DEUTSCHES ELEKTRONEN-SYNCHROTRON DESY

DESY 90-024
March 1990

An Introduction to Transputers

T. Woeniger
Deutsches Elektronen-Synchrotron DESY, Hamburg

ISSN 0418-9833

NOTKESTRASSE 85 · 2 HAMBURG 52

DESY 90-024
March 1990

ISSN 0418-9833

An Introduction to Transputers

Torsten Woeniger
DESY - Hamburg

March 12, 1990

Abstract

An effective inter processor connection is one of the major problems of the data acquisition systems of modern high energy experiments. Transputers are fast microprocessors which can be easily interfaced with each other via serial connections called links. An introduction into the concepts and the performance of transputers is given. As an example two transputer hardware modules are presented. Several parts of the ZEUS data acquisition system (Global Second Level Trigger, Eventbuilder, Calorimeter-Readout etc.) are based on these hardware modules.

Contents

List of Figures

List of Tables

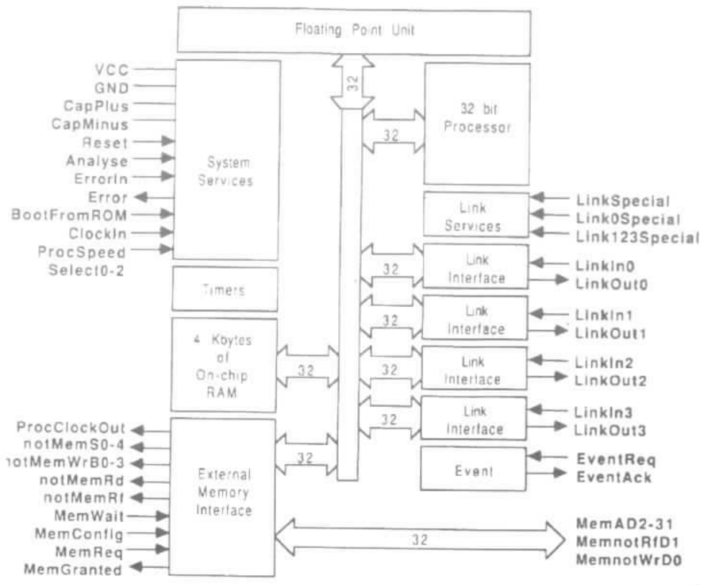


Figure 1: Block diagram of the T800 transputer

1 Transputers

1.1 Hardware Structure of Transputers

Transputers are powerful microprocessors with fast serial interfaces. These serial interfaces, called links, provide a very efficient tool for inter processor communication.
In figure 1 a schematic overview of the T800 Transputer is given. The T800 is the most advanced transputer on the market. All components which are shown in this figure are integrated on one 84 Pin Grid Array Chip.
The central part of each microprocessor is the Central Processor Unit (CPU). The 32 bit CPU of the T800 has been designed as a RISC [1] - Processor. This processor design differs from that of the conventionally designed processor which is called a CISC [2] - Processor. Until the mid-eighties all microprocessors used the CISC design. In comparison to CISC - Processors which have a large instruction set with many different instructions and addressing modes, the instruction set of a RISC - Processor is very small. However, the CISC - Processors need a relatively large number of clock cycles for their complex instructions (up to several hundred) because these are combinations of many simple instructions, called microcode instructions.

[1]RISC = Reduced Instruction Set Computer
[2]CISC = Complex Instruction Set Computer

In contrast, a RISC - Processor has its own logic for most commands. With this fast fixed logic such a processor is able to execute most commands within very few clock cycles.
The on-chip memory is a fast RAM (4 Kbyte on a T800); it is faster than the external memory : there is no need for external drivers because the electrical capacities of the bus lines inside the chip are much smaller. The access time of the internal RAM can be compared with the access time of registers in conventional processors.
The T800 is the only member of the transputer family which has a floating point unit (FPU) on the chip. The format of the FPU follows an international standard [3] . For the same reason that an on-board memory is faster than an external memory, an on-chip FPU is faster than an external floating point coprocessor. The FPU, which works independently of the CPU, shows the remarkable effect that it performs floating point operations slightly faster than the CPU performs integer arithmetics.
Other useful hardware units which are integrated on the chip are the timers. These are two clocks, each of them incrementing a special register. The slow clock is ticking every 64 μs and the fast one every single μs. Because a timer can start a process, there is a wide field of applications for these timers. Examples for these applications are high accuracy time measurements, time delays for suspended processes, etc.
The connections between a transputer and the outside world can be classified as the System Service Connections, the External Memory Interface, the Events, and the Links.
Firstly, the transputer like any other computer, needs electrical power, external clock frequency, etc. The transputer gets these via the on-chip circuits which belong to the group of System Service Connections.
Secondly, there is the need for an External Memory Interface. The fast on-chip RAM is too small for large programs or huge amounts of data, and a fast processor needs both for its work. To reduce the number of pins and amount of board space for bus lines, the external 32 bit address and 32 bit data lines are multiplexed together on to 32 bus lines. The memory configuration (wait cycles, refresh interval etc.) can be defined externally over some additional lines and the refresh unit which is required for DRAMs is also integrated on the chip.
A computer has to react also to events from the outside world. In this case a conventional computer would be 'interrupted' by interrupt lines and routines. In a transputer the 'events' are responsible for the interrupts. If a signal comes from the outside world (e.g. a keyboard or a trigger) a program can be started which handles the necessary actions (e.g. gets a byte from the keyboard or a data block from an ADC-card). In comparison to the interrupt system of a conventional computer, the transputer's events do not have as many possibilities but are much easier to program. This is especially true in the case of high level languages programming.
However, the greatest advantage of a transputer compared to a normal microprocessor consists in the transputer links. Links are high speed serial interfaces which have a very effective protocol. Each link is served by its own DMA controller so that it can operate more or less independently of the rest of the chip. Further details of the performance of the system are presented in chapter 2.
One advantage of the transputer concept is that the necessary hardware around a transputer chip is as simple and unique as possible. For that goal the chip designers tried to place as much as possible on the chip itself. One example is the implementation of the processor clock.

[3] ANSI-IEEE 754-1985 Floating Point Representation

Transputer type	Wordlength	internal RAM	Linkspeed	No.of Links	Comments
T 212	16 Bit	2 Kbyte	slow	4	
M 212	16 Bit	2 Kbyte	slow	2	Disk Controller
T 222	16 Bit	4 Kbyte	fast	4	
T 414	32 Bit	4 Kbyte	slow	4	
T 425	32 Bit	4 Kbyte	fast	4	
T 800	32 Bit	4 Kbyte	fast	4	with FPU

Table 1: List of the available transputers

Although the internal processor clock speeds of the T800 are 17.5, 20 and 25 MHz, only a 5 MHz quartz signal is needed from the outside. The higher frequencies are generated on the chip.
Table 1 shows an overview of the transputers which are available today.

1.2 Software Structure of Transputers

A conventional processor has several registers for data on the chip. This provides the processors with a fast access to the most often used variables. But whenever a completely new process is started, the content of all registers has to be stored in the memory. Subsequently the data for the new process have to be reloaded into the registers of the processor. Reasons for this so called task switching may be either a task swapping inside a multitasking machine or an interrupt that has to be handled. Conventional processors need between 20 and several hundred μs for task switching.
Another way of handling processor registers is the so called Workspace Concept. Processors using this concept keep all their registers in memory. On the processor chip itself are only registers for addressing the actual program command (Program Counter) and the location of the registers in memory (Workspace Pointer). If the processor has to switch to another task only the contents of the Program Counter and the Workspace Counter have to be changed. It is obvious that this can be done very quickly. The argument against the Workspace Concept is that an access to an on-chip register is always faster than an access to the external memory. For this reason the first processors with the Workspace Concept (TMS 99XX) had fast task switching instructions, but were fairly slow at every other computing task.
To avoid these problems, transputers are designed to have the advantages of both systems. In figure 2 the structure of the main registers of a transputer is shown. There is a Workspace Register which points to a list which is called Locals. This local list contains the local variables and the pointers of this process. One sees also a Program Pointer (labled `Next instr.`) which points to the next instruction in the program. The three registers above (A, B, C) are the so called evaluation stack which is organized like a LIFO [4]. The evaluation stack is used as a short time memory for most of the commands. With these registers the transputer has all speed advantages of permanent registers on the chip. In order to speed up task switching even further there is a hardware scheduler on the chip. It stores the data from the scheduling

[4] Last In First Out

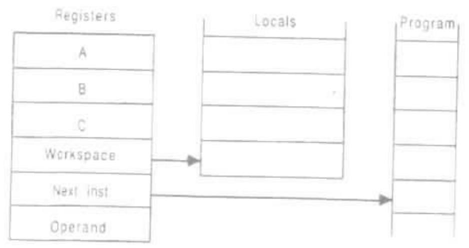


Figure 2: Structure of the main registers of the transputer

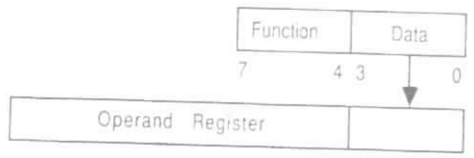


Figure 3: Format of the transputer assembler commands

of all the processes into several internal registers which are not indicated in the above figure. With this highly advanced hardware scheduling mechanism the transputer switches from one process to another in 1μs. Also, suspended processes consume no CPU power. This hardware helps to avoid performance degradation in case of several concurrently running processes. If a program can be more easily written using several concurrent processes there is no reason why one should not do this.

To reach high computing speed, the structure of the assembler commands is also optimized. Together with the RISC-design of the hardware there is also a RISC-design of the assembler commands implemented. The register (Operand) at the bottom of figure 2 is used for this purpose. The idea of the RISC assembler is the following. The most often used instructions, which include Function and Data, are only eight bits long (figure 3). The first four of these bits define the function, the second four bits the data. Thirteen of the possible sixteen commands (2^4), the so called direct commands, are used in this way. Measurements have shown that 70 % of the executed instructions can be encoded in these single byte commands. There are 16 of these instructions which are only one byte long. Two of these sixteen commands are the so called prefix instructions (2X and 6X in table 2). These instructions have to be used if the operand is larger than 15. In this case one prefix instruction (pfix) is required for each additional four bits of the operand. The other prefix instruction, called negative prefix (nfix), is used as the first prefix if the operand is negative. The prefix instructions tell the processor that the data of this 8 bit command has to be combined with the data of the next command by shifting the command in the operand register eight bits to the left. This means

Memory Code	Mnemonic	Processor Cycles	Name
0X	j	3	jump
1X	ldlp	1	load local pointer
2X	pfix	1	prefix
3X	ldnl	2	load non-local
4X	ldc	1	load constant
5X	ldnlp	1	load non-local pointer
6X	nfix	1	negative prefix
7X	ldl	2	load local
8X	adc	1	add constant
9X	call	7	call
AX	cj	2	conditional jump (not taken)
		4	conditional jump (taken)
BX	ajw	1	adjust workspace
CX	eqc	2	equals constant
DX	stl	1	store local
EX	stnl	2	store non-local
FX	opr	-	operate

Table 2: The 16 one byte instructions of the T800

that the data length has been doubled by using one prefix function. By placing a number of these prefix commands one after the other, the data in principle can be extended to any size. Here is a simple example of this prefix command :

```
                                 Program      Hex.     Memonic assembler

1. direct command                 x := 5       45      ldc 5 -- load constant 5
   ( 4-bit operand)                            DX      stl x -- store to x


2. with a prefix function        x := 35       23      pfx 3 -- prefix load 3
   ( 8-bit operand)                            45      ldc 5 -- load constant 5
                                                             -- prefix was const.
                                               DX      stl x -- store constant to x
```

In the first example (direct command) the constant 5 is assigned to the variable x. The transputer assembler program which does this task can be seen on the right side. At first the constant 5 is loaded into the evaluation stack. Then the content of this stack register is stored into the memory location of the variable x. In the second example with the prefix command the constant 35 is assigned to the variable x. Here the transputer assembler program starts

with a prefix command which contains the bit pattern of the constant 3. Up to now it is not clear to the transputer whether the bit pattern is part of a long command or of a long constant with the numeral 3. The next command to load the constant 5 into the evaluation stack identifies the previous bit pattern as the first numeral of the constant 35. Finally this constant is stored into the memory location of the variable x.
The last of the sixteen possible function codes is called operate. It causes the following operand to be interpreted as an instruction code. While the data block consists of four bits, sixteen additional instructions can be defined without using the prefix function. But as mentioned above it is possible to extend the data code up to any arbitrary length. With the combination of the operate and prefix operation it is then possible to extend the number of available instructions indefinitely.
In a CISC - Processor the assembler commands are also ordered into groups. But while the commands of a RISC - Processor are ordered into groups according to how often they are used, the commands of a CISC - Processor are ordered into logical groups (arithmetical commands, copy commands etc.). While the advantage of the RISC processors is the enhanced execution speed, the advantage of the ordering into logical groups becomes obvious when a person wants to program in assembler language. Due to the ordered system the assembler commands of a CISC - Processor are more easy to learn.
This conflict can be solved with an efficient high level language which supports the RISC architecture; OCCAM 2 is such a language. It supports all the fast possibilities of the transputer architecture. Because transputers and OCCAM had been developed together, no other language can compete on the transputer with the performance of OCCAM. This is especially the case with communication tasks. OCCAM is based on the theories of C.A.R. Hoare, who had developed a concept for inter process communication [Hoare85].
Nowadays several other languages like C, Pascal and Fortran have also been developed for transputers. To describe the possibilities of these languages on a transputer, the idea of parallel processes has to be presented in some more detail.
A sequential process is normally called a program. It is a list of statements which are written in a definite order and which will be executed one after the other. In OCCAM 2, where the hierachy of the statements is defined by indentation, such a program would look like this :

```
SEQ
  statement.1
  statement.2
  statement.3
      .
      .
      .
  statement.n
```

Several of these sequential processes can be executed together in parallel :

```
PAR
  sequential.process.1
```

```
  sequential.process.2
  sequential.process.3
          .
          .
          .
  sequential.process.n
```

While a sequential process is finished when the last statement has been executed, a group of parallel processes is finished when all the sequential processes have been finished. These sequential and parallel processes can be grouped together in any arbitrary order:

```
SEQ
  statement.1
  PAR
    statement.2
    SEQ
      statement.3
      statement.4
    statement.5
  statement.6
```

In the above example the commands will be executed in the following order: First the statement.1 will be started. When the statement.1 is finished three processes are started and executed in parallel. The first process consists only of the statement.2, the second process consists of the statement 3 and 4 which will be executed one after the other, and the third process consists of statement.5. When all these three processes have been finished the last process, statement.6, can be executed.
It is also possible to give tasks different priorities. This has the effect that tasks which have low priority are only handled if processing of the tasks with higher priority cannot be continued at the moment. This feature is a useful tool for process optimisation.
As shown above several parallel processes can be created with OCCAM 2. In figure 4 each bubble stands for a separate process, which means that in this example five processes are running in parallel. What happens when one process needs some data from another process ? For this purpose a communication system between the processes is needed. In OCCAM the communication system of C.A.R. Hoare [Hoare85] is used. This interprocess communication is done via so called channels. A channel is a unidirectional point to point connection to exchange data between processes. In the example of figure 5 the process A sends the value of the variable a over the channel which is called 'connection to the process B'. In the process B the data is received and stored into the variable c. The synchronisation between processes is also achieved by this process communication. This means that when process A comes to the statement `'connection ! a'` it will wait until the process B comes to the statement `'connection ? c'`. This is also true if the process B comes to the statement `'connection ? c'` first. Then process B will wait until process A comes to the statement `'connection ! a'`. The situation when one process is forever waiting for another process is called a deadlock. Parallel processes have to be carefully designed to avoid these situations. In [Ari82] several problems are described which may cause deadlocks. How the processes of

Figure 4: Five processes without communication

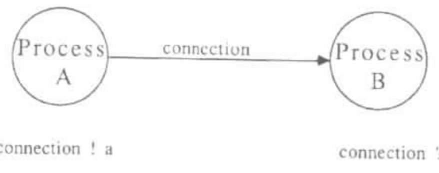
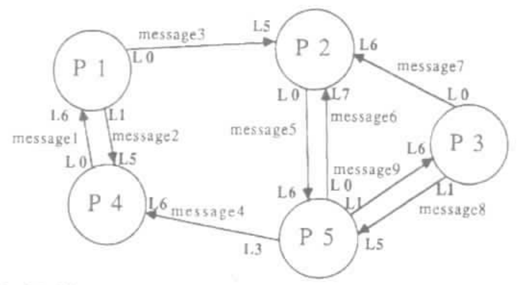


Figure 5: A channel sends data from one process to another



L0, L1, L2, L3 : outgoing links
L4, L5, L6, L7 : incoming links

message1, ... , message9 : logical channel names

Figure 6: Several processes are communicating with channels

figure 4 may communicate over channels can be seen in the example shown in figure 6.
So far the simple structure of OCCAM programs has been shown. What about the other languages which are available for transputers ? For these languages one has to distinguish between the ordinary versions (C, Pascal and F77) without parallel extensions and their enhanced derivates (Parallel C, Parallel F77). Using the ordinary languages it is only possible to write a sequence of sequential statements and combine them in a procedure. This procedure can then be called by an OCCAM program which builds the 'harness' for the procedure. This is quite different when an enhanced language like Parallel C or Parallel Fortran 77 is used. Additional commands like PAR, SEQ, send and receive over a channel have been integrated into the structure of these languages. These commands offer almost the same parallel processing possibilities as OCCAM, with the benefits of C's combined variable types (structures) and pointers still can be used. However there are still strong reasons for using OCCAM on the transputers.
The first reason is that no other language on transputers can compete with respect to the execution speed and compactness of the executable code. But while the Fortran programs are only able to obtain 50 % of the speed of OCCAM, the best C compilers achieve 90 %. This time measurement is valid for sequential processes without communication. OCCAM is the only language which can directly access the hardware ports of links. All other languages need an additional software interface written in OCCAM for their communications over the links. The delay which is introduced by this interface can only be avoided if OCCAM everywhere is used.
In conclusion OCCAM should be used when maximum computing speed or a large amount of inter process communications is required.

1.3 Communication Between Processes

As mentioned above a transputer has four bidirectional serial communication units. These communication units are called links. How can the channels (of process-process communica-

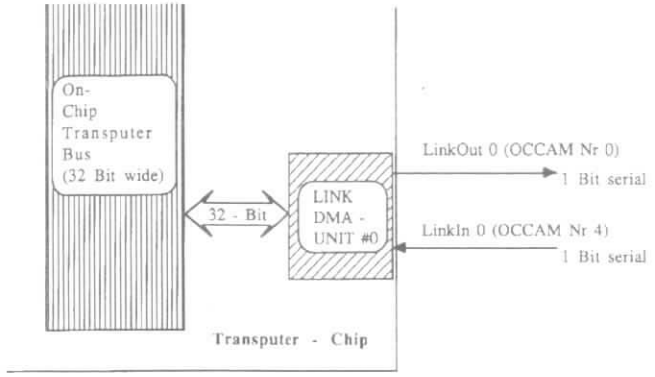


Figure 7: One Link DMA Unit

tion) be associated with the links and how can a process be placed on a specified transputer? Channels are logical connections between processes for the unidirectional exchange of data or information. Because a link is a bidirectional serial connection, two channels can be placed on one link. Figure 7 shows a schematic overview of one link DMA Unit. Therefore it is understandable that the links are serial connections. If a parallel system had been chosen much more pins for the links would have been required. In figure 8 the four outgoing connections are associated with the numbers 0 - 3 while the incoming connnections refer to the numbers 4 - 7. These numbers are the OCCAM addresses of the 32 bit wide ports of the links at the internal bus. An example is shown in figure 6.
In order to explain the link procedure first the whole system with five concurrently running processes will be developed on one transputer. The basic structure of the OCCAM code (without some formal surroundings) will look like this :

```
CHAN OF ANY message1, message2, message3,
            message4, message5, message6,
            message7, message8, message9 :
PAR
  process.1(message1, message2 ,message3)
  process.2(message3, message5 ,message6, message7)
  process.3(message7, message8 ,message9)
  process.4(message1, message2 ,message4)
  process.5(message4, message5 ,message6, message8, message9)
```

At the top nine channels (message1, ...,message9) are defined. Then the five processes are started to work in parallel. The processes are written as procedures with the channels as arguments. This makes it easier to distribute the processes later onto several transputers. It

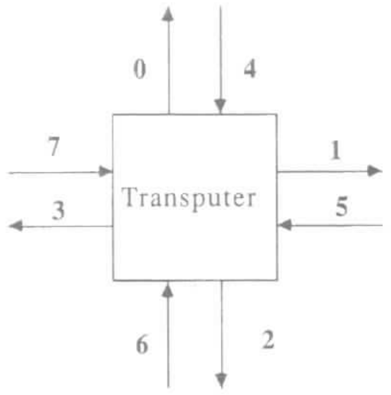


Figure 8: A transputer with its four bidirectional links

cannot yet be seen that the channel 'message1' at process.1 will be the entry point of the data from the channel 'message.1' of process.4. However the list clearly shows which processes are communicating with each other.
When the program runs without any deadlocks on a single transputer the performance of the system can be enhanced by placing the processes on several transputers. Two restrictions limit the number of transputers on which the processes can be placed: firstly, no single sequential process can be distributed over several transputers. While this is a restriction, which in principle holds for all distributed processor systems, the other restriction arises from the hardware possibilities of transputers. Today a transputer chip offers only four bidirectrional links. That means that from the processes which are placed on each transputer, only four incoming and four outgoing channels to the processes on other transputers can be created. The example shown can be distributed onto up to five different transputers.
The following lines show how this will be done in OCCAM2 which is the actual version of OCCAM. It is assumed that the links of the five transputers are physically connected as in figure 6.

```
CHAN OF ANY message1, message2, message3,
            message4, message5, message6,
            message7, message8, message9 :
PLACED PAR
  PROCESSOR 1 T8
    PLACE message1 AT 6
    PLACE message2 AT 1
    PLACE message3 AT 0
    process.1(message1, message2 ,message3)
  PROCESSOR 2 T2
    PLACE message3 AT 5
    PLACE message5 AT 0
    PLACE message6 AT 7
    PLACE message7 AT 6
    process.2(message3, message5 ,message6, message7)
  PROCESSOR 3 T4
    PLACE message7 AT 0
    PLACE message8 AT 1
    PLACE message9 AT 6
    process.3(message7, message8 ,message9)
  PROCESSOR 4 T4
    PLACE message1 AT 0
    PLACE message2 AT 5
    PLACE message4 AT 6
    process.4(message1, message2 ,message4)
  PROCESSOR 5 T8
    PLACE message4 AT 3
    PLACE message5 AT 6
    PLACE message6 AT 0
    PLACE message8 AT 5
    PLACE message9 AT 1
    process.5(message4, message5 ,message6, message8, message9)
```

The program looks very similar to the previous single transputer version. The PAR statement is exchanged by a PLACED PAR and in front of each process the following line has been added :

```
PROCESSOR processor_number processor_type
```

Any arbitrary integer can be used for the processor_ number. It can be used as a processor identification by the user. The processor_ type (T8 = T800, T4 = T414 or T425, T2 = T212 or T222) is necessary because the various transputer-types are compatible only at the source code level. So the compiler has to know for which transputer the executable code has to be generated. The line

```
PLACE channel AT link_nr
```

connects the logical channels to the physical links.
In the example considered it also would be possible to distribute the processes only over 2, 3, or 4 transputers.

1.4 Additional Hardware for Transputers

Transputer links offer a very efficient means for data transmission between processors. Additional hardware devices have been developed for an even more efficient use of these links.

Two of these devices are the Crossbar Switch (C004) and the link to port converter (C012). While the crossbar switch provides a better connectivity between several transputers the link to port converter allows a unique interface to other processors.

1.4.1 Crossbar Switch (C004)

A crossbar switch for transputers is a device which allows the connection between any pair of transputer links which are connected to it. A single C004 crossbar, which has 32 bidirectional links, can set up 16 bidirectional point-to-point connections. Physically the C004 is an 84-pin integrated circuit which can be controlled by a transputer over a control-link. As a further advantage the output signals are regenerated on the chip.
However, a delay of 175 ns to the signal transmission is introduced by the C004 crossbar switch. As it can be seen in table 4 (refer to chapter 2.1) this leads to a reduced data rate which corresponds to an additional cable length of 20 m.
If larger crossbars are required, combinations of the C004 chip can be used. If a 32 $\times$32 crossbar switch is needed one can combine two C004 chips. In this configuration each bidirectional link will use a single line from each of the two chips. By combining several crossbar switches it is also possible to extend the number of links per crossbar. A bidirectional 48 $\times$48 crossbar switch can be constructed with three pairs of C004 chips. In principle it is possible to construct with C004s crossbar switches of arbitrary size. But the delay introduced leads here to a pratical limit (see [IMSSP89]).

1.4.2 Link to Port Converter (C012)

The C012 is an integrated circuit which converts a transputer link to an eight-bit wide port of a conventional microprocessor and the other way round. With the C012 it is possible to communicate over a data bus with a link. This chip is very often used for interfaces between transputers and conventional processors. Another application for the C012 is the general broadcasting system which is described in chapter 3.2.

Link rate :	unidirectional one channel of a link used	bidirectional both channels of a link used
[Mbit/s]	[Kbytes/s]	[Kbytes/s]
5	450	670
10	910	1250
20	1740	2350

Table 3: Uni- and bidirectional linkspeeds (fast link protocol)

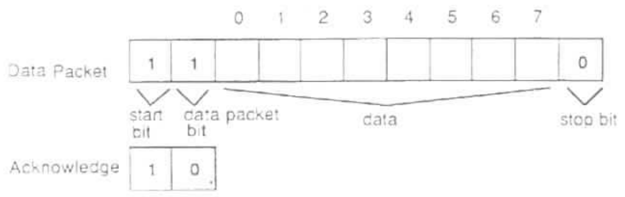


Figure 9: Protocol structure of the link messages

2 Performance of Transputers

2.1 The Links

Transputer links are serial communication lines which are intended to be used for inter transputer communications. On a serial communication line the data and the control signals are sent over only one electrical signal path, whereas parallel communication systems use several separate lines for control signals and data. A parallel system is obviously faster but it also requires more space (pins on chips, board space etc.). Therefore it is technically more difficult to build an integrated system with parallel lines on a single chip such as for a transputer.
For a further saving of space the transputer links are bidirectional. That means that the transputer can send and receive data in parallel over the same link. In this case the data rate in both directions together is less than twice the unidirectional transfer rate. This reduction occurs because the ' acknowledge ' signal and the data bits have to share the lines (refer to 3).
Each link consists of a LinkIn and a LinkOut connection. The LinkOut connection from one transputer has to be connected with the LinkIn connection of another transputer and vice versa. After one data byte is sent over the LinkOut pin the sending transputer waits at the LinkIn pin for an acknowledge of the data. The data formats are shown in figure 9. On these lines there are two electical states defined as High(H), for a logical 1, and Low(L), for a logical 0. As you can see in the figure, two startbits (set to high) are sent before each data byte and a stopbit (set to low) is sent after each data byte. Then the acknowledge is

received. As mentioned before in table 1 of chapter 1, the older transputers have a slower link protocol then the newer ones. In the old link protocol the acknowledge is sent once the whole dataword (Byte) has been received. Transputers with the new link protocol already send the acknowledge bits when it has received the first bit of the data word. Due to this feature the new link protocol is a factor of two faster over short distances than the older protocol. It is also understandable that both link protocols are fully compatible with each other. All new transputers use the fast link protocol. All the measurements given in this note (unless stated otherwise) have been performed on transputers with the fast link protocol.
The link speed of 20 Mbits/s is the fastest linkspeed available at present; it is the ideal speed for short distances with good electrical conditions. However linkspeeds of 5, 10 and 20 Mbits/s can be chosen for each transfer. Obviously the data rate is directly correlated with the link speed and at first glance it is difficult to understand why a transputer should not communicate with the full link speed. But a higher link speed also requires better electrical conditions. For instance the electrical conditions become worse when the cable length increases. These problems are described in some detail in [Rygol87]. There is another reason why long cables are reducing the data rate. At longer distances the data words need a longer time to arrive at the receiving transputer. Due to this the acknowledge bits will be sent later and will also arrive later at the transmitting transputer. In summary the data throughput decreases with increasing cable length. This is shown in table 4. These data rates are calculated approximately with the formula :

$$TransferRate = \frac{1}{0.55 + CableLenght\,[m] * 0.01} \left[\frac{Mbyte}{s}\right]$$

Our measurements agree well with this formula which can be derived from the medium speed of an electrical signal and the protocol structure of transputer links.

2.2 Copying From Memory to Memory

The performance of a copying process depends strongly on the use of coding techniques which allow the compiler to use the most efficient machine instructions. If an array of variables is copied with a loop around a single assignment (like $a[i] := b[i]$) the data is only copied with 1 Mbyte/s. With some tuning (opening the loop and using abreviations, see also [Atkin87]) one can achieve a copy speed of 2 Mbyte/s. Like in other high level languages the main limitation comes here from the range checking routines of OCCAM. This is for the case that the copying is done inside the external memory (100 ns access time). The faster internal memory (see chapter 1.1) is only 10% faster. This behaviour changes dramatically, when the special blockcopy command of OCCAM is used. The statement

$$[array1\ FROM\ start1\ FOR\ end1] := [array2\ FROM\ start2\ FOR\ end2]$$

copies data with 10 MByte/s in the external memory. The above OCCAM statement is directly translated into the blockcopy assembler command of the transputer which operates at the speed of the memory. A restriction of the blockcopy command is that the two arrays should not overlap. In this case it is still faster to copy the data at first into a third array and then copy it to the destination array with the block copy command.

2.3 Memory and Link Actions in Parallel

Several test measurements have been done for the case that the CPU copies data from one place in the memory to the other while the link units perform DMA actions in parallel. At first the results were in contradiction with the assumption that the link actions are independent of the other actions of the transputer. Copying of arrays in the memory was used because it is obvious that this action puts the most load on the memory. The memory is the common bottleneck. All data which go to or come from the link have to be buffered in the memory. For the tests the T800 was used because of its fast link protocol. At the test it was running at 20 MHz and the memory had an access time of 100 ns. The linkspeed was set to 20 Mbit/s and the linklength was negligible (< 0.3 m). Table 5 summarizes the results of these measurements. With increasing number of links used not only the memory copy rate but also the linkspeed decreases. This test was done with all processes running at low priority. The transputer allows only two classes of processes which can be selected by the user. The processes can either run on high or on low priority. In the next test the link actions were given high priority and the results for the link performance turned out to be much better. As shown by table 6 the memory copy rate decreases to 6.45 Mbyte/s, whereas all four links are working with the full unidirectional speed. In a network of processes it is always true that the communication tasks have to have priority. Otherwise most of the processor time is lost by waiting for data (see also [Atkin87]).

2.4 Conclusion

A few years ago transputers established a new generation of microprocessors, based on completly new concepts.
A popular argument against transputers is, that today even faster standard processors are on the market. Because of the rapid development in the field, no processor is the fastest on the market for long. Benchmark tests have the problem that each processor is given the tests it is best suited for. The available transputer families are at least a factor two faster than the fastest processors of established families like MC 680X0 and Intel 80X86. Faster transputers with clock frequencies up to 35 MHz and full pin compatibility have been announced.
A transputer is able to compete favorably with even faster processors because of its networking capabilities. Other microprocessor manufacturers are planning processors with separate DMA channels. This would allow to place the DMA controllers on separate chips so that there would be no restriction on the number of links. However, there are several arguments for the transputer link solution: The integration of everything on one chip allows very compact and also very cheap systems. Furthermore, the integration also sets a standard for the protocols which reduces the problems with interfaces to a minimum.
The most important restriction of transputers at present is the number of links per chip which is four. This number in future will increased to probably 8 links. In order to achieve the equivalent increase in data throughput the interaction system between the processor and the links will be changed. At the moment all data are transferred via the bottleneck of the internal bus. With four links the internal bus and the memory are just able to handle all activities without any perturbation. But this will be different when more links want to access the internal bus. Therefore future transputer chips will include a completely separate part for the link handling. This link control part will also be able to connect two links with each

other without any intervention of the transputer CPU.
Another cumbersome part of the transputer is the absence of several features in OCCAM which are convenient in other high level languages like Pascal, Modula 2 or C. The missing features are specially structured variable types and pointers. A dynamic memory allocation (like malloc in C) is missing too. Hopefully there will soon be extensions of OCCAM which provide these features.
In summary the transputer concept appears to be the most reliable processor network system available. Transputers are relatively easy to program and are supported by reliable development tools including a post mortem debugger.

Delay : ns	Cable length m	Transfer rate Mbyte/s
0 - 50	0 - 5	> 1.53
50 -100	5 - 10	1.42
100 - 150	10 - 15	1.32
150 - 200	15 - 20	1.25
200 - 250	20 - 25	1.17
250 - 300	25 - 30	1.11
300 - 350	30 - 35	1.04
350 - 400	35 - 40	1.00
400 - 450	40 - 45	0.95
450 - 500	45 - 50	0.90

Table 4: The transfer rate in relation to the cable length for a 20 Mbit/s link rate

number of links used (unidirectional) :	4	3	2	1	0	1
memory copy speed [Mbyte/s] :	7.88	8.29	8.74	9.25	9.9	-
average link speed [Mbyte/s] :	0.991	1.04	1.10	1.16	-	1.77

Table 5: Linkspeed and memory copying with all processes running at low priority

number of links used (unidirectional) :	4	3	2	1	0	1
memory copy speed [Mbyte/s] :	6.45	7.30	8.17	9.05	9.9	-
average link speed [Mbyte/s] :	1.70	1.72	1.74	1.76	-	1.77

Table 6: Linkspeed and memory copying with the link process running at high priority

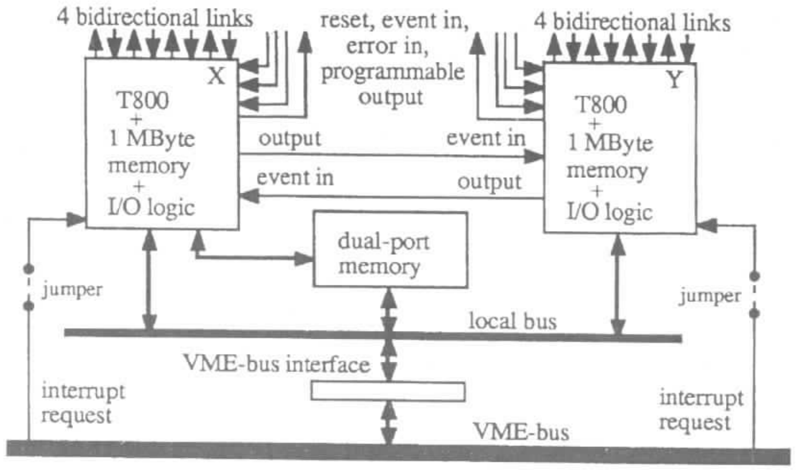


Figure 10: Block diagram of the ZEUS 2TP-module (preseries-modules)

3 Transputer Hardware at ZEUS

3.1 The 2-Transputer VME Board

Transputers will play an important part in the ZEUS data aquisition system. For this purpose at NIKHEF-H a VME board with two INMOS transputers has been developed. Despite the fact that the 2TP-Board was originally planned for the GSLT [5] it has been designed as a general purpose VME-transputer board. The 2TP-Board documentation is well provided by NIKHEF and can be found in :

- [Waard87] Hardware Documentation
- [Bot87] & [Bot88a] General Descriptions
- [Bot88b] Test Software
- [Bot89] Libraries

In view of the detailed documentation only a brief overview of the module is presented here. The structure of the 2TP-Board is shown in figure 10. Each of the two transputers (called X and Y) on the board has its own private memory (1 or 4 MByte), its own four bidirectional link connections, and some additional I/O connections (reset, event in etc.). [6] All actions on

[5] Global Second Level Trigger

[6] The link and the I/O connections are physically connected to the a and c row of the P2 connector from the VME-bus. An additional adapter which has to be placed on the back of the VME backplane amplifies the signals at the link cables for connections with other transputers.

the hardware described so far can be done independently. But it is obvious that this will be different when the transputers are communicating with the VME-bus since on the VME-bus at a time only one Master is allowed. When one transputer wants to access the VME-bus it has to do this in the following way. First the transputer has to make an access on the local bus. This local bus is connected to both transputer modules, a dual-ported memory (DPM), and a VME-bus interface. Once the transputer has established this connection, it is able to access the VME-bus via the VME-bus interface. As mentioned before there is also a dual ported memory connected to the local bus. This enables both transputers to access the dual ported memory over the local bus. The name dual ported memory comes from the fact that this memory is connected with a further port to the X-transputer module. The additional bus between the X-transputer module and the dual-ported memory increases the power of the 2TP-Board dramatically. With this connection it is possible that transputer X accesses the DPM while an additional access to the DPM from the Y-transputer module or the VME-bus is handled simultaneously. By accessing the DPM from both transputer modules, the DPM can be used as a connection between these transputers. This possibility is also supported by the two event lines between the two transputers. By using the event lines the protocol for a transputer-to-transputer connection with the DPM becomes very similar to that of a link connection between transputers, like a fast 5th link.
All the connections between the 2TP-Board and the outside world are done through the P1 and P2 connectors of the VME-bus. While the VME-bus uses the P1 connector (96 pins) and the row b from the connector P2 (32 pins), the rows a and c from P2 (32 pins each) are used by the special transputer lines (links, reset etc.).
At the moment the VME-bus interface consists of several chips. To reduce the required board space and access time a gate array (ASIC) is under design. This gate array will contain the whole transputer VME-bus interface on a single chip. With this ASIC the performance of the 2TP-Board will be enhanced. Also, several additional features like a real triple ported memory (TPM) instead of a dual ported memory and more status registers are added.
With these features and the fast VME-bus interface the 2TP-Board provides an ideal hardware environment for a VME-bus based world.

3.2 The Reset and Broadcast System

For the control of the ZEUS Global Second Level Trigger (GSLT) a multi purpose module has been designed at NIKHEF. A block diagram of this so called "Controller and Switch Box" can be seen in figure 11. The CSB is controlled by a T222 transputer which can be connected to the outside world with two of its links. These links can be used to combine several of the CSBs to a chain. This allows the configuration of a controller for linking an arbitrary number of transputers.
The CSB can be divided into the following logical sections :

Switchboxes : The remaining two links of the T222 transputer are connected to two C004-Crossbar Switches. These C004 chips are described in chapter 1.4.1.

Link Adaptors : In contrast to other transputers the T222 (and T212) does not have a multiplexed address and data bus. At the T222 there is a separate

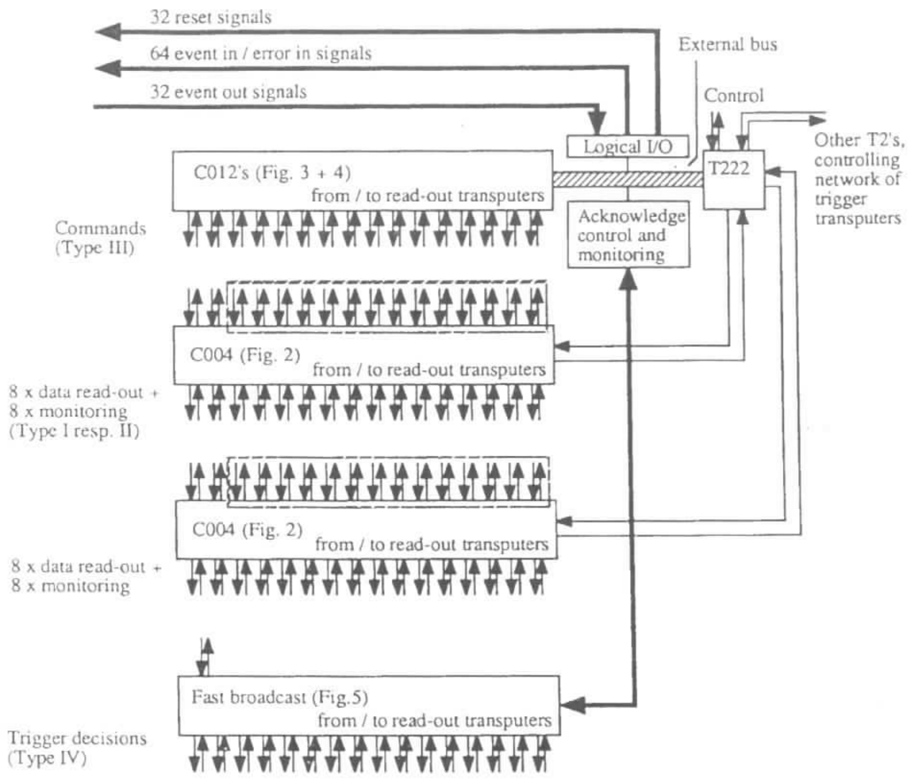

32 reset signals
64 event in / error in signals
32 event out signals
External bus
Control
Logical I/O
T222
Other T2's, controlling network of trigger transputers
C012's (Fig. 3 + 4)
from / to read-out transputers
Acknowledge control and monitoring
Commands (Type III)
C004 (Fig. 2)
from / to read-out transputers
8 x data read-out + 8 x monitoring (Type I resp. II)
C004 (Fig. 2)
from / to read-out transputers
8 x data read-out + 8 x monitoring
Fast broadcast (Fig.5)
from / to read-out transputers
Trigger decisions (Type IV)

Figure 11: Block diagram of the Controller and Switch Box (CSB)

16bit address and data bus available. This makes it very easy to connect the C012 link adapters to the transputer bus. The CSB provides 16 of these port link adapters connected to the bus of the T222. With these adapters and additional control registers a direct communication between the T222 and 16 transputers is possible. This communication is not as fast as a direct link-to-link connection, but it is a very nice possibility for the exchange of control information between one master and several slave transputers.

Logical I/O Lines : The reset, event_ in and event_ out lines belong to this group. These lines allow one single CSB to perform control and reset functions for up to 32 transputers. In the following one possible solution for that will be shown.
The CSB sends an event signal to a specific transputer. In this transputer this event invokes a selftest process. If this selftest finishes without an error the tested transputer sends back an event signal to the CSB. This controlling action is done in parallel with the normal activities of the transputers. With that simple protocol the CSB can find out which transputers of the system are still alive. When a hardware error occurs in the system the CSB will know which transputer has failed. In this case the CSB can reset the transputer network. After all transputers have been reset the system can be booted and restarted.

Fast Broadcast : Sometimes it is quite useful to have the possibility for a fast broadcast. A broadcast is the transmission of data from a single source to several receivers. An example for a broadcasted data package may be a global status information. In the fast broadcast module of the CSB this is arranged by some dedicated hardware.
In principle one transmission line is fanned out to all receivers and the logical anded acknowledge from all the transputers goes back to the transmitter. Due to this hardware connection only an unidirectional data traffic is possible on this system.

4 Acknowledgements

I would like to thank all the people who supported my work. Especially all the people from the transputer group of NIKHEF-H. These people developed the hardware which is decribed in chapter 3. Further on a lot of discussions provided me with the knowledge and information for my work.
Further on I would like to thank W.Vogel who strongly supported me in any aspect of my work. Last not least I would like to thank E.Lohrmann and G.Wolf for carefully reading the manuscript and further suggestions to make this report more readable.

References

[Ari82] *Principles of Concurrent Programming*
M. Ben-Ari, Prentice-Hall International, 1982

[Atkin87] *Performance Maximisation (Technical note 17)*
Phil Atkin, Inmos Ltd, 1987

[Bot87] *A two-transputer VME module for the ZEUS experiment*
H.Boterenbrood, S.C. Goble, G.N.M. Kieft, J.C. Vermeulen, A.J. de Waard, L.W.Wiggers, NIKHEF-H, 1987

[Bot88a] *The ZEUS Two-Transputer VME-Module, an Overview*
H.Boterenbrood, NIKHEF-H, 1988

[Bot88b] *Testsoftware for the 2TP-module*
H. Boterenbrood NIKHEF-H, 1988

[Bot89] *2TPlib, a library of 2TP-module cycles*
H. Boterenbrood NIKHEF-H, 1989

[DOB89] *The Use and Possible Abuse of Transputer Links*
C.Bizeau, A.Bogaerts, R.W.Dobinson, D.R.N.Jeffery, W.Lu, C.Parkman and Y.Perrin CERN, Geneva, 3 May 1989

[Hoare85] *Communicating Sequential Processes*
C.A.R. Hoare, Prentice-Hall International, 1985

[IMS84] *Occam Programming Language Manual*
Prentice-Hall Inc., 1984

[IMSAS89] *Transputer Application Notebook : Architecture and Software*
Inmos Databook Series, Inmos Ltd, 1989

[IMSLI86] *Connecting Inmos Links*
Inmos Application Note, Inmos Ltd, 1986

[IMSOC85] *The implementation of Occam on the IMS T414*
Inmos Ltd, internal, 1985

[IMSREF86] *Transputer Reference Manual* Inmos Ltd, 1986

[IMSSP89] *Transputer Application Notebook : Systems and Performance*
(includes [Atkin87] and [Rygol87])
Inmos Databook Series, Inmos Ltd, 1989

[OCCAM88] *Occam 2 Reference Manual*
Prentice Hall Inc., 1988

[Packer88] *Simpler real-time programming with the transputer (Technical note 51)*
Jamie Packer, Inmos Ltd, 1988

[Richter85] *Betriebssysteme*
Lutz Richter, B.G. Teubner Stuttgart, 1985

[Rygol87] *Connecting Inmos Links (Technical note 18)*
Michael Rygol and Trevor Watson, Inmos Ltd, 1987

[Stein88] *OCCAM2 Die Programmiersprache für parallele Verarbeitung*
Ralf Steinmetz, Dr. Alfred Hüthig Verlag Heidelberg, 1988

[T2IMS86] *T2 T4 transputer instruction set*
Inmos Ltd. internal, 1986

[T4IMS85] *IMS T414 Reference Manual*
Inmos Ltd, 1985

[TDS86] *Transputer Development System*
Beta Release Document, Inmos Ltd, 1986

[Verm88] *Transputers Practice and Experience I*
J.C. Vermeulen, NIKHEF-H, 1988

[Waard87] *Design of the ZEUS 2 Transputer VME Card*
A.J. De Waard, NIKHEF-H, 1987