Reprinted from Electronic Product Design, August 1986

Developing transputer application circuits

The design, implementation and uses of an lnrnos Transputer evaluation board when hosted by the IBM PC, is described here by Stephen Ghee

To complement the launch of the Inmos Transputer in October 1985, a number of evaluation boards were designed to let users assess for themselves the performance and ease of use of the transputer. A number of hosts to control the boards were considered, and the IBM personal computer was chosen due to its wide usage and large programming base. In describing the design, implementation and uses of the board, this article covers it both as an Occam engine in development of Occam for transputer systems, and also the use of the board as an acc-elerator for computational intensive tasks that can be passed to the board from host programs.

The transputer is a complete computer on a chip. There is a 32bit processor, with a 5Onsec processor cycle, 2kbytes of fast (5Onsec cycle) static RAM, four serial communications `Links' (for external communica-tions) and a programmable memory interface (which allows up to 4Gbytes physical memory external to the transputer).

The transputer and the programming language Occam evolved together. Occam was designed to simplify the task of concurrent programming. An Occam program is made up of a number of processes which can be declared to run sequentially or concurrently. Concurrent processes, which cannot use shared resources, communicate across Occam channels. These channels are single direction, point to point connections between processes, and give synchronised message com-munication.

Concurrent programming evolved as it became clear that many programs could be split into a number of tasks which could be operated on independently and use some form of message passing for passing results, parameters and synchronisation.

On standard machines, implementing concur-

Fig 1

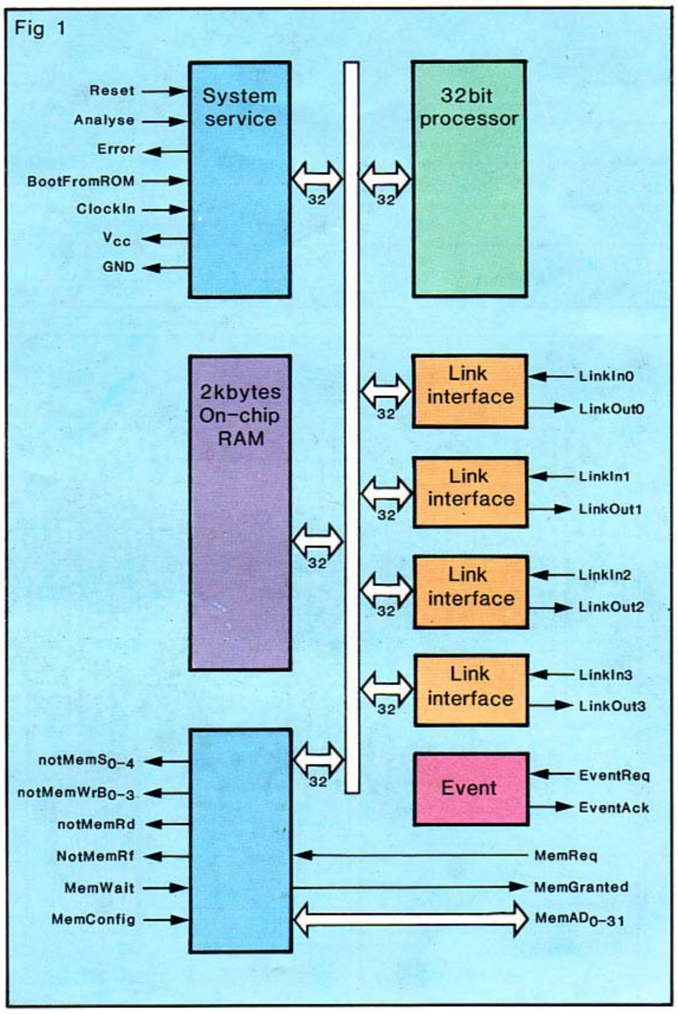


Fig.1 Transputer block diagram

Fig 2

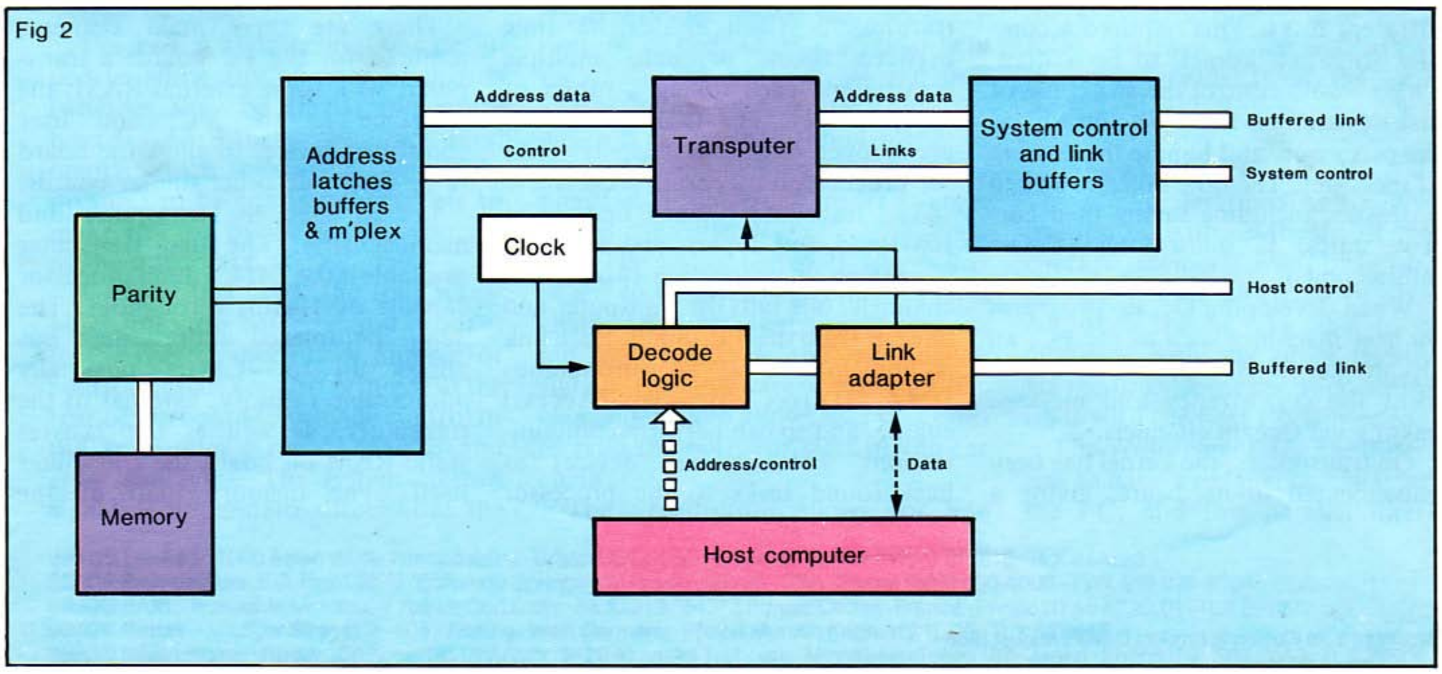


Fig.2 B004 schematic

Fig 3

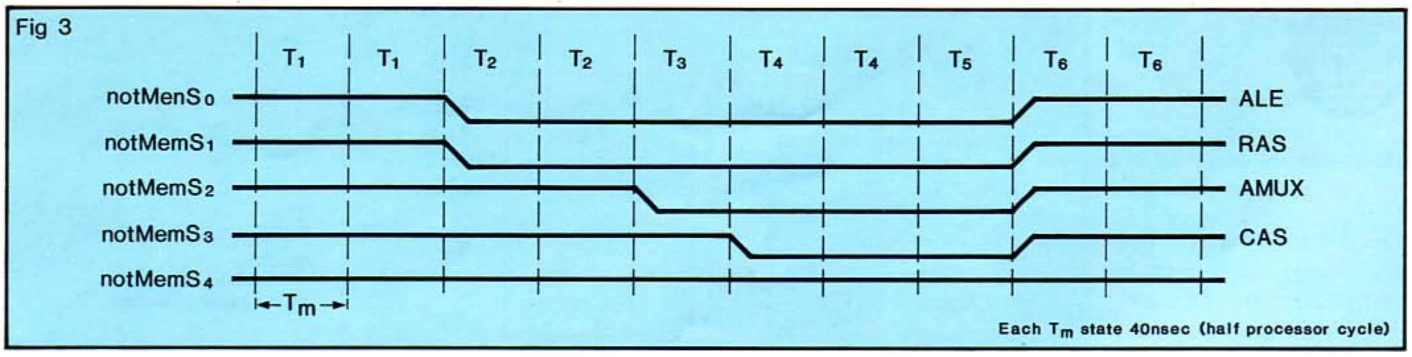
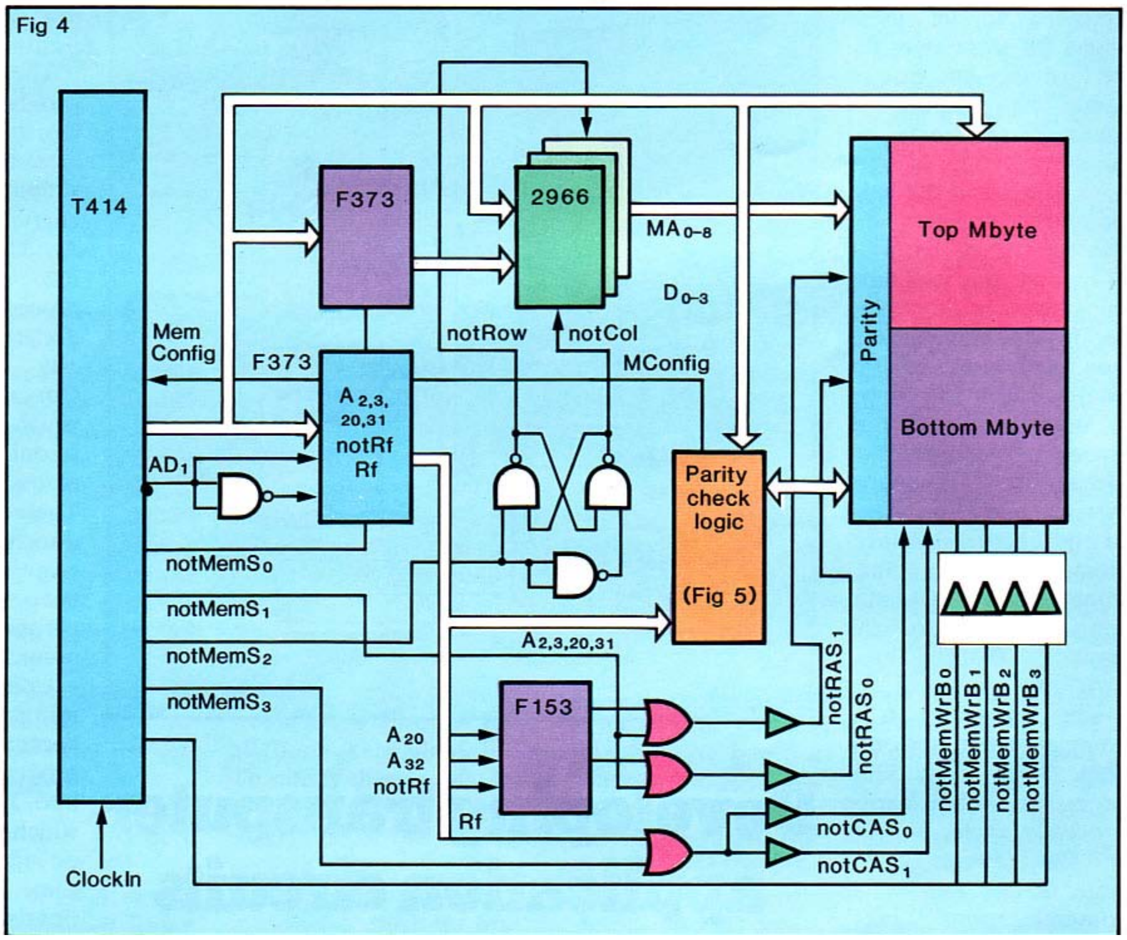


Fig.3 Memory interface configuration (5 processor cycles)

Fig.4 Circuit for the transputer memory

Fig.5 Schematic of the parity generator and checker

Fig 4



rency was solved by making the processor share its time between the different tasks. This required a complex software `kernel' to be written, which would control the switching of tasks (including itself) in and out of the processor, and handle the passing of messages. The time taken to switch processes, including saving their current states, is quite long (a few milliseconds).

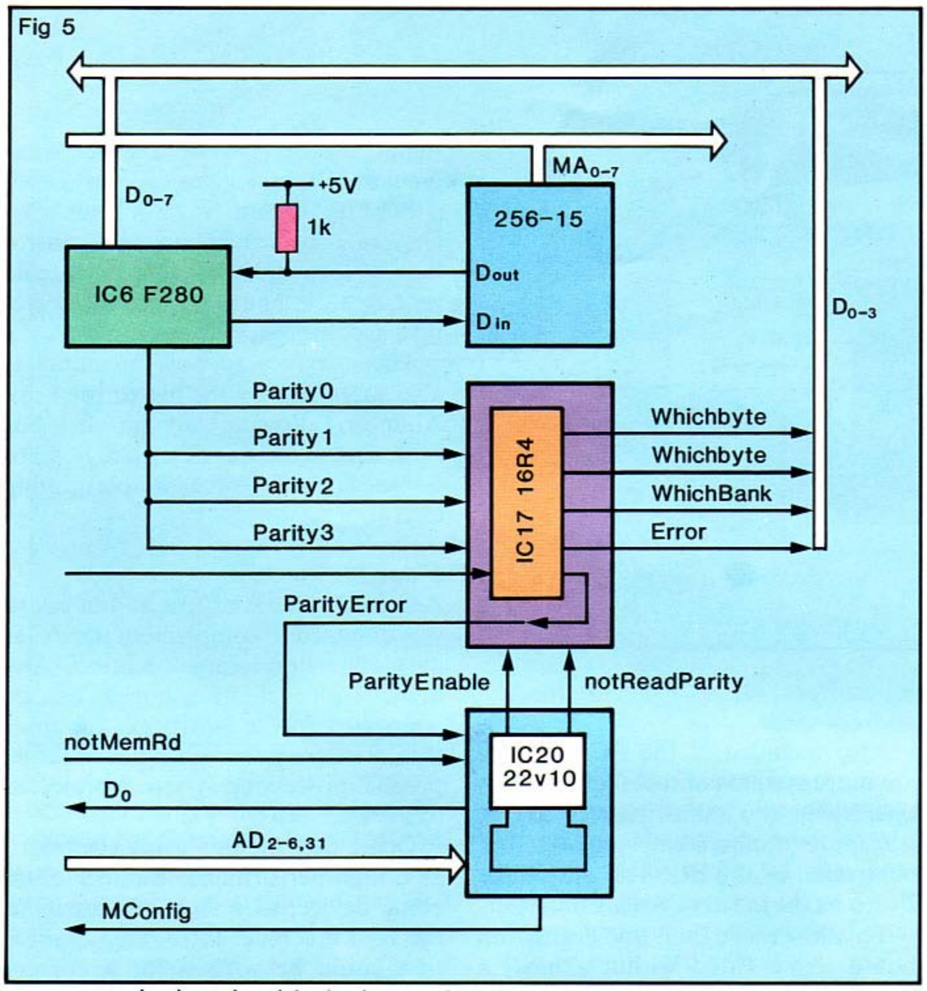
When developing Occam programs on host machines such as the PC, an Occam kernel is supplied to implement the concurrency and message passing via Occam channels. On transputers, the kernel has been implemented in hardware, giving a submicrosecond task switch. Occam processes can be mapped onto one transputer, which shares its time between them, or onto multiple transputers, each taking a subset of the processes. The Occam channels are mapped onto the transputer links for processes on separate processors.

The transputer's links operate at 10MBaud, full duplex, and each link is capable of supporting two Occam channels, one into the transputer and one out from the transputer. Each link is implemented as an autonomous DMA (Direct Memory Access) engine, and so can perform communications with external devices as background tasks to the processor with negligible performance degradation.

There are three main elements required for the PC board: a transputer, with some external RAM; the interface to the PC; and user controlled devices to allow the board to be used with other similar boards.

Let us take the transputer and memory first. The first transputer available is the T414, a 32bit processor capable of 10mips throughout. The 32bit multiplexed address/data bus allows up to 4Gbytes physically addressable memory, external to the transputer, as well as the 2kbytes static RAM on board the transputer itself. The memory map of the

Fig 5



transputer is signed, with the internal RAM starting at the most negative address (Hex 80000000 to Hex 800007FF). The external memory starts at Hex 80000800.

For the PC add-in board, it was decided to give the user up to 2Mbytes external RAM, mapped in to the negative half of the available address space. For this amount of RAM on an IBM f'orm-factor board, dynamic RAM had to be used. Also for such a large amount of memory, a parity checking system also had to be implemented.

Communication link

The communication with the host PC is handled using an Inmos C002 Link Adaptor. This device can convert serial link data into byte-wide parallel data, and visa versa. The device allows simple interfacing with standard bus architectures, appearing as a memory mapped peripheral.

A number of system control signals are also provided which give the user the possibility of connecting a number of transputer boards in the add-in board via the Inmos links, and to allow the add-in board to control the system of transputers. All signals are software controlled.

To ease the task of designing memory systems, the transputer has an on board programmable memory interface. With this interface it is possible to configure the external memory cycle of the transputer to be any width (to suit slow or fast memories), and a number of programmable strobes are supplied, which can be programmed to give signals such as RAS and CAS for dynamic memories. Automatic refresh, over a selectable refresh cycle time, can also be chosen. This eliminates the need to design complex timing generators and so cuts down the number of devices required on the board.

To give up to 2Mbytes of RAM on an IBM form factor board, 256k x 1 DRAMs were chosen. The memory interface needs to be configured to suit the cycle times of these devices. For evaluation purposes, 150nsec parts were chosen.

The external memory cycle is split into six states. These `T states' are as follows:-

1. Address set up
2. Address hold
3. Read cycle tri-state/write cycle data set up
4. Extended for WAIT states
5. Read or write data
6. End tri-state/data hold

The configuration of the cycle is done by specifying the number of each of these states required to give the timing needed by the particular memory device used. The minimum is one `T state' per section of the memory cycle, the maximum being four. There is also a wait input to the memory interface which can be used to extend the cycle time by inserting extra T, states. A package is included in the transputer development system for calculating the configuration.

Each T state of the memory interface is 40nsec long (on the current 80nsec processor cycle part), and so this is the minimum resolution to which external memory cycle can be programmed. For the 150nsec parts used for the memory the configuration shown in Fig. 3 was drawn up. The refresh interval is set to be 54 processor cycles and early write is enabled.

The transputer has a number of predefined configuration patterns that can be used, but the configuration described in Fig. 3 does not match any of these so we must supply the configuration from an external device.

For this particular board a PAL was chosen due to its small size and number of functions available, not only for the configuration but for other peripherals (described later).

The PAL is programmed as a finite state machine which responds to the addresses output by the transputer during the configuration stage. For each address, a bit is fed into the MemConfig pin.

Once the memory interface has been configured correctly, the logic for controlling the memory must be designed. As the memory interface controller on the transputer generates all of the signals we need, the only logic components required are buffer and latch devices-the buffers are required for the long track lengths along the memory array.

The transputer memory bus is a 32bit, multiplexed address/data bus. As the address is not present for all of the cycle we must latch the upper addresses we need for the DRAMs. Two latches are used to capture the signals needed, the latches being enabled by the falling edge of notMemS$_0$ at the start of T$_2$ At this point we can be sure that the addresses are stable on the bus. At the same time we can strobe in the row address (the low order address bits present on the bus during T_2) to the DRAMs using the RAS signal derived from $notMemS_1$. Once the row address has been latched by the RAM, the column address must be presented to the RAM address inputs.

When stable, the column address can be strobed in using the CAS signal derived from $notMemS_3$. $notMemS_2$ is used to perform the switching of the row-column addresses. The circuit for the memory is shown in Fig. 4.

Two banks of 1Mbyte each have been implemented. The banks are selected by one of two RAS signals, these being decoded from the state of A_{20} (bank select address bit), and the refresh signal given out on AD_1 (MemNotRefresh). The CAS circuitry generates a single CAS strobe from the $notMemS_3$ strobe, which is split, buffered, and fed to the two memory banks. The refresh signal (inverted AD1 for the CAS generator) is used to disable CAS during refresh cycles.

A flip-flop is used to create the non-overlapping strobe signals needed to switch the memory address from Row to Column. The flip-flop is triggered by the $notMemS_2$ strobe.

For such a large amount of dynamic memory, a parity checking circuit needs to be included. The transputer has overall control of the parity system, as the parity checker exists as a number of memory mapped locations. Simple memory accesses from processes allow parity to be enabled/ disabled, the parity error flag read and routing of the parity error flag to other transputers. Fig. 5 shows the schematic of the parity generator and checker.

The parity control is handled by IC20. This PAL does the address decoding and handles the registers that are available to the user (these registers control the Enable/disable parity, read the parity error flag, and route parity error to the main board error line).

For each byte in main memory, there is a parity bit stored in associated addresses in the parity RAM. On a memory write cycle, each byte is converted to odd parity, the 9th parity bit being written to the parity RAM. On read, the parity bit for the particular address is passed into the parity checker along with the byte from main memory and parity checking takes place. If the parity is odd, the Parity(x) output signal will remain low. If an even parity situation is found, the Parity(x) signal will be asserted. This signal is read by the parity decoding logic in PAL IC17.

On memory read cycles, the four ParityError signals from the parity checkers are latched, and if parity is enabled (ie ParityEnable is asserted), the signals are decoded to indicate whether a parity error has occurred,

Fig.6 Hierarchy of board connection

Fig.7 Up, Down subsystem schematic

Fig 6

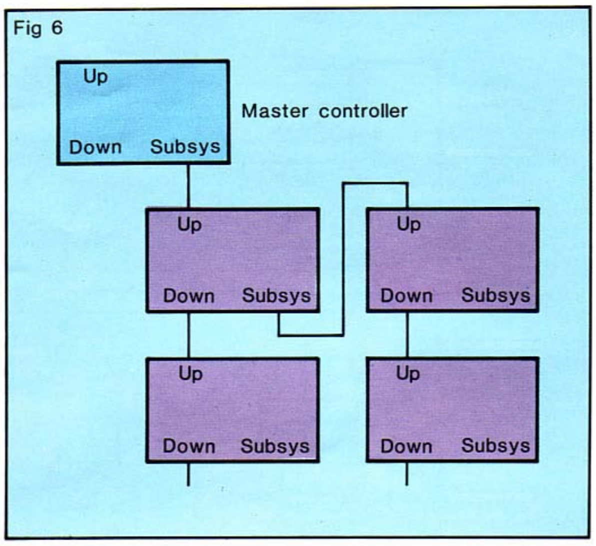


Fig 7

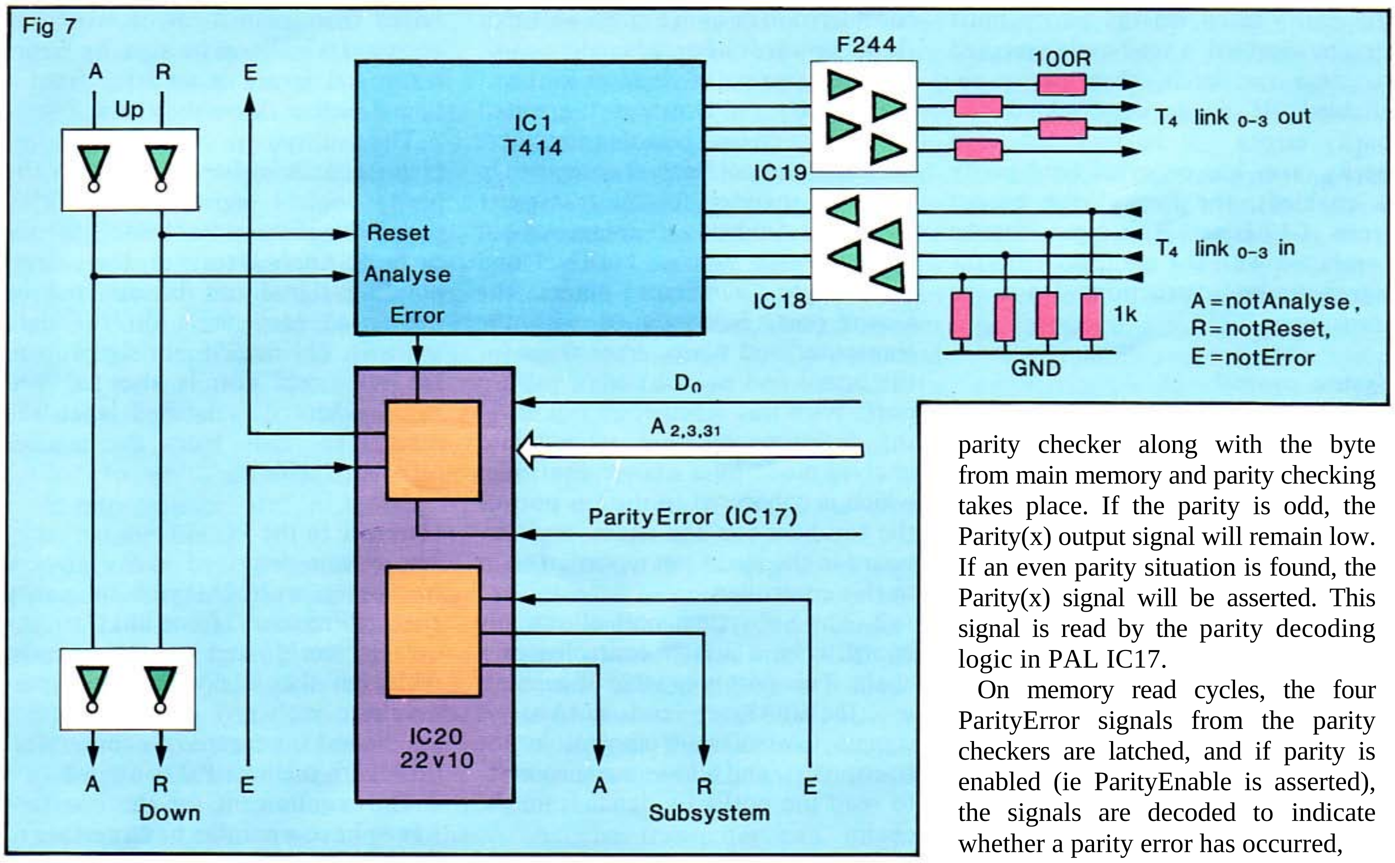


Fig 8

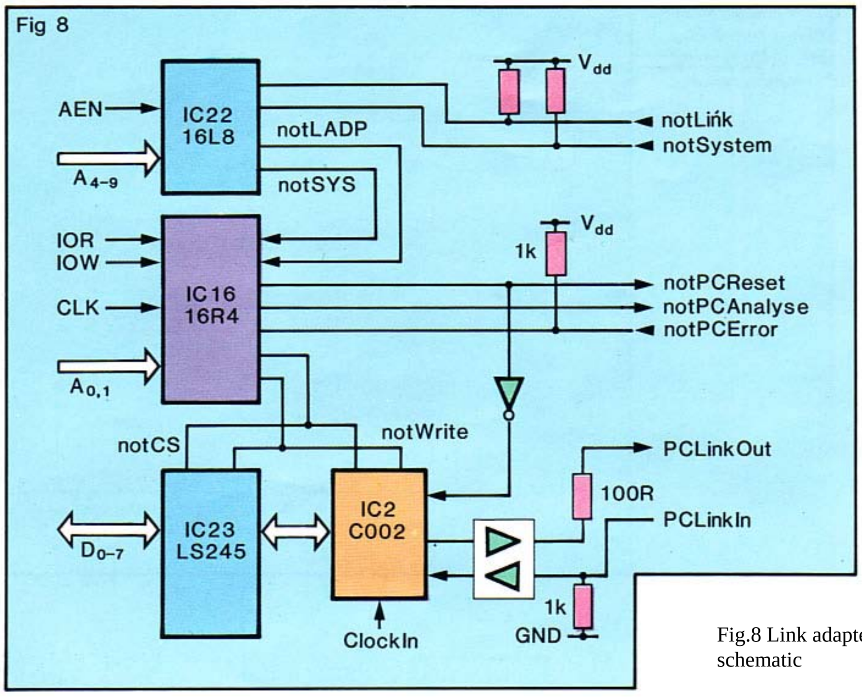


Fig.8 Link adapter schematic

and which byte and bank the error occurred in. The output lines of IC17 chosen for these decoded signals have tri-state buffers which are enabled by the notReadParity signal from IC20. This is asserted when the transputer does a memory read from the ParityError register.

If parity is enabled, then once a parity has occurred, its state (ie byte, bank) will be latched and no further parity errors will overwrite the first. Reading a parity error will not reset the parity latch, so the parity must first be disabled, a read cycle executed to clear the latch, then parity re-enabled. If parity is disabled, no parity errors will be reported. If a parity error has occurred (and parity is enabled), the ParityError output from IC17 is set. This signal can be combined with the main board error signal under instructions from the transputer.

System control

The board architecture was designed to allow any number of boards (whether they be Inmos evaluation boards or other manufacturers' boards which follow the same architecture) to be connected together to make full use of the parallel concept of the transputer.

To meet this need, the four transputer links are buffered and brought to the rear connector. A number of control signals are also brought to the rear of the board to provide control of the transputer's reset, analyse and error functions. The system control signals are standard throughout the transputer evaluation boards, and to understand the needs for the hardware to implement these control lines, the system architecture of the boards needs to be examined. Fig. 6 shows how a hierarchical network of boards can be built into a large system.

The Up, Down and Subsystem ports all carry the same control signals. These are: notReset; notAnalyse; and not Error. notReset and notAnalyse flow down the system (if the configuration of the system is considered to be as in Fig. 6), notError flows upwards. For a single board, there are two paths we must look at.

1. The Up and Down ports are used to daisy chain boards together. notReset and notAnalyse enter the Up port, are inverted for the transputer Reset and Analyse and are passed out to the Down port as notReset and notAnalyse. notError enters the Down port, is combined with the transputer and parity error flags for the board and passed out of the Up port. With this scheme, all boards in the chain can be reset, or put into analyse mode by a master controller which is connected to the Up port of the top board in the chain, and any board in the chain can report an error to this controller.

2. The Subsystem port allows any board to be a master controller of a chain. This port is capable of generating the notReset and notAnalyse signals (by software control in the transputer), and allows user processes to read the notError signal from the chain.

The architecture allows for any board in a chain to act as a master for another chain, allowing large systems to be split up into smaller subsystems, each with its own local controller.

The subsystem signals are handled by the 22v10 PAL IC20 (which handles the parity controls also).

The logic required for the Up, Down ports is simple. The notUpReset and notUpAnalyse for the daisy chaining of boards enter the board from the Up port, are inverted (IC21) to give tile correct polarity for the transputer Reset and Analyse, and are then inverted again for output at the Down port.

The second inverter is used to provide the drive current required between boards. Both inputs (from the Up port) are pulled high with 1k resistors to prevent spurious resets, etc, occurring when no external boards are connected.

The notDNError signal enters the board from the Down port and is fed to IC20 (again, a 1k pull up resistor is used to hold the line at the `no error' state when no other boards are connected). In IC20, the notDNError is combined with ParityError (IC17) and the T4 Error signal from the transputer and fed out to the notUpError output of the PAL. This signal can inform the controlling board that some form of error has occurred whether it be a parity error, transputer error, or an error from a board further down the chain.

The subsystem signals are implemented as registers similar to the parity control registers. By writing certain bit patterns to the notSSReset or notSSAnalyse register, the corresponding signal can be asserted or de-asserted (depending on the data written). The notSSError signal from the subsystem port is also fed into IC20, where it is latched when the transputer reads from the register associated with it.

Interface to the PC

The circuit described above gives a transputer with 2Mbytes of parity checked memory four links to the outside world, and control signals. Although this is now able to communicate with any other transputer via the links, a means of communications with the host PC is needed.

The requirement for the interface was to have a parallel bus interface to the PC, and this parallel interface

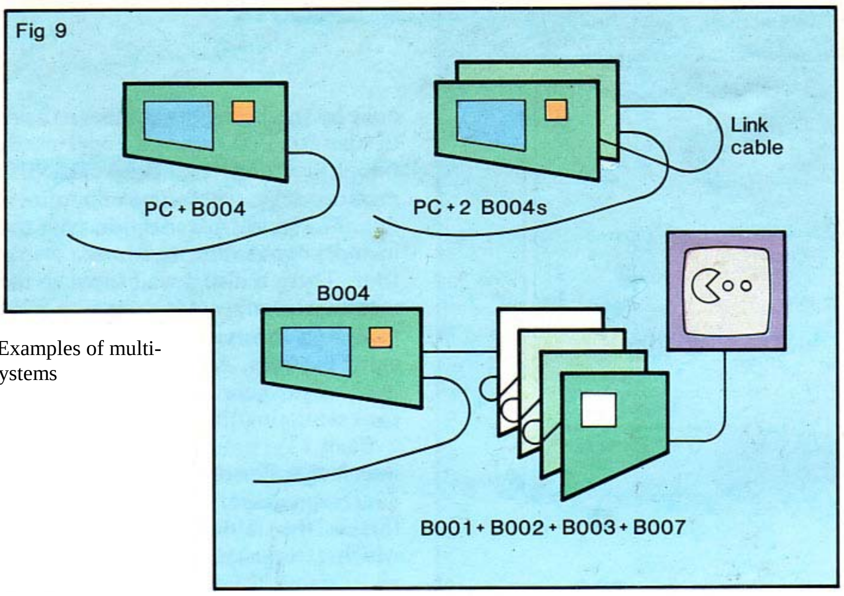


Fig.9 Examples of multi-boar systems

communicating with a transputer. There were two possible methods: memory map the bus communication hardware into the transputer's external memory; or communicate with the transputer via one of the serial links.

The second method was chosen, as it maps onto the transputer concept of communication via Occam channels. In this case the host computer will appear as a process at the end of a channel mapped on one of the transputer's links.

This method, however, means that the transputer selected to communicate with the host must use one link solely for this purpose.

To make this sort of interface possible, Inmos has also produced devices which convert parallel data into serial data, and vica versa, following the protocol of the Inmos links.

The IMSC002 link adaptor has a bi-directional 8bit bus to allow easy connection to standard bus architectures. A number of standard control lines, such as chip-enable and read/write, are included. The device takes parallel data, converts it into serial form and passes it out of the `output link'. Incoming serial data is converted into parallel form, to be read by the parallel bus.

The circuit to perform the communication with the PC is outlined in Fig. 8. IC22 is a PAL which handles the address decoding from the bus. A_4 to A_9 are decoded, as well as AEN (when AEN is active, DMA is active on the PC bus and so the board must be de-gated from the bus). Using a PAL for the address decoding allows the board address to be altered by changing the PAL. The board is currently located at address HEX 150-163 in I/O address space.

Also included at the PC interface are more system control signals which operate in the same manner as the transputer subsystem signals, but controlled by the PC. This allows the PC to be the master system controller. To allow more than one evaluation board to he fitted within a host, a method of selecting only one link adaptor and system control circuit to respond to accesses by the host was required. This selection method would allow all other boards to have all four links available for general use. To satisfy these needs, it was decided to take the link from the link adaptor to the rear of the board and to use a jumper plug to connect a transputer link to the link adaptor. A mechanism is included which informs the decoding logic that tile jumper plug is in place and the board can respond to the PC. The system control signals (controlled by the PC) are also taken to the rear connector, using a jumper plug to connect to the Up port, and using a similar selection mechanism to the links.

The selection mechanism involves an input of the selection PAL (1C22) to be pulled low if one of the jumper plugs is inserted. Only if the line is low will the corresponding select signal (notLADP or notSYS) be asserted.

The notSYS (PC system control is being accessed) and notLADP (link adaptor access) signals are used in conjunction with the IOW, IOR and Clock signals from the bus to generate the access timing sequence for the link adaptor.

As the transputer can be programmed to supply all the timing signals necessary to build an external memory system, the design of a transputer system consists mainly of buffering address, data and control lines to provide sufficient drive current to take board capacitance, etc, into account.

The interface to host machines is also simplified by the use of the Link Adaptor, allowing any parallel bus (not just the PC bus described here) to be used, with sonic simple timing alterations.

Using the board

As stated at the start, the add-in board was designed to complement the PC as an Occam development station. Any number of evaluation boards can be connected to the board via the links (Fig. 9 shows a few examples of large, parallel processing systems connected to a PC).

Other uses include using the board as a high-performance number crunching device, as a slave processor to the host machine. Here, any application could be written (or a current application modified) to pass data to the board which will perform certain tasks on that data, before passing it back to the host machine for displaying, etc. The interface software sees the board as a number of memory mapped devices. For example, a high performance flight simulator could be written, the transputer system doing the complex trigonometry involved with aircraft position, windowing, 3D to 2D conversions and passing vector information to the host computer for display. By using Multiple boards, the tasks could be split between many processors, giving improvement in performance.

Programming the board for specific applications is done using the Transputer development systems, using Occam. In the near future, it will also be possible to program the transputer system in 'C', Pascal and Fortran, as well as Occam. With these language compliers, it will be possible to take existing algorithms, and re-compile them for transputer applications. Also, by using Occam as the harness to describe the concurrency of the system, it is possible to run multiple processes, which could be written in any of the four languages.

Stephen Ghee is with Inmos. The company can be contacted on 0272 290861.