is taken high at the start of **T6**. (**5.2**)
Output

notMemRf — Refresh indicator. **notMemRf** goes low a processor cycle before **T1** and remains low until the end of **T6**. (**5.5**)
Output

notMemS0

AddressValid strobe, LatchEnable, or ChipEnable for external memory. The **notMemS0** signal goes low at the start of **T2** and goes high at the start of **T6**. (**5**)
Output

notMemS1

External memory strobe with configurable width, normally

used for RowAddressStrobe, AddressLatchEnable, or ChipEnable. (**5**)
Output

notMemS2 - S4

Three separate external memory strobes with configurable delay. Their falling edge can be configured to go low between zero and 31 periods **Tm** after the start of **T2**, and they go high at the start of **T6**. The conventional use suggested for the strobes is:

notMemS2 use as AddressMultiplex for dynamic RAMs

notMemS3 use as CAS for dynamic RAMs

notMemS4 use as wait state generator

If the system does not use dynamic RAMs, **notMemS2** and **notMemS3** may be used as wait state generators with different delays from **notMemS4**. (**5**)
Output

notMemWrB0 - B3

Separate write strobe for each byte of external memory. The **notMemWrB** signal for each byte only goes low if the relevant byte is to be written. The write strobes may be configured to be either early or late. Early write strobes allow a wide write pulse to static RAM and allow common I/O for dynamic RAM. Late writes allow the data to be strobed by either the falling edge or the rising edge of **notMemWrB**. (**5.3**)
Output

ProcClockOut

The processor clock output, in phase with the memory interface. The processor clock frequency is a multiple of the input clock frequency. The multiple differs for different speed parts. (**5.4**, **7.1**)
Output

Reset

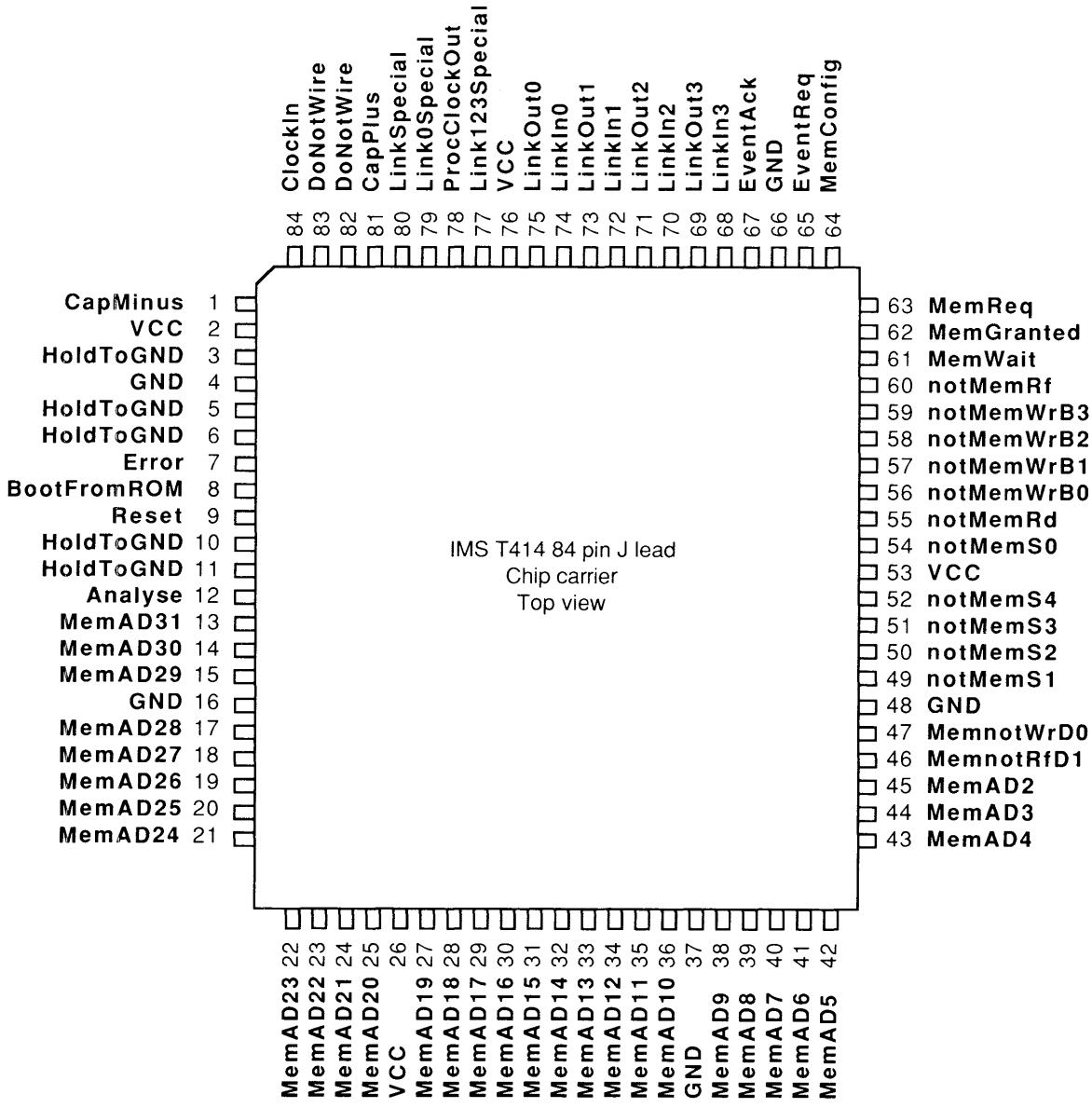
The falling edge of **Reset** initialises the T414, triggers the sequence which configures the memory interface and then starts the processor executing a bootstrap routine. If **Analyse** is asserted during reset, then the memory interface is not reconfigured, and sufficient state is preserved to permit software analysis. (**3.1**)
Input

VCC

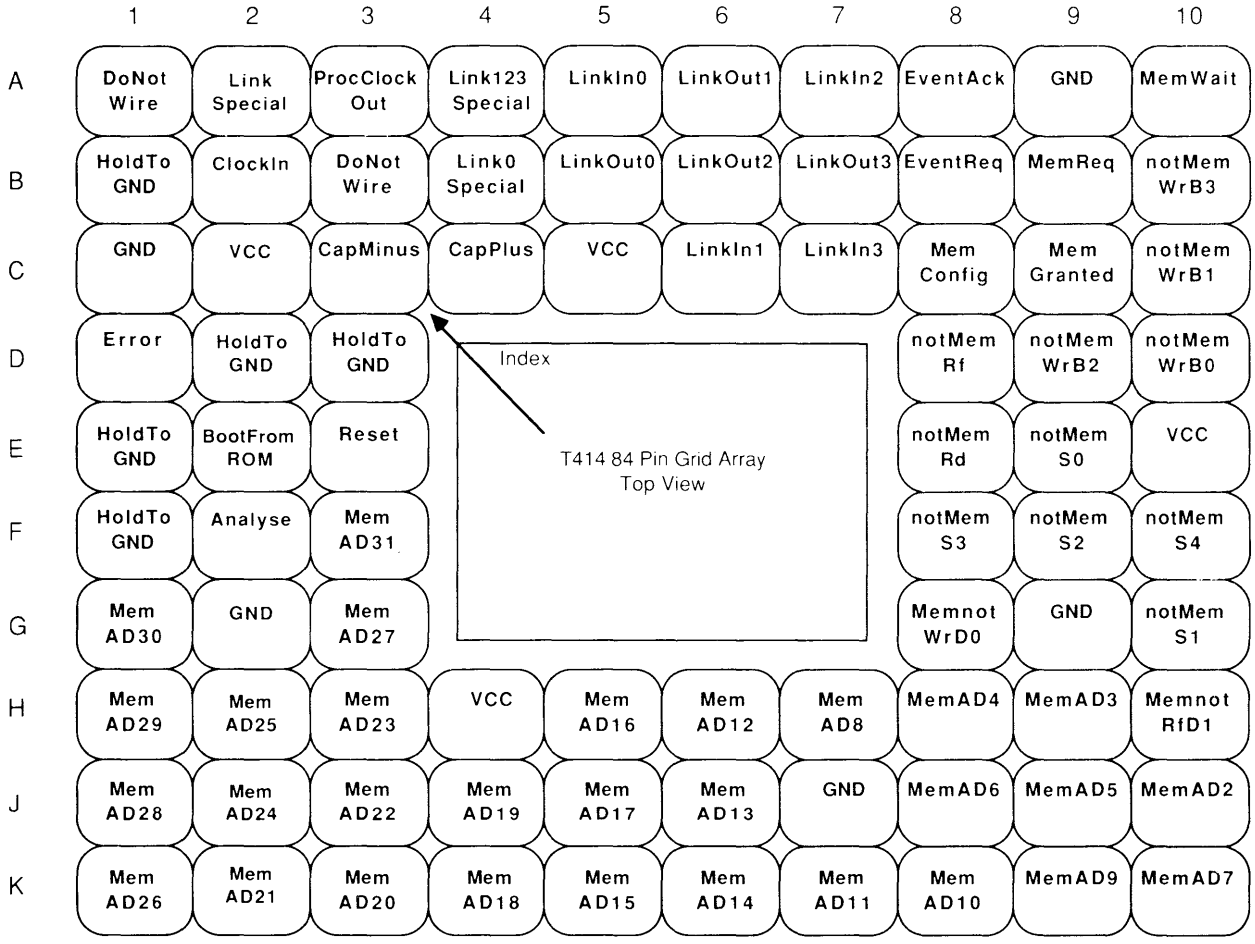
Power supply, nominally 5 V. There are several **VCC** pins to minimize inductance within the package. **VCC** should be decoupled to **GND** by at least one 100 nF low inductance (such as ceramic) capacitor.

The T414 is available in an 84 pin J-Lead chip carrier or 84 pin Grid Array.

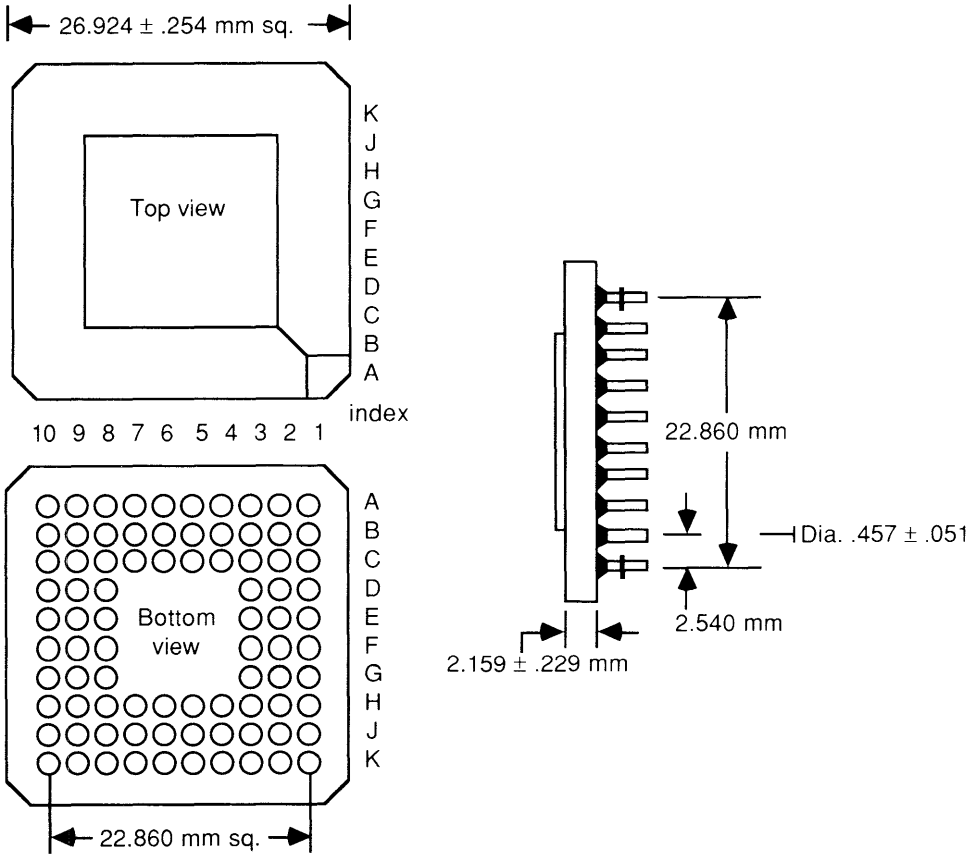
10.1 J-Lead chip carrier



10.2 Pin Grid Array

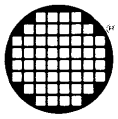


10.3 Package Dimensions



inmos®

Product data



IMS T212 transputer

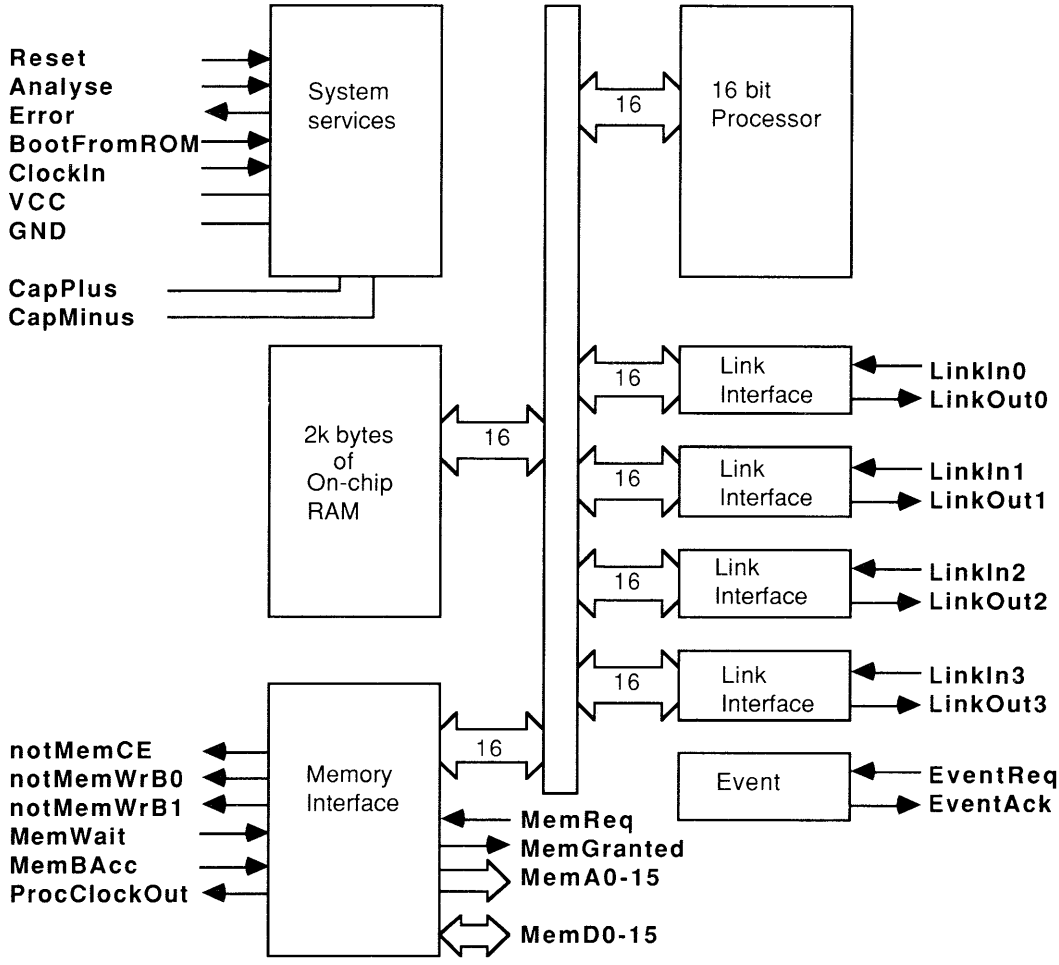
The IMS T212 is a 16-bit member of a family of transputers, all of which are consistent with the INMOS transputer architecture, described in the transputer architecture manual.

This manual details the product specific aspects of the IMS T212 and contains data relevant to the engineering and programming of the device.

Other information relevant to transputer products is contained in the occam programing manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

The 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.

This edition of the manual is dated October 27, 1986.


Reset
Analyse
Error
BootFromROM
ClockIn
VCC
GND
CapPlus
CapMinus
System services
16
16 bit Processor
2k bytes of On-chip RAM
16
16
Link Interface
LinkIn0
LinkOut0
16
Link Interface
LinkIn1
LinkOut1
16
Link Interface
LinkIn2
LinkOut2
16
Link Interface
LinkIn3
LinkOut3
Event
EventReq
EventAck
notMemCE
notMemWrB0
notMemWrB1
MemWait
MemBAcc
ProcClockOut
Memory Interface
16
MemReq
MemGranted
MemA0-15
MemD0-15

IMS T212

The IMS T212 integrates a 16-bit microprocessor, four standard transputer communications links, 2K bytes of on-chip RAM, a memory interface and peripheral interfacing on a single chip, using a 1.5 micron CMOS process.

Processor

The design achieves compact programs, efficient high level language implementation and provides direct support for the occam model of concurrency. Procedure calls, process switching and interrupt latency are all sub-microsecond.

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.

Links

The T212 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.

Memory

The 2K bytes of static RAM provide a maximum data rate of 40 MBytes/sec with access for both the processor and links.

Memory interface

The T212 can directly access a linear address space up to 64 Kbytes, and has a 16-bit wide data bus and a 16-bit wide address bus, non-multiplexed, providing a data rate of up to 20 MBytes/sec, and supporting word or byte organisation. The data bus can be dynamically configured to be 16-bits or 8-bits wide.

Peripheral interface

The memory controller supports memory mapped peripherals, which may use DMA. Links may be interfaced to peripherals via an INMOS link adaptor. A peripheral can request attention via the event pin.

Time

The processor includes timers for both high and low priority processes.

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. The error can be handled internally by software or externally by sensing the error pin. System state is preserved for subsequent analysis.

The optimal method of programming the T212 processor is to use occam. See the occam programming manual for full details. Several standard languages are also supported. These include C, Fortran and Pascal.

The processor provides direct support for the occam model of concurrency and communication. It has a scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. The number of registers which hold the process context is small and this, combined with fast on-chip RAM, provides a sub-microsecond process switch time.

Process communication is implemented by memory to memory block move operations. These fully utilize the bandwidth available from the on-chip RAM.

2.1 T212 types

The implementation of occam for the T212 transputer supports the following types:

CHAN OF *type* Each communication channel provides communication between two concurrent processes. Each channel allows the communication of data of the specified type.

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 **n** in the range $0 <= \mathbf{n} < 256$

INT Signed integers **n** in the range $-32768 <= \mathbf{n} < 32768$

INT16 Signed integers **n** in the range $-32768 <= \mathbf{n} < 32768$

INT32 Signed integers **n** in the range $-2^{31} <= \mathbf{n} < 2^{31}$

INT64 Signed integers **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 T212 process multiplexing

The T212 supports two levels of priority - in occam notation, a **PRI PAR** (priority parallel) process may have two components. The priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

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.

Low priority processes

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 its 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.

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

To ensure that each low priority process proceeds, high priority processes should never occupy the processor continuously for a period of time equal to a timeslice period. A good guideline is to ensure that, if there are a total of **n** high priority processes, then each limits its activity to much less than 1/**n**'th of any 1 millisecond period.

Interrupt latency

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, maximum 53 cycles (assuming use of on-chip RAM).

2.3 T212 Error flag

Expressions which cause arithmetic overflow are invalid, and processes which cause array bound violations are invalid. If the compiler is unable to check that a given construct contains only valid expressions and processes, then extra instructions are compiled to perform the necessary checks at runtime. If the result of the check indicates that an invalid expression or invalid process has occurred, then the processor's **Error** flag is set.

In the implementation of occam, the offending process stops when the **Error** flag is set.

The T212 can be initialised so that the processor halts when the **Error** flag is set. The appropriate initialisation sequence is provided by the development system.

If the processor has been halted as the result of an error, the links continue with any outstanding transfers, and the transputer may be analysed. See section 3.2.

When a high priority process pre-empts a low priority process, the current value of the **Error** flag is saved; the **Error** flag itself is maintained. When, finally, there are no high priority processes able to run, the current state of the **Error** flag is lost, and the preserved state is restored as part of commencing to execute the pre-empted low priority process.

This ensures that the state of the **Error** flag is preserved during the evaluation of an expression or a sequence of assignments.

2.4 T212 memory map

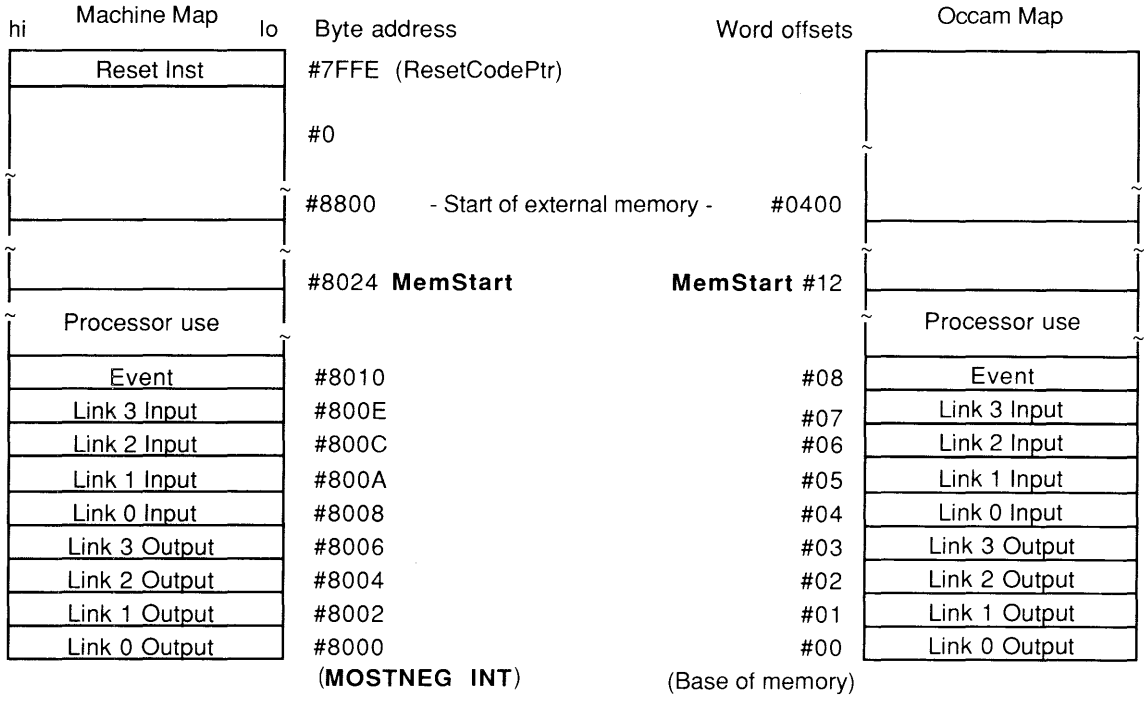
The address space of the T212 is signed and byte addressed. Words are aligned on two-byte boundaries. Addresses in the range #8000 to #87FF reference on-chip memory.

The first 18 words of the address space are used for system purposes. The next available location is, by convention, referred to as **MemStart**.

A suitable declaration of **MemStart**, for example is

```
VAL MemStart IS #12 :
```

The programmer can access locations in memory by using the occam mechanism of placement. This allows variables of any type to be placed at a location specified as a word offset from the base of memory. The placement of a byte array allows access to the byte components of a word. For example, a byte array could be placed in internal memory to optimize access, or a similar array could be placed in external memory to address one or more memory mapped devices.



This schema for calculating addresses is used to provide word length independent code, as though memory were an array of type **INT** (**[]INT**). The link addresses will always be found within the lower bounds of the memory array space.

The top of address space is used, by convention, for ROM based code. If the transputer is configured to bootstrap from ROM, then the processor commences execution from address #7FFE. Tools are supplied with the compiler and development system to enable the generation and placement of ROM images.

2.5 T212 timer

At the standard **ClockIn** cycle rate of 5MHz, the high priority timer has a resolution of one microsecond and cycles approximately every 65 milliseconds.

The low priority timer has a resolution of 64 microseconds. One second is exactly equal to 15625 ticks. Cycles approximately every four seconds.

2.6 T212 event pins

An attention-seeking peripheral may signal the T212 via the **EventReq** pin, which the T212 handshakes using **EventAck**. The T212 implements a hardware channel to allow a low to high transition on the **EventReq** pin to be communicated to a process as a synchronizing message.

An occam channel (which must already have been declared) may be associated with **EventReq** pin by a channel placement. The conventional name and the value used for this channel are given by

```
PLACE Event AT 8 :
```

Event behaves like an ordinary occam channel, and a process may synchronize with a low to high transition on the **EventReq** pin by using the occam construct:

```
Event ? signal
```

The process waits until the channel **Event** is ready. The channel is made ready by the transition on the **EventReq** pin (this may occur before the process attempts to input).

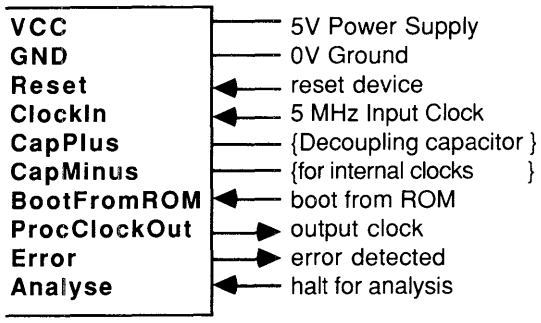
When the process is able to proceed, and if it executes at high priority, then it will take priority over any low priority process which may be executing when the transition occurs on the **EventReq** pin.

2.7 T212 link placement

The link addresses will always be found within the lower bounds of the memory array. The conventional names and the values used for these channels are

```
PLACE Link0Output AT 0 :
PLACE Link1Output AT 1 :
PLACE Link2Output AT 2 :
PLACE Link3Output AT 3 :
PLACE Link0Input  AT 4 :
PLACE Link1Input  AT 5 :
PLACE Link2Input  AT 6 :
PLACE Link3Input  AT 7 :
```

The system services comprise the clocks, power and initialisation used by the whole of the transputer. The **Reset** and **Analyse** inputs enable the T212 to be initialised, for example on power up, or halted in a way which preserves its state for subsequent analysis. Whilst the T212 is running both **Reset** and **Analyse** are held low. The **Error** signal is directly connected to the processor's **Error** flag. See section 2.3.



3.1 Reset

The T212 is initialised by pulsing **Reset** high whilst holding **Analyse** low. Operation ceases immediately and all state information is lost.

The processor then bootstraps. If the **BootFromROM** input is high it will start to execute code starting from address #7FFE. If **BootFromROM** is low it will bootstrap from a link, that is, it will load a program from a link and then execute it.

When initialising following power-on, a time is specified during which **VCC** must be within specification, **Reset** must be high, and the input on **ClockIn** must be oscillating. **Reset** is taken low after the specified time has elapsed.

During power on reset all link inputs must be held low. (N.B. all link outputs are made low by reset.)

3.2 Analyse

A system built from transputers may be brought to a halt in a consistent state which is preserved for subsequent analysis. This analysis is performed in a manner similar to bootstrapping. The transputer development system includes appropriate bootstrap and analysis software.

A signal may be applied to the **Analyse** inputs of all the transputers in the system. The system is analysed by first taking **Analyse** high. This causes each transputer to halt, after a short period of time, in a consistent internal state. When the system has halted, **Reset** is taken high for the specified hold time, after which it is taken low. **Analyse** is then taken low, at which time each transputer bootstraps.

When **Analyse** is taken high, the processor will halt within three timeslice periods (approximately three milliseconds), plus the time taken for any high priority process to cease processing. Any outputting links continue until they complete the remainder of the current word. Input links will continue to receive data. Provided that there are no delays in sending acknowledgements, the links in a system will therefore cease activity within a few microseconds. Sufficient time must be allowed to allow the processor to halt and link traffic to cease before **Reset** is asserted.

The system must be designed so that links connected to the external world are quiet during reset.

3.3 Bootstrapping and analysis of a "failed" system

The transputer has two methods of bootstrapping. Firstly, the conventional bootstrap which occurs after **Reset** and secondly, an analysis bootstrap which occurs after **Reset** plus **Analyse**. This second method allows the state of a system, which could well be a complex network of transputers, to be examined. For example, memory continues to be refreshed so that data is not lost. In both cases the mechanism used is very similar and the bootstrap or analysis code may be provided from either the external memory of the transputer, often in ROM, or if the transputer is a part of a network, via its communication links.

3.3.1 Bootstrapping

It is possible to bootstrap the T212 either by executing code held in ROM or by executing code received on a link. In addition, prior to bootstrapping from a link, it is possible to read or write to any memory location in a transputer's memory map via a link.

3.3.2 Bootstrapping from ROM

To bootstrap from ROM, the **BootFromROM** input is wired to **VCC**.

The T212 bootstraps from ROM by executing a process at low priority. Control is transferred to the top two bytes in memory (at #7FFE), which will invariably contain a backward jump into ROM. **`Memstart`** (#8024) is used as the location of the process workspace.

3.3.3 Bootstrapping from a link

To bootstrap from a link, the **BootFromROM** input is wired to **GND**.

The T212 bootstraps from a link by waiting for the first byte (the control byte) to arrive on any of the four link inputs.

If the value of the control byte is two, or greater, it is considered to be a count of the number of bytes to be input. The following bytes are then placed into memory at **`MemStart`** (#8024) and the T212 begins to execute the input code as a process at low priority by transferring control to **`MemStart`**. The memory space immediately above the loaded code is used as the process workspace.

The mechanism of bootstrapping from any link allows a network of transputers to be bootstrapped without the need for ROM or any external memory.

3.3.4 Peeking and Poking

A unique feature of the transputer allows any location of transputer memory, be it internal or external, to be read or written to from a link. This allows, with appropriate software, an external memory system to be debugged without the need to run a program on the system under development.

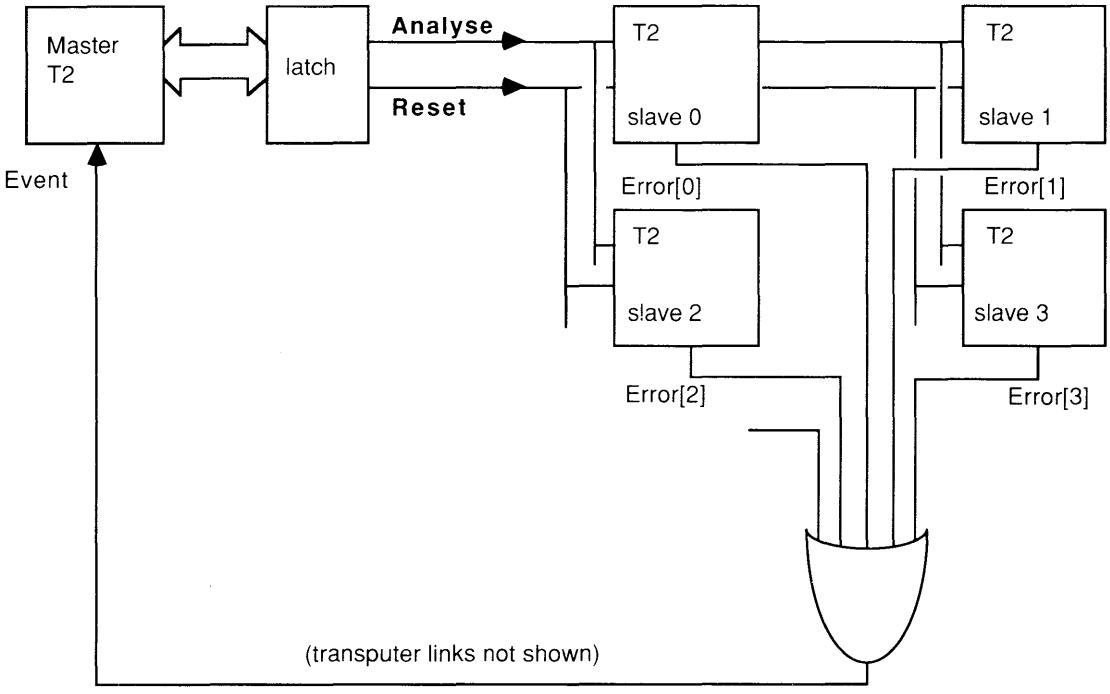
If, whilst the transputer is waiting to boot from a link, it receives a zero byte then a word address is input, followed by a word of data which is written to that address. The transputer then returns to the state of awaiting a message from any link.

If the first byte received is one, then a word address is input, a word of data is read from that address and is output down the corresponding output link. The transputer will then return to the state of awaiting a message from any link.

3.4 Using Error and Analyse

Analyse may be used in conjunction with the **Error** output to isolate errors in a multi-transputer system. A transputer which has signalled an error may be halted and subsequently analysed under the control of a 'master' transputer, using the methods described above. Or this could be achieved by connecting the 'OR' of all the **Error** pins to the **EventReq** pin on the master transputer, and connecting the **Analyse** and **Reset** pins to the master transputer as a 'peripheral'.

Error treatment in multi-transputer system

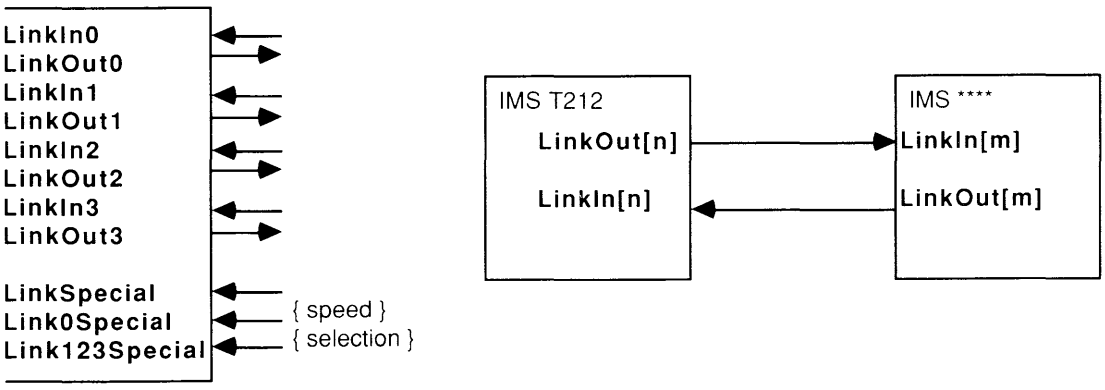


4.1 Standard transputer links

The T212 provides four standard links, each providing two uni-directional point-to-point occam channels.

The links implement the standard inter transputer communications protocol. The links are connected by wiring a **LinkOut[n]** to a **LinkIn[m]** and **LinkIn[n]** to a **LinkOut[m]**. Unused link inputs should be held to ground. **GND**.

Link connection



The T212 links wait until each full byte has been received before outputting the corresponding acknowledge packet. An acknowledge packet can be received at any time following the transmission of a data packet start bit.

At a link speed of 10Mbits/sec, data is transmitted at about 400Kbytes/sec in each direction, and the combined (bidirectional) data rate when the link carries data in both directions at once is 800Kbytes/sec.

4.1.1 Link speed selection

Link speed selection depends on the settings of three pins, **Link0Special**, **Link123Special** and **LinkSpecial**. These are shown in the tables below, a 0 indicating the signal should be held to **GND**, and a 1 indicating the signal should be held to **VCC**.

Table 1. Speed settings for Link 0.

LinkSpecial	**Link0Special**	**Speed at 5MHz** Mbits/sec	**Unidirectional data rate** Kbytes/sec	**Bidirectional data rate** Kbytes/sec
0	0	10	400	800
0	1	5	200	400
1	0	10	400	800
1	1	20	800	1600

Table 2. Speed settings for Links 1,2 and 3.

LinkSpecial	**Link123Special**	**Speed** Mbits/sec	**Unidirectional data rate** Kbytes/sec	**Bidirectional data rate** Kbytes/sec
0	0	10	400	800
0	1	5	200	400
1	0	10	400	800
1	1	20	800	1600

The data rates are reduced when performing transfers using slow external memory.

The memory interface has a 16-bit address bus and a 16-bit data bus. Word reads and writes are performed in two processor cycles, a processor cycle being defined as one cycle of **ProcClockOut**.

Wait states can be introduced by taking **MemWait** high. This signal is sampled during the first processor cycle of the memory access. If it is found to be high, processor cycles are added to the memory access until subsequent sampling detects **MemWait** to be low, at which time the memory access proceeds normally.

The T212 memory interface will by default perform word access at even memory locations. Byte accessing can be achieved by taking **MemBAcc** high. The state of this signal is latched during the second half of the first cycle. The first byte is accessed at the word address, and the second byte is accessed at the word address + 1 at which time **MemA0** is high.

Two write enable signals control which byte of data is to be written by the memory interface. These are active low and are called **notMemWrB0** and **notMemWrB1**.

The active low signal **notMemCE** is used to enable external memory on both read and write cycles. External memory cycles are divided into 4 T-states, **T1** to **T4**. Each T-state can be defined as follows:

T1 Address and Write Enables set-up period.
T2 Data set-up for writes.
T3 Access time extension for slow RAMs.
T4 Bus turnround, end of cycle.

Note: When internal memory is accessed **MemA0-15** takes the value of the internal address and **notMemWrB0-1** takes the value of the interupt write enables. The signal **notMemCE** however is not asserted.

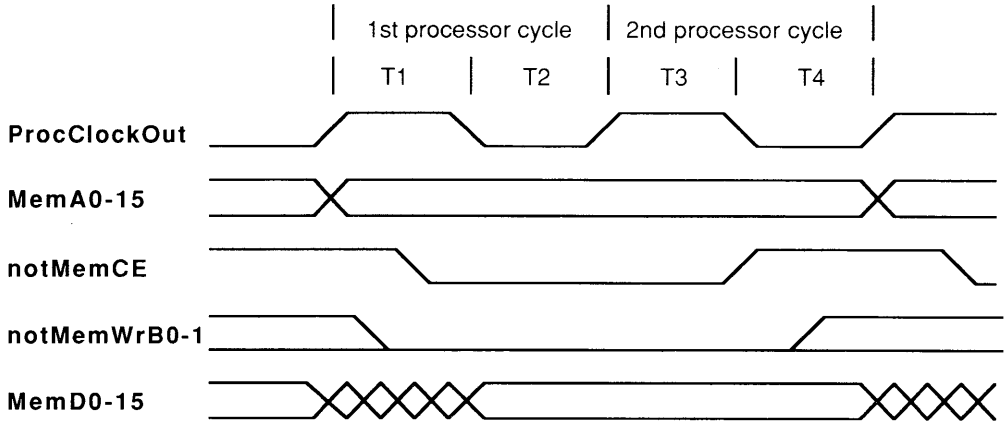
5.1 Word access

This is the default mode and also the more efficient. The processor uses **notMemWrB0** and **notMemWrB1** on write cycles to select which bytes are to be written.

Write cycles

The processor supports early write cycles in both word and byte accessing modes. Write enables are asserted before the chip enable signal **notMemCE** is set low. This reduces memory access time and therefore the risk of bus contention.

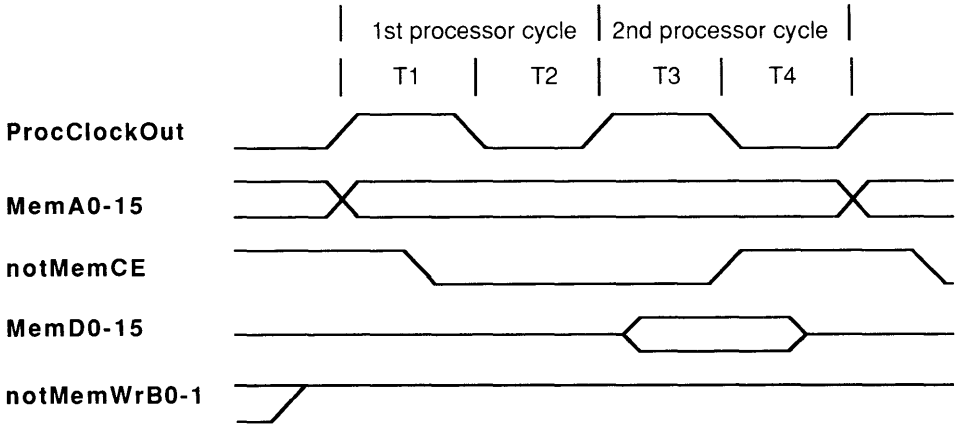
Word Write Cycle



Read cycles

During this cycle the data bus is allowed to float from the time that a write enable is deasserted until the memory system or peripherals drive the data bus. Memory data is latched at the end of **T3** and must be held until **notMemCE** has gone high.

Word Read Cycle



5.2 Byte access

Byte access mode uses the **MemD0-D7** pins to access byte data buses.

The processor performs two cycles for each read during byte access. The least significant byte of a word is expected on **MemD0-D7** on the first cycle and the most significant byte is expected on **MemD0-D7** on the second cycle.

Similarly, the processor performs two cycles for each write during byte access. The least significant byte of a word is placed on **MemD0-D7** on the first cycle and the most significant byte is placed on **MemD0-D7** on the second cycle.

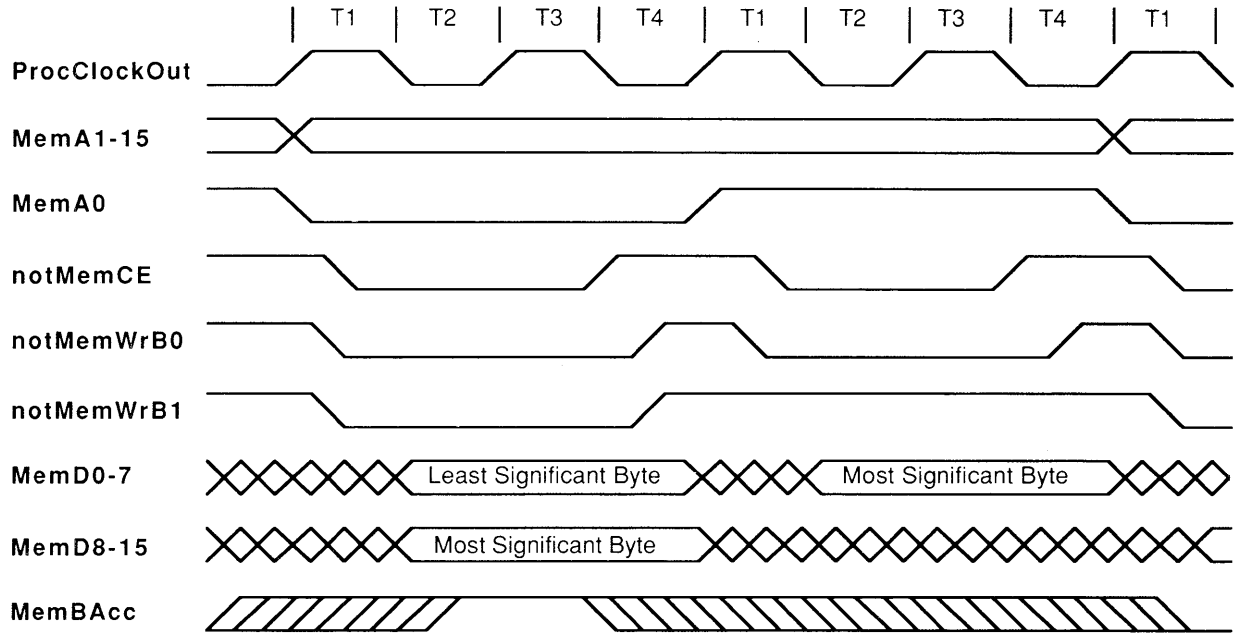
When the processor accesses a single byte, the interface still performs two cycles, one of which is a dummy cycle.

To enable byte access, **MemBAcc** must be taken high before the end of **T2**. **MemBAcc** must be asserted before **T3** of the first byte access, and can be deasserted at the beginning of **T4** on the second byte access.

As the processor still expects 16-bit data internally, the memory interface deals with the task of assembling 2 bytes into a word and vice versa, and asserting the appropriate control strobes. This means that in byte access mode two memory accesses are performed each time the memory interface reads or writes data.

MemA0 is low for the least significant byte access and high for the most significant byte access.

Write Cycle in Byte Access Mode



5.3 The use of MemWait

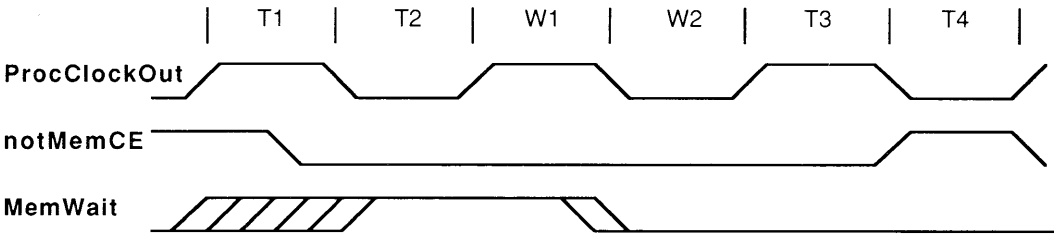
Memory cycle times can be extended by asserting **MemWait** at the correct time.

MemWait is sampled during **T2** and so must be set up before this state occurs.

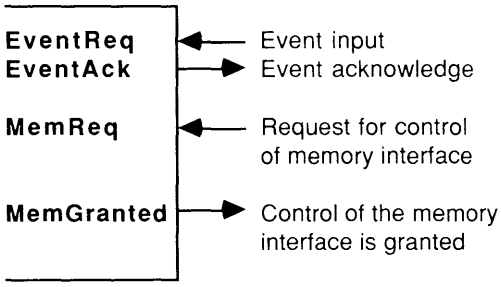
The memory interface of the T212 has been designed to interface to a simple wait state generator such as a digital delay line.

When **MemWait** is held high, the memory interface will add a further two Tstates. Further cycles will be added as long as **MemWait** is held high during **T2**.

Single Wait State Timing

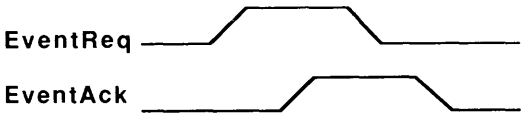


Event Request block diagram



The two event signals, **EventReq** and **EventAck**, together provide a handshaken interface with an occam process executing in the processor.

Event request signal protocol



External logic takes **EventReq** high when the logic wishes to communicate with a process in the transputer. The rising edge of **EventReq** makes an external channel ready to communicate with the process. (This channel is additional to the external channels of the links.) When both the channel is ready and a process is ready to input from the channel, then the processor takes **EventAck** high and the process is scheduled. At any time after this point the external logic may take **EventReq** low, following which the processor will set **EventAck** low. After **EventAck** goes low, **EventReq** may go high to indicate the next event. Any further communication or synchronization necessary (for example, to tell external logic that the process has acted in response to the event) must be programmed explicitly.

If the process has high priority, and there is no other high priority process already running, then the maximum latency is 53 processor cycles, assuming that all memory accesses are to on-chip RAM. The typical latency is 19 processor cycles.

EventReq should be held low on reset.

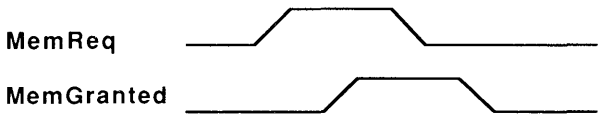
DMA via the memory interface

Peripheral controllers may access the whole of the external memory address space for DMA transfers using the asynchronous signals **MemReq** and **MemGranted**, which provide a handshake protocol for requesting and acknowledging control of the external address and data buses.

If **MemReq** is taken high the processor will finish the current memory cycle before allowing a peripheral to take control of the external data and address buses. In byte access mode, this means that control will not be given until the 2nd cycle has been performed.

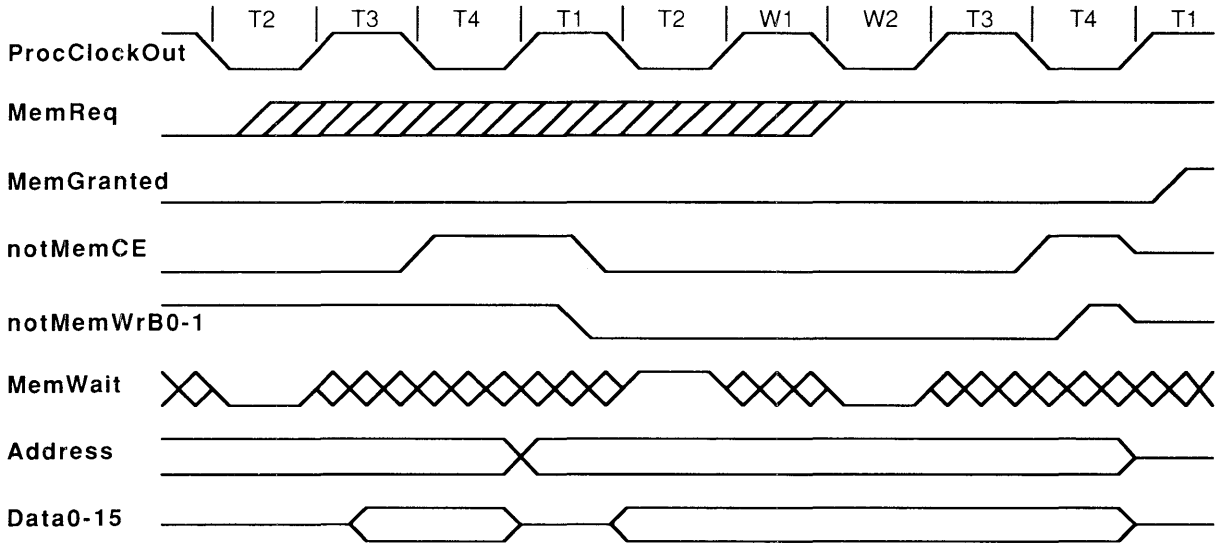
When the memory interface has completed its cycle, the T212 tristates the bus and asserts **MemGranted**. The peripheral cannot use the interface until it has received **MemGranted**. When the peripheral device deasserts **MemReq**, the processor releases **MemGranted** and starts memory access.

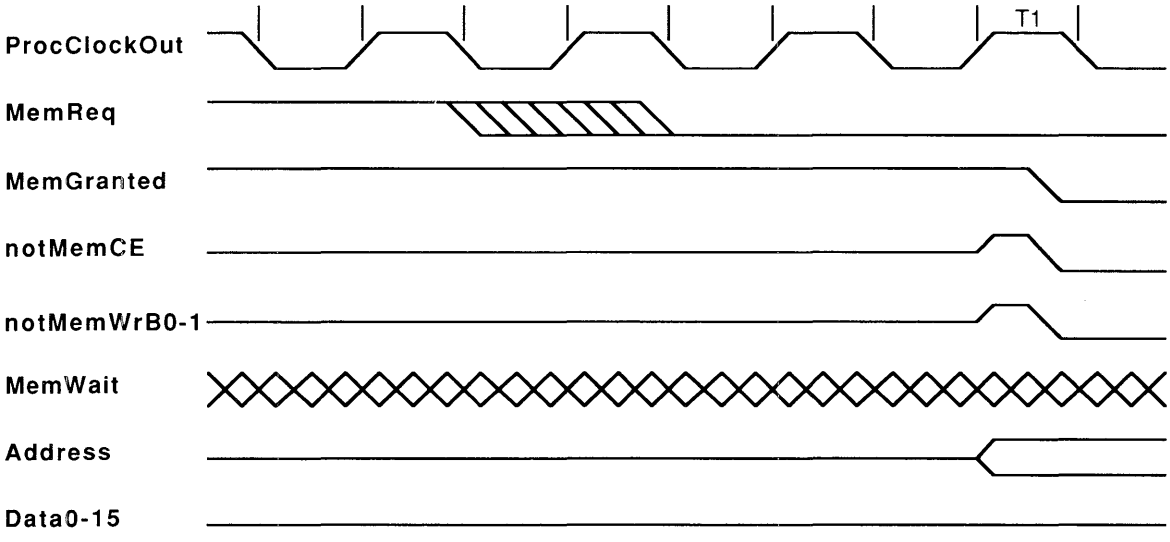
Memory Request Handshake



The processor and links can still continue to operate to internal memory while DMA proceeds to external memory, provided the processor or links do not try to use the external memory during DMA. The acknowledge signal **MemGranted** will function regardless of the state of **Reset** and **Analyse**.

Memory Request including Wait State





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.

7.1 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 following abbreviations are used to represent the quantities indicated.

np number of component processes
ne number of processes earlier in queue
r 1 if INT 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 multiplier is 0)

Performance table

	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** 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 *type*	3.1	3.1
[size]CHAN OF *type*	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**

	Size (bytes)	**Time** (cycles)
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
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 call		
	3.5+(nparams-2)*1.1 +nvecparams*2.3	16.5+(nparams-2)*1.1 +nvecparams*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}

Figures in curly brackets 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.

7.1.1 Fast multiply, TIMES

The T212 has a fast multiplication instruction ('product'). If **Tb** is the position of the most significant bit set in the multiplier, then 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 occam on the transputer take advantage of this instruction. The modulo operator **TIMES** is mapped onto the instruction and is treated as non-commutative, the expression on the right of the expression being the multiplier.

The fast muliplication instruction is also used in occam implementations for the multiplication implicit in multi-dimensional array access.

7.1.2 T212 arithmetic procedures

A set of predefined procedures (**PROC**s) are provided to support the efficient implementation of multiple length and floating point arithmetic. In the following table, **n** gives the number of places shifted and all parameters are assumed to be local. Full details of these procedures are provided in the occam reference manual supplied as part of the development system and available as a separate publication.

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

PROC		Cycles	+Cycles for Parameter Access (Assuming local variables)
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

7.1.3 Floating point operations

Floating point operations are provided by a run-time package, which requires approximately 2000 bytes of memory for the double length arithmetic operations, and 2500 bytes for the quadruple length arithmetic operations. The following table summarizes the estimated performance of the package.

		Processor cycles (typical)	Processor cycles (worst)
REAL32	+, -	530	705
	*	650	705
	/	1000	1410
	<, >, =, >=, <=, <>	60	60
REAL64	+, -	875	1190
	*	1490	1950
	/	2355	3255
	<, >, =, >=, <=, <>	60	60

7.1.4 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** is 1 for no wait states.

The number of additional cycles required to access data in external memory is **e**. 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 (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 the following table, **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**.

	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**)

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.

	e	1	2	3	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

7.2 T212 speed selections

The following table illustrates the designation of the T212 speed selections.

Designation	**Instruction throughput**	**Processor clock speed**	**Processor cycle time**	**Input clock frequency**
IMS T212-17	8.75 MIPS	17.5 MHz	57 ns	5 MHz
IMS T212-20	10 MIPS	20 MHz	50 ns	5 MHz

Parameters given in this section will be revised as a result of fuller characterization.

8.1 Absolute maximum ratings

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	0	7.0	V	1, 2, 3
VI,VO	Input or output voltage on any pin	-0.5	**VCC**+0.5	V	1, 2, 3
OSCT	Output short circuit time (one pin)		1	S	1
TS	Storage temperature	-65	150	°C	1
TA	Ambient temperature under bias	-55	125	°C	1
PD	Power dissipation rating		1	W	

Notes

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

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

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

8.2 Recommended operating conditions

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	4.5	5.5	V	1
VI,VO	Input or output voltage	0	**VCC**	V	1,2
II	Input current		±25	mA	3
CL	Load capacitance on any **MemAD** pin		50	pF	
TA	Operating temperature range	0	70	°C	

Notes

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

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

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

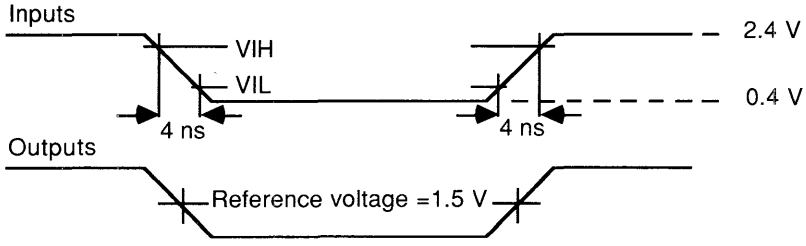
8.3 DC characteristics

Parameter		Conditions	Min	Max	Unit
VIH	High level input voltage		2.0	**VCC**+0.5	V
VIL	Low level input voltage		-0.5	0.8	V
II	Input current	**GND**<**VI**<**VCC**		±10	μA
VOH	Output high voltage	**IOH**=2mA	**VCC**-1		V
VOL	Output low voltage	**IOL**=4mA		0.4	V
IOS	Output short circuit current	**GND**<**VO**<**VCC**		50	mA
IOZ	Tristate output current	**GND**<**VI**<**VCC**		±10	μA
PD	Power dissipation			0.7	W
CIN	Input capacitance	f=1 MHz		7	pF
COZ	Output capacitance tristate	f=1 MHz		10	pF

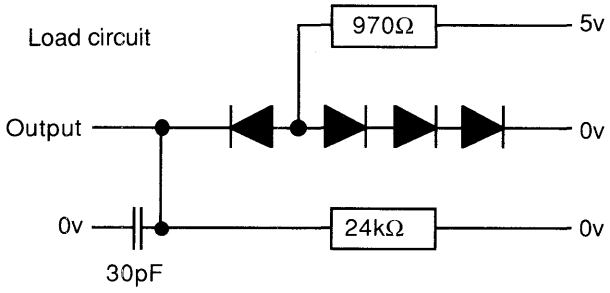
Note :
4.5 V < **VCC** < 5.5 V
0 °C < **TA** < 70 °C
input clock frequency = 5 MHz
All voltages are with respect to **GND**

8.4 Measurement of AC characteristics

Reference points for AC measurements



Load circuit for AC measurements



The load circuit approximates to two Schottky TTL loads, with total capacitance of 30 pF.

8.5 Connection of INMOS serial links

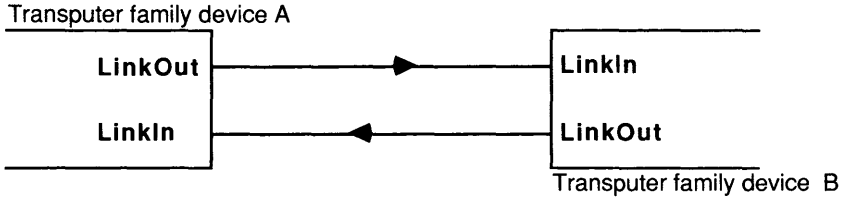
INMOS serial links can be connected in 3 different ways depending on their environment:

1 Directly connected

2 Connected via a series matching resistor

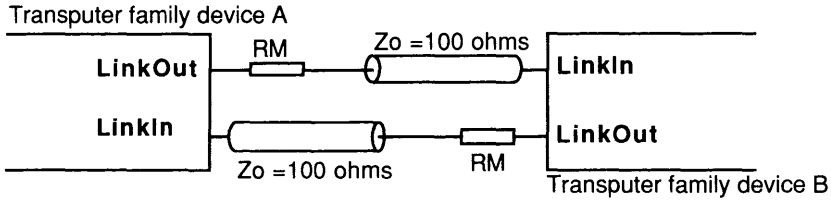
3 Connected via buffers

Direct connection



Direct connection is suitable for short distances on a printed circuit board.

Matched line

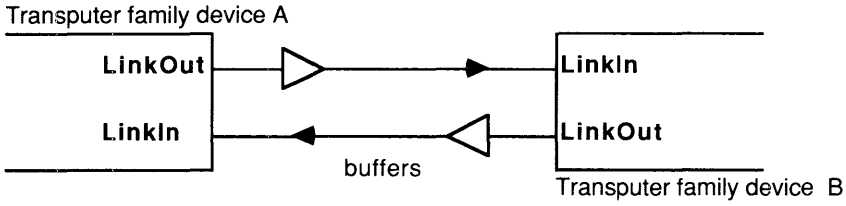


For long wires, approximately >30 cm, then a 100ohm transmission line should be used with series matching resistors.

Parameter		Nom	Max	Unit
RM	Series matching resistor for 100 ohm line.	47		ohm
TD	Delay down line		0.4	bit time

Note that if two connected devices have different values for **TD**, the lower value should be used. With series termination at **LinkOut** the transmission line reflection must return within 1 Bit time. Otherwise line buffers should be used.

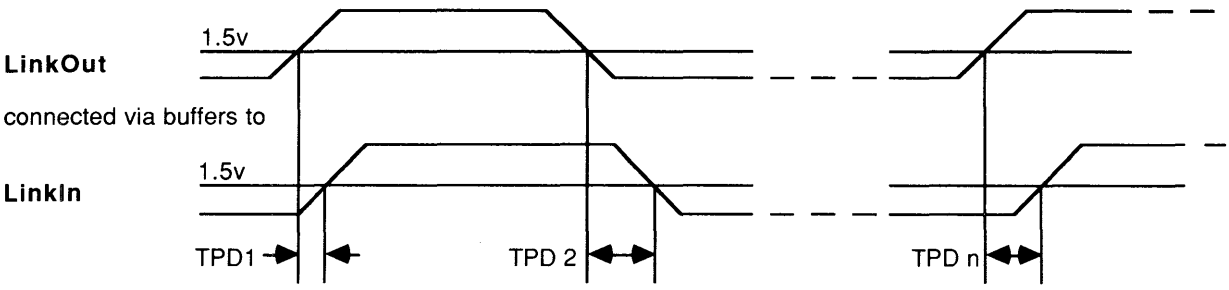
Buffered links



If buffers are used their overall propagation delay, TPD, should be stable within the skew tolerance.

Parameter	Max	Unit
Skew in buffering at 5 Mbits/sec	30	ns
Skew in buffering at 10 Mbits/sec	10	ns
Skew in buffering at 20 Mbits/sec		ns
Rise and fall time of **LinkIn** (10% to 90%)	20	ns

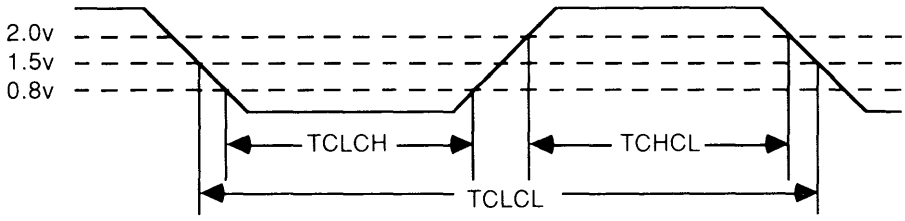
The above figures indicate that buffered links can be realised at 5Mbits/sec and 10Mbits/sec. For the case of 20 Mbits/sec, the maximum value can only be specified as a result of further characterisation.



The absolute value of TPD is immaterial because data reception is asynchronous. However, TPD will vary from moment to moment because of ground noise, variation in the power supplies of buffers and the difference in the delay for rising and falling edges. This will vary the length of data bits reaching **LinkIn**. *Skew* is the difference between the maximum and minimum instantaneous values of TPD.

8.6 AC characteristics of system services

ClockIn waveform



Parameter		Min	Nom	Max	Unit	Note
TCHCL	Clock pulse width high	40			ns	1
TCLCH	Clock pulse width low	40			ns	1
	Rise and fall time of **ClockIn** (10% to 90%)			10	ns	1
TCLCL	Clock period		200		ns	2
	Difference in frequencies of **ClockIn** for two devices connected by a link			400	ppm	6
Cap	Decoupling capacitor between **CapPlus** and **CapMinus**	1			μF	3
TRHRL	**Reset** pulse width high	8			**ClockIn** periods	4
	Time **VCC** must be valid and **Clockin** running before **Reset** goes low	10			ms	5
	BootFromROM setup time before **Reset** or **Analyse** goes low	0				
	BootFromROM hold time after **Reset** goes low	50			ms	
	Analyse pulse width	8			**Clockin** periods	

Notes

1 The clock transitions must be monotonic within the range between **VIH** and **VIL**.

2 The **TCLCL** parameter is measured between corresponding points on consecutive falling edges.

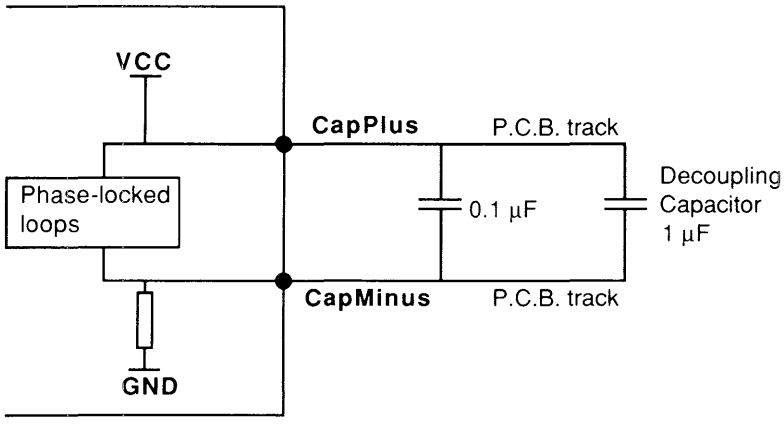
3 A 1 μF tantalum or ceramic low inductance, low leakage capacitor must be connected between **CapPlus** and **CapMinus** to decouple the supply to the on-chip clock generator. An additional good quality 0.1 μF ceramic capacitor should be connected in parallel with this. PCB track lengths to the capacitor should be minimised. No power supply should flow in these tracks.

4 Link inputs must be held low during reset. Reset forces link outputs low.

5 At power on reset.

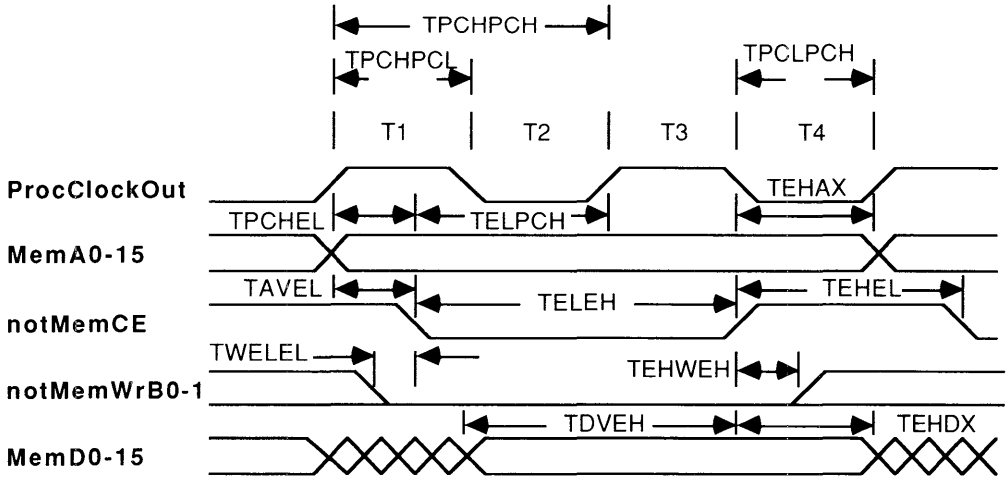
6 This value allows the use of low cost 200ppm crystal oscillators.

Recommended PLL Decoupling.

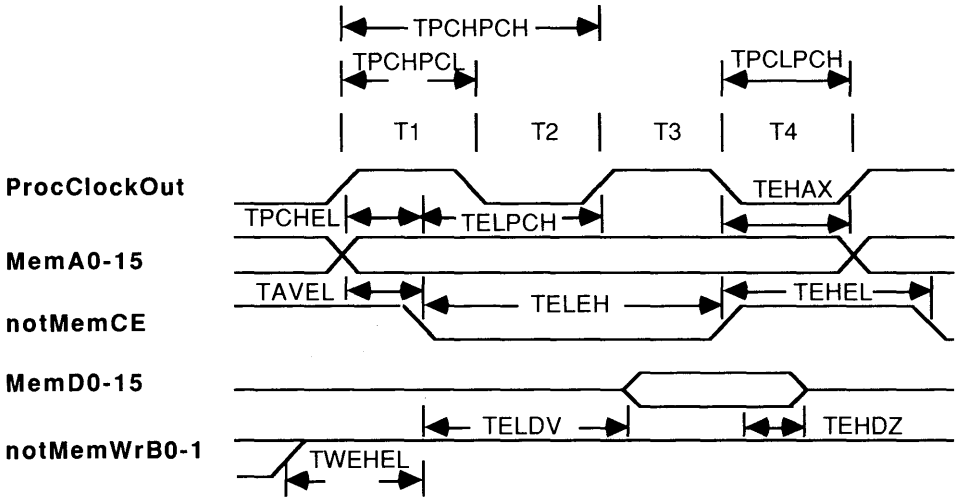


8.7 Memory interface AC characteristics

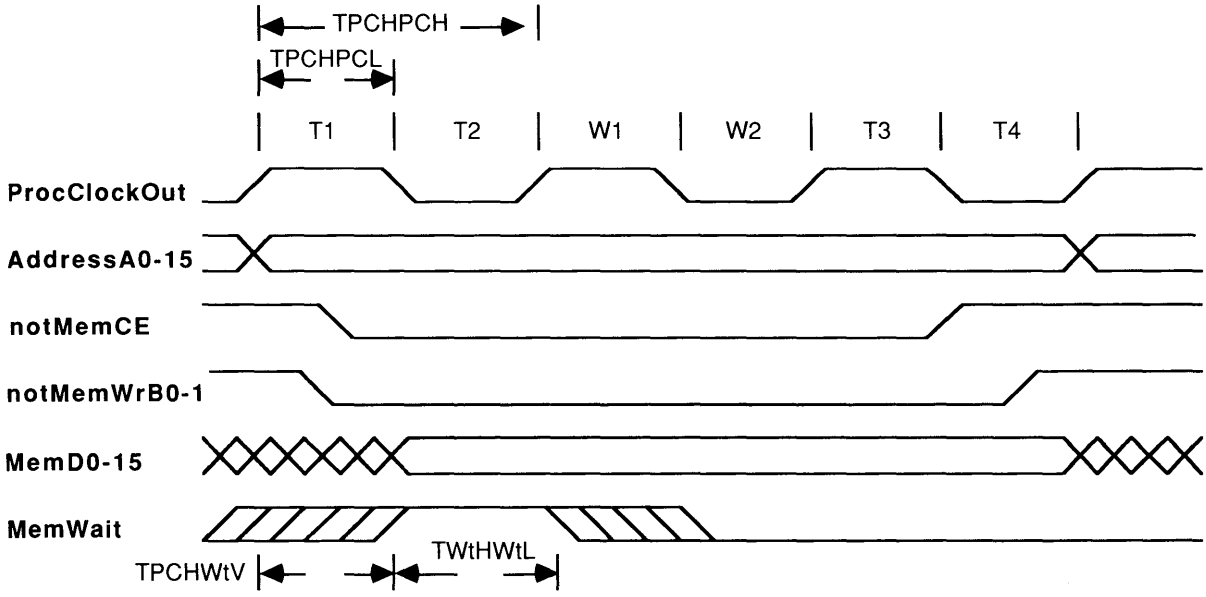
Word Write Cycle A.C timing.



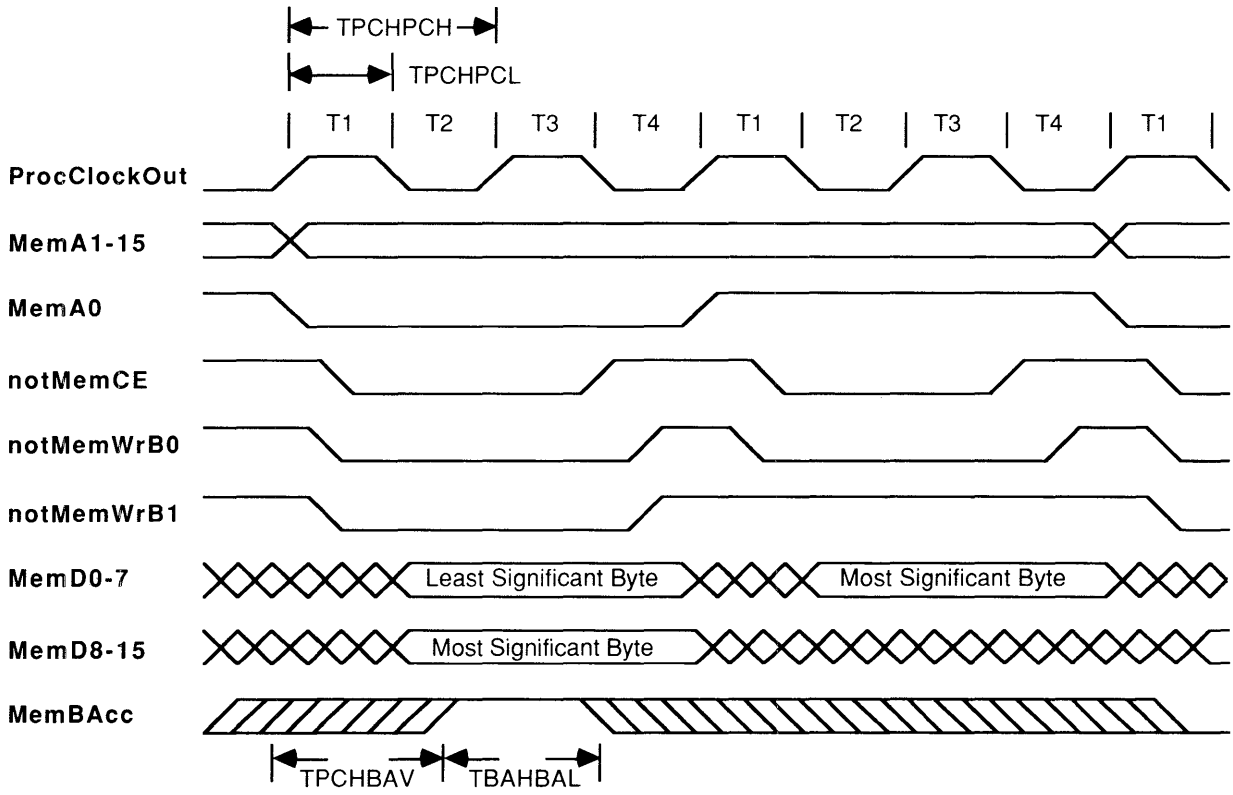
Read Cycle A.C timing.



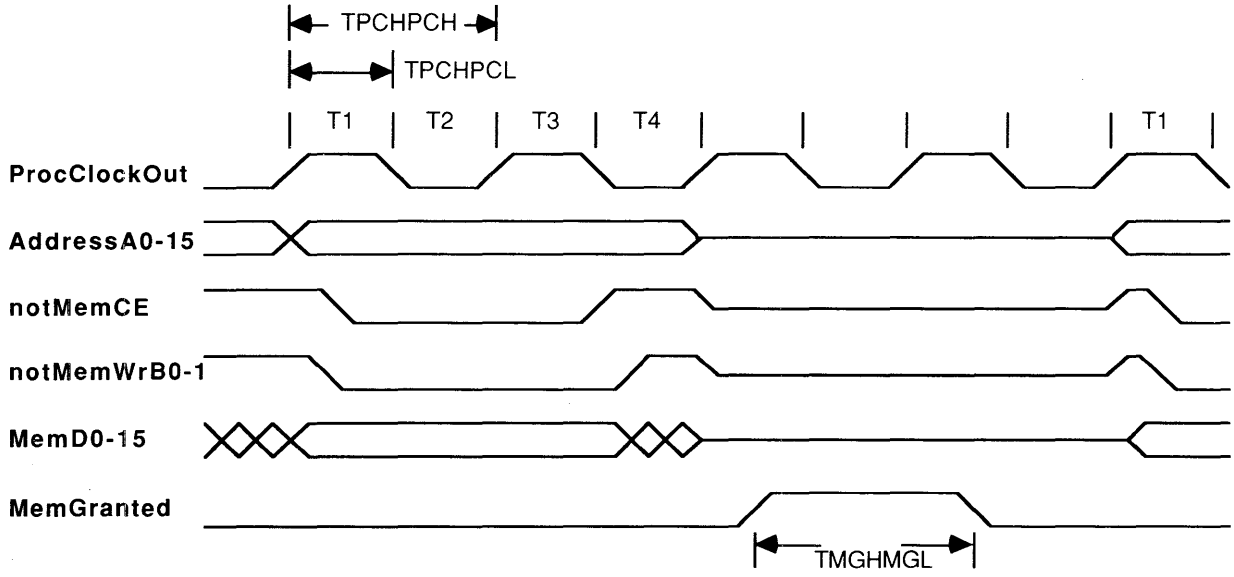
A.C timing for a single Wait State during a Write Cycle.



A.C timing to assert Byte Access.



A.C timing to assert Memory Request.



The AC characteristics of the memory interface are dependent upon the speed of the transputer and the use of wait states.

Note

The parameter table can be used to calculate figures for all T212 speed selections.

n represents the value used in the internal divide network to provide frequency multiplication. Processors therefore with the following cycle times have a corresponding division factor **n** of the period of **ClockIn** (200 ns) as follows:

ProcClockOut Period	n
44 ns	4.5
50 ns	4.0
57 ns	3.5

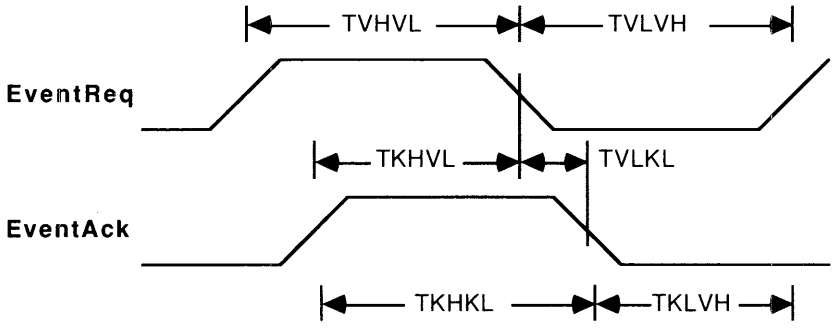
Parameters given in this section will be revised as a result of fuller characterization.

Parameter		Minimum	Maximum	Unit
TPCHPCH	**ProcClockOut** period	200/**n**	-	ns
TPCHPCL	**ProcClockOut** pulse width high	(**TPCHPCH**/2)-1	-	ns
TPCLPCH	**ProcClockOut** pulse width low	**TPCHPCH**-**TPCHPCL**	-	ns
TPCHEL	**ProcClockOut** rising edge to Chip Enable low	1	-	ns
TELPCH	Chip Enable low to end of first cycle	**TPCHPCH**-1	-	ns
TEHAX	Address hold time	**TPCHPCH**/2	-	ns
TAVEL	Address setup time	**TPCHPCH**/4	-	ns
TELEH	Chip Enable low time	5*__TPCHPCH__/4	-	ns
TEHEL	Chip Enable high time	3*__TPCHPCH__/4	-	ns
TWELEL	Write command setup time	3	-	
TEHWEH	Write command hold time	**TPCHPCH**/2	-	ns
TDVEH	Data setup time	**TPCHPCH**	-	ns
TEHDX	Data hold time	**TPCLPCH**	-	ns
TELDV	Chip Enable access time	-	(5*__TPCHPCH__/4)-23	ns *
TEHDZ	Read data hold time	0	-	ns
TWEHEL	Write command deassert	0	-	ns
TPCHWtV	**ProcClockOut** rising edge to Wait sampled	-	(3*__TPCHPCH__/4)-23	ns *
TWtHWtL	Wait hold time per wait state	10	-	ns
TPCHBAV	**ProcClockOut** rising edge to ByteAccess valid	-	(3*__TPCHPCH__/4)-23	ns *
TBAHBAL	ByteAccess hold time	10	-	ns
TMGHMGL	**MemGranted** pulse width high	**TPCHPCH**	-	ns

* : This parameter includes a constant 23ns delay, which consists of 16ns delay through the output pad and 7ns delay through the input pad.

8.8 Peripheral interfacing AC characteristics

Event signals waveforms



Parameter		Min	Max	Unit
TVHVL	**EventReq** pulse width high	2		processor cycles
TVLVH	**EventReq** pulse width low	2		processor cycles
TVLKL	Falling edge delay from **EventReq** to **EventAck**	0	2	processor cycles
TKHKL	**EventAck** pulse width high	2		processor cycles
TKHVL	Delay from **EventAck** to falling edge of **EventReq**	0		
TKLVH	Delay from falling edge of **EventAck** to next **EventReq**	0		

Full details of each signal are provided in the referenced section.

Analyse Signal used to investigate the state of a T212. The signal brings the processor, links and clocks to a halt within approx three timeslice periods (approx 3 milliseconds). **Reset** may then be applied, which will not destroy state information nor reinitialise the memory interface. After **Reset** has been taken low, **Analyse** may be taken low after which the processor will execute its bootstrap routine. (**2.3**, **3.4**)
Input

BootFromROM If this is held to **VCC**, then the boot program is taken from ROM. Otherwise the T212 awaits a bootstrap message from a link. (**3.3.1**)
Input

CapMinus The negative terminal of an internal power supply used for the internal clocks. This must be connected via a 1 microfarad capacitor to **CapPlus**. If a tantalum capacitor is used **CapMinus** should be connected to the negative terminal. (**8.6**)

CapPlus The positive terminal of an internal power supply used for the internal clocks. This must be connected via a 1 microfarad capacitor to **CapMinus**. If a tantalum capacitor is used **CapPlus** should be connected to the positive terminal.

ClockIn Input clock from which all internal clocks are generated. The nominal input clock frequency for all transputers, of whatever wordlength and speed, is 5 MHz. (**8.6**)
Input

DoNotWire These are output pins reserved for INMOS use. They should be left unconnected.
Output

Error The processor has detected an error and remains high until **Reset**. Errors result from arithmetic overflow, by array bounds violations or by divide by zero. The **Error** flag can also be set by an instruction, to allow other forms of software detected error. (**2.3**)
Output

EventAck The **EventReq** and **EventAck** are a pair of handshake signals for external events. External logic takes **EventReq** high when the logic wishes to communicate with a process in the T212. The rising edge of **EventReq** causes a process to be scheduled. The processor takes **EventAck** high when the process is scheduled. At any time after this point the external logic may take **EventReq** low, following which the processor will set **EventAck** low. (**6**, **8.8**)
Output

EventReq Request external event. See description of **EventAck** above.
Input

GND Power supply return and logic reference, 0 V. There are several **GND** pins to minimize inductance within the package.

HoldToGND These are input pins reserved for INMOS use. They should be held to ground either directly or through a resistor of less than 10K ohms.
Input

HoldToVCC This input pin is reserved for INMOS use. It should be held to the power supply.
Input

LinkIn0 - 3 Input pins of the standard links. A **LinkIn** pin receives a serial stream of bits including protocol bits and data bits. Link inputs must be connected to another Link output or tied to **GND**. The Link input must not be tied high or left floating.
Input

LinkOut0 - 3
Output pins of each of the standard links. A **LinkOut** pin may be left floating or may be connected to one (and only one) **LinkIn** pin. As long as the skew specification is met, the connection may be via buffers. (**4, 8.5**)
Output

LinkSpecial
LinkSpecial selects the non-standard speed to be either 20MBits/sec when high or 5MBits/sec when low. (**4.1.1**)
Input

Link0Special
Link 0 operates at the non-standard speed selected by **LinkSpecial** if this pin is wired high. (**4.1.1**)
Input

Link123Special
Links 1, 2 and 3 operate at the non-standard speed selected by **LinkSpecial** if this pin is wired high. (**4.1.1**)
Input

MemA0 - 15
16-bit wide address bus.
Output

MemD0 - 15
16-bit wide data bus.
Input/Output

MemBAcc
This pin when high causes the memory interface to perform byte access.
Input

MemReq
This input is used by external devices to access the memory interface. When **MemReq** is sampled high, and if there is no processor request outstanding, the address and data lines are taken high impedance and **MemGranted** is taken high.
Input

MemGranted
See description above of **MemReq**.
Output

MemWait
Wait input for the memory interface. If, at the time **MemWait** is sampled, the input is low, the interface cycle proceeds. Otherwise, the interface is held until the input is sampled and found to be low. (**5.4**)
Input

notMemCE
This active low pin is taken low when the address bus is ready for external of the memory cycle.
Output

notMemWrB0-B1
These are the lest significant and most significant bytes respectively. and are active low, write enable pins.
Output

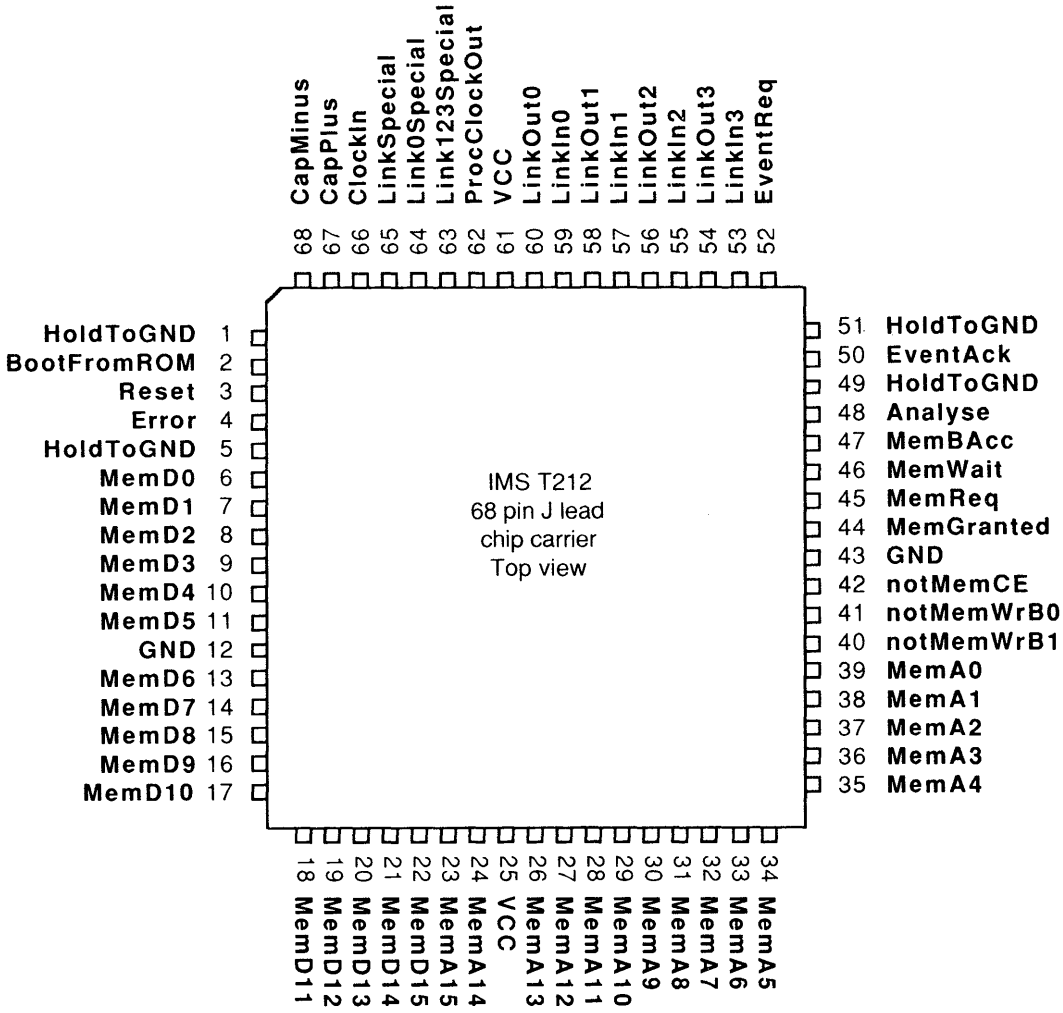
ProcClockOut
The processor clock which is output in phase with the memory interface. The processor clock frequency is a multiple of the input clock frequency. The multiple differs for different speed parts. (**7.1**)
Output

Reset
The falling edge of **Reset** initialises the T212 and then starts the processor executing a bootstrap routine.
Input

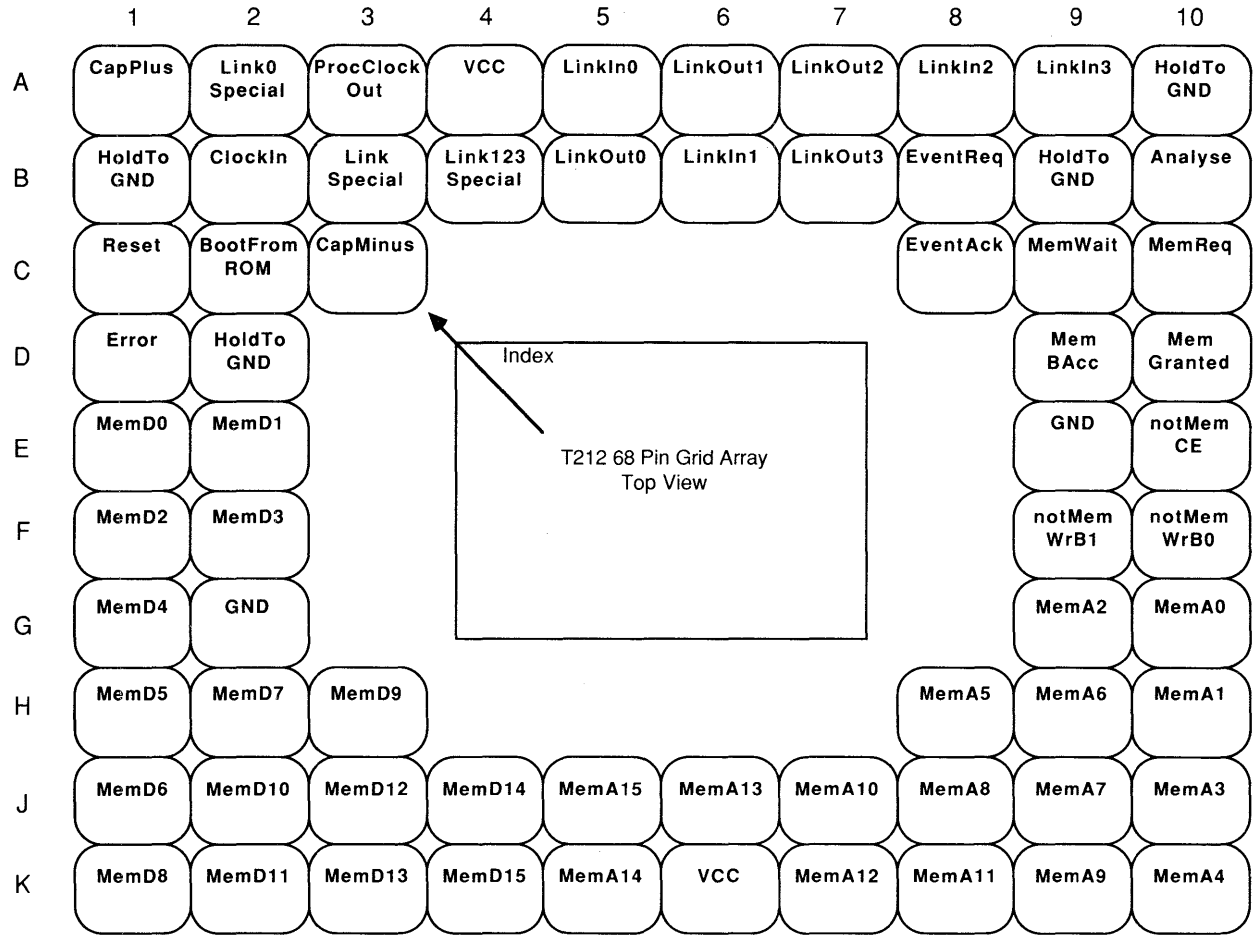
VCC
Power supply, nominally 5 V. There are several **VCC** pins to minimize inductance within the package. **VCC** should be decoupled to **GND** by at least one 100 nF low inductance (such as ceramic) capacitor.

The T212 is available in an 68 pin J-Lead chip carrier or 68 pin Grid Array.

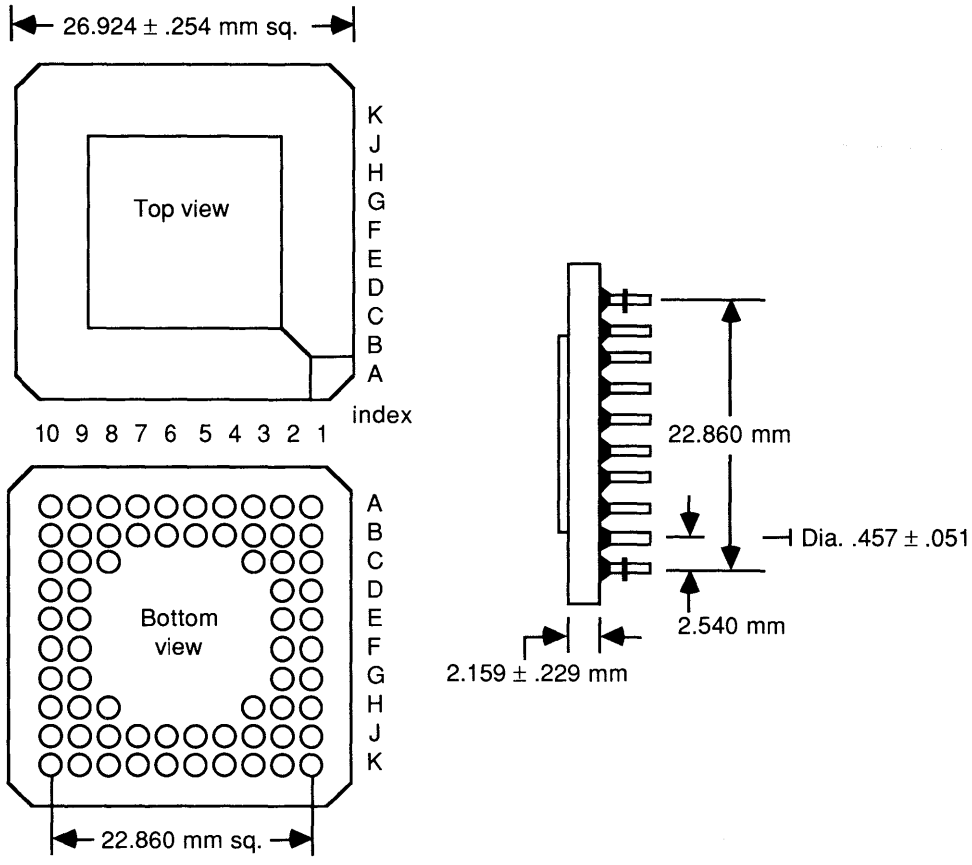
10.1 J-Lead chip carrier



10.2 Pin Grid Array



10.3 Package Dimensions



inmos®

Product data

IMS C011 link adaptor

The IMS C011 link adaptor is a universal high speed system interconnect, providing full duplex communication according to the INMOS serial link protocol. The link protocol provides synchronised message transmission using handshaken byte streams. Data reception is asynchronous, allowing communication to be independent of clock phase.

The IMS C011 converts the bidirectional serial link data into parallel data streams. It can be used to freely interconnect transputers, INMOS peripheral controllers, I/O subsystems, and microprocessors of different families.

The serial links can be operated at differing speeds; two C011 link adaptors can connect high and low speed links whilst maintaining the synchronised message transmission provided by the link protocol.

This manual details the product specific aspects of the IMS C011 and contains data relevant to the engineering and programming of the device.

Other information relevant to all transputer products is contained in the occam programming manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

This edition of the manual is dated October 27, 1986.

The IMS C011 link adaptor allows the user the flexibility of configuring it in one of two modes, and one of two serial link speeds for each mode.

Configuration depends upon the wiring of **SeparateIQ** pin (16) and is as follows:

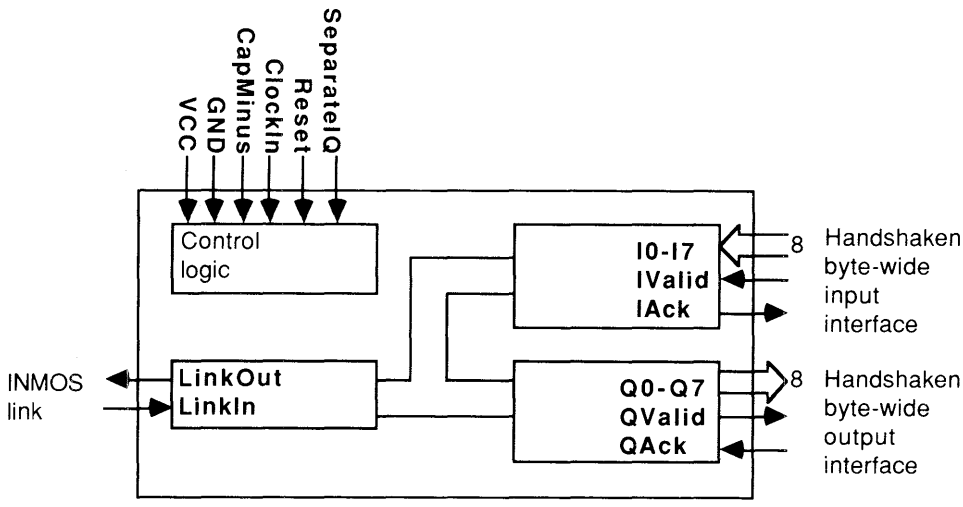
SeparateIQ pin	Mode	Speed of link at Clock=5MHz
Wire to **VCC**	Mode 1	10 Mbits/sec
Wire to **ClockIn**	Mode 1	20 Mbits/sec
Wire to **GND**	Mode 2	10 or 20 Mbits/sec

See Section 8 for pin descriptions and package diagrams.

For compatibility with INMOS products, the clock frequency should be 5MHz.

The following chapters describe the two different modes. Where there is no mention of mode, it may be assumed that the text applies to both modes.

Mode 1 Block Diagram



In mode 1 the link adaptor converts between an INMOS serial link and two independent fully handshaken byte-wide interfaces. One interface is for data coming from the serial link and one for data going to the serial link.

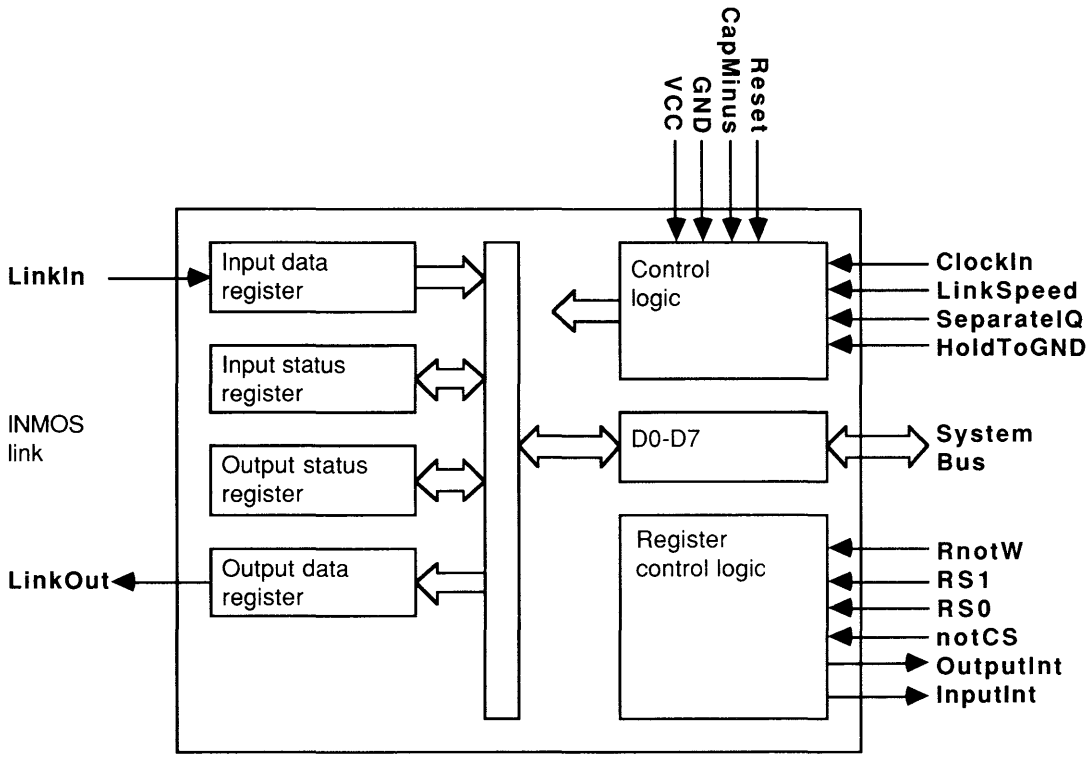
The serial link enables the link adaptor to communicate with another link adaptor, a transputer or an INMOS peripheral processor. Data reception is asynchronous which allows communication to be independent of clock phase. Transfers may proceed in both directions at the same time.

This mode provides programmable I/O pins for a transputer.

The serial links can be operated at differing speeds; when connected by their parallel ports two C011 link adaptors can connect high and low speed links whilst maintaining the synchronised message transmission provided by the link protocol.

The IMS C011 converts the bidirectional serial link data into parallel data streams. It can be used to fully interconnect transputers, INMOS peripheral controllers, I/O subsystems, and microprocessors of different families.

Mode 2 Block Diagram

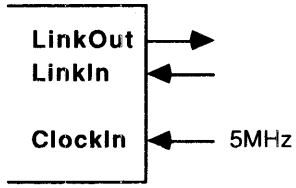


In mode 2 the link adaptor provides an interface between an INMOS serial link and a microprocessor system bus, via an 8-bit bi-directional interface.

This mode has status/control and data registers for both input and output. Any of these can be accessed by the byte wide interface at any time. Two interrupt lines are provided, each gated by an interrupt enable flag. One presents an interrupt on output ready, and the other on data present.

The IMS C011 converts the bidirectional serial link data into parallel data streams. It can be used to fully interconnect transputers, INMOS peripheral controllers, I/O subsystems, and microprocessors of different families.

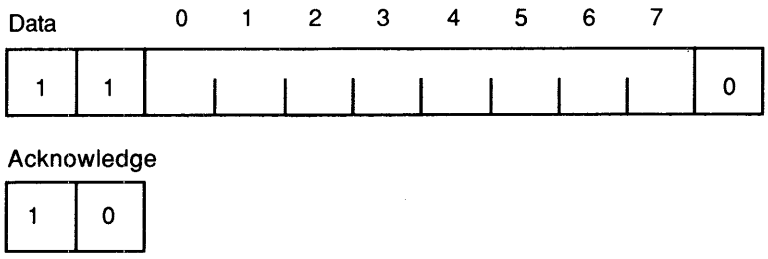
Standard Clock Input



The INMOS serial links are standard across all products in the transputer product range. All transputers will support a standard communications frequency of 10 Mbits/sec, regardless of processor performance. Thus transputers of different performance can be connected directly and future transputer systems will be able to communicate directly with those of today.

Each link consists of a serial input and a serial output, both of which are used to carry data and link control information.

Link protocol

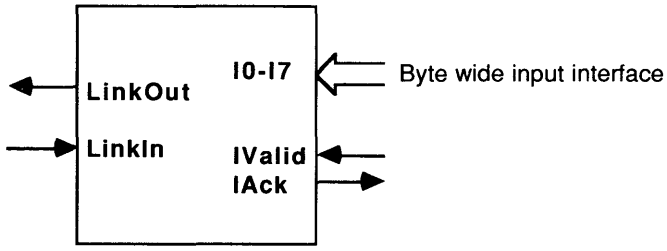


A message is transmitted as a sequence of bytes. After transmitting a data byte, the sender waits until an acknowledge has been received, signifying that the receiver is ready to receive another byte. The receiver can transmit an acknowledge as soon as it starts to receive a data byte, so that transmission can be continuous. This protocol provides handshaken communication of each byte of data, ensuring that slow and fast transputers communicate reliably.

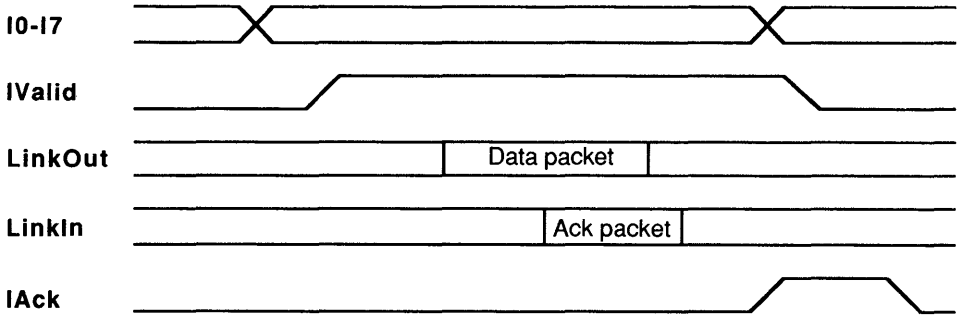
When there is no activity on the links they remain at logic 0, **GND** potential.

A 5 MHz input clock is used, from which internal timings are generated. Link communication is not sensitive to clock phase. Thus, communication can be achieved between independently clocked systems as long as the communications frequency is within the specified tolerance.

Output to link

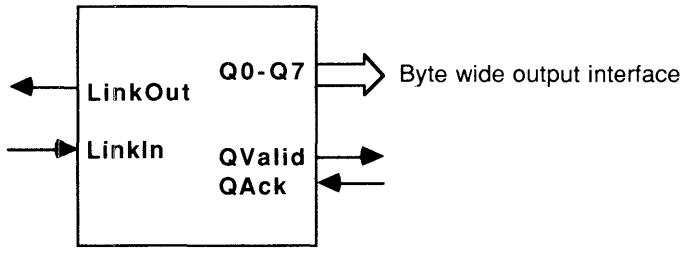


Timing of output to link

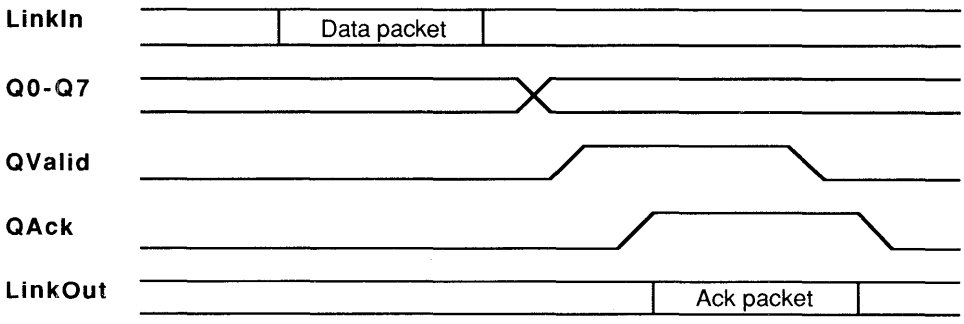


IValid and **IAck** provide a simple two wire handshake. Data is presented to the link adaptor on **I0-I7**, and **IValid** is taken high to commence the handshake. The link adaptor transmits the data through the serial link, and acknowledges receipt of the data on **IAck** when it has received the acknowledgment from the serial link. **IValid** is then taken low, and the link adaptor completes the handshake by taking **IAck** low.

Input from link



Timing of input from link



The link adaptor receives data from the serial link, presents it on **Q0-Q7**, and takes **QValid** high to commence the handshake. Receipt of the data is acknowledged on **QAck**, and the link adaptor then transmits an acknowledgement on the serial link. The link adaptor takes **QValid** low and the handshake is completed by taking **QAck** low.

In mode 2 the link adaptor is controlled at the parallel interface by reading and writing status/control registers, and by reading and writing data registers. Two interrupt lines are provided. One indicates that the link adaptor is ready to output a byte to the link, and the other indicates that it is holding a byte which it has read from the link.

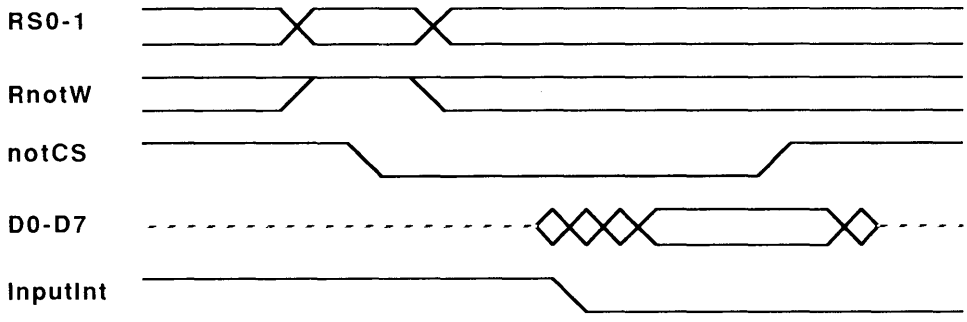
Parallel interface

One of the four registers is selected by **RS0** and **RS1**. If a new value is to be written into the selected registers, it is set up on **D0-D7** and **RnotW** is taken low. **notCS** is then taken low. On read cycles, the current value of the selected register is placed on **D0-D7**.

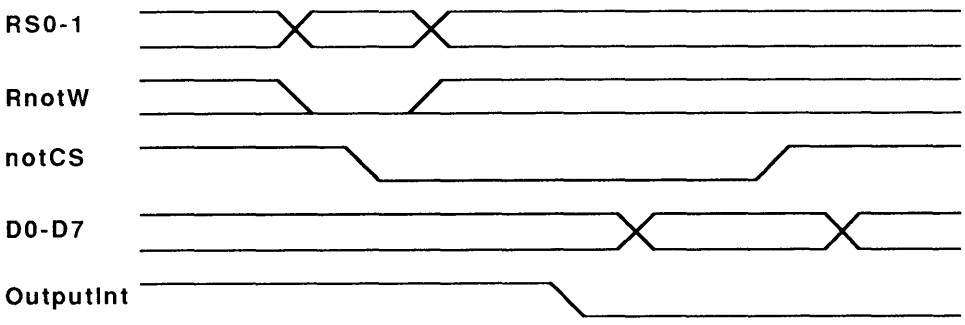
RS1	RS0	RnotW	Function
0	0	1	**D0-D7** := input data register
0	1	0	output data register := **D0-D7**
1	0	1	**D0-D7** := input status register
1	0	0	input status register := **D0-D7**
1	1	1	**D0-D7** := output status register
1	1	0	output status register := **D0-D7**

Note : Writing to the input data register has no effect and reading the output data register will result in undefined data on the data bus. Unused bit positions must be set to zero in both the input status register and the output status register.

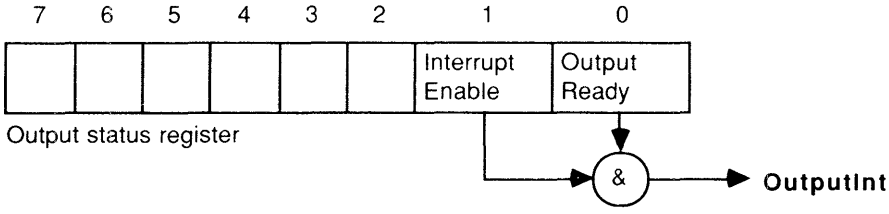
Read register timing diagram



Write register timing diagram

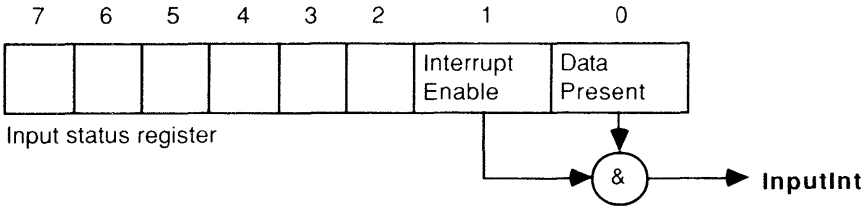


Output to link



The output ready status bit indicates that the serial link is ready to send a byte of data. The bit is set high on reset, and when the link adaptor receives an acknowledgement from the serial link. It is reset low when a data byte is written to the output data register on the parallel interface. **OutputInt** is set high if both output ready and interrupt enable are set high. Output interrupt enable is set low on reset.

Input from link



The data present status bit indicates that the serial link has received a byte of data. The bit is reset low when the data byte is read from the input data register on the parallel interface, this causes an acknowledgement to be transmitted on the serial link. **InputInt** is set high if both data present and interrupt enable are set high. Input interrupt enable and **data present** are both set low on reset.

Note: Parameters given in this section will be revised as a result of further characterization.

7.1 Absolute maximum ratings

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	0	7.0	V	1, 2, 3
VI,VO	Input or output voltage on any pin	-0.5	**VCC**+0.5	V	1, 2, 3
OSCT	Output short circuit time (one pin)		1	s	1
TS	Storage temperature	-65	150	°C	1
TA	Ambient temperature under bias	-55	125	°C	1
PD	Power dissipation rating		600	mW	

Notes

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

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

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

7.2 Recommended operating conditions

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	4.5	5.5	V	1
VI,VO	Input or output voltage	0	**VCC**	V	1,2
CL	Load capacitance on any pin		50	pF	
TA	Operating temperature range	0	70	°C	

Notes

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

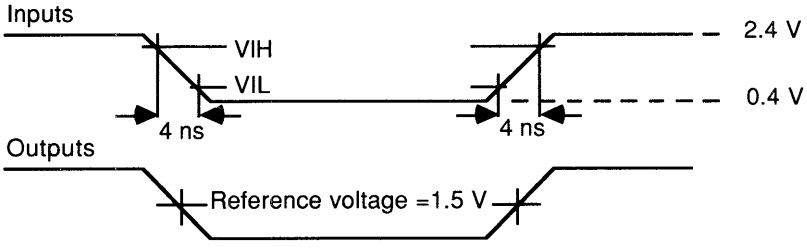
2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.

7.3 DC characteristics

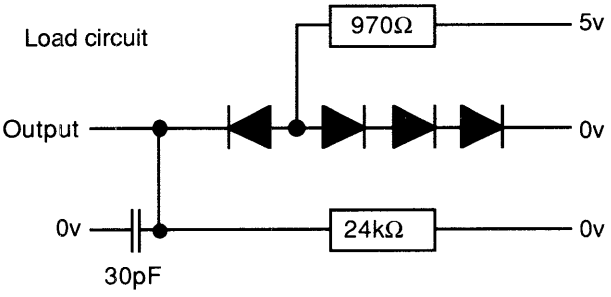
Parameter		Conditions	Min	Max	Unit
VIH	High level input voltage		2.0	**VCC**+0.5	V
VIL	Low level input voltage		-0.5	0.8	V
II	Input current	**GND** < **VI**< **VCC**		±200	μ A
VOH	Output high voltage	**IOH** = -2mA	**VCC**-1		V
VOL	Output low voltage	**IOL** = 4mA		0.4	V
IOS	Output short circuit current	**GND** < **VO** < **VCC**		50	mA
IOZ	Tristate output current	**GND** < **VI** <**VCC**		±200	μ A
PD	Power dissipation			100	mW
CIN	Input capacitance	f = 1 MHz		7	pF
COZ	Output capacitance in tristate	f = 1 MHz		10	pF

Note : 4.5 V < **VCC** < 5.5 V
0 °C < **TA** < 70 °C
Input clock frequency = 5 MHz
All voltages are with respect to **GND**

7.4 Measurement of AC characteristics



Reference points for AC characteristics



Load circuit for AC measurements

The load circuit approximates to two TTL loads, with a total capacitance of 30 pF.

7.5 Connection of INMOS serial links

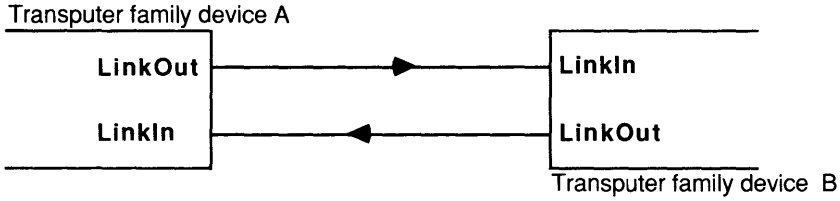
INMOS serial links can be connected in 3 different ways depending on their environment:

1 Directly connected

2 Connected via a series matching resistor

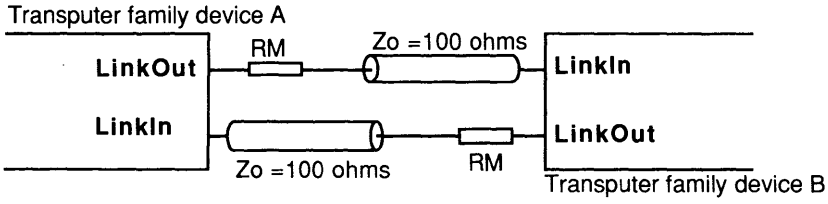
3 Connected via buffers

Direct connection



Direct connection is suitable for short distances on a printed circuit board.

Matched line

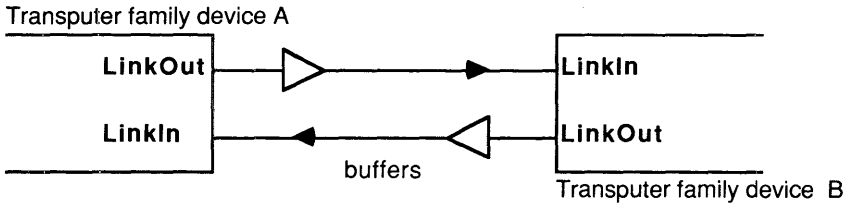


For long wires, approximately >30 cm, then a 100ohm transmission line should be used with series matching resistors.

Parameter		Nom	Max	Unit
RM	Series matching resistor for 100 ohm line.	47		ohm
TD	Delay down line		0.4	bit time

Note that if two connected devices have different values for **TD**, the lower value should be used. With series termination at **LinkOut** the transmission line reflection must return within 1 Bit time. Otherwise line buffers should be used.

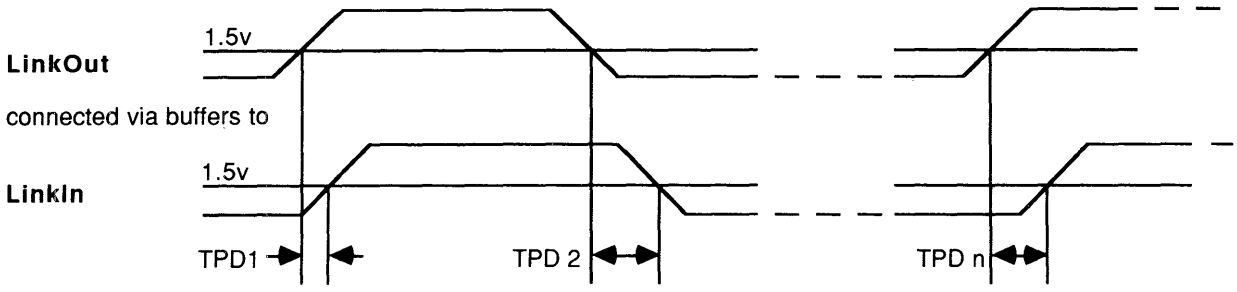
Buffered links



If buffers are used their overall propagation delay, TPD, should be stable within the skew tolerance.

Parameter	Max	Unit
Skew in buffering at 5 Mbits/sec	30	ns
Skew in buffering at 10 Mbits/sec	10	ns
Skew in buffering at 20 Mbits/sec		ns
Rise and fall time of **LinkIn** (10% to 90%)	20	ns

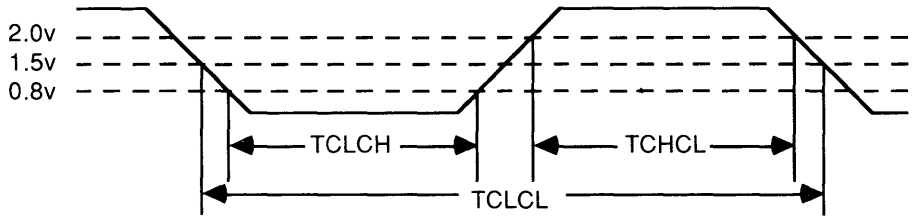
The above figures indicate that buffered links can be realised at 10Mbits/sec. For the case of 20 Mbits/sec, the maximum value can only be specified as a result of further characterisation.



The absolute value of TPD is immaterial because data reception is asynchronous. However, TPD will vary from moment to moment because of ground noise, variation in the power supplies of buffers and the difference in the delay for rising and falling edges. This will vary the length of data bits reaching **LinkIn**. *Skew* is the difference between the maximum and minimum instantaneous values of TPD.

7.6 AC characteristics of system services

ClockIn waveform



ClockIn Waveform diagram

Parameter		Min	Nom	Max	Unit	Note
TCHCL	Clock pulse width high	40			ns	1
TCLCH	Clock pulse width low	40			ns	1
	Rise and fall time of **ClockIn** (10% to 90%)			10	ns	1
TCLCL	Clock period		200	400	ns	2
	Difference in frequencies of **ClockIn** for two devices connected by a link			400	ppm	4
TRHRL	**Reset** pulse width high	8			**ClockIn** periods	3
	Time **VCC** and **ClockIn** valid before reset is taken low		10		ms	3,5

Notes

1 The clock transitions must be monotonic within the range between **VIH** and **VIL**.

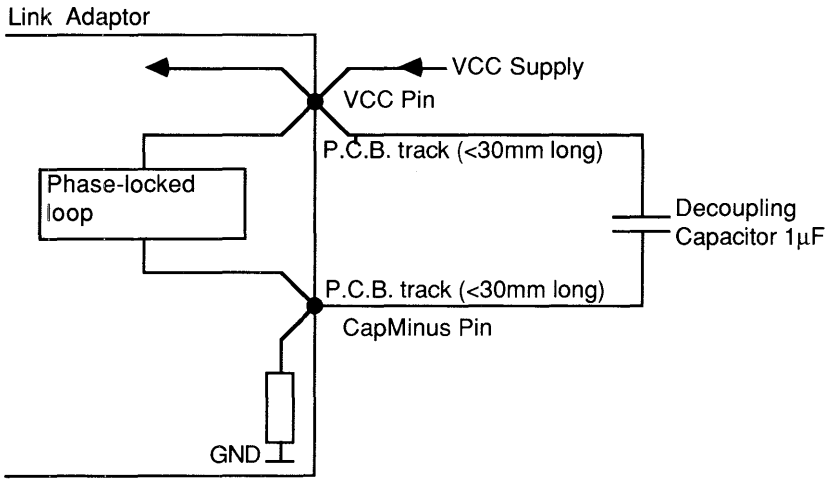
2 The **TCLCL** parameter is measured between corresponding points on consecutive falling edges.

3 During reset, **LinkIn** should be held low. Note that reset forces **LinkOut** to be low.

4 This value allows the use of low cost 200ppm crystal oscillators.

5 When powering up.

Recommended PLL decoupling



Notes

Parameter		Min	Nom	Max	Unit	Note
C	Decoupling capacitor	1			µF	1
	A.C. noise between **VCC** and **GND**			200	mV	2,3
	A.C. noise between **VCC** and ground reference of load capacitance			200	mV	4

1 The decoupling capacitor should be a high-quality tantalum (or other) type, with low series resistance (<1 ohm), and low impedance at high frequency (<10 ohm at 100 MHz). It should be connected between **VCC** and **CapMinus**, with short tracks, and with a star point at the **VCC** pin, as shown.

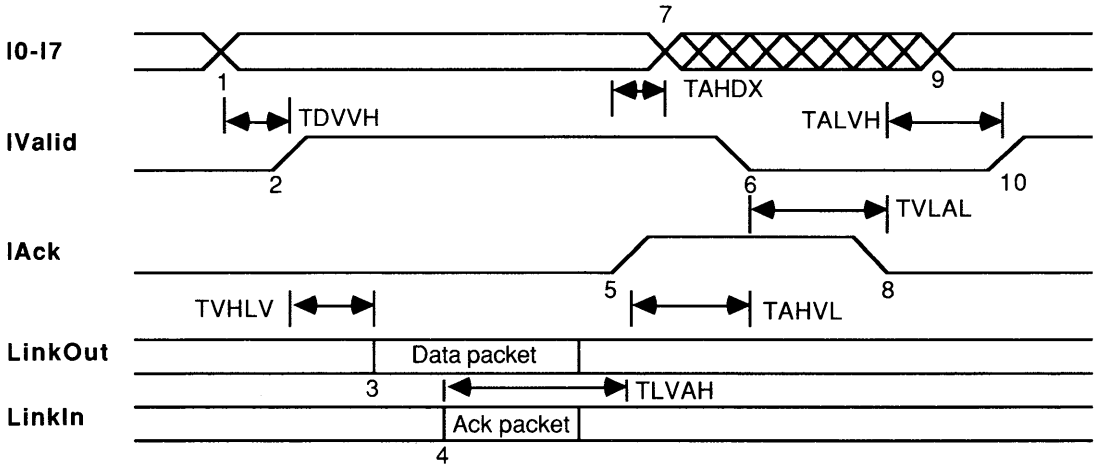
2 Requires a 0.1µF. capacitor between **VCC** and **GND**, close to the chip.

3 Peak to peak at all frequencies above 100 KHz.

4 Peak to peak at all frequencies above 30 MHz.

7.7 AC characteristics of parallel interface

7.7.1 Mode 1 Output to link



Event details

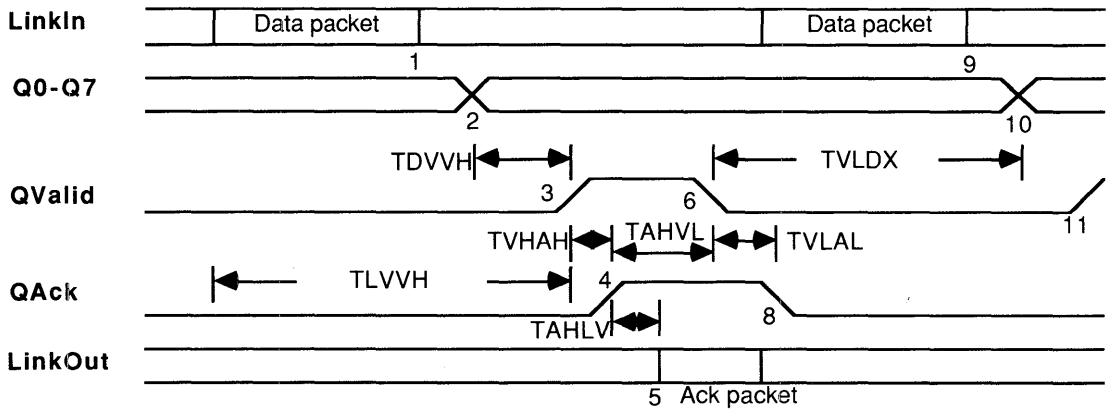
1 Data is presented on **I0-7**
2 The presence of valid data is indicated by raising **IValid**
3 The C011 commences transmitting the data on **LinkOut**
4 The C011 receives an acknowledge packet on **LinkIn**
5 The C011 acknowledges the data by taking **IAck** high
6 **IValid** is taken low
7 Data is removed from **I0-7**
8 The C011 completes the handshake by taking **IAck** low
9 Another data byte is presented on **I0-7**
10 The presence of valid data is indicated by raising **IValid**

Dependencies

Ensure that	1 precedes 2	by **TDVVH**
	5 precedes 6	by **TAHVL**
	5 precedes 7	by **TAHDX**
	8 precedes 10	by **TALVH**
The C011 ensures that	2 precedes 3	by **TVHLV**
	4 precedes 5	by **TLVAH**
	6 precedes 8	by **TVLAL**
The link protocol ensures that	3 precedes 4	

Parameter		Min	Max	Unit
TDVVH	Data setup before **IValid** high	5		ns
TVHLV	**IValid** high to byte on link			
	(no ack on link)	0.8	1.8	bit time
	(ack on link)	0.8	3.8	bit time
TLVAH	ack to **IAck** high		3	bit time
TAHVL	**IAck** high to **IValid** low	0		ns
TVLAL	**IValid** low to **IAck** low	1	4	bit time
TALVH	**IAck** low to **IValid** high	0		ns
TAHDX	Data hold from **IAck** high	0		ns

7.7.2 Mode 1 input from link



Event details

1 The C011 receives on **LinkIn** the data packet to be output
2 The C011 places the data on **Q0-7**
3 The C011 indicates the presence of valid data by raising **QValid**
4 The data is acknowledged by taking **QAck** high
5 The C011 transmits an acknowledge packet on **LinkOut**
6 The C011 takes **QValid** low
7 The C011 no longer holds the data outputs **Q0-7** valid
8 The handshake is completed by taking **QAck** low
9 The C011 receives on **LinkIn** another data packet
10 The C011 places the data on **Q0-7**
11 The C011 indicates the presence of valid data by raising **QValid**

Dependencies

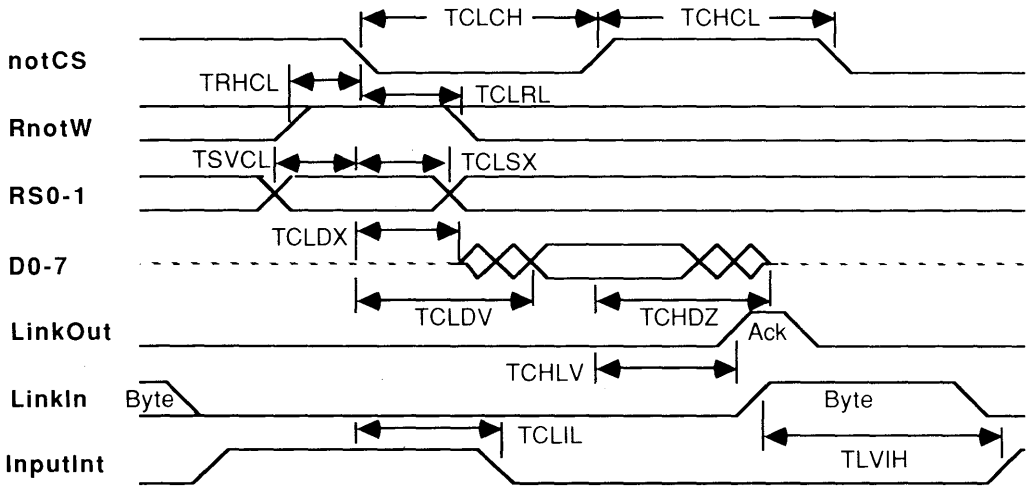
Ensure that	3 precedes 4	by **TVHAH**
	6 precedes 8	by **TVLAL**
The C011 ensures that	1 precedes 2	
	2 precedes 3	by **TDVVH**
	4 precedes 5	by **TAHLV**
	4 precedes 6	by **TAHVL**
	6 precedes 10	by **TVLDX**
	8 precedes 11	

Parameter		Min	Max	Unit
TDVVH	Data setup before **QValid** high	15		ns
TVLDX	Data hold after **QValid** low	11		bit time
TLVVH	Byte to **QValid** high	12		bit time
TVHAH	**QAck** setup time	0		ns
TVLAL	**QAck** hold time	0		ns
TAHLV	**QAck** high to Ack on link			
	(no byte on link)	0.8	1.8	bit time
	(byte on link)	0.8	12.8	bit time
TAHVL	**QAck** high to **QValid** low	1.8		bit time

7.7.3 Mode 2

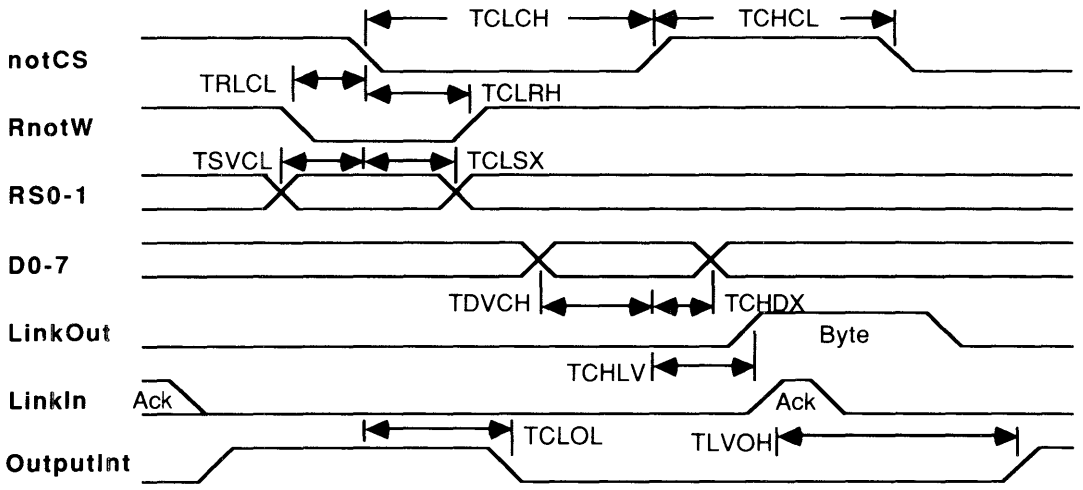
AC characteristics of parallel interface

Read cycle



Parameter		Min	Max	Unit
TCLCH	**notCS** low time	50		ns
TCHCL	time between cycles	50		ns
TRHCL	**RnotW** setup	5		ns
TCLRL	**RnotW** hold	5		ns
TSVCL	Register select setup	5		ns
TCLSX	Register select hold	5		ns
TCLDX	**notCS** to output active	5		ns
TCLDV	**notCS** to output valid		40	ns
TCLDZ	**notCS** high to output invalid	0	25	ns
TCHLV	**notCS** high to Ack on link			
	(no byte on link)	0.8	1.8	bit time
	(byte on link)	0.8	12.8	bit time
TCLIL	**notCS** low to **InputInt** low		25	ns
TLVIL	Byte to **InputInt** high		12	bit time

Write cycle



Parameter		Min	Max	Unit
TCLCH	**notCS** low time	50		ns
TCHCL	time between cycles	50		ns
TRLCL	**RnotW** setup	5		ns
TCLRH	**RnotW** hold	5		ns
TSVCL	Register select setup	5		ns
TCLSX	Register select hold	5		ns
TDVCH	Data setup	15		ns
TCHDX	Data hold	5		ns
TCHLV	**notCS** high to byte on link			
	(no Ack on link)	0.8	1.8	bit time
	(Ack on link)	0.8	3.8	bit time
TCLOL	**notCS** low to **OutputInt** low		25	ns
TLVOH	Ack to **OutputInt** high		3	bit time

8.1 Mode 1

CapMinus Negative end of decoupling capacitor.

ClockIn 5MHz input clock.
Input

GND Ground.

I0-I7 Byte-wide input data.
Input

IAck Link adaptor acknowledges that input data has been received.
Output

IValid Input data on **I0-I7** is valid.
Input

LinkIn INMOS serial link input.
Input

LinkOut INMOS serial link output.
Output

Q0-Q7 Byte-wide output data.
Output

QAck Acknowledge to link adaptor that data output from the link adaptor has been received.
Input

QValid Output data on **Q0-Q7** is valid.
Output

Reset Reset link adaptor.
Input

SeparateIQ This pin selects operating modes 1 or 2. In addition to this, it controls the link speed in mode 1 operation. If wired to **VCC**, mode 1 is selected with a link speed of 10 Mbits/sec. If wired to **ClockIn**, mode 1 is selected but with a link speed of 20 Mbits/sec. Wiring to **GND** selects mode 2.
Input

VCC Star point for +5 volt supply input and positive end of decoupling capacitor.

8.2 Mode 2

CapMinus Negative end of decoupling capacitor.

ClockIn 5MHz input clock.
Input

D0-D7 Bi-directional databus.
Input/Output

DoNotWire Signal reserved for INMOS use. Must be left unconnected.

GND Ground.

HoldToGND Signal reserved for INMOS use. Must be held to **GND**. Failure to do so may cause damage to the device.
Input

InputInt Input interrupt.
Output

LinkIn INMOS serial link input.
Input

LinkOut INMOS serial link output.
Output

LinkSpeed Determines the speed of the links. If wired to **GND** link speed is 10 Mbits/sec, if wired to **VCC** speed is 20 Mbits/sec.
Input

notCS Chip select input.
Input

OutputIn Output interrupt.
Output

Reset Reset link adaptor.
Input

RnotW Read or write register.
Input

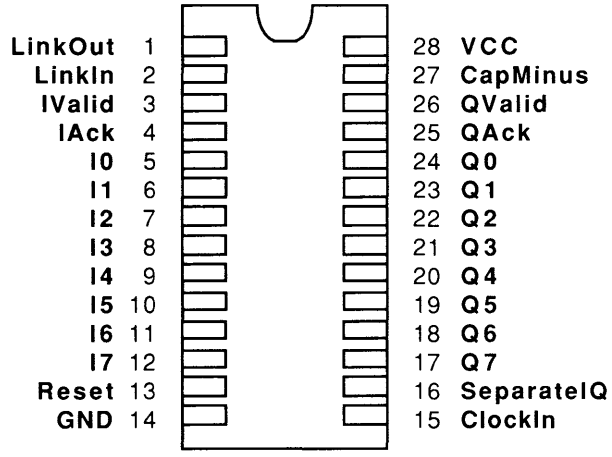
RS0-RS1 Register select lines; used in conjunction with **notCS** and **RnotW** to control the selection of the internal register to be read or written.
Input

SeparateIQ This must be wired to **GND** to select mode 2.
Input

VCC Star point for +5 volt supply input and positive end of decoupling capacitor.

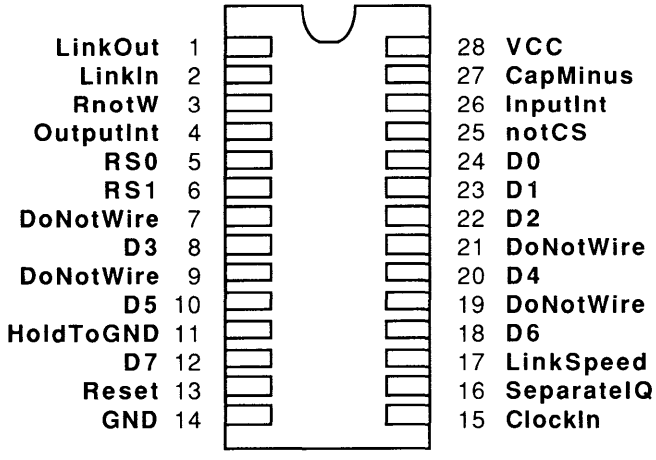
The C011 is available in a 28 pin plastic package.

Package mode 1



C011 Mode 1 pin out

Package mode 2



C011 Mode 2 pin out

inmos®

Product data

IMS C012 link adaptor

The IMS C012 link adaptor is a universal high speed system interconnect, providing full duplex communication according to the INMOS serial link protocol. The link protocol provides synchronised message transmission using handshaken byte streams. Data reception is asynchronous, allowing communication to be independent of clock phase.

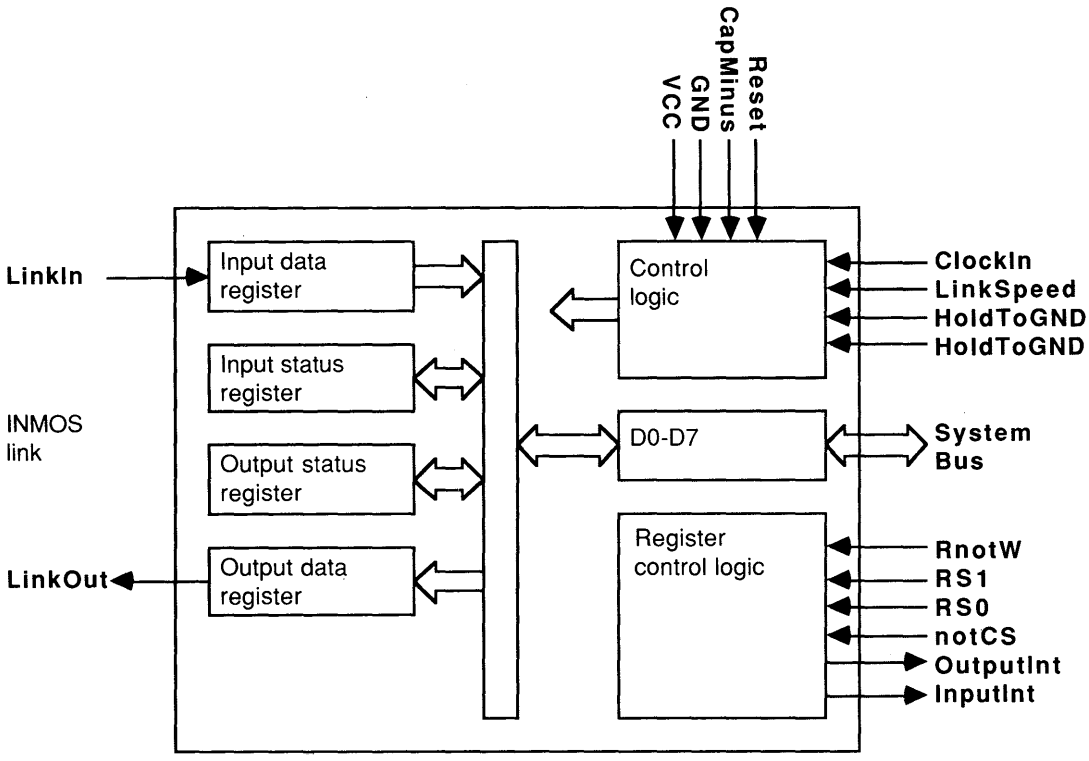
The IMS C012 converts the bidirectional serial link data into parallel data streams. It can be used to freely interconnect transputers, INMOS peripheral controllers, I/O subsystems, and microprocessors of different families.

This manual details the product specific aspects of the IMS C012 and contains data relevant to the engineering and programming of the device.

Other information relevant to all transputer products is contained in the occam programming manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

This edition of the manual is dated October 27, 1986.

C012 Block Diagram

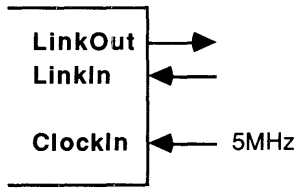


The C012 link adaptor provides an interface between an INMOS serial link and a microprocessor system bus, via an 8-bit bi-directional interface.

The device provides status/control and data registers for both input and output. Any of these can be accessed by the byte wide interface at any time. Two interrupt lines are provided, each gated by an interrupt enable flag. One presents an interrupt on output ready, and the other on data present.

The IMS C012 converts the bidirectional serial link data into parallel data streams. It can be used to fully interconnect transputers, INMOS peripheral controllers, I/O subsystems, and microprocessors of different families.

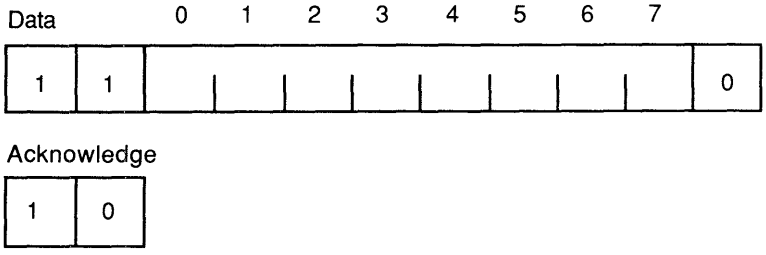
Standard Clock Input



The INMOS serial links are standard across all products in the transputer product range. All transputers will support a standard communications frequency of 10 Mbits/sec, regardless of processor performance. Thus transputers of different performance can be connected directly and future transputer systems will be able to communicate directly with those of today.

Each link consists of a serial input and a serial output, both of which are used to carry data and link control information.

Link protocol



A message is transmitted as a sequence of bytes. After transmitting a data byte, the sender waits until an acknowledge has been received, signifying that the receiver is ready to receive another byte. The receiver can transmit an acknowledge as soon as it starts to receive a data byte, so that transmission can be continuous. This protocol provides handshaken communication of each byte of data, ensuring that slow and fast transputers communicate reliably.

When there is no activity on the links they remain at logic 0, **GND** potential.

A 5 MHz input clock is used, from which internal timings are generated. Link communication is not sensitive to clock phase. Thus, communication can be achieved between independently clocked systems as long as the communications frequency is within the specified tolerance.

The IMS C012 link adaptor is controlled at the parallel interface by reading and writing status/control registers, and by reading and writing data registers. Two interrupt lines are provided. One indicates that the link adaptor is ready to output a byte to the link, and the other indicates that it is holding a byte which it has read from the link.

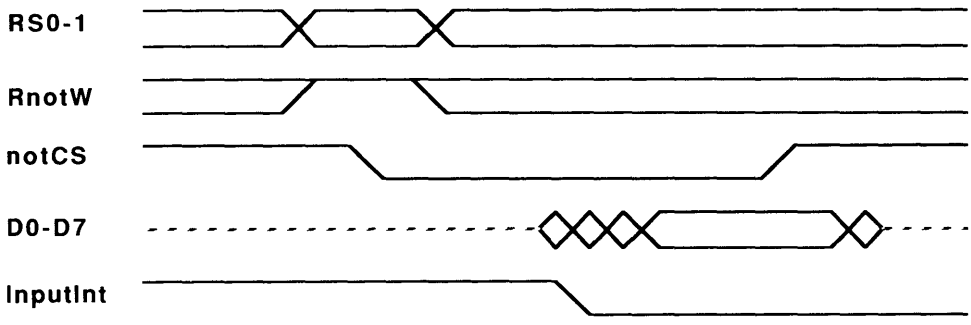
Parallel interface

One of the four registers is selected by **RS0** and **RS1**. If a new value is to be written into the selected registers, it is set up on **D0-D7** and **RnotW** is taken low. **notCS** is then taken low. On read cycles, the current value of the selected register is placed on **D0-D7**.

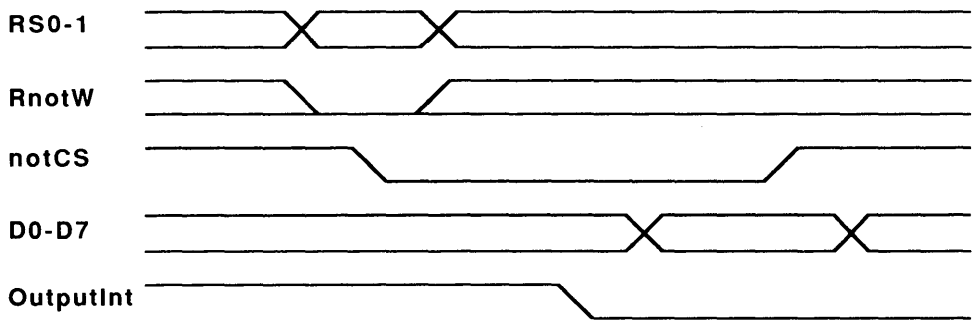
RS1	RS0	RnotW	Function
0	0	1	**D0-D7** := input data register
0	1	0	output data register := **D0-D7**
1	0	1	**D0-D7** := input status register
1	0	0	input status register := **D0-D7**
1	1	1	**D0-D7** := output status register
1	1	0	output status register := **D0-D7**

Note : Writing to the input data register has no effect and reading the output data register will result in undefined data on the data bus. Unused bit positions must be set to zero in both the input status register and the output status register.

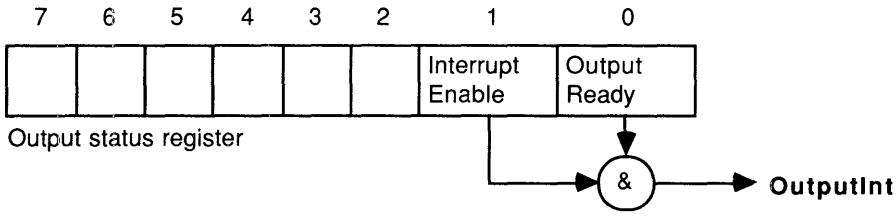
Read register timing diagram



Write register timing diagram

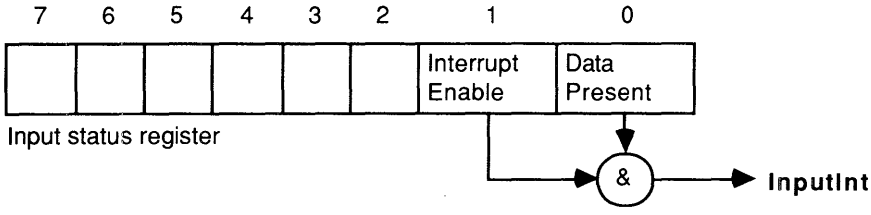


Output to link



The output ready status bit indicates that the serial link is ready to send a byte of data. The bit is set high on reset, and when the link adaptor receives an acknowledgement from the serial link. It is reset low when a data byte is written to the output data register on the parallel interface. **OutputInt** is set high if both output ready and interrupt enable are set high. Output interrupt enable is set low on reset.

Input from link



The data present status bit indicates that the serial link has received a byte of data. The bit is reset low when the data byte is read from the input data register on the parallel interface, this causes an acknowledgement to be transmitted on the serial link. **InputInt** is set high if both data present and interrupt enable are set high. Input interrupt enable and **data present** are both set low on reset.

Note: Parameters given in this section will be revised as a result of further characterization.

4.1 Absolute maximum ratings

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	0	7.0	V	1, 2, 3
VI,VO	Input or output voltage on any pin	-0.5	**VCC**+0.5	V	1, 2, 3
OSCT	Output short circuit time (one pin)		1	s	1
TS	Storage temperature	-65	150	°C	1
TA	Ambient temperature under bias	-55	125	°C	1
PD	Power dissipation rating		600	mW	

Notes

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

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

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

4.2 Recommended operating conditions

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	4.5	5.5	V	1
VI,VO	Input or output voltage	0	**VCC**	V	1,2
CL	Load capacitance on any pin		50	pF	
TA	Operating temperature range	0	70	°C	

Notes

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

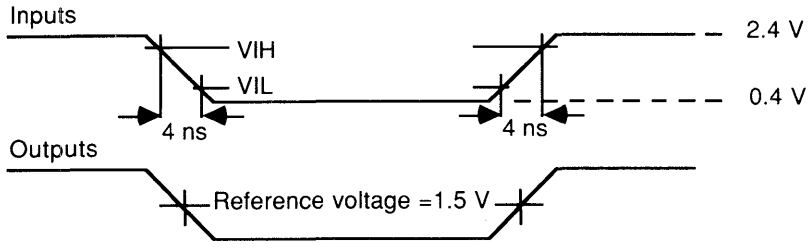
2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.

4.3 DC characteristics

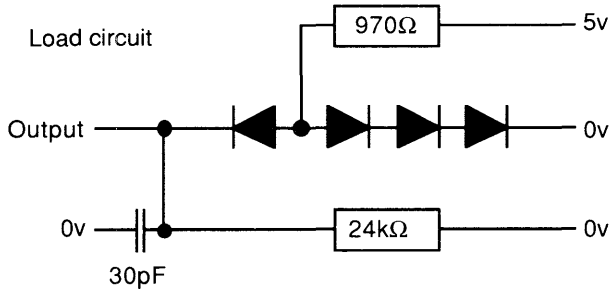
Parameter		Conditions	Min	Max	Unit
VIH	High level input voltage		2.0	**VCC**+0.5	V
VIL	Low level input voltage		-0.5	0.8	V
II	Input current	**GND** < **VI**< **VCC**		±200	μ A
VOH	Output high voltage	**IOH** = -2mA	**VCC**-1		V
VOL	Output low voltage	**IOL** = 4mA		0.4	V
IOS	Output short circuit current	**GND** < **VO** < **VCC**		50	mA
IOZ	Tristate output current	**GND** < **VI** <**VCC**		±200	μ A
PD	Power dissipation			100	mW
CIN	Input capacitance	f = 1 MHz		7	pF
COZ	Output capacitance in tristate	f = 1 MHz		10	pF

Note : 4.5 V < **VCC** < 5.5 V
0 °*C* < **TA** < 70 °*C*
Input clock frequency = 5 MHz
All voltages are with respect to **GND**

4.4 Measurement of AC characteristics



Reference points for AC characteristics



Load circuit for AC measurements

The load circuit approximates to two TTL loads, with a total capacitance of 30 pF.

4.5 Connection of INMOS serial links

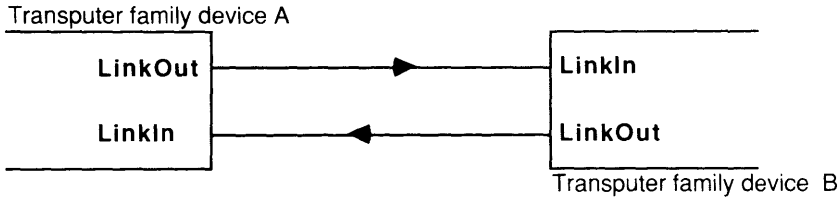
INMOS serial links can be connected in 3 different ways depending on their environment:

1 Directly connected

2 Connected via a series matching resistor

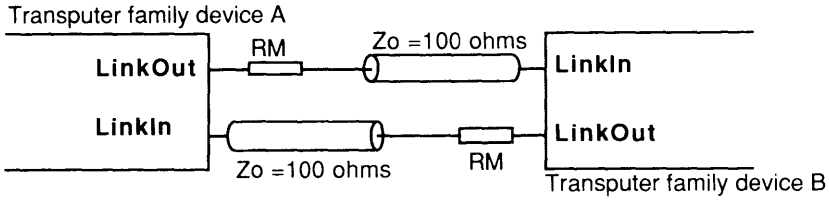
3 Connected via buffers

Direct connection



Direct connection is suitable for short distances on a printed circuit board.

Matched line

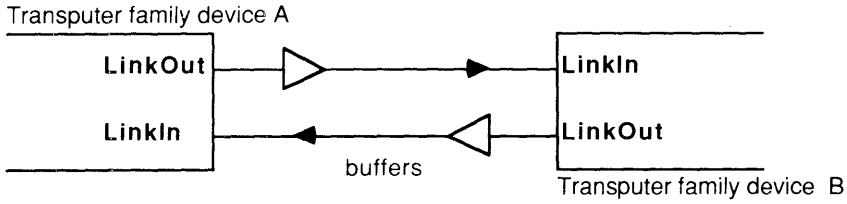


For long wires, approximately >30 cm, then a 100ohm transmission line should be used with series matching resistors.

Parameter		Nom	Max	Unit
RM	Series matching resistor for 100 ohm line.	47		ohm
TD	Delay down line		0.4	bit time

Note that if two connected devices have different values for **TD**, the lower value should be used. With series termination at **LinkOut** the transmission line reflection must return within 1 Bit time. Otherwise line buffers should be used.

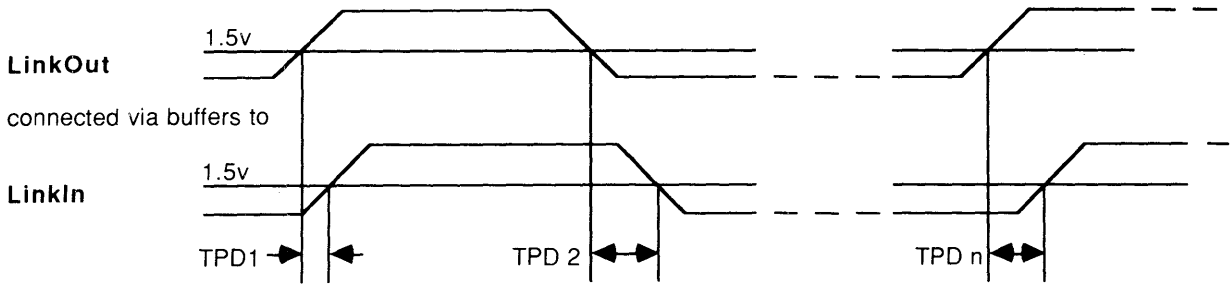
Buffered links



If buffers are used their overall propagation delay, TPD, should be stable within the skew tolerance.

Parameter	Max	Unit
Skew in buffering at 5 Mbits/sec	30	ns
Skew in buffering at 10 Mbits/sec	10	ns
Skew in buffering at 20 Mbits/sec		ns
Rise and fall time of **LinkIn** (10% to 90%)	20	ns

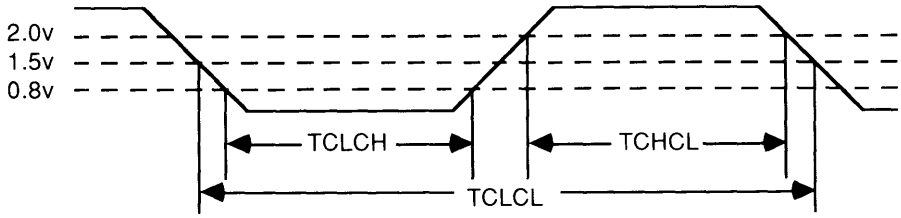
The above figures indicate that buffered links can be realised at 10Mbits/sec. For the case of 20 Mbits/sec, the maximum value can only be specified as a result of further characterisation.



The absolute value of TPD is immaterial because data reception is asynchronous. However, TPD will vary from moment to moment because of ground noise, variation in the power supplies of buffers and the difference in the delay for rising and falling edges. This will vary the length of data bits reaching **LinkIn**. *Skew* is the difference between the maximum and minimum instantaneous values of TPD.

4.6 AC characteristics of system services

ClockIn waveform



ClockIn Waveform diagram

Parameter		Min	Nom	Max	Unit	Note
TCHCL	Clock pulse width high	40			ns	1
TCLCH	Clock pulse width low	40			ns	1
	Rise and fall time of **ClockIn** (10% to 90%)			10	ns	1
TCLCL	Clock period		200	400	ns	2
	Difference in frequencies of **ClockIn** for two devices connected by a link			400	ppm	4
TRHRL	**Reset** pulse width high	8			**ClockIn** periods	3
	Time **VCC** and **ClockIn** valid before reset is taken low		10		ms	3,5

Notes

1 The clock transitions must be monotonic within the range between **VIH** and **VIL**.

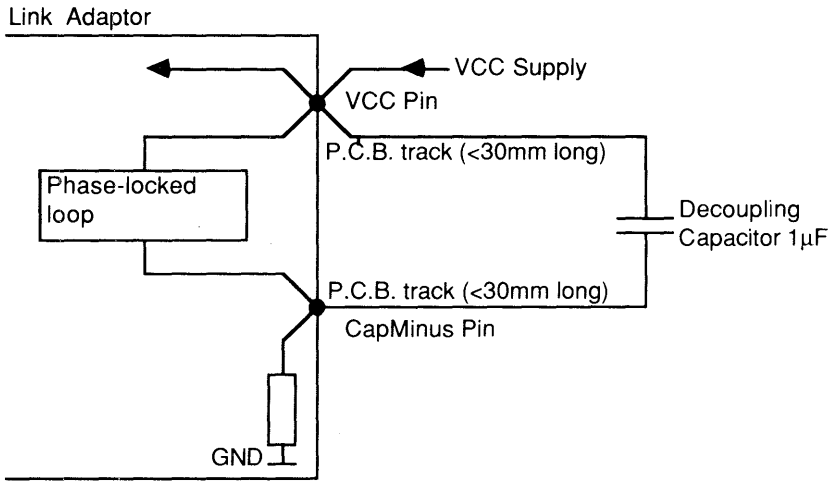
2 The **TCLCL** parameter is measured between corresponding points on consecutive falling edges.

3 During reset, **LinkIn** should be held low. Note that reset forces **LinkOut** to be low.

4 This value allows the use of low cost 200ppm crystal oscillators.

5 When powering up.

Recommended PLL decoupling



Notes

Parameter		Min	Nom	Max	Unit	Note
C	Decoupling capacitor	1			μF	1
	A.C. noise between **VCC** and **GND**			200	mV	2,3
	A.C. noise between **VCC** and ground reference of load capacitance			200	mV	4

1 The decoupling capacitor should be a high-quality tantalum (or other) type, with low series resistance (<1 ohm), and low impedance at high frequency (<10 ohm at 100 MHz). It should be connected between **VCC** and **CapMinus**, with short tracks, and with a star point at the **VCC** pin, as shown.

2 Requires a 0.1μF. capacitor between **VCC** and **GND**, close to the chip.

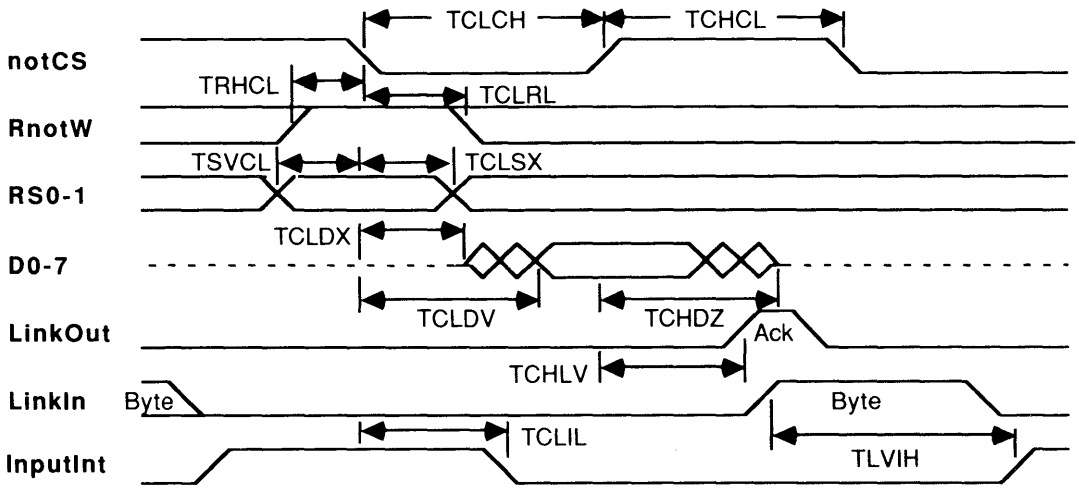
3 Peak to peak at all frequencies above 100 KHz.

4 Peak to peak at all frequencies above 30 MHz.

4.7 AC characteristics of parallel interface

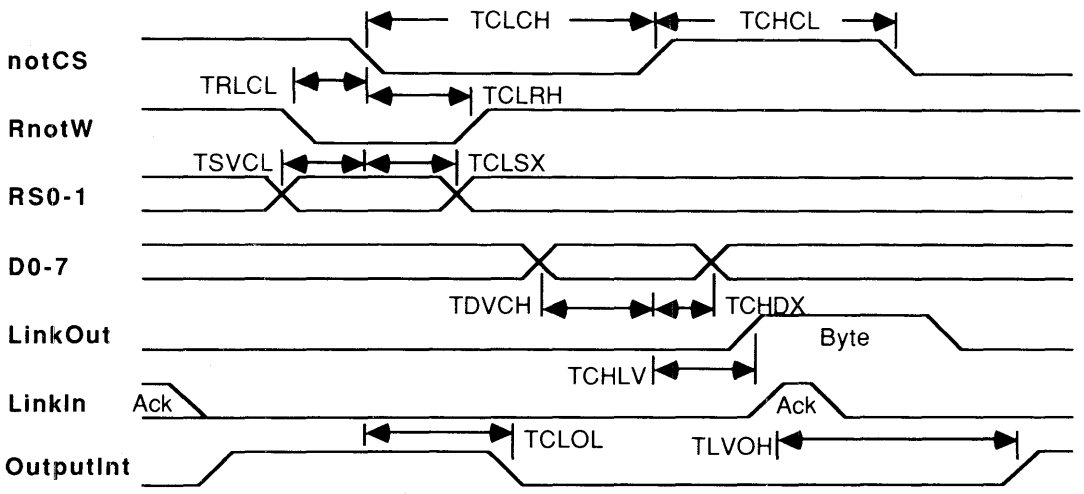
AC characteristics of parallel interface

Read cycle



Parameter		Min	Max	Unit
TCLCH	**notCS** low time	50		ns
TCHCL	time between cycles	50		ns
TRHCL	**RnotW** setup	5		ns
TCLRL	**RnotW** hold	5		ns
TSVCL	Register select setup	5		ns
TCLSX	Register select hold	5		ns
TCLDX	**notCS** to output active	5		ns
TCLDV	**notCS** to output valid		40	ns
TCLDZ	**notCS** high to output invalid	0	25	ns
TCHLV	**notCS** high to Ack on link			
	(no byte on link)	0.8	1.8	bit time
	(byte on link)	0.8	12.8	bit time
TCLIL	**notCS** low to **InputInt** low		25	ns
TLVIL	Byte to **InputInt** high		12	bit time

Write cycle



Parameter		Min	Max	Unit
TCLCH	**notCS** low time	50		ns
TCHCL	time between cycles	50		ns
TRLCL	**RnotW** setup	5		ns
TCLRH	**RnotW** hold	5		ns
TSVCL	Register select setup	5		ns
TCLSX	Register select hold	5		ns
TDVCH	Data setup	15		ns
TCHDX	Data hold	5		ns
TCHLV	**notCS** high to byte on link			
	(no Ack on link)	0.8	1.8	bit time
	(Ack on link)	0.8	3.8	bit time
TCLOL	**notCS** low to **OutputInt** low		25	ns
TLVOH	Ack to **OutputInt** high		3	bit time

CapMinus Negative end of decoupling capacitor.

ClockIn 5MHz input clock.
Input

D0-D7 Bi-directional databus.
Input/Output

GND Ground.

HoldToGND Signal reserved for INMOS use. Must be held to **GND**. Failure to do so may cause damage to the device.
Input

InputInt Input interrupt.
Output

LinkIn INMOS serial link input.
Input

LinkOut INMOS serial link output.
Output

LinkSpeed Determines the speed of the links. If wired to **GND** link speed is 10 Mbits/sec, if wired to **VCC** speed is 20 Mbits/sec.
Input

notCS Chip select input.
Input

OutputIn Output interrupt.
Output

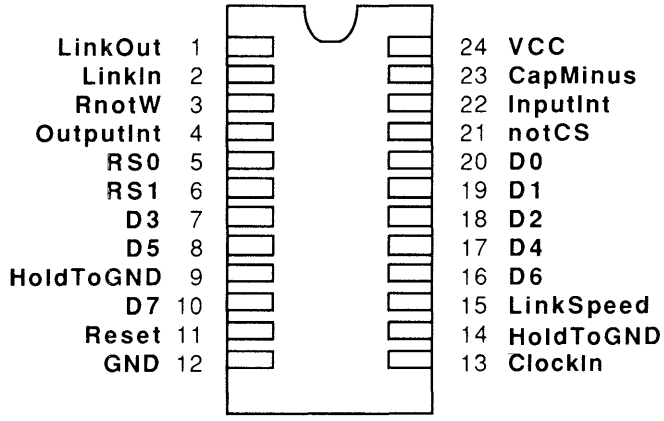
Reset Reset link adaptor.
Input

RnotW Read or write register.
Input

RS0-RS1 Register select lines; used in conjunction with **notCS** and **RnotW** to control the selection of the internal register to be read or written.
Input

VCC Star point for +5 volt supply input and positive end of decoupling capacitor.

The C012 is available in a 24 pin plastic package.



C012 pin out

inmos®

inmos®

INMOS Corporation
PO. Box 16000 • Colorado Springs, Colorado 80935 • (303) 630-4000 • Easy Link 62944936
10190 Bannock Street, Suite 137 • Denver, Colorado 80221 • (303) 252-4100 • Easy Link 62482610
6025-G Alantic Blvd., Norcross, Georgia 30071 • (404) 242-7444 • TWX 810-751-0015
Suite 3001, #2 Westborough Business Park • Westborough, Massachusetts 01581 • (617) 366-4020 • TWX 510-601-6099
9841 Broken Land Parkway, Suite 113 • Columbia, Maryland 21046 • (301) 995-0813 • TWX 710-862-2872
8300 Norman Center Drive • Minneapolis, Minnesota 55437 • (612) 831-5626 • TLX 509995
2620 Augustine Drive, Suite 180 • Santa Clara, California 95054 • (408) 727-7771 • TWX 910-338-2151
23505 S. Crenshaw Blvd , Suite 201 • Torrance, California 90505 • (213) 530-7764 • TWX 910-347-7334
5025 Arapaho, Suite 400 • Dallas, Texas 75248 • (214) 490-9522 • TWX 910-861-0034
INMOS Limited
1000 Aztec West • Almondsbury • Bristol BS12 4SQ • England • Tel (0454) 616616 • TLX 851-444723
INMOS SARL
Immeuble Monaco • 7 rue Le Corbusier SILIC 219 • 94518 Rungis Cedex • France • Tel (1) 4687-22-01 • TLX 201222
INMOS GmbH
Danziger Strasse 2 • 8057 Eching • West Germany • Tel (089) 319-1028 • TLX 522645

72 TRN 006 03