TRANSPUTER BASED REAL-TIME PROCESS CONTROL

by

Sasidhar V. Challa

Submitted in Partial Fulfillment of the Requirements

for the degree of

Master of Science in Engineering

in the Electrical Engineering

Program

S. R. Parameswaran 9/1/92

Adviser Date

Sally M. Hotchkiss September 2, 1992

Dean of the Graduate School Date

YOUNGSTOWN STATE UNIVERSITY

August, 1992

ABSTRACT

TRANSPUTER BASED REAL-TIME PROCESS CONTROL

Sasidhar V. Challa

Master of Science in Engineering

Youngstown State University, 1992

A transputer based real-time process control system is designed and its performance is studied. The emphasis is placed on the hardware and software aspects of the transputer, i.e., installation, networking and interfacing to the test control system. The process control system consists of a stirred tank heater containing water. The temperature of the water is measured using a thermocouple and controlled using a proportional control algorithm. The control algorithm is implemented on a transputer network using OCCAM, which is the parallel processing language for the transputer. Software development, implementation and user interface are achieved through an IBM Personal Computer (PC/AT clone). Data acquisition and control are achieved through a Data Acquisition/Control Unit that is interfaced to the transputer network and hence to the user via the IBM PC.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Dr. Salvatore R. Pansino and Dr. Dilip K. Singh for their advice and assistance during the course of this work.

I am especially grateful to the Department of Chemical Engineering for providing the neccessary hardware which made this thesis possible.

I thank Prof. Robert H. Foulkes and Prof. Samuel J. Skarote for their advice and valuable comments during the documentation of my thesis.

I thank Ms. Anna Mae Serrecchio for her cooperation in formatting the thesis and last but not least I thank my fellow students Hung Ta and Tariq Alvi for their helpful hints and suggestions.

TABLE OF CONTENTS

LIST OF SYMBOLS/ABBREVIATIONS

SYMBOL	DEFINITION	UNITS OR REFERENCE
ANSI	American National Standards Institute	
CMOS	Complimentary Metal Oxide Semiconductor	
CPU	Central Processing Unit	
DACU	Data Acquisition and Control Unit	
Δ	Delta	
DMA	Direct Memory Access	
DOS	Disk Operating System	
DRAM	Dynamic Random Access Memory	
EEROM	Electrically Erasable Read Only Memory	
EMI	External Memory Interface	
EPROM	Erasable Programmable Read Only Memory	
FLOP	Floating Point Operation	
FOPDT	First Order Plus Dead Time	
FPU	Floating Point Unit	
G	Giga	1×10^9
GPIB	General Purpose Interface Bus	
IC	Integrated circuit	
IEEE	Institute of Electrical and Electronic Engineers	

SYMBOL	DEFINITION	UNITS OR REFERENCE
ITEM	INMOS Transputer Evaluation Module	
K	Kilo	1×10^3
K_c	Controller Gain	
LED	Light Emitting Diode	
M	Mega	1×10^6
MIPS	Million Instructions Per Second	
MMS	Module Motherboard Software	
PAL	Programmable Array Logic	
RAM	Random Access Memory	
ROM	Read Only Memory	
SRAM	Static Random Access Memory	
τ	Process time constant	Seconds
TDS	Transputer Development System	
TRAM	Transputer Module	
TTL	Transistor to Transistor Logic	

LIST OF FIGURES

FIGURE PAGE

LIST OF TABLES

CHAPTER I

INTRODUCTION

While spectacular progress in VLSI technology has been realized over the last 15 years to increase component density, less dramatic improvements in clock speed have been forthcoming. These realities suggest that control engineers might profitably investigate parallel processing solutions to meet increasingly demanding requirements. This interest has been further stimulated by the availability of the Inmos transputer which provides a flexible element for the support of parallel processing for real-time applications.

I.1 Real-time Systems and process control

One of the fastest expanding areas of computer exploitation is that involving applications, whose prime function is *not* that of information processing, but which nevertheless require information processing in order to carry out their prime function. A microprocessor-controlled washing machine is a good example of such a system. Here the prime function is to wash clothes; however, depending on the type of clothes to be washed, different 'wash programs' must be executed. These types of computer applications are generically called **real-time** or **embedded**. The Oxford

Dictionary of Computing gives the following definition of real-time system:

> **Any system in which the time at which the output is produced is significant. This is usually the case because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.**

Automatic process control is concerned with maintaining process variables: temperatures (as in the present study), pressures, flows, compositions at some desired operating value. Processes are dynamic in nature since changes are always occurring. Hence if actions are not taken on time, the important process variables-those related to safety, product quality, and production rates-will not achieve design values. Thus real-time systems find extensive application in process control.

I.2 Transputers

The INMOS transputer family is a range of system components each of which combines processing, memory and interconnect in a single VLSI chip. A concurrent system can be constructed from a collection of transputers which operate concurrently and communicate through serial communication links. Such systems can be designed and programmed in OCCAM, a language based on communicating

processes. In addition each transputer product contains special circuitry and interfaces adapting it to a particular use. For example, a peripheral control transputer, such as a General Purpose Interface Bus (GPIB) controller, has interfaces tailored to the requirements of the IEEE-488 test and instrumentation system bus signals.

The software required for the present study is written in OCCAM, which is the true parallel processing language for the transputer. Nevertheless, transputers may also be programmed using parallel FORTRAN and parallel C.

I.3 Objective and outline of the thesis

For the present study, a transputer based real-time experimental process control set up is considered. The emphasis is placed on the hardware aspects of the transputer, i.e., the links and interface to the input/output (I/O). Communication with the control system is achieved via the DACU by using command codes [3]. The GPIB OCCAM [8] software is used to perform the command and data I/O operations on the DACU.

The objective of the present study is to design and build a transputer based hardware/software system and implement it in controlling a simple process. The system consists of a stirred tank heater whose temperature is measured and controlled using a proportional control algorithm. The control is achieved using a hardware system

consisting of a transputer network which is programmed in the OCCAM language. The thesis problem is reduced into the following steps:

(1) Install the transputer hardware involving an IBM PC/AT clone and the transputer motherboards.

(2) Install the software including the OCCAM toolset [10], DOS device driver and modules to run the transputer motherboards, in the PC.

(3) Interface the process control test system to the transputer network (via the IBM PC/AT and the DACU).

(4) Estimate the process control parameters for the test system and formulate the control equations for a proportional control algorithm.

(5) Implement the algorithm in the system using OCCAM software.

CHAPTER II

DESCRIPTION OF THE TRANSPUTER HARDWARE AND SOFTWARE

II.1 Overview of the transputer family

The transputers encountered during the course of the present study are the **IMS T414, M212, C004, C012, T222** and **T800.** Their features are described briefly.

The **IMS T414** is a 32 bit CMOS microcomputer with 2 Kbytes of on-chip RAM for high speed processing, a configurable memory interface and four INMOS communication links. The instruction set achieves efficient implementation of high-level languages and provides direct support for the OCCAM model of concurrency when used either as a single transputer or a network. Although the T414 provides high-performance arithmetic and microcode support, it lacks a floating point unit (FPU) on the chip and its efficiency in terms of speed is thus restricted [11].

The **IMS M212** is a 16 bit peripheral processor configured for connection to soft sectored winchester and floppy disk drives. Two byte-wide programmable bidirectional ports are provided to control and monitor disk functions such as head position, drive selection and disk status. The M212 is programmed as a normal transputer, permitting peripheral control facilities to be built into

the device and thus reducing the load on the traditional central processor of a computer [11].

The **IMS C012** link adaptor is a communications device enabling the INMOS serial communication link to be connected to parallel data ports and microprocessor buses. The **IMS C004** is a programmable link switch and it provides a full cross bar switch between 32 link inputs and 32 link outputs. The C012 and C004 features are described in more detail on pages 52 through 55.

The **IMS T222** and **T800** transputers (modules) were used extensively in designing the hardware network for the present study and hence their features are described below.

The IMS T800 transputer

For convenience of description, the IMS T800 operation is split into the basic blocks shown in Fig. II.1. The various blocks in Fig. II.1 are described below.

32 bit CPU

The 32 bit CPU contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the external memory interface (EMI).

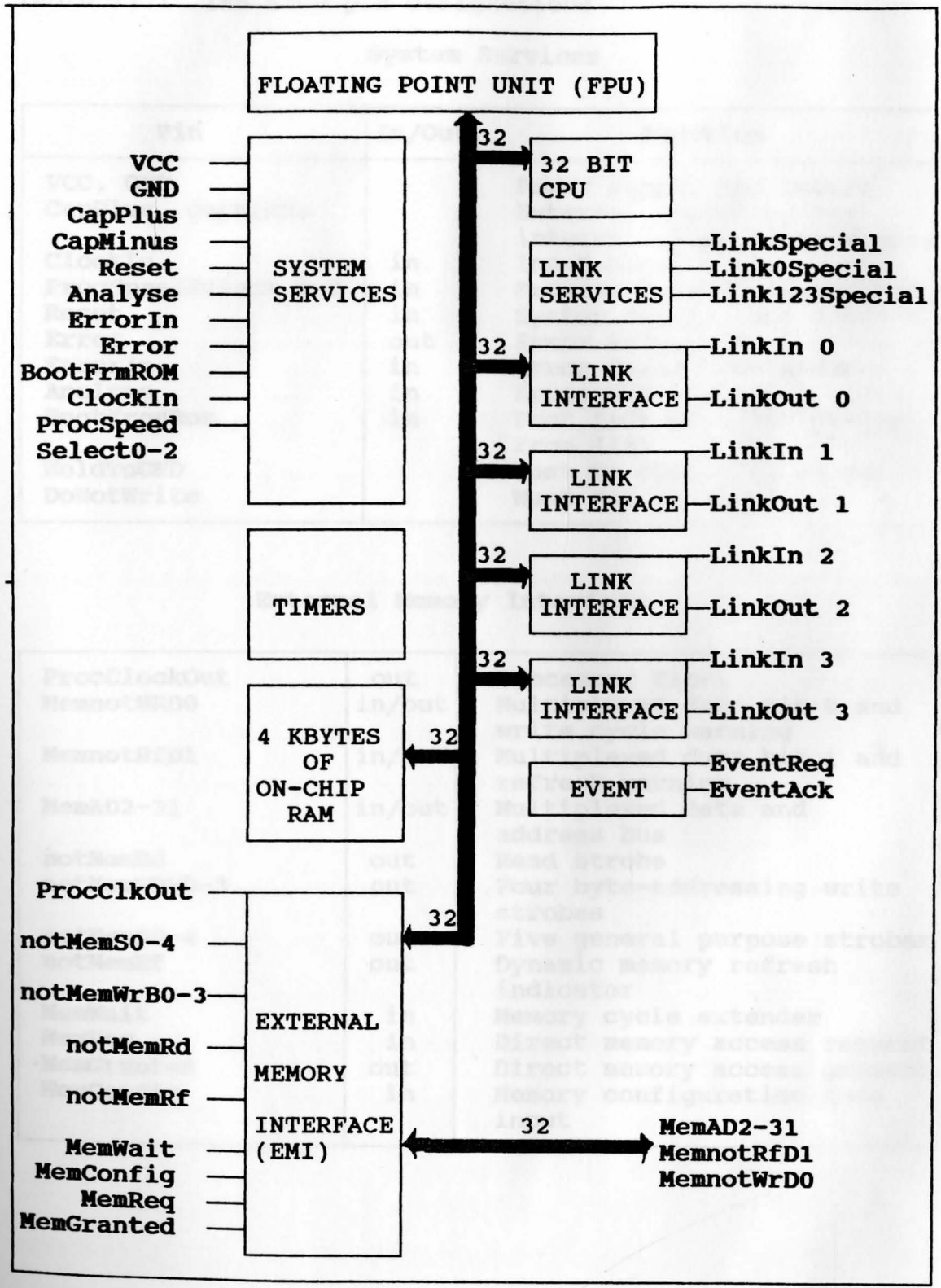


Fig. II.1 IMS T800 architecture

Table II.1 IMS T800 pin designations

System Services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
ClockIn	in	Input clock
ProcSpeedSelect 0-2	in	Processor speed selectors
Reset	in	System reset
Error	out	Error indicator
ErrorIn	in	Error daisychain input
Analyse	in	Error analysis
BootFromRom	in	Boot from external ROM or from link
HoldToGND		Must be connected to **GND**
DoNotWrite		Must not be wired

External Memory Interface

ProcClockOut	out	Processor Clock
MemnotWRD0	in/out	Multiplexed data bit 0 and write cycle warning
MemnotRfD1	in/out	Multiplexed data bit 1 and refresh warning
MemAD2-31	in/out	Multiplexed data and address bus
notMemRd	out	Read strobe
notMemWRB0-3	out	Four byte-addressing write strobes
notMemS0-4	out	Five general purpose strobes
notMemRf	out	Dynamic memory refresh indicator
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemConfig	in	Memory configuration data input

Event/Link

EventReq	in	Event request
EventAck	out	Event request acknowledge
LinkIn0-3	in	Four Serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec
Link0Special	in	Select special Speed for link 0
Link123Special	in	Select special speed for links 1,2,3

The design of the transputer processor exploits the availability of fast on-chip memory by having only six registers (Fig. II.2), which are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

(1) The workspace pointer which points to an area of store where local variables are kept.

(2) The instruction pointer which points to the next instruction to be executed.

(3) The operand register which is used in the formation of instruction operands.

(4) The A, B and C registers which form an evaluation stack.

A, B and C are sources and destinations for most arithmetic and logical operations; loading a value into the stack pushes B into C, and A into B. Storing a value from

A, pops B into A and C into B. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity. Further register details are given in [11].

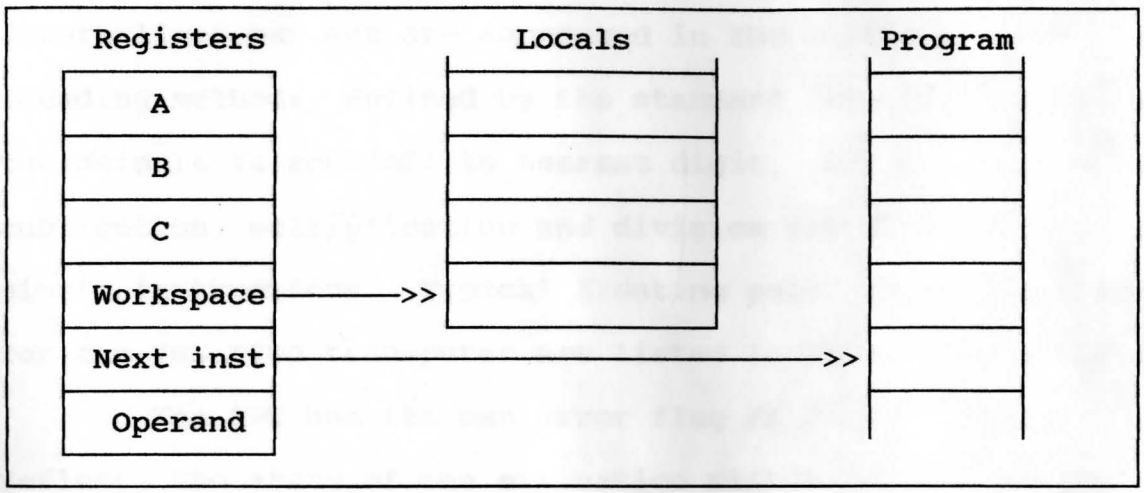


Fig. II.2 T800 registers

64 bit FPU

The 64 bit FPU provides single and double length arithmetic to floating point standard (ANSI-IEEE 754-1985). It is able to perform floating point arithmetic concurrently with the CPU, sustaining in excess of 2.25 Mflops on a 30MHz device. All data communication between memory and the FPU occurs under control of the CPU.

The FPU includes a three-register floating-point evaluation stack which contains the FA, FB and FC registers.

Each of the registers can hold either 32 bit or 64 bit data. When a floating point value is loaded into any of the registers an associated flag indicates the size of the loaded value. The FPU has been designed to operate on both single length (32 bit) and double length (64 bit) floating point numbers, and returns results which conform to the ANSI-IEEE 754-1985 floating point arithmetic standard. Denormalized numbers are supported in the hardware. All rounding methods, defined by the standard, are implemented; the default is roundoff to nearest digit. Basic addition, subtraction, multiplication and division are performed by single instructions. Typical floating point operation times for the IMS T800 transputer are listed in Table II.2 below.

The FPU has its own error flag *FP_Error*. This reflects the state of the evaluation within the FPU and is set in circumstances where invalid operations, division by zero or overflow exceptions would be flagged.

Table II.2 Floating point operation times

	Single Length		Double Length	
Operation	**T800-20**	**T800-30**	**T800-20**	**T800-30**
add	350 ns	233 ns	350 ns	233 ns
subtract	350 ns	233 ns	350 ns	233 ns
multiply	550 ns	367 ns	1000 ns	667 ns
divide	850 ns	567 ns	1600 ns	1067 ns

Further details on the operation of the FPU can be found [11].

System services

System services include all the necessary logic to initialize and sustain operation of the device. They also include error handling and analysis facilities. Some of the pin functions that are specific to the transputer architecture are described below:

CapPlus, CapMinus are connected externally by a low leakage, low inductance 1 uF capacitor for the internally derived power supply for the internal clocks.

Reset is assertive high and the falling edge initializes the transputer, triggers the memory configuration sequence and starts the bootstrap routine.

Analyze will halt the transputer at the nest descheduling point if it is taken high while the transputer is running.

ErrorIn, Error together indicate that an error was detected. An internal error can be caused by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly. The error pin carries the OR'ed output of the internal error flag and the errorin input.

BootFromROM allows the transputer to be externally bootstrapped when connected to high (e.g. to **VCC**).

ClockIn is the standard clock input supplied by the user. High frequency internal clocks are derived from ClockIn and it must be derived from a crystal oscillator since stability is important.

ProcSpeedSelect0-2 pins are used to vary the processor speed in discrete steps as shown in the **Table II.3** below.

Table II.3 ProcSpeedSelect0-2 pin selections

Proc Speed Select2	Proc Speed Select1	Proc Speed Select0	Processor Clock Speed MHz	Processor Cycle Time nS	Notes
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			Invalid
1	1	0	17.5	57.1	
1	1	1			Invalid

Memory

The IMS T800 has 4 Kbytes of On-chip RAM (static memory) for high rates of data throughput. Each internal memory access takes one processor cycle ProcClockOut. The transputer can also access 4 Gbytes of external memory space. Internal and external memory are part of the same linear address space.

Internal memory starts at the most negative address #80000000 and extends to #80000FFF. User memory begins at #80000070; this location is given the name *MemStart*. External memory space starts at #80001000 and extends up through #00000000 to #7FFFFFFF.

External Memory Interface

The external memory interface (EMI) allows access to a 32 bit address space, supporting dynamic and static RAM as well as ROM and EPROM. The associated pin functions are defined in [11].

Event

EventReq and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes EventReq high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes EventAck high and the process, if waiting, is scheduled. EventAck is removed after EventReq goes low.

Link Interface(s)

Four INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link consists of an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received. The IMS T800 link data and acknowledge packets appear in Fig. II.3 below.

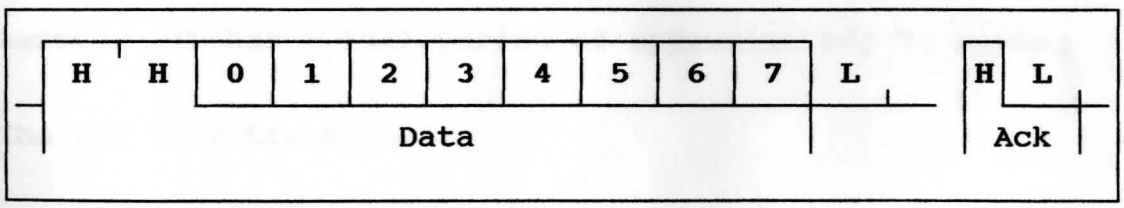


Fig. II.3 Link data and acknowledge

Link speeds can be set by **LinkSpecial, Link0Special** and **Link123Special.** The link 0 speed can be set independently. Table 10.0 shows uni-directional and bi-directional data rates in Kbytes/second for each link speed; **LinknSpecial** is to be read as **Link0Special** when selecting link 0 speed and as **Link123Special** for the others. Data rates are quoted for a transputer using internal memory, and

will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Timers

The transputer has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 milliseconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

The IMS T222 transputer

The IMS T222 transputer has a simlar architecture compared to a T800 transputer with the following significant differences:

(1) The **On-chip RAM** is limited to 2K bytes.

(2) Absence of a **Floating Point Unit.**

(3) **Data bus** and **processor** size is limited to 16 bits.

(4) **External Memory Interface** is limited to a 16 bit address and data bus.

II.2 The OCCAM software model

Concurrency

The world which we inhabit is inherently concurrent. Events happen in both space and time. It is possible for events to occur in the same place one after the other in time (i.e., sequentially), and equally possible for events to occur in different places at the same time (i.e., in parallel or concurrently). The terms **concurrent** and **parallel** have similar but distinct meanings and it is important that they are not confused. Two entities are said to be executing in parallel if at some instant in time both are actually executing. Entities are described as concurrent if they have the potential for executing in parallel. A concurrent programming language will therefore have more than one distinct thread of control.

OCCAM overview

The programming model for transputers is defined by OCCAM. In OCCAM processes are connected to form concurrent systems. Each process can be regarded as a black box with internal state, which can communicate with other processes using point to point communication channels. Processes can be used to represent the behavior of many things: a logic gate, a microprocessor, a machine tool or as in the present thesis, a stirred tank heater.

The processes themselves are finite. Each process starts, performs a number of actions and then terminates. An action may be a set of sequential processes performed one after one another. Since a process is itself composed of processes, some of which may be executed in parallel, a process may contain any amount of internal concurrency, and this may change with time as processes start and terminate. A pair of concurrent processes communicate using a one-way channel connecting the two processes. One process outputs a message to the channel and the other process inputs the message from the channel.

The key concept is that communication is synchronized and unbuffered. If a channel is used for input in one process, and output in another, communication takes place when both processes are ready. The value to be output is copied from the outputting process to the inputting process; the inputting and outputting processes then proceed. Thus communication between processes is like the handshake method of communication used in hardware systems. Since a process may have internal concurrency, it may have many input channels and output channels performing communication at the same time. The property of unbuffered communication between processes can be exploited specifically in real-time applications since time losses are minimized.

OCCAM can be used to program an individual

transputer; the transputer shares its time between the concurrent processes and channel communication is implemented by moving data within the memory (Fig. II.4a). (The numbers indicate the transputer link numbers and P, Q, R indicate the individual processes). When OCCAM is used to program a network of transputers, each transputer executes the process allocated to it (Fig. II.4b). Communication between OCCAM processes on different transputers is implemented directly by transputer links. Thus the same OCCAM program can be implemented on a variety of transputer configurations, with one configuration optimized for cost, another for performance, and another for an appropriate balance of cost and performance.

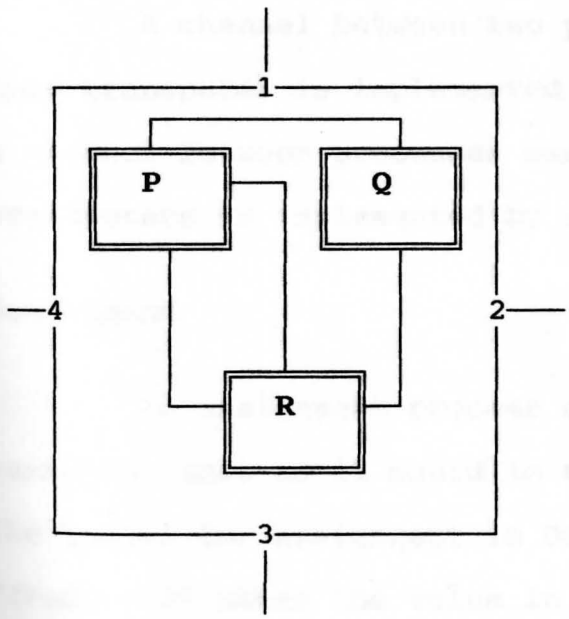


Fig II.4a Single transputer

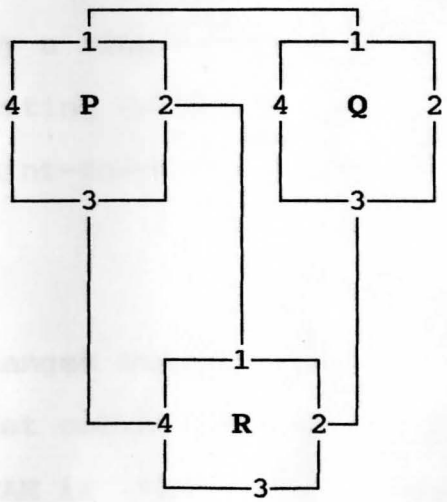


Fig. II.4b Transputer network

Process

The definition of a process is the same as that of a program in an ordinary sequential language; in **OCCAM** more than one process may be executing at the same time, and processes can send messages to one another. All **OCCAM** programs are built from combinations of three kinds of primitive processes. They are assignment, input and output.

Channels

Communication between processes is achieved by means of channels. **OCCAM** communication is point-to-point, synchronized and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links.

Assignment

An assignment process changes the value of a variable, just as it would in most conventional languages. The symbol for assignment in OCCAM is ":=". The process "*fred* := 2" makes the value in the variable *fred* two.

Input

An input process inputs a value from a channel into

a variable. The symbol for input in OCCAM is "?". The input process "*chan3* ? *fred*" takes a value from a channel called *chan3* and puts it into variable *fred*.

An input process cannot proceed until a corresponding output process on the same channel is ready.

Output

An output process outputs a value to a channel. The symbol for output in **OCCAM** is "!". The output process "*chan3* ! 2" outputs the value 2 to a channel called *chan3*. An output process cannot proceed until a corresponding input process on the same channel is ready.

Communication

Communication over a channel only occurs when both input and output processes are ready. If during the execution of a program an input process is reached before its corresponding output process is reached, the input will wait until the output becomes ready. If the output is reached first, it will wait for its input; the communication is synchronized.

SKIP and STOP

SKIP represents a process that starts, does nothing and then finishes. It might be used in a partly completed program in place of a process which will be written later.

STOP process starts but never proceeds and never finishes. It might be used, like **SKIP**, to stand in for a process which has yet to be written.

Combining processes

Several primitive processes can be combined into a larger process by specifying that they should be performed one after the other, or all at the same time. This larger process is called a construction and it begins with an **OCCAM** keyword which states how the component processes are to be combined.

SEQ Construct

SEQ is short for 'sequence' which explains the way in which the processes within this construct are to be executed, i.e., one after another.

```
Ex:   SEQ
        chan3 ? fred
        jim fred + 1
        chan4 ! jim
```

Notice the way that the processes which make up this **SEQ** process are indented by two characters from the word **SEQ**, so that they line up under the **Q**. This is not merely to make the program look prettier, but is the way that **OCCAM** knows which processes are part of the **SEQ**.

PAR construct

PAR is short for 'parallel' and hence all the component processes of a **PAR** start to execute simultaneously.

```
Ex:   PAR
        SEQ
          chan3 ? fred
          fred := fred + 1
        SEQ
          chan4 ? jim
          jim := jim + 1
```

The first two-character indent tells **OCCAM** that the **PAR** process consists of two **SEQ** processes. The second level of indentation shows that each **SEQ** is composed of two primitive processes. In a **PAR**, the written order of the component processes is irrelevant as they are all performed at the same time. All the component processes in a **PAR** start at the same time, and the **PAR** itself terminates when all its component processes have terminated. Several other constructs and the OCCAM syntax are well illustrated in [1] and [13].

II.3 The OCCAM2 toolset for the IBM PC

The OCCAM2 toolset is a set of software tools for developing transputer programs on host systems. Used with the OCCAM libraies, it provides a complete environment for

developing programs on transputers and transputer networks. This toolset allows OCCAM programs to be written using any convenient text editor. Programs are then compiled and linked using programs that are resident on the host or running on the transputer board. Self-booting code for single transputers and multitransputer networks is produced using separate tools, and loaded from the host system down the transputer link.

The OCCAM2 toolset is intended for developing programs on transputers and transputer boards that are loaded from the host via a transputer link. The tools that are used in compiling, linking and downloading the OCCAM2 software onto the designed system are listed:

(1) **ICHECK**

(2) **OCCAM**

(3) **ILINK**

(4) **ICONF**

(5) **ISERVER**

The file extensions that have been frequently encountered in the present software design are listed:

(1) **.btl** - Output file from **ICONF** tool. Loadable code file extension for boot from link boards.

(2) **.cxy** - Linked code output file from **ILINK** tool.

(3) **.inc** - Input file to the **OCCAM** tool consisting of predefined constants and channel protocols.

(4) **.lib** - Output library file from the **ILIBR** tool.

Consists of compiled code.

(5) **.map** - Output file from the **ICONF** tool. Consists of the configuration map in the ASCII format.

(6) **.occ** - Input file to the **OCCAM** tool consists of the OCCAM source code.

(7) **.pgm** - Input file to the **ICONF** tool consists of the configuration description source file.

(8) .txy - Output file to the **OCCAM** tool consists of the compiled code.

The second and third characters *x* and *y* are listed:

x	**Class**
2	T212, T222, M212
4	T414
5	T425
8	T800
a	T800, T425, T414
b	T425, T414
c	T800, T425

y	**Mode**
h	Halt
s	Stop
u	Undefined
x	Universal

ICHECK

The OCCAM 2 checker **icheck** performs a syntax check of the full OCCAM 2 product language but produces no object code. The syntax checking performed by the checker is similar to that of the compiler but it generates more data and displays more information about the errors. By taking advantage of the regular structure of the OCCAM programs **icheck** can recover from errors that cause the compiler to abort, and thereby perform a more comprehensive check. The

checker recognizes the compiler directives **#INCLUDE**, **#USE**, **#IMPORT**, **#SC**, **#OPTION**, and **#COMMENT**. Directives are described in more detail in [10].

To invoke the checker the following command line is used:

icheck *filename* {*options*}

where: *filename* is the name of the file containing the source code. If a file extension is not specified, the extension **.occ** is assumed. If the filename is omitted a brief "help" information is displayed.

options is a list, in any order, of one or more of the options available [10].

The options that have been frequently used for checking the software for the present system are:

(1) **T8** - Check for a T800 processor (the process resident on the T800 based **'ROOT TRAM'** in the system).

(2) **T2** - Check for a T222 processor (the process resident on the T222 based **'GPIB TRAM'** in the system).

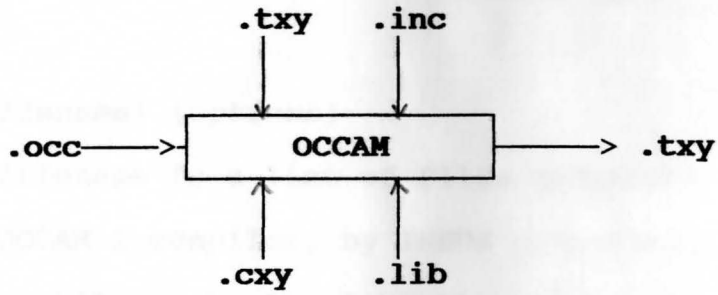
(3) **B** - Displays error messages in brief (single line) format.

Details regarding usage, other options and error messages can be obtained from [10].

OCCAM

The toolset compiler implements the **OCCAM** 2 language targeting to IMS T222, T800 and several other transputers [11]. Each compilation of a program must be targeted at a specific transputer or transputer class and in one of four execution error modes. All components of a program that are to be run on the same transputer must be compiled with the same target processor and error mode.

Six directives, extensions to standard OCCAM are recognized by the OCCAM 2 compiler. These are **#USE**, **#INCLUDE**, **#IMPORT**, **#OPTION**, **#COMMENT** and **#SC**. Compiler directives are described in [10]. The operation of the compiler in terms of the file extensions is described below.



To invoke the compiler the following command line is used:

occam *filename* {*options*}

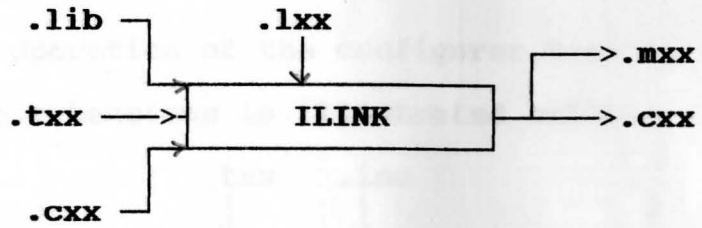
where: *filename* is the name of the file containing the source code. If a file extension is not specified, the extension .occ is assumed. If the filename is omitted brief help information is displayed.

options is a list, in any order, of one or more options available [10].

ILINK

The linker links compiled code into a single object file, resolving all external references. Code files can be separately compiled program units or library files.

The operation of the linker in terms of file extensions is shown below.



To invoke the linker the following command line is used:

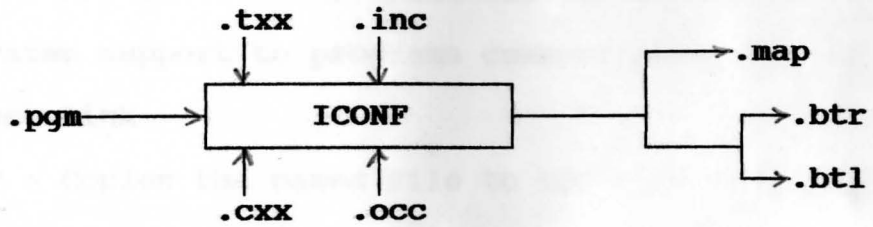
ilink {*filename*} {*options*}

where: *filename* is a list of files generated by the OCCAM 2 compiler, by INMOS compatible compilers, by the librarian, or by the linker. If the 'o' option is not specified the name of the first file in the list is used to generate the output file.
options is a list of any of the available options [10].

ICONF

The configurer takes a configuration description and produces either an object code file ready for loading into a network of transputers, or a configuration map describing the allocation of code and placement of the channels. A configuration description describes how code is to be run on a network of transputers. Code to be run on separate processors is declared as separately compiled units, or included as OCCAM source.

The operation of the configurer tool in terms of toolset file extensions is illustrated below.



To run the configurer the following command line is used:

iconf *filename* {*options*}

where: *filename* is the file containing the configuration code. If no file extension is given **.pgm** is assumed.

options is a list of options available from [10].

ISERVER

The host file server **iserver** provides two functions:

(1) Loading programs and controlling transputer networks.

(2) Runtime access to host services for application programs.

To run the host file server the following line is used:

iserver {*options*}

where: *options* is a list of one or more options listed in [10].

The options that have been used for the present design are

(1) **SB** - boots the program contained in the named file.

(2) **SS** - serves the link, that is, provides host system support to programs communicating on the host link.

(3) **SC** - Copies the named file to the root transputer link.

A more detailed description of the **iserver** an other tools and their implementation aspects are given in [10].

The OCCAM Libraries

A comprehensive set of libraries and 'include' files are provided with the toolset. Some form part of the standard support for the OCCAM language (the compiler libraries), others are user-level libraries to support standard programming tasks such as terminal i/o and file access.

Compiler libraries

The compiler libraries are used internally by code generated by the compiler. With a number of exceptions [10] they are not intended for direct use by the programmer. The compiler references them automatically by searching the directories specified by the **ISEARCH** host environment variable [11].

Maths libraries

The maths libraries provide trigonometric and logarithmic functions for all transputer types supported by the toolset.

I/O libraries

Two libraries containing routines to assist with i/o are provided with the toolset. They are **Hostio** and **Streamio** libraries.

The **Hostio** library is used for file handling, Host access and Terminal i/o. The **Streamio** library is used for general character-based i/o using stream protocols [11], and for controlling the screen display.

Details of these and other libraries are detailed in [11].

CHAPTER III

DESCRIPTION OF THE TRANSPUTER MODULES AND MOTHERBOARDS

III.1 Transputer modules (TRAMs)

TRAMs are small, cost-effective sub-assemblies of transputers and other circuitry (often RAM) with a simple but efficient 16 signal interface standard profiled in modular sizes. The interface accommodates 4 serial transputer links for interprocessor communication, power supply and system signals.

This standard allows the TRAMs to be mounted onto a variety of motherboards which provide specific host interface hardware. Each motherboard can connect to a number of TRAMs and provides facilities for configuring a network of TRAMs for the user specified topology under software control. A software package is provided for motherboards which allows this task to be undertaken with minimum effort [4].

All TRAMs are based upon a single module profile with a defined pin layout. This single format is known as "size 1". The schematic picture of the size 1 TRAM is shown below in Fig. III.1.

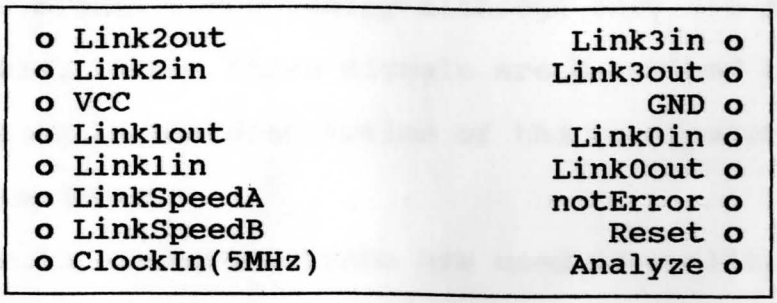


Fig. III.1 Size 1 TRAM footprint

Larger TRAMs are simply a multiple of the size 1 footprint. Thus, a **"size 2"** TRAM occupies two of the sockets into which a size 1 TRAM will plug. In order to avoid confusion, discussions about motherboards always refer to **"slots"**. A slot is one position into which a size 1 TRAM may be plugged. So, a motherboard which has ten slots may have ten size 1 TRAMs or five size 2 or two size 4 and two size 1 or one size 8 or even six size 1 and one size 4. The common pins that are available from the TRAMs are listed below.

STANDARD TRAM PINs

Transputers and therefore TRAMs require three signals to be connected to them to allow them to initialize, and debug so that they can signal an error. These signals are *Reset* for resetting, *Analyze* to allow debugging, and *NotError* to signal an error on a transputer or TRAM (Fig. III.1). These three signals are collectively known as *system services*. The system services for a TRAM are treated

as a single signal conceptually although they are actually three signals. These three signals are described along with the other pins in the description of the transputer architecture above.

The following two TRAMs are used in realizing the network for the process control system.

(1) **IMS B403** (T800 based TRAM)

(2) **IMS B421** (T222 based TRAM)

III.2 IMS B403 TRAM

The IMS B403 is a very compact compute module which provides a full 2Mbytes of memory and still maximizes performance capability. This is achieved by extending the principle of fast on the chip RAM to include 32Kbytes of static RAM which cycles as fast as possible. Any technique which puts most frequently accessed memory locations near the bottom of memory will speed up the processing. This TRAM is the most popular board for running INMOS TDS or Toolset packages.

The IMS B403 packs 11 sq. cm of silicon onto a board the size of a credit card. Four IMS B404s fit onto the IMS B008 in a single slot of the IBM PC. Fifty IMS B403s fit into an ITEM [6], to give 100 Mbytes, 625 MIPS, 250 MWhetstones, with space to spare for other modules. The schematic of the IMS B403 appears in Fig. III.2.

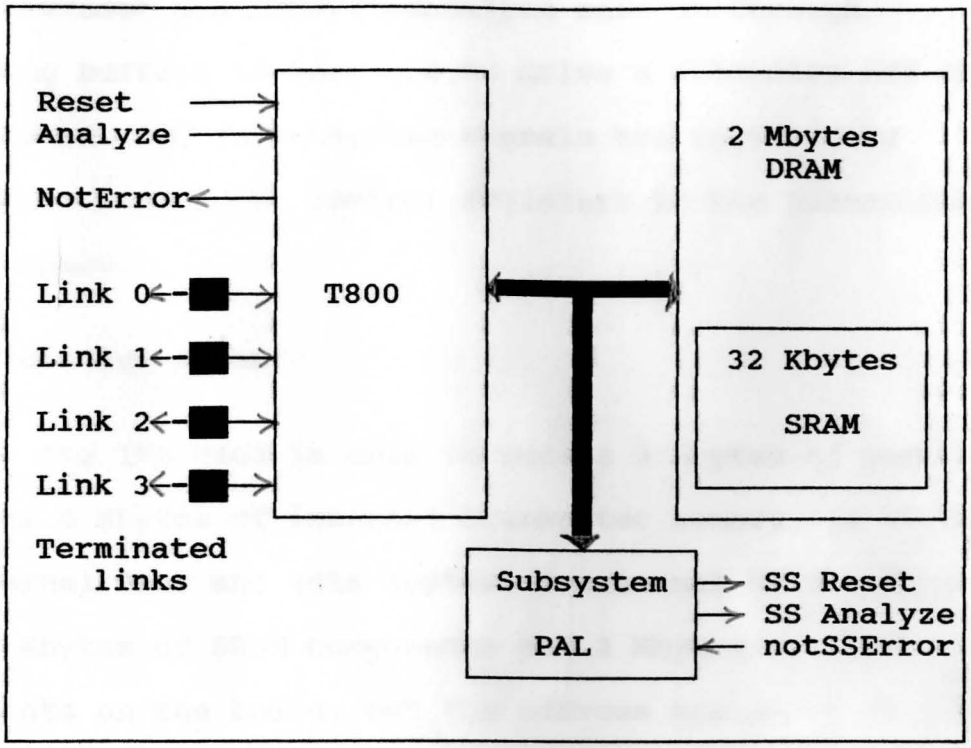


Fig. III.2 IMS B403 TRAM schematic

All the pins except the Subsystem PAL (Programmed Array Logic) pins are described in the transputer hardware section.

Subsystem Signals

The IMS B403 has a subsystem port in addition to the usual TRAM signals. This enables the TRAM to reset or analyze a subsystem of other TRAMs and/or motherboards. The polarity of these signals is the same as that of **Reset**, **Analyze** and **notError** standard TRAM signals. Therefore the IMS B403 subsystem can drive other TRAMs on the same motherboard with no intermediate logic. However,

SubSystemReset and **SubSystemAnalyze** must go through inverting buffers if they are to drive a subsystem off the motherboard. These subsystem signals are accessed by writing or reading to control registers in the transputer memory space.

Memory configuration

The IMS B403 is able to access 2 Mbytes of memory. There is 4 Kbytes of internal transputer memory, 28 Kbytes of external SRAM and 2016 Kbytes of external DRAM. There are 32 Kbytes of SRAM components and 2 Mbytes of DRAM components on the board, but the address spaces of each type of memory are superimposed. The total memory available is limited to 2 Mbytes. This is sufficient to enable the Transputer Development System (TDS) to be run on a single IMS B403 TRAM.

Location of external memory

Table III.1 shows the start address of the external SRAM and Table III.2 shows the start address of the external DRAM on the IMS B403 (the # sign indicates a hexadecimal number). The internal RAM on the IMS T800 occupies the first 4 Kbytes of address space. Since the internal memory on the IMS T800 is 1 cycle, the external SRAM is 3 cycle and the DRAM is 4 (or 5) cycle, a memory speed hierarchy is established. This architecture allows the programmers to

Table III.1 External SRAM addresses

	Hardware byte address
From:	#80001000
To:	#80007FFF

Table III.2 External DRAM addresses

	Hardware byte address
From:	#80008000
To:	#801FFFFF

structure their code for optimum performance, and will become of greater significance when the next faster version of the transputer becomes available.

Subsystem register locations

The subsystem register addresses start at hardware address #00000000 in all TRAMs that utilize a 32-bit processor, allowing software compatibility between TRAMs. These registers are located as shown in Table III.3. Setting bit 0 in either the reset or the analyze registers asserts the corresponding signal. Similarly, clearing bit 0 deasserts the signal. When an error occurs in the subsystem, bit 0 of the error location becomes set. Byte locations #00000008 and #0000000C are unused. The subsystem registers are repeated at every sixteenth byte location in the positive address space. See Fig. III.3.

Table III.3 Subsystem register addresses

Register	Hardware Byte Address
SubSystemReset (write only)	#00000000
SubSystemAnalyze (write only)	#00000004
notSubSystemError (read only)	#00000000

Fig. III.4 shows the schematic picture of the IMS B403 TRAM. Since the IMS B403 contains CMOS components, all normal precautions to prevent static damage should be taken.

The IMS B403 is supplied with spacer pin strips attached to the TRAM pins on the underside of the board. These spacers perform two functions. Firstly, they help to protect the TRAM pins during transit. Secondly, they can be used to space the TRAMs off the motherboard. If there are no components mounted on the motherboard TRAM slot, then the spacer strips are removed before the TRAM is inserted.

If the subsystem pins are required, a 3-way header strip is plugged into the solder-side sockets on the IMS B403.

The IMS B403 is plugged into the motherboard with the silk screened triangle marking pin 1 on the TRAM aligned with the silk screened triangle that appears in the corner of the appropriate TRAM slot.

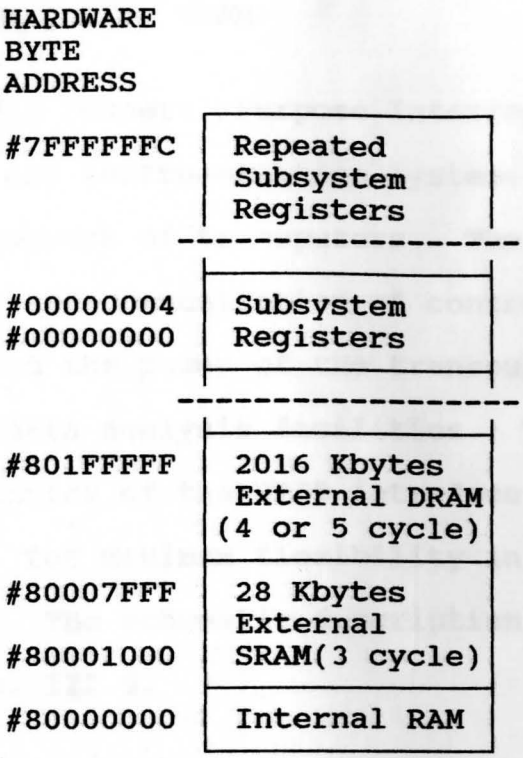


Fig. III.3 IMS B403 subsystem register memory map

Mechanical details and Installation

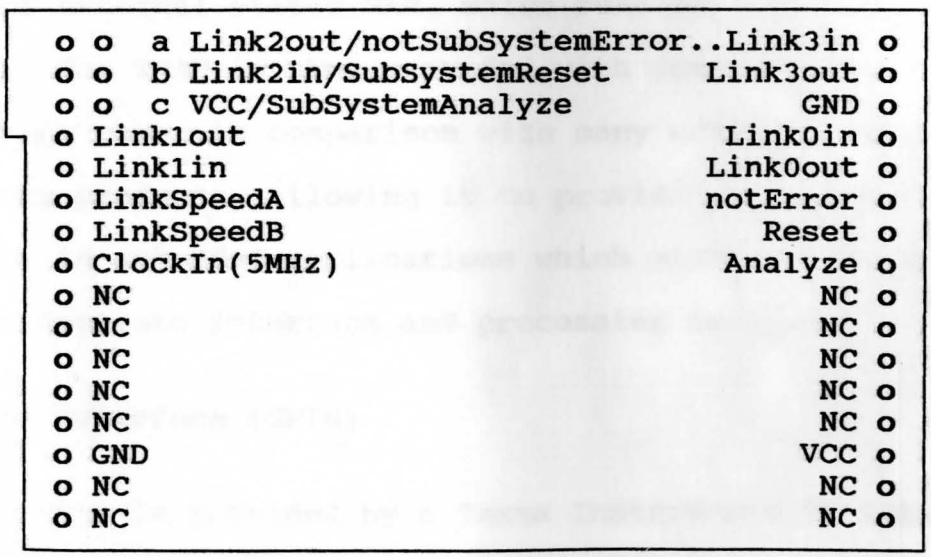


Fig. III.4 IMS B403 schematic

III.3 IMS B421 (GPIB) TRAM

The GPIB (General Purpose Interface Bus) TRAM allows IEEE-488 test and instrumentation systems to be directly connected to network of transputers. The parallel interface permits high speed communication of control and measurement information, and the power of the transputer can provide sophisticated data analysis facilities. The user can define the characteristics of the GPIB interface in terms of address, etc., for maximum flexibility in system configuration. The schematic description of the TRAM appears in Fig. III.5.

Onboard Transputer System

The IMS B421 TRAM has an IMS T222-20 transputer with 4K bytes of fast internal RAM. This is supplemented by 48K bytes of external static RAM, which runs without wait states. The TRAM is thus provided with considerable processing power in comparison with many existing IEEE-488 interface products, allowing it to provide a compact solution in embedded applications which might otherwise require separate interface and processing modules.

IEEE-488 Interface (GPIB)

This is provided by a Texas Instruments TMS9914A GPIB controller, in conjunction with SN75160B and SN75162A buffers. These devices allow the IMS B421 to act as a

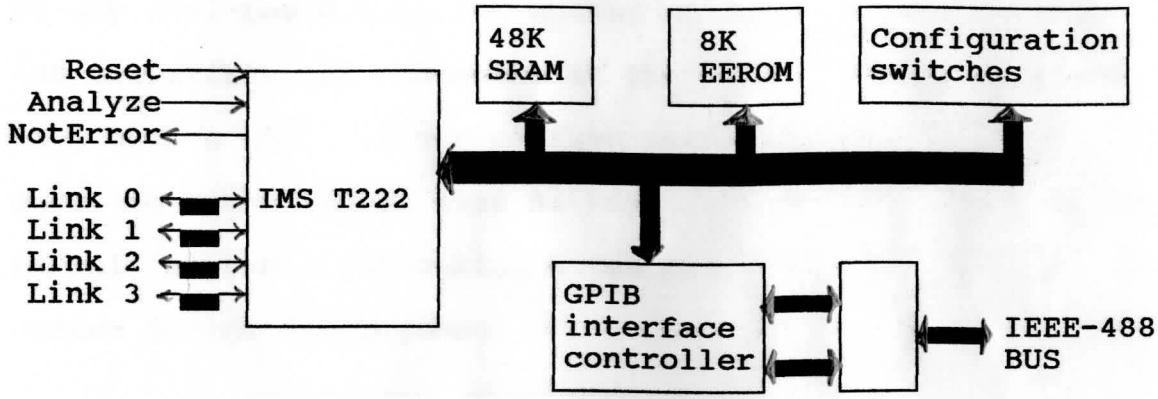


Fig. III.5 IMS B421 architecture

System Controller, non-system Controller, Talker or Listener, and ensure full electrical compliance with the IEEE-488 standard.

Electrically Erasable Read Only Memory (EEROM)

The IMS B421 TRAM contains an EEROM device of 8 Kbyte capacity. This is provided essentially to assist in implementing the requirements of IEEE-488.2, which calls for compliant devices to accept, retain and return various identifying information upon demand. The content of these messages cannot be determined in advance by INMOS, so the EEROM is provided as a non-volatile means of retaining the necessary character strings, which may be conveniently stored in the device by IMS F001 [8] commands. The device capacity is more than enough to store the information required for compliance with the standard, so the remainder may be allocated to any purpose defined by the user, and is easily accessed via additional IMS F001 commands [8].

26-way dual-row 0.1 in.IDC socket at one end, and an IEEE-488 compatible IDC connector at the other. It is stressed that such a connection minimizes performance as well as cost, so cable length must be kept as short as possible. It is only suitable for making a temporary connection to a system during development.

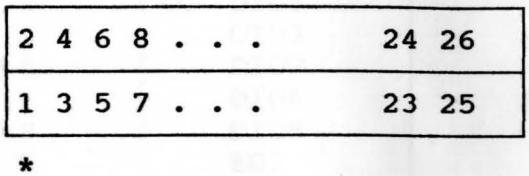


Fig. III.6 J1 - IEEE 488 connector

Table III.4 below shows the J1 signal assignments. Further details of the IEEE-488 standard appear in [3].

Auxiliary connector J2

The IMS B421 also has a similar connector, J2, (Fig. III.7) with four pins arranged in a single row on 0.1 inch pitch.

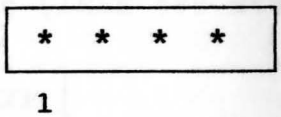


Fig. III.7 J2 auxiliary connector

Pin 1 of J2 is marked with a yellow dot; pin numbering proceeds along the row of contacts. Connection to J2 may be made with an IDC connector and ribbon cable, as described for J1 above; the termination at the far end of the cable is

at the user's discretion.

Table III.4 J1 Signal assignment

IMS B421 J1	IEEE-488 signal name	IEEE-488 compatible connector pin number
1	DIO1	1
2	DIO5	13
3	DIO2	2
4	DIO6	14
5	DIO3	3
6	DIO7	15
7	DIO4	4
8	DIO8	16
9	EOI	5
10	REN	17
11	DAV	6
12	Gnd	18
13	NRFD	7
14	Gnd	19
15	NDAC	8
16	Gnd	20
17	IFC	9
18	Gnd	21
19	SRQ	10
20	Gnd	22
21	ATN	11
22,23,24	Gnd	23
25, 26	Gnd	not used

Table III.5 J2 pin (IEEE 488 status) assignment

IMS B421 J2 Pin	signal name
1	IEEE-488 SRQ line status
2	IEEE-488 IFC line status
3	IEEE-488 REN line status
4	TRIG

The status of the three IEEE-488 lines is provided (Table III.5) so that the hardware in which the IMS B421 is

embedded may directly receive Service Request messages, Interface Clear messages, and Remote/local status. Note that the J2 pins have the same logic polarity as the IEEE-488 lines, i.e. TTL logic 0 level indicates TRUE.

The TRIG signal is produced by the IMS B421 when it receives a Group Execute Trigger message [8]. The pin goes to TTL logic 1 level to indicate this event, and may be used to trigger other embedded functions such as some form of data acquisition hardware.

Bus address jumpers, JP1 to JP5

The intended address value is indicated as a binary number on these jumpers (Table III.6). The encoding considers bit significance to begin at JP1 and increase in numeric sequence to JP5; presence of a jumper forces a bit's contribution to zero, whereas absence allows it to contribute its weighted value. For example, for the present project, the address 28_{10} is encoded as follows:

Table III.6 B421 base address jumper pin assignment

Jumper	Status	Bit Significance	Contribution
JP1	Present	1	(none)
JP2	Present	2	(none)
JP3	Absent	4	4
JP4	Absent	8	8
JP5	Absent	16	16
	Address		16+8+4 = 28

Device capability jumpers, JP6 and 7

The IEEE-488 capability of the device is selected via these jumpers to be a controller, talk & listen as shown in Table III.7:

Table III.7 B421 device capability jumper pin assignment

JP7	JP6	Capability
Absent	Absent	Controller, talk & listen

Further details of the jumper selection appear in [7].

Bus drive selection jumper, JP8

When this jumper is present the IMS F001 software will configure the IMS B421 for open-collector drive of IEEE-488 data signals. When the jumper is absent, as in the case of configuration for the present project, the tri-state drive is selected.

Data protect jumper, JP9

The IMS F001 software takes the presence of this jumper to indicate that EEROM contents are NOT protected; IMS F001 commands which involve modification of EEROM contents will be accepted and obeyed. Conversely, absence of this jumper is taken to mean that EEROM contents should not be altered. Any IMS F001 command which seeks to perform such an alteration will be rejected.

Memory configuration

Fig. III.8 shows a memory map for the system.

Subsystem register locations

The subsystem registers are not implemented in the standard way on the IMS B421, but are part of the I/O ports.

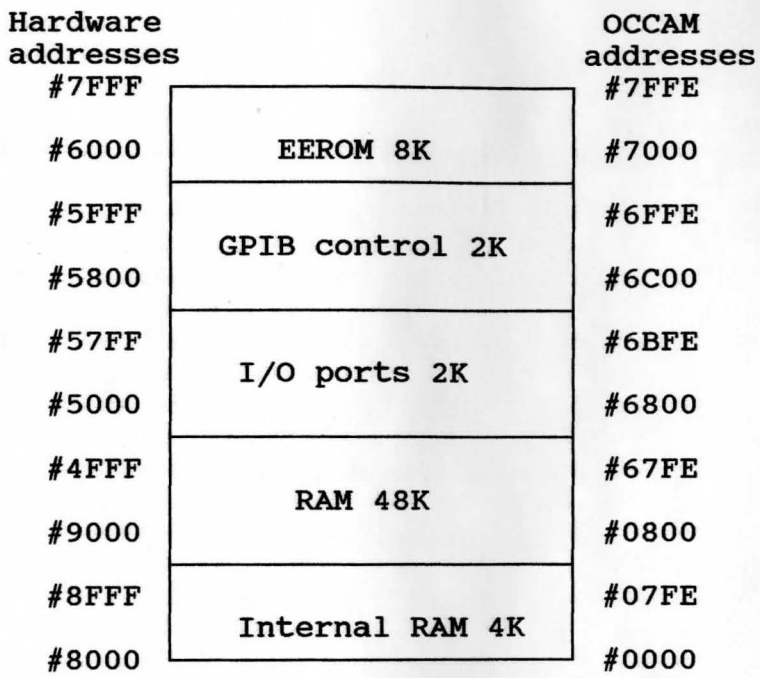


Fig. III.8 IMS B421 memory map

The IMS B421, whose schematic is shown in Fig. III.9, is also composed of CMOS components and hence all normal precautions to prevent static damage should be taken. Similar to the IMS B403, the IMS B421 is also supplied with spacer pin strips and the 3-way header strip (for subsystem signals). Further details appear in [7].

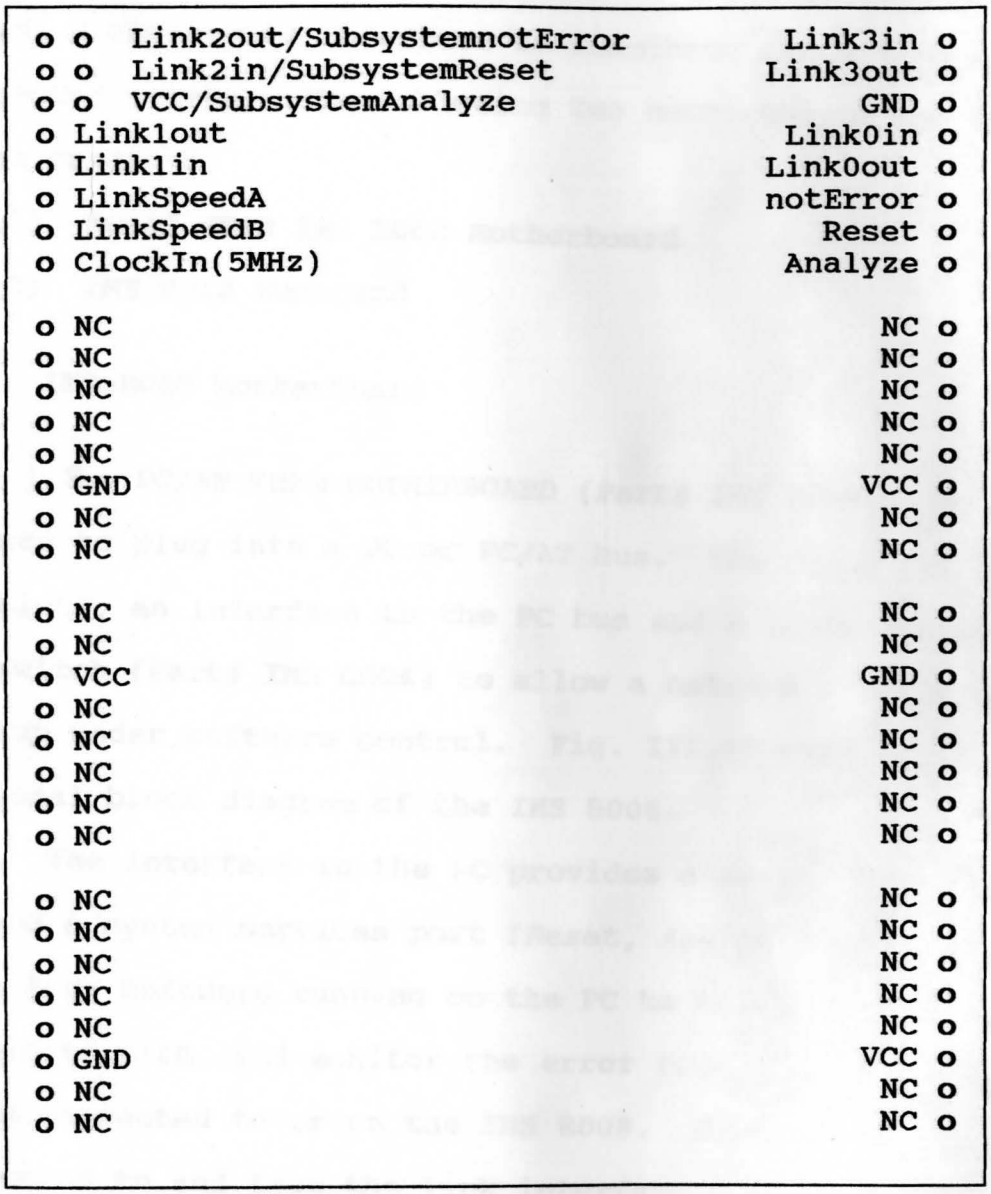


Fig. III.9 IMS B421 footprint

III.4 Transputer motherboards

A TRAM motherboard has a number of slots into which TRAMs can be plugged. Each of these slots contains the necessary connections to power, clock, reset signals

and the transputer links. The motherboard provides a method of connecting TRAMs together and may also include special circuitry to interface to something other than a transputer system. The following two motherboards are used in the project:

(1) **PC/AT TRAM IMS B008 Motherboard**

(2) **IMS B012 Eurocard**

III.5 IMS B008 Motherboard

The PC/AT TRAM MOTHERBOARD **(Part# IMS B008)**, is designed to plug into a PC or PC/AT bus. The board has ten TRAM slots, an interface to the PC bus and a programmable link switch **(Part# IMS C004)** to allow a network of TRAMs to be setup under software control. Fig. III.10 provides a functional block diagram of the IMS B008.

The interface to the PC provides a single transputer link and a system services port (*Reset*, *Analyze* and *Error*). This allows software running on the PC to reset, analyze, communicate with, and monitor the error flag of a transputer network connected to or on the IMS B008. Data can be transferred to and from the link interface using programmed I/O or a DMA transfer mechanism allowing data transfer to go on without processor intervention. Interrupts can be generated on link events; on error being asserted, or at the end of a DMA transfer, freeing the processor from polling the IMS B008 to detect these events.

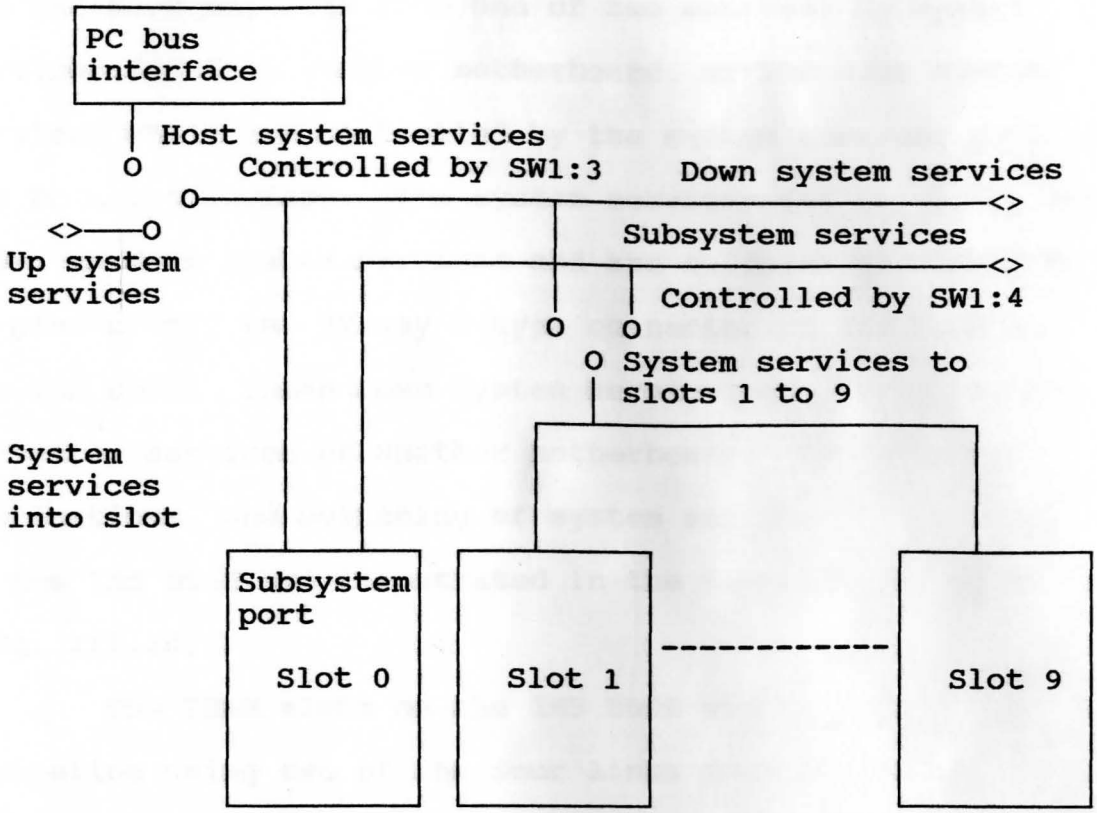


Fig. III.10 IMS B008 functional block diagram

The system services are generated by the system services port of the IMS B008 PC bus interface, or by a *subsystem port* on a TRAM. TRAMs with a subsystem port have three extra connections which are made via a row of three sockets on the underside of the TRAM. The IMS B008 has a corresponding row of three sockets underneath the slot 0 position only. To connect the subsystem port on the TRAM to the IMS B008 a strip of three double-ended pins is inserted in the sockets in the TRAM and the IMS B008. The specific details of installation may be obtained from [4].

System services for a TRAM plugged into slot 0 on

the IMS B008 and come from one of two sources: *Up* system services fed from another motherboard, or the *Host* system services which are controlled by the system services port of the PC bus interface. The system services fed to slot 0 are known as *down* system services and are buffered and connected to pins of P2, the 37 way D-type connector at the back of the IMS B008. These *Down* system services can be fed to the *Up* system services of another motherboard. The sources, destinations, and switching of system services on the IMS B008 are illustrated in the functional diagram (Fig. III.10).

The TRAM slots on the IMS B008 are connected into a pipeline using two of the four links from each slot. The remaining two links from slots 1 to 9 and link 3 from slot 0 are connected to the programmable link switch which allows these links to be connected together via software. Control and configuration (programming) of the link switch is performed by a 16-bit transputer (IMS T222).

The IMS B008 performance is monitored by the following prominent devices whose operation is briefly described:

(1) Programmable Link Switch (IMS C004)

(2) Network Configuration Processor (IMS T222)

(3) Link Interface Adaptor (IMS C012)

(4) PC BUS Interface

Programmable Link Switch (IMS C004)

The IMS C004 is a programmable link switch designed to provide a full crossbar switch between 32 link inputs and 32 link outputs. The IMS C004 is internally organized as a set of thirty 32-to-1 multiplexers. Each multiplexer has associated with it a six bit latch, five bits of which select one input as the source of data for the corresponding output. The sixth bit is used to connect and disconnect the output. These latches can be read and written by messages sent on the configuration link via **ConfigLinkIn** and **Config LinkOut**.

The output of each multiplexer is synchronized with an internal high speed clock and regenerated at the output pad. This synchronization introduces, on an average, a 1.75 bit time delay on the signal. Since the signal is not electrically degraded in passing through the switch, it is possible to form links through an arbitrary number of link switches. It is also possible to use a single IMS C004 as a component of a larger link switch.

Inputs and outputs of the IMS C004 are individually identified by numbers in the range 0 to 31. A configuration message consisting of one, two or three bytes is transmitted on the configuration link. The IMS C004 must be hard reset after power up by pulsing the Reset pin high for the minimum time specified.

The link bandwidth may be lower than for a simple

transputer-to-transputer connection, depending upon the type of transputers on the TRAMs at both ends of the link which passes through an IMS C004. For instance, two IMS T800 transputers connected together will give uni-directional link bandwidth of 1.7 Mbytes/s. However, with one IMS C004 switching the link, the link bandwidth is 1.3 Mbytes/s. With two IMS C004s switching the link, as is the case with some board-to-board links using IMS B008s, the link bandwidth will be 800 Kbytes/s.

Theoretically it is possible to change the configuration of the IMS C004s while a program is executing on the TRAM array. This may be useful, for example, in a system which needs a particular network during a data gathering phase but a completely different network during a data processing phase. The basic idea is that providing there is no traffic on a link, the path it takes through an IMS C004 can be switched. After switching, processing can proceed using the new network. Obviously this requires careful synchronization between all the programs in all the TRAMs; this is usually achieved via the links which are being switched.

Network Configuration Processor (IMS T222)

The network configuration processor, a 16 bit transputer (the IMS T222), is used to route configuration data on the IMS B008. Network configuration data is

received on link 1, known as *ConfigUp* of the IMS T222 (referred to as the T2 for brevity) either from a TRAM in slot 0 (root TRAM) or from another board's network configuration processor if the board is in a pipeline of boards. This pipeline of network configuration processors is made in a similar way as that of a pipeline of TRAMS on multiple boards. The link *ConfigDown* on the first board in the configuration pipeline connects to *ConfigUp* on the next board and so on down the pipeline.

Software running on the T2 examines the configuration data for connections to be made by the IMS C004 to which the T2 has a link connection. This connection data is extracted from the configuration data and the connections are made by sending a set of messages to the IMS C004 via link 3 of the T2. Configuration data for the boards further down the configuration pipeline is then sent out on link 2 (*ConfigDown*). This pipeline arrangement of T2 processors is used on all transputer motherboards that have IMS C004 link switches on them. Thus arbitrary large networks of mixed transputer motherboards can be configured by sending configuration data down a single link, *ConfigUp*, at the head of the configuration pipeline.

Link Interface Adaptor (IMS C012)

The IMS C012 provides for full duplex transputer

link communication with standard microprocessor and sub-system architectures, by converting bi-directional serial link data into parallel data streams. Status and data registers for both input and output ports can be accessed across the byte-wide bi-directional interface. Two interrupt outputs are provided: one to indicate input data available and one for output buffer empty.

The IMS C012 link runs at either the standard speed of 10 Mbit/s or at the higher speed of 20 Mbit/s. Data reception is asynchronous, allowing communication to be independent of clock phase. The various link adaptor registers and their memory addresses are listed in table III.3. The link adaptor input and output processes are described below.

LINK ADAPTOR INPUT PROCESS

A data byte received on the C012 link is transferred into the *input data register* and the *data present* flag set in the *input status register*. If interrupts are enabled, a link data input interrupt is generated. A processor controlling the PC bus will, either in response to the interrupt or in a polling loop, examine the *input status register*. The *data present* flag will be set to signify that valid data is in the *input data register*. The processor then reads the data byte.

If a DMA transfer from the IMS B008 to the PC memory has been set up then the DMA logic in the PC and the control

logic on the board will transfer the data byte from the *input data register*. The process then reads the data byte. A new data byte can now be received and the process repeats.

LINK ADAPTOR OUTPUT PROCESS

The *output ready* flag will be set in the *output status register*. If interrupts are enabled for this event, an interrupt will be generated. The processor, either after receiving an interrupt or in a polling loop, reads the *output status register*. It will determine from the *output ready* flag that a byte may be written to the *output data register*. It then writes the byte to the *output data register*. The byte is transmitted on the C012 link output. When the link adaptor is ready to transmit another byte the *output ready* flag will be set. The DMA logic in the PC and the control logic on the board will transfer the data byte from PC memory to the *output data register* without intervention from the processor if a DMA transfer from PC memory to the IMS B008 has been set up.

PC BUS Interface

The IMS B008 has been designed to work when plugged into either a PC/AT bus slot or a PC bus slot. The bus interface on the IMS B008 has four functions to perform:

(1) Convert the 8 bit parallel transfers on the PC bus to serial link transfers, and vice versa.

(2) Provide a system services port.

(3) Control DMA transfers.

(4) Generate interrupts on link interface events, on the assertion of transputer error, or on DMA transfer end.

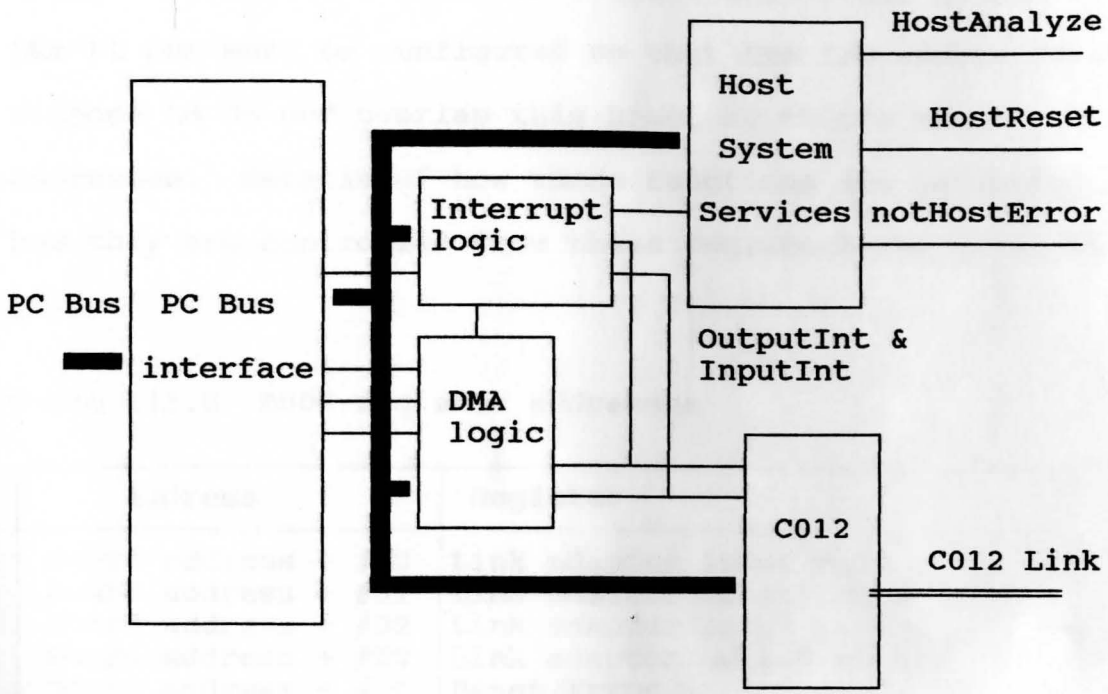


Fig. III.11 PC bus interface functional diagram

The block diagram of the PC bus interface is given in Fig. III.11. The PC the PC bus interface has a number of registers which are mapped into the I/O address space of the PC bus, separate from the memory address space. They are located on a thirty-two byte long block of I/O addresses decoded by the IMS B008. This thirty-two byte block can have a base address of #150, #200, or #300 set by option switches. A memory map of the registers is given in table 3.3. Only nine of the thirty-two locations have

registers mapped into them; writing to the remainder of the I/O locations will have no effect and reading these locations will result in un-defined data being returned.

The IMS B008 is however still driving the bus when these addresses are read, this means that other boards on the PC bus must be configured so that the I/O addresses they respond to do not overlap this block of thirty two addresses. Details of how these functions are performed and how they are controlled from these registers are given in [4].

Table III.8 B008 register addresses

Address	Register
Board address + #00	Link adaptor input data
Board address + #01	Link adaptor output data
Board address + #02	Link adaptor input status
Board address + #03	Link adaptor output status
Board address + #10	Reset/Error
Board address + #11	Analyze
Board address + #12	DMA request
Board address + #13	Interrupt enable
Board address + #14	DMA and interrupt channel select

III.6 Support software for the B008

The S708 software supports the use of an IMS B008 board in an IBM PC/AT. The S708 is a software package consisting of the following tools which are used for loading transputers via a server.

(1) MS-DOS device driver (**S708DRIV.SYS**)

(2) INMOS server (**ISERVER.EXE**)

(3) Module Motherboard Software (**MMS2.B4**)

The DOS device driver is provided to interface the IMS B008 to the DOS operating system. The INMOS server enables programs to be run on the B008. It also loads programs to transputer networks and provides file and terminal services to the executing program. The module motherboard software (MMS) is used to set the programmable switches on the B008 motherboard and any other transputer boards connected to the B008. These switches determine the topology of the transputers hosted on the B008 and associated boards. The MMS also contains a network mapper (*worm*) program which is used to explore the inter--connections of these transputers and provide a means of checking the topology.

MS-DOS device driver

A DOS device driver (**S708DRIV.SYS**) is used to interface the IMS B008 to the DOS operating system. Having physically installed both the hardware and the software components in the PC it is necessary to tell the DOS to recognize the new device. This is done by adding the following line describing the device driver to the CONFIG.SYS file in the root directory of the boot disk.

DEVICE = *pathname* [**/A** *address*] [**/D** *chan* ¦ **N**] [**/N** *name*] [**/I** *int_num*]

Pathname is the full DOS pathname of the

device driver file.

Address is the I/O address of the B008 card, as set by the hardware switches on installation.

Chan is the DMA channel number (0, 1, 3, or N). If 'N' is specified then the driver will not attempt to use the DMA facilities of the B008.

Name is the DOS device name which the device will assume. If INMOS software products are to be used then the default name to be used is 'LINK1'. The name cannot be more than the DOS limit of eight characters.

Int_num is the interrupt request line used by B008, which should be set to the correct value for the board. The default is Irq3.

The following line is used in CONFIG.SYS to describe the device driver for the present project.

DEVICE = c:\S708\S708DRIV.SYS /A 150 /LINK1 /D 1

Driver file name = S708DRIV.SYS

Pathname = c:\S708\S708DRIV.SYS

Address = 150 (HEX)

Name = LINK1

Channel (DMA) = 1

NOTE: The default interrupt request line **IRQ3** is used.

INMOS Server

Programs are run on the B008 by using the server program provided. The server loads programs to transputer networks and provides file and terminal services to the executing program. Both the module motherboard software and the WORM are executed this way. All the sources for the ISERVER are found in the **ISERVER** directory and **ISERVER.EXE** is the executable copy of the ISERVER.

A detailed explanation of the ISERVER tool appears in [10].

Module Motherboard Software

The range of INMOS Module Motherboards and Modules allows many different configurations of modules and the connections between them to be specified without making physical changes to the boards. The configuration is performed by sending configuration data to the IMS C004 link switch(es) on the board(s). The MMS is designed to make it easy to generate data needed to configure a system of motherboards. The MMS presents a menu-driven interface and provides interactive control of a motherboard or a system of motherboards.

A terminal description file called **PCMMS.ITM** is used by the MMS and hence a line is included in the autoexec.bat file as follows:

```
set ITERM = PCMMS.ITM
```

Running the MMS

To run the MMS, the following command line is entered at the DOS prompt:

iserver /sb mms2.b4 *softwire hardwire*

replacing *softwire* and *hardwire* by files containing the softwire description and hardwire description respectively. The MMS will display the following menu options and prompt **key command.** At this point selection is made for the command codes listed on the menu.

H - Help

Q - Quit

S - Set C004 links

C - Check source files

T - Toggle diagnostics

N - Network mapper

M - Manual command entry

L - Change link numbers

V - View source files

R - Reset subsystem

I - Initialize C004s

B - Create a bootable file

O - Create an OCCAM table

In order to be able to configure the links connecting the IMS C004s on the motherboards the MMS reads files, known as the **'softwire '** and **'hardwire'** files.

Hardwire definition

This section describes how to define the hardware configuration of a motherboard system. The MMS needs to know how the slots, IMS C004s and edges are connected together on the board in order to be able to determine whether a particular set of softwire connections is possible or not. The **hardwire** file contains a description of the hardware configuration of the boards being used. Once this description has been set up no changes will have to be made unless physical changes are made to the motherboard system. The following sections will describe what is required in each section of a board definition, including some examples. Given below is a sample **'hardwire'** file for two IMS B008 motherboards connected in a chain.

```
DEF B008ONE  -- FIRST B008 IN THE CHAIN IS NAMED B008ONE
  -- DESCRIPTION OF THE COMPONENTS ON THE B008
  SIZES
    T2 1
    C4 1
    SLOT 10
    EDGE 10
  END
  -- SPECIFY THE LINK NUMBER (CONFIG LINK) IN THE T2
  -- TO C4 CONNECTION
  T2CHAIN
    T2 0, LINK 3 C4 0
  END
  -- DESCRIPTION OF THE ACTUAL WIRE CONNECTIONS ON THE BOARD
  HARDWIRE
    -- DESCRIPTION OF THE SLOT TO SLOT CONNECTIONS
```

```
    SLOT 0, LINK 2 TO SLOT 1, LINK 1
    SLOT 1, LINK 2 TO SLOT 2, LINK 1
       .
       .
    SLOT 8, LINK 2 TO SLOT 9, LINK 1

    -- DESCRIPTION OF THE C4 LINK SWITCH TO SLOT CONNECTIONS

    C4 0, LINK 10 TO SLOT 0, LINK 3
    C4 0, LINK 1 TO SLOT 1, LINK 0
       .
       .
  END
DEF B008TWO   -- SECOND B008 IN THE CHAIN IS NAMED B008TWO

  -- DESCRIPTION OF THE COMPONENTS ON THE B008

  SIZES
    T2 1
    C4 1
    SLOT 10
    EDGE 10
  END

  -- SPECIFY THE LINK NUMBER (CONFIG LINK) IN THE T2
  -- TO C4 CONNECTION

  T2CHAIN
    T2 0, LINK 3 C4 0
  END

  -- DESCRIPTION OF THE ACTUAL WIRE CONNECTIONS ON THE BOARD

  HARDWIRE

    -- DESCRIPTION OF THE SLOT TO SLOT CONNECTIONS

    SLOT 0, LINK 2 TO SLOT 1, LINK 1
    SLOT 1, LINK 2 TO SLOT 2, LINK 1
       .
       .
    SLOT 8, LINK 2 TO SLOT 9, LINK 1

    -- DESCRIPTION OF THE C4 LINK SWITCH TO SLOT CONNECTIONS

    C4 0, LINK 10 TO SLOT 0, LINK 3
    C4 0, LINK 1 TO SLOT 1, LINK 0
       .
       .
  END

  -- END THE HARDWIRE DESCRIPTION
```

PIPE B008ONE, B008TWO END

Softwire Definition

The softwire connections allow links on modules on a motherboard to be connected to other modules and edges, without requiring a direct hardwired route between the two. Instead the MMS routes the channels via the IMS C004s on the motherboard. The connections that can be made depends on how the IMS C004s and the module slots are physically connected to each other.

A *softwire* description corresponding to the *hardwire* example above has the following basic structure:

```
SOFTWIRE

  PIPE B008ONE -- CONNECTIONS FOR THE FIRST BOARD
    SLOT 1, LINK 0 to SLOT 3, LINK 0
      .
      .
  PIPE B008TWO -- CONNECTIONS FOR THE SECOND BOARD
    SLOT 2, LINK 3 TO EDGE 3
      .
      .
END
```

Further details regarding *softwire*, *hardwire* and multiple board connections appear in [4].

III.7 IMS B012 Eurocard

IMS B012 is a eurocard TRAM motherboard which is designed to fit into standard card cages such as the INMOS ITEM or used for stand-alone operation. It has slots for up to 16 TRAMs - the smallest that can be accommodated being of 'size 1'. Each module site, or 'slot', has connections for

four INMOS links which are designated *link 0*, *link 1*, *link 2* and *link 3*. TRAMs which are larger than size 1 can be mounted on the B012. A larger module occupies more than one slot and need not use all if the available link connections provided by the slots which it occupies.

The B012 has two IMS C004 link switch ICs. These devices are able to connect together links from the slots and 32 links which are available on an edge connector. The connections can be changed by control data passed to the board down a configuration link, which may come from some master system or from one of the TRAMs on the B012 itself.

Hardware Description

The 16 module sites or slots provided by the IMS B012 are 16-pin sockets in accordance with the TRAM Specification [12]. The slots are numbered as shown on the board silk screen and in Fig. III.12. The IMS B012 has two DIN41612 96-way edge connectors, P1 and P2. These carry almost all signals and power to/from the board and are easily identified from the board silk screen printing and from Fig. III.12. P2 carries power, pipeline and configuration links and system control signals (reset and analyze and error).

Full details of the connections to every pin on P1 and P2 are to be found in [5].

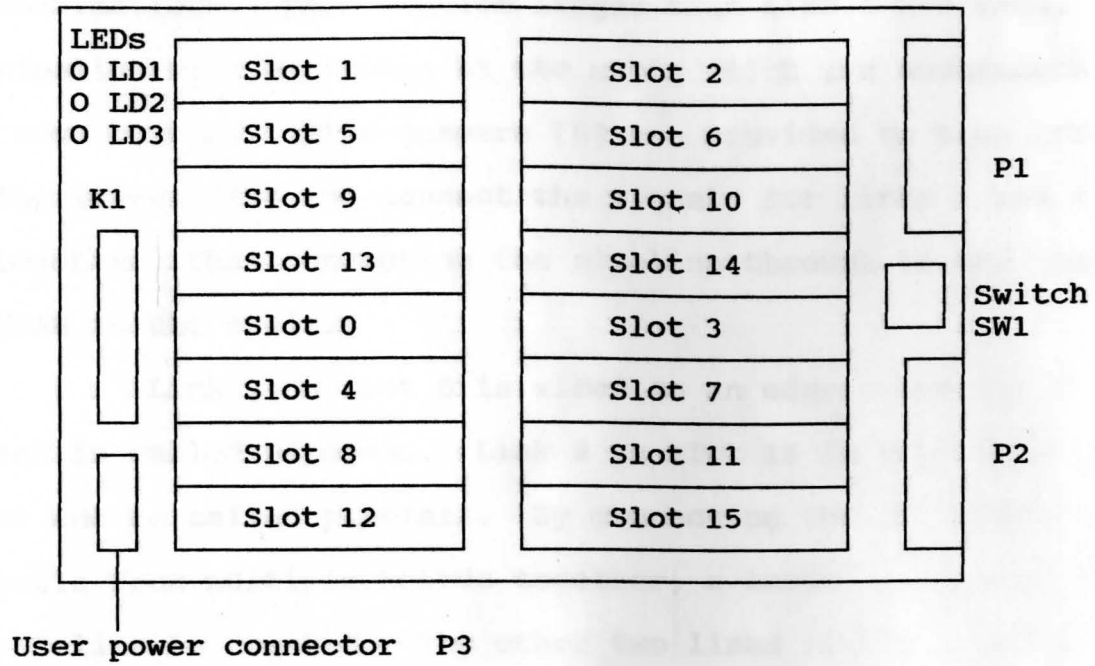


Fig. III.12 IMS B012 schematic

Link Connections

Two links from each slot (links 1 and 2) are used to connect the 16 slots as a 16-stage pipeline (in a pipeline, multiple processors are connected end-to-end as in Fig. III.13.)

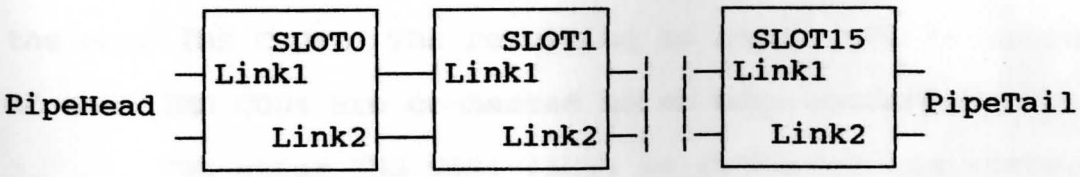


Fig. III.13 Pipeline connection for B012 slots

The pipeline is actually broken by jumper block by jumper block K1 [5]. K1 will usually be jumpered in the standard way to give a 16-stage pipeline but can allow other

combinations. When modules larger than size 1 are used, the pipeline will be broken at the slots which are underneath large modules. Pipe-jumpers [5] are provided to plug into the unused slot and connect the signals for links 1 and 2 together, thus connecting the pipeline through to the next TRAM in the chain.

Link 1 on slot 0 is wired to an edge connector (P2) and is called *pipehead*. Link 2 on slot 15 is also taken to P2 and is called *pipetail*. By connecting the pipe heads and tails from multiple boards together, a large, multi-board pipeline is created. The other two links (links 0 and 3) of each slot are, in general, connected to two IMS C004 programmable link switches (For detailed information on the IMS C004 see IMS C004 link switch data sheet).

The link output signals from all the link 0s on all the slots (16 signals) are connected to 16 inputs of one IMS C004 (IC2). The link input signals from all the link 3s on all the slots (16 signals) are connected to 16 outputs of the same IMS C004. The remaining 16 inputs and 16 outputs of that IMS C004 are connected to an edge connector (P1).

The other IMS C004 (IC3) is connected similarly, except that 16 of its inputs are connected to the outputs of all link 3s on all the slots, and 16 of its outputs are connected to the inputs of all link 0s on all the slots. The remaining inputs and outputs are connected to P1. The schematic appears in the Fig. III.14 below.

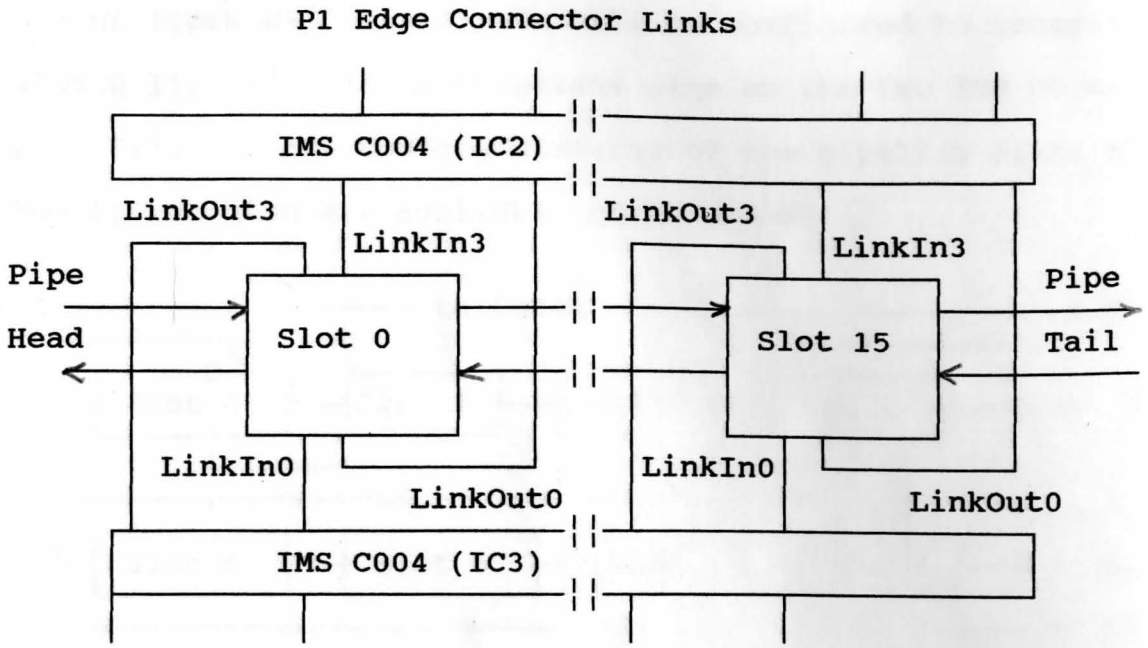


Fig. III.14 IMS C004 to slot connections

By hardwiring two of the edge connector links together off the board, any of the slot link 0s can be routed to another slot link 0, via the two connected edge links. Slot 0 link 0 (shown to be connected directly to the appropriate C004s in the diagram) is actually connected to edge connector P2, along with the respective pins from the IMS C004s. A link jumper connector which is supplied with the board is used to make the connection between the slot 0 link 0 and the IMS C004s. Slot 0 link 0 is taken to P2 in order to provide two links (links 0 and 1) which are directly connected to module 0 on an edge connector. The IMS C004s can be conveniently bypassed in this way, should the application demand it.

Similarly slot 0 link 3 is connected to pins on

jumper block K1. Usually K1 will be configured to connect slot 0 link 3 to the appropriate pins on the two IMS C004s. Fig. III.15 shows the organization of the pipeline links and the links which are available on P2 and K1.

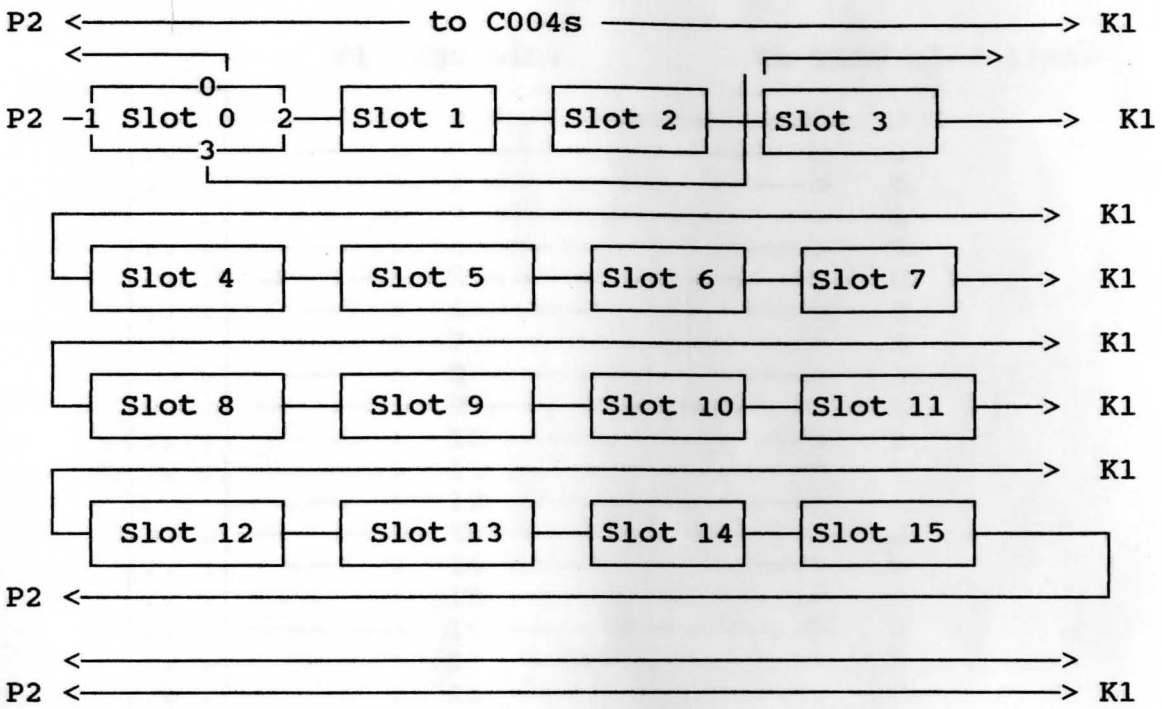


Fig. III.15 Pipeline links and links on P2 and K1

P1 Links

Connector P1 has three rows of 32 pins. All the pins in column "a" are connected to ground. All the pins in column "b" are link *inputs* and all the pins in column "c" are link *outputs*. At each of the 32 positions along P1, the three pins from rows a, b and c together carry one link. These signals may be connected to devices with link ports in any way the user desires, especially in a stand-alone application for the Eurocard. Fig. III.16 shows the P1 pin

assignment.

P2 links

When the IMS B012 is used in an INMOS ITEM card cage the P2 connections are easy since a built-in back-to-back

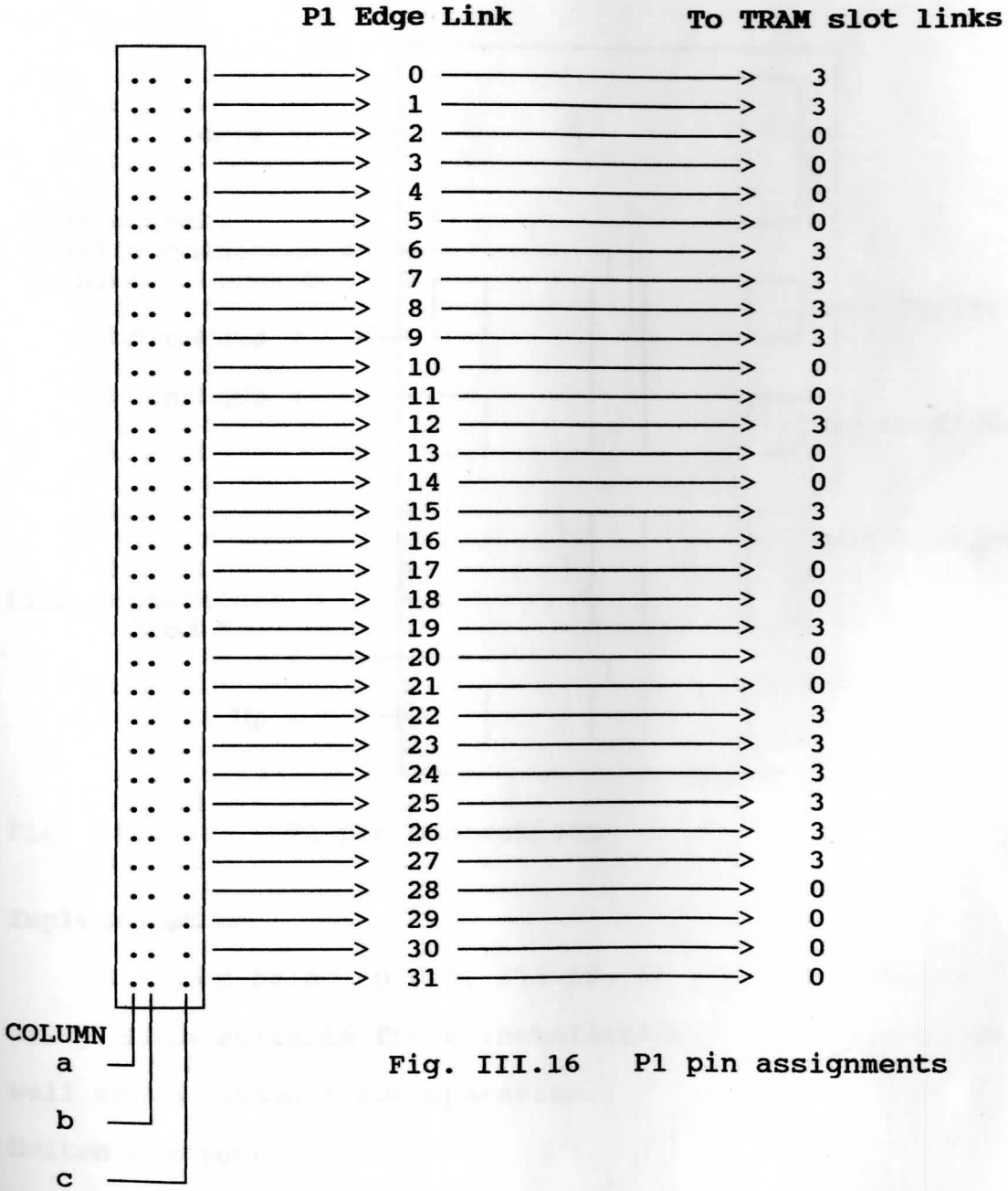


Fig. III.16 P1 pin assignments

connector is provided which allows link and reset cables to be connected to P2. However, for stand-alone applications or in a card cage other than ITEM, the back-to-back connector supplied may be useful. The P2 connections are schematically represented in Fig. III.17.

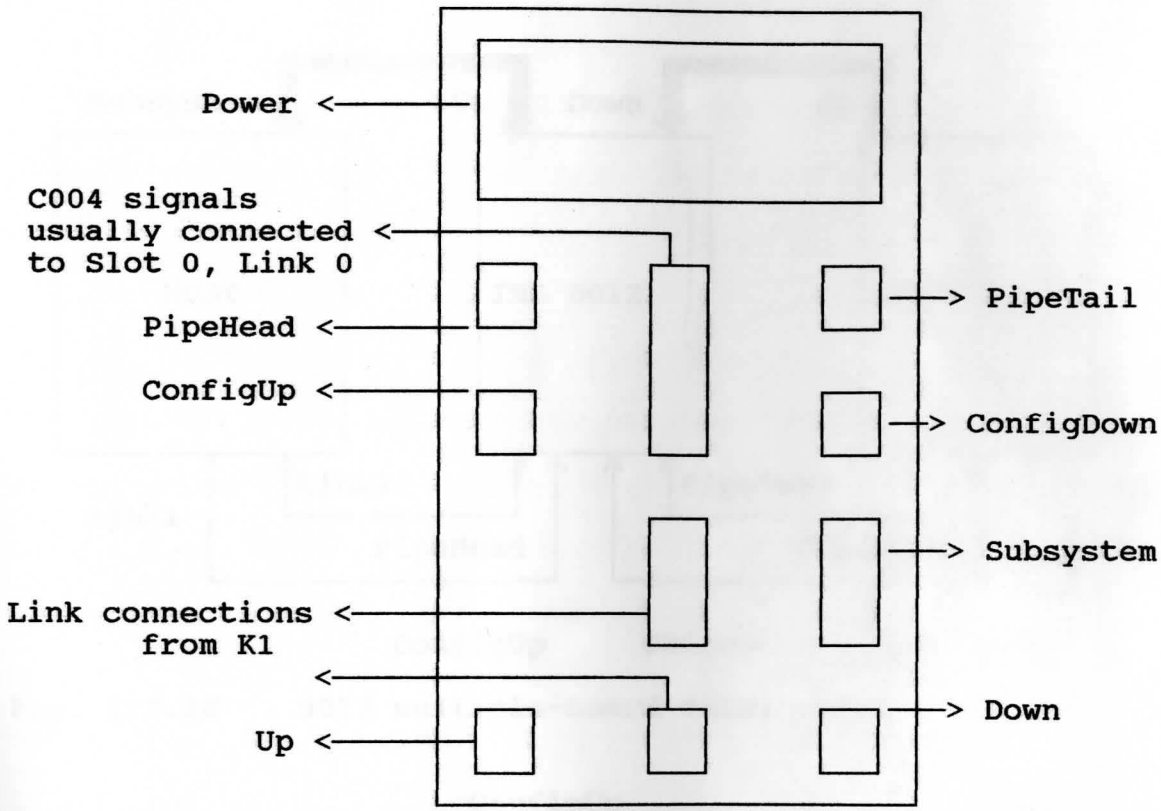


Fig. III.17 P2 pin connections

Implementation

Given below in Fig. III.18, is a multiple-board daisy chain suitable for a installation in a card cage as well as for stand-alone operation.

Switch Configuration

The IMS C004 devices are controlled by an IMS T212

16-bit transputer. The IMS T212 has four links. Links 0 and 3 are connected to the two IMS C004s (link 0 to IC2 and link 3 to IC3). Link 1 is available on edge connector P2 and is called *ConfigUp*. Link 2 is available on P2 and is called *ConfigDown*. The organization of these links is shown in Fig. III.19.

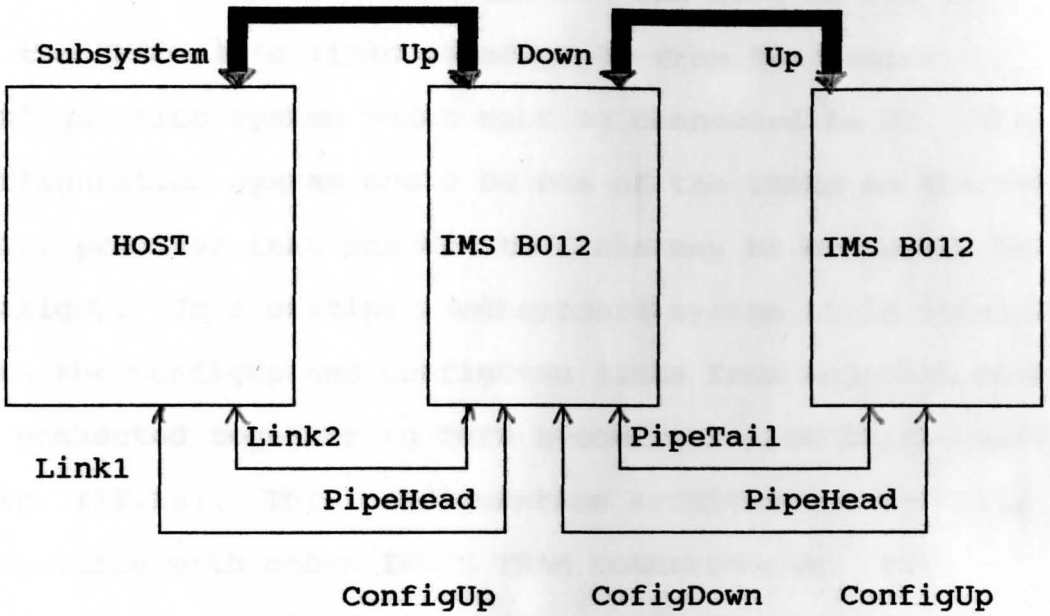


Fig. III.18 B012 multiple-board daisy chain

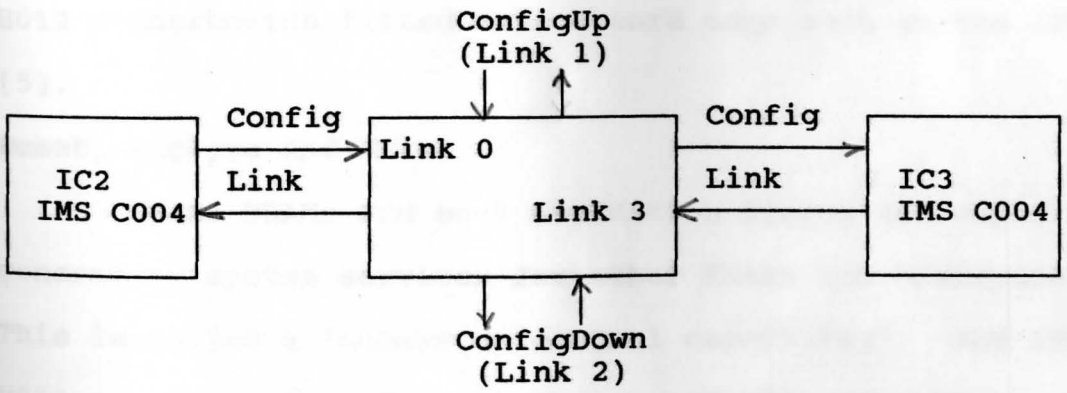


Fig. III.19 B012 config links organization

The switch connections are made according to information sent to the IMS C004 down its **ConfigLink** (Fig. III.18). The two IMS C004s on the IMS B012 allow 64 link connections to be made under software control using the module motherboard software (MMS). Further details of the switch connections can be obtained from (B012 user guide/reference manual).

Configuration data for the IMS C004 is fed into one of the IMS T212's links (ConfigUp) from the master configuration system which must be connected to P2. The configuration system could be one of the TRAMs on the IMS B012, provided that one of its links may be connected to ConfigUp. In a multiple motherboard system it is intended that the ConfigUp and ConfigDowm links from adjacent boards be connected together to form a configuration daisy-chain (Fig. III.19). This configuration architecture is fully compatible with other INMOS TRAM motherboards. For instance, an IMS B008 fitted to an IBM PC/XT or PC/AT or compatible, may be part of a system containing multiple IMS B012 motherboards fitted into a card cage such as the ITEM [5].

Reset, Analyze and Error

Some TRAMs and most evaluation boards are capable of generating system services for other TRAMs and transputers. This is called a 'subsystem control capability'. The IMS B012 can be connected to another board with subsystem control and can also accommodate one TRAM with subsystem

control. Furthermore, the IMS B012 can generate subsystem control signals for other boards.

The Reset, Analyze and Error pins of TRAMs (and transputers) referred to collectively as 'system services' are available for slot 0 only. In order to use these pins it is necessary to have a module with subsystem capability installed in slot 0. The system service signals for slot 0 are buffered and output on edge connector P2 as the 'Down' pins. This allows system services for multiple boards to be daisy-chained, the 'Down' of one board being connected to the 'Up' of the next (Fig. III.18). Fig. III.20 below shows the complete organization of the system services.

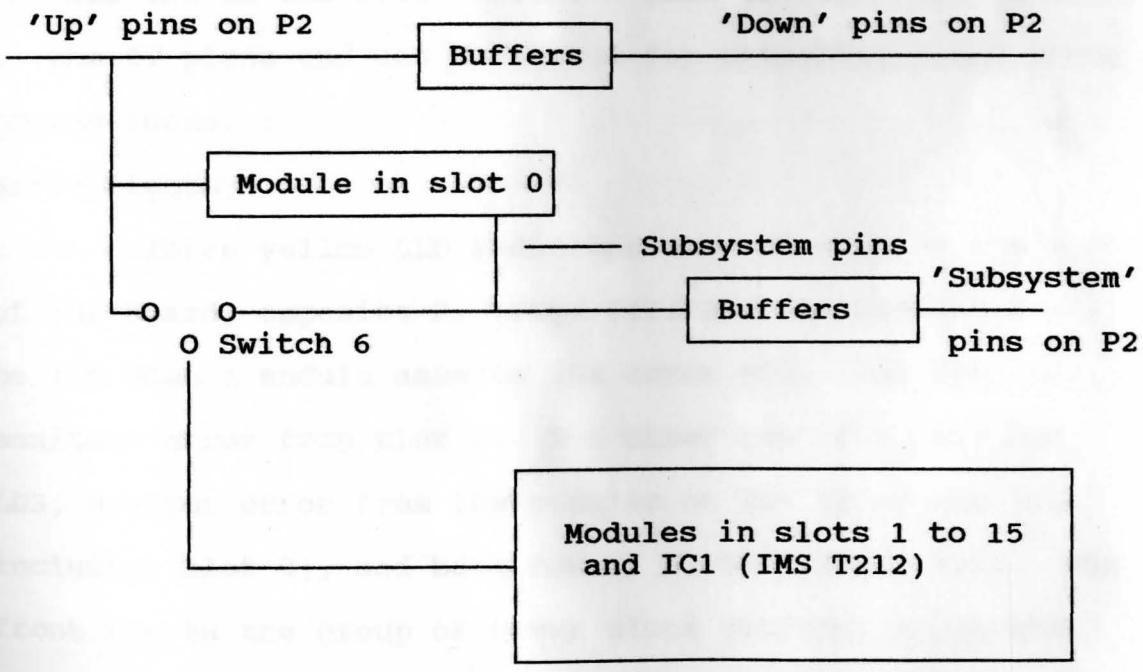


Fig. III.20 B012 System services organization

Power Connections

A four-pin power connector (designated P3) is mounted near the front edge of the board as shown in Fig. III.12 (page 63). P3 is wired to 0V, +5V and via a wide PCB track to 2 pins on P2. This connector is the kind used on most floppy disk drives and when the appropriate pins on P2 are wired to 12V, P3 may be used to power disk drives or similar equipment. Users may take other power signals to P3, such as ECL power supplies. These power pins can carry up to 3A of current and pin 1 can have up to 50V with respect to GND.

There is a pin post fitted in one corner of the board (marked GND on the silk screen). This is connected directly to the 0V plane and can be useful for attaching scope probe ground leads.

Error lights

Three yellow LED indicators are mounted on the edge of the board, opposite P1 (Fig. III.12). An indicator will be lit when a module asserts its error pin. One LED, LD1, monitors error from slot 0. The other two LEDs, LD2 and LD3, monitor error from the modules on the front row (not including slot 0), and back row of slots respectively. The front row is the group of seven slots situated along the front panel side of the board (not including slot 0). The back row is the group of eight slots situated along the opposite edge of the board (Fig. III.12).

Cables

INMOS has developed a standard cable set for evaluation boards. The connectors on all INMOS boards and modules are designs to be compatible with these cables. The IMS B012 is provided with a cable set which include many short link cables (which can be used to link edge links to each other), some standard and long link cables (which can be used to connect multiple cables together), a power cable, some system services cables (reset, analyze and error) and two DIN 41612 connectors which when plugged into P1 and P2 allow cables to be connected to the board.

DIL Switch

Each of the six switches which make up SW1 controls one signal on the board. When a switch is *on* the signal is *low* and when the switch is *off* the signal is *high*. Further information on the switch settings can be obtained from [5].

Eurocard (B012) Stand-Alone Implementation

The B012 is designed not only for installation in the ITEM [6] but also for stand-alone operation. The various connections and requirements for stand-alone operation are described below.

Power connections

Power is supplied to the board via the P3 connector (Fig. III.12) using an INMOS standard power cable.

Link connections

Links 0 and 3 of the various slots are available

from the P1 edge link connector (Fig. III.16). Links 1 and 2 of each slot are pipelined (Fig. III.13) and link 1 (*PipeHead*) and link 2 (*PipeTail*) of slots 0 and 15 respectively are available on the edge link connector P2 (Fig. III.17). Further the P3 links (*ConfigUp*, *ConfigDown*, *System Services* etc) can be accessed by connecting a back-to-back connector (Fig. III.17, page 68). The purpose of this connector is to allow multiple small leads (link cables and reset cables) to be plugged and unplugged at the same time. In addition the back-to-back connector has keying pins either removed or sleeved to make it difficult to orientate standard INMOS link and reset cables incorrectly.

III.6 Inmos Transputer Evaluation Module (ITEM)

The ITEM is a modular cabinet that has been designed to accommodate up to 10 INMOS double extended Eurocard transputer boards. For example, the B211 will accept boards such as the IMS B012 (double extended Eurocard) used for the present project. The ITEM provides a simple means of connecting transputer boards together with the necessary power and cooling requirements to provide potential supercomputing power, The ITEM is consequently suitable as an evaluation vehicle for large transputer systems or as a powerful embedded accelerator accessible from a host.

The ITEM is designed to be upgradeable to meet the user's changing requirements as a project evolves. Further

evaluation boards can be easily added to give additional functionality such as disk storage or graphics. If greater computing power is required, the multiple ITEMs can be linked and stacked to build even larger transputer arrays. An ITEM complete with 10 IMS B012 motherboards has a maximum processing power of 3200 MIPs/360 MFLOPs sustainable power, when each one is loaded with 16 T800-G20 Transputer Modules or TRAMs. The connector panel at the rear of the ITEM includes:

(1) Four BNC connectors for linking the ITEM to color monitors.

(2) Two 25-way D connectors for RS232 connection terminals, computers, and peripherals.

(3) Two 37-way D connectors, each of which can carry 12 transputer links, and three system service ports. A cable is supplied with the ITEM to connect to these D connectors.

Further details of ITEM appear in [6].

CHAPTER IV

DESCRIPTION OF THE PROCESS CONTROL SYSTEM

The block diagram of the system is shown in Fig. IV.1.

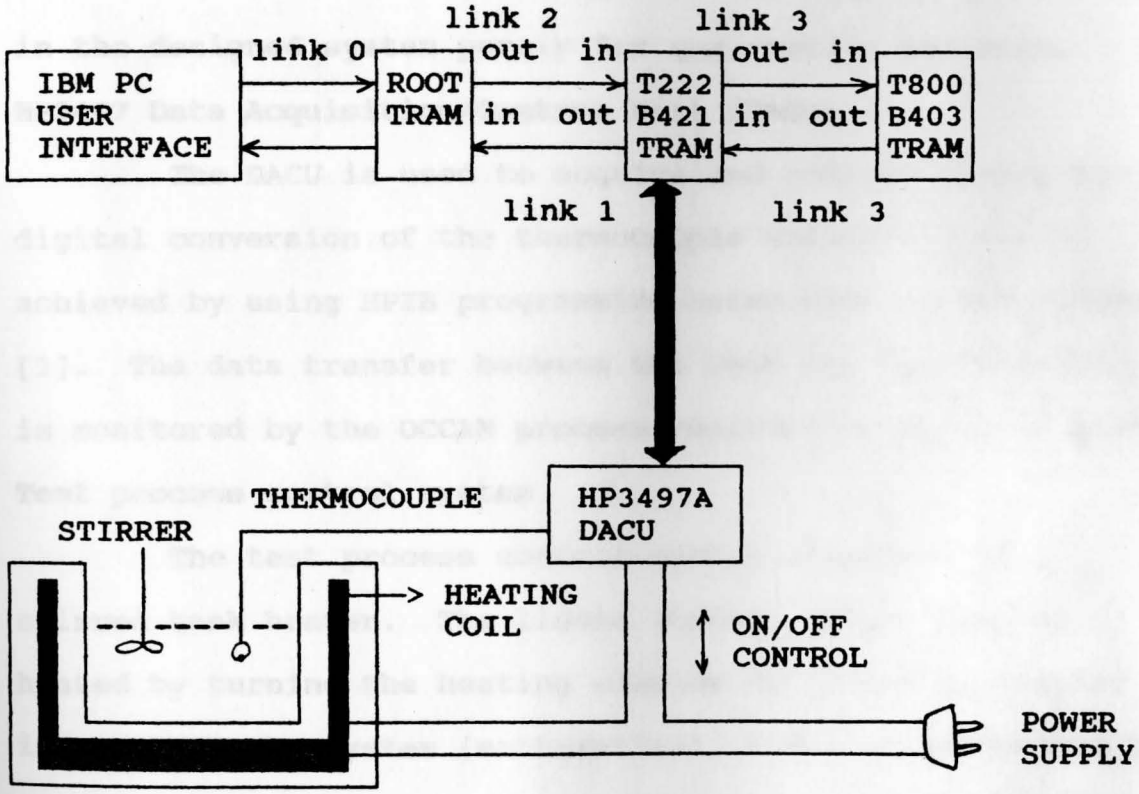


Fig. IV.1 System block diagram

The various blocks in the diagram are described below.

IBM PC/AT Clone

The PC is used essentially for user I/O. It also holds the OCCAM Toolset [10] and the IMS B008 TRAM

motherboard required for compiling and downloading the OCCAM software on to the transputer network. The IMS B008 also holds the IMS B403 (root) TRAM.

ITEM

The ITEM rack is used essentially to hold the Eurocard which in turn holds the GPIB (T222) TRAM. Note that both the TRAMs used in the system can be mounted on the IMS B008 motherboard in the IBM PC. The ITEM is implemented in the designed system purely for qualitative analysis.

HP3497 Data Acquisition/Control Unit (DACU)

The DACU is used to acquire and achieve analog to digital conversion of the thermocouple voltage. This is achieved by using HPIB programming using HPIB command codes [3]. The data transfer between the DACU and the GPIB TRAM is monitored by the OCCAM process residing on the GPIB TRAM.

Test process control system

The test process control system comprises of a stirred tank heater. The liquid (water) in the tank is heated by turning the heating element **ON** or **OFF** at regular intervals. The system (mathematical) model is determined by conducting dynamic tests and the necessary control equations are formulated. A proportional control algorithm is implemented to control the temperature of the water in the tank.

IV.1 Test system description

The controlled variable in the test control system

of the stirred tank heater is the temperature of the liquid in the tank. The automatic control system is designed to manipulate the heating element to keep the water temperature at its desired value or set point in spite of the various disturbances. Flow rate of the liquid in the tank is regulated by adjusting the input and output pumps.

The process control is achieved through a proportional control algorithm. The *controlled quantity*, i.e., the **temperature** of the heating element, is effected by the *manipulated variable* which is the **heat** supplied to the liquid. *Disturbance* (characterised by the deviation in the desired temperature setting owing to the steady flow of water) enters the process and tends to drive the controlled quantity away from its set-point condition. The control algorithm then maintains the set-point of the *controlled quantity* (temperature) by adjusting the *manipulated variable* (heat) and hence reduces the effect of the *disturbance*.

The feedback controller determines changes needed in the heat to compensate for the disturbance that upset the process or for changes in set point. The control algorithm designed for the present system implements *proportional control* in which the controller output is algebraically proportional to the error input signal to the controller.

IV.2 Process Characterization

The typical block diagram of a feedback control loop is shown below in Fig. IV.2. The various signals and

transfer functions (Laplace transforms) are described below.

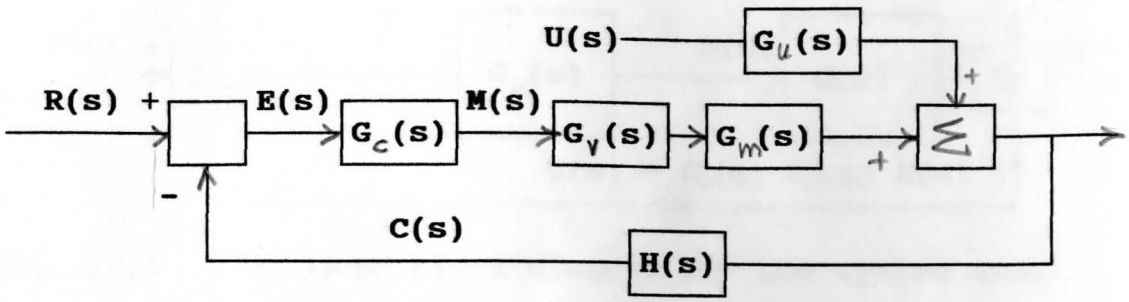


Fig. IV.2 A typical feedback control system

M(s) - the controller-output signal

C(s) - the transmitter-output signal

E(s) - the error signal

U(s) - the disturbance signal

$G_c(s)$ - the controller transfer function

$G_v(s)$ - the transfer function of the final control element

$G_m(s)$ - the process transfer function between the controlled and the manipulated variable

$G_u(s)$ - the process transfer function between the controlled variable and the disturbance

H(s) - the transfer function of the sensor-transmitter

Using simple block diagram algebra manipulations and for the system model under consideration the block diagram of Fig. IV.2 is further reduced as shown below. In this

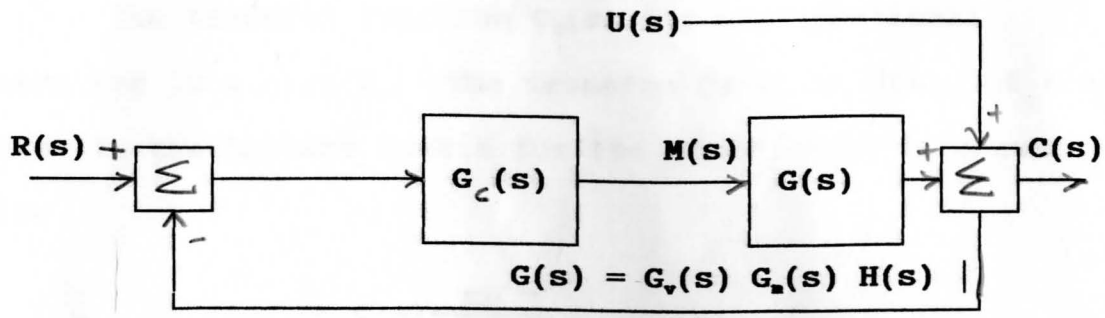


Fig. IV.3 Reduced block diagram for the system model

diagram there are only two blocks in the control loop, one for the controller and the other for the rest of the components of the loop. The advantage of this simplified representation is that it highlights the two signals in the loop that can be usually observed and recorded: the controller output **M(s)** (the power output control to the heating element) and the transmitter signal **C(s)** (the thermocouple voltage). Therefore, the lumping of the transfer functions of the control valve, the process, and the sensor-transmitter into a single block is not just a convenience, but a practical necessity. This combination of the transfer functions is represented by **G(s)**. This combined transfer function is approximated by low-order models for the purpose of characterizing the dynamic response of the process. Thus the characterized "process" includes the dynamic behavior of the control valve and the sensor/transmitter. The stirred tank heater system under study is modeled as a **First-order Plus Dead Time (FOPDT)** system.

The transfer function $G_c(s)$ for a proportional controller is a gain K_c. The transfer function G(s), of the system in the Laplace domain for the **FOPDT** model is shown below

$$C(s) = \frac{Ke^{-t_o s}}{\tau s+1} \quad \ldots\ldots (1)$$

and the controller gain

$$K_c = (1/K) * (t_o/\tau) \quad \ldots\ldots (2)$$

This model characterizes the process by three parameters: system gain K, dead time or transportation lag t_o, and system time constant τ. The problem of determining the parameters for this loop is solved by performing a dynamic flow test as outlined below.

VI.3 Dynamic test and system model

The controller is placed in "manual" mode (i.e. the loop is opened) and the level of water in the tank is kept steady. Heat is applied to the water at a constant rate and the temparature values are acquired until a steady state is reached. A graph with temparature vs time, also known as a *process reaction curve* (Appendix B), is plotted and is interpreted to obtain the parameters required for formulating the system equations.

The procedure for the actual test for the designed system is

(1) Water is admitted into the tank heater and the

output pump is turned on.

(2) The input flow (coming from the tap source) and the output flow (controlled by the pump) are adjusted until the level in the tank is maintained constant.

(3) The data acquisition program **data_acq.bas** is run on the PC in the **LabWindows** [15] environment and simultaneously power is supplied to the heating element at the desired percentage (60%), by adjusting the heater dial manually.

(4) Temparature values are acquired in 5 second intervals until the system has reached a steady state (constant temparature) and the graph of temparature Vs time is obtained.

(5) From the acquired data the required **FOPDT** model parameters for the proportional control algorithm are obtained.

In the absence of the disturbances, and for the conditions of the test, the block diagram of Fig. IV.3 is redrawn as in Fig. IV.4.

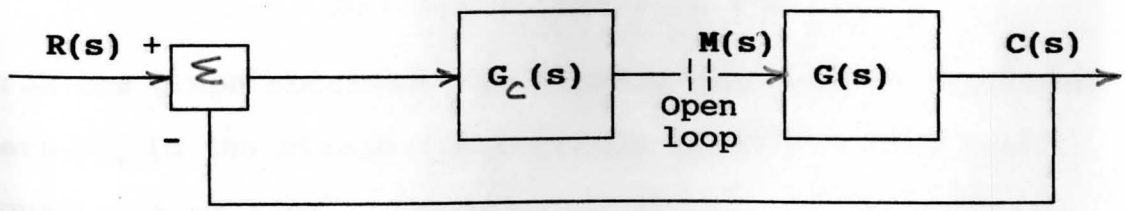


Fig. IV.4 Block diagram for the dynamic test

The response of the transmitter output signal is given by

$$C(s) = G(s)\ M(s)$$

For a step change in controller output of magnitude Δm and a FOPDT model, we have

$$C(s) = \frac{Ke^{-t_0 s}}{\tau s+1} \cdot \frac{\Delta m}{s} \quad \ldots\ldots (3)$$

Expanding this expression by partial fractions, we obtain

$$C(s) = K.\Delta m.e^{-t_0 s}.\left[\frac{1}{s} - \frac{\tau}{\tau s+1}\right] \quad \ldots\ldots (4)$$

Thus Inverting with the help of Laplace transforms and applying the real translation theorem of Laplace transforms [16], we get

$$C(s) = K.\Delta m.U(t-t_0).e^{\frac{-(t-t_0)}{\tau}} \quad \ldots\ldots (5)$$

Where the unit step function $U(t - t_0)$ indicates that

$$\Delta c(t) = 0 \text{ for } t \leq t_0 \quad \ldots\ldots (6)$$

The term Δc is the change of the transmitter output from its initial value:

$$\Delta c(t) = c(t) - c(0) \quad \ldots\ldots (7)$$

From the graph obtained from running the dynamic test, the term ΔC_s is the steady state change in c(t). Thus from equation 5 we find

$$\Delta C(s) = \lim \Delta C(t) = K.\Delta m \quad \ldots\ldots (8)$$

From the above result and knowing that the model response must match the process reaction curve at steady state, the steady state gain K is calculated, which is one of the model parameters:

$$K = \frac{\Delta C_s}{\Delta m} \quad \ldots\ldots (9)$$

The dead time or *transportation lag* t_o and the time constant τ are determined by the following method proposed by Dr.Cecil L. Smith [14].

The values are obtained from the *process reaction curve* by selecting two points in the region of high rate of change. The two points recommended are

$$\{t_o + (1/3) * \tau\} \quad \ldots\ldots (10)$$

and

$$\{t_o + \tau\} \quad \ldots\ldots (11)$$

To locate these points we make use of equation 5.

$$C(t_o + \tau) = K.\ m.[1-e^{(-1)}] = 0.632\ \Delta C_s \quad \ldots\ldots (12)$$

$$C\ (t_o + (1/3)*\ \ ) = K.\ m.[1-e^{-(1/3)}] = 0.238\ \Delta C_s \quad \ldots\ldots (13)$$

These two points are labeled t_2 and t_1, respectively, in the *process reaction curve* appearing (Appendix B). The values of t_o and τ are then obtained by the simple solution of the following set of equations:

$$t_o + \tau = t_2 \quad \ldots\ldots (14)$$

$$t_o + (1/3)*\tau = t_1 \quad \ldots\ldots (15)$$

which reduces to

$= (3/2)* (t_2 - t_1)$ ..... (16)

$t_o = t_2 - \tau$ ..... (17)

where t_1 is the time at which

$\Delta C = 0.283 \Delta C_s$ ..... (18)

and t_2 is the time at which

$\Delta C = 0.632 \Delta C_s$ ..... (19)

The actual values obtained from the *process reaction curve* are listed below.

The minimum and maximum values of c(t) are,

$C(t_{min}) = 17.24\ ^{\circ}C$ ..... (20)

$C(t_{max}) = 21.32\ ^{\circ}C$ ..... (21)

The steady state value of c(t)

$\Delta C_s = 21.32 - 17.24 = 4.08\ ^{\circ}C$ ..... (22)

To calculate t_1 and t_2,

$\Delta C_{s1} = 0.283\ \ C_s = 1.16\ ^{\circ}C$ ..... (23)

$\Delta C_{s2} = 0.632\ \ C_s = 2.58\ ^{\circ}C$ ..... (24)

$\Delta C_1 = \ C_{s1} + C(t_{min}) = 1.16 + 17.24 = 18.4\ \ C$ ..... (25)

$\Delta C_2 = \ C_{s2} + C(t_{min}) = 2.58 + 17.24 = 19.82\ \ C$ .....(26)

From the *process reaction curve*

t_1 (at ΔC_{s1}) = 23.0 * 5 = 115 seconds ..... (27)

t_2 (at ΔC_{s2}) = 40.0 * 5 = 200 seconds ..... (28)

Hence using equation 16

τ = 127.5 seconds ..... (29)

and dead time from equation 17

t_o = 72.5 seconds ..... (30)

Also the process gain K is evaluated using equation 9

$$K = (4.08/0.6) = 6.8 \quad \ldots\ldots \quad (31)$$

Note that m=0.6 (or 60%) is the value at which the heater dial is set manually for the duration of the dynamic test. Thus the controller gain from equation 2 is

$$K_c = (1/6.8)*(127.5/72.5) = 0.26 \quad \ldots\ldots \quad (32)$$

IV.4 Proportional controller implementation aspects

The proportional controller is the simplest type of controller, with the exception of on-off controller [14]. The equation which describes its operation is the following:

$$m(t) = m_b + K_c * (e(t)) \quad \ldots\ldots \quad (33)$$

where

m(t) = output from the controller

r(t) = set point (or reference temperature)

c(t) = controlled variable (temperature of the liquid in the tank)

e(t) = error signal. This is the difference between the reference temperature and the actual temperature of the liquid in the tank i.e.,

$$e(t) = r(t) - c(t) \quad \ldots\ldots \quad (34)$$

K_c = controller gain (calculated to be .26 or 26% from equation 2.

m_b = bias value of the controller.

Equations 33 and 34 show that the output of controller is proportional to the error between the

reference temperature and actual temperature read from the thermocouple sensor. The proportionality is given by the controller gain K_c. This gain, or controller sensitivity, determines how much the output from the controller changes for a given change in error. In reality, for the system under study, this proportional action is achieved in a slightly different manner as outlined below.

To achieve proportional control action on the system the following scheme is implemented on the system.

(1) Temperature is acquired from the thermocouple in a free running mode.

(2) The error value is evaluated using equation 34.

(3) The controller output m(t) is evaluated using equation 33.

(4) A control interval time (Interval) is chosen such that

Interval = [r(t) * K_c] seconds ..... (35)

(5) The heater is turned **ON** for a time period of

ON.time = [e(t) * K_c] seconds ..... (36)

and **OFF** for a time period of

OFF.time = [Interval - **ON**.time] seconds ..... (37)

The above scheme is used since the energy supplied to the heating element installed in the tank is not controllable continuously using the DACU i.e., the heater can only be turned **ON** or **OFF**. Hence proportional control is achieved by having the heater-**ON** and heater-**OFF** time periods

proportional to the error signal as governed by the proportional gain value K_c, within the control interval time.

The procedure is illustrated in the following example.

Let

$r(t) = 50.0\ ^{o}C$

$c(t) = 25.0\ ^{o}C$

then for a proportional gain value of

$K_c = 0.26$

the control interval is obtained from equation 35 as,

Interval = 13 seconds

and from equation 34

$e(t) = 25.0\ ^{o}C$

thus the **ON** time period

ON.time = 6.5 seconds

and the **OFF** time period

OFF.time = 6.5 seconds

The proportional control algorithm is implemented on the transputer network using OCCAM2. Two separate processes are written to achieve the proportional control and they are mapped on to the transputer network. The software is listed and the implementation on the transputer network is illustrated in the following sections.

IV.5 Software outline

Two OCCAM processes are written to separately handle the data and command input/output (I/O) and proportional control action as described below.

(1) Process **PROC_IO** (I/O process): This process acquires the data (thermocouple voltage) from the DACU by sending the appropriate HPIB command codes. The data acquired, which is in ASCII format, is passed on to the control process **PROC_CON** (control process). The acquire process is mapped onto the GPIB TRAM which interacts with the DACU via GPIB programming.

(2) Process **PROC_CON** (control process): This process converts the ASCII data to a 32-bit floating point number. The thermocouple voltage is converted to a temparature value and the necessary proportional control operation is performed. The ON/OFF control message is passed on back to the acquire process which in turn relays the appropriate command to the DACU. Eventually the DACU turns the tank heater **ON** or **OFF** for the required time period as determined by the software.

The descriptive listing of the two processes appears in appendix A.

IV.6 Network configuration and software implementation

The system is set up as outlined for performing the dynamic test in section IV.3 above. As outlined in section III.4 the necessary *Hardwire* and *Softwire* are created for the IMS B008 motherboard. The contents of the *Hardwire* file are listed in section III.4. For the present network both the IMS B403 (T800 based TRAM) and the IMS B421 (T222 based GPIB TRAM) are installed on the B008 motherboard. Further the root TRAM on the B008 is not used as part of the network and hence the B403 TRAM is made to interact with the host (IBM PC) by skipping the link 2 of the root TRAM. This network is depicted in Fig. IV.5.

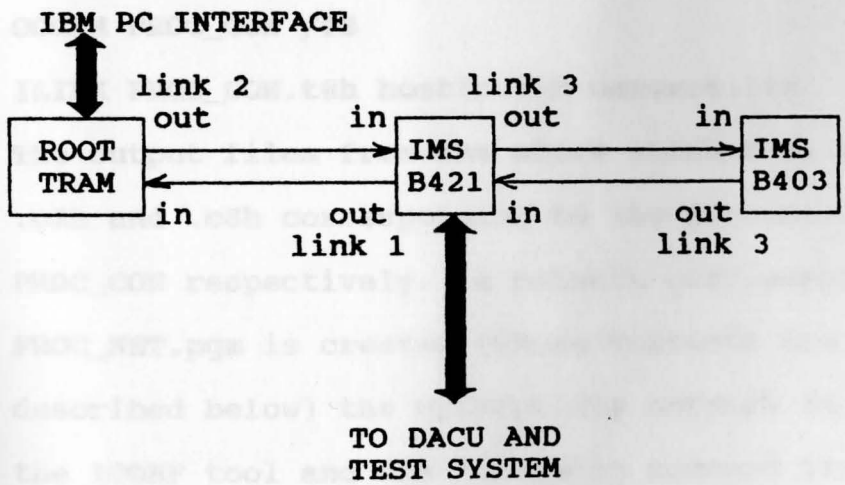


Fig IV.5 Transputer network schematic

The two OCCAM process are compiled, linked and downloaded onto the transputer network in the IBM PC (DOS environment) as outlined below.

The process **PROC_IO** is checked for syntax errors using the following command line.

ICHECK PROC_IO /T2 /B

The process is then compiled and linked with the appropriate library modules using the following command lines.

OCCAM PROC_IO /T2

and

ILINK PROC_IO.t2h hostio.lib convert.lib f001.lib f001io.lib f001llev.lib

similarly the process **PROC_CON** is checked, compiled and linked using the following command lines.

ICHECK /T8 /B

OCCAM PROC_CON /T8

ILINK PROC_CON.t8h hostio.lib convert.lib

The output files from the above **ILINK** tool have extensions **.c2h** and **.c8h** corresponding to the processes **PROC_ACQ** and **PROC_CON** respectively. A network configuration file **PROC_NET.pgm** is created (whose contents are listed and described below) the appropriate network is configured using the **ICONF** tool and the following command line.

ICONF PROC_NET

As outlined in section III.4 the necessary *Hardwire* and *Softwire* files are created for the IMS B008 motherboard. Note for the present network both the IMS B403 (T800 based TRAM) and the IMS B421 (T222 based GPIB TRAM) are installed

on the B008 motherboard. Further the root TRAM on the B008 is not used as part of the network and hence the B403 TRAM is made to interact with the host (IBM PC) by skipping the link 2 of the root TRAM and placing the host link on link 1 of the B403 TRAM. This network is schematically represented below.

CHAPTER V

SUMMARY

V.1 Findings

The problem of designing and implementing a transputer network for a real-time application was successfully tackled and the overall system performed as expected. The performance of the individual components of the system is discussed.

PC TRAM motherboard (B008)

The IBM PC served as a good platform for installing and evaluating both the hardware and software required to support the B008. The B008 interfaced well with the PC bus and all its features were well exploited. The module motherboard software required for the set up and evaluation of transputer networks performed well. The S708 [4] DOS device driver supported the B008 without any hitch.

OCCAM2 toolset

All the features of the OCCAM2 toolset performed as expected. The software for the system performed well without any major debugging problems.

Eurocard (B012)

The Eurocard B012 performed well in the ITEM [6] but its performance was not evaluated for stand-alone operation.

HP3497A Data Acquisition/Control Unit (DACU)

The DACU responded well to the programming from the GPIB transputer. It played an important role in converting analog thermocouple voltage to digital format and subsequent transfer of data to the PC for analysis and control.

Test system

The test system responded well to the control action and the thermocouple voltage was acquired without any problem. The input and output pumps performed well in maintaining the level of water constant in the tank.

Transputer network

The software and hardware aspects of networking the transputers were dealt with successfully. The required hardware connections were established with the standard cables supplied by INMOS. The transputer link connections were adequately established using the MMS software supplied by INMOS.

Control Algorithm

The proportional control algorithm was modified to accomodate the hardware and software constraints imposed by the test system. The modified algorithm was successfully implemented over the network and the expected results were obtained.

V.2 Conclusions

The transputer proved to be an effective computing

tool in implementing the proportional control algorithm for the designed test system. The OCCAM programming language provided not only the simplicity and structure of a high-level language but also the flexibility of an assembly language [1]. In designing and realizing the entire system, transputer networking (hardware) and GPIB programming (using HP3497A command codes [3]) proved to be more time-consuming compared to the control algorithm design and OCCAM coding. This happened since the transputer hardware was installed for the first time. Students interested in further study in this area are recommended not to delve into the hardware details (except for removal or addition of transputers to the network) but to concentrate on the relevant software, i.e, OCCAM (or PARALLEL C), for programming and the MMS for linking the custom made networks.

The documentation supplied by INMOS regarding their products is found to be more descriptive than functional. It is recommended that only the sections describing the actual implementation aspects be studued to save time.

V.3 Recommendations

The proportional control algorithm implemented for the present study has been modified to suit the available hardware. It is recommended that an analog amplifier be designed and interfaced to the heating coil so that the power supplied to the heating element can be controlled

continuously.

Also, the data acquisition and control operations could be performed by designing an interface compatible with GPIB programming. This would not only eliminate the need for the DACU but also speed up the data transfer rate upto 20 Mbits/s, which is the peak rate achievable by the transputer links. Further the network could be expanded to monitor several other parameters (i.e., pressure, frequency etc.) with ease with such an interface as outlined in the figure V.1.

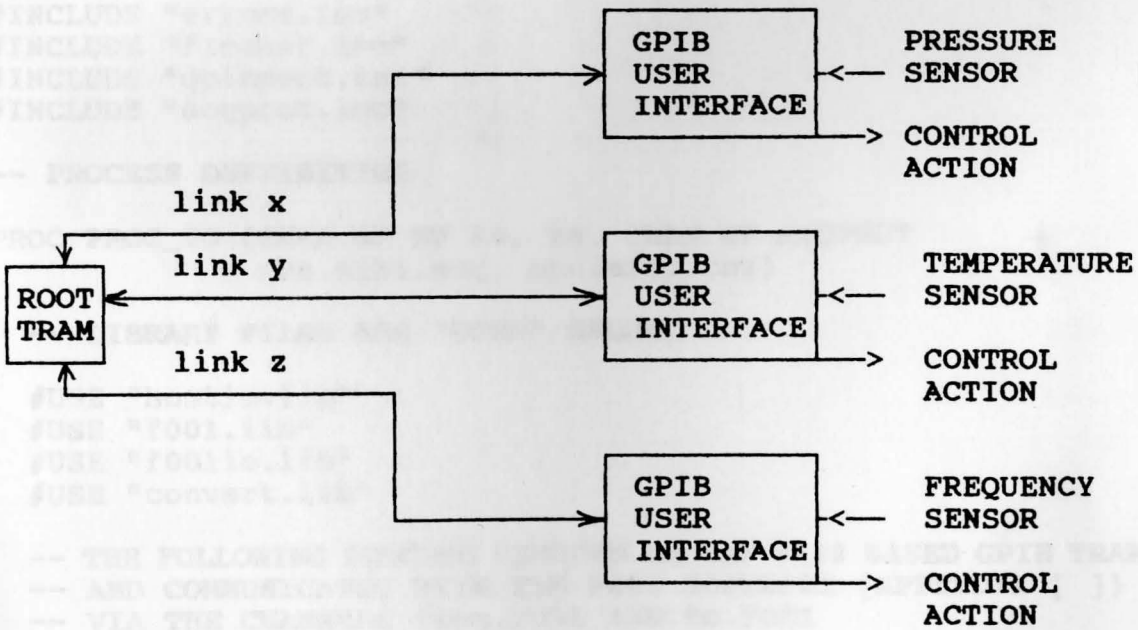


Fig. V.1 A typical transputer based multi-sensor network

APPENDIX A

SOFTWARE LISTING

PROCESS PROC_IO.OCC

```
-- PROCESS PROC_IO.OCC ACQUIRES THE THERMOCOUPLE VOLTAGE (IN
-- ASCII FORMAT), INTERACTS WITH THE PROCESS PROC_CON.OCC AND
-- SENDS THE 'ON' OR 'OFF' COMMANDS TO THE HEATER VIA THE
-- DACU.
--------------------------------------------------------------
-- THE VARIOUS PROTOCOLS ARE DEFINED IN THE 'INCLUDE' FILES
-- LISTED BELOW.

#INCLUDE "hostio.inc"
#INCLUDE "errors.inc"
#INCLUDE "f1const.inc"
#INCLUDE "gpibprot.inc"
#INCLUDE "acqprot.inc"

-- PROCESS DEFFINITION

PROC PROC_IO (CHAN OF SP fs, ts, CHAN OF ACQPROT
              chn.aski.acq, chn.aski.cnv)

  -- LIBRARY FILES ARE 'USED' BELOW.

  #USE "hostio.lib"
  #USE "f001.lib"
  #USE "f001io.lib"
  #USE "convert.lib"

  -- THE FOLLOWING PROCESS RESIDES ON THE T222 BASED GPIB TRAM
  -- AND COMMUNICATES WITH THE F001 SOFTWARE (APPENDIX [ ])
  -- VIA THE CHANNELS from.F001 AND to.F001

  PROC hpibt2 (CHAN OF F001PROT from.F001, to.F001)

  -- DECLARATION OF VARIABLES

  INT16 source, pri.address, mode, drivers, error, count.mssg:
  INT16 tx.period, term, countm, count.control, tk.address:
  INT len1, t, i, iterate:
  BYTE result:
  [1]INT16 la.list:
  REAL32 check.end, check.two, check.three:
```

```
[12]BYTE data, data.aski.acq, data.aski.conv, avg.volt:
[12]BYTE scrap, temp, er, switch, avg.ref:
[4]BYTE mesage:
[5]BYTE ON.or.OFF, mssg:
[6]BYTE Sread.val:
BOOL error.itn, loop:

-- SET UP ROUTINES FOR THE GPIB TRAM APPEAR BELOW.
-- DETAILS OF THE ROUTINES APPEAR IN THE F001 MANUAL [8].

SEQ

  F001.START.SETUP (from.F001, to.F001, error)

  pri.address := 28 (INT16)
  source := default
  F001.SET.GPIB.ADDRESS (from.F001, to.F001, source,
                         pri.address, error)

  mode := system.controller
  F001.SET.DEVICE.MODE (from.F001, to.F001, source, mode,
                        error)

  drivers := tristate
  F001.SET.BUS.DRIVERS (from.F001, to.F001, source, drivers,
                        error)

  tx.period := 500 (INT16)  -- ms
  F001.SET.TIMEOUT (from.F001, to.F001, tx.period, error)

  term := LF.term
  F001.SET.TX.TERMINATOR (from.F001, to.F001, term, error)

  F001.SET.RX.TERMINATOR (from.F001, to.F001, term, error)

  F001.END.SETUP (from.F001, to.F001, error)

  F001.SEND.IFC (from.F001, to.F001, error)

  F001.SET.REN (from.F001, to.F001, TRUE, error)

  mesage := "AR  "    -- RESET THE ANALOG CHANNELS IN THE
                      -- DACU
  countm := 4 (INT16)
  la.list [0] := 9 (INT16)
  tk.address := 9 (INT16)
  F001.SEND (from.F001, to.F001, la.list, countm, mesage,
             error)

  mesage := "    "    -- CLEAR THE DACU COMMAND BUFFER
  F001.SEND (from.F001, to.F001, la.list, countm, mesage,
             error)
```

```
-- THE REFERENCE INPUT VALUE IS TAKEN IN FROM THE
-- KEYBOARD.

so.write.string (fs, ts, "ENTER THE REFERENCE INPUT
                  REQUIRED  ")
so.read.echo.line (fs, ts, len1, Sread.val, result)
so.write.nl (fs, ts)

chn.aski.acq ! medm; Sread.val

-- NUMBER OF SAMPLES TO READ (AND AVERAGE) IS INPUT
-- BELOW.

so.write.string (fs, ts, "ENTER ITERATION VALUE ")
so.read.echo.int (fs, ts, iterate, error.itn)
so.write.nl (fs, ts)

t := iterate
i := t

loop := TRUE

WHILE loop
  SEQ
    mesage := "AR  "  -- RESET THE ANALOG CHANNELS IN THE
                      -- DACU

    F001.SEND (from.F001, to.F001, la.list, countm,
               mesage, error)
    mesage := "    "
    F001.SEND (from.F001, to.F001, la.list, countm,
               mesage, error)

    mesage := "AI1 "  -- INPUT VOLTAGE VALUE FROM ANALOG
                      -- CHANNEL 1 i.e., THERMOCOUPLE
                      -- PROBE VOLTAGE (T_PROBE) IN
                      -- ASCII FORMAT.

    F001.SEND (from.F001, to.F001, la.list, countm,
               mesage, error)
    F001.RECEIVE (from.F001, to.F001, tk.address, scrap,
                  error)

    WHILE t > 0 (INT)
      SEQ
        check.two := 10.0 (REAL32)

        -- SEND FLAG (10.0) TO PROC_CON TO INDICATE
        -- T_PROBE ACQUISITION (AND AVERAGING).

        chn.aski.acq ! float; check.two
```

```
    F001.SEND (from.F001, to.F001, la.list, countm,
               mesage, error)

    F001.RECEIVE (from.F001, to.F001, tk.address,
                  data, error)

    data.aski.acq := data

    -- SEND T_PROBE TO PROC_CON

    chn.aski.acq ! long; data.aski.acq

    -- RECEIVE THE T_PROBE AVERAGE VOLTAGE

    chn.aski.cnv ? CASE long; data.aski.conv

    avg.volt := data.aski.conv
    t := t - 1 (INT)
so.write.string (fs, ts, "THE AVERAGE PROBE
                  VOLTAGE IS [volts] ")
so.write.string.nl (fs, ts, avg.volt)

mesage := "AR  "
F001.SEND (from.F001, to.F001, la.list, countm,
           mesage, error)

mesage := "    "
F001.SEND (from.F001, to.F001, la.list, countm,
           mesage, error)

mesage := "AI19"  -- INPUT ANALOG CHANNEL 19
                  -- i.e., REFERENCE JUNCTION
                  -- (T_REF) VOLTAGE VALUE.

F001.SEND (from.F001, to.F001, la.list, countm,
           mesage, error)
F001.RECEIVE (from.F001, to.F001, tk.address,
              scrap, error)

SEQ
  WHILE i > 0 (INT)
    SEQ
      check.three := 20.0 (REAL32)

      -- SEND FLAG (20.0) TO PROC_CON TO INDICATE
      -- T_REF ACQUISITION (AND AVERAGING).

      chn.aski.acq ! float; check.three
      la.list [0] := 9 (INT16)
      tk.address := 9 (INT16)

      mesage := "AI19"
```

```
        F001.SEND (from.F001, to.F001, la.list,
                   countm, mesage, error)

        F001.RECEIVE (from.F001, to.F001, tk.address,
                      data, error)

        data.aski.acq := data

        -- SEND THE REFERENCE JUNCTION VOLTAGE
        -- (ASCII) VALUE TO PROC_CON FOR AVERAGING.

        chn.aski.acq ! long; data.aski.acq

        -- RECEIVE AVERAGE T_REF VALUE.

        chn.aski.cnv ? CASE long; data.aski.conv
        avg.ref := data.aski.conv
        i := i - 1 (INT)
t := iterate
i := t
so.write.string (fs, ts, "THE AVERAGE REFERENCE
                  VOLTAGE IS [volts]  ")
so.write.string.nl (fs, ts, avg.ref)
check.end := 30.0 (REAL32)

-- SEND FLAG (30.0) TO PROC_CON TO INDICATE END
-- OF ACQUISITION FROM DACU.

chn.aski.acq ! float; check.end

-- RECEIVE AND DISPLAY THERMOCOUPLE TEMPARATURE
-- VALUE FROM PROC_CON.

chn.aski.cnv ? CASE long; temp
so.write.string (fs, ts, "MEASURED TEMPARATURE IS
                 [DEG. C] ")
so.write.string (fs, ts, temp)
chn.aski.cnv ? CASE long; temp
so.write.string.nl (fs, ts, temp)

-- RECEIVE 'ON.time' VALUE FROM PROC_CON.

chn.aski.cnv ? CASE long; switch
so.write.string (fs, ts, "ON.time")
so.write.string.nl (fs, ts, switch)

mssg := "AR   "
count.mssg := 5 (INT16)
F001.SEND (from.F001, to.F001, la.list, count.mssg,
           mssg, error)
```

```
      mssg := "      "
      F001.SEND (from.F001, to.F001, la.list, count.mssg,
                 mssg, error)

      -- RECEIVE ON.or.OFF MESSAGE AND SEND THE 'ON' AND
      -- 'OFF' COMMANDS TO THE HEATER VIA THE DACU.

      chn.aski.cnv ? CASE md.srt; ON.or.OFF
      so.write.string.nl (fs, ts, ON.or.OFF)
      F001.SEND (from.F001, to.F001, la.list,
                 count.control, ON.or.OFF, error)

      chn.aski.cnv ? CASE md.srt; ON.or.OFF
      so.write.string.nl (fs, ts, ON.or.OFF)
      F001.SEND (from.F001, to.F001, la.list,
                 count.control, ON.or.OFF,
                 error)

      -- RECEIVE AND DISPLAY ERROR VALUE FROM PROC_CON

      chn.aski.cnv ? CASE long; er
      so.write.string.nl (fs, ts, "ERROR   REFERENCE ")
      so.write.string (fs, ts, er)

      mesage := "SI  "  -- SYSTEM INITIALIZE THE DACU.
      F001.SEND (from.F001, to.F001, la.list, countm,
                 mesage, error)

    mssg := "AR   "
    F001.SEND (from.F001, to.F001, la.list, count.mssg,
               mssg, error)

    F001.DEVICE.CLEAR (from.F001, to.F001, TRUE, la.list,
                       error)
    F001.SET.REN (from.F001, to.F001, FALSE, error)

    so.exit (fs, ts, sps.success)
  :
  -- THE F001 (APPENDIX [ ]) AND hpibt2 (ABOVE) PROCESSES ARE
  -- RUN ON THE B421 TRAM IN PARALLEL.

  CHAN OF F001PROT from.F001, to.F001:
  SEQ
    PAR
      F001 (from.F001, to.F001)
      hpibt2 (from.F001, to.F001)
:
```

PROCESS PROC_CON.OCC

```
-- PROCESS PROC_CON.OCC RECEIVES THE THERMOCOUPLE VOLTAGES
-- FROM THE PROCESS PROC_IO.OCC, EVALUATES THE THERMOCOUPLE
-- TEMPARATURE, DETERMINES THE 'ON.time' AND 'OFF.time' AND
-- SENDS THE APPROPRIATE 'ON' OR 'OFF' COMMAND TO THE PROCESS
-- PROC_IO.OCC.

#INCLUDE "acqprot.inc"
PROC PROC.CON(CHAN OF ACQPROT chn.aski.acq, chn.aski.cnv)
  #USE "convert.lib"
  [5]BYTE ON, OFF:
  [6]BYTE Rread.val:
  [12]BYTE aski, aske, asi, ase, askt, er.val, switch.val:
  [12]BYTE time:
  REAL32 read.val, test, size.test:
  REAL32 count.probe, real.data.probe, sum.probe, avg.probe:
  REAL32 count.ref, real.data.ref, sum.ref, avg.ref,
  REAL32 V, V1, TT, TT1a, TT2, TT3, TT3.old, TT4:
  REAL32 step, tac, eror, eror.i, interval, switch.on.point:
  REAL32 switch.off.point:
  INT len.probe, len.ref, len.temp, len.er, debut:
  INT len.switch, ON.time, OFF.time, timenow, len:
  BOOL error, tester, repeat:
  TIMER clock:

  -- CONSTANTS ASSOCIATED WITH THE EVALUATION OF THE
  -- TEMPARATURE FROM THE THERMOCOUPLE VOLTAGE ARE LISTED
  -- BELOW.

  VAL R0 IS 0.75004344E-6(REAL32):
  VAL R1 IS 0.505321995E-4(REAL32):
  VAL R2 IS 0.2348050017E-7(REAL32):
  VAL P0 IS -0.3595568424(REAL32):
  VAL P1 IS 19750.87948(REAL32):
  VAL P2 IS -175116.5425(REAL32):
  VAL P3 IS 18212965.58(REAL32):
  VAL P4 IS -2831128435.0(REAL32):
  VAL P5 IS 271508383300.0(REAL32):
  VAL P6 IS -1.38014121E+13(REAL32):
  VAL P7 IS 3.7924384326E+14(REAL32):
  VAL P8 IS -5.371925517E+15(REAL32):
  VAL P9 IS 3.0840255439E+16(REAL32):
  VAL CON1 IS 100.0(REAL32):

  SEQ
    -- INITIALIZATION OF VARIABLES
```

```
tester := TRUE
count.probe := 0.0 (REAL32)
sum.probe := 0.0 (REAL32)
avg.probe := 0.0 (REAL32)
count.ref := 0.0 (REAL32)
sum.ref := 0.0 (REAL32)
avg.ref := 0.0 (REAL32)

-- HPIB COMMANDS FOR HEATER 'ON' AND 'OFF'
ON := "DC4,0"
OFF := "DO4,0"

-- RECEIVE REFERENCE TEMPARATURE VALUE

chn.aski.acq ? CASE medm; Rread.val

STRINGTOREAL32 (error, read.val, Rread.val)

repeat := TRUE

WHILE repeat
  SEQ
    WHILE tester
      SEQ
        chn.aski.acq ? CASE float; test -- RECEIVE 'FLAG'
        size.test := test
        IF
          size.test = 10.0 (REAL32)
            SEQ
              count.probe := count.probe + 1.0 (REAL32)
              chn.aski.acq ? CASE long; asi
              STRINGTOREAL32(error,real.data.probe,asi)
              sum.probe := sum.probe + real.data.probe

              -- CALCULATE AND SEND AVERAGE T_PROBE VALUE

              avg.probe := sum.probe/count.probe
              REAL32TOSTRING (len.probe, ase, avg.probe,
                              2, 6)
              chn.aski.cnv ! long; ase
              tester := TRUE
          size.test = 20.0 (REAL32)
            SEQ
              count.ref := count.ref + 1.0 (REAL32)
              chn.aski.acq ? CASE long; aski
              STRINGTOREAL32 (error, real.data.ref, aski)
              sum.ref := sum.ref + real.data.ref

              -- CALCULATE AND SEND AVERAGE T_REF VALUE

              avg.ref := sum.ref/count.ref
```

```
            REAL32TOSTRING (len.ref, aske, avg.ref, 2,
                            6)
            chn.aski.cnv ! long; aske
            tester := TRUE
        TRUE
          tester := FALSE

-- USE T_PROBE AND T_REF VALUES TO CALCULATE THE
-- THERMOCOUPLE TEMPARATURE (TT3 = CENTIGRADE AND
-- TT2 = FARENHEIT)

TT :=  avg.ref * 10.0 (REAL32)
V1 := R0 + (TT*(R1+(TT*R2)))
V := V1 + avg.probe
TT1a := P0 + (V * (P1 + (V * (P2 + (V * (P3 + (V * (P4
        + (V * (P5 + (V * (P6 + (V * (P7 + (V * (P8 +
        (V * P9))))))))))))))))
TT2 := ((32.0(REAL32)) + ((1.8(REAL32)) * TT1a))

TT3 := (((TT1a * CON1) + (0.5(REAL32)))) / (CON1)

-- ERROR VALUE IS EVALUATED

eror := read.val - TT3

-- SEND TEMPARATURE VALUE TO PROC_IO

REAL32TOSTRING (len.temp, askt, TT3, 2, 2)
chn.aski.cnv ! long; askt

REAL32TOSTRING (len.temp, askt, TT3.old, 2, 2)
chn.aski.cnv ! long; askt

-- EVALUATE 'CONTROL INTERVAL',  'ON.time' AND
-- 'OFF.time'.

interval := read.val * kc

switch.on.point := eror * kc
switch.off.point := interval - switch.on.point

REAL32TOSTRING (len.switch, switch.val,
                 switch.on.point, 2, 4)
chn.aski.cnv ! long; switch.val

ON.time := INT ROUND (switch.on.point * 15625.0
                      (REAL32))
OFF.time := INT TRUNC (switch.off.point * 15625.0
                       (REAL32))

clock ? timenow
INTTOSTRING (len, time, timenow)
```

```
        chn.aski.cnv ! long; time

        -- SEND HEATER 'ON' COMMAND TO PROC_IO

        chn.aski.cnv ! short; ON

        clock ? AFTER timenow PLUS ON.time

        -- AFTER ON.time TURN HEATER 'OFF' FOR A DURATION OF
        -- 'OFF' TIME.

        chn.aski.cnv ! short; OFF

        clock ? AFTER timenow PLUS OFF.time

        -- SEND ERROR VALUE TO PROC_IO

        REAL32TOSTRING (len.er, er.val, eror, 2, 4)
        chn.aski.cnv ! long; er.val

        tac := tac + step
        tester := TRUE
        count.probe := 0.0 (REAL32)
        sum.probe := 0.0 (REAL32)
        avg.probe := 0.0 (REAL32)
        count.ref := 0.0 (REAL32)
        sum.ref := 0.0 (REAL32)
        avg.ref := 0.0 (REAL32)
:
```

PROCESS PROC_NET.PGM

```
-- PROCESS PROC_NET.PGM DESCRIBES THE NETWORK CONFIGURATION
-- WITH THE LINK CONNECTIONS BETWEEN PROCESSORS EXPLICITLY
-- SPECIFIED.

#INCLUDE "hostio.inc"
#INCLUDE "linkaddr.inc"
#INCLUDE "acqprot.inc"

-- OUTPUT FILES FROM THE `ILINK` TOOL ARE USED

#USE "PROC_IO.c2h"
#USE "PROC_CON.c8h"
CHAN OF SP fs, ts:
CHAN OF ACQPROT chn.aski.acq, chn.aski.cnv:

-- PROCESSES PROC_IO AND PROC_CON ARE PLACED PARALLEL ON THE
-- INDIVIDUAL TRANSPUTERS

PLACED PAR
  PROCESSOR 0 T222
    PLACE fs AT link1.in:
    PLACE ts AT link1.out:
    PLACE chn.aski.acq AT link3.out:
    PLACE chn.aski.cnv AT link3.in:
    PROC.IO (fs, ts, chn.aski.acq, chn.aski.cnv)
  PROCESSOR 1 T800
    PLACE chn.aski.acq AT link3.in:
    PLACE chn.aski.cnv AT link3.out:
    PROC.CON (chn.aski.acq, chn.aski.cnv)
```

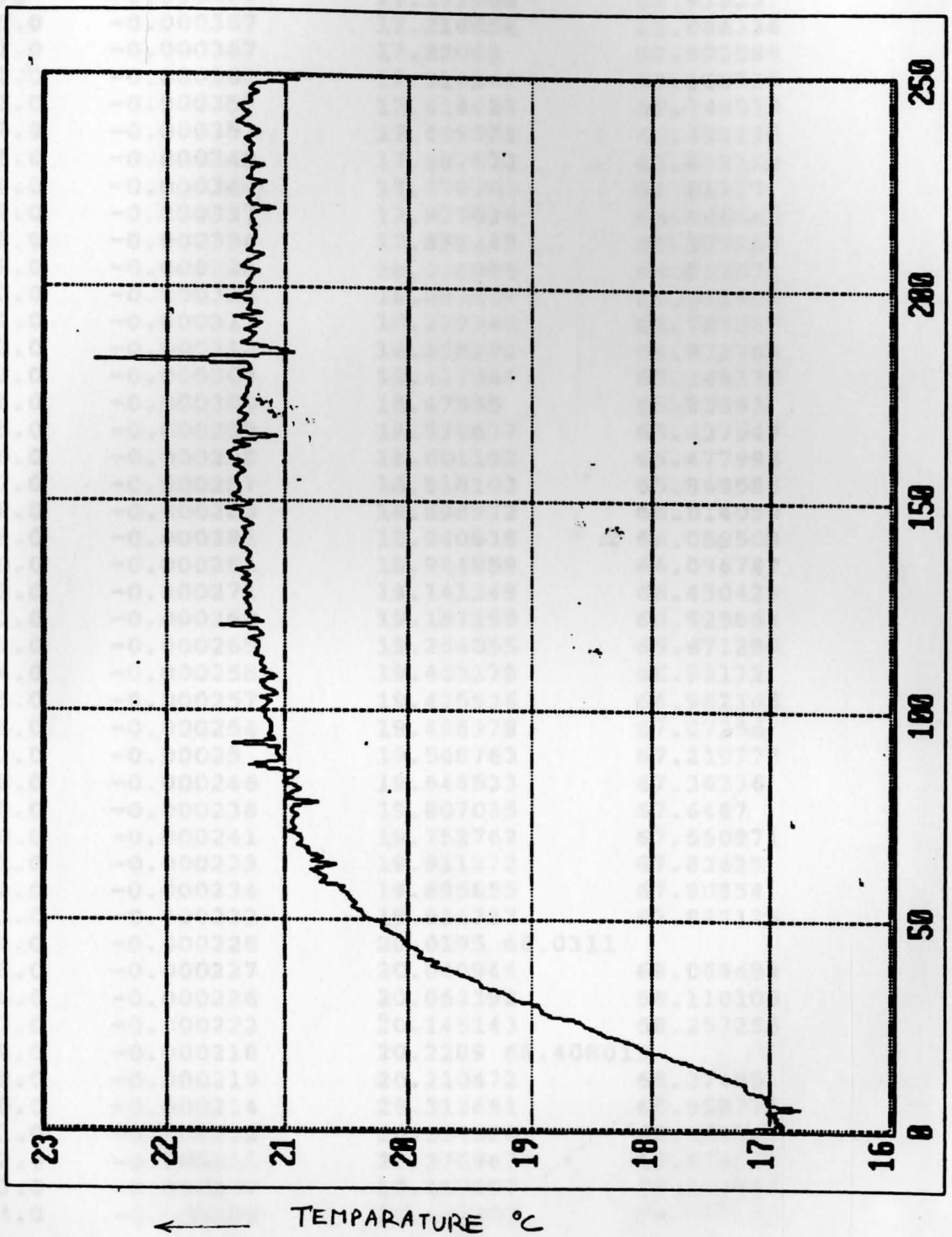
PROGRAM DATA_ACQ.BAS

```
REM ************************************************************
REM * THE FOLLOWING QuickBasic PROGRAM IS RUN IN THE Lab -
REM * - Windows ENVIRONMENT TO ACQUIRE THE THERMOCOUPLE
REM * VOLTAGE AND HENCE DETERMINE THE TEMPARATURE, WHILE
REM * CONDUCTING THE `DYNAMIC TEST`.  THE PROCESS REACTION
REM * CURVE SHOWS THE TEMPARATURE Vs TIME PROFILE.
REM ************************************************************

DIM voltage#(1000), refvoltage#(1000), temp#(1000), T1#(1000)

REM ************************************************************
REM * THE CONSTANTS ASSOCIATED WITH  THE CALCULATION OF THE
REM * TEMPARATURE ARE LISTED BELOW.
REM ************************************************************

RO = -0.75004344E-6
R1 = .505321995E-4
R2 = .2348050017E-7
P0 = -.3595568424
P1 = 19750.87948
P2 = -175116.5425
P3 = 18212965.58
P4 = -2831128E3
P5 = 2715083833E2
P6 = -1.38014121E13
P7 = 3.79242384326E14
P8 = -5.371925517E15
P9 = 3.0840255439E16

CLS

INPUT "ENTER THE NUMBER OF SAMPLES REQUIRED ";A

REM ************************************************************
REM * CALL THE INITIALIZING ROUTINE FOR THE HP3497A
REM ************************************************************

CALL hp3497a.init (9)

REM ************************************************************
REM * LOOP 'I' TIMES TO ACQUIRE THE THERMOCOUPLE PROBE AND
REM * REFERENCE VOLTAGES AND CALCULATE THE TEMPARATURE VALUE
REM * USING THE CONSTANTS LISTED ABOVE.
REM ************************************************************

FOR I = 0 TO A-1
```

```
CALL hp3497a.init (9)
CALL hp3497a.Conf.Elapsed.Time (0)
CALL hp3497a.Conf.Elapsed.Time (2)
CALL hp3497a.Read.Elapsed.Time (del%)

REM *********************
REM * WAIT FOR 5 SECONDS
REM *********************

while del% < 5
CALL hp3497a.Read.Elapsed.Time (del%)
wend
CALL hp3497a.init (9)

REM ************************************************************
REM * CALL THE ROUTINE TO READ ANALOG CHANNEL #1 (PROBE
REM * VOLTAGE)
REM ************************************************************

CALL hp3497a.ReadCh (1, volt#)

REM ************************************************************
REM * CALL THE ROUTINE TO READ ANALOG CHANNEL #19 (REFERENCE
REM * VOLTAGE)
REM ************************************************************

CALL hp3497a.ReadCh (19, refvolt#)

refvoltage#(I) = refvolt#
voltage#(I) = volt#
T = refvoltage(I)  * 10
V1 = R0 + T*(R1 + T * R2)
V = V1 + voltage#(I)
T1#(I) = P0+V*(P1+V*(P2+V*(P3+V*(P4+V*(P5+V*(P6+V*(P7+V*(P8+
         V*P9)))))))
T2 = 32+1.8*T1#(I)
T3 = (T1#(I)*100+.5)/100
T4 = (T2*100+.5)/100
PRINT I;voltage#(I);T3;T4

del% = 0

NEXT I

REM************************************************************
REM * CALL THE ROUTINES TO RESET AND PLOT THE APPROPRIATE
REM * CURVES
REM************************************************************

CALL GrfLReset (0, 0, 1, 1)
CALL GrfYCurv2D (T1#(), 600)
CALL HardCopy

END
```

APPENDIX B

Process reaction curve



Data for the process reaction curve

INDEX	THERMOCOUPLE VOLTAGE (VOLTS)	°C	°F
1.0	-0.000383	16.887117	62.392811
2.0	-0.000379	16.966035	62.534863
3.0	-0.00038	16.948571	62.503429
4.0	-0.000376	17.030506	62.65091
5.0	-0.00039	16.760833	62.165499
6.0	-0.000376	17.033526	62.656347
7.0	-0.000376	17.03554	62.659971
8.0	-0.000377	17.018077	62.628538
9.0	-0.000369	17.177906	62.91623
10.0	-0.000367	17.216854	62.986336
11.0	-0.000367	17.22088	62.993584
12.0	-0.000362	17.318244	63.168839
13.0	-0.000357	17.418621	63.349518
14.0	-0.000353	17.499521	63.495138
15.0	-0.000348	17.597872	63.672169
16.0	-0.000344	17.678761	63.81777
17.0	-0.000337	17.817036	64.066665
18.0	-0.000334	17.878449	64.177207
19.0	-0.000324	18.076095	64.532971
20.0	-0.000323	18.097569	64.571623
21.0	-0.000317	18.217348	64.787227
22.0	-0.000313	18.298202	64.932763
23.0	-0.000307	18.417964	65.148335
24.0	-0.000304	18.47935	65.25883
25.0	-0.000299	18.578637	65.437547
26.0	-0.000298	18.601108	65.477995
27.0	-0.000287	18.818103	65.868586
28.0	-0.000283	18.898922	66.014059
29.0	-0.000281	18.940838	66.089508
30.0	-0.000281	18.944859	66.096747
31.0	-0.000271	19.141349	66.450429
32.0	-0.000269	19.183258	66.525864
33.0	-0.000265	19.264055	66.671299
34.0	-0.000258	19.403178	66.92172
35.0	-0.000257	19.425636	66.962145
36.0	-0.000254	19.486978	67.07256
37.0	-0.00025	19.568763	67.219773
38.0	-0.000246	19.648533	67.36336
39.0	-0.000238	19.807055	67.6487
40.0	-0.000241	19.752762	67.550971
41.0	-0.000233	19.911272	67.83629
42.0	-0.000234	19.895855	67.80854
43.0	-0.000232	19.936737	67.882126
44.0	-0.000228	20.0195 68.0311	
45.0	-0.000227	20.040944	68.069699
46.0	-0.000226	20.063392	68.110105
47.0	-0.000222	20.145143	68.257258
48.0	-0.000218	20.2289 68.408019	
49.0	-0.000219	20.210472	68.37485
50.0	-0.000214	20.312651	68.558771
51.0	-0.000212	20.354524	68.634144
52.0	-0.000211	20.376967	68.674541
53.0	-0.000207	20.457697	68.819854
54.0	-0.000209	20.422857	68.757143
55.0	-0.000206	20.48516	68.869288
56.0	-0.000206	20.489178	68.87652
57.0	-0.000203	20.551478	68.98866

INDEX	THERMOCOUPLE VOLTAGE (VOLTS)	°C	°F
58.0	-0.000204	20.535064	68.959115
59.0	-0.000195	20.713916	69.281049
60.0	-0.000198	20.659658	69.183384
61.0	-0.000195	20.722955	69.297318
62.0	-0.000191	20.803664	69.442595
63.0	-0.000202	20.594005	69.06521
64.0	-0.000192	20.793278	69.423901
65.0	-0.000197	20.700173	69.256312
66.0	-0.000191	20.819732	69.471517
67.0	-0.000191	20.823748	69.478747
68.0	-0.000187	20.905456	69.625822
69.0	-0.000189	20.870628	69.563131
70.0	-0.000188	20.894068	69.605322
71.0	-0.000184	20.97778	69.756004
72.0	-0.000187	20.923531	69.658357
73.0	-0.000185	20.967396	69.737312
74.0	-0.000186	20.951991	69.709583
75.0	-0.000188	20.917164	69.646895
76.0	-0.000186	20.960024	69.724043
77.0	-0.000187	20.946627	69.699929
78.0	-0.000198	20.737994	69.324389
79.0	-0.000185	20.993503	69.784306
80.0	-0.000185	20.998524	69.793343
81.0	-0.000191	20.887014	69.592626
82.0	-0.000189	20.93088	69.671584
83.0	-0.000184	21.033007	69.855412
84.0	-0.000189	20.941926	69.691467
85.0	-0.000191	20.905091	69.625163
86.0	-0.000181	21.105325	69.985584
87.0	-0.00017	21.323943	70.379098
88.0	-0.000185	21.036682	69.862027
89.0	-0.000184	21.061123	69.906021
90.0	-0.000183	21.085563	69.950014
91.0	-0.000184	21.07016	69.922288
92.0	-0.000164	21.464531	70.632155
93.0	-0.000183	21.099621	69.975318
94.0	-0.000181	21.14348	70.054265
95.0	-0.000182	21.130086	70.030155
96.0	-0.000182	21.134102	70.037384
97.0	-0.000182	21.139123	70.046422
98.0	-0.00018	21.182981	70.125366
99.0	-0.00018	21.188002	70.134403
100.0	-0.000179	21.21244	70.178393
101.0	-0.000181	21.179628	70.119331
102.0	-0.000184	21.126392	70.023506
103.0	-0.00018	21.209088	70.172358
104.0	-0.000187	21.080184	69.940331
105.0	-0.000182	21.180292	70.120526
106.0	-0.000181	21.205735	70.166324
107.0	-0.000181	21.212764	70.178975
108.0	-0.000188	21.082856	69.945142
109.0	-0.000181	21.223809	70.198857
110.0	-0.000183	21.189994	70.137989
111.0	-0.000182	21.215437	70.183786
112.0	-0.000183	21.200035	70.156064
113.0	-0.00018	21.264314	70.271764
114.0	-0.000181	21.249917	70.24585
115.0	-0.000182	21.237528	70.22355

INDEX	THERMOCOUPLE VOLTAGE (VOLTS)	°C	°F
117.0	-0.000178	21.325238	70.381428
118.0	-0.000185	21.196345	70.14942
119.0	-0.000182	21.259619	70.263314
120.0	-0.000167	21.556875	70.798374
121.0	-0.000186	21.193998	70.145196
122.0	-0.00018	21.315523	70.363942
123.0	-0.000179	21.342973	70.413351
124.0	-0.000181	21.308156	70.350681
125.0	-0.000181	21.314181	70.361526
126.0	-0.000179	21.359039	70.44227
127.0	-0.000182	21.306814	70.348265
128.0	-0.00018	21.351672	70.42901
129.0	-0.000181	21.337276	70.403097
130.0	-0.000183	21.305472	70.34585
131.0	-0.00018	21.368742	70.459736
132.0	-0.000183	21.317522	70.36754
133.0	-0.000178	21.419623	70.551321
134.0	-0.000185	21.289735	70.317523
135.0	-0.000184	21.315177	70.363318
136.0	-0.000183	21.340618	70.409113
137.0	-0.000185	21.307811	70.350059
138.0	-0.000184	21.333252	70.395854
139.0	-0.000181	21.397525	70.511546
140.0	-0.000183	21.364719	70.452494
141.0	-0.000184	21.351328	70.42839
142.0	-0.000185	21.337937	70.404286
143.0	-0.000184	21.365387	70.453696
144.0	-0.000188	21.291738	70.321128
145.0	-0.000185	21.357017	70.438631
146.0	-0.000184	21.383462	70.486232
147.0	-0.000185	21.369068	70.460322
148.0	-0.000178	21.510998	70.715796
149.0	-0.000189	21.303453	70.342215
150.0	-0.000186	21.367728	70.45791
151.0	-0.000182	21.452419	70.610354
152.0	-0.000185	21.400199	70.516357
153.0	-0.000186	21.386808	70.492255
154.0	-0.000184	21.43066	70.571189
155.0	-0.000185	21.420283	70.55251
156.0	-0.000186	21.405889	70.5266
157.0	-0.000189	21.354671	70.434409
158.0	-0.000185	21.438359	70.585047
159.0	-0.000189	21.366723	70.456101
160.0	-0.000184	21.470829	70.643492
161.0	-0.000183	21.496269	70.689284
162.0	-0.000186	21.44405	70.595291
163.0	-0.000193	21.314166	70.361498
164.0	-0.000187	21.438695	70.585651
165.0	-0.000206	21.074785	69.930613
166.0	-0.000189	21.411916	70.537449
167.0	-0.00019	21.398527	70.513348
168.0	-0.00019	21.405557	70.526002
169.0	-0.000193	21.356349	70.437427
170.0	-0.000194	21.34095	70.40971
171.0	-0.000191	21.407232	70.529018
172.0	-0.000194	21.355011	70.43502
173.0	-0.000195	21.342625	70.412726
174.0	-0.000194	21.368068	70.458522

INDEX	THERMOCOUPLE VOLTAGE (VOLTS)	°C	°F
176.0	-0.000196	21.341288	70.410319
177.0	-0.000197	21.327898	70.386217
178.0	-0.000198	21.316517	70.365731
179.0	-0.000195	21.381797	70.483234
180.0	-0.000194	21.407239	70.52903
181.0	-0.0002	21.297773	70.331991
182.0	-0.000198	21.342633	70.412739
183.0	-0.000134	22.590759	72.659367
184.0	-0.000199	21.336274	70.401294
185.0	-0.00022	20.935505	69.679909
186.0	-0.000207	21.196003	70.148806
187.0	-0.000197	21.395195	70.507352
188.0	-0.000201	21.324562	70.380212
189.0	-0.000204	21.273343	70.288018
190.0	-0.0002	21.358041	70.440475
191.0	-0.000204	21.287407	70.313332
192.0	-0.000208	21.216768	70.186183
193.0	-0.000197	21.437382	70.583287
194.0	-0.000209	21.21041	70.174738
195.0	-0.000203	21.333946	70.397103
196.0	-0.000205	21.30114	70.338052
197.0	-0.000205	21.308172	70.350709
198.0	-0.000194	21.528771	70.747788
199.0	-0.000205	21.322236	70.376024
200.0	-0.000207	21.290434	70.318781
201.0	-0.000205	21.335295	70.399531
202.0	-0.000206	21.323915	70.379047
203.0	-0.000208	21.292114	70.321805
204.0	-0.000211	21.239889	70.227801
205.0	-0.000204	21.381834	70.483302
206.0	-0.000207	21.330618	70.391112
207.0	-0.000211	21.258978	70.26216
208.0	-0.000214	21.208762	70.171771
209.0	-0.000208	21.332298	70.394137
210.0	-0.000211	21.28108	70.301945
211.0	-0.000211	21.286104	70.310987
212.0	-0.000212	21.273719	70.288695
213.0	-0.000213	21.262339	70.268211
214.0	-0.000211	21.306197	70.347155
215.0	-0.000213	21.276405	70.293529
216.0	-0.000213	21.282433	70.30438
217.0	-0.000212	21.308883	70.351989
218.0	-0.000214	21.276077	70.292939
219.0	-0.000224	21.089931	69.957875
220.0	-0.000213	21.310565	70.355017
221.0	-0.000212	21.33601	70.400817
222.0	-0.000215	21.284792	70.308626
223.0	-0.000214	21.311242	70.356236
224.0	-0.000216	21.279442	70.298995
225.0	-0.000221	21.189386	70.136895
226.0	-0.000214	21.333346	70.396023
227.0	-0.000216	21.300541	70.336974
228.0	-0.000216	21.307575	70.349634
229.0	-0.000222	21.198103	70.152585
230.0	-0.000213	21.380895	70.48161
231.0	-0.000217	21.310263	70.354473
232.0	-0.000218	21.297879	70.332183
233.0	-0.000217	21.323325	70.377984
234.0	-0.000219	21.293534	70.324362

APPENDIX C

3497A COMMANDS

ANALOG

AC chan# [chan#] (chan# = 0 to 999)
ANALOG CLOSE.
Closes 1 to 4 channels (one per decade) of analog assemblies.

AER, n = 0 to 2 (AE0 = EXT INCR OFF; AE1 = EXT INCR ON; AE2 = FAST SCAN)
ANALOG EXTERNAL INCREMENT.
Enables or disables the EXT INCR port. In FAST SCAN (AE2), multiframe BBM Sync is ignored. In AE1, external pulse into EXT INCR port increments channel closed to next channel.

AF chan# (chan# = 0 to 999)
ANALOG FIRST CHANNEL.
Selects first channel to be closed in an analog sequence but does not close channel.

AI chan# (chan# = 0 to 999)
ANALOG INPUT.
Closes channel and triggers DVM to take a measurement.

AL chan# (chan# = 0 to 999)
ANALOG LAST CHANNEL.
Selects last channel to be closed in an analog sequence but does not close channel.

AO slot#, chan #, VAL (slot# 0 to 89; chan# 0 or 1; value 0 to ±10238 (VDAC) or 0 to 10238 (IDAC))
ANALOG OUTPUT.
Sets the output voltage level for the VDAC and output current level for the IDAC. VDAC output is -10.2378V to +10.2378V in 2.5 mV increments. IDAC output is 0 - 20 mA (5 μA increments) or 4 - 20 mA (4 μA increments).

AR
ANALOG RESET.
Opens analog assembly channels in 3497A and 3498A and sets VF1, VT1, VR6, VW0, VS0, AS0, AF0 and AL999.

AS
ANALOG STEP.
Performs software channel advance from the presently closed channel to next channel. Repeating the command sequences channels from AF to AL and back to AF. If AF < AL, channels increment. If AF > AL, channels decrement.

AV chan# (chan# = 0 to 999)
ANALOG VIEWED CHANNEL.
Dedicates display to channel selected but does not close channel and does not affect other 3497A operations. Display is updated when channel closed and measurement taken.

VOLTMETER

VAn (VA0 = Autozero OFF; VA1 = Autozero ON)
VOLTMETER AUTOZERO.
With autozero on, DVM takes measurement between each reading. With autozero off, DVM makes autozero measurement before first reading and when DVM switched to new range.

VCn n = 0 to 3 (0 = OFF; 1 = 10 μA; 2 = 100 μA; 3 = 1 mA)
VOLTMETER CURRENT SOURCE RANGE.
Programs output of DVM current source to 1 of 3 values: 10 μA, 100 μA or 1 mA.

VDn n = 3 to 5 (3 = 3 1/2 digits; 4 = 4 1/2 digits; 5 = 5 1/2 digits)
VOLTMETER DISPLAY.
Selects number of digits to be displayed on front panel and sets voltmeter integration time. Max reading rate for 60 Hz operation is 300 readings/sec (Autozero OFF). Max rate for 50 Hz is 250 readings/sec.

VFn n = 1 to 3 (1 = ASCII; 2 = Packed BCD; 3 = Time ASCII Char#)
VOLTMETER FORMAT.
Selects the output format for transmission of data over the bus, when voltmeter storage is off (VS0).

VNn n = 1 to 999
VOLTMETER NUMBER READINGS/TRIGGER.
Sets number of readings taken per trigger pulse input. Readings are taken sequentially and output over the bus in format set by VFn.

VRn n = 1 to 5 (1 = 0.1V; 2 = 1.0V; 3 = 10 V; 4 = 100V; 5 = Autorange)
VOLTMETER RANGE.
Sets range of DVM. Maximum overrange capability for each range is 120% of full-scale. In autorange, DVM upranges at 120% of full-scale and downranges at 11% of full-scale.

VSn n = 0 to 2 (0 = Storage OFF; 1 = Store in ASCII; 2 = Packed BCD)
VOLTMETER STORAGE.
Store up to 80 readings in ASCII (50 for Serial Data) or up to 100 readings in Packed BCD (96 in Serial Data). Use VS without number to transfer readings to controller.

VTn n = 1 to 4 (1 = Internal; 2 = External; 3 = Software; 4 = Hold)
VOLTMETER TRIGGER.
Set one of four trigger modes. In internal, DVM automatically takes another reading when present one completed. In external, trigger signal input to EXT TRIG port causes DVM to take n readings/trigger (as set by VNnn). In software, command causes DVM to trigger and take n readings as set by VNn. In hold, DVM pauses and does not take measurements.

VWn = 0 to 999999
VOLTMETER WAIT.
Causes the DVM to wait n x 100 μsec between each reading. Maximum wait time is 99.9999 sec.

DIGITAL

DC slot# chan#, chan# (slot# = 0 to 89; chan# = 0 to 15)
DIGITAL CLOSE.
For Option 110 assembly, command connects NO contact to common. For Option 115 assembly, command closes channel relays. Channels not specified remain in previous state.

DE slot# value (slot# = 0 to 4; value = 0 to 377 (Oct))
DIGITAL INTERRUPT ENABLE.
Enables the Option 060 assembly to send an interrupt to the 3497A when channel bits selected by the command are set true (by external input to the assembly).

DI slot# (slot# = 0 to 4)
DIGITAL INTERRUPT STATUS.
Used to determine interrupt status of bits 0 - 7 in the Option 060 assembly. Also used to determine cause of interrupt from the Option 060 assembly.

DL slot# (slot# = 0 to 89)
DIGITAL LOAD.
For Option 060 assembly, returns octal value (0 - 177777) of contents of 16 input channels. For Option 110 assembly, returns octal value (0 - 177777) of condition of 16 output channels. For Option 115 assembly, returns octal value (0 - 377) of condition of 8 channel relays.

DO slot# chan# chan# (slot# = 0 to 89; chan# = 0 to 15)
DIGITAL OPEN.
For Option 110 assembly, connects NC contact to common for channels specified. For Option 115 assembly, opens relays in channels specified. Relays in channels not specified remain in previous state.

DP slot# (slot# = 0 to 89)
DIGITAL READ.
For HP-IB, DR returns same information as DL command, except that readings are continuously updated. For Serial Data, with SO1 in effect returns continuously updated readings. With SO0 in effect, returns one reading per command.

DS slot# (slot# = 0 to 4; Value = 0 to 377 Octal)
DIGITAL INTERRUPT SENSE.
Sets edge transition sense which will cause channel 0 - 7 bits to be set in an Option 060 assembly. Polarity sense set by octal value. Polarity sense 1 = chan bit set by low-to-high transition.

DV slot# (slot# = 0 to 89)
DIGITAL VIEWED SLOT.
Dedicates the front panel display to slot specified. To exit this mode, use DV without slot specifier.

DW slot# value (slot# = 0 to 89; value = 0 to 177777 (octal))
DIGITAL WRITE.
For Option 110 assembly, connects NO or NC contact to common as specified by octal value. For Option 115 assembly, opens or closes relays as specified by octal value. All chans of assy in slot addressed are effected by DW command.

SYSTEM

SA

SYSTEM ALARM.
Initiates an audible alarm (BEEP).

SC (Serial Data)

SYSTEM CLEAR.
For Serial Data operation, the SC command is similar to BREAK message, except that SC does not clear the command buffer or return the 3497A to local mode. SC clears system errors but does not reset VF2, VF3 or clear voltmeter storage.

SDn

SD0 = Display OFF
SD1 = Display ON

SYSTEM DISPLAY.
SD0 turns off the 8-digit display and CHANNEL lights for faster reading rates. With SD0, only data entered with SVn command affects display.

SEn (HP-IB)

n = 0 to 377 (octal)

SERVICE REQUEST ENABLE.
SE sets the SRQ mask bits which enables 3497A to send an interrupt to the controller when specified system conditions occur.

SEn (Serial Data)

n = 0 to 377 (octal)

SERVICE REQUEST ENABLE.
SE sets the interrupt mask bits which enables 3497A to send an interrupt to the controller when specified system conditions occur.

SI

SYSTEM INITIALIZE.
Sets the digital assemblies and the DVM to initial conditions but does not affect the analog assemblies.

SLn (Serial Data)

SL0 = Keyboard Enabled
SL1 = Keyboard Disabled

SYSTEM LOCK.
Used to disable the front panel keys so that commands can't be entered from the front panel. With SL1, 3497A can't be returned to local mode unless SL0 is sent or power is turned off.

SOn (HP-IB)

SO0 = Output immed
SO1 = Output reading on controller request

SYSTEM OUTPUT WAIT.
When SO1 in effect, two modes to return data to controller. With VS0, 3497A takes measurement and waits for controller request to transfer data. With VS1 or VS2, 3497A takes n readings (as set by VNn) and waits for controller request to transfer.

SOn (Serial Data)

SO0 = Cont output
SO1 = One output/comd

SYSTEM SINGLE/CONTINUOUS OUTPUT.
SO1 enables 3497A to send a single reading/command for commands which normally return continuous data, such as ST, VT1, DR *slot#*, TD and CR *slot#*,3.

SR slot#,n

slot# = 0 to 89;
n = 0 to 7
SR slot#,0 = Read sig
SR slot#,0 - 7 = Read register (Option 140)

SYSTEM READ.
Use SR *slot#*,0 to determine type of assembly in slot (except analog assemblies). Use SR *slot#*, 0 through 7 to read register n in slot addressed (Option 140 only).

SR (Serial Data)

STATUS REGISTER READ.
The SR command returns a six-bit octal value of the status register true bits.

STn

ST0 = Self Test OFF
ST1 = Self Test ON

SELF-TEST
ST1 causes 3497A to perform internal self-test. 0E8 returned if self-test passes.

SVn

n = ±999999

SYSTEM VIEW.
When the display is turned off by an SD0 command, the SV command writes data specified by n to the display.

SW slot#,register#,data

slot# = 0 to 89
register# = 0 to 7
data = 0 to 377

SYSTEM WRITE.
Use SW to write data to any assembly directly controlled by the main processor (i.e. digital assemblies).

TIMER

TA HH MM SS

Hours = 0 to 24
Minutes = 0 to 59
Seconds = 0 to 59

TIME ALARM (SET).
Sets 3497A timer. If SRQ mask (HP-IB) or interrupt mask (Serial Data) has been set for time alarm, interrupt sent to controller when time on real-time clock matches time set by TA.

TD MMDDHHMMSS or TD DDMMHHMMSS

TIME OF DAY (SET).
Sets 3497A real-time clock to programmed time.

TD

TIME OF DAY (READ).
Reads time of day from real-time clock. Data returned has format MM:DD:HH:MM:SS or (European) DD:MM:HH:MM:SS.

TEn

TE0 = RESET
TE1 = HALT
TE2 = START

ELAPSED TIME (CONTROL).
Use TEn to monitor elapsed time from start of an operation. Use the TE command (without a number) to read time elapsed since TE2 command received.

TE

ELAPSED TIME (READ).
Use TE to read elapsed time (1 sec increments) since elapsed timer control started by TE2 command. Data returned has format DDDDDD sec.

TI HH MM SS

TIME INTERVAL.
Use TIn to generate pulses from TIMER port with periods from 1 sec to 24 hr. If SRQ or interrupt mask set, 3497A sends interrupt for every pulse output.

TOn

n = 0 to 9999

TIME OUTPUT.
Use TOn to generate pulses from TIMER port with periods from 100 μsec to 0.9999 sec (in 100 μsec increments). Period output is n x 100 μsec. Interrupt not available with TOn command.

COUNTER

CE slot# n

slot# = 0 to 4
n = 0 to 2

0 = No interrupts enabled
1 = Interrupt on measurement complete
2 = Interrupt on overflow

COUNTER ENABLE INTERRUPTS.
Enables counter to send an interrupt to 3497A when specified interrupt condition occurs. If 3497A is set for Digital Interrupt, interrupt is sent to controller.

CF slot# n

slot# = 0 to 89
n = 0 to 6

0 = Counter Stop
1 = Count Up
2 = Count Down
3 = Avg 1000 Periods
4 = Avg 100 Periods
5 = Measure 1 Period
6 = Measure 1 Period

COUNTER FUNCTION.
Sets mode of operation for the counter and starts the function. CT command MUST be set before CF command is executed. For n = 3 to 6, CT *slot#*, 1 and 2 set period measurements and CT *slot#*, 3 and 4 set pulse width measurements.

CR slot# n

slot# = 0 to 89
n = 1 to 3

1 = Read without wait
2 = Read with wait
3 = Read continuously

COUNTER READ.
Allows the results of counter measurements to be read in one of three ways.

CS slot# value

slot# = 0 to 89
value = 0 to 999999

COUNTER SET.
Sets the start point (0 to 9999999) for the Count Up or Count Down functions. Also sets number of pulses in Pulse Output mode (start point value = twice the number of pulses output).

CT slot#,n

slot# = 0 to 89
n = 1 to 4

1 = Rising/Rising Edges
2 = Falling/Falling Edges
3 = Rising/Falling Edges
4 = Falling/Rising Edges

COUNTER TRIGGER.
Selects edge of input signal on which to trigger counter. For Count Up or Count Down, CT *slot#*, 1 and 3 perform same function as do CT *slot#*, 2 and 4.

BIBLIOGRAPHY

Books

1. Burns, Alan. Programming in OCCAM2. Addison-Wesley, 1988.
2. Coughanowr. Process Systems Analysis and Control. McGrawhill, 1991.
3. HEWLETT PACKARD Ltd. HP3497A Data Acquisition/Control Unit Operating and Service Manual. 1982.
4. INMOS Ltd. B008 User Guide and Reference Manual. 1988.
5. INMOS Ltd. B012 User Guide and Reference Manual. 1988.
6. INMOS Ltd. B211 ITEM Reference Manual. 1987.
7. INMOS Ltd. B421 Engineering Data. 1990.
8. INMOS Ltd. F001A -1 GPIB OCCAM. 1988.
9. INMOS Ltd. OCCAM2 Reference Manual. 1988.
10. INMOS Ltd. OCCAM2 Toolset for IBM/NEC PC. 1989.
11. INMOS Ltd. The Transputer Data Book. 1989.
12. INMOS Ltd. Transputer Devices and *ig* Systems Data Book. 1989.
13. Pountain, Dick and May, David. A Tutorial Introduction to OCCAM Programming. McGrawhill, 1987.
14. Smith and Corripio. Principles and Practice of Automatic Process Control. John Wiley and Sons, 1985.

15. National Instruments Ltd. Labwindows User Guide and Reference Manual. 1988.
16. Phillips, Charles and Harbor, Royce. Feedback Control Systems. Prentice Hall, 1988.

Articles

17. Colin Whitby-Strevens., "Transputers-Past, Present, and Future", IEEE Micro, 8, 16-27, 1990.
18. Halang W. A., "Education of Real-Time Systems Engineers", Microprocessing and Microprogramming, 25, 71-76, 1989.
19. Homewood, M., David, S., May, D., and Shepard, Roger., "The IMS T800 Transputer", IEEE Micro, 7, 10-26, 1987.
20. Katab, A., "A Multiprocessor Architecture for Robot-Arm Control", Microprocessing and Microprogramming, 24, 673-680, 1988.