Lecture notes on

transputers

Overview of lectures about the technical description of transputers

GENERAL PRINCIPS OF TRANSPUTERS

I. System design rationale

II. System architecture rationale

III. Occam model
- Overview
- Processes
- Constructions

IV. Physical architecture
- INMOS serial links
- Link electrical specifications
- System services

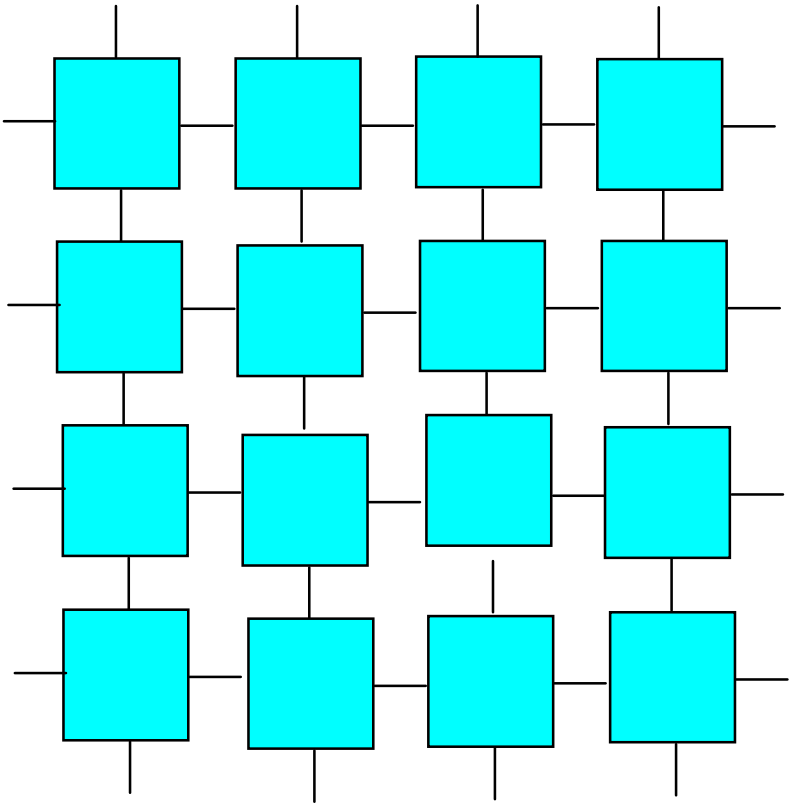
V. Basic architecture and concepts of transputer
- Programable device
- VLSI technology
- Simplified processor with microcoded scheduler
- Transputer internal architecture
- Sequential processing
- Instructions
- Direct functions
- Prefix instructions
- Indirect functions
- Efficiency of encoding
- Support for concurrency
- Communication
- Internal channel communication
- Communication links
- Timer
- Alternative
- Floating point instructions
- Optimising use of the stack
- Concurrent operation of FPU and CPU

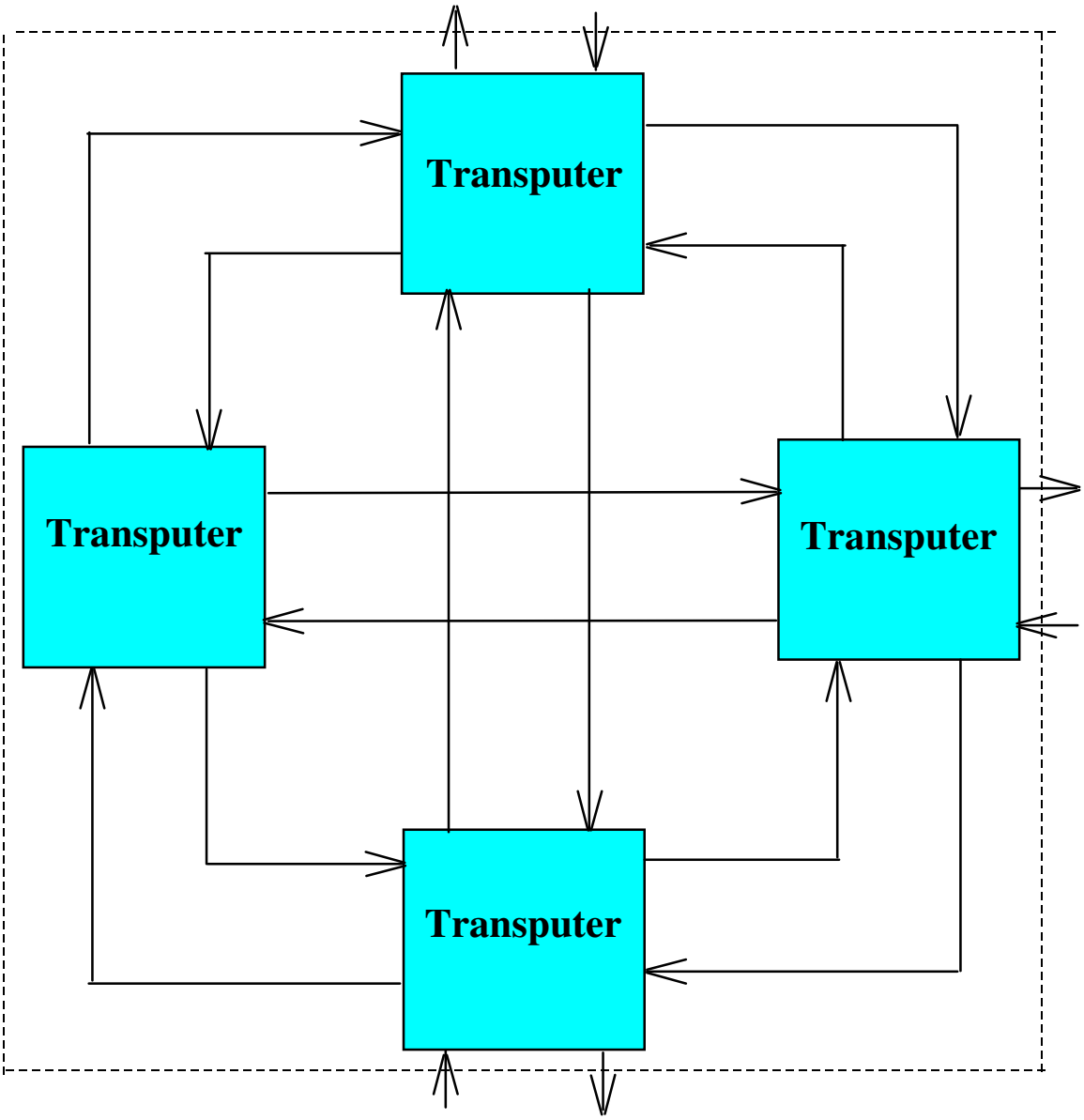
ADVANCED FEATURES OF TRANSPUTERS (T9000)

General princips of transputers

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

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:





A node of four transputers

I. System design rationale I.

Processes:

- standard software and
- standard hardware buildings blocks.

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

The software building blocks is the process:

A system is designed in terms of and 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 managable set of processes.

Occam is based on these concepts, and provides the definition of the transputer architecture from the logical point of view.

I. System design rationale II.

Processes can be implemented in hardware:

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

Other hardware implementations of the process are possible. For example, a transputer with a different instruction set may be used to provide a different cost/performance trade-off. Alternatively, an implementation of the process may be designed in terms of hard-wired logic for enhances 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 graphic drawing and display engine) is defined as an occam process, and implemented in hardware with 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.

A transputer can be programmed to perform a specialized function, and be regarded as a “black box” thereafter. Some processes, preprogrammed transputers and hardware processes. Such a system can, itself, be regarded as a component of 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 has been designed to permit a network of programmable components to have any desired topology, limited only by the number of links on each computer. The architecture minimizes the constrainst on the size of such systems, and the hierarchical structuring provided by occam simplifies the task of system design and programming.

II. System architecture rationale I.

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 standart links, each of which can be connected to a link of some other component. This allows transputer networks of arbitary size and topologyto be constructed.

Point to point communication links have many advantages over multiprocessor 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 communication bandwith does not saturate as the size of the system increases. Rather, the larger the number of transputers in the system, the higher the total communication bandwith of the system. However large the system, all the connections between transputers can be short and local.

Local memory

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

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

II. System architecture rationale II.

Link protocol

To provide synchronised 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 on 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 sinchronized communication of occam. The use of a protocol providing for the transmission of an arbitrary sequence of bytes allows transputers of different word lenght 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 toensure 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 consits of a start bit folowed by a zero bit. The acknowledge signifies both that -

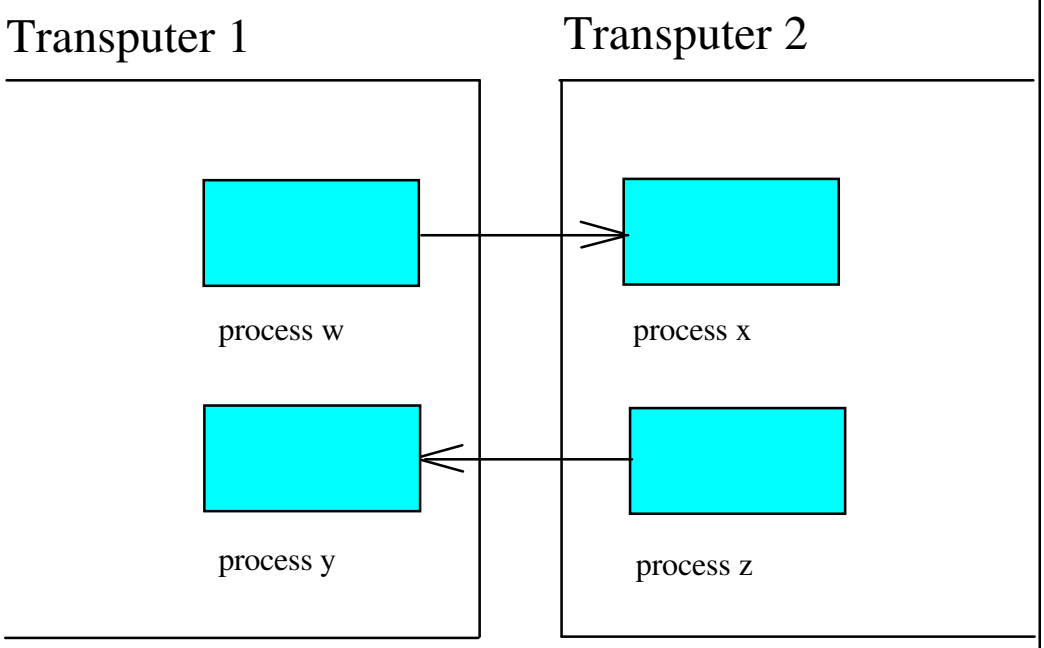
a process was able to receive the acknowledged byte,

- and that the receiving link is able to receive another byte.

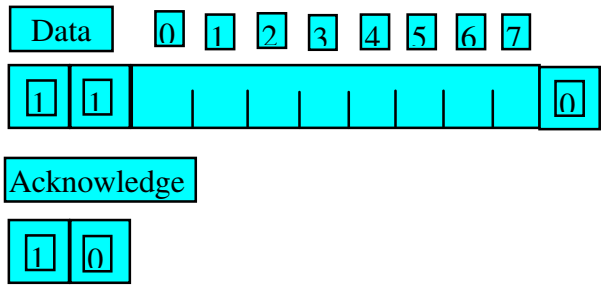
The sending link reshedules the sending process only after the acknoledge for the final byte of the message has been received.

II. System architecture rationale III.

Link of two transputers

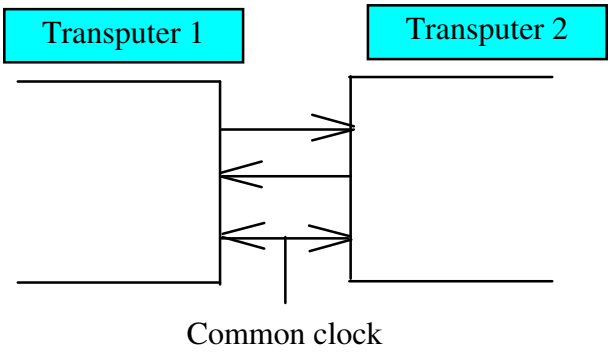


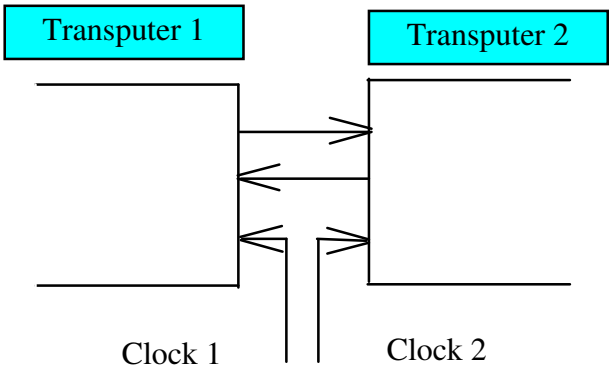
Link protocol



II. System architecture rationale IV.

Clocking transputers





III. Occam model I.

How to access and control the resources of transputers using occam?

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 behavior of many things, for example a logic gate, a microprocessor, a machine tool or an office.

Process

Start **Actions** **End**

- set of sequential processes performed one after another, as in a conventional programming language
- set of parallel processes to be performed at the same time a one another

Input **Assignment process** **Output**

- computes the value of an expression and sets a variable to the value

III. Occam model II.

The concept of communication:

- synchronised 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 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 input channels and output channels performing communication at the same time.

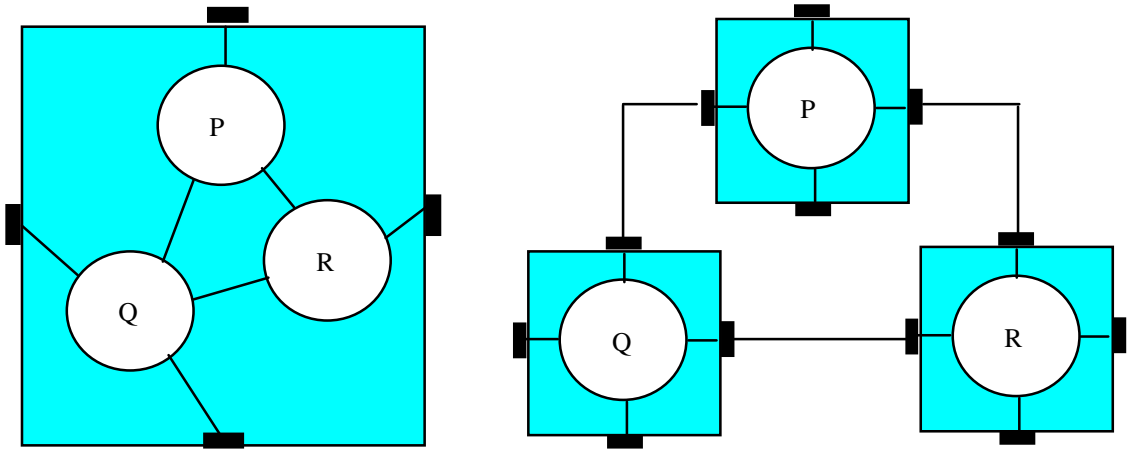
Every transputer implements the occam concepts of concurrency and communication. As a result, occam can be used to program an individual transputer or to program a network of transputers. When occam is used to program an individual transputer, the transputer shares its time between the concurrent processes and channel communication is implemented by moving data with 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 variety of transputer configurations, with one configuration optimised for cost, another for performance, or another for an appropriate balance of cost and performance.

III. Occam model III.

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 instructions sets are designed to enable simple, direct and efficient compillation of occam. Programming of I/O, interrupts and timing is standard on all transputers and conforms to occam model.

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



Mapping processes onto one or several transputers

III. Occam model IV.

Processes

After process starts execution, performs a number of actions, and then either stops and terminates. Each action may be an assignment, an input or an output. An assignment changes the value of the 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 assignmentis 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.

III. Occam model V.

Output

An output indicated by the symbol !. The example:

```
c ! e
```

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

Constructions

A number of processes can be combined to form a construct. A construct is itself a process and can therefore be used as a component of another construct. Each component process of a construct is written two spaces futher 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 constructs.

Sequence

A sequential construct is represented by

```
SEQ
  P1
  P2
  P3
```

The component process 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 program written in conventional programming languages. Note, however, that they provide the performance and efficiency equivalent to that of an assembler for a conventional microprocessor.

III. Occam model V.

Parallel

A parallel construct is represented by

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

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

The parallel construct is unique in 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 optimised so as to incur minmal process sheduling overhead.

Communication

Communication between two processes is the essence of occam programming. At its simplest it requires two processes executing in parallel and a channel joining them:

```
INT x:
CHAN OF INT comm:
PAR
  comm ! 2
  comm ! x
```

This trivial program merely outputs the value 2 from one process and inputs it into the variable x in the second. Its overall effect is exactly as if we had a single process which assigned 2 to x.

III. Occam model VI.

Alternation

A process may use several channels for input, or several for output, or both, in order to communicate with one other process. So we can make choices according to the state of channels. This is made possible by the ALT construction, whose name is short for alternation.

ALT joins together any number of components into a single construction, but the component part of an ALT called alternatives are some more complicated.

The simplest kind of ALT has as each alternative an input process followed by a process to be executed. The ALT watches all the input processes and executes the process associated with the first input to become ready. Thus ALT is basically a first-past-the-post race between a group of channels, with only the winner's process being executed:

```
CHAN OF INT chan1, chan2, chan3 :
INTx:
ALT
  chan1 ? x
     .... first process
  chan2 ? x
     .... second process
  chan3 ? x
     .... third process
```

If chan2 were the first to produce an input, then only the second process would be executed. Here choice is being decided in the time dimension, the inputs causing the program to wait until one of them is ready.

IV. Physical architecture I.

INMOS serial links

All transputers have several links. The
- link protocol and
- electrical characteristics

form are standard for all INMOS transputer and peripheral products.

All transputers support a standard communication frequency of 10 Mbits/sec. Some devices also support other data rates. Maintaining a standard communication 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 number of links and any communication speeds in addition of to the standard speed of 10 Mbits/sec, are given in the product data for each product.

Link electrical specifications

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.

IV. Physical architecture II.

System service

Powering up and down, running and stopping

At all times the specification of input voltages with respect to the GDN 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 specificator 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 starts after reset. The transputer obeys a bootstrap program which can either be in off-chip ROM can be received from one of the links. How to specify where the bootstrap program is taken from depends upon the type of the transputer is beeing used. The program will normally load up a large program either from ROM or 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 owerflow, array bounds violation or divide by zero, causes an error flag to be set in 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 occured.

Clock distribution

All transputers operate from a standard 5 Mhz 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 differences 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.

Bootstrapping from ROM or from 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 transfering 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.

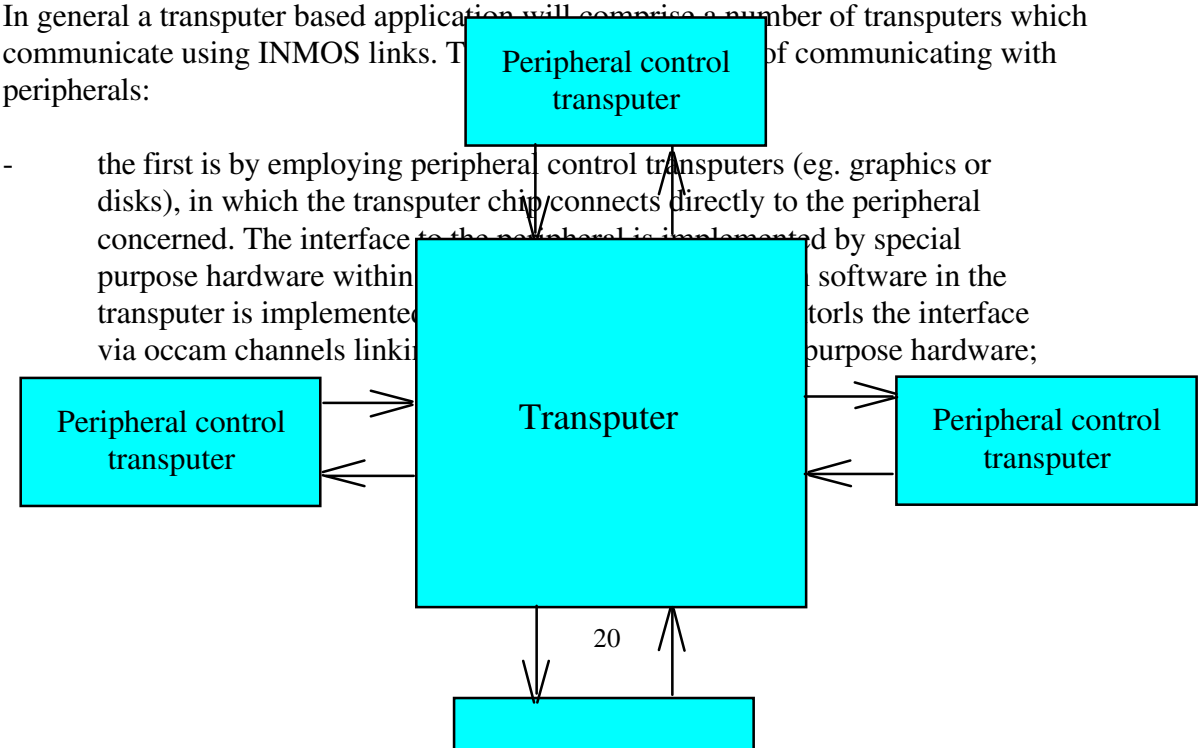
Messages subsequently arriving on the links are not acknowledged until the transputer processor obeys a process which inputs from them. The loading of a network of transputers is controlled by the transputer development system, which ensures that the first message each transputer receives is the bootstrap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

Peripheral interfacing

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

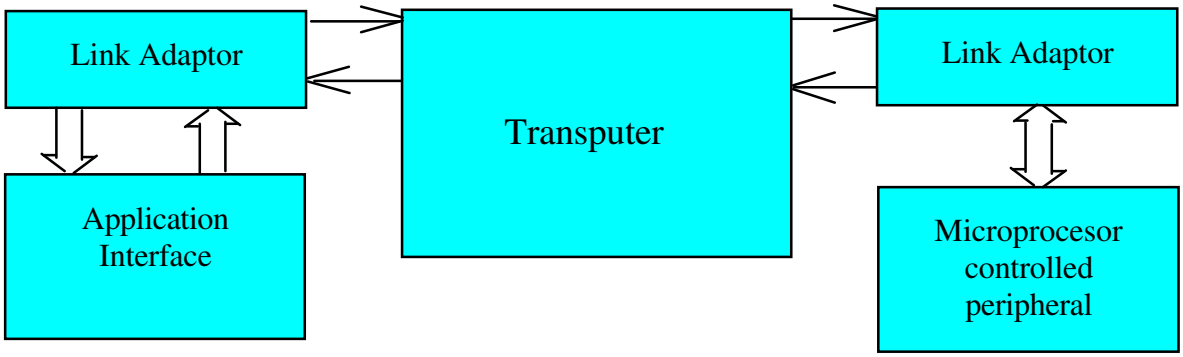
In general a transputer based applica[illegible]mber of transputers which communicate using INMOS links. T[illegible]of communicating with peripherals:

- the first is by employing peripheral control transputers (eg. 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[illegible] software in the transputer is implemente[illegible]torls the interface via occam channels linki[illegible]purpose hardware;



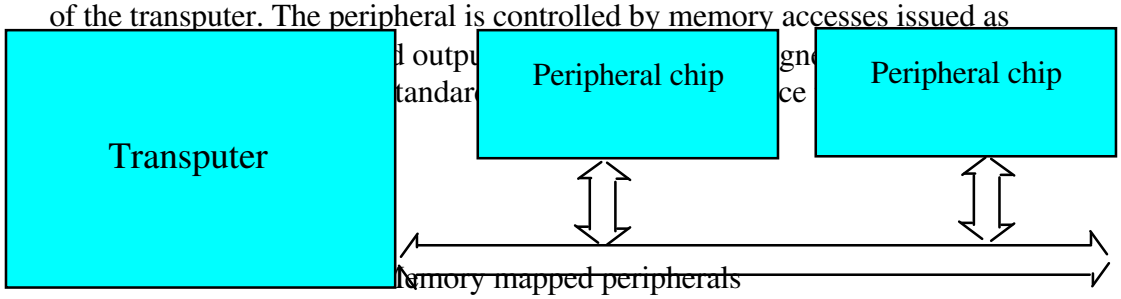
Transputer with peripheral control transputers

- the second method is by employing link adaptors. These devices convert between a link and a specialised 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 the transputer. The peripheral is controlled by memory accesses issued as



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

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