TRANSPUTER EDUCATION KIT

User Guide

Installation

Schematics

Glossary

Theory of Operation

CSA Computer System Architects

UserGuide

Contents

1 Introduction

"What we are witnessing today is the last hurrah of serial processing."
—Joel Birnbaum
Hewlett-Packard

1.1 Forward

Congratulations. With your purchase of a Transputer Educational Kit from Computer System Architects, you have invested in the most promising computer technology available. This exciting technology is multiprocessing. Transputers are revolutionary parallel microprocessors that make multiprocessing viable and affordable today.

I frequently interface with scientists and engineers throughout North America as a marketing manager for SGS-Thompson-Inmos, the company that manufactures the transputer and teamed with CSA to build your kit. Many of these technical people are achieving scientific breakthroughs and are building futuristic applications using the same development tools you now own.

The importance of what this kit will teach you cannot be overstated. The powerful concepts and tools are unique, oftentimes contrary to mainstream methodologies promoted by the technical and academic communities.

When transputer technology was being born in the early 1980s, it was championed by visionary thinkers and imaginative programmers looking to extend the boundaries of computer science. But conventionalists regarded the transputer as a whimsical idea that would amount to nothing more than an interesting toy. One eminent scientist even postulated a law of diminishing performance to disprove benchmarks obtained by large transputer systems.

Today the transputer is broadly regarded as an innovative machine at the forefront of computer technology. It delivers unmatched performance for many applications and ranked fourth in world sales of 32-bit microprocessors last year. Major Fortune 500 companies and large military contractors are using transputers to build next generation systems, and transputer applications are playing a role in America's space program. Let me briefly examine this dramatic turn-around and relate it to what you will be learning.

We live in an age of tremendous discovery. If you are in your forties, half of the world's knowledge has been produced since you left school. This explosive growth of science is being made possible by computers that extend our ability to manage information and solve problems which stagger the human mind.

Applications for computers have increased far beyond the basic "number crunching" they were invented for. Besides calculating mathematical expressions, computers also process, analyze, control, synchronize, and even characterize data.

Some computer systems convert signal and image information into speech and vision. Graphics computers transform mass numbers, humanly indigestible in their raw form, into three-dimensional maps of the human body, subsurface geological features, protein molecules, and objects in outer space.

Other computers are used as smart, responsive, "real-time controllers" for focusing cameras, robot motion, anti-lock breaking, nuclear reactors, and weapon guidance. Perhaps most amazing are neurocomputers which implement neural network paradigms to produce machine learning or "artificial intelligence."

Scientific investigation and computer technology have become almost inseparable. Even though the performance of today's computers is impressive, the quest for knowledge continues to demand more computational power. Major problems on all frontiers of science are so complex they overwhelm even the most powerful supercomputers, capable of executing 1.5 billion calculations per second.

Such computers achieve their speed with advanced process technologies that reduce the size of chip components and enhance their conductive properties. Semiconductor devices holding two million components are now in production, but designers are running into limitations imposed by the laws of physics. These limitations are dictated by the wavelength of light used to etch circuit patterns on wafers, heat dissipated from current resistance in conductors, and leakage of electrons as insulation layers become increasingly thin.

Some exotic solutions are being explored to overcome these speed barriers. One is the creation of photonic circuits that use flashes of light, instead of electric current, to transmit digital information through strands of glass called optical fibers. These fibers dissipate no heat and require no electrical insulation. The most radical concept for a photonic circuit is the "biochip," a three-dimensional device made of organic (carbon based) molecules. These tiny molecular switches could be packed together in far greater densities than semiconductor components.

These prospects are exciting but they will not become commercial realities for some time, and the price of process technology in terms of unit and system costs is already soaring. Supercomputers utilizing Gallium Arsenide or supercooled CMOS circuits require very fast memories and cost millions of dollars to own and operate.

Increasingly, multiprocessing is being adopted around the world as the sensible alternative. Instead of using process technology to achieve raw speed, problems are solved more quickly by using dozens or even thousands of parallel computers networked together. This strategy is not unlike brain cells, called neurons, which are tied together in massive networks that allow them to quickly recognize patterns like smells or process information for split second reasoning.

Nature has elected to process information concurrently because the universe is enormously parallel. Any system that can be viewed as a collection of simultaneous (parallel) parts or events is said to possess concurrency. Virtually all systems in astronomy, genetics, physics,

geology, biology, and chemistry exhibit concurrency because natural phenomena rarely occur in a nice serial fashion - the way most programmers develop their code.

Computers used for speech regulation, neural networking, jet engine control, medical diagnosis, and airport scheduling all describe pieces of the real world. To accurately model nuclear reactions, global weather, biosystems, experimental drugs, formation of new stars, and air bag expansion in a car crash, scientists and engineers visually simulate the net effect of many simultaneous forces, events, and physical laws. Although some computers may be faster at performing rote calculations, the transputer is specifically designed to handle this complexity.

Conventional computers and programs are "sequential," meaning one instruction or task must be completed before the next can begin. Parts of a problem are always executed in serial order, even if they actually take place at the same time. When sequential programs are used to describe real phenomena, the impact of real-time interactions is lost and only simplistic models are possible.

Conversely, transputers are "parallel" machines. They achieve their speed by executing the composite parts of a problem together as they occur. A single transputer handles different parts by "juggling" rapidly between them, or a group of transputers simultaneously on parts distributed among them.

The method of programming is essentially the same for either approach. In the second case, transputers are combined as building blocks in multiprocessor networks to obtain nearly linear performance enhancement. In other words, a network of N transputers will typically execute code N times faster than a single transputer.

Multiprocessing is as much a conceptual advancement as it is a technological advancement. Although the transputer's architecture is being studied by computer manufacturers everywhere, the real trick to multiprocessing is learning how to write code that exploits the parallelism of problems.

The most far-reaching breakthrough in this area is the Occam programming language. Parallel programming is not trivial to master, but Occam provides a framework for developing programs which express parallelism explicitly, in an understandable way. Occam can also be proven mathematically, making it attractive for theoretical research and secure applications.

Because its architecture is designed to execute the Occam model of concurrency, the transputer is sometimes regarded as a hardware implementation of the language. This unprecedented level of integration between software and hardware is part of the reason behind the transputer's impressive performance. Other parallel programming languages exist, but most are written and supported only by PhDs and none have been transformed into an innovative computer architecture.

Occam is being successfully taught to undergraduate students around the world, and some universities have made it part of their required curriculum. National user groups have formed a worldwide community of students, faculty, and industry professionals who use Occam because it makes the description of concurrent systems straightforward and comprehensible.

Transputers, like ordinary microprocessors, can also be programmed in standard high-level languages. Your kit includes comprehensive tools and documentation for both Occam and C.

A company called Logical Systems has built a special "Parallel C" compiler that embeds Occam's powerful constructs within C code to ease the transition from serial to parallel programming. This useful compiler also makes the porting of existing C applications onto transputer systems possible. An increasing number of industry applications are combining Occam with more familiar languages like C in pragmatic ways.

CSA recommends using your single transputer board to become familiar with this kit's extensive software development tools. Look at the exercises in your workbook and try to start writing and debugging code. It is important to approach Occam with an open mind until its use feels natural. First time programmers will sometimes learn quicker than experienced programmers who have an ingrained sequential approach to writing code.

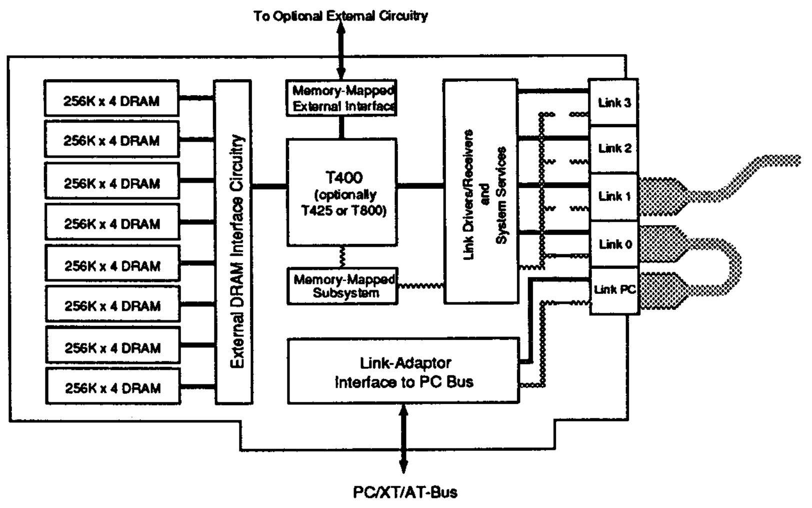
Additional PC add-in boards can be purchased from CSA to construct inexpensive multiprocessor systems for accelerating your code. These boards are easy to interconnect in arbitrary network configurations with plug-in link cables. These networks may be located inside one PC chassis or distributed among separate PCs in a parallel processing laboratory.

With four additional 1 Mbyte boards, you can build a formidable PC multiprocessor for under $1,500 which delivers 50 MILLION INSTRUCTIONS PER SECOND (MIPS). No other technology in the world places this kind of performance within reach of typical consumers. Most 25 MIPS workstations sell for $30,000 and up. A Hypercube machine is several orders of magnitude more expensive than a comparable transputer network, and is an engineering nightmare by comparison. Sizable transputer systems can be created in classrooms or teaching labs if each student purchases their own kit for the course. Some commercial systems in use today have several thousand transputers.

It is our hope that your state-of-the-art transputer kit becomes a source of valuable training and fun. You are now one of many pioneers exploring today's newest, most powerful, and fastest growing computer technology.

Mark Hopkins
Strategic Projects Manager
SGS-Thomson Microelectronics

1.2 Brief Theory of Operation


To Optional External Circuitry
256K x 4 DRAM
256K x 4 DRAM
256K x 4 DRAM
256K x 4 DRAM
256K x 4 DRAM
256K x 4 DRAM
256K x 4 DRAM
256K x 4 DRAM
External DRAM Interface Circuitry
Memory-Mapped External Interface
T400 (optionally T425 or T800)
Memory-Mapped Subsystem
Link Drivers/Receivers and System Services
Link 3
Link 2
Link 1
Link 0
Link PC
Link-Adaptor Interface to PC Bus
PC/XT/AT-Bus

The Transputer Education Kit board consists of three sections: the transputer with its local memory, the PC/Link Interface and the External Interface. The transputer is a RISC-like microprocessor which was designed with multiprocessing in mind. The PC/Link Interface provides data and control communication between the transputer and the PC. The External Interface allows the transputer to control external hardware.

One of the biggest challenges in multiprocessing is communicating data between processors. The designers of the transputer addressed this challenge with what is called a transputer link. A transputer link is a high-speed bidirectional serial communication path for interconnecting transputers and connecting transputers to peripherals or other computers. Link communications are controlled by simple but fast co-processors residing on the same chip as the transputer. These co-processors allow the transputer to overlap computation and link communication, letting the transputer compute while data is sent and received. This results in much higher rates for interprocessor data transfers without stopping computation. Each transputer link can transfer data at over 1 Mbyte of data per second in each direction giving a total data transfer rate of 2.3 Mbytes of data per second. A transputer with two links is capable of transferring 4.7 Mbytes of data per second. With four links this increases to 9.4 Mbytes of data per second.

Another challenge of multiprocessing is in software development. To ease system development, the processor design should allow one program to run on various numbers of processors with little or no software modification. The transputer's hardware was designed to efficiently handle multiple processes running on a single processor. It uses the same communication model (even the same instructions) when multiple processes are running on one processor as when the multiple processes are distributed over multiple physical processors. This means that a parallel program intended to run on many transputers can be developed and debugged on one transputer and then run on single or multiple transputers. Many programs can be designed to use all processors that are available at run-time; from just one to 100 or more, often yielding an almost linear increase in performance.

The transputer also implements a process scheduler directly in the hardware. The low-overhead of this scheduler allows the programmer to subdivide his task into multiple concurrent processes and run one or more of these processes on the same processor without using significant processor power for process switching. In fact, multiple concurrent processes must be running on each transputer in order to take advantage of their ability to overlap communication with computation. Buffer processes are created for each link that either receive internal data from the compute processes and send it out a link, or receive external data from a link and send it to the compute processes. The buffer processes send and receive data while the compute processes compute. The transputer's process scheduler supports processes at two priority levels and time-slices processes operating at low priority.It also manages processes waiting on interrupts from external events or one of the two on-chip timers.

Another added feature of the transputer is that, depending on the model, either 2K or 4K of fast memory is included on the same chip with the processor and link circuitry. Using this memory, programs can be run on the transputer without any off-chip memory. The C Toolset included with the Transputer Education Kit allows you to write and execute parallel C programs using only this on-chip memory. When off-chip memory is provided, this on-chip memory can be used to hold code and variables that are used frequently. Because this on-chip memory is at least three times as fast as off-chip memory, placing often-used code and variables in on-chip memory can significantly increase performance.

Also included on the Transputer Education Kit circuit board is ciruitry that provides a byte-wide interface to external hardware and program control of eight LED's (light emitting diodes). LED's can be installed on the board and then turned on or off under program control as a debugging aid. The byte-wide I/O port gives the ability to control external hardware projects and, when used in conjunction with the signals that control the LED's, this I/O port can control virtually any eight bit microprocessor bus peripheral chip.

The PC/Link Interface portion of the circuit board is used to communicate between the transputer and the host PC. The interface essentially adds a link to the PC so that it can fit into the transputer's scheme of communication. The PC/Link Interface section of the board is completely independent of the transputer and its local memory. The two are connected only when the PC Link is connected to the transputer by external cables or on-board jumpers. The PC Link could even be connected to a transputer on a different board or ignored if desired.

The combination of a transputer processor, PC/Link Interface and the External Interface makes the circuit board provided with your Transputer Education Kit a powerful co-processor that can be used to explore the concepts of parallel processing and real-time control. With additional Kit boards or Kit Add-On-Processor boards (available from CSA) you can explore and take advantage of true parallel processing.

2 Installation Guide

2.1 Getting Started

The first step in installing your Transputer Education Kit is filling out your warranty registration card. During the hardware installation you will be given a number that validates your warranty. Write this number on the warranty registration card and return it to CSA. At this time also check your packing slip with the contents of the box. If you find any discrepancies, please call CSA at 1-800-753-4CSA or (801) 374-2300.

The CSA Transputer Education Kit circuit board will work in any IBM PC compatible system. The software requires at least 512K of memory and a hard drive with at least three megabytes of free disk space. Six megabytes of free disk space are required if you plan on using both C and Occam.

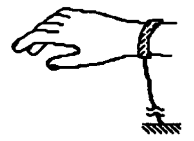
2.2 Hardware Installation

IMPORTANT: Install the circuit board and run all installation tests before changing factory jumper settings or adding external memory.

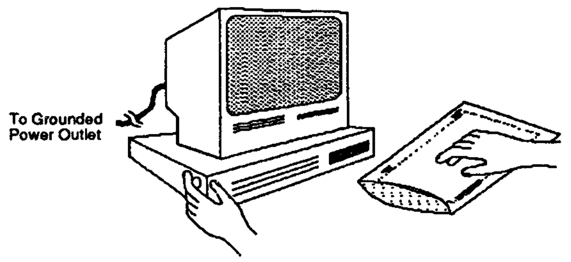
2.2.1 Anti-static Precautions

Several components on the Transputer Education Kit board are static-sensitive and can be damaged through improper handling.

CSA recommends that you not remove the Transputer Education Kit board from its anti-static envelope during pre-installation handling and inspection. Whenever you are handling the board you should use a grounded wrist strap and conductive pad to ensure a static-free environment.


Example of a Grounding Wrist Strap

If proper static-protective equipment is not available, you can reduce the risk of damage to static-sensitive devices by following certain precautions: Before removing the Transputer Education Kit board from its anti-static envelope, touch a grounded surface such as an exposed screw on the back of your computer's chassis (not the monitor) with one hand and the anti-static envelope with your other. Wait at least twenty seconds, then still touching ground, reach inside the envelope and remove the board. Handle the board by its edges and avoid touching the board's components as much as possible.



To Reduce Static Risk

Keep one hand on the chassis to provide ground while inspecting the board or adjusting jumper settings. You can place the board on the envelope, using it as a work surface provided you dissipate the static charge from the bag as described above. If you remove your hand from the board or the bag, touch the chassis to discharge any built-up static electricity before touching them again.

2.2.2 Board Installation

Your Transputer Education Kit circuit board may be installed in either an eight-bit or a sixteen-bit PC bus slot. To install the board, turn off all power to the computer and remove the chassis cover. Select the slot you wish to use. Remove the blank I/O bracket on the back of the PC chassis and save the screw. Insert the board into the slot and make sure that it is seated properly. Fasten the board's edge connector bracket to the PC chassis with the screw from the original bracket. Replace the chassis cover.

2.2.3 Running Hardware Checks

Once your board is installed, turn the power on and check that the board is installed and running correctly. To do this, insert the Test and Demo disk supplied with your kit into the floppy drive and type the following at the command line (if you are not using floppy drive a, replace a with the appropriate drive letter):

```
a:kit_test
```

You should see the following display on your screen:

```
CSA KitTest Version 1.00
Installation and Diagnostic tests for the
Transputer Education Kit.

PC/Link found at 150 (hex) SS found at 160 (hex)

1)  Get warranty number
2)  Show/Select PC/Link Interface address(es)
3)  PC/Link loopback test
4)  Channel I/O test
5)  On-chip memory test
6)  External memory test
7)  Show/Set/Clear error flag
0)  Quit
```

If the fourth line is not as shown above (i.e. `PC/Link found at...`), the PC/Link Interface is not jumpered correctly. Refer to section **3.3 PC/Link Interface Settings** for the correct default settings. If you still have problems, consult chapter **5 Troubleshooting**.

When you have the screen as shown above, select Channel I/O test, by typing a `1` at the prompt. The following should appear on the screen (dots should continue to be printed until you press a key):

```
2 kbytes/dot
Press a key to stop
.....................................................
```

If you do not get this response, refer to section **3.5 System Services Jumpers** for the correct default settings. If you still have problems, consult chapter **5 Troubleshooting**. Type any key to end this portion of the test. The two lines below will be displayed, followed by the original menu:

```
Channel I/O between PC and transputer (kbytes/sec):
111.009 - channel I/O
```

Get your warranty validation number by typing 2 at the prompt. Write this number in the appropriate blank in the warranty registration card and return the card to CSA.

2.2.4 Mandelbrot Demonstration

There is a simple demonstration program included on the Test and Demo disk that graphically displays the well-known Mandelbrot fractal image. It requires no external memory, so if your board passed `kit_test` in the last section, you can run this demonstration.

To run the program, insert the Test and Demo disk in the floppy drive and type the following at the command line (if you are not using floppy drive a, replace a with the appropriate drive letter):

```
a:man
```

The program will then prompt you as follows for the type of graphics card you have in your PC:

```
CSA Mandelzoom Version 2.01 for PC

h)ercules c)ga e)ga v)ga:
```

Enter the card type at the prompt by typing an `h`, `c`, `e` or `v`, and the following will be displayed:

```
Nodes found:  1
using fixed-point arith.

After frame is displayed:

Home - Displays the zoom box. Use the arrow keys to
       move and size the zoom box.  Pressing Home a
       second time expands the contents of the zoom
```

```
          box to a full-screen image.
   Ins  - Alternates the function of the arrow keys between
          moving and sizing the zoom box.
   PgUp - Resets the display to the first image.
   End  - Quit
   -- Press a key to continue --
```

Press any key and the program will draw the Mandelbrot set on the screen. The commands above allow you to select and enlarge any portion of the screen.

2.2.5 Additional Memory

The transputer chip itself incorporates either 2K or 4K of on-chip memory depending on the processor model. This is sufficient to allow you to learn a great deal about the transputer and parallel processing, although the size of the program and data will be somewhat limited. If your board was ordered with no additional memory and you would like to add some, you can do so now (see section **3.2 Adding Memory**).

2.3 Software Options and Memory

The entry-level Transputer Education Kit comes with a T400 processor which incorporates 2K of on-chip memory. Even without any additional memory it is an incredibly powerful computer which can be used effectively to learn about parallel processing and explore the transputer architecture. However, having only 2K of memory does impose some limitations.

The C compiler delivered with the kit produces executable programs for the transputer but actually runs on the PC (i.e. it is a cross-compiler) so the compiler itself is not affected by the lack of additional memory on the transputer. C programs require a minimum of 4-8K of memory when using the standard C I/O libraries. Because of this requirement, C programs which use standard C I/O cannot be run using only the transputer's on-chip memory. There is, however, an aletrnative I/O interface provided with this kit that requires no additional memory. It provides for keyboard input and screen display but no file access. If you have an entry-level Transputer Education Kit with no additional memory, section **2.5.1 A C Program In On-Chip Memory** contains an example program using this I/O interface. For more information on running programs using only the on-chip memory of the transputer, see section **6 C In On-Chip Memory** of this manual and the **C Toolset User's Manual**.

Included with the C compiler is a transputer assembler. Like the C compiler, the assembler runs on the PC (i.e. it is a cross-assembler) and is not affected by any lack of additional transputer memory. When writing in assembly language, the transputer's 2K of on-chip memory suffices to learn all the features of the transputer instruction set.

As contrasted with the C compiler, the Occam compiler runs on the transputer with the PC acting only as a host server, and requires at least 1 Mbyte of memory. Therefore, Occam programs cannot be compiled on the entry-level Transputer Education Kit unless additional memory is added. Occam programs compiled on another computer can be loaded and run on the Transputer Education Kit as long as memory requirements are small (less than 2K for a T400 and less than 4K for a T425/800/805).

With 1 Megabyte of memory in addition to the on-chip memory, the Transputer Education Kit can run both the Logical Systems C and Occam Toolsets with no limitations.

2.4 Software Installation

Before installing the software, you should have successfully completed the hardware installation procedures outlined above.

The following section describes the standard installation procedure for the Occam and C Toolsets. For possible installation options, please see the file `install.doc` on Disk 1 of the disks of the software being installed.

IMPORTANT: If you are already using other versions of the Occam or C Toolsets, installing this software will overwrite your previously installed version.

2.4.1 Installing the C Toolset

The T400 Kit C Toolset comes on two 360 Kbyte floppy disks and requires 2 Mbytes of free space on your hard disk. The professional-level C Toolset (optional) comes on three 360 Kbyte floppy disks and requires 3 Mbytes of free space. If you are installing the professional-level C Toolset and are short on hard disk space see the installation option instructions in the `install.doc` file on Disk 1.

To install this toolset, insert Disk 1 into your floppy disk drive. Run the batch file `install.bat` on Disk 1, giving as parameters the drive letter of the floppy disk drive and the drive letter of the hard drive on which the Toolset will be installed.

For example, if your floppy disk drive is a, and the drive on which you want the Toolset installed is c, type:

```
   a:install a c
```

You will be instructed to insert the other disks as the installation proceeds. The installation takes 10-15 minutes on an IBM PC/AT.

The installation procedure creates a directory called `\lsc`. All the programs necessary to install the Toolset are copied to this directory and all components of the Toolset itself are copied into sub-directories of `\lsc`.

Once the software is installed, there are a few additions and changes you need to make to the DOS environment variables before running the compiler. First, add the directory `\lsc\bin` to the DOS path variables. For example, if the software is installed on drive c, and your present path variable is `c:\system`, the command to add `\lsc\bin` to your path variable is:

```
   path=c:\system;c:\lsc\bin
```

Next, add the three environment variables `ppinc`, `tlib`, and `linkname` with the following commands:

```
   set ppinc=c:\lsc\include
   set tlib=c:\lsc\lib
   set linkname=0
```

To avoid changing the path and setting the three environment variables every time you use the C Toolset, add the four commands above to your `autoexec.bat` file or place them in a separate batch file which you run before using the C Toolset.

2.4.2 Installing the Occam Toolset

The T400 Kit Occam Toolset comes on six 360 Kbyte floppy disks and requires 2.5 Mbytes of free space on your hard disk. The professional-level Occam 2 Toolset (optional) comes on twelve 360 Kbyte floppy disks and requires 5.6 Mbytes of free space. If you are installing the professional Occam Toolset and are short on hard disk space, see the instructions in the `install.doc` file on Disk 1 for information on installing only the parts you will use.

To install this toolset, insert Disk 1 into your floppy disk drive. Run the batch file `install.bat` on Disk 1, giving as parameters the drive letter of the floppy disk drive and the drive letter of the drive on which the Toolset will be installed.

For example, if your floppy disk drive is a, and the drive on which you want the Toolset installed is c, type:

```
a:install a c
```

You will then be given information on how to proceed with the installation. You will be asked yes or no questions during the process. For the standard installation answer yes to all the questions by typing `y` each time you are prompted. No carriage return is necessary.

When all the disks have been read, which takes 5-10 minutes, the installation can be allowed to proceed unattended. A full installation takes 20-50 minutes on IBM PC/AT.

The installation procedure creates a directory called `\itools`. All the programs necessary to install the Toolset are copied to this directory and all components of the Toolset itself are copied into sub-directories of `\itools`.

Once the software is installed, there are a few additions and changes you need to make to the DOS environment variables before running the compiler. First, add the directory `\itools\tools` to the DOS path variables. For example, if the software is installed on drive c, and your present path variable is `c:\system`, the command to add `\itools\tools` to your path variable is:

```
path=c:\system;c:\itools\tools
```

Next, add the four environment variables `IBOARDSIZE`, `ISEARCH`, `ITERM` and `TRANSPUTER` with the following commands:

```
set IBOARDSIZE=#100000
set ISEARCH=c:\itools\libs\
set ITERM=c:\itools\iterms\ibmpc.itm
set TRANSPUTER=#150
```

If your board has more than 1 Mbyte of memory, line one must be modified. Replace `#100000` with `#200000` if your board has 2 Mbytes and with `#400000` if it has 4 Mbytes.

To avoid changing the path and setting the four environment variables every time you use the Occam Toolset, add the four commands above to your `autoexec.bat` file or place them in a separate batch file which you run with one command before using the Occam Toolset.

2.5 Your First Transputer Program

The following section contains examples of how to compile four different versions of a simple program: one in C, one in C using only on-chip memory, one in transputer assembly language, and one in Occam. The program prints "Hello, World" to the screen then echoes characters typed at the keyboard to the screen.

2.5.1 A C Program in On-Chip Memory

If you have installed the C Toolset on your computer as instructed in section **2.4.1 Installing the C Toolset**, you are ready to compile and run a simple C program that uses only on-chip memory and can be run on any Transputer Education Kit.

Type the following commands to make a directory named `\ltxample` on your hard drive and change to that directory:

```
mkdir \ltxample
cd    \ltxample
```

Copy the file `ltxample.c` to this directory from the directory `\lsc\example` with the following command:

```
copy \lsc\example\ltxample.c
```

The file `ltxample.c` contains the following C program which displays `Hello, World` on the screen, and then echoes all characters typed at the keyboard to the screen. It uses the Lt I/O library as described in section **6 C in On-Chip Memory**.

```
         #include <stdio.h>
         #include <conc.h>
         #include <ltio.h>
         #define TRUE 1

main()
{        int ch;
         lt_printf("Hello, World\n");
         while (TRUE)
                 {
                 ch = lt_getch();
                 lt_printf("%c", ch);
                 }
}
```

Type the following command line to compile the program (substitute `lsc8` for `lsc4` if your board contains a T800/805 transputer):

```
ltlsc4 ltxample
```

The batch file `ltlsc4` runs the C preprocessor, the C compiler, the C Toolset assembler and linker. Approximately two pages of status messages are displayed during the compilation. If these status messages are not displayed or if errors are reported during the compilation, go back to section **2.4.1 Installing the C Toolset** and make sure the environment variables are set correctly and that installation was complete.

After the program is successfully compiled, type the following line to load and run the program on the transputer:

```
ld-one ltxample ltio
```

You should see the following line displayed on the screen:

```
Hello, World
```

After this line is displayed, the program will echo any characters typed on the keyboard to the screen. Program operation is terminated by typing a Control-C or Control-Break.

The procedure used to compile and run this program will work for any C program that runs in on-chip memory, uses the Lt I/O library and is contained in one source file (include files do not count as separate files). Just replace `ltxample` in the commands above with the name of the file you are using. If you are compiling several different files and then linking them together, or writing programs to run on more than one transputer, refer to the **C Toolset Users Manual**. For more information on the C tools refer to the **C Toolset Users Manual**. For more information on writing programs using the Lt I/O library refer to chapter **6 C in On-Chip Memory**.

2.5.2 A C Program

If you have installed the C Toolset on your computer as instructed in section **2.4.1 Installing the C Toolset** and have 1 Mbyte or more of memory, you are ready to compile and run a simple C program.

Type the following commands to make a directory named `\cexample` on your hard drive and change to that directory:

```
mkdir \cexample
cd   \cexample
```

Copy the file `examp.c` to this directory from the directory `\lsc\example` with the following command:

```
copy \lsc\example\example.c
```

The file `example.c` contains the following C program which displays `Hello, World` on the screen, and then echoes all characters typed at the keyboard to the screen:

```
#include <stdio.h>

main()
{      char ch;
       printf("Hello, World\n");
       ch = getch();
       while(ch != '\\')
             { printf("%c", ch);
               ch = getch();
             }
}
```

Type the following line to compile the program (substitute `lsc8` for `lsc4` if your board contains a T800/805 transputer):

```
lsc4 example
```

The batch file `lsc4` runs the C preprocessor, the C compiler, the C Toolset assembler and linker. Approximately two pages of status messages are displayed during the compilation. If these status messages are not displayed or if errors are reported during the compilation go back to section **2.4.1 Installing the C Toolset** and make sure you have the environment variables set correctly and that installation was complete.

After the program is successfully compiled, type the following line to load and run the program on the transputer:

```
ld-one example cio
```

You should see the following line displayed on the screen:

```
Hello, World
```

After this line is displayed, the program will echo any characters typed on the keyboard to the screen. Program operation is terminated by typing a backslash (\).

The procedure used to compile and run this program will work for any C program that is contained in one source file (include files do not count as separate files). Just replace `example` in the commands above with the name of the file you are using. If you are compiling several different files and then linking them together, or writing programs to run on more than one transputer, refer to the **C Toolset Users Manual**. For more information on the C tools refer to the **C Toolset Users Manual**.

2.5.3 An Assembly Program

If you have installed the C Toolset on your computer as instructed in section **2.4.1 Installing The C Toolset**, you are ready to compile and run a simple transputer assembly language program.

Type the following commands to make a directory named `\tasmxamp` on your hard drive and change to it:

```
mkdir \tasmxamp
cd \tasmxamp
```

Copy the file `example.tal` to this directory from the directory `\lsc\example` with the following command:

```
copy \lsc\example\example.tal
```

The file `example.tal` contains the following transputer assembly program which displays `Hello, World` on the screen, and then echoes all characters typed at the keyboard to the screen:

```
;     Definitions.
;
#define BOOT_CHAN_OUT   1      /*Boot channel output address */
#define BOOT_CHAN_IN    2      /*Boot channel inpout address */
#define TEMP            3      /*temp storage */

#define A_PARAM         1      /* Parameter offset to "A" */
#define B_PARAM         2      /* Parameter offset to "B" */
#define C_PARAM         3      /* Parameter offset to "C" */

str   .db   "Hello, World\r\n"

;
;     Main program.
;
_main
      ajw   -20                ;Leave room for a few variables
;
;     Get input bootstrap channel address (held in register "C" on
entry).
;
      stl   TEMP               ;pop A
      stl   TEMP               ;pop B
      stl   BOOT_CHAN_IN       ;Save boot input link address
;
;     Compute corresponding output channel address.
;
      ldl   BOOT_CHAN_IN       ;Save boot input link address
      ldc   4
      bcnt                     ;Areg will be 8 for T2, 16 for T4/T8
      xor
      stl   BOOT_CHAN_OUT
;
;     Print messages for user.
;
      ldl   BOOT_CHAN_OUT      ;load output link address
      ldc   str                ;load address of the string
      call  @printstring

;
;     Echo characters.
;
echo
      ldlp  TEMP
      ldl   BOOT_CHAN_IN
      ldc   1
      in
```

```
      ldl   BOOT_CHAN_OUT
      ldl   TEMP
      outbyte
      j     @echo
;
;     The following routine prints the '/0'terminated character string
;     whose address is in register A.  The link channel address to use
is
;     in register B.
;
printstring
      ldl   A_PARAM            ;Get string pointer
      stl   C_PARAM            ;Make a copy
;
length
      ldl   C_PARAM            ;String character address
      lb                       ;Fetch corresponding byte
      cj    @write             ;Done, if zero termination
      ldl   C_PARAM            ;Increment pointer
      adc   1
      stl   C_PARAM
      j     @length
;
;     Determine string length and write it out.
;
write
      ldl   A_PARAM            ;String start pointer
      ldl   C_PARAM            ;String end pointer
      ldl   A_PARAM            ;String start pointer
      sub                      ;String length
      ldl   B_PARAM            ;Get link address
      rev                      ;Length in A, link in B, start in C
      out                      ;Write string out
      ret

      .end
```

Type the following command line to assemble the program:

```
lsa example
```

The batch file `lsa` runs the C preprocessor, the transputer assembler and linker. Approximately two pages of status messages are displayed during the assembly. If these status messages are not displayed, or if errors are reported during the process, go back to section **2.4.1 Installing the C Toolset** and make sure you have the environment variables set correctly and that installation was complete.

After the program is successfully compiled, type the following line to load and run the program on the transputer:

```
ld-one example tio
```

You should see the following line on the screen.

```
Hello, World
```

After this line is displayed, the program will echo any characters typed on the keyboard to the screen. Program operation is terminated by typing a Control-C or Control-Break.

The procedure used to compile and run this program will work for any transputer assembly language program that is contained in one source file (include files do not count as separate files). Just replace `example` in the commands above with the name of the file you are using. If you are compiling several different files and linking them together, or writing programs to run on more than one transputer refer to the **C Toolset Users Manual**. For more information on the C tools refer to the **C Toolset Users Manual**.

Both the C and Occam Toolsets support in-line assembly code. For more details refer to the respective Toolset documentation.

2.5.4 An Occam Program

If you have installed the Occam Toolset on your PC as instructed in section **2.4.2 Installing the Occam Toolset** and have 1 Mbyte or more of memory on your transputer board, you are ready to compile and run a simple Occam program.

Type the following commands to make a directory named `\occexamp` on your hard drive and change to that directory:

```
mkdir \occexamp
cd \occexamp
```

Copy the file `example.occ` to this directory from the directory `itools\examples` with the following command:

```
copy \itools\example\example.occ
```

The file `example.occ` contains the following Occam program which displays `Hello, World` on the screen, and then echoes all characters typed at the keyboard to the screen:

```
#INCLUDE "hostio.inc"

PROC simple (CHAN OF SP fs, ts, []INT memory)
  #USE "hostio.lib"
  BYTE char, result:
  SEQ
    so.write.string(fs, ts, "Hello, World*C*N")
    so.getkey(fs, ts, char, result)
    WHILE char <> '\'
      SEQ
        so.write.char(fs, ts, char)
        so.getkey(fs, ts, char, result)
    so.exit(fs, ts, sps.success)
:
```

Type the following command line to compile the program (substitute `occ5` or `occ8` for `occ4` if your board contains a T425 or T800/805 transputer):

```
occ4 example
```

The batch file `occ4` runs the Occam compiler, linker and booter. If errors are reported during the compilation go back to section **2.4.2 Installing the Occam Toolset** and make sure you have the environment variables set correctly and that installation was complete.

After the program is successfully compiled, type the following line to load and run the program on the transputer:

```
iserver /sb example.b4h
```

You should see the following line on the screen:

```
Hello, World
```

After this line is displayed, the program will echo any characters typed on the keyboard to the screen. Program operation is terminated by typing a backslash (\).

The procedure used to compile and run this program will work for any Occam program that is contained in one source file (include files do not count as separate files). Just replace `example` in the commands above with the name of the file you are using. If you are compiling several different files and then linking them together, or writing programs to run on more than one transputer refer to the **Occam Toolset Users Manual**. For more information on the Occam tools refer to the **Occam Toolset Users Manual**.

2.6 The Next Step

If you want to change the default jumper settings or add multiple boards to your setup, continue on to Chapter 3; otherwise, you are ready to explore the world of transputing. Here we present a few possible directions to take in your exploration of transputers and parallel processing.

If you know little or nothing about the transputer and its architecture and would like to learn more about it while learning Occam, read the **Tutorial Introduction to Occam Programming** and refer to it while working through chapters 1-5 of the **Occam Toolset User's Manual** and the examples in the **Workbook**.

If you want to write C programs and learn the transputer architecture as well as parallel processing start by working through the examples in section **15.4** of the **C Toolset Users Manual**. You will probably also want to read the **Workbook** even though it is couched in Occam, as it contains a great deal of information that is not language specific.

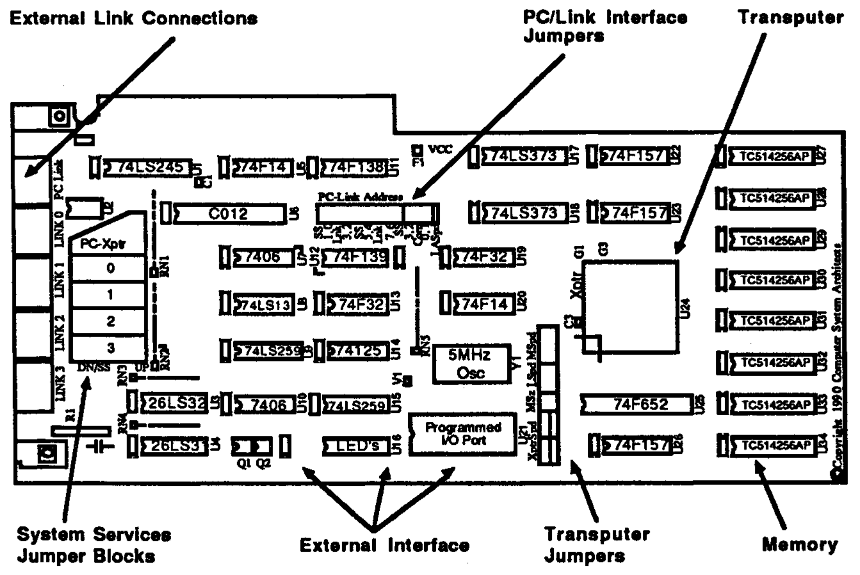
If you want to get to know the architecture of the transputer at the machine language level read the **Transputer Instruction Set** section of the **Transputer** book and use it in conjunction with the assembler furnished in the C Toolset. For further information, order the **Transputer Databook** and **Communicating Process Architecture**, available from CSA.

If you want to explore hardware interfacing and real-time control, refer to section **4 External Interface** of this book, and also the hardware interfacing examples in the back of the **Workbook**. You might want to order either the user 8-bit parallel port interfacing kit or transputer serial link interfacing kit to facilitate building the example circuits on your own. For more detailed hardware information on all aspects of the Transputer order the **Transputer Databook** , available from CSA.

Whichever direction you take, you are at the start of an exciting educational and computing experience. Happy transputing!

3 Installation Options

3.1 Board Layout


External Link Connections
PC/Link Interface Jumpers
Transputer
System Services Jumper Blocks
External Interface
Transputer Jumpers
Memory

The Transputer Education Kit board is designed for experimentation with transputers and multiprocessing. All active components are socketed for ease of repair should they somehow become damaged during a hardware experiment. Use the exact components described below to replace any damaged devices.

3.1.1 List of Components

Component #	Description	Component #	Description
c1,2	CAP 1.0 µf ceramic	RN4	180/390 8 pin resistor net.
C1	CAP 10 µf tantalum	U1	74LS245
C2,3,5-23,25	CAP 0.1 µf ceramic	U2	DS8921
C26-33	CAP 0.47 µf	U3	26LS32
P1-5	MiniDIN connectors	U4	26LS31
R1	Resistor	U5	74F14
RN1,2,5	1.5k 10 pin resistor net.	U6	IMSC012P-20
RN3	220/330 8 pin resistor net.	U7	7406

Component #	Description	Component #	Description
U8	74LS13	U19	74F32
U9	74LS259	U20	74F14
U10	7406	U21	DIP24 socket (600mil)
U11	74F138	U22	74F157
U12	74F139	U23	74F157
U13	74F32	U24	IMS TXXX (transputer)
U14	74LS125	U25	DIP 24 socket (for optional 74LS652)
U15	74LS259	U26	74F157
U16	DIP16 socket	U27-34	256Kx4 memory
U17	74LS373	Y1	5 Mhz Osc.
U18	74LS373		

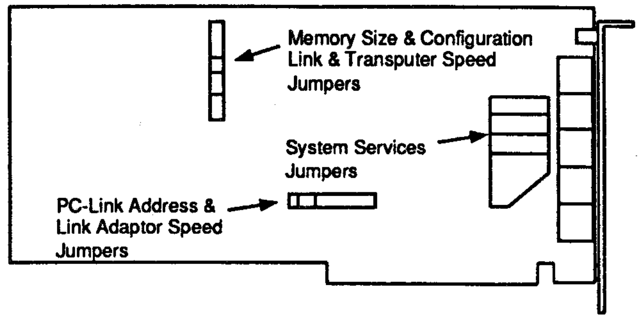
3.1.2 Optional Components

The following components are only needed when interfacing with external hardware and are optional. The components U16, U25, Q1 and Q2 are available from CSA. U21 is a socket that provides signals for external interfaces. These options are further explained in section **4 External Interface**:

Component #	Description	Component #	Description
U16	8 LED's	U25	74LS652
U21	I/O port	Q1, Q2	Power transistors

3.1.3 Jumper Locations

Three groups of jumperblocks are outlined on the silk-screen of the Transputer Education Kit board. Shunts are placed across the pins within these jumper blocks to allow hardware changes to the board or to interconnect boards in a multi-transputer system. Their use is detailed in sections **3.2-3.6**.


Memory Size & Configuration Link & Transputer Speed Jumpers
System Services Jumpers
PC-Link Address & Link Adaptor Speed Jumpers

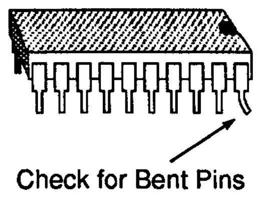
Jumper Block Locations

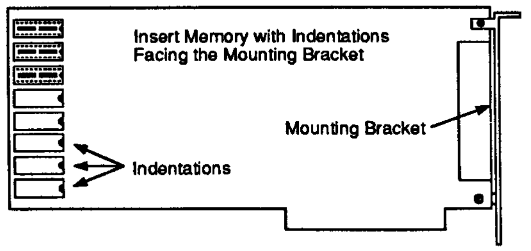
3.2 Adding Memory

The Transputer Education Kit is designed to accommodate 8 pieces of 256Kx4 (or 1Mx4) dynamic random access memory (DRAM) in 300 mil. 20 pin dual-in-line-packages (DIPs). Adapter daughter boards are available from CSA for other memory configurations. Please contact CSA for information on memory expansion adaptors (i.e. 2Mb using 16 pieces of 256x4 DRAM or 4Mb using 8 pieces of DRAM in zig-zag-in-line-packages(ZIPs)).

The board is preconfigured to accept 1 Megabyte of memory. If 2 or 4 Megabytes are to be added, see section **3.2.3 Memory Size** for the correct jumper settings. Any user modifications to the Transputer Education Kit board voids the CSA warranty. CSA-performed upgrades are covered under warranty.

When installing memory, be sure to take precautions to avoid damaging static-sensitive components. On the top surface of each memory chip is an indentation marking its pin orientation. When installed, these chips must be aligned so that the marked end of each chip is facing the mounting bracket on the far board edge. Inspect the rows of pins on each chip before inserting it into its socket. Bent pins can be carefully straightened with needle-nose pliers to fit the socket.



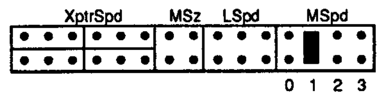


After the eight memory chips have been inserted into their sockets, examine each chip from both sides and from the ends to be sure that all pins are seated properly. Also check to see that the indentations are all aligned with each other and are facing the mounting bracket.

WARNING: Improperly inserted memory may result in damage to the board and/or the memory when power is applied. This damage cannot be covered under CSA's warranty.

3.2.1 Memory Speed

The MSpd jumper block is preconfigured for one extra cycle for external memory access (if a 20MHz processor is installed, as with the entry-level kit). This extra cycle is commonly referred to as a wait state. The numbers 0-3 below the block correspond with the possible number of wait states. If you are adding your own memory to the board, see the table below to determine the correct number of wait states.

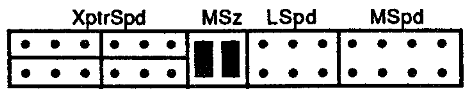


MSpd Set for 1 Wait State

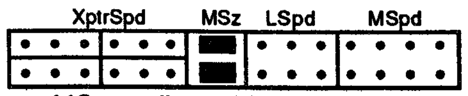
256Kx4 MEMORY TYPES		*WAIT STATES REQUIRED*		
Manufacturer	Part Number	20 MHz	25 MHz	30 MHz
Fujitsu	MB81C4256-10	1	2	3
	MB81C4256-12	2	3	–
Hitachi	HM514256AP-80	0	1	2
	HM514256AP-10	1	2	3
	HM514256AP-12	2	3	–
Intel	I 21014-08	0	1	2
	I 21014-10	1	2	3
Motorola	MC514256P-08	0	1	2
	MC514256P-10	1	2	3
	MC514256P-12	2	3	–
NEC	MSM514256-08C	0	1	2
	MSM514256-10C	0	1	2
	MSM514256-12C	1	2	3
OKI	MSM514256-10	1	2	3
	MSM514256-12	2	3	–
Texas Inst.	TMS44C256-80	0	1	2
	TMS44C256-10	1	2	3
	TMS44C256-12	2	3	–
	TMS44C256-15	3	–	–
Toshiba	TC514256P-70	0	1	2
	TC514256P-80	0	1	2
	TC514256P-10	1	2	3

3.2.2 Memory Size

The MSz jumper block is preconfigured for 1 Mbyte of memory. It is selectable for 1, 2 or 4 Megabytes but is irrelevant if no off-chip memory is installed. 2 Mbyte and some 4 Mbyte configurations require a daughter board which can be ordered from CSA.



MSz configured for 1 Mbyte



MSz configured for 2 or 4 Mbytes

3.2.3 Memory Tests

When you have installed memory as indicated above, use `kit_test` on the Test and Demo disk to test that memory for proper operation. To do this, insert the Test and Demo disk in the floppy drive and type the following at the command line (if you are not using floppy drive a, replace a with the appropriate drive letter):

```
a:kit_test
```

You should see the following display on your screen:

```
CSA KitTest Version 1.00
Installation and Diagnostic tests for the
Transputer Education Kit.

PC/Link found at 150 (hex) SS found at 160 (hex)

1)  Get warranty number
2)  Show/Select PC/Link Interface address(es)
3)  PC/Link loopback test
4)  Channel I/O test
5)  On-chip memory test
6)  External memory test
7)  Show/Set/Clear error flag
0)  Quit
```

Select the external memory test by typing 6 at the prompt and you will see the following:

```
How much memory is installed in the board
1)  1 Mbytes
2)  2 Mbytes
3)  4 Mbytes
0)  Return to main menu
->
```

Select the option that indicates the amount of memory that you have installed. The following lines will be displayed if you have installed 1Mb of DRAM:

```
Checking 1 Meg of memory
To test all possible combinations let the test go through
at least 67 passes
Type any key to continue
```

Type any key. The program will repeatedly check the memory and display the results on the screen. If the test finds no errors, the results will be displayed as follows:

```
Pass number 1
writing data.....................
checking data......................
There are 0 memory errors
```

If the test does find errors, the number of the defective chip will be displayed on the screen:

```
pass number 1
writing data.....................
checking data......................
There are 461822 memory errors

chip U28 is bad
```

Type any key to stop the test. If the test finds errors, replace the defective chip and run the test again.

3.3 PC/Link Interface Settings

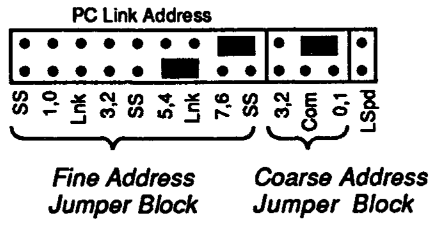
3.3.1 PC/Link Interface Address

The PC/Link Interface portion of the Transputer Education Kit circuit board performs two distinct functions: 1) it interfaces a bidirectional transputer-style link to the PC host, and 2) it allows the PC to generate and respond to a set of transputer control signals referred to as system services. The PC sees the bidirectional transputer link as a byte-wide I/O device. This device has several registers which the PC can address to read link data in, to write link data out, and to determine the status of the link input and output registers (i.e. link-data-ready or link-ready-for-data). System services signals consist of transputer reset, analyse, and error. The reset signal, as the name implies, is the means by which the PC host can reset or initialize,the attached transputer or network; analyse provides a mechanism for debugging a transputer network; and error is used to notify the PC host of an error condition existing in one or more transputers within an attached network.

Although each of these signals is discussed in great detail in the accompanying documentation, it is sufficient for most transputer users to understand only that every transputer within a network must be provided with a complete set of these signals. Without an incoming link a transputer would have no source of program, nor of data on which to operate. Without an output link a transputer would have no way to report computed results (or for that matter, to pass along program and data to other transputers within a network). And without system services signals there would be no way to initialize or reinitialize, nor to properly debug, even a single transputer.

The software included with the Transputer Education Kit copes very well with the details of the PC/Link Interface, and manages to insulate the novice from those details with but one possible exception. The portion of the software which runs on the PC expects the PC/Link Interface registers to be within a specific PC I/O address range. The Transputer Education Kit circuit board is preconfigured at the factory to appear within this address range. In the unlikely event that your PC has an already installed device which conflicts with those addresses, or in the equally unlikely event that you are using an early model IBM-brand PC, or very early PC-compatible (which didn't decode those addresses for I/O), you might need to alter some on-board jumper settings (in the jumper block labeled PC Link Address), and to instruct the software to utilize a different-than-normal address range.

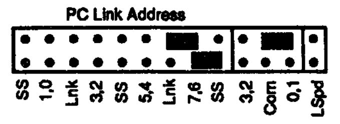
The PC Link address can be selected from various addresses in the range of 100-370h. The coarse address (the hundreds digit) jumper block selects between 100, 200 and 300. The fine address jumper block selects from 00 through 70 and distinguishes between PC Link and system services addresses. Separate fine addresses must be chosen for system services and the PC Link. The Transputer Education Kit board is preconfigured with the PC link addressed at 150 and system services at 160. These addresses will work for most installations. In some very early PC compatibles, 150 and 160 are reserved addresses. In this case addresses 300 and 310 can be used. The following diagram shows the jumpers as set at the factory. Other address configurations are explained below.



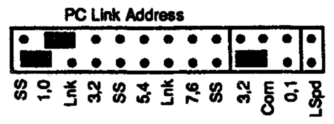
To set a coarse address of 100, 200, or 300 (do not select 000, it is reserved for PC operations), place a shunt between a numbered pin and a common pin in the coarse address selection block. Placing a shunt between pin 1 and **Com** chooses a coarse address of 100, placing a shunt between pin 2 and **Com** selects 200, etc.

In addition to selecting the tens digit the fine address block is also used to distinguish between the PC Link and system services addresses. Placing a shunt between **Lnk** and the pin numbered 0 chooses the PC Link fine address 00. Placing a shunt between **SS** and the pin numbered 1 chooses the system services fine address 10. The form of the addresses of such a configuration would be **Lnk** = X00 and **SS** = X10, where X is the coarse address chosen.

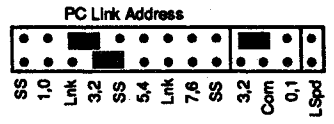
The **Lnk** and **SS** addresses are actually just base addresses. The actual addresses used by **SS** and **Lnk** range between the base address and the base address + 7. That is, when **Lnk** is at 150 and SS is at 160, the addresses in the block 150-157 and 160-167 are used.



Lnk=160 SS=170



Lnk=300 SS=310



Lnk=220 SS=230

Most transputer software expects the PC Link at 150 and SS at 160, however, many software packages allow any address as long as the PC Link address is 10 less than the SS address, for example PC Link = 160, SS = 170. The Occam Toolset provided with the Transputer Education Kit uses 150 and 160 unless another adress is specified in which case it allows any address where the PC Link address is 10 less than the SS address. The C Toolset uses 150 and 160 by default and can run at that address or at the address 300, 310.

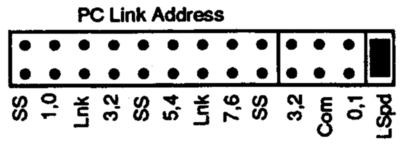
If you are using more than one PC/Link Interface in the same PC they must be at separate addresses. If you want to put put more than one Transputer Education Kit board in a PC but only plan to use one PC/Link Interface remove all shunts from the coarse and fine address blocks of the PC/Link Interface(s) you are not using.

Both the PC Link and PC Link System Services signals can be accessed from a PC program. For detailed information see appendix **A.2 Memory Mapped I/O Interface Addresses**

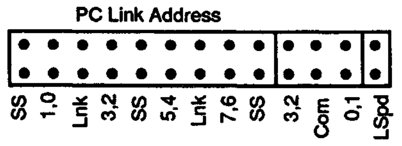
3.3.2 PC Link Speed

The PC Link can be set to operate at 10 or 20 Megabits per second (Mb/s). The PC Link speed must correspond with the speed of the link to which it is connected (usually Link0 of the transputer on the same Transputer Education Kit circuit board - see section **3.4.2 Link**

Speed). The LSpd jumper block is used to select the PC Link speed. Placing a shunt on the pins in the LSpd jumper block results in a PC Link speed of 10 Mb/s. Removing the shunt in the LSpd jumper block results in a PC Link speed of 20 Mb/s. The PC Link speed is preconfigured to 20 Mb/s at the factory.



LSpd shunted for 10 Mb/s

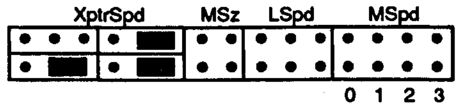


LSpd open for 20 Mb/s

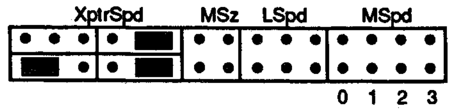
3.4 Transputer Settings

3.4.1 Transputer Speed

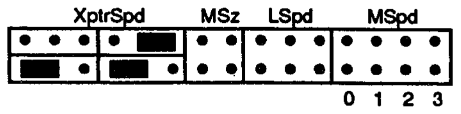
The XptrSpd jumper block is preconfigured to 20, 25, or 30 MHz depending on the processor installed in your board. **IMPORTANT:** Processors are not reliable when operated above their rated speed but can be reliably operated slower than their rated speed.



XptrSpd configured to run at 20 MHz



XptrSpd configured to run at 25 MHz

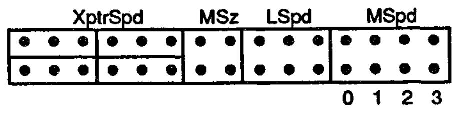


XptrSpd configured to run at 30 MHz

There are many possible shunt combinations in the XptrSpd jumper block. The three shown above are the only combinations supported on the Transputer Education Kit board.

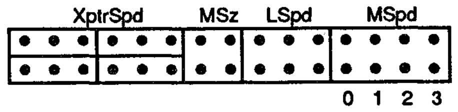
3.4.2 Link Speeds

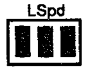
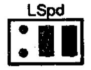
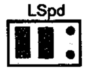
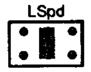
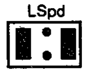
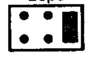
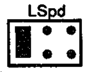
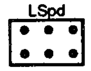
The transputer links can be set to operate at 5, 10 or 20 megabits per second (Mb/s). The speed of Links 1, 2 and 3 are always the same and Link 0 can be set to a different speed. The T400 processor delivered with the entry-level Transputer Education Kit has only two links. The Transputer Education Kit circuit board is designed to accommodate other transputer models which do have four links.



L0=20 Mb/s L1-3=20 Mb/s
LSpd Unshunted as Preconfigured

Shunts are placed in the LSpd jumper block to select the various link speed combinations as indicated in the table below. Although link communication is independent of processor speed, communications across a link can be performed only when the link speeds are set the same at both ends of the link. The transputer links are set at the factory to run at 20 Mb/s.



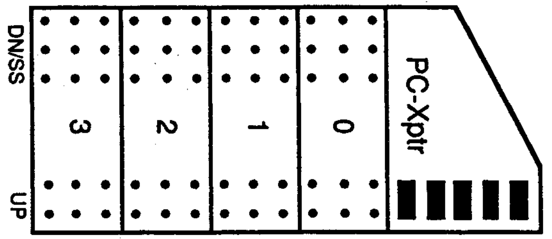
LSpd	L0	L1-3	LSpd	L0	L1-3
	= 10	10		= 5	10
	= 10	10		= 20	20
	= 10	5		= 5	5
	= 10	20		= 20	20

The signals in the LSpd jumper block are often referred to in transputer documentation as LinkSpecial, Link0Special, and Link123Special. LinkSpecial is on the pair of pins on the right, Link123Special is in the center pair and Link0Special is on the left pair.

Note that 5 and 10 Mb/s speeds can be used for more reliable communications when long distances (10-40 meters) exist between transputers or when transputers are operated in electrically noisy environments. Although the transputer links can be set to 5, 10 or 20 Mb/s the PC Link is limited to speeds of 10 and 20 Mb/s.

3.5 PC-Xptr Jumpers

The PC-Xptr jumper block is used to connect the PC/Link Interface to the transputer. It is preconfigured at the factory as shown below. This configuration connects the PC Link to Link 0 of this same board's transputer circuitry. It also connects the PC/Link Interface system services signals to that transputer.



The system services jumpers as preconfigured

Jumper blocks 0-3 are used when interconnecting multiple transputers. But remember that the T400 processor which is delivered with the entry level Transputer Education Kit has only two links, and thus blocks 2 and 3 will not be active with those systems. The following section shows how to set up multi-transputer networks.

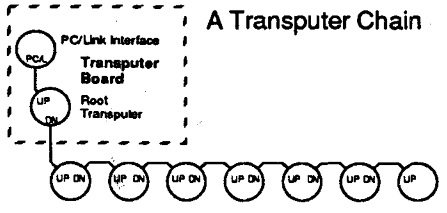
3.6 Setting Up Multi-Transputer Networks

Transputers are designed to communicate with each other over high-speed, bidirectional, serial links. The Transputer Education Kit supports this capability in an extremely convenient and flexible way by providing several 8-pin mini-DIN external connectors (along the rear edge of the circuit board) to which cables can be attached in order to "link together" multiple boards. These connectors accommodate industry-standard cables (compatible with Macintosh printer cables), and carry not only bidirectional link data, but system services signals as well (transputer reset, analyse and error). Cables can range in length from a few inches to 50 feet or more, and can interconnect transputers residing within a number of different PC's or expansion chassis throughout the room. Within the constraint of the number of links per transputer, the user can wire up an entire network of transputers in any of a number of possible configurations.

In such a network , one transputer, called the root, is connected to a PC host via a PC/Link Interface. This is usually accomplished using the PC-Xptr jumpers (as described in section 3.5), but could alternatively be accomplished with an external cable as described later on. The root transputer is simply the one through which communication with the PC host occurs. That is, it's the one through which the network is initialized (i.e. reset) and loaded with program and input data. It is also the transputer through which run-time I/O messages must pass. When an error condition is encountered, it's through this root transputer that the network is debugged.

Connecting transputers into a network is really quite a straightforward process, as far as the links are concerned. With T400's that's especially true. Since these transputer models have only two links each, the only possible network topology is a linear chain as shown below.

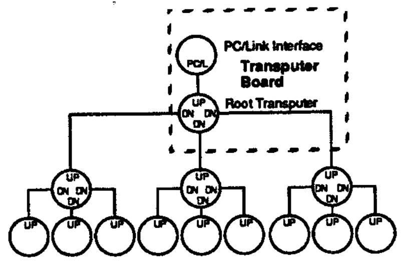
A Transputer Chain



Don't be dismayed with the simplicity of this configuration. It is not only an easy one to learn with, but it is, in fact, one of the most popular configurations in use by seasoned practitioners.

With T425's or T805's one can, however, go further, and configure a tree or a toroidal mesh as shown next.

A Transputer Tree



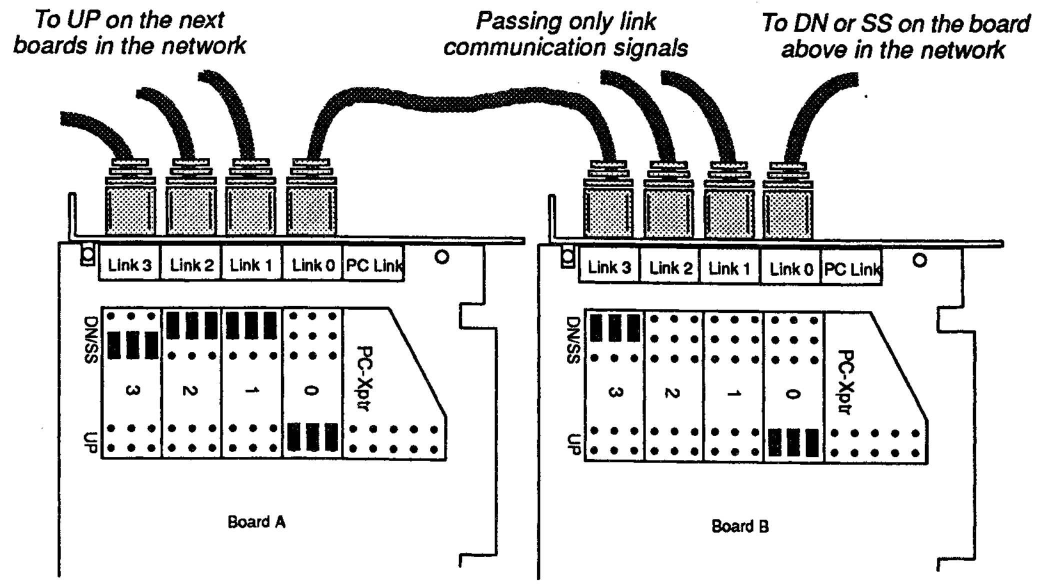
As can be deduced from the above diagrams, link signals are point-to-point. That is, a link connection on one transputer connects to one and only one link connection on another transputer. (Although not usually done, it is also possible to connect two links on one transputer together with an external connection.) The only way to have one link go to several links or to have several links multiplexed onto one link is to use a transputer. In the case of many links to one, the transputer reads data from the links to be multiplexed and sends it to one link. In the case of one link to many the transputer reads data from the one link and sends it to each one of the other links. If one link is to go to more than three links, or more than three links are multiplexed onto one link, then more than one transputer must be used since transputers come with a maximum of four links.

The above diagrams and foregoing discussion have been concerned with link interconnections. As was stated earlier, however, the same cables which carry link communication also carry transputer system services signals which are used to initialize a transputer network and detect run-time error conditions. The nature of these system services signals is quite different from that of the link signals, and the topic of system services signal distribution is somewhat more complex than the corresponding topic for links.

Transputer system services are actually comprised of three separate unidirectional signals; two flowing in one direction (reset and analyse) and one in the other (error). The direction of reset and analyse is always outward from the PC host (and therefore from the root transputer), and the direction of error is always inward. Reset and analyse are generally initiated by the PC host and then propagated throughout the entire network, while error may be asserted simultaneously by any one or more of the processors out in the network, with the logical OR of those assertions being asserted back at the host. An exception to this rule can occur, however, in that it is possible to configure a network such that some intermediate processor can act as sub-host to a sub-set of processors in that network. In this case the program running in the sub-host controls reset and analyse and responds to error for that sub-network. That is, when it comes to system services signals, the PC host is usually the master and the transputers the slaves, but it is possible, if desired, to configure a hierarchy of master/slave relationships throughout the network.

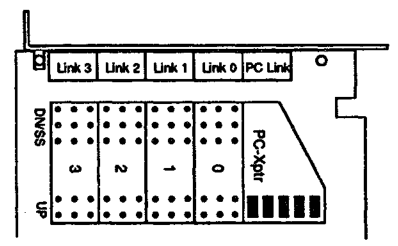
Actually, besides the three signals which comprise system services, there are also three categories of system services, designated up, down, and sub-system (UP, DN, and SS) Every transputer, whether a single root attached directly to the PC host or one of many transputers in a network, must have a source of system services (UP). Furthermore, every Transputer Education Kit transputer is capable of either passing along a copy of those same signals (DN), or of generating a new set of them under local program control (SS), for the benefit of the remaining transputers in a network. Thus one transputer's DN (or SS) is another transputer's UP.

Using the jumper blocks labeled 0, 1, 2, and 3 on the Transputer Education Kit circuit board, it is possible to specify (using shunts supplied with the Transputer Education Kit) over which cable the transputer will receive its system services (UP), and over which cables (if any) system services will be sent out. Jumper block 0 corresponds to the Link 0 cable connector, jumper block 1 to the Link 1 cable connector, and so on. (Remember that the T400 transputer has only 2 links.) In the diagram below, the transputer on Board A is configured to receive its system services (UP) over its Link 0 cable, to send copies of those signals (DN) out over its Link 1 and Link 2 cables, and to send locally originated system services (SS) out over its Link 3 cable. The transputer on Board B is also configured to receive its system services (UP) over its link 0 cable, but will send only one copy of those signals (DN) out over its Link 3 cable. Note that transputer B's DN is transputer C's UP. That is, both transputer A and transputer B are slaves to the same PC host or transputer sub-host.

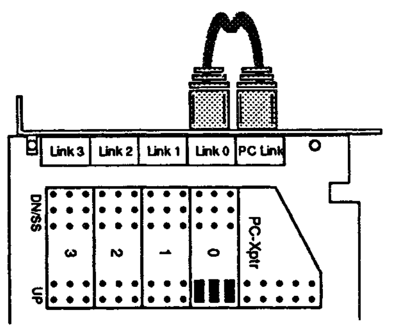

To UP on the next boards in the network
Passing only link communication signals
To DN or SS on the board above in the network
Link 3
Link 2
Link 1
Link 0
PC Link
DN/SS
UP
3
2
1
0
PC-Xptr
Board A
Board B

You will notice that there are five cable connections on the end of the Transputer Education Kit board rather than 4 as might be expected given that a transputer has a maximum of four links. The fifth connector is for the PC Link. The PC/Link Interface converts signals from the PC bus to transputer link and system services compatible format. This PC Link is the means by which the PC communicates with the root transputer.

Until now, the connection between the PC Link and Link 0 has been made by connecting the shunts in the PC-Xptr jumper block. These shunts also connected the system services signals from the PC/Link Interface to the transputer on the same board. The equivalent connection could be made by removing the shunts in the PC-Xptr block, placing shunts in block 0 on the UP pins, and connecting a cable between PC Link and Link 0.


Connection Using Shunts

=


Connection Using External Cable

The PC-Xptr jumper block is thus not really necessary, but is provided for the convenience of not having to use any external cables on single-transputer systems. When shunted, the shunts in the PC-Xptr block connect the PC Link to Link 0 of the transputer and the PC/Link Interface system services to the UP system services of the transputer. When the shunts are removed, and in the absence of an external cable, there is absolutely no connection between the PC and the transputer on the board being considered. The transputer circuitry is simply drawing power from the PC. The transputer could derive its UP system services from another board, or even from another PC/Link Interface on a different board.

The following sections contain examples of three common transputer network configurations with the jumper and link cable connections necessary to implement them. Studying these examples below will help you become familiar with the concepts necessary to create your own configurations. When you feel comfortable with these ideas, you might make your own configuration using one of the examples below as a starting point.

IMPORTANT: As a safety check to avoid illegal jumper settings, observe the following three rules when placing shunts:

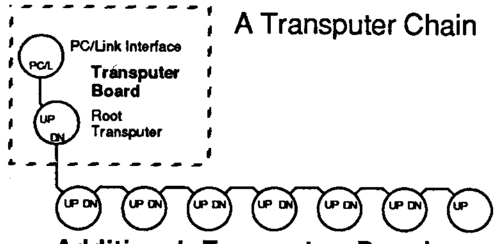
- Only one jumper block on each board may be shunted to UP.
- In any single jumper block, only one section (UP, DN, or SS) may be shunted.
- If the PC-Xptr jumper block is shunted, jumper block 0 must be left empty.

The Check utility on the Test and Demo disk is very useful in setting up networks. This utility is included on the Test and Demo disk in an archived format. The instructions to install and use Check are in the `read.me` file in the `check` subdirectory of the Test and Demo disk.

Check loads itself into a network of transputers and prints the actual link connections on the screen. To set up a multi-transputer network, verify that each board works individually, then build the network one board at a time. Run Check after each board is added to see if the network connections are as you intended.

For a more technical description of the link and system services signals discussed above see appendix **A.3 Board Schematics and Block Diagram** and the Technical Specifications in the **Transputer** book. For information on how to generate system services signals in software on the PC see section. For information on how to generate SS system services signals on the tranputer see section

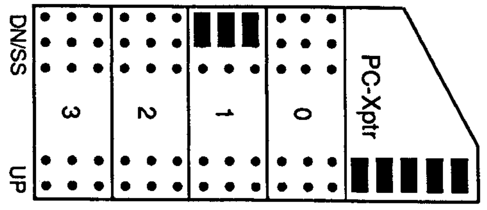
3.6.1 Chain Jumper Settings



Additional Transputer Boards

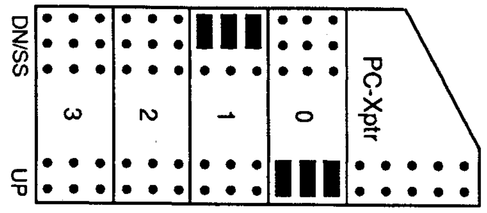
A transputer chain uses only two links to join multiple transputers in a linear chain. The root transputer is connected to the PC host with the PC/Link Interface using the PC-Xptr jumper block and to the second transputer in the chain. The root transputer receives system services signals from the PC/Link Interface and passes them down the chain through its Link 1 cable. The second transputer receives system services signals on its Link 0 cable and passes them further through its Link 1 cable to the next transputer, continuing until the chain terminates.

On the root transputer board, place shunts in the PC-Xptr jumper block and in jumper block 1 as shown below. The PC-Xptr shunts connect the PC/Link Interface system services to the root transputer and the PC Link to that transputer's Link 0. The shunts in jumper block 1 enable a copy of the PC/Link Interface system services to be passed over an external cable to the next transputer in the chain.



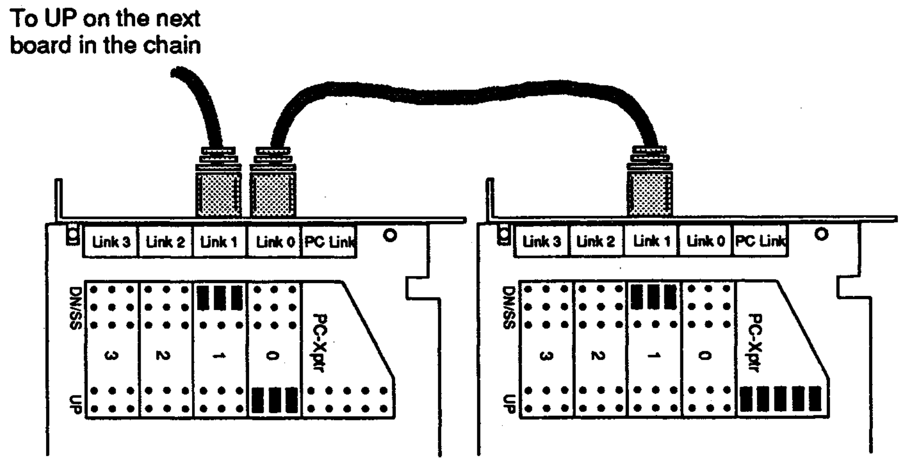
Root board connected to the PC/Link Interface and to DN

The second transputer in line (and all following) receives system services through its Link 0 cable and passes them on to the next transputer through its Link 1 cable. Jumper block 0 has shunts on UP to receive system services and jumper block 1 has shunts on DN to send system services to the next transputer down the line.



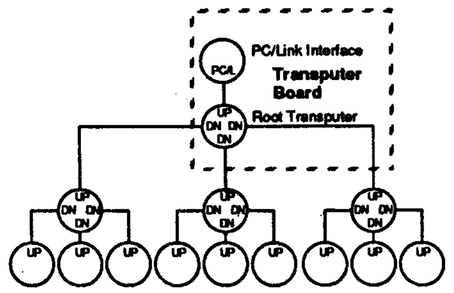
Following boards in the chain receiving through Link 0 and sending through Link 1

External cables, as shown below, interconnect each transputer's Link 1 to the next transputer's Link 0, as well as interconnecting each transputer's DN system services to the next transputer's UP system services.



Second Board   Root Transputer Board

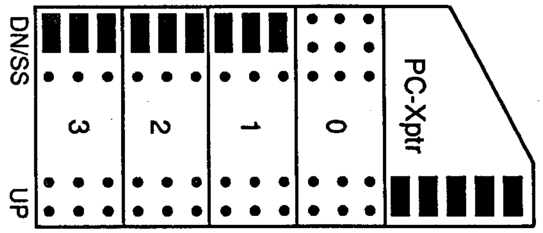
3.6.2 Tree Jumper Settings



Additional Transputer Boards

Configuring a transputer network as a tree requires the use of T425 or T805 transputer models, as T400s have only two links. The root transputer is connected to the PC/Link Interface and to the three transputers on the second level of the tree. The root transputer receives system services signals from the PC/Link Interface and passes them on to the transputers on the second level of the tree through Link connectors 1, 2 and 3. The second level transputers receive system services signals on their Link 0 connectors and pass them to the next level transputers through their Link connectors 1, 2 and 3 and so on until all transputers have a source of system services.

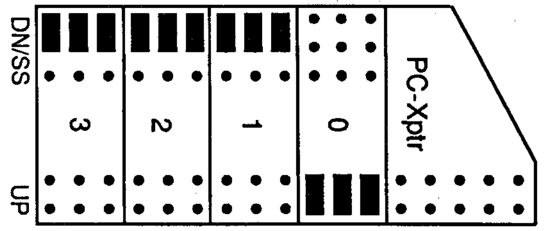
To start off this tree network, shunts on the root transputer board are placed in jumper blocks PC-Xptr, 1, 2 and 3 as shown below. The PC-Xptr shunts connect the PC/Link Interface system services to the root transputer and the PC Link to Link 0 of the root transputer. Jumper blocks 1, 2 and 3 have shunts on DN to send copies of the PC/Link Interface system services to transputers in the next level of the tree.



Root board connected with the PC/Link Interface through Link 0 and passing on through Links 1 - 3

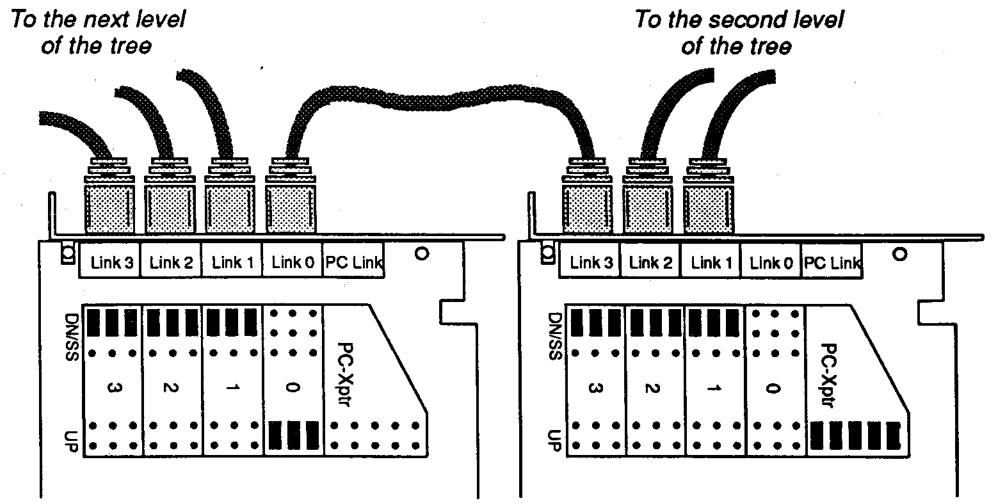
The remaining transputer boards (those at level two and below) should have shunts placed in jumper blocks 0, 1, 2 and 3 as shown next. For each of these boards, system services UP comes through the Link 0 connector, and copies of it (DN) are sent to the next level of transputers via the Link 1, 2 and 3 connectors.

Note that all shunts are removed from the PC-Xptr jumper block on these boards.



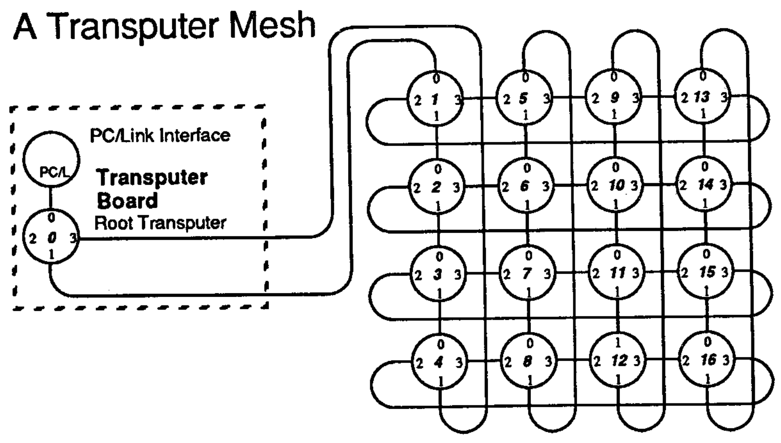
Following boards in the tree receiving through Link 0 and passing on through Links 1 - 3

External cables are connected from link connectors shunted for DN to link connectors shunted for UP as shown below.


To the next level of the tree
To the second level of the tree
Link 3
Link 2
Link 1
Link 0
PC Link
DN/SS
UP
3
2
1
0
PC-Xptr

Second Level Board **Root Transputer Board**

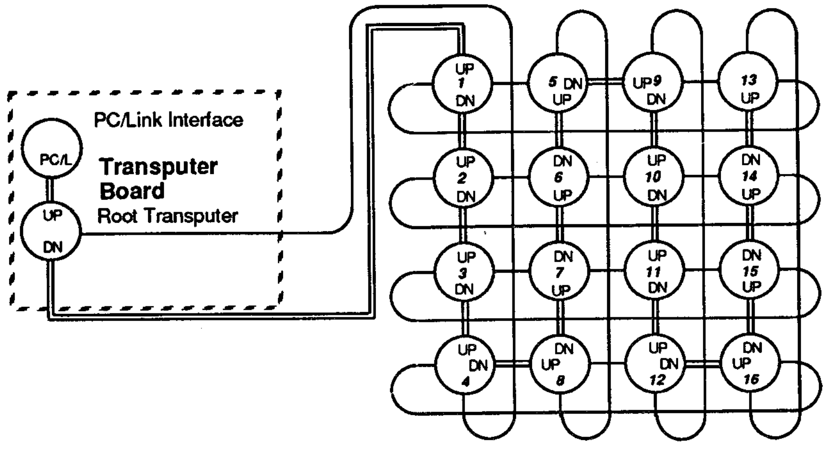
3.6.3 Mesh Jumper Settings



Additional Transputer Boards

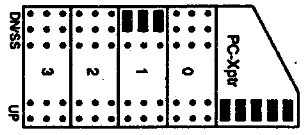
This example connects T425 or T805 transputers into a toroidal mesh. The system services connections in this example are different from those in the previous two examples. In those examples, every link cable carried system services signals as well. In a mesh network, some cables carry only link communication signals.

The diagram above shows only the link connections for a transputer mesh. In the diagram below, double lines indicate the link cables that carry system services signals as well. System services are passed from the root transputer to the other transputers in a linear chain even though the link interconnections form a toroidal mesh.

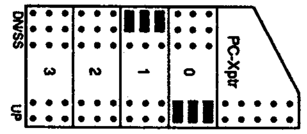


Additional Transputer Boards

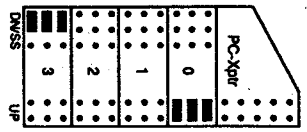
The following diagrams show the jumper connections for some of the transputers in this mesh. You should be able to figure out the connections for the rest of the boards using these as guides.



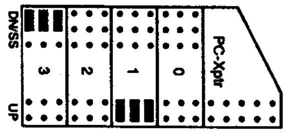
Root Transputer Board



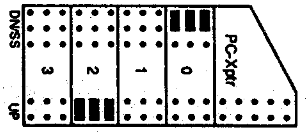
Board Number 1



Board Number 4



Board Number 5



Board Number 8

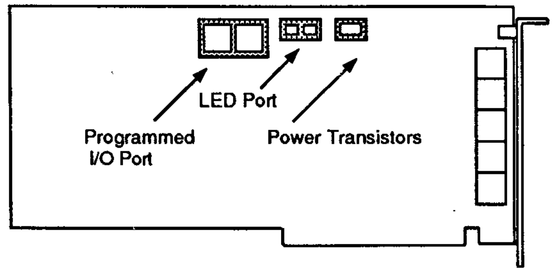
Note that in each of the preceding examples, each transputer had one, and only one, UP system services connection. Without an UP, a transputer could not be initialized (i.e. reset). With more than one UP connection, a system services "loop" might occur, and once a reset was asserted, it would last until the power was shut down.

4 External Interface

4.1 Description

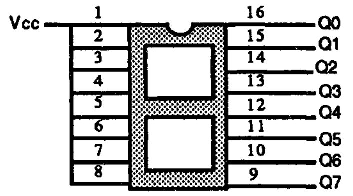
The External Interface area of the Transputer Education Kit consists of a Programmed I/O port, an LED port and two optional power transistors. The Programmed I/O port provides a byte-wide I/O port for hardware interfacing. The LED port can be populated with light-emitting diodes (LED's) for use as a software debugging aid, or as a set of 8 control signals in conjunction with the I/O port for a more versatile hardware interface. Power connections are included on the Programmed I/O port and the LED port. This eliminates the necessity of an extra power supply for hardware interfaces with modest current requirements (< 500 milliamps). The two power transistors provide a modest switchable power source for interfaces attached via a link.

There are two basic ways to interface external hardware with the Transputer Education Kit. The first method uses the Programmed I/O port signals and the LED signals when needed. This provides a versatile, memory-mapped hardware interface to the transputer. Power connections are brought to these ports so that for many projects, no external power supply is needed. See section **4.2** for examples using this interface. The second method of interfacing is to communicate with the interface circuitry through a link cable. An Inmos C012 link adaptor chip can be used to convert a serial link into a byte-wide output port and a byte-wide input port with associated control signals. Installing the optional power transistors provides up to 250 milliamps of current to a hardware interface connected through a link cable. Section **4.1.3 Power Transistors** explains how power is sent along the cable. The **Workbook** included as part of the Transputer Education Kit contains several interface examples that use a link and a link adaptor chip.



4.1.1 LED Port

The LED port consists of 8 transputer writeable memory-mapped signals Q0-Q7 and 8 Vcc connections. These are made available to the user through a 16 pin DIP socket having the pinout shown below. 8 LED's (available from CSA) can be plugged directly into this socket and used as a debugging aid. These signals can also be used in conjunction with the Programmed I/O port to provide a more versatile hardware interface.



LED Pinouts

Indicator lights that can be turned on or off under program control are useful in debugging parallel programs. They serve as an alternative to displaying information on the screen. In most transputer networks only one transputer has a direct connection to the screen and displaying information from the other transputers can sometimes be difficult if not impossible. The LED's can be used by the programmer to indicate the status of program execution on the board. When multiple processes are running on one transputer, often only one process has direct access to the screen. The LED's can be used to display information from the processes without access to the screen. As you experiment and gain experience you will find other uses for these LED's.

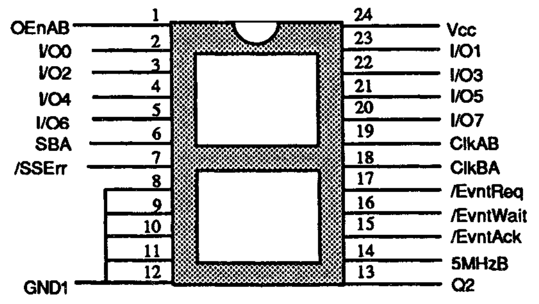
The LED signals can be used in conjunction with the Programmed I/O port to provide a very versatile hardware interface that can control most byte-wide microprocessor bus peripheral chips as well as many other devices. Sections **4.2.2** and **4.2.3** contain examples of an A/D converter and a D/A converter that use the LED port in conjunction with the Programmed I/O port. Appendix **A.2.4 External Interface LED Port and Power Transistors** tells how to write this port in software. Sample C and Occam procedures illustrating how to control the LED signals are provided on the Examples disk.

IMPORTANT: Q0 and Q1 perform double duty, in that they are the same memory mapped latches that are used for SS system services reset and analyse signals (although inverted). Also Q2 doubles as the reset signal for the Programmed I/O Port. Keep this in mind if you use SS (see section **3.6** and **A.2**) or the Programmed I/O Port in conjunction with the LED signals.

4.1.2 Programmed I/O Port

The Programmed I/O port provides a byte-wide bidirectional parallel port that can be used for input and/or output to external circuitry provided by the customer. Most of the port signals are generated and received by a 74LS652 chip (not included with the Transputer Education Kit). It is available from CSA as well as from electronic component vendors. The signals on the Programmed I/O port not connected to the 74LS652 are connected to circuitry included on all Transputer Education Kit configurations.

The Programmed I/O Port socket (the empty 24-pin socket along the top edge of the board) makes available the following signals: a) 8 shared data I/O bits going to and from the transputer via the 74LS652 chip; b) 4 status and control signals related to the 74LS652; c) 3 transputer event signals (event being the transputer's equivalent of an external interrupt); d) a 5 MHz clock; e) 2 transputer programmed I/O signals (one coming from the transputer and the other going to it); and f) 6 conductors carrying power and ground (1 power and 5 ground).



Programmed I/O Port Pinouts

The pinout for this port is shown in the diagram above. A brief description of the signal on each pin of the Programmed I/O port socket is given below. Following these discriptions are short discussions on how the Programmed I/O port could be used. See Appendix **A.2.5 External Interface Programmed I/O Port** for details on reading from and writing to the I/O port from the transputer. See Appendix **A.3 Board Schematics** for more details on the hardware implementation of the Programmed I/O port.

74LS652 Octal Bus Transceiver/Register Signals

The 74LS652 is an octal bus transceiver/register chip. This symmetrically bidirectional device consists of bus transceiver circuitry, a pair of D-type registers (one for each direction), and control circuitry. As configured in the Transputer Education Kit, the customer's external circuitry can, at any point in time, read data written into the 652's "output" register by the transputer, write data into the 652's "input" register for later reading by the transputer, or present data to the transputer via the 652 in write-through mode (i.e. bypassing the "input" register). For more information on the 652, consult the block diagram in Appendix A.3 and the data sheet for the 74LS652 (included with the chip when ordered from CSA).

I/O0-7 (pins 2, 23, 3, 22, 4, 21, 5, 20): The 652's eight data I/O bits. Made available to the customer's circuitry such that it can pass byte-wide data back and forth to and from the transputer. These lines can be used in only one direction at a time as controlled by OEnAB.

OEnAB (pin 1): Controlled by the customer's circuitry. This signal determines the direction of the I/O port. When asserted high this signal enables output of the 652's eight data I/O bits such that the data written by the transputer into the 652's "output" register is presented to the customer's circuitry. When held low, this signal causes the 652 to disable data outputs (i.e. it takes these signals to a high impedance state) allowing the customer's circuitry to present data to the 652 so it can be read by the transputer.

ClkBA (pin 18): Controlled by the customer's circuitry. The rising edge of this signal strobes data generated by the customer's circuitry into the 652's "input" register. Prior to taking this signal from low to high, the customer's circuitry should have taken OEnAB low and then placed the desired data onto the 652's data I/O pins, I/O0-7. Note that the transputer has no way of detecting that this signal has been asserted. The customer's circuitry must notify the transputer in some other way, possibly by using the EvntReq or the /SSErr signals.

ClkAB (pin 19): Controlled by the transputer. When the program running on the transputer attempts to write a byte to the 652's output register, this normally-high signal is momentarily taken low (for 1/5 - 4 processor cycles depending on memory speed selection which results in a pulse width of 75 - 200 nanoseconds when using a 20 MHz processor). This signal is brought to one of the pins on the Programmed I/O Port socket so that the customer's external circuitry can detect when new data has been made available to it by the transputer.

SBA (pin 6): Controlled by the customer's circuitry. When held low this signal causes data made present on the 652's 8 I/O pins by the customer's circuitry to bypass the 652's "input" register. In this case, when the transputer reads a byte from the 652, it will be reading data directly from the customer's circuitry via the data lines I/O0-7. If this signal is held high, when the transputer reads a byte from the 652, it will be reading the data last loaded into the 652's "input" register by the customer's circuitry.

Transputer Event Signals

The Event signals provide an asychronous handshake interface between an internal transputer process and external circuitry. The Event signals are used to implement the transputer's equivalent of an external interrupt. They can also be used to perform handshaking for data transfer via the Programmed I/O port. For more information on the Event signals see p. 58 of the Transputer Technical Specifications in the **Transputer Databook.**

/EvntReq (pin 17): Controlled by the customer's circuitry. /EvntReq is the complement of the transputer's EvntReq signal. This is equivalent to an external interrupt signal. When asserted low by the customer's circuitry, this /EvntReq signal will present an Event request to the transputer, which will cause a transputer process which performs an input on Event to be rescheduled. This signal should only be asserted low if /EvntAck is in its quiescent high state, and should then only be returned to high upon /EvntAck being asserted low.

/EvntAck (pin 15): Controlled by the transputer. /EvntAck is the complement of the transputer's EvntAck signal. This /EvntAck signal will be asserted low by the transputer upon rescheduling of a transputer process which was suspended after performing an input on Event (i.e. following assertion of /EvntReq by the customer's circuitry). Once asserted low by the transputer, /EvntAck will remain low until deassertion of /EvntReq by the customer's circuitry.

/EvntWait (pin 16): Controlled by the transputer. /EvntWait is the complement of the transputer's EvntWait signal. This /EvntWait signal will be asserted low by the transputer whenever a transputer process performs an input on Event. It will return to its quiescent high state following the assertion by the customer's circuitry of /EvntReq, as soon as the process which performed the input on Event is rescheduled for execution. The EvntWait signal is not implemented on earlier transputer models such as the T414 and the T800. The T400, T425 and T805 do, however, implement EvntWait.

Miscellaneous Signals

Q2 (pin 13): Controlled by the transputer. This is just a copy of the LED Port's Q2 signal; brought to the Programmed I/O Port for convenience. The transputer can cause this signal to go high or low by writing a 1 or a 0, respectively, to the appropriate bit of the LED Port.

/SSErr (pin 7): Controlled by the customer's circuitry. Like Q2, /SSErr does double duty, and derives its signal name from its other duty as the Error signal for the transputer's SS system services. See section **3.6 Setting Up Multi-Transputer Networks** for more info on system services. /SSErr is a single-bit signal which can be controlled by the customer's circuitry, and read by the transputer. It should be noted that the use of /SSErr in this context will result in a conflict if SS System Services is also utilized.

5MHzB (pin 14): This signal is the complement of the crystal oscillator-generated clock used by the transputer, and is made available to the customer's circuitry through the Programmed I/O Port as a convenience.

Vcc (pin 24): +5vdc derived from the PC host via the Kit circuit board. Capable of supplying several hundred milliamps, if necessary, to power the customer's circuitry.

Gnd (pins 8-12): Digital ground, or common, derived from the PC host via the Kit circuit board. Provides an electrical reference for the Programmed I/O Port signals, but can also be used in conjunction with Vcc to supply power to the customer's circuitry.

Using the Programmed I/O Port

Although the Programmed I/O port can be used to interface to external hardware in a wide variety of ways, both by itself and in conjunction with the LED port, the discussion here will be limited to three methods of interfacing that illustrate the basic use of the port. Two of these methods use the Programmed I/O Port by itself and the third uses it in conjunction with the LED port signals. The examples in section **4.2** illustrate these three methods.

The first interface method employs I/O0-7, OEnAB, ClkBA, ClkAB, SBA, Q2, and /SSErr (i.e. all of the Programmed I/O Port signals except those dealing with Event). Q2 might be used to reset the customer's circuitry under transputer program control. I/O0-7, of course, carry the data to be transferred. OEnAB determines the direction of data flow, and when directed toward the transputer, ClkBA and SBA determine the manner in which the data flows through the interface (i.e. registered or direct). Finally, ClkAB and /SSErr can be used to indicate when data is available in one direction or the other (i.e. for handshaking). Data transfer is one way only unless the customer's circuitry contains enough intelligence to interpret direction change commands sent as data. Section **4.2.4** illustrates this type of interface.

The second interfacing method is similar to the first, but uses the transputer's Event lines for handshaking during data transfer. As with the first method, Q2 might be used to reset the external circuitry, I/O0-7 would be for the transfer of data, and OEnAB, ClkBA and SBA would determine the direction and manner of data transfer. The signaling of data availability (i.e. handshaking) would be handled using the Event signals, leaving /SSErr as a status or flag bit. The use of ClkAB is not required. Section **4.2** illustrates this type of interface.

The third method adds the LED Port signals to the Programmed I/O Port to emulate a byte-wide microprocessor interface. This method results in there being 8 data bus lines (I/O0-7), 8 lines for address, chip select, and read/write signals (Q0-7 of the LED Port), and one status line (/SSErr). Using this method, SBA is tied low, and ClkAB and ClkBA are not used, and OEnAB could be tied to one of the LED Port bits. Section **4.2.2** and **4.2.3** provide examples of this type of interface.

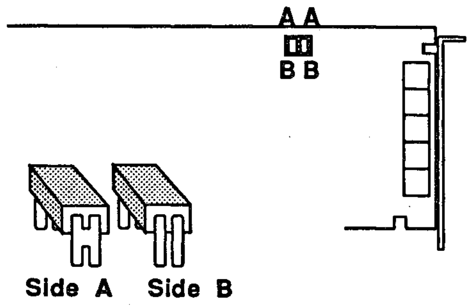
4.1.3 Power Transistors

As mentioned in the introduction to this chapter, there are two methods of implementing hardware interfaces with the Transputer Education Kit board. The first uses the Programmed

I/O port and the LED port signals as described in sections **4.1.1** and **4.1.2** and illustrated with the examples in section **4.2 Examples**. Power for these projects is available through the socket connector eliminating the need for an external power supply. The second method of interfacing provides communication with the external hardware using a link. Examples illustrating this method of hardware interfacing are given in the **Workbook** included with the Transputer Education Kit.

Circuitry is included on the Transputer Education Kit board which allows one of the signal wires in the link cable to provide power to an interface connected through a link cable. Two power transistors which are needed for this circuitry to function are not provided in the entry level kit but are available from CSA or electronic component vendors. The power provided by this circuitry can be turned on and off under program control.

The power transistors are controlled by the system services SS analyse signal which in turn are controlled by Q1 of the LED port signals (see section **4.1.1 LED Port**). Writing a 0 to Q1 turns the power on and writing a 1 turns the power off. Depending on the output voltage of your PC's power supply, this power source can provide between 200 and 250 milliamps while still keeping the voltage at the end of the link cable above 4.75 Volts. Two procedures to switch the power on and off are provided on the Examples disk.

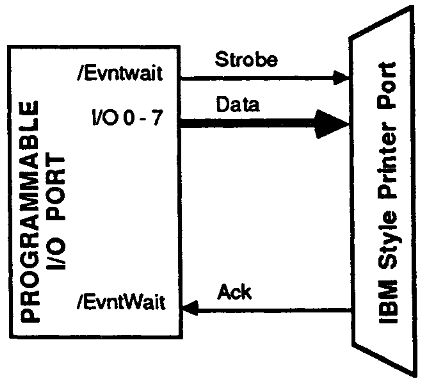


Each power transistors (Part # IRFD9123) comes in a 4 pin package. On one side of the Transistor, two pins are connected together. This connection serves as a heat sink and also as the marker indicating chip orientation. The diagram above identifies the two sides of the transistors as A and B and shows how they are placed in the power transistor socket.

4.2 Examples

4.2.1 Printer Interface

This design interfaces a printer to the Programmed I/O port using a subset of the Centronics parallel printer interface standard, which is used by most parallel dot matrix printers including Epson and IBM. This interface provides the transputer with direct access to a standard printer device.



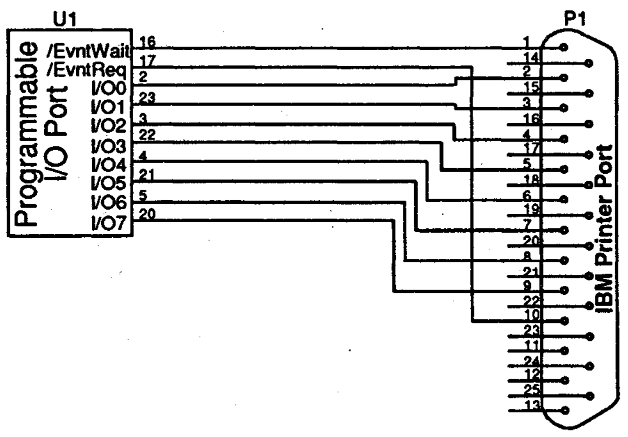
Parallel Printer Interface Block Diagram

The protocol used to send data to the printer is described here in a non-technical manner. Most printer manuals have a section explaining this protocol in more detail. The controlling device writes the data to the eight I/O pins. It then takes the Strobe line low to signal the printer that the data is ready to be read. When the printer has read the data, it sends a low pulse on the Ack line indicating to the controlling device that it has the data and the cycle is complete.

The interface described in this example implements the communication protocol using the Programmed I/O port. The transputer program writes the byte to be sent to the Programmed I/O port. It then performs an input on the Event channel. The transputer program will wait here until /EvntReq goes low. When the transputer is waiting for an input on the Event channel, it holds /EvntWait low. This takes the printer's Strobe line low, indicating to the printer that the data is ready. When the printer has read the data, it takes Ack low. Ack is connected to /EvntReq which also goes low. This allows the Event channel input to complete, which takes EvntWait and Strobe high and completes the protocol.

Although intended to provide simple access to a printer, this interface could be used to make print spoolers or provide graphical output on a printer with such capabilities. There are sample programs in both Occam and C on the Test and Demo disk which illustrate how to send data to the printer. No extra IC's are involved in this interface; only a cable with the proper end-connections. The simplest cable has a 24 pin DIP header on one end and a parallel printer port connector on the other end. Another option, if the cable must reach more than one or two feet, is to make a short cable with a 24 pin DIP header on one end and a PC-type 25 pin connector on the other end and then plug this into a standard PC printer cable.

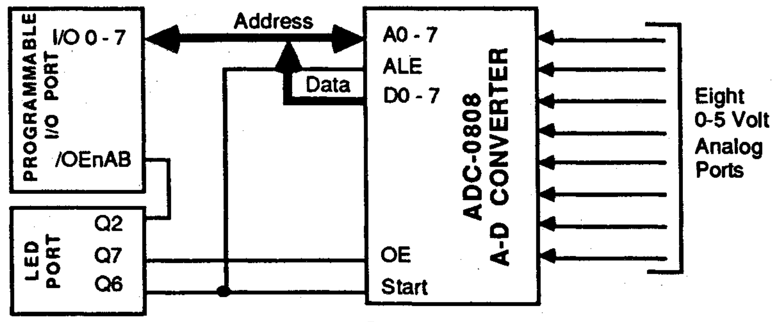
Sample programs showing how to send characters to the printer are provided in both Occam and C on the Test and Demo disk.



Parallel Printer Interface Schematic

4.2.2 Eight Channel A to D Converter

This project uses the Programmable I/O port together with the LED port signals to control an 8 bit, 8 channel analog to digital converter. With this A/D circuit the transputer can read up to 8 different voltages that are between 0 and 5 volts.



Eight Channel A/D Converter Block Diagram

The National Semiconductor ADC0808 is used for the actual analog to digital conversion. The ADC0808 is an A/D converter with additional bus-interface logic to simplify interfacing to an 8 bit microprocessor. This interface uses the Programmed I/O port as an 8 bit bus and Q2, Q6 and Q7 of the LED port to provide the control signals.

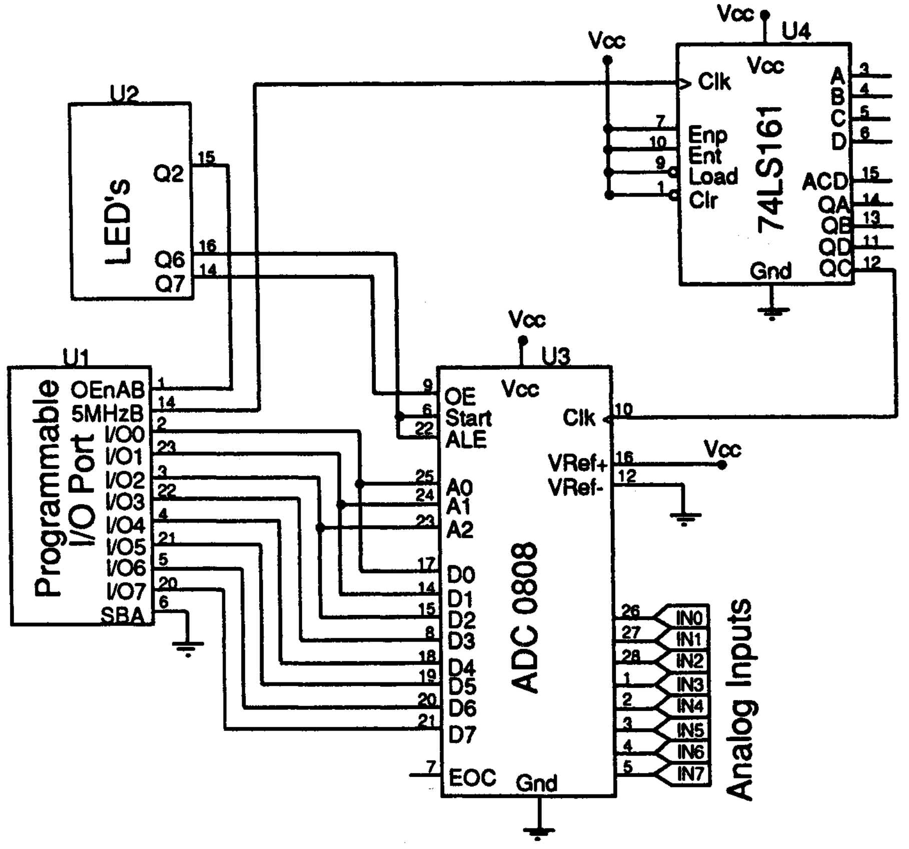
The following process converts an analog signal to a digital value and reads it. Q2, Q6 and Q7 are low at the beginning of the conversion. Q2 is taken high, setting the Programmable I/O port for output. The channel number to be read is written to the Programmable I/O port. Q6 is taken high to latch the channel number into the ADC0808 and start the conversion cycle. Q6 is then taken low and the transputer uses its internal timer to wait while the conversion takes place (at least 120 µs). When the conversion is complete, Q2 is taken high to set the Programmable I/O port for input. Q7 is taken high enabling data from the the ADC0808 onto the Programmable I/O port. The value is read from the Programmable I/O port. Q7 is taken low to disable the output of the ADC0808, and everything is ready for another conversion.

Sample programs for the A/D converter circuit are provided in both Occam and C the Test and Demo disk.

The National ADC0808 has no sample and hold capability, so is only accurate for slowly changing signals. The pin-compatible Texas Instruments ADC0808 contains on-chip track and hold circuitry which allows it to digitize signals up to 4 KHz. This is enough to do simple experiments with voice digitizing. You will notice that the Programmable I/O and LED port signals are used in a way that allows this project to be combined with the digital to analog converter in section **4.2.3 Two Channel D to A Converter** to form a simple combination D/A, A/D conversion system.

Parts List

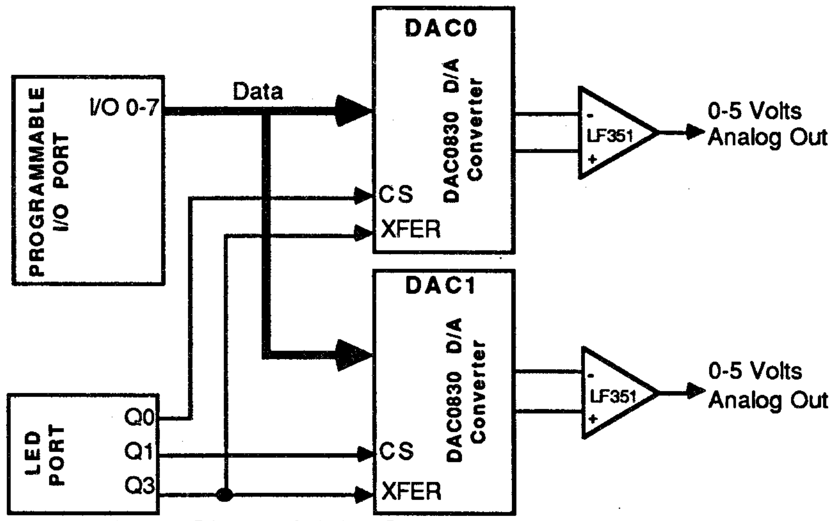
Qty.	Description
1	National ADC0808 8 Channel A/D Converter
1	74LS161 4 Bit Synchronous Counter



Eight Channel A/D Converter Schematic

4.2.3 Two-Channel D to A Converter

This project uses the Programmable I/O port together with the LED port signals to control two 8 bit digital to analog converters. This interface gives the Transputer Education Kit board two analog outputs that can be varied between 0 and 5 volts.



Two Channel D/A Converter Block Diagram

The National Semiconductor DAC0830 is a fast (1 μs conversion time) 8 bit D/A converter with an 8 bit microprocessor interface and internal latches to allow for simultaneous updating of more than one DAC0830. This interface uses the Programmed I/O port as an 8 bit bus and Q0, Q1 and Q3 of the LED port to provide the control signals.

There are two steps to to writing a new value to the D/A converters (DAC). In the first step, CS (Chip Select) is used to place a byte in an internal latch. At this point, the output of the DAC is the same as it was before the byte was written to the internal latch. Next XFER is taken high which moves the byte in the internal latch into the DAC output register. After the transfer, the DAC's output reflects the new value. The two steps are necessary so that the output of two or more DACs can be updated simultaneously. To do this, the new bytes for all DACs are first written to the internal latch. XFER is then taken high and low to update the outputs simultaneously. The LF351 op amps convert the current output of the DACs to a voltage output.

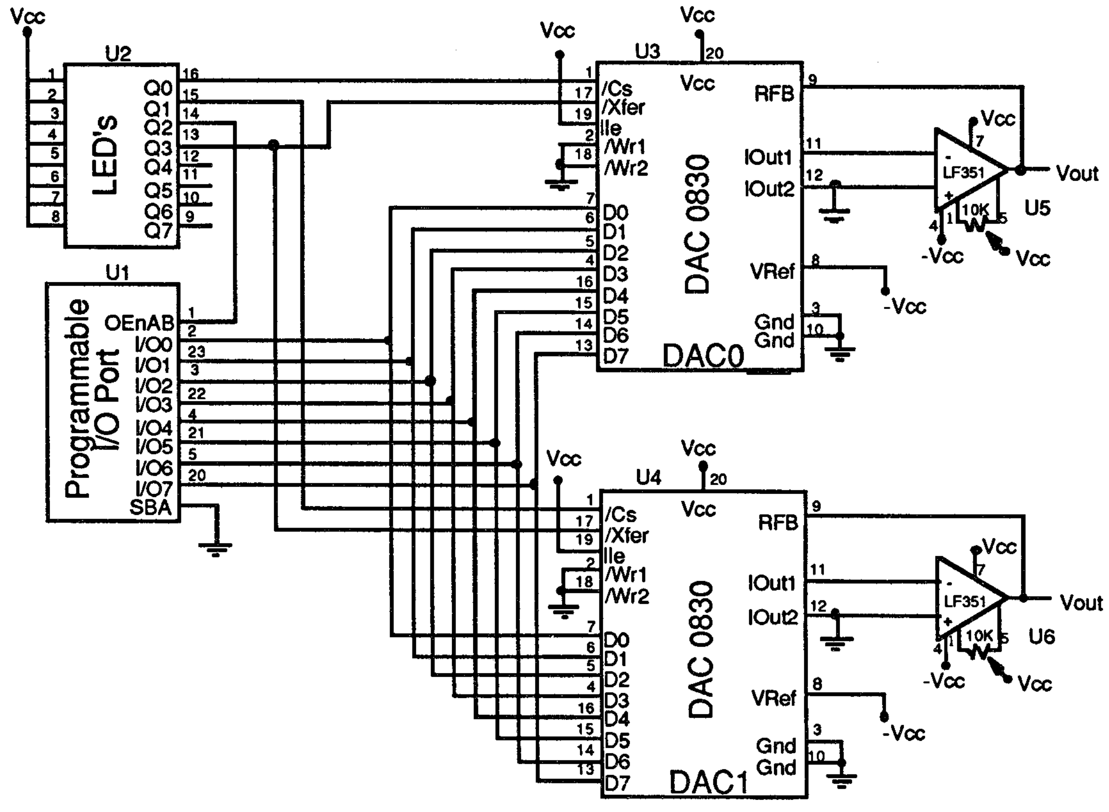
To update both DACs, start with the control signals Q0, Q1 and Q3 low. Write the byte for DAC0 to the I/O port. Then take Q0 high to select DAC0 and write the byte on the I/O port to the DAC's internal latch. Take Q0 low to deselect DAC0. Write the byte for DAC1 to the I/O port. Then take Q1 high to select DAC1 and write the byte on the I/O port to the DAC's internal latch. Take Q0 low to deselect DAC1. Take Q3 high to transfer the bytes from the internal latches on both DACs to the output registers. The output voltages will then reflect the new values that were written.

Sample programs for the D/A converter in both Occam and C are provided on the Test and Demo disk.

This interface gives the transputer access to two fast (1μs) 8 bit D/A Converters. It can be used in music synthesis, signal generation, control and many other applications. It is designed to accept the pin-compatible National Semiconductor DAC1230 12 bit D/A Converter with slight modifications. One more control line must be provided for the Converters, and the software must be changed. See the Data Sheet of the DAC1230 for more detail. You will also notice that the Programmable I/O and LED port signals are used so that this project can be combined with the analog to digital Converter in section **4.2.2 Eight Channel A to D Converter** to form a simple combination D/A, A/D conversion system.

Parts List

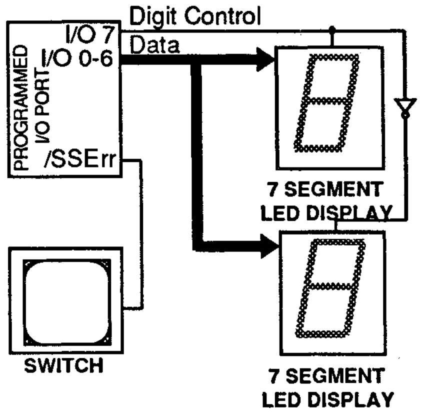
Qty.	Description
2	National DAC0830 8 Bit Converter
2	LF351 JFET Operational Amplifier
2	10KΩ Potentiometer



Two Channel D/A Converter Schematic

4.2.4 Two-digit LED Display

This project uses the Programmable I/O port to control two seven segment LED displays and to read a switch. It might be used as a debugging aid with more functionality than is available using LED's plugged into the LED port. The simple parallel program which controls this display illustrates a simple use of the transputer's on-chip timer.



Two-Digit LED Display Block Diagram

The hardware interface is quite simple. I/O0-I/O6 are connected to the seven segments of both LED Displays through current limiting resistors. I/O7 turns on one digit of the display after the other in rapid succession. The values on I/O0- I/O6 are changed each time I/O7 is changed, so that a different value is displayed on each digit. The switching is fast enough that both digits appear to be continually lit.

The switch is connected to SSErr which is read by the transputer. SSErr is normally held high by a pull-up resistor. Pressing or toggling the switch connects SSErr to ground.

The software to use this display contains a parallel process that does nothing but update the display. This display process creates a byte that contains the correct segment values for the low order digit in bits 0-6 and the value in bit 7 set to turn on the low order digit display. This byte is written to the I/O port which displays the low order digit on the LED display. Using the transputer timer, the display process waits for 5 milliseconds then creates a byte for the high order digit in the same manner as it created the byte for the low order digit. This byte is then written to the I/O port which displays the high order digit on the LED display. The display process waits 5 more milliseconds then starts all over by creating the byte for the low order digit.

Each time the display process turns off one digit and turns on the other, it checks to see if there is a new value to display. The main program runs in parallel with the display process. It sends the byte to be displayed to the display process and then continues execution. The display process uses very little of the transputer's processing power and requires no intervention from the main program except initial setup.

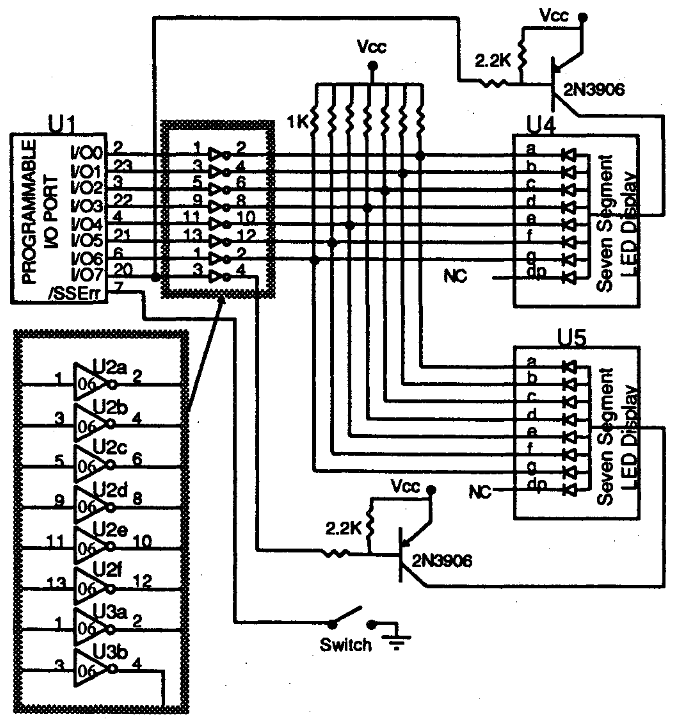
The switch gives a way for a program to wait for outside input before continuing. This is useful when you want to see the value on the display before a new one is written. To pause until the switch is closed the program reads and checks SSErr every 20 milliseconds until it goes low.

The program then waits 20 milliseconds to allow the switch to stop bouncing and then reads SSErr again. If it is still low, it is assumed that the switch has been closed and program execution continues. The inclusion of the switch in this interface allows a program to write a byte to the display and then pause until the programmer closes the switch.

Sample programs illustrating the use of the two-digit LED display are provided in both Occam and C on the Test and Demo disk.

Parts List

Qty.	Description
2	Common Anode LED Displays
2	3906 NPN Bipolar Transistors
2	7406 Open Collector Hex Inverter
4	2.2K Bias Resistors
8	1K Current Limiting Resistors
1	SPST Switch



Two-digit LED Display Schematic

5 Troubleshooting

This chapter consists of two sections. Section **5.1 Simple Solutions** contains suggestions to help resolve some commonly encountered difficulties. Section **5.2 Step by Step Diagnostics** contains directions for testing the board and software one element at a time, isolating the problem and suggesting possible solutions. When the suggested solutions don't help you fix the problem, the results of the diagnostics provide CSA Technical Support with the information necessary to help you further. To help us better serve you, please have the results of the diagnostic tests in section **5.2 Step by Step Diagnostics** ready when you call CSA Technical Support at 1-800-753-4CSA.

It is important to keep things simple when you are troubleshooting. Stay with the default settings whenever possible. Get the board to work with the fewest possible changes from the default, then add changes one by one. If you are using Add-On-Processor boards, get the Transputer Education Kit board working first, then add extra boards one at a time.

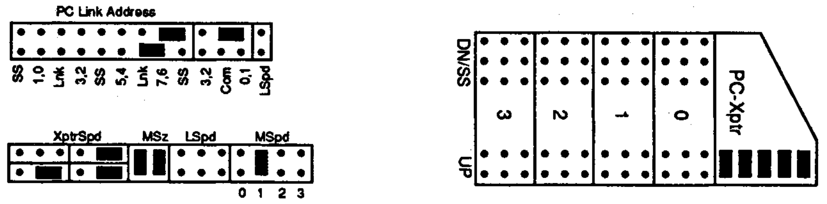
IMPORTANT: To avoid damaging the Transputer Education Kit board or your PC, always turn off the power before changing any settings or removing any boards.

5.1 Simple Solutions

This sections contains suggestions that will resolve many difficulties encountered while installing or modifying the hardware or software in the Transputer Education Kit. If your problem is related to one of the following categories, try the solutions listed here; if not, go to section **5.2 Step by Step Diagnostics**.

5.1.1 Installing the Board

If you've just installed your board and it doesn't work, check the jumpers for the correct settings. The settings made at the factory (shown in the diagram below) will work in most cases. If your installation requires changes, start by making only the changes necessary for your board to work in your system. Add other changes one by one, running `kit_test` (on Test and Demo disk 1) after each change until you isolate the problem.

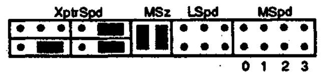


Default Jumper Settings

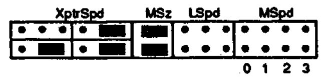
If your board still doesn't work, try putting it in another slot. If this doesn't solve your problem, go to section **5.2 Step by Step Diagnostics.**

5.1.2 Installing the Memory

If you have installed memory in your board and it fails the external memory test in `kit_test`, check that all chips are correctly oriented in their sockets (see the diagram in section **3.2 Adding Memory**). Next check the memory for pins bent during the installation. Pins sometimes bend under the chip where they are easily overlooked. Double-check the memory size and speed jumper blocks for appropriate settings.



Jumper settings for 1 Mbyte



Jumper settings for 2 or 4 Mbytes

Set the memory speed in the MSpd jumper block as indicated by the table in section **3.2.2 Memory Speed**

If these steps don't solve your problem, go to section **5.2 Step by Step Diagnostics.**

5.1.3 Software

The most common cause of software malfunction is incorrectly configured hardware. If you have successfully worked through section **2.2 Hardware Installation**, refer to section **5.1.1 Installing the Board.**

The second most common cause of software malfunction is incorrect environment variables. Review section **2.4 Software Installation** if you are using the default PC/Link Interface address and have installed the software as outlined in chapter **2 Installation Guide.** If you are using a PC/Link Interface address other than the default or have modified the installation process, review the file `install.doc` on disk 1 of the software installation disks for more information on environment variables.

If these suggestions don't solve your problem, go to section **5.2 Step by Step Diagnostics.**

5.1.4 Miscellaneous

If you have problems with your Transputer Education Kit board after recently installing another board in your system, there may be an address conflict. Check the documentation for the new board to see which I/O addresses it uses. If I/O addresses for both boards are in the same range (default for the Transputer Education Kit board is 150-170h), the addresses for one of the boards must be changed. Section **3.3.1 PC/Link Interface Address** explains how to change the Transputer Education Kit address. If you do change the PC/Link Interface address, run `kit_test` following the instructions in section **5.2.1 PC/Link Interface** to assure you the address selected is the one you intended.

5.2 Step by Step Diagnostics

The following section uses the `kit_test` program on the Test and Demo disk to exercise various sections of the board individually. The tests in `kit_test` build upon each other and should be thus run in the sequence indicated below. If problems persist after working through the diagnostic tests, call CSA Technical Support (1-800-753-4CSA) with the results of these tests.

5.2.1 PC/Link Interface

All communication between the PC and the transputer is through the PC/Link Interface. If the PC/Link Interface is not functioning correctly, nothing else will work. To check the PC/Link Interface operation insert the Test and Demo disk in the floppy drive and type the following at the command line (if you are not using floppy drive a, replace a with the correct drive letter):

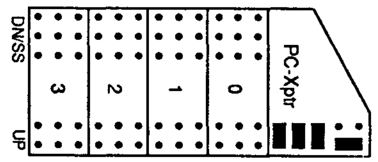
```
a:kit_test
```

You should see the following display on your screen:

```
CSA KitTest Version 1.00
Installation and Diagnostic tests for the
Transputer Education Kit.

PC/Link found at 150 (hex) SS found at 160 (hex)

1)  Get warranty number
2)  Show/Select PC/Link Interface address(es)
3)  PC/Link loopback test
4)  Channel I/O test
5)  On-chip memory test
6)  External memory test
7)  Show/Set/Clear error flag
0)  Quit
```

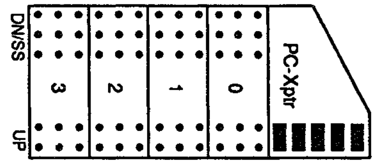
If the addresses indicated in line four are not the addresses you intended, check the PC/Link Interface address jumper settings. Section **3.3.1 PC/Link Interface Address** explains address selection. After you are sure the jumpers are correct, run the test again; if addresses indicated in line four are correct, continue on; if they are not, call CSA Technical Support.

Run the PC/Link loopback test in `kit_test` to make sure the PC/Link is sending and receiving data correctly. To run this test, the jumpers in the PC-Xptr jumper block should be shunted as shown in the diagram below. Turn the PC off when making the indicated changes.



Jumper settings for PC loopback test in `kit_test`

Turn the PC back on and run `kit_test`. Select the loopback test by typing 2 at the prompt. The results of the test will be displayed on the screen. If the board passes the loopback test, turn the PC off and restore jumper block PC-Xptr to its original configuration. Turn the PC back on and continue with the next section. If the board fails the test, call CSA Technical Support.



Jumper block PC-Xptr preconfigured settings

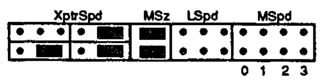
5.2.2 Transputer

This section tests the transputer. Run `kit_test` and type 4 at the prompt to select the Channel I/O test. This tests communication between the PC and transputer. If the board passes the Channel I/O test, continue on. If it fails the Channel I/O test, check that the PC Link speed matches the transputer Link 0 speed. Sections **3.3.2 PC Link Speed** and **3.4.2 Link Speeds** show how to select the link speed. When you are sure the link speed settings are correct, run the Channel I/O test again. If the board passes, continue on; if not, call CSA Technical Support.

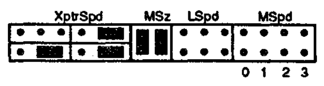
Run `kit_test` and type 5 at the prompt to select the On-chip memory test. This tests the transputer's 2 or 4K of on-chip memory. If on-chip memory passes the test, continue on. If it does not, call CSA Technical Support.

5.2.3 Memory

This section tests the external memory on the Transputer Education Kit board. Run `kit_test` and type 6 at the prompt to select the External memory test. This tests the external memory and identifies any defective chip(s) by their socket number (see section **3.1 Board Layout**). If the memory passes the test, continue on to the next section. If the memory fails the test, check the memory size and speed jumper settings using the diagram below as a guide.



Jumper settings for 1 Mbyte



Jumper settings for 2 or 4 Mbytes

Set the memory speed in the MSpd jumper block as indicated by the table in section **3.2.2 Memory Speed**

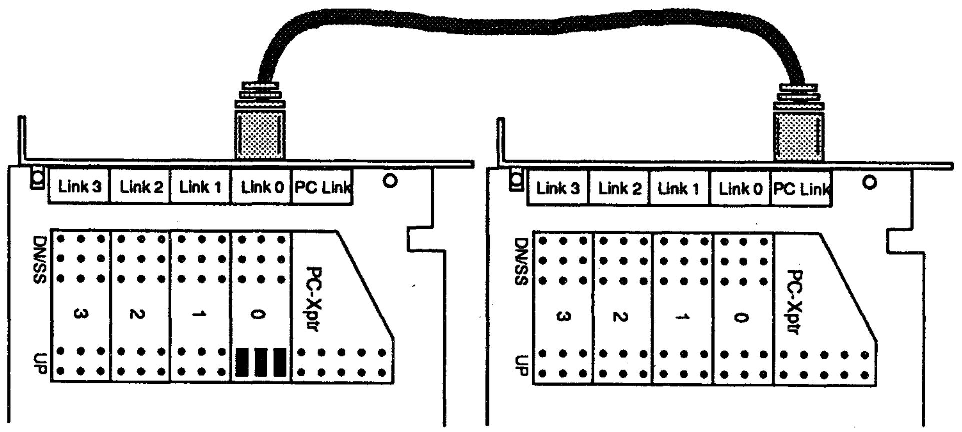
When you are sure the memory jumpers are correct, run the External memory test again. If results are positive, continue onto the next section; if they are not, check the indicated memory chips for pins bent during installation and for proper chip orientation (see section **3.2 Adding Memory**). If no problems are found during a visual inspection, swap the indicated defective memory chip with another memory chip and run the External memory test again. If the error follows the swapped chip, replace it with a new one. If the test finds the same socket number defective, call CSA Technical Support.

5.2.4 Software

If you have isolated and solved any problems while working through the hardware diagnostic section, you may well have solved your software problems. If the problems persist, check the environment variables. Review section **2.4 Software Installation** if you are using the default PC/Link Interface address and have installed the software as outlined in chapter **2 Installation Guide**. If you are using any PC/Link Interface address other than the default or have modified the installation process, review the file `install.doc` on disk 1 of the software installation disks for more information on environment variables. If these suggestions don't solve your problem, call CSA Technical Support.

5.2.5 Add-On-Processor Boards

To test a single Add-On-Processor board, connect its transputer to the PC/Link Interface of your Transputer Education Kit board using the jumper settings and cables as indicated in the diagram below. You can then run the tests outlined in sections **5.2.2-5.2.3** to test the hardware of the Add-On-Processor board.



Add-On-Processor board **Transputer Education Kit board with PC/Link Interface**

To connect an Add-On-Processor board to the PC/Link Interface

5.2.6 Multi-Board Networks

If you are having difficulties setting up or debugging a network of transputers, test each board in the system individually. Test all Transputer Education Kit boards as explained in sections **5.2.1-5.2.3**. Test all Add-On-Processor boards as explained in section **5.2.5**. When all boards pass these preliminary tests, proceed with the network setup.

Use the Check utility described in section **3.6 Setting Up Multi-Transputer Networks** to test the network connections. Start with the root board, then add other boards one at a time, running the Check utility after connecting each additional board, until the network is fully functional.

6 C In On-Chip Memory

6.1 The Basics

The compiler in the C Toolset is what is called a cross-compiler. That is, the compiler itself runs on the PC host, but generates code for the transputer. (This is not the case with the Occam Toolset compiler wherewith the compiler itself must be run on a transputer.) If you program in C, you could, in fact, perform all the early stages of code development without even having the transputer installed. Only when you are ready to load, execute and debug, would you actually make use of it.

Once a C program has been compiled and downloaded by the PC host to the transputer (or to a network of them) the job of the PC is not over. While the transputer(s) are busy computing, the PC must be available as an I/O device, since without the PC, the transputer has no means of accessing a display, keyboard or disk files. Thus every PC-based transputer application runs with complementary code executing concurrently on both the PC and the transputer(s). At the very least, the PC runs a simple I/O "server" program (several versions of which are supplied with the Kit software). The server allows the root transputer (the one attached to the PC) to execute what appear to be standard I/O functions. However, rather than directly accessing I/O devices, these functions pass special I/O request messages over a transputer link (via the Kit's PC/Link Interface) to the server program, which in turn actually performs the I/O.

The PC could, of course, run an application-specific program – such as an interactive graphical front end – which communicates with the transputer programs using an application-specific message protocol. In this case, the program in the root transputer would entirely avoid the use of the standard I/O functions.

The transputer's standard C I/O functions operate in conjunction with a server called cio. This set of functions is very complete (adhering to ANSI standards), and as a result, the transputer-resident portion of the code occupies from 6K to 8K of transputer memory. Since transputer on-chip memory is limited to from 2K (for the T400) to 4K (for the T425 or T805), the Transputer Education Kit circuit board cannot support standard C I/O unless off-chip memory is installed.

A special cut-down version of C I/O, called the Lt I/O Library, has been provided as an alternative, however, so that purchasers of Kits with no off-chip memory installed can productively use and experiment with the transputer. With Lt I/O (which stands for Link-terminal I/O) the user can perform screen and keyboard I/O functions which are similar to their standard C I/O counterparts, but require very little transputer memory. Lt I/O also contains simplified heap management (i.e. dynamic memory allocation) procedures.

So if your Transputer Education Kit was purchased with no off-chip memory installed, and if you haven't already installed some yourself, you can still get started, using C and the Lt I/O

Library. Lt I/O is small enough to coexist with a hundred or so lines of C code, even within the 2K limit of the T400 on-chip memory.

The remainder of this chapter contains a description of the LT I/O Library functions, as well as instructions on how to selectively divide the transputer's on-chip memory between stack, code and heap, and helpful hints for conserving memory when running parallel processes.

6.2 The Lt I/O Library

Sections **6.2.1-6.2.3** contain examples illustrating the Lt I/O Library procedures. Section **6.2.4 Lt I/O Library Reference** contains an in-depth description of each procedure. The batch files `ltlsc4` and `ltlsc8` are included with the C Toolset and should be used when compiling programs to run exclusively in on-chip memory.

To load a program which uses the Lt I/O Library onto a single transputer, you should type the following command line:

```
ld-one foo ltio
```

where `foo` represents the name of the program which you wish to load.

6.2.1 Output Using the Lt I/O Library

The Lt I/O Library output functions, lt_putch, lt_puts and lt_printf, are similar to their standard C counterparts, putch, puts and printf. Below is a program illustrating their use:

```
#include <ltio.h>

main()
{
      int i;

      lt_printf("\nHere are the lt output procedures");
      lt_puts("\nThis puts a string on the screen");
      lt_puts("\nNow let's print five characters");
      for (i='a'; i< 'f'; i++)
            lt_putch(i);
      lt_printf("\nHere are all the possible lt_printf combinations");
      lt_printf("\nA string %s, two characters %c%c",
            "This is a string", 'r', 's');
      lt_printf("\nThe number 47 printed in decimal %d, octal %o",
            47, 47);
      lt_printf("\n unsigned decimal %u and both hex formats %X, %x",
            47, 47);
      lt_printf("\nFinally we print a void pointer %p", &i);
}
```

6.2.2 Input Using the Lt I/O Library

The Lt I/O Library input functions, lt_getch, lt_gets and lt_atoi are similar to their standard C counterparts, getch, gets and atoi. Below is a program illustrating their use:

```
#include <ltio.h>

main()
{     char string[80] ;
      int num, ch;
      lt_printf("\nThese are the input possibilities");
      lt_printf(" with the lt library");
      lt_printf("\nYou can input a string");
      lt_printf("\nInput a string:");
      lt_gets(string);
      lt_printf("\nThe string input was %s ", string);
      lt_printf("\nYou input a string and convert into a number");
      lt_printf("\nEnter a number");
      lt_gets(string);
      num = lt_atoi(string);
      lt_printf("\nThe number is %d", num);
      lt_printf("\nFinally we echo characters till you type a |");
      do
            {     ch = lt_getch();
                  lt_putch(ch);
            }     while (ch ! = '|');
}
```

6.2.3 Lt I/O Heap Management

The Lt I/O Library heap management (dynamic memory allocation) procedures, lt_malloc and lt_free, are similar to their standard C counterparts, malloc and free. Lt heap management procedures differ from the standard C heap management procedures in that the Lt procedures maintain no linked list of allocated memory blocks. Lt heap management procedures use a stack model instead. As memory is allocated using lt_malloc, the bottom of free memory moves up. When a block of memory is deallocated using lt_free, all memory above that block is also deallocated. Because of this, memory allocation must be carefully ordered if you intend to allocate some memory, free it, and then reallocate it later in the program.

The standard C heap management procedures (malloc, free etc.) can also be used when writing programs that run exclusively in on-chip memory only, although they require more space. Both sets of heap management procedures, standard and Lt, cannot be combined in the same program, however. If your program uses any of the standard C heap management functions, or other standard C functions which use malloc internally, you cannot use lt_malloc or lt_free.

Below is a program illustrating the use of these procedures:

```
#include <ltio.h>

main()
{     int *array;
      char *string;

      array = lt_malloc(100 * sizeof(int));
      string = lt_malloc(80);
```

6.2.4 Lt I/O Library Reference

lt_atoi
convert a string to a number

Synopsis
#include <ltio.h>
int atoi(const char *cptr)

Description
The **lt_atoi** function converts an ASCII numeric string pointed to by **cptr** into the equivalent integer. The strings are assumed to be base 10, white space is allowed before the numeric string, and the first unrecognized character stops the conversion.

atoi recognizes integral numbers consisting of an optional sign and digit sequence.

Return Value
lt_atoi returns the converted value.

Related Function
atoi

lt_free
free heap memory

Synopsis
#include <ltio.h>
void lt_free(void *ptr)

Description
The **lt_free** function returns the previously allocated region of heap memory pointed to by **ptr** and all regions of memory above **ptr** to the heap free storage pool.

Return Value
None

Related Functions
lt_malloc, malloc, free

lt_getch
read a character

Synopsis
#include <ltio.h>
int lt_getch(void)

Description
The **lt_getch** macro reads, without echoing, a single character from the keyboard. When reading a cursor or function key, lt_getch must be called twice (the second call gets the actual scan code). To get a functional version, precede the function call with:

```
    #undef lt_getc
```

Return Value
lt_getch returns the character, or portion thereof, read. There is no error return.

Related Functions
getch, getchar, lt_putch, putchar

lt_gets
read a line

Synopsis
#include <ltio.h>
char *lt_gets(char *ptr)

Description
The **gets** function reads characters from the keyboard into memory starting at **ptr**. Characters are read until a newline character is read (it is discarded). A '\0' character is added after the last character read.

You must ensure that the character array (pointed to by **ptr**), is large enough to hold the longest possible input line, otherwise corruption of other data or code is likely.

Return Value
lt_gets returns **ptr**. It returns NULL if an error was detected.

Related Functions
gets, lt_puts, puts,

lt_malloc
allocate heap memory

Synopsis
#include <ltio.h>
void *lt_malloc(size_t size)

Description
The **lt_malloc** function allocates a region of heap memory large enough to hold an object of **size** bytes.

Return Value
If it is impossible to satisfy the request, or if **size** is zero, a NULL pointer is returned. Otherwise, a pointer is returned to the start of the allocated region.

Related Functions
malloc, lt_free, free

lt_putch
write a character

Synopsis
#include <ltio.h>
int lt_putc(int ch)

Description
The **lt_putch** macro writes the character specified by **ch** to the screen. Since **lt_putch** is defined to be a macro, the arguments may be evaluated more than once, and must not have side-effects. To get a functional version, precede the function call with:

```
#undef putc
```

Return Value
lt_putch returns the character written.

Related Functions
putchar, lt_getch, getchar

lt_puts
write a line

Synopsis
#include <stdio.h>
int lt_puts(const char *ptr)

Description
The **lt_puts** function writes the string pointed to by **ptr** to the screen. A newline is written to replace the terminating '\0' character.

Return Value
lt_puts returns EOF if an error occurs; otherwise it returns a non-negative value.

Related Functions
puts, lt_gets, gets

lt_printf
simple formatted write

Synopsis
#include <stdio.h>
void lt_printf(const char *format, ...)

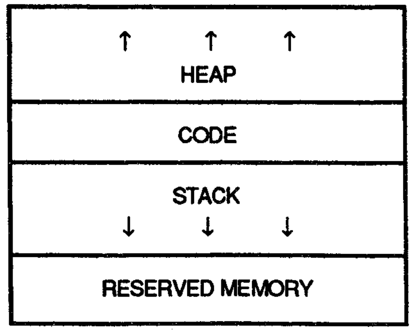
Description
The **lt_printf** function is a stripped down version of the normal printf function that works with the ltio server. **Lt_printf** doesn't support floating point, option flags, field widths or precision values. It also doesn't support any return value

Return Value
None

Related Functions
printf

6.3 Optimizing Memory Use

The diagram below contains a map of transputer on-chip memory usage. The first 70 (hex) words are reserved for use by the transputer, and overwriting them leads to undefined transputer behavior. The stack is placed above the reserved memory and grows downward from the start of the code area. The code area contains executable code and any static variables. The heap consists of all remaining on-chip memory.



Transputer On-Chip Memory Usage

`ltlsc4` sets the boundary between stack and code at 80000400 (hex) dividing memory evenly between code and stack. The linker allows you to place this boundary at any address you wish by modifying the load address (where the code is placed) and the stack address (where the stack starts). Three versions of `ltlsc.bat` are shown below. The first is the version that comes with the C Toolset and sets the boundary between code and stack at 80000400 (hex). The second places the boundary at 80000200 (hex) reserving more memory for code, and the third places the boundary at 80000600 (hex) giving more stack space.

Batch File 1 (standard `ltlsc4` batch file):

```
pp %1.c -dT414
tcx %1 -c
tasm %1 -ct
echo FLAG c >%1.lnk
echo LIST %1.map >>%1.lnk
echo INPUT %1.trl >>%1.lnk
echo ENTRY _lt_main >>%1.lnk
echo LIBRARY t4lib.tll >>%1.lnk
echo LOAD    0x80000400 >>%1.lnk
echo STACK   0x80000400 >>%1.lnk
tlnk %1.lnk
```

Batch File 2 (batch file dividing memory at 80000200):

```
pp %1.c -dT414
tcx %1 -c
tasm %1 -ct
echo FLAG c >%1.lnk
echo LIST %1.map >>%1.lnk
echo INPUT %1.trl >>%1.lnk
echo ENTRY _lt_main >>%1.lnk
echo LIBRARY t4lib.tll >>%1.lnk
echo LOAD    0x80000200 >>%1.lnk
echo STACK   0x80000200 >>%1.lnk
tlnk %1.lnk
```

Batch File 3 (batch file dividing memory at 80000600):

```
pp %1.c -dT414
tcx %1 -c
tasm %1 -ct
echo FLAG c >%1.lnk
echo LIST %1.map >>%1.lnk
echo INPUT %1.trl >>%1.lnk
echo ENTRY _lt_main >>%1.lnk
echo LIBRARY t4lib.tll >>%1.lnk
echo LOAD    0x80000600 >>%1.lnk
echo STACK   0x80000600 >>%1.lnk
tlnk %1.lnk
```

6.4 Multi-processing in On-Chip Memory

It is possible to write concurrent (i.e. multiple process) transputer programs that execute exclusively in on-chip memory. The Mandelbrot fractal demonstration program on the Test and Demo disk has seven concurrent processes, and not only runs in on-chip memory, but will run in parallel on as many processors as can be found in a multi-transputer network.

As standard C does not support transputer-style multi-processing, a special library of functions is provided with the C Toolset for the transputer. Basically, there are functions to start and to stop processes, and to pass messages between them. A few of these functions will be used below, but with little explanation. A complete list of them, along with descriptions of their usage, can be found in the C 89.1 book which came with the Kit.

The functions most frequently used in process instantiation are ProcAlloc and ProcRun. ProcAlloc dynamically allocates space in memory for the process state, and initializes that memory with the proper content. ProcRun is used to start execution of the newly created process. Unfortunately for the programmer using on-chip memory only, the function ProcAlloc references the function malloc which requires extensive memory. But fortunately there is a lower level process named ProcInit which can be employed in conjunction with the smaller lt_malloc to accomplish the same job as ProcAlloc.

Concurrent processes can communicate with each other (and are synchronized) by passing messages over "channels." Multiple processes executing within the same processor use memory locations for channels. Processes executing in true parallel fashion on different processors use the transputers' serial links as channels.

Commonly used channel communication functions include ChanAlloc, ChanInChar and ChanOutChar. ChanAlloc dynamically allocates a channel location and initializes it. ChanInChar and ChanOutChar are used at opposite ends of a channel to pass information between two processes (and to synchronize them). Unfortunately, again, ChanAlloc references malloc. But fortunately, again, a lower level process is available, named ChanReset, which can be used along with lt_malloc to do accomplish the same job as ChanAlloc.

Below are three versions of a simple program which utilizes a pair of concurrent processes (the main process, plus another called echo which is instantiated on the fly. The first version uses the standard C libraries, ProcAlloc, ProcRun, ChanAlloc, ChanInChar and ChanOutChar. The second version uses ProcInit and lt_malloc in place of ProcAlloc, and ChanReset and lt_malloc in place of ChanAlloc. The third version uses ProcInit and ChanReset in place of ProcAlloc and ChanAlloc, but uses static variables instead of dynamically allocated memory.

Program 1 (using standard C memory allocation):

```
#include <ltio.h>
#include <conc.h>
#define TRUE 1

int echo(Process *p, Channel *in, Channel *out);

main()
{
      Process  *echo_proc;
      Channel  *to_echo,  *from_echo;
      int ch;
      to_echo  =  ChanAlloc();  /*  Allocate and initialize */
      from_echo  =  ChanAlloc();     /* memory for channels */

                              /* Allocate and set up memory for*/
                              /* the echo process */
      echo_proc  =  ProcAlloc(echo,  100,  2,  to_echo,  from_echo);
      ProcRun(echo_proc);      /*Start the echo process running*/
      while  (TRUE)
           {  ch  =  getch();/*Read a character from the keyboard*/
              ChanOutChar(from_echo);      /*Send it to echo*/
              ch =ChanInChar(from_echo);/*Read it back from echo*/
             putchar(ch);            /*print it on the screen*/
```

```
            }
    }

    int echo(Process *p, Channel *in, Channel *out)
    {     int ch;
          while  (TRUE)
              {
                    ch = ChanInChar(in);/*Read a char from the main process*/
                    ChanOutChar(out, ch);/*Send it back to the main process*/
              }
    }
```

Program 2 (using lt_malloc):

In this program, ChanAlloc is replaced by a call to lt_malloc and then a call to ChanReset; ProcAlloc is replaced by two calls to lt_malloc and then a call to ProcInit.

```
    #include <ltio.h>
    #include <conc.h>
    #define TRUE 1

    int echo(Process *p. Channel *in, Channel *out);

    main()
    {     Process  *echo_proc;
          char *echo_ws;
          Channel  *to_echo,  *from_echo,
          int ch;

               /*Allocate memory for Channels*/
          to_echo      = lt_malloc(sizeof(Channel));
               /*Allocate Process structure*/
          from_echo    = lt_malloc(sizeof(Process));
               /*Initialize channels*/
          ChanReset(to_echo);
          ChanReset(from_echo);

          echo_ws      = lt_malloc(100);/*Allocate memory for workspace*/
                                       /*Allocate Process structure*/
          echo_proc    = lt_malloc(sizeof(Process));
                      /*Initialize workspace and Process structure*/
          ProcInit(echo_proc, echo, echo_ws, 100, 2, to_echo, from_echo);
          ProcRun(echo_proc);     /*Start the echo process running*/
          while  (TRUE)
               {  ch = getch();/*Read a character from the keyboard*/
                  ChanOutChar(from_echo);     /*Send it to echo*/
                  ch = ChanInChar(from_echo);/*Read it back from echo*/
                  putchar(ch);            /*print it on the screen*/
               }
    }

    int echo(Process *p, Channel *in, Channel *out)
    {     int ch;
          while  (TRUE)
              {
                    ch = ChanInChar(in);/*Read a char from the main process*/
```

```
                    ChanOutChar(out, ch);/*Send it back to the main process*/
              }
    }
```

Program 3 (using static variable declaration):

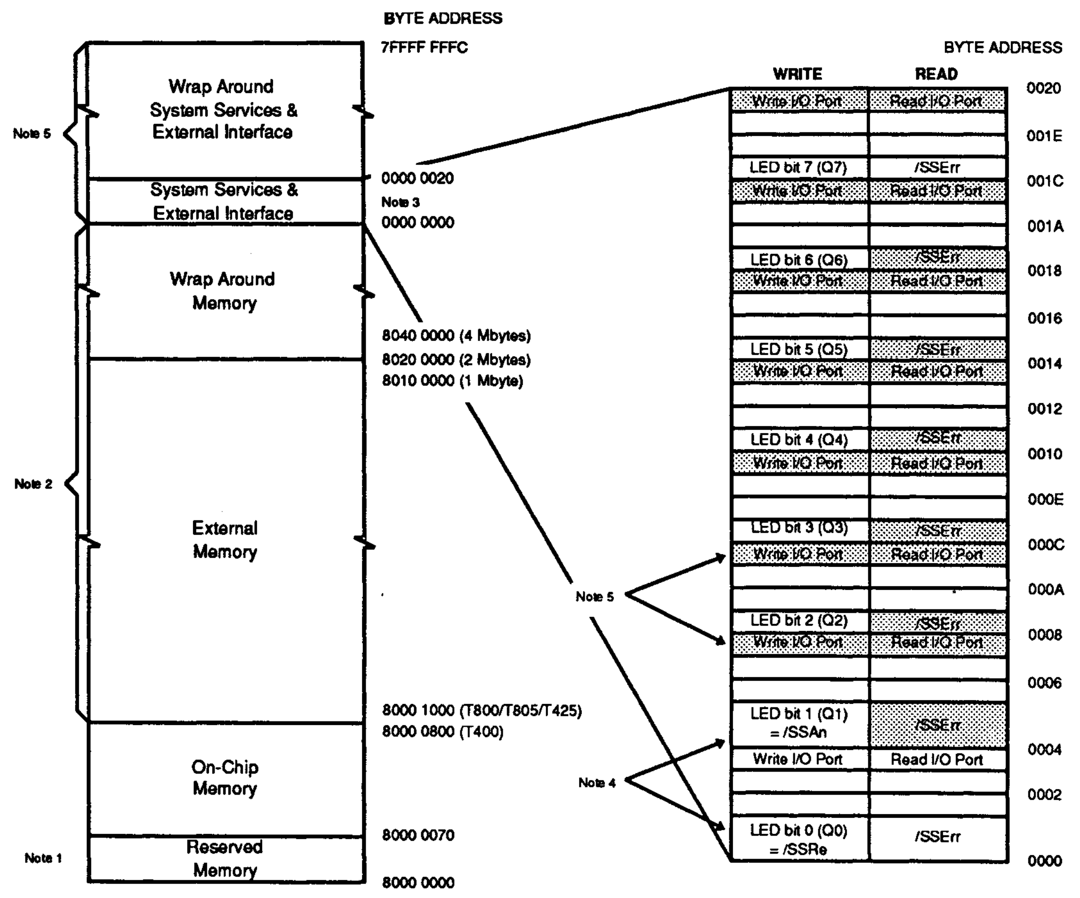
In this version, ChanAlloc is replaced by declaring a channel variable and a pointer and by assigning the pointer to the channel. The channel is reset, then passed to various procedures. ProcAlloc is replaced by declaring two static variables of the correct size and calling ProcInit.

```
    #include <ltio.h>
    #include <conc.h>
    #define TRUE 1

    int echo(Process *p. Channel *in, Channel *out);
    static char echo_ws[100];

    main()
    {
          Process *echo_proc, ep;
          Channel *to_echo, *from_echo, te, fe;
          int ch;

          to_echo     = &to;       /*Initialize channel pointers*/
          from_echo   = &fo;       /*to address of te and fe*/
          ChanReset(to_echo);      /*Initialize channels*/
          ChanReset(from_echo);

          echo_proc   = &ep;  /*Initialize Process pointer to echo_proc*/
                           /*Initialize workspace and proc structure*/
          ProcInit(echo_proc, echo, echo_ws, 100, 2, to_echo, from_echo);
          ProcRun(echo_proc);      /*Start the echo process running*/
          while  (TRUE)
               {  ch = getch();/*Read a character from the keyboard*/
                  ChanOutChar(from_echo);     /*Send it to echo*/
                  ch = ChanInChar(from_echo);/*Read it back from echo*/
                  putchar(ch);            /*print it on the screen*/
               }
    }

    int echo(Process *p, Channel *in, Channel *out)
    {     int ch;
          while  (TRUE)
              {
                    ch = ChanInChar(in);/*Read a char from the main process*/
                    ChanOutChar(out, ch);/*Send it back to the main process*/
              }
    }
```

A Appendix

A.1 Memory Map

The transputer (with the exception of the 16-bit models) has a flat 32-bit address space. Contrary to what might be expected, memory addresses are considered to be signed numbers, and as such the lowest memory location is addressed by the negative integer having the greatest possible magnitude, 8000 0000 (hex). Note in the memory map below that the transputer's on-chip memory is in the same memory space as external DRAM. When these on-chip locations are referenced by a program, external memory in the same region is disabled. See Note 2 for a way to get at this blocked out section of external memory.



Note 1 This area is used by the transputer and is not available for code or data. For further information look in the Transputer Technical Specifications in the **Transputer** book.

Note 2 There are 32 address lines on the transputer. Only 20 are used when one Mbyte of memory is installed and 22 when two or four Mbytes of memory are installed. The other address lines are not used. The result of this (called memory wrap around) is that the same memory cell can be written or read at two different addresses. For example, in a board with 1 Mbyte of memory, the following addresses all refer to the same memory cell: 80008000, 80108000, 80208000, etc. In a board with two or four Mbytes the following addresses refer to the same memory cell: 80008000, 80408000, 80808000, etc. On a two Mbyte board. half the addresses in the memory region don't access any memory at all. The addresses 80000000 - 80200000 do access memory, 80200000 - 80400000 do not, 80400000 - 80600000 do access memory, etc.

Note 3 The External Interface is described in Chapter 4

Note 4 The SS (SubSystem) system service signals (see section 3.6) are decoded in this range. The LED Port bits 0 and 1 (Q0 and Q1) are equivalent to the SS Reset and Analyse (/SSRes, /SSAn).

Note 5 The I/O space, like the memory space, is not totally decoded (that is, not all 32 address lines are used). The I/O port can be written or read at any address that can be described in the form 3 + 4n (n being an integer). Some addresses where the I/O port can be written or read are 3, 7, 11, 15, etc. The /SSErr can be read at any address described in the form 8n (i.e. 0, 8, 16, etc.). The LED bits appear every 32 bytes. They are written at address described by 4b + 32n where b is the bit number and n is any integer. /SSRe and /SSAn are controlled by writing bits 0 and 1 (Q) and Q1) of the LED port respectively. To maintain compatibility with future products, use the lowest possible address to read or write to the I/O signals (these are the ones that are not shaded in the map above).

A.2 Memory Mapped I/O Interface Addresses

A.2.1 PC Link

PC I/O Addr	Write	Read
Lnk+0	Invalid	Byte from Link
Lnk+1	Byte to Link	Invalid
Lnk+2	Input Status	Input Status
Lnk+3	Output Status	Output Status

The PC Link is implemented with an Inmos C012 link adaptor which offers a microprocessor interface, converting serial link data to byte-wide parallel data and vice versa. The C012 sends bytes from the PC host to the PC Link and conversely allows the PC host to receive bytes from the PC Link. It has four registers: an input data register, an input status register, an output data register and an output status register. Only one bit from each of the status registers is used by the Transputer Education Kit. Bit 0 of the input status register is automatically set to indicate that a byte has been brought in over the link and is ready to be read from the input data register. Reading the input data register resets Bit 0 of the input status register. Bit 0 of the output status register is automatically set whenever the output data register is ready for another byte. Writing to the output data register causes this bit to be reset until the written byte has passed out the link and the output data register is again empty.

To read a byte from the PC Link, you can poll the input status register until Bit 0 is set then read the byte from the input data register.

To write a byte to the PC Link, you can poll the output status register until Bit 0 is set then write the byte to be sent to the output data register.

Lnk in the table above refers to the **Lnk** address selected in the PC Link Address jumper block. For example if **Lnk** was selected to be 150 (hex) then the four addresses 150, 151, 152 and 153 are used to access the C012 Link Adaptor chip. Note that these addresses are in the I/O space of the PC and not the memory space. See section **3.3.1 PC/Link Interface** for more information on selecting the PC/Link Interface addresses.

Example procedures demonstrating how to read and write the link are included on the Examples disk. Three different versions are supplied; one in Microsoft C, one in Turbo C, and one in 8086 assembly language.

For more information on the Inmos C012 Link Adaptor chip see the **Transputer Databook**.

A.2.2 PC Link SS (SubSystem) System Services

PC I/O Addr.	Write	Read
SS+0	PC Link Reset	PC Link Error
SS+1	PC Link Analyse	Invalid

PC Link Reset is asserted by writing a 1 to **SS** + 0. It is deasserted by writing a 0 to **SS** + 0.

PC Link Analyse is asserted by writing a 1 to **SS** + 1. It is deasserted by writing a 0 **SS** + 1.

PC Link Error is read at **SS** + 0. The state of the Error line is indicated in Bit 0. If Bit 0 is 1 the Error line is asserted. If Bit 0 is 0 the Error line is deasserted.

The actual signals associated with these locations are active low. That means writing a 1 to PC Link Reset puts the Reset line low, etc.

For more information on the timing needed to reset or analyse a transputer refer to the T805 data sheet section of the **Transputer** book.

SS in the table above refers to the **SS** address selected in the PC Link Address jumper block. For example, if **SS** was selected to be 160 (hex), then the addresses 160, 161 are used to read and write the system services signals. Note that these addresses are in the I/O space of the PC and not the memory space. See section **3.3.1 PC/Link Interface** for more information on selecting the PC/Link Interface addresses.

A small note about nomenclature. **SS** for the PC/Link Interface does not actually stand for SubSystem as indicated above but rather for System Services. The PC/Link Interface system services work in the same manner as SubSystem system services on a transputer in that it a allows a subsystem to be reset by the transputer. The addressing is scheme is the same as can be seen by looking at the address diagram for this section and the address diagram for the transputer SS section below.

Example procedures demonstrating how to read and write PC Link system services and reset a transputer are furnished in Microsoft C, Turbo C and 8086 assembly. These procedures are included on the Examples disk.

A.2.3 Transputer SS (SubSystem) System Services

Xptr Addr.	Write	Read
0	SS Reset	SS Error
4	SSAnalyse	Invalid

SS Reset is asserted by writing a 1 to address 0. It is deasserted by writing a 0 to address 0.

SS Analyse is asserted by writing a 1 to address 4. It is deasserted by writing a 0 to address 4.

SS Error is read at address 0. The state of the Error line is indicated in Bit 0. If Bit 0 is 1 the Error line is asserted. If Bit 0 is 0 the Error line is deasserted.

The actual signals associated with these locations are active low. That means writing a 1 to SS Reset puts the Reset line low, etc. For more information on the timing needed to reset or analyse a transputer refer to the T805 data sheet section of the **Transputer** book.

Example procedures demonstrating how to read and write the SS system services of a transputer and reset a sub-system (SS) are furnished in C and Occam. These procedures are included on the Examples disk.

Note 1 The SS system services signals share a common word address with the Programmed I/O port. The difference is that the SS system services signals are mapped into the low order byte and the Programmed I/O signals are mapped into the high order byte. For this reason these signals should be modified with a byte write rather than a word write. If this is not done writes to SS system services will very likely change some of the signals on the Programmed I/O port. The inverse is also true. Word writes to the Programmed I/O port will affect SSReset.

A.2.4 External Interface LED Port and Power Transistors

Qn	*Address in decimal*	*Qn*	*Address in decimal*
Q0	0	Q4	16
Q1	4	Q5	20
Q2	8	Q6	24
Q3	12	Q7	28

The LED signals Q0-Q7 are taken high or low by writing the desired signal level to the low order byte of the word addresses in the diagram above. To write to Qn, write to the address 4n. Writing a 1 in bit 0 sets the line high (turning the LED off); writing a 0 sets the line low (turning the LED on). These lines are low on power up and are not effected by a transputer reset. That is they retain their state between program runs. Sample C and Occam procedures are provided on the Examples disk illustrating how to write these signals.

The power transistors are controlled by the system services SS analyse signal which in turn is controlled by Q1 of the LED port signals (see section 4.1.1 **LED Port**). Writing a 0 to Q1 turns the power on and writing a 1 turns the power off. Sample C and Occam procedures are provided on the Examples disk illustrating how to write these signals.

Note 1 Q0 and Q1 perform double duty as they are the same signals memory-mapped signals that are inverted to generate SS system services reset and analyse signals. Keep this in mind if you use SS (see section **3.6** and **A.2.3**) or the programmed I/O port in conjunction with the LED signals.

Note 2 This port shares common word addresses with the Programmed I/O port. The difference is that the LED port is written with the low order byte and the Programmed I/O port is written with the high order byte. For this reason these signals should be modified with a byte write rather than a word write. If this is not done writes to this port will very likely change some of the signals on the Programmed I/O port. The inverse is also true. Word writes to the Programmed I/O port will affect one of the signals of the LED port.

Note 3 The I/O port which includes the LED port is not completely decoded. This means that each bit of the LED port may be written at many addresses. That is Q0 can be written at 0, 32, 64 etc. Q1 can be written at 4, 36, 68. Any address of the form 32i + 4n will work when n is the signal number (for Q3 n = 3) and i is any number that results in an address with bit 31 set to 0.

A.2.5 External Interface Programmed I/O Port

Xptr Addr.	Write	Read
0	SS Reset	SS Error
4	SSAnalyse	Invalid

The signals of the Programmed I/O port that are directly under program control are I/O0-7, Q2 and /SSErr.

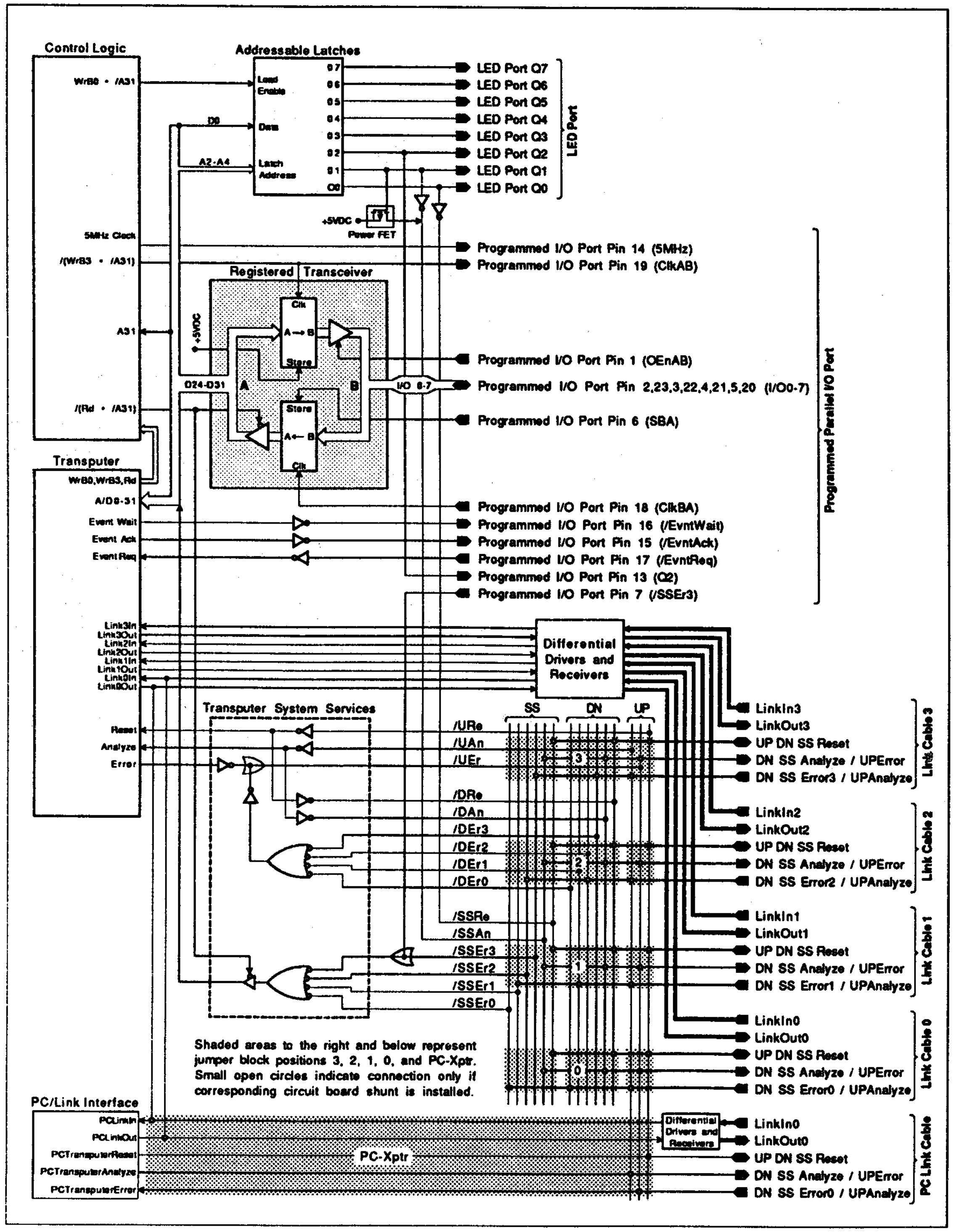
The programmed I/O port signals I/O0-7 are accessed by reading or writing the high order byte of the word at physical memory location 0. I/O0 corresponds to bit 0 of this byte and I/O7 corresponds to bit 7. The signal level presented on the I/O lines is a high if the corresponding bit is 1 and low if the corresponding bit is 0. These signals are low on power-up and are not affected by a transputer reset so they retain their state between program runs. Sample C and Occam procedures are provided on the Examples disk illustrating how to write these signals.

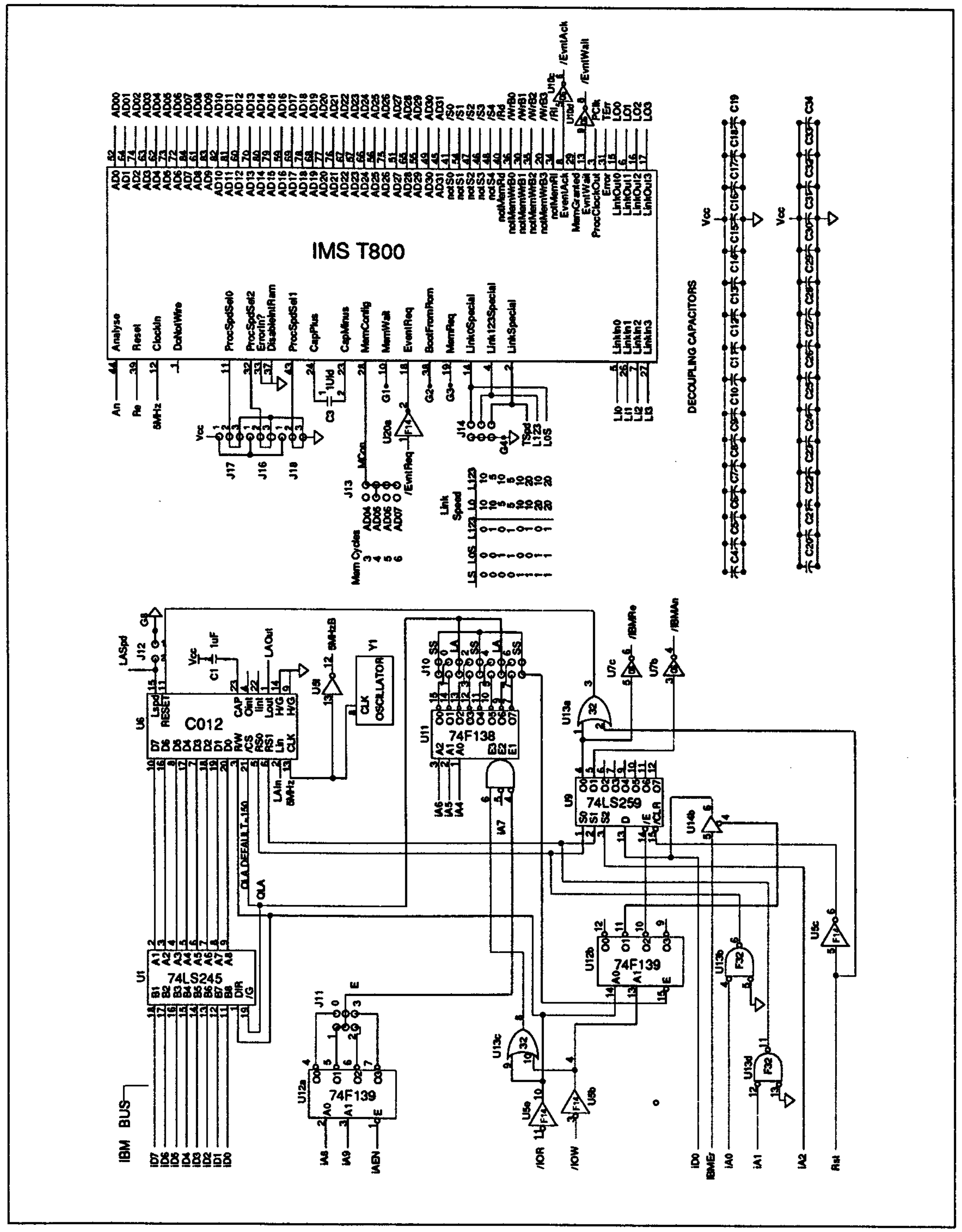
Q2 is one of the signals from the LED port. Its operation is described in the preceding section. /SSErr is the Error signal for the transputer's SS error. It's operation is described in Appendix **A.2.3**. Note that this signal cannot be used on the Programmed I/O port if the transputer's SS system services are connected.

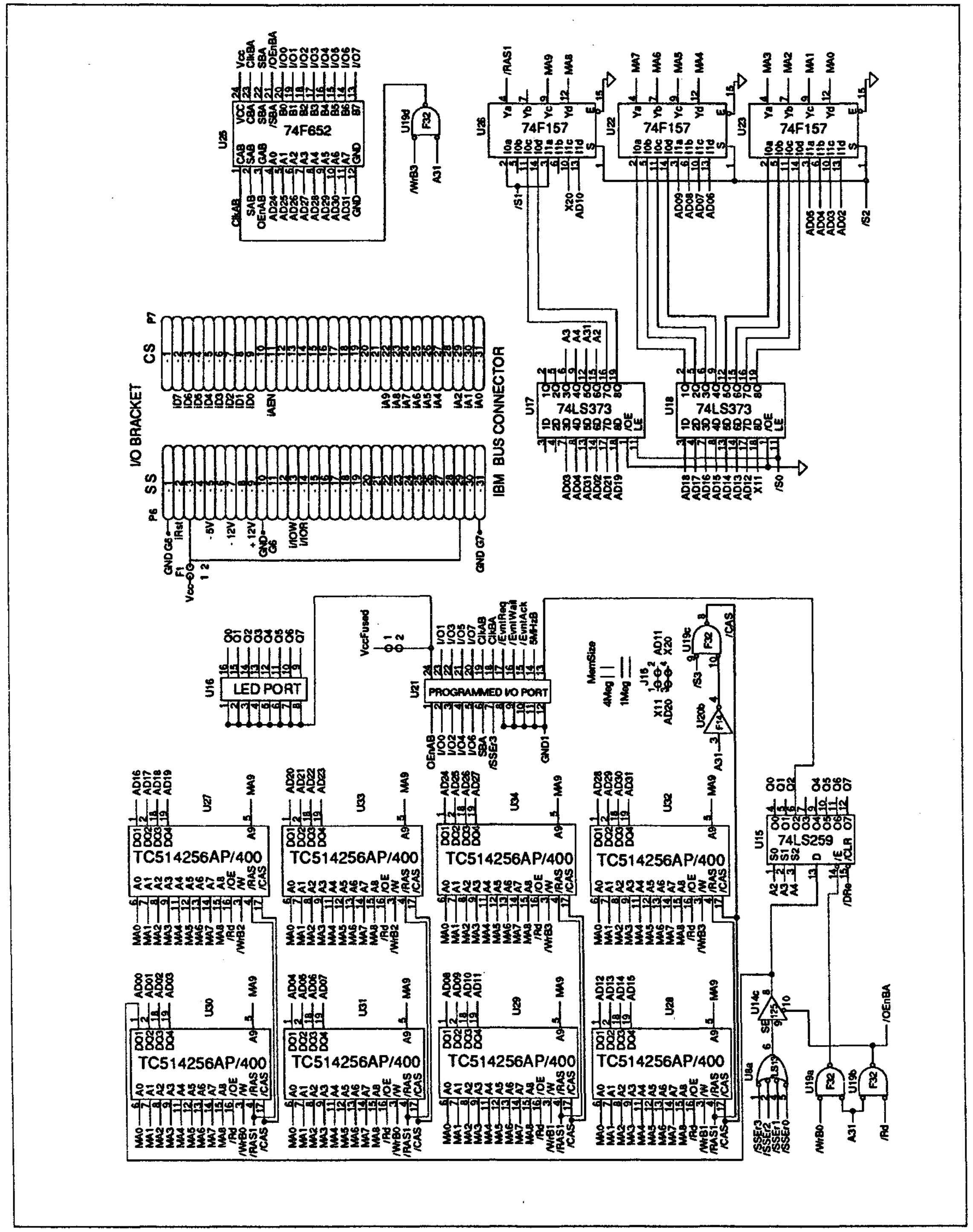
Note 1 This port shares common word addresses with the LED port. The difference is that the Programmed I/O port is mapped into the high order byte and the LED port is mapped into the low order byte. For this reason these signals should be modified with a byte write rather than a word write. If this is not done writes to this port will affect one of the signals of the LED port. The inverse is also true. Word writes to the LED port will very likely change some of the signals on the Programmed I/O port.

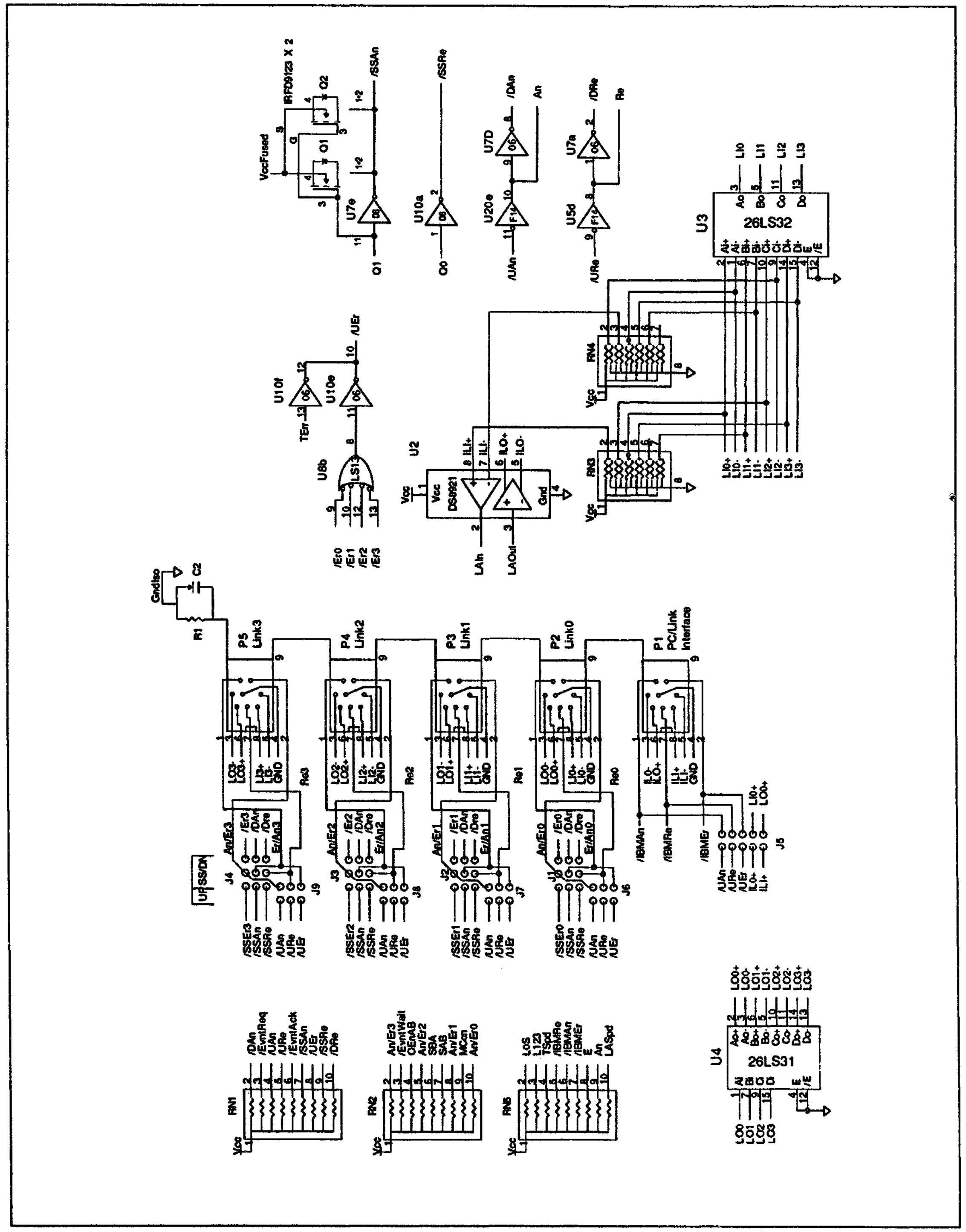
Note 2 The I/O space which includes the Programmed I/O port is not completely decoded. This means that the Programmed I/O port may be written at any physical address that can be described by the equation 3 + 4n where n is any integer that results in an address with bit 31 set to 0.

A.3 Board Schematics and Block Diagram



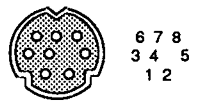




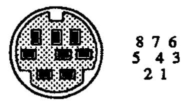


A.4 External Cable Schematics

A.4.1 Cables between Transputer Education Kit Boards



8 Pin MiniDIN Cable Connector Pinout



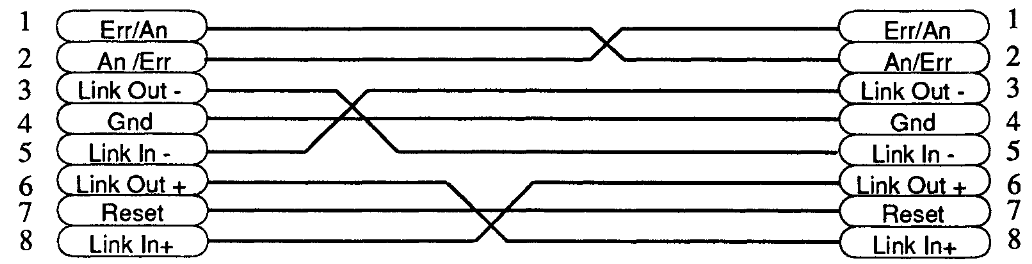
I/O Connector Pinout

	SS	DN	UP	None
1	/SSAn	/DNAn	/UPErr	NC
2	/SSErr	/DNErr	/UPAn	NC
3	LinkOut-	LinkOut-	LinkOut-	LinkOut-
4	Gnd	Gnd	Gnd	Gnd
5	LinkIn-	LinkIn-	LinkIn-	LinkIn-
6	LinkOut+	LinkOut+	LinkOut+	LinkOut+
7	/SSRes	/SSRes	/SSRes	/SSRes
8	LinkIn+	LinkIn+	LinkIn+	LinkIn+

Link cable I/O pinouts for corresponding system services jumper connections (SS, DN, UP, None).

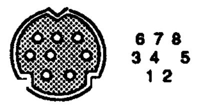
8 Pin MiniDIN Cable Connector

8 Pin MiniDIN Cable Connector

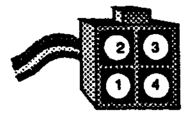


Note that pins 1 and 2 are used differently, depending on the UP, DN and SS jumper settings in jumper blocks 0, 1, 2 and 3 of the Kit circuit boards at each end of a given cable. As such, pin 1 of the connector is always an output signal, and pin 2 is always an input.

A.4.2 Education Kit to CSA PART Series (Internal Connectors).



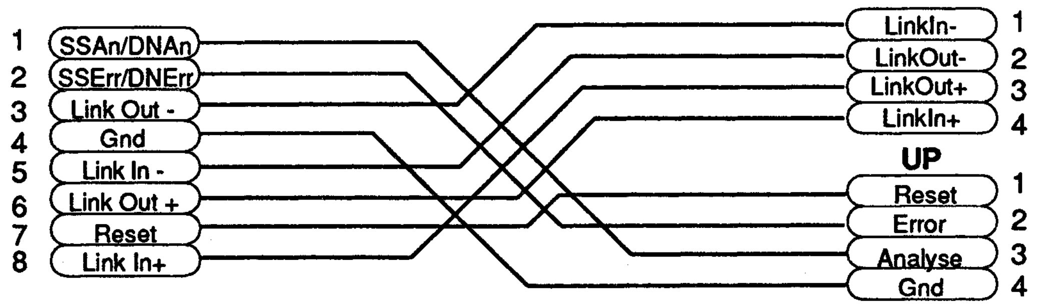
8 Pin MiniDIN Cable Connector Pinout



2x2 Cable Connector Pinout

8 Pin MiniDIN Cable Connector

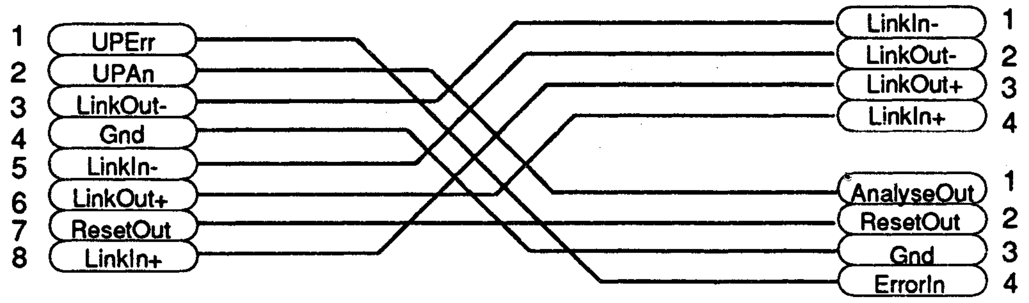
2x2 Cable Connectors



Transputer Education Kit board reset by CSA PART board
Transputer Education Kit board receives UP system services

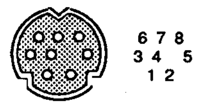
8 Pin MiniDIN Cable Connector

2x2 Cable Connectors



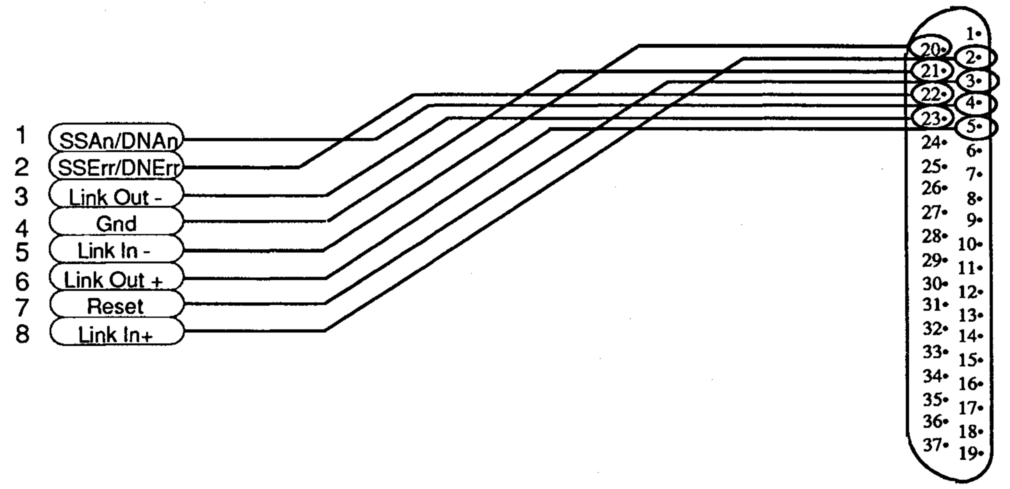
CSA PART board reset by Transputer Education Kit board
CSA PART board receives UP system services

A.4.3 Education Kit to CSA PART Series (External Connector)

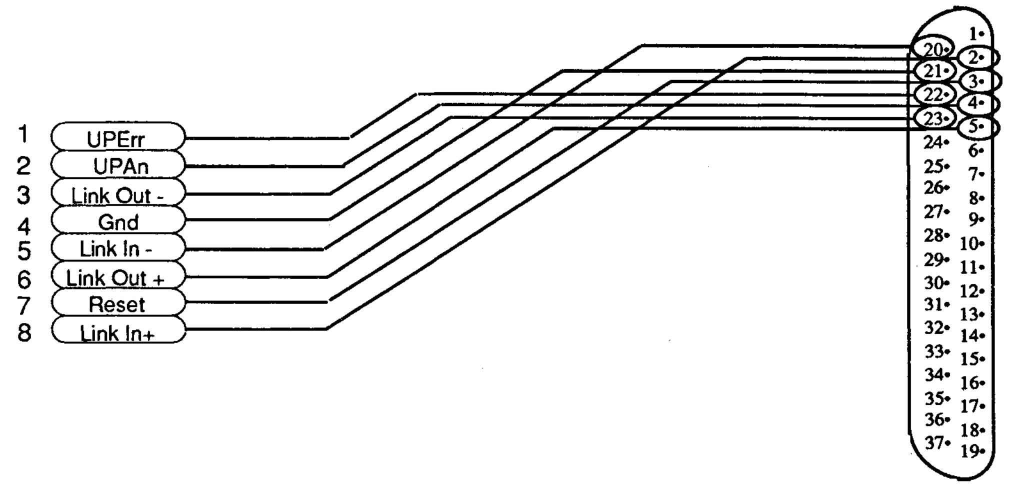


8 Pin MiniDIN Cable Connector Pinout

D-Subminiature 37 pin shell AMP 745498-1 or equivalent

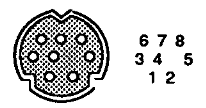


CSA PART board reset by Transputer Education Kit board
CSA PART board receives UP system services

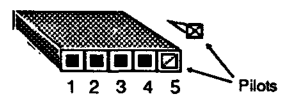


Transputer Education Kit board reset by CSA PART board
Transputer Education Kit board receives UP system services

A.4.4 Education Kit to Inmos Boards



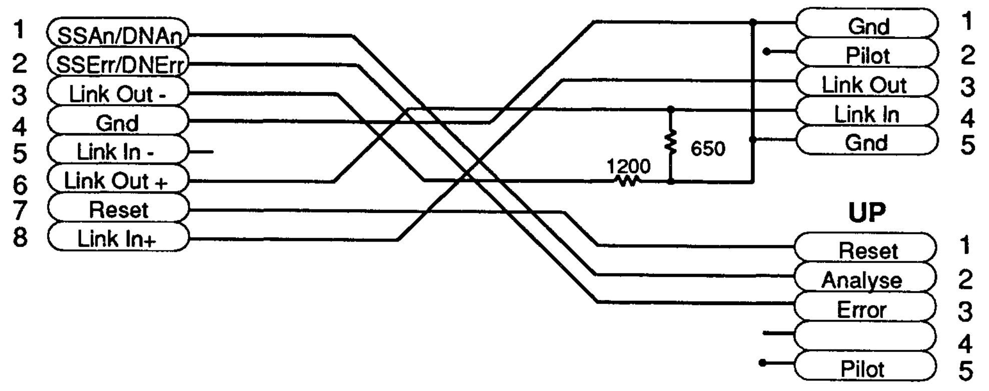
8 Pin MiniDIN Cable Connector Layout



1x5 Connector Layout

8 Pin MiniDIN Cable Connector

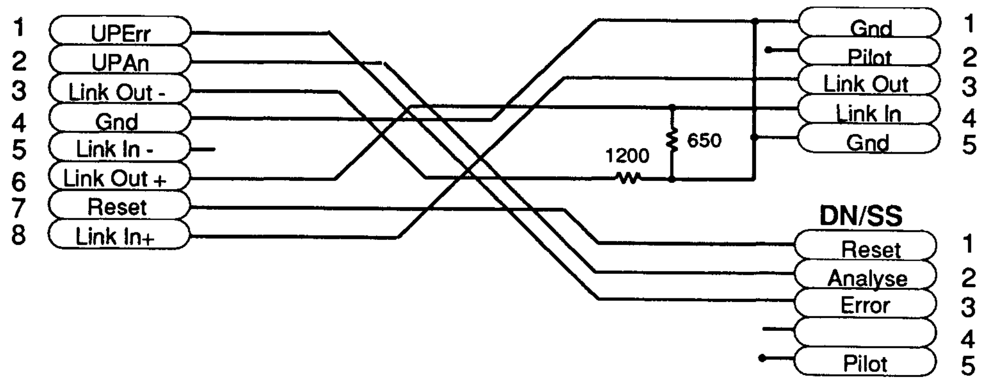
1x5 Cable Connectors



Transputer Education Kit board reset by Inmos board
Transputer Education Kit board receives UP system services

8 Pin MiniDIN Cable Connector

1x5 Cable Connectors



Inmos board reset by Transputer Education Kit board
Inmos board receives UP system services

Index