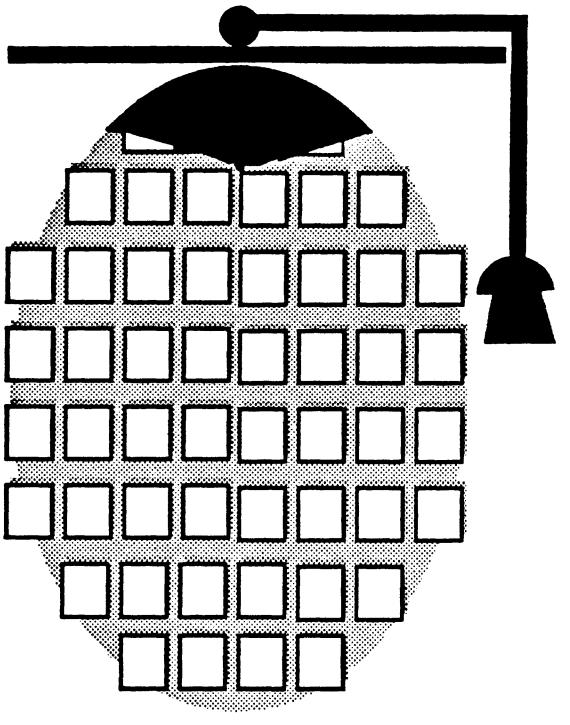


INMOS Transputer and Occam Courses

Parallel C

SGS-THOMSON MICROELECTRONICS

INMOS is a member of the SGS-THOMSON Microelectronics group.

inmos

Parallel C on the Transputer
INMOS Training Course

This course refers to the INMOS IMS D711d software.


, inmos, IMS and occam are trade marks of INMOS Limited.

inmos

Course plan

Monday	Transputers and boards Multi-processor programming Compiling C
Tuesday	Channels Multi-task programs Multi-transputer systems Priority and time
Wednesday	Threads Flood configuring Other tools C implementation

inmos

Programming languages

Languages supported by INMOS on the transputer:

C	Occam
FORTRAN	Pascal

Host machines supported by INMOS:

IBM PC and compatibles

NEC PC

VAX	Sun 3 and 4

inmos

Transputers and boards

Transputer architecture

Parallel processing

The transputer family

INMOS boards

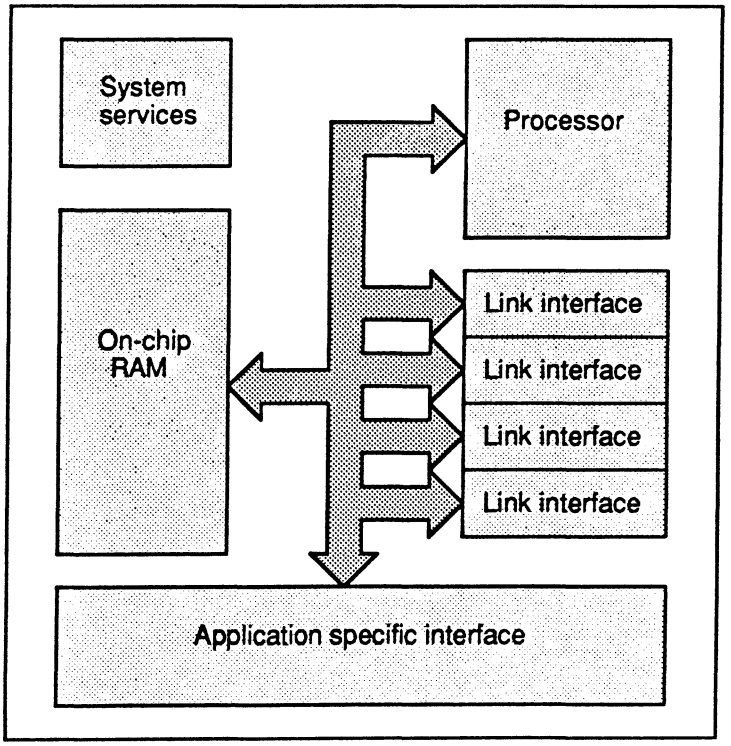
C environment

inmos

Transputer architecture

inmos

Transputer architecture

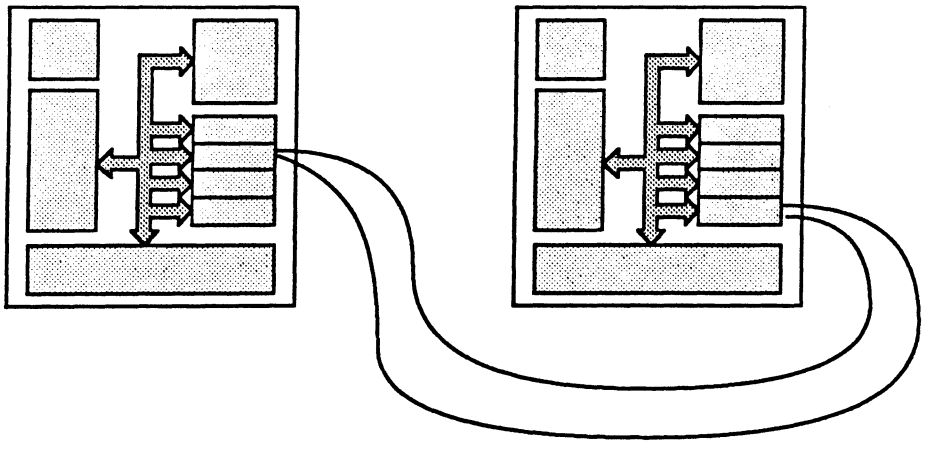


inmos

Parallel processing

inmos

Connecting transputers

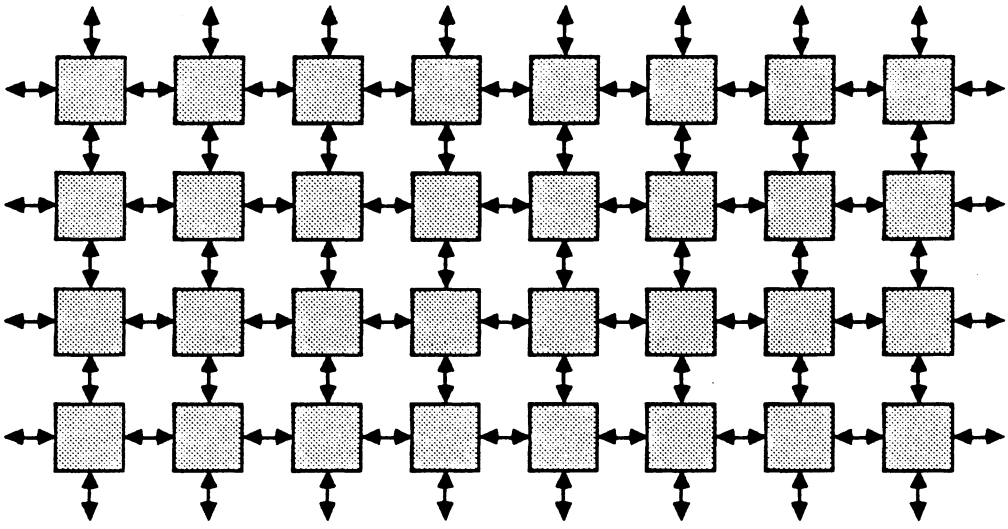


Clock frequencies within 200 ppm.

Clock phases may differ.

inmos

Transputer networks

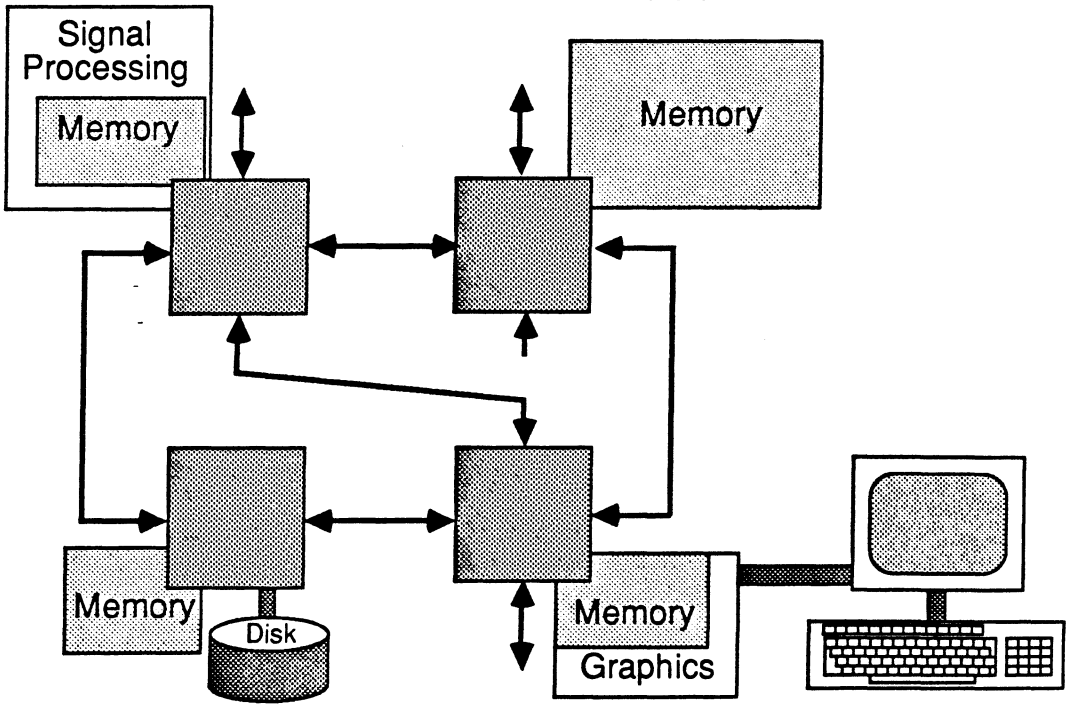


Simple link connections for point-to-point communication.

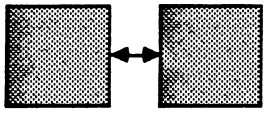
Spare links can connect to peripherals via link adaptors.

inmos

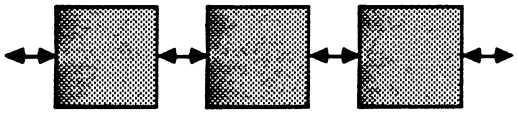
Local resources



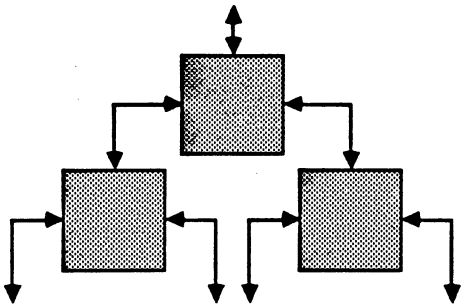
Connectivity



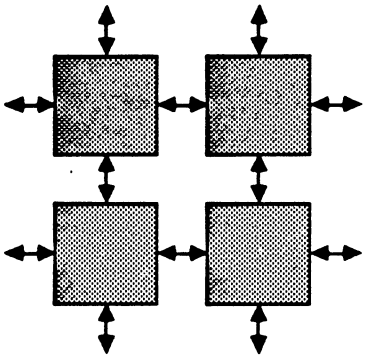
One link connects two transputers.



Two links allow pipelines.



Three links for tree structures.

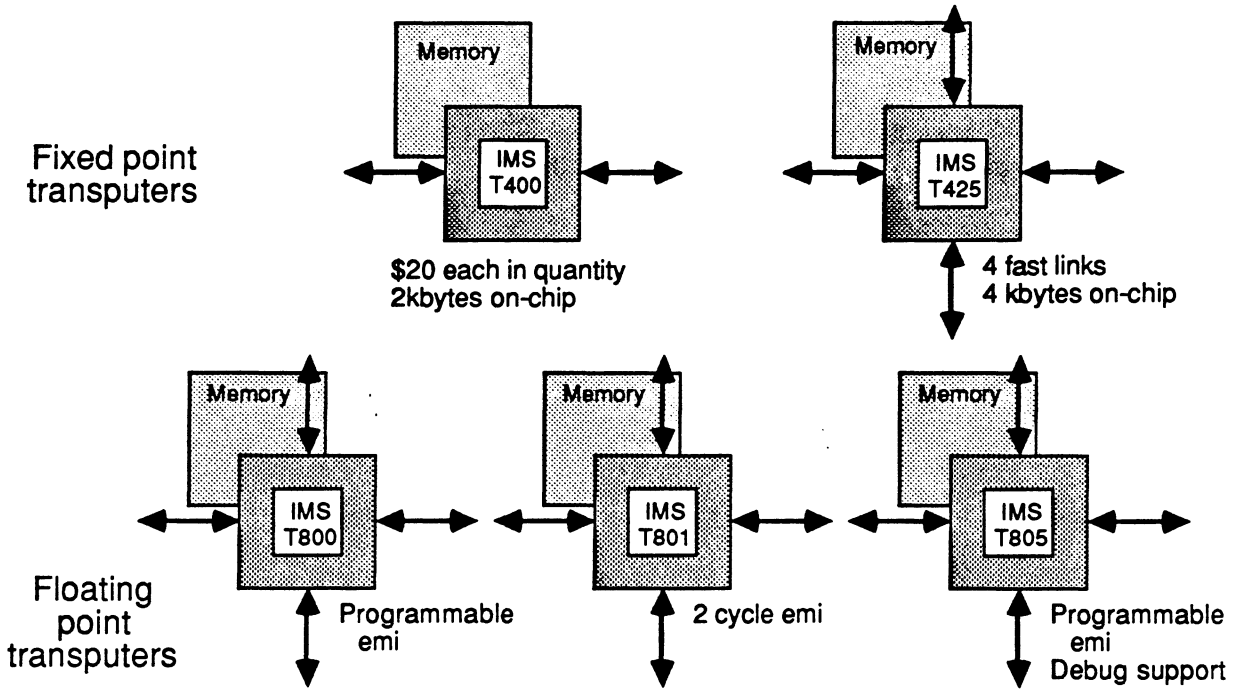


Four links for square arrays.

The transputer family

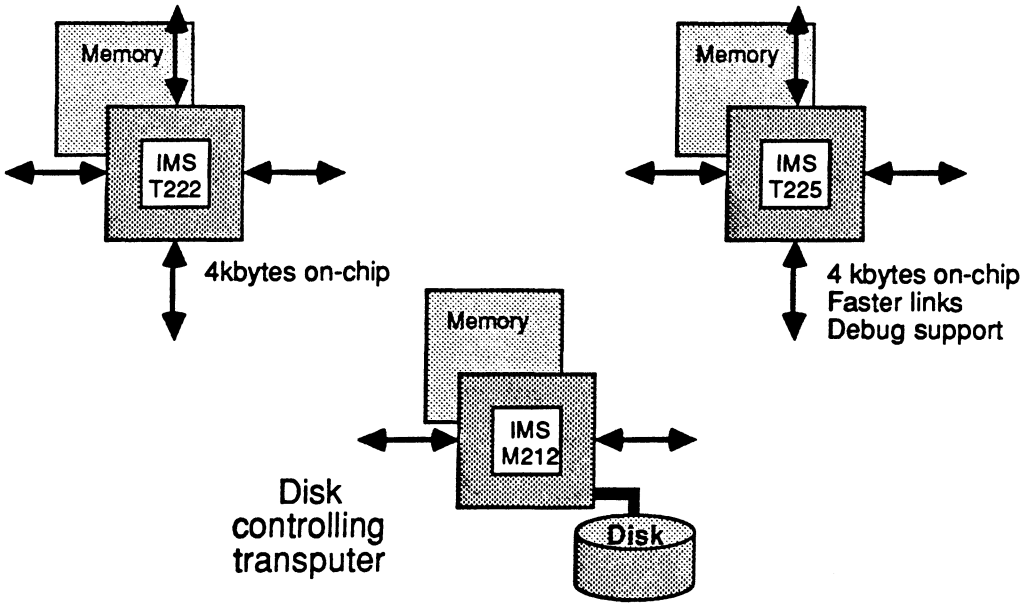
inmos

32 bit transputers



inmos

16 bit transputers



inmos

Transputer differences

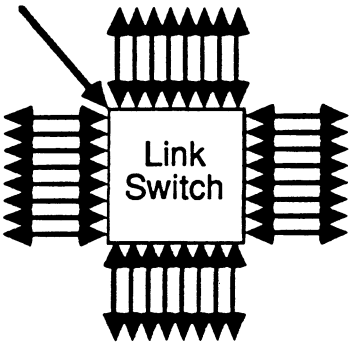
	T212	T222	T225	T414	T400	T425	T800	T801	T805
On-chip memory (bytes)	2k	4k	4k	2k	2k	4k	4k	4k	4k
Floating point hardware	No	No	No	No	No	No	Yes	Yes	Yes
Word length (bits)	16	16	16	32	32	32	32	32	32
Number of links	4	4	4	4	2	4	4	4	4
Overlapped acknowledge	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Double-buffered link output	No	No		No	Yes	Yes	Yes	Yes	Yes
Programmable DRAM controller	No	No	No	Yes	Yes	Yes	Yes	No	Yes
2 cycle external read / write	Yes	Yes	Yes	No	No	No	No	Yes	No
Number of pins	68	68	68	84	84	84	84	100	84
ErrorIn pin	No	No	No	No	Yes	Yes	Yes	No	Yes
RefreshPending pin	No	No	No	No	Yes	Yes	No	No	Yes
EventWaiting pin	No	No	No	No	Yes	Yes	No	Yes	Yes
ProcSpeedSelect pins	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
DisableIntRAM pin	No	Yes	Yes	No	Yes	Yes	Yes	No	Yes
MemBAcc pin	Yes	Yes	Yes	No	No	No	No	No	No
Separate address and data	Yes	Yes	Yes	No	No	No	No	Yes	No
Multiplexed address and data	No	No	No	Yes	Yes	Yes	Yes	No	Yes

inmos

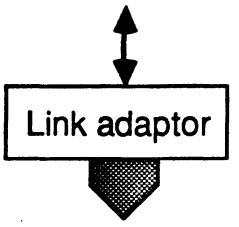
Transputer instruction set differences

	T212	T222	T225	T414	T400	T425	T800	T801	T805
Floating point unit (53)	No	No	No	No	No	No	Yes	Yes	Yes
fptesterror	No	No	No	No	Yes	Yes	Yes	Yes	Yes
Floating point support (5)	No	No	No	Yes	Yes	Yes	No	No	No
fmul	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
2D block moves (4)	No	No	No	No	Yes	Yes	Yes	Yes	Yes
CRC operations(2)	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Bit operations (3)	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Break point debugging (9)	No	No	Yes	No	Yes	Yes	No	Yes	Yes
Fast negative prod	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
dup	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
pop	No	No	Yes	No	Yes	Yes	No	Yes	Yes
wsubdb	No	No	No	No	Yes	Yes	Yes	Yes	Yes
lddevid	No	No	Yes	No	Yes	Yes	No	Yes	Yes
ldmemstartval	No	No	Yes	No	Yes	Yes	No	Yes	Yes

Link chips



For software control
of topology



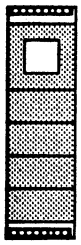
For connection to
buses and peripherals

INMOS boards

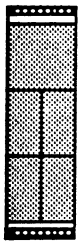
INMOS board range

Compute TRAMs	IMS B410	T801	160k sram	size 2
	IMS B405	T800	8M dram	size 8
	IMS B417	T800	64k sram, 4M dram	size 4
	IMS B403	T414/T800	1M sram	size 4
	IMS B404	T425/T800	32k sram, 2M dram	size 2
	IMS B411	T425/T800	1M dram	size 1
	IMS B401	T4xx/T80x	32k sram	size 1
	IMS B416	T222	64k sram	size 1
	IMS B402	T222	8k sram	size 1
Graphics TRAMs	IMS B419	Graphics G300		size 6
	IMS B408	Graphics drawing and storage		size 8
	IMS B409	Graphics display driver		size 8
i/o TRAMs	IMS B407	Ethernet		size 8
	IMS B415	Link interface		size 1
	IMS B422	SCSI interface		size 2
	IMS B421	GPIB interface		size 4
Other TRAMs	IMS B418	ROM		size 2
	IMS B420	Vector processing		size 4
Motherboards	IMS B008	IBM PC XT/AT		10 slots
	IMS B015	NEC PC		4 slots
	IMS B012	Double Eurocard		16 slots
	IMS B014	VME slave		8 slots

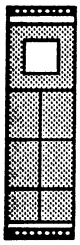
Compute only TRAMs



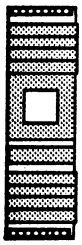
IMS B402
16 bit
8k sram



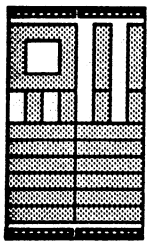
IMS B416
16 bit
64k sram



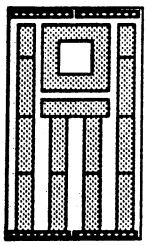
IMS B401
32k sram



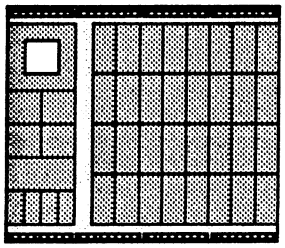
IMS B411
1M dram



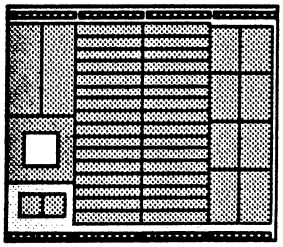
IMS B404
32k sram
2M dram



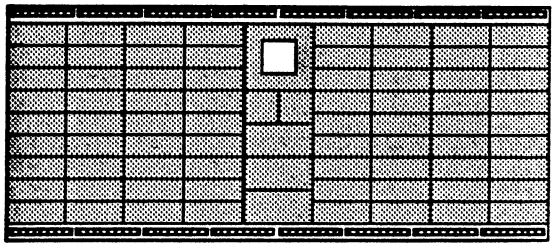
IMS B410
T801
160k sram



IMS B403
1M dram



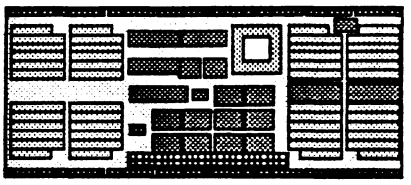
IMS B417
64k sram+4M dram



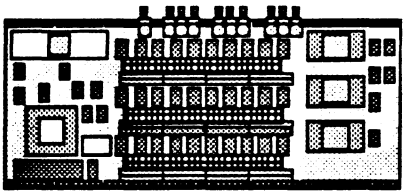
IMS B405
8M dram

inmos

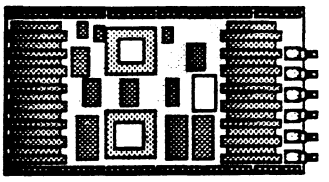
Special application TRAMs



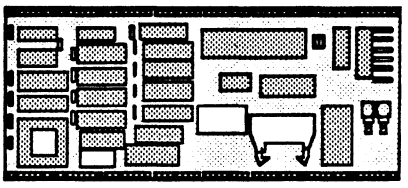
IMS B408
Graphics drawing and storage



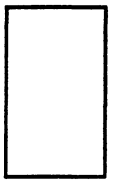
IMS B409
Graphics display driver



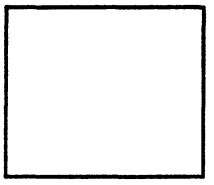
IMS B419
G300 Graphics



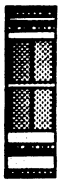
IMS B407
Ethernet



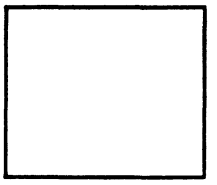
IMS B422
SCSI i/f



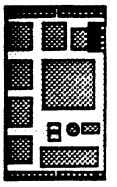
IMS B421
GPIB i/f



IMS B415
link i/f



IMS B420
Vector processor



IMS B418
ROM

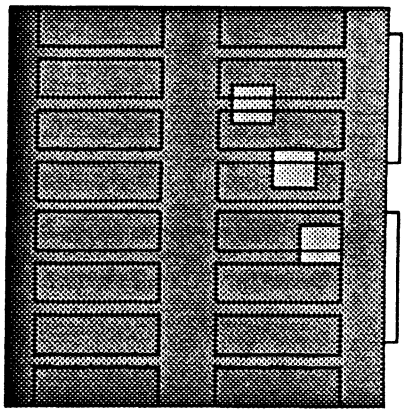
SGS-THOMSON MICROELECTRONICS

inmos

Motherboards

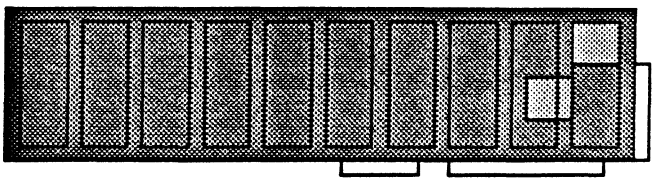
IMS B012

Double ext eurocard
IMS T212
and 2xIMS C004
16 module slots



IMS B008

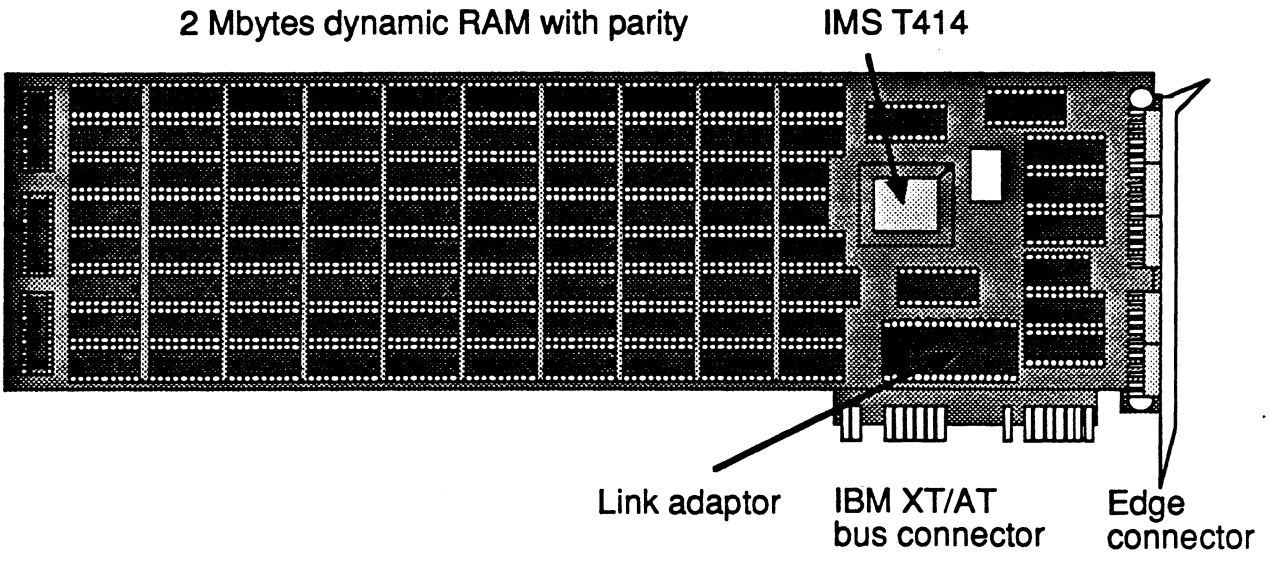
PC add-in board
IMS T212
and IMS C004
10 module slots



SGS-THOMSON MICROELECTRONICS

inmos

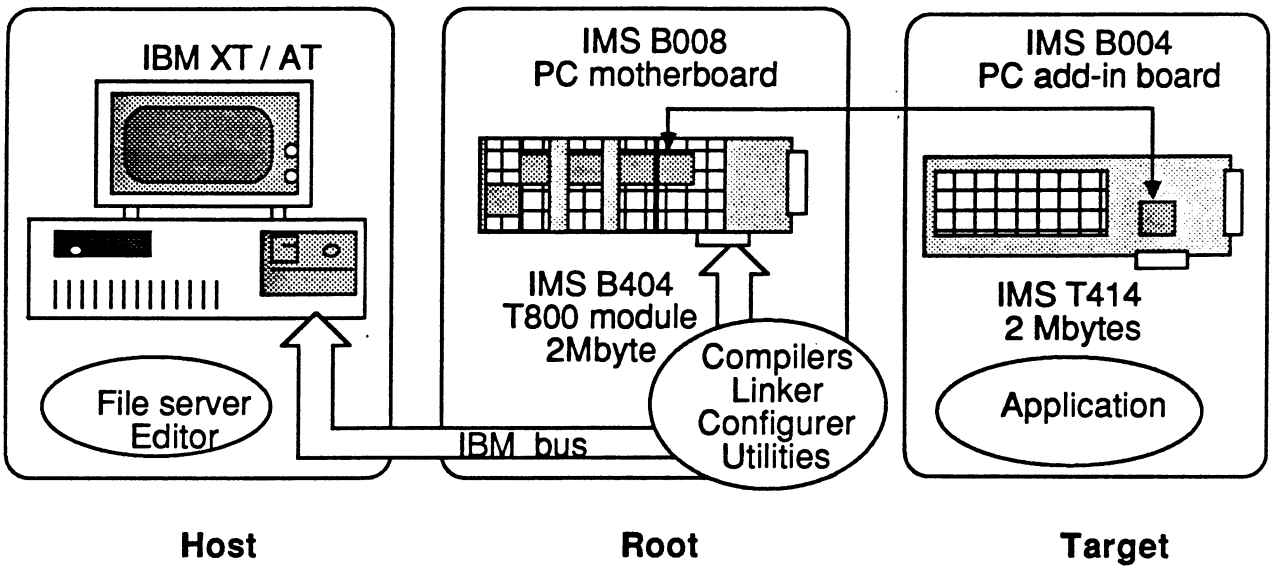
IMS B004 PC add-in board



inmos

Toolset environment

Training course system



Editors

The Toolset is designed for use with your favourite editor,

'make' utility and other software tools.

Training course editors:

Editor	Command
Folding MicroEmacs	fold <filename>
MicroEmacs	emacs <filename>
PCwrite	ed <filename>
Sidekick	Ctrl + Alt
Edlin	edlin <filename>

inmos

Folding MicroEmacs macros

Cursor movement:-

Ctrl←	Word left
Ctrl→	Word right
Home	Top of file
End	Bottom of file
Page Up	Page Up
Page Dn	Page Down

Editing:-

Del	Delete ch forwd
Alt F7	Delete line
F3	Cut line
F4	Copy line
Sh F4	Paste line
Sh F3	Clear paste buffer
F9	Search and Replace

Folds:-

F5	Enter
F6	Exit
Sh F5	Open
Sh F6	Close
F7	Mark
F8	Create fold
Sh F7	Remove fold

Files:-

Alt F5	New file
Alt F6	Toggle buffers
Alt F2	Save
F2	DOS shell
Sh F2	Finish
Sh F9	Run checker
Sh F10	Locate error

inmos

PCwrite

Cursor movement:-

Home	Start of line
End	End of line
Ctrl←	Word left
Ctrl→	Word right
Alt +	Top of file
Alt -	Bottom of file
PgUp	Scroll up
Pg Dn	Scroll down

Editing:-

Del	Delete ch forwd
Ctrl bksp	Delete word left
Ctrl Del	Delete word right
Scroll lock	Toggle insert/overwrite
F3	Mark / copy
F4	Mark / delete
F6	Mark / move
F5	Unmark
F8	Change case

Files:-

F1 F2	Save and exit
F1 F3	Save
F1 F9 F2	Exit without save

Help:-

Esc	F keys
Shift F1	Display F keys
F1 F1	General

Editor practical

Create a directory with DOS command md.

Enter directory with DOS command cd.

Either install your own editor

or familiarise yourself with one of the three editors provided.

inmos

Multi-processor programming

The transputer model

Channels

Parallel programming

Table building exercise

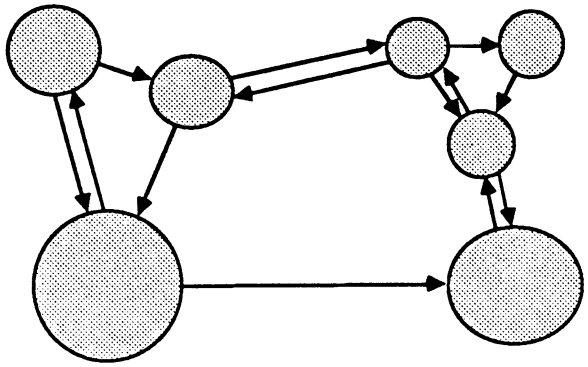
inmos

The transputer model

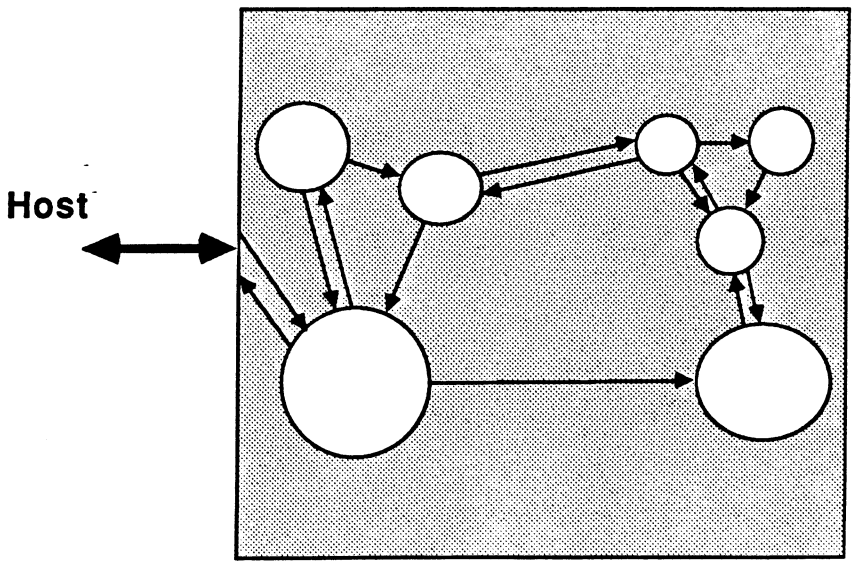
Communicating sequential processes

Parallel tasks run independently.

One way point to point channels.



Development system

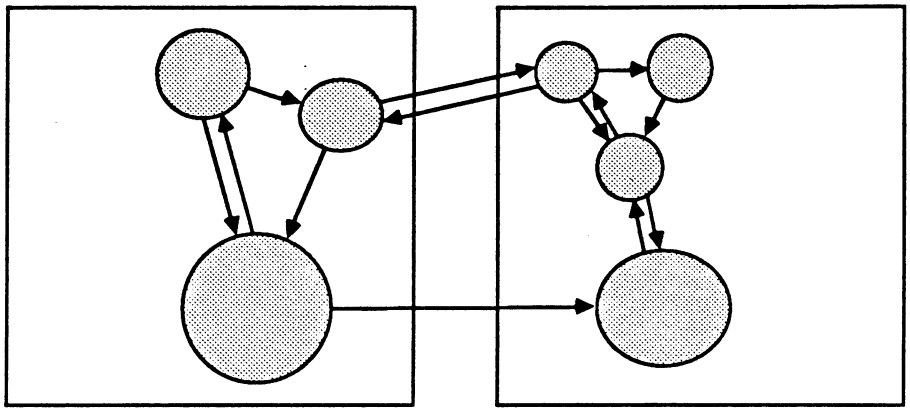


inmos

Multi-transputer system

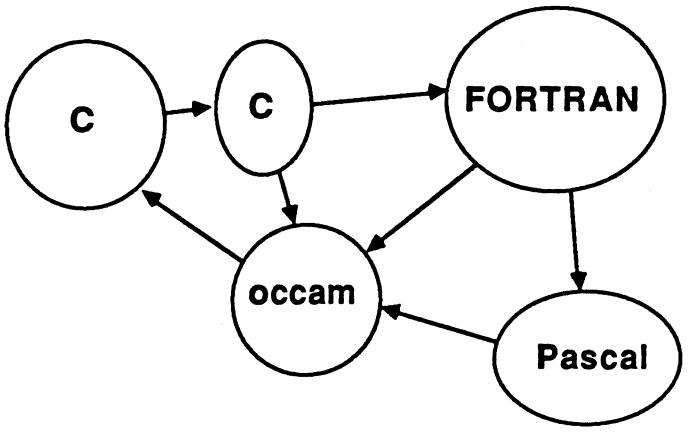
Independent processors connected by two-way links.

Concurrent processing on each processor.



inmos

Mixed languages

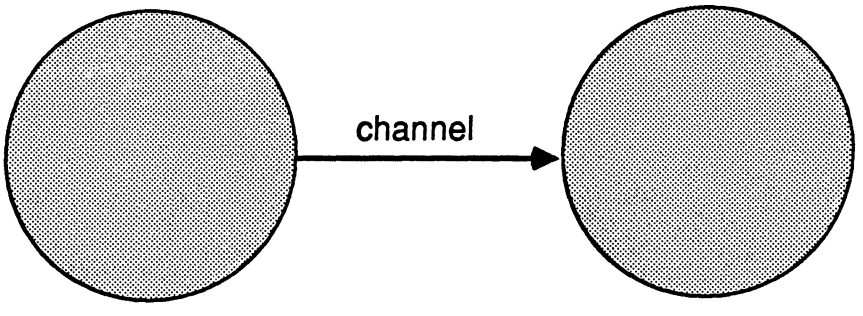


inmos

Channels

inmos

Channels



Point to point

One way

Unbuffered

Synchronised

Control of channels

A channel allows two processes to communicate.

The C programmer has complete control of

input and output.

When two tasks are connected by a channel

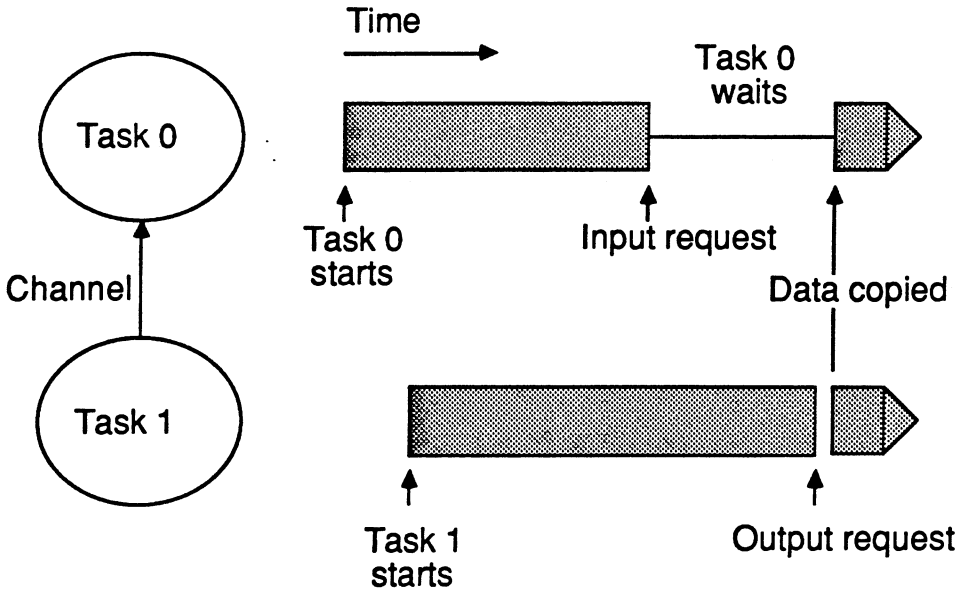
one must output,

the other must input.

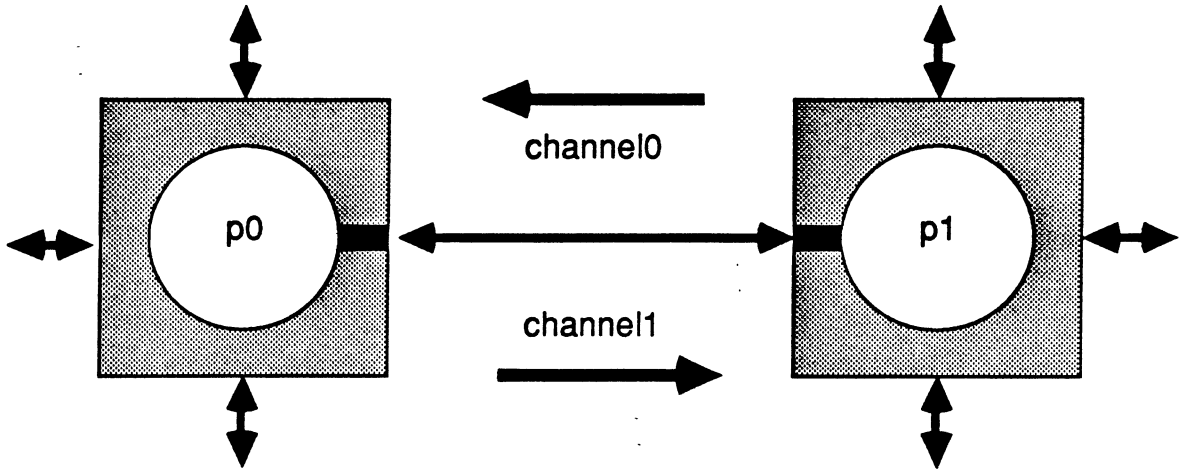
Both tasks must be ready.

The first task ready always waits.

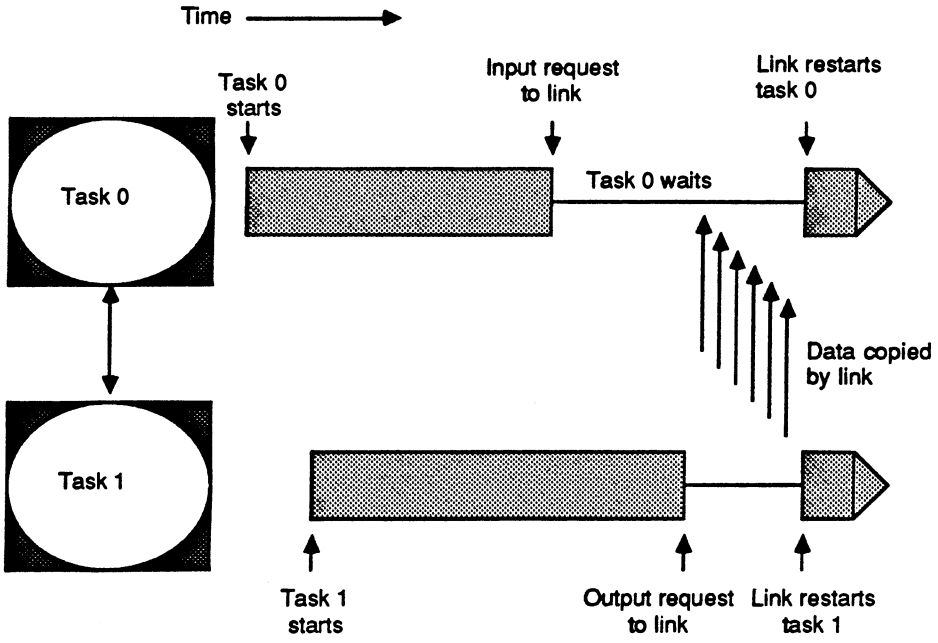
Synchronisation on channels



Links as channels



Link synchronisation

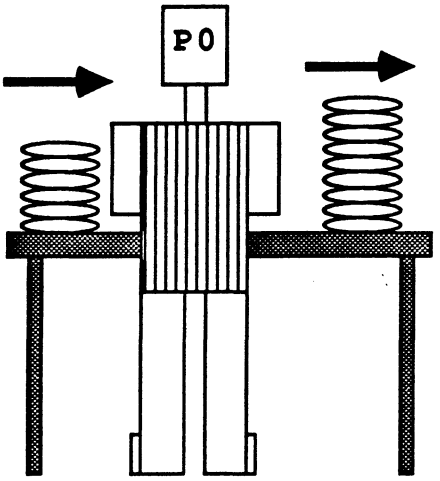


Parallel programming

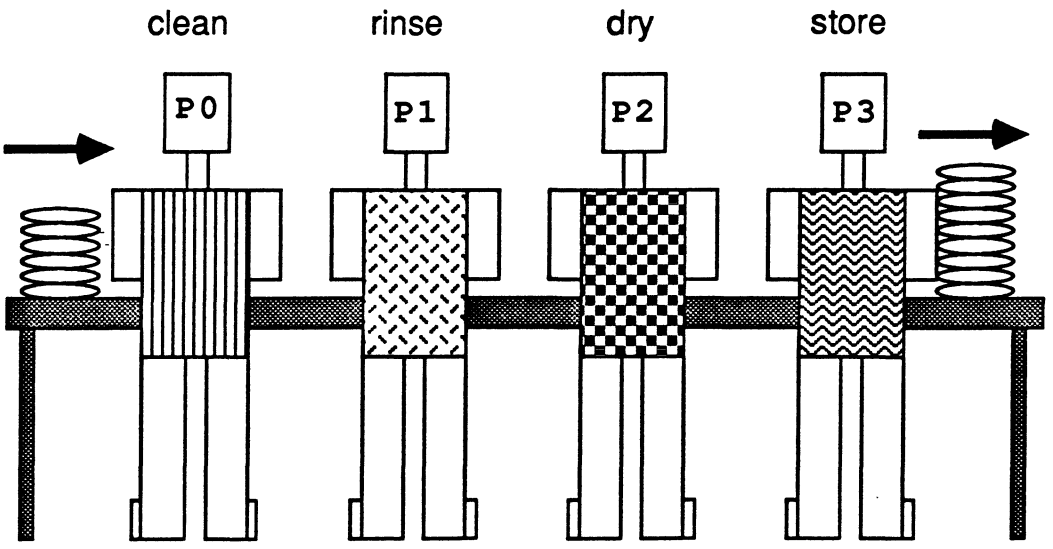
Sequential program


```
wash_dishes ()
{
    for (;;)
    {
       clean_dish ();
       rinse_dish ();
       dry_dish ();
       store_dish ();
    }
}
```

Automatic dishwasher

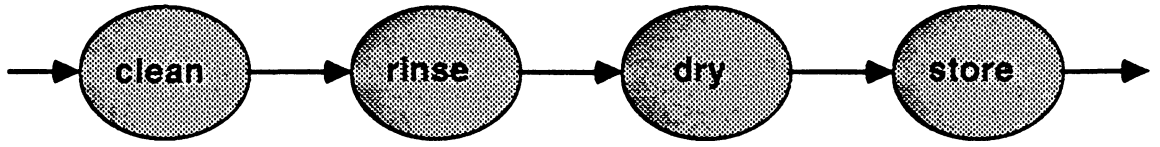


Cooperating processes



Resources are local to each processor.

Parallel tasks in C

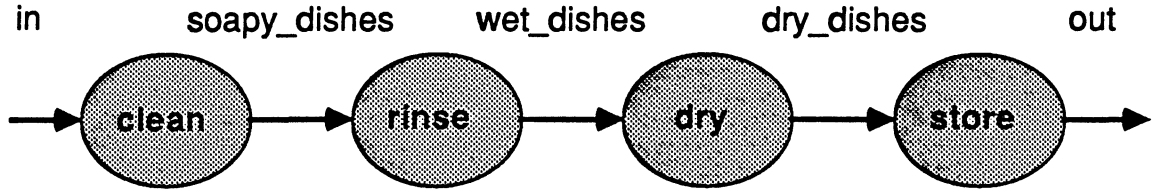


Each task is a complete C program.

Communicating in C

Parallel tasks communicate via channels.

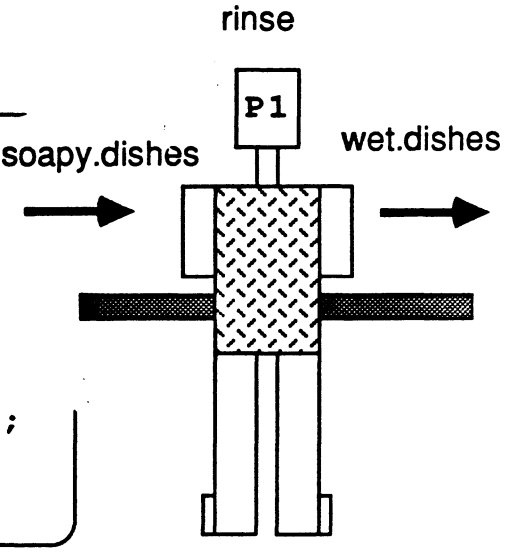
Channels should be given names.



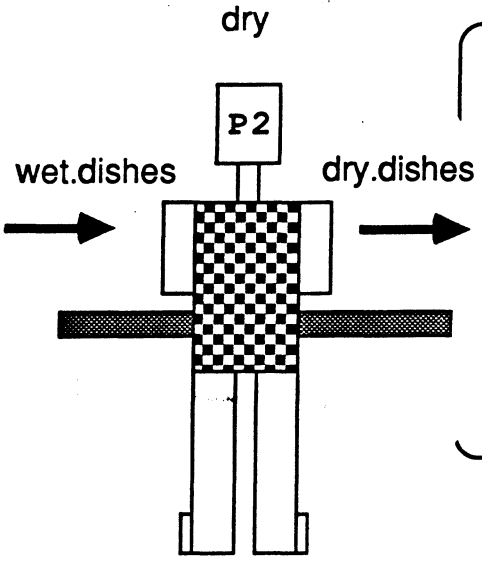
The rinse process

```
main  (...)
...   parameters
{
   int dish;
   for (;;)
    {
       input(soapy_dishes,dish);
       rinse_dish (dish);
       output(soapy_dishes,dish);
    }
}
```

wet



The dry process




```
main  (...)
...   parameters
{
   int dish;
   for (;;)
    {
       input(wet_dishes,dish);
       rinse_dish (dish);
       output(dry_dishes,dish);
    }
}
```

Table building exercise

```
build_table ()
{
    for (;;){
       struct wood table_top, set_of_legs, table;
       set_of_legs = make_set_of_legs ();
       table_top = make_table_top ();
       table = assemble_table(set_of_legs, table_top);
    }
}
```

How do you code a production line

with three people ?

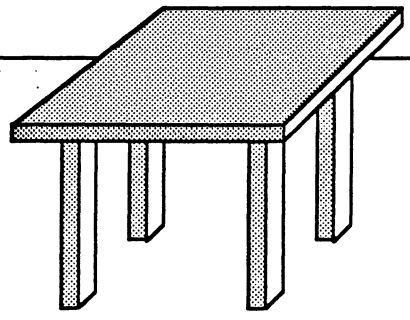
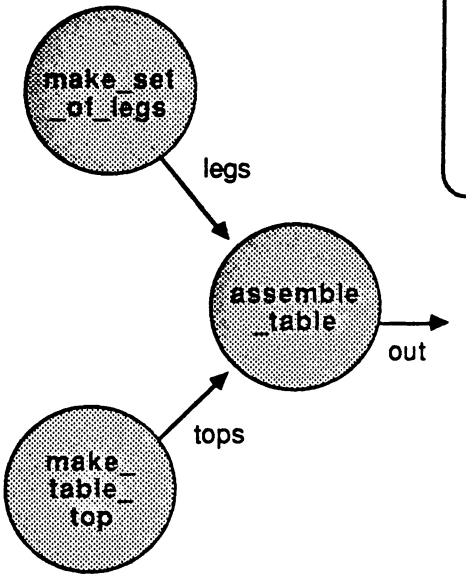


Table building - possible solution page 1




```
main (...)
...  parameters
{
   for (;;)
   {
     struct wood set_of_legs;
     set_of_legs = make_set_of_legs ();
     output (legs, set_of_legs);
   }
}
```

```
main (...)
...  parameters
{
   for (;;)
   {
     struct wood table_top;
     table_top = make_table_top ();
     output (tops, table_top);
   }
}
```


Table building - possible solution page 2

```
main (...)
...  parameters
{
   for (;;){
     struct wood table_top, set_of_legs, table;
     input(legs, set_of_legs);
     input(tops, table_top);
     table = assemble_table (set_of_legs, table_top);
     output (out, table);
   }
}
```

inmos

Compiling C

Overview

Compilers

Linker

Bootstrap tool

Make

Server

Hello World practical

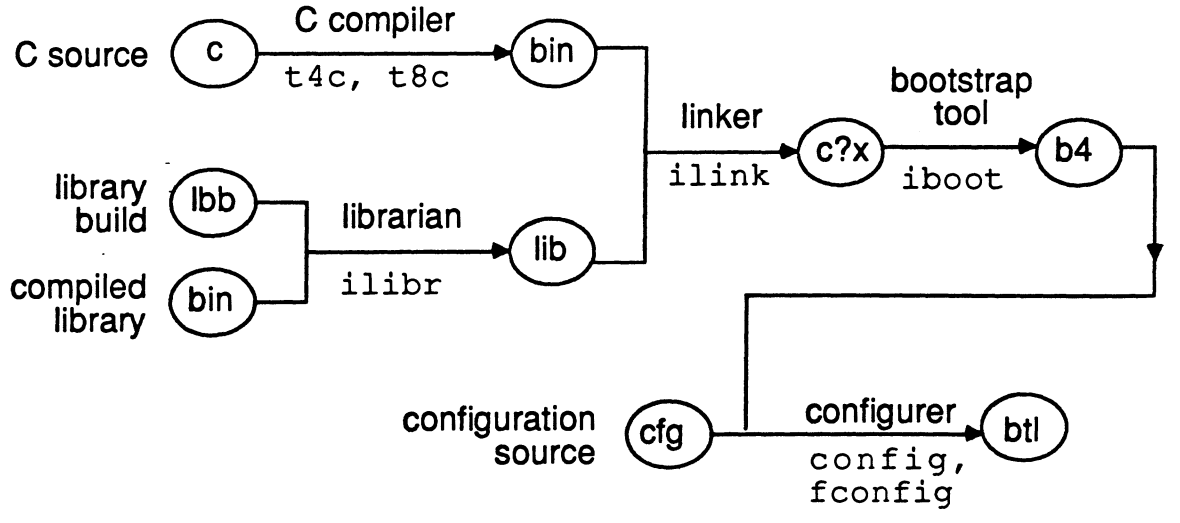
inmos

Overview

Tools

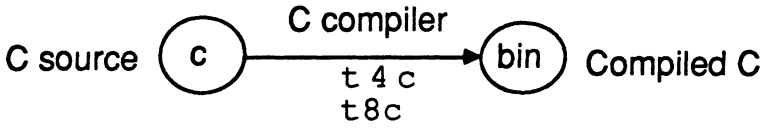
`t4c`	C compiler for IMS T414	`ilibr`	Librarian
`t8c`	C compiler for IMS T800	`iserver`	Server
`ilink`	Linker	`decode`	Binary decoder
`iboot`	Bootstrap tool		
`config`	Configurer		
`fconfig`	Flood configurer		

Compilation steps



Compilers

C compilers




```
t8c myc
```

Compiler switches

C	Check only
FB *filename*	Output file - default *~~cbx~~ bin
FL *filename*	Put listing in file
S	Single length floating point arithmetic
T4 / T8 / T8A	Target transputer type
V	Display progress messages

Indirection

Using PC-DOS hosts, to redirect screen output to a file:

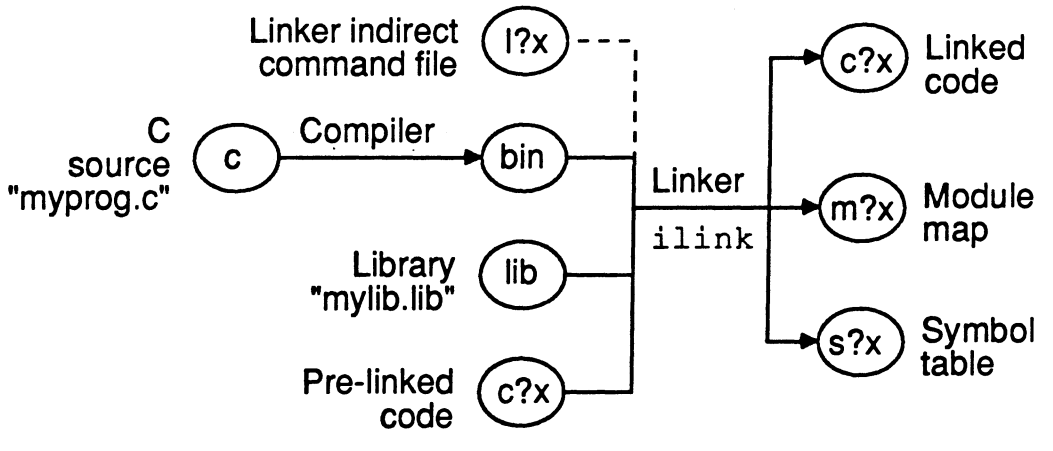
```
t4c myprog > error.dat
```

inmos

Linker

inmos

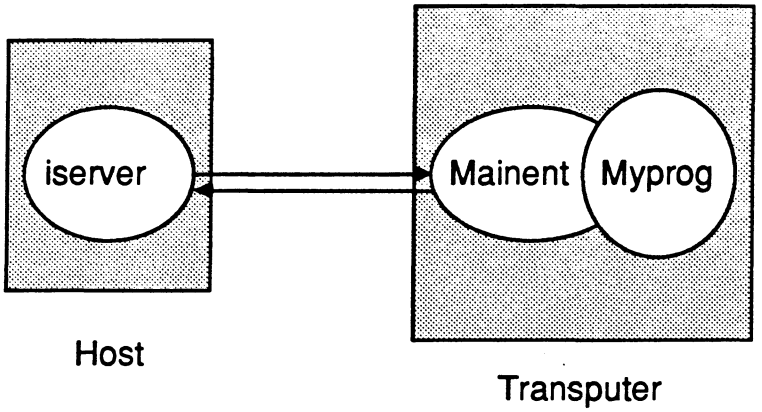
Linking




```
ilink mainent.c8x myprog.bin crtl.lib /o myprog.c8x
```

Output file option is generally necessary.

Mainent



Mainent acts as interface between a single task program and the iserver.

On PC system, mainent.c4x and mainent.c8x are in directory \tc2v0\libs.

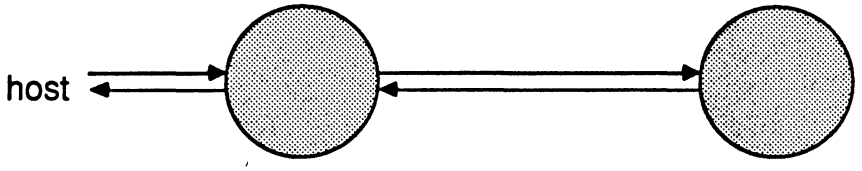
Must be linked first.

Run time libraries

The run time libraries contain all the standard libraries.

Two libraries:

crtl.lib for i/o tasks sacrtl.lib for standalone tasks



On PCs, both libraries in directory \tc2v0\libs.

Indirect linker file

Lists the files to be linked together and options .

Extension l?x.

File: myprog.l8x

```
mainent.c8x
myprog.t8x
crtl.lib
/o myprog.c8x
```

```
ilink /f myprog.l8x
```

is equivalent to:

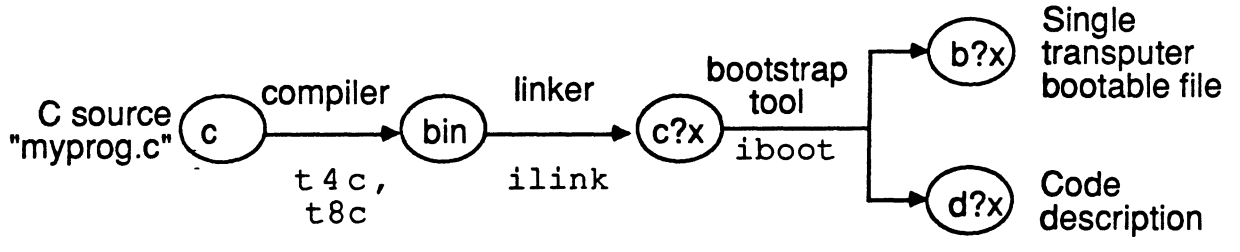
```
ilink mainent.c8x myprog.t8x crtl.lib /o myprog.c8x
```


Linker options

F *filename*	Indirect linker file
O *filename*	Output file - default *.cxx
I	Information
U	Unresolved references allowed
E	Extends linker capacity by making 2 passes
Q *(symbol, ..)*	Quick libraries - code placed low in memory

Bootstrap tool

Single transputer target




```
iboot myprog.c8x
```

The bootstrap tool adds the loading information

Bootstrap options

O *filename*	Output file - default *.bxx
I	Information
S *stacksize*	Stack size in words
C	Task image for configuration

Make

Overview

Make is a standard system program management tool.

A single command compiles, links and configures as needed.

When source is edited, make will recompile etc only the minimum of files.

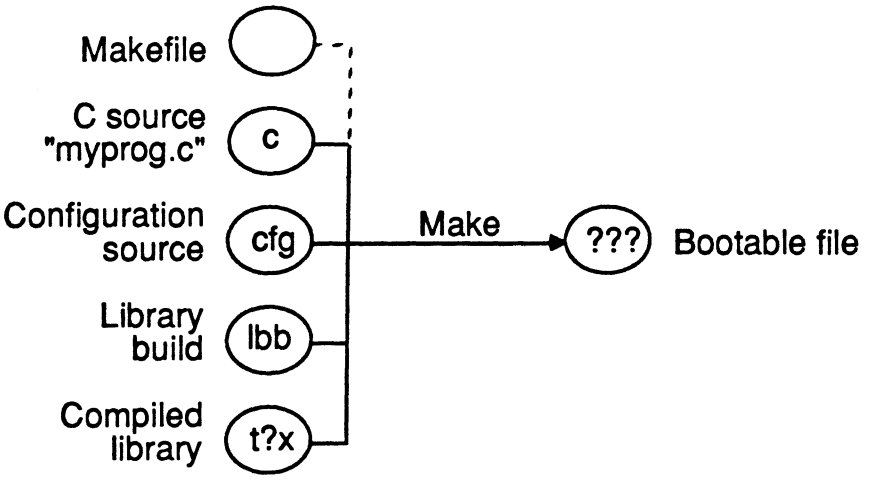
Not supplied with the toolset -

public domain, available freely.

Uses a dependency makefile.

Borland, UNIX and GNU makes supported.

Using make




```
make -f myprog
```

Building makefiles

```
output.fil:◊input1.fil input2.fil
◊toolname param1 param2
```

For each file:

line 1 gives file name and lists files on which it depends;

line 2 gives command line to generate file.

◊ indicates tab character.

Complete makefile

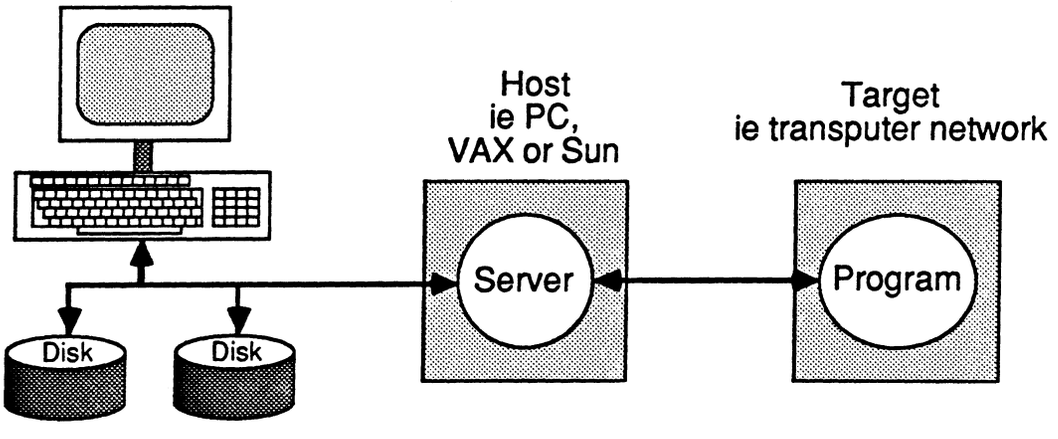
```
myprog.b8x:◊myprog.c8x
◊iboot myprog.c8x

myprog.c8x:◊myprog.l8x myprog.bin
◊ilink /f myprog.l8x

myprog.bin:◊myprog.c
◊t8c myprog
```

The first file to be created is the last in the make file.

Other makes use the opposite order.

Server

Server purpose



Server boots target, loads program

and provides access to host services.

Running a program

To run a program called program.b8x:-

```
iserver /se /sb program.b8x
```

To continue with a program without rebooting:-

```
iserver /se /ss data
```


Server options

SB *filename*	Boot file onto root transputer.
SE	Transputer error flag polled.
SI	Information.
SL *address*	Link address or device name.
SR	Reset root transputer.
SS	Rerun program without rebooting.

Hello World practical

Write code to display a short message on the screen.

Create a directory for your work with the command md.

Enter the directory with the command cd.

Create a subdirectory for this practical.

Build a makefile and an indirect linker file.

Compile your program using make.

Create a batch file to boot the transputer and run a program.

inmos

Channels

Channels

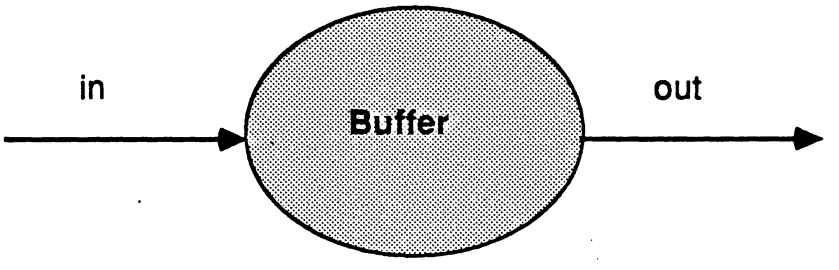
Using channels

Change case practical

inmos

Channels

Buffer



Integer buffer

```
#include <chan.h>

int_buffer (in, out)
   CHAN *in, *out;
{
   int n;
   for (;;)
    {
      chan_in_word (&n, in);
      chan_out_word (n, out);
    }
}
```

Array buffer

```
#include <chan.h>
#define length 80

array_buffer (in, out)
  CHAN *in, *out;
{
  char string[length];
  for (;;)
   {
     chan_in_message (length, string, in);
     chan_out_message (length, string, out);
   }
}
```


Character buffer

```
#include <chan.h>

char_buffer (in, out)
  CHAN *in, *out;
{
  char letter;
  for (;;)
   {
     chan_in_byte (&letter, in);
     chan_out_byte (letter, out);
   }
}
```

Channel i/o functions

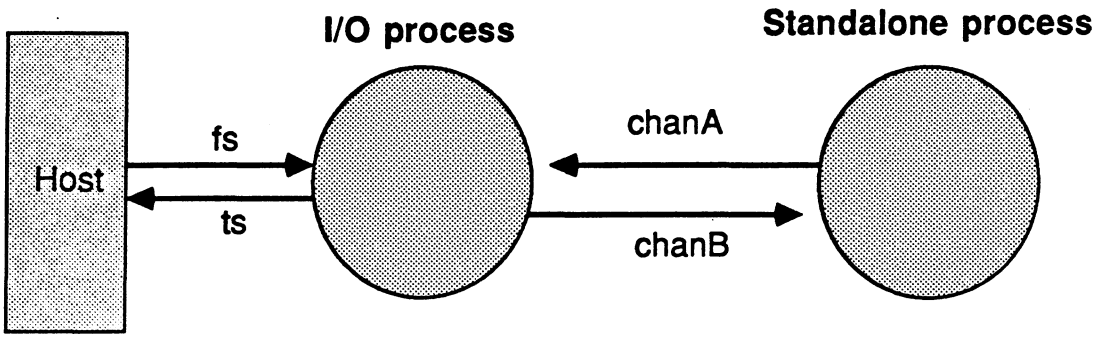
Function	Parameters	Parameter type
chan_in_byte	buffer channel	pointer CHAN pointer
chan_in_word	buffer channel	pointer CHAN pointer
chan_in_message	no of bytes buffer channel	integer pointer CHAN pointer
chan_out_byte	data channel	byte, eg char CHAN pointer
chan_out_word	data channel	word, eg int CHAN pointer
chan_out_message	no of bytes buffer channel	integer pointer CHAN pointer

Parameters of main

Fixed parameters give access to channels.

```
#include <chan.h>

main (argc, argv, envp, in_ports, inlen, out_ports, outlen)
   int argc;                          /* no of tokens in argv */
   char *argv[];                      /* token strings */
   char *envp[];                      /* NULL */
   CHAN *in_ports[], *out_ports[]; /* channel pointer arrays */
   int inlen, outlen;                 /* sizes of
                                         channel pointer arrays */

{
   int value, i;
   chan_out_word (value, out_ports[i]);
}_
```

Multi-task program



Reserved channels

	IO process	Standalone process
in_ports[0]	unused	unused
out_ports[0]	debug	debug
in_ports[1]	standard input	
out_ports[1]	standard output	

Using channels

Communication security

Unmatched communications cause deadlock.

Only occam checks communications.

In C, the programmer is responsible.

Use good programming practice to avoid errors:

- draw data flow diagrams;
- put input and output in library subprograms;
- use meaningful names.

Variable size arrays

```
in_array (message, len, in)
   char message[];
   int len;
   CHAN *out;
{
   chan_in_word(len, in);
   chan_in_message(len, message, in);
}

out_array (message, len, out)
   char message[];
   int len;
   CHAN *out;
{
   chan_out_word(len, out);
   chan_out_message(len, message, out);
}
```


Structures

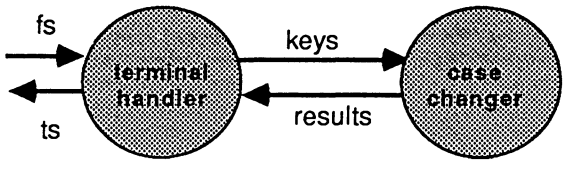
```
struct record {int n; float x; char ch};

struct record in_record(in)
   CHAN *in;
{
   struct record r;
   chan_in_message (sizeof(r), &r, in);
   return(r);
}

out_record (r, out)
   struct record r;
   CHAN *out;
{
   chan_out_message (sizeof(r), &r, out);
}
```

Unions

```
union quantity {int no; double weight;};


out_quantity (q, out)
   union quantity q;
   CHAN *out;
{
   chan_out_message (sizeof(q), &q, out);
}


in_quantity (q, in)
   union quantity q;
   CHAN *in;
{
   chan_in_message (sizeof(q), &q, in);
}
```


Change case practical

Write a case changing program consisting of two tasks.



The terminal handler alternately reads the keyboard and the results channel. It passes on all characters from the keyboard to the case changer. When it receives any character from the results channel it displays it on the screen. When it receives a '%' character it terminates the server and itself and passes on the terminate message to the case changer.

The case changer process changes upper case letters to lower case and lower case to upper and sends the result back down the results channel. Any other characters are returned unchanged. When it receives a terminate message it terminates itself.

Compile all the code.

inmos

Multi-task programs

Configuration

Task attributes

The configurer

Configuration practical

inmos

Configuration

Tasks

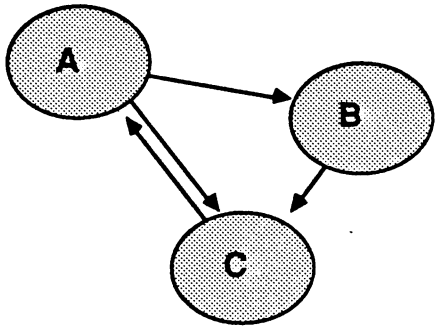
Separate C programs.

Communication by channels.

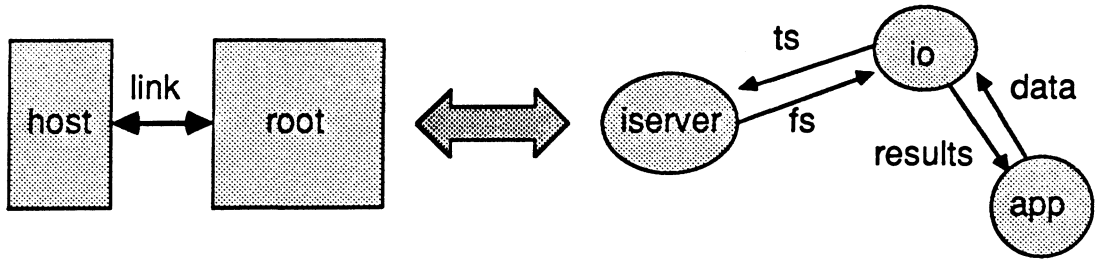
Joined by the configurer.

Separate data space, code and libraries.

All data initialised.



Configuration purpose



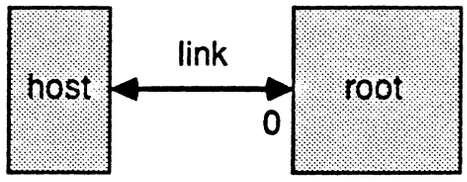
Describe hardware.

Describe tasks.

Connect tasks.

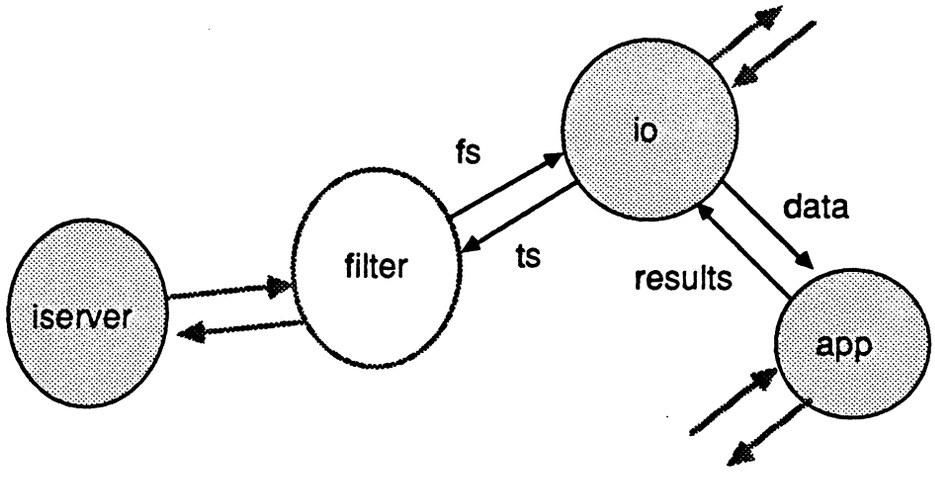
Map software onto hardware.

Describe hardware



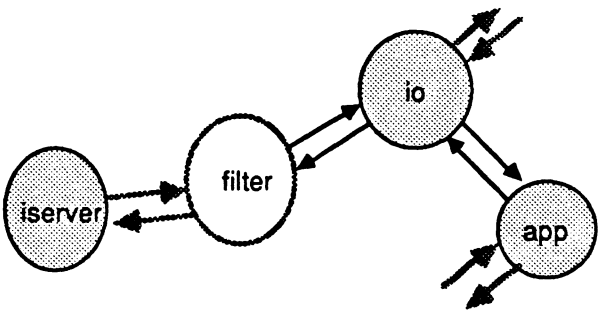

```
processor  host
processor  root

wire  link  root[0]  host[0]
      !host connected to root link 0
```


Hidden software



Filter is a standard task provided with Parallel C.

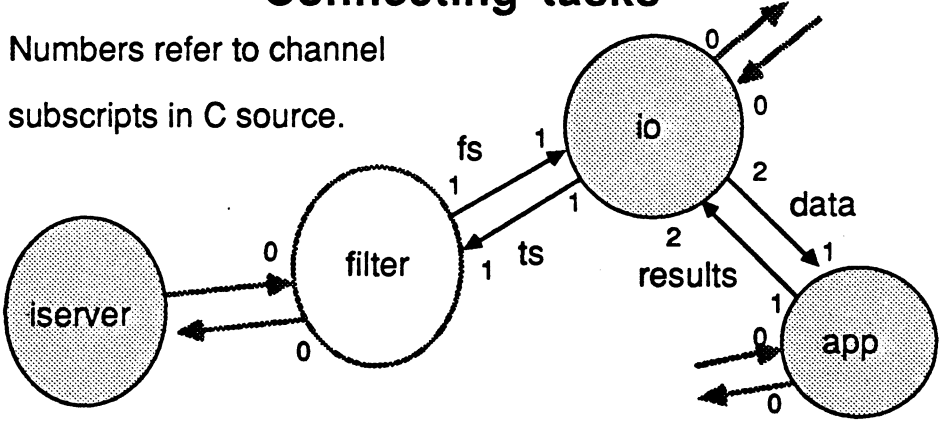
Describe tasks




```
task io        ins=3 outs=3
task app      ins=2 outs=2 data=10k
task filter   ins=2 outs=2 data=10k
task iserver ins=1 outs=1
```

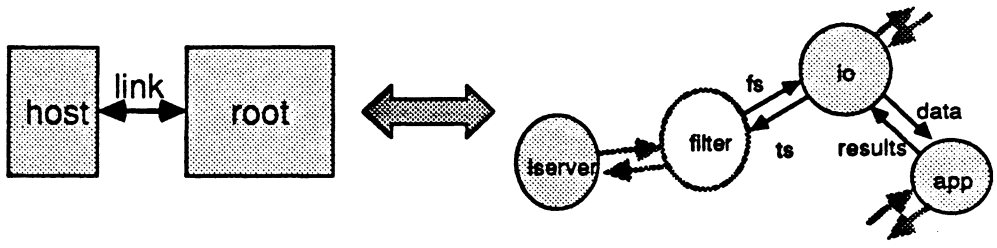

Connecting tasks

Numbers refer to channel subscripts in C source.




```
connect ?          filter[0]   iserver[0]
connect ?          iserver[0]  filter[0]

connect fs         filter[1]   io[1]
connect ts         io[1]       filter[1]

connect data      io[2]       app[1]
connect results   app[1]      io[1]
```

Mapping




```
place iserver host

place io       root
place app      root
place filter   root
```


Complete configuration

```
processor host
processor root

wire link root[0] host[0]

task io       ins=3 outs=3
task app      ins=2 outs=2 data=10k
task filter   ins=2 outs=2 data=10k
task iserver  ins=1 outs=1

place iserver host

place io       root
place app      root
place filter   root

connect ?         filter[0]   iserver[0]
connect ?         iserver[0]  filter[0]

connect fs        filter[1]   io[1]
connect ts        io[1]       filter[1]

connect data      io[2]       app[1]
connect results   app[1]      io[2]
```

inmos

Task attributes

inmos

Channels

Number of input and output channels

defined by INS and OUTS respectively.

Includes hidden channels.

Must be given for all tasks.

```
task io       ins=3 outs=3
task app      ins=2 outs=2 data=10k
task filter   ins=2 outs=2 data=10k
task iserver ins=1 outs=1
```

Task file

Default filename for linked code is

task name with extension .b4

Use file attribute to override default.

```
task io  ins=3 outs=3 file=termhand.b8x
task app ins=2 outs=2 data=10k
```


Priority

Transputer has two levels of priority:

NOTURGENT and URGENT.

Default is NOTURGENT.

```
task io  ins=3 outs=3 urgent
task app ins=2 outs=2 data=10k
```

Memory allocation

Either total data space is declared in bytes

or stack and heap (static) separately.

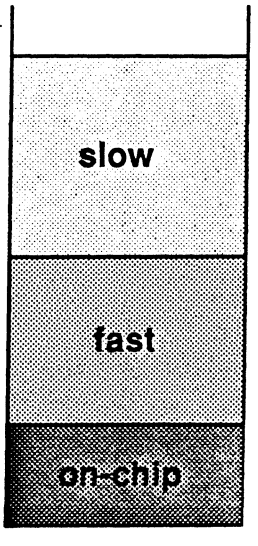
Scaling factors k and M may be used.

Minimum stack 1k, minimum heap 5k.

At most one task per processor may have free memory.

```
task io    ins=3 outs=3
task app1  ins=3 outs=3 data=10k
task app2  ins=2 outs=2 -
                stack=1k heap=10k
```


On-chip memory



For maximum performance

place critical data / code

in fastest memory.

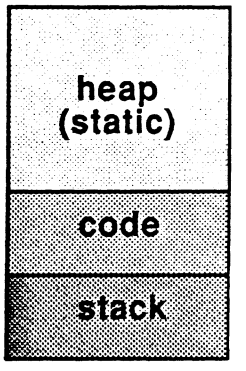
3 cycle read / write
(2 cycle on IMS T801)
33 Mbytes/sec

4 kbyte
single cycle read / write
100 Mbytes/sec

Using on-chip memory

Stack, code or heap may be placed on chip using OPT attribute.

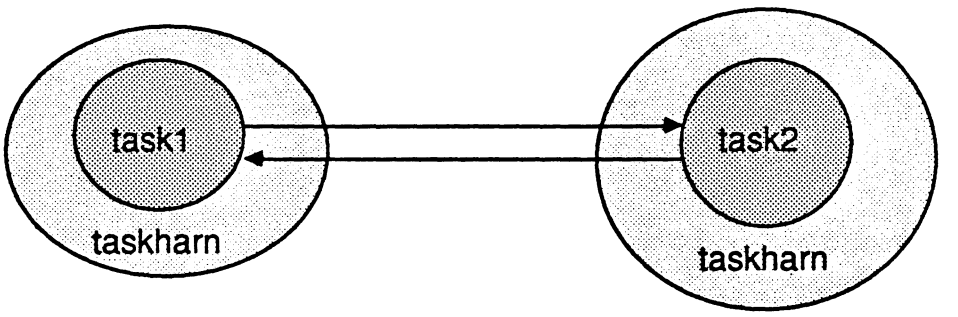
The linker places code in the order of linking, starting on fastest memory.




```
task io    ins=3 outs=3
task app1  ins=2 outs=2 data=10k
task app2  ins=2 outs=2 stack=1k -
      heap=10k opt=stack opt=code
```


The configurer

Linking and booting



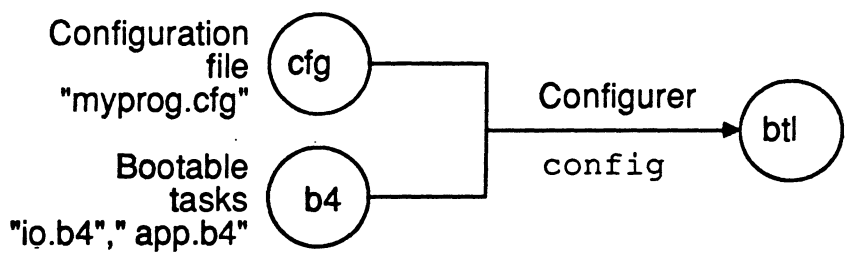
Each user task must be linked with taskharn.

```
ilink taskharn.t8x task1.bin sacrtl.lib /o task1.c8x
```

Add bootstrap to each task with /c option and output file *.b4.

```
iboot task1.c8x /o task1.b4 /c
```


Configuring



Any number of tasks on each processor.

```
t8c  io
ilink taskharn.t8x io.bin crtl.lib /o io.c8x
iboot io.c8x /o io.b4 /c
t8c  app
ilink taskharn.t8x app.bin sacrtl.lib /o app.c8x
iboot app.c8x /o app.b4 /c
config myprog.cfg myprog.btl
```

Makefile

```
myprog.btl:◊io.b4 app.b4 myprog.cfg
◊config myprog.cfg myprog.btl

io.b4:◊io.c8x
◊iboot io.c8x /c /o io.b4

io.c8x:◊io.l8x io.bin
◊ilink /f io.l8x

io.bin:◊io.c
◊t8c io

app.b4:◊app.c8x
◊iboot app.c8x /c /o app.b4

app.c8x:◊app.l8x app.bin
◊ilink /f app.l8x

app.bin:◊app.c
◊t8c app
```


Indirect linker files

File: io.l8x

```
taskharn.t8x
io.bin
crtl.lib
/o io.c8x
```

File: app.l8x

```
taskharn.t8x
app.bin
sacrtl.lib
/o app.c8x
```

Change case practical - part 2

Write a configuration file for your change case code.

Write a makefile and indirect linker files.

Compile the code and run it.

inmos

Multi-transputer systems

Board connections

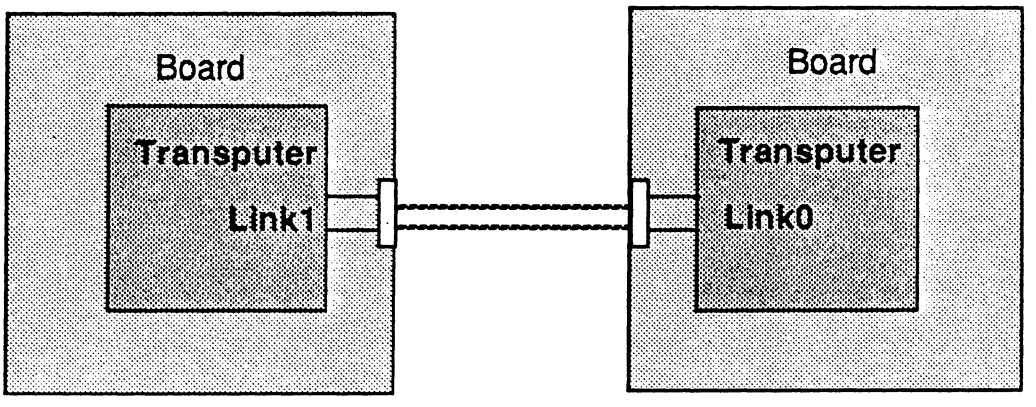
Course system

Board wiring practical

inmos

Board connections

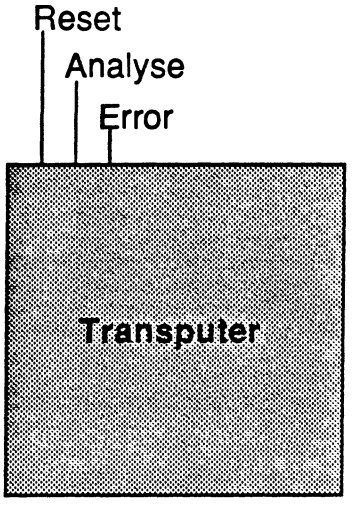
Links



Link cables are multicoloured pairs of twisted pairs.

End plugs have pin 4 blanked off.

Reset pins



Reset pin to initialise.

Analyse pin to debug

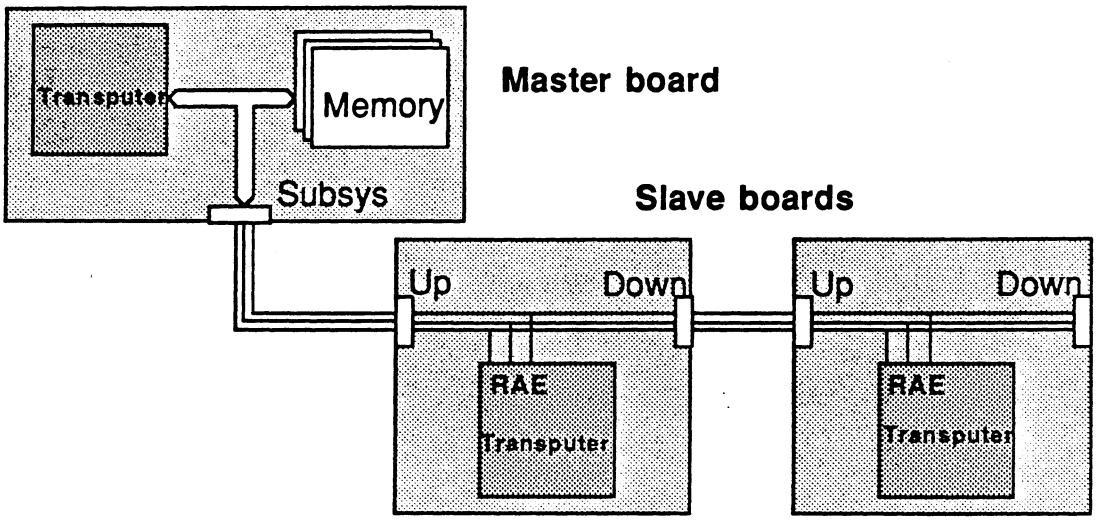
Error pin to detect error.

Connection to these three pins

allows control of the transputer.

inmos

Reset ports

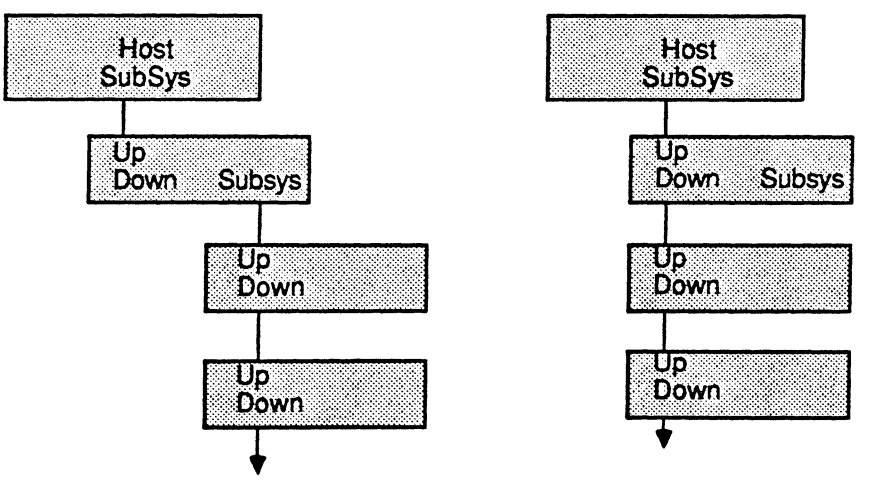


Up/Down propagates reset, analyse and error by hardware.

SubSys is a latch accessed by software.

inmos

INMOS board system control



Reset cables are grey or black three core.

Up end plugs have pin 5 blanked off.

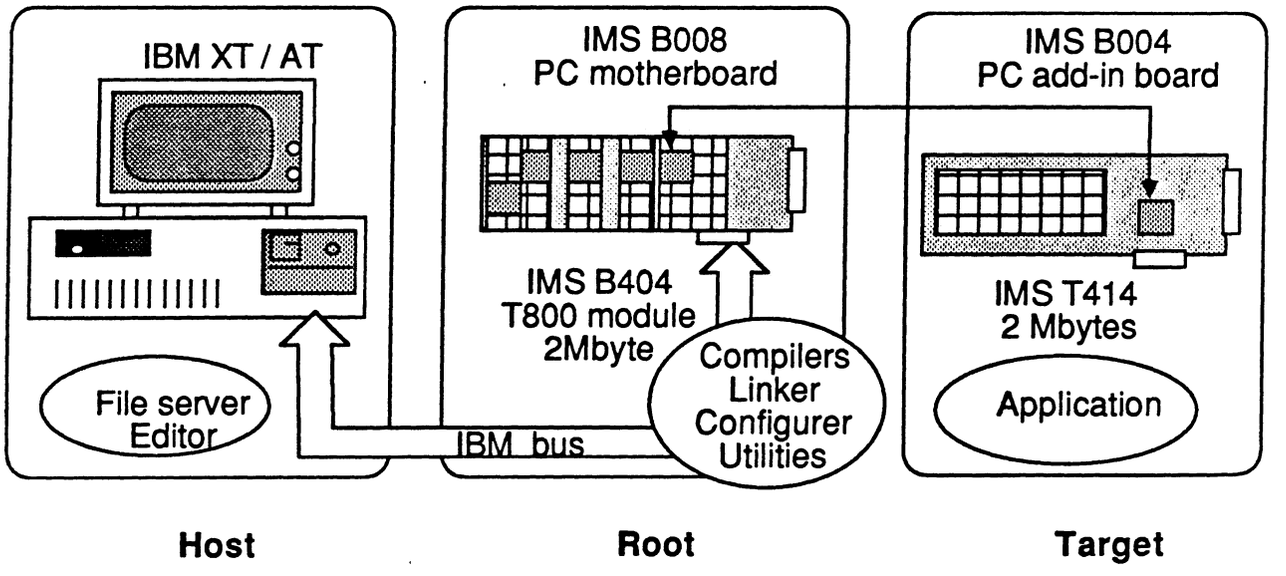
Down and Subsys end plugs have pins 4 and 5 blanked off.

inmos

Course system

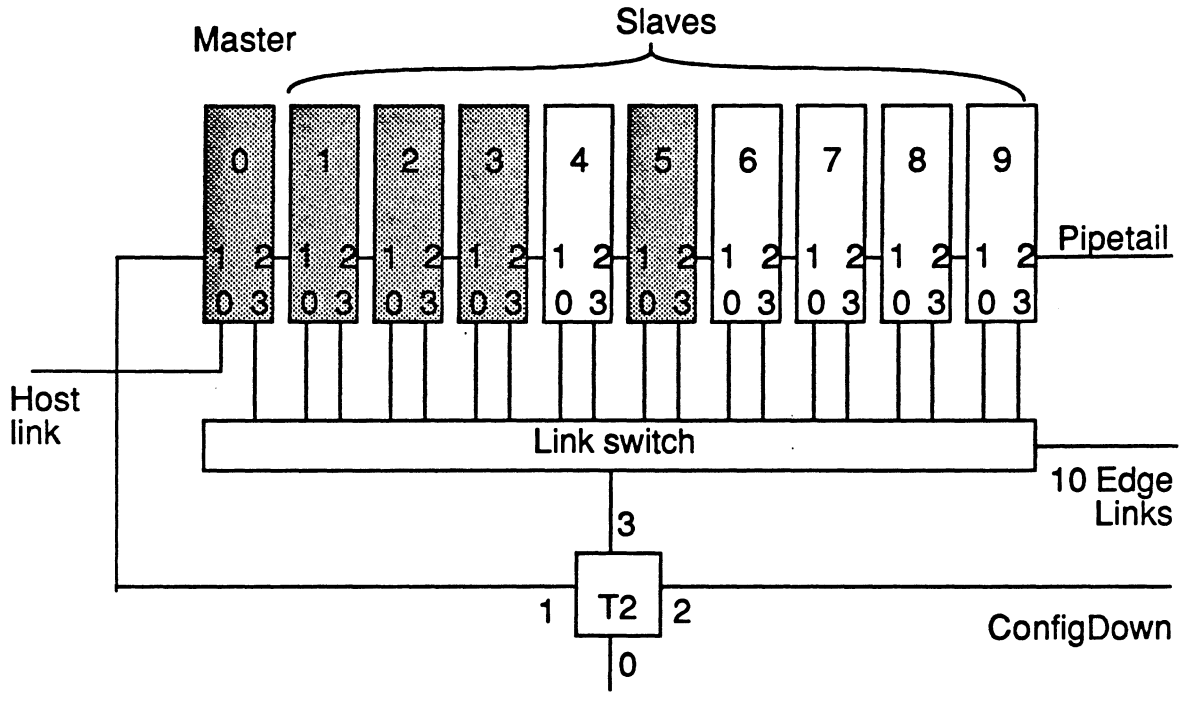
inmos

Training course system



inmos

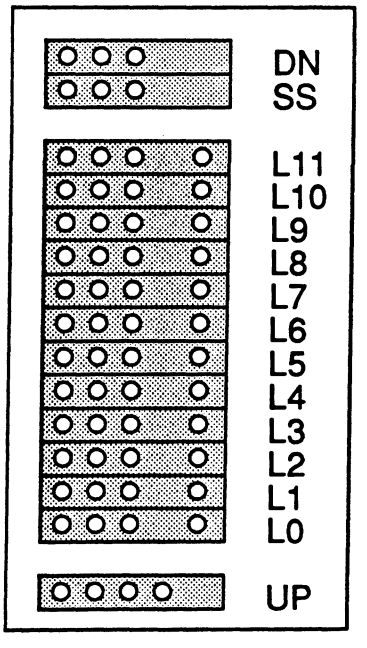
IMS B008 PC motherboard



SGS-THOMSON MICROELECTRONICS

inmos

PC motherboard break-out board pinout



Down
Subsystem

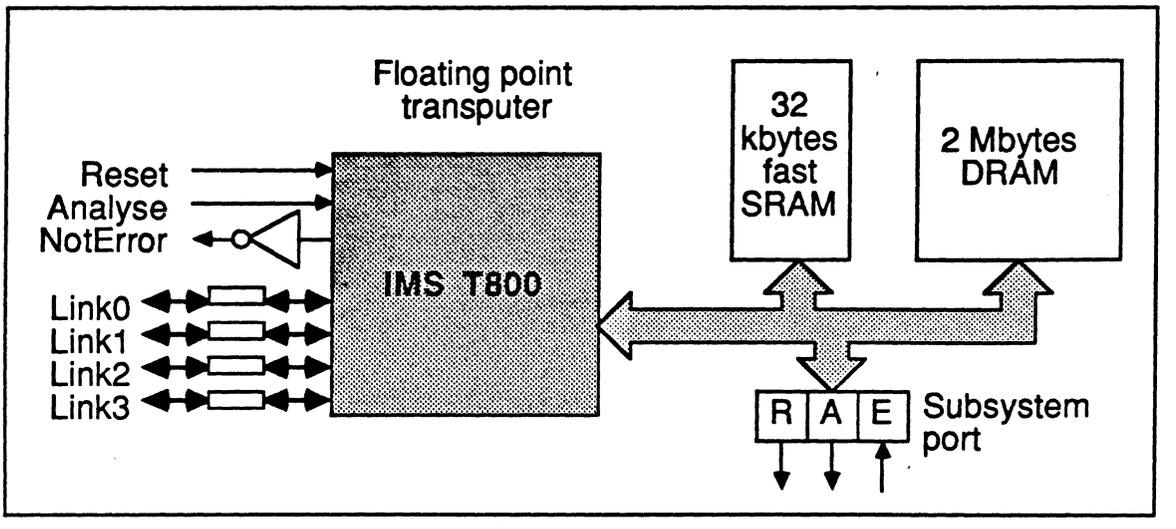
ConfigDown
PipeTail
Edge Link 9 (= Patch link 1)
Edge Link 8 (= Patch link 0)
Edge Link 7
Edge Link 6
Edge Link 5
Edge Link 4
Edge Link 3
Edge Link 2
Edge Link 1
Edge Link 0

Up

SGS-THOMSON MICROELECTRONICS

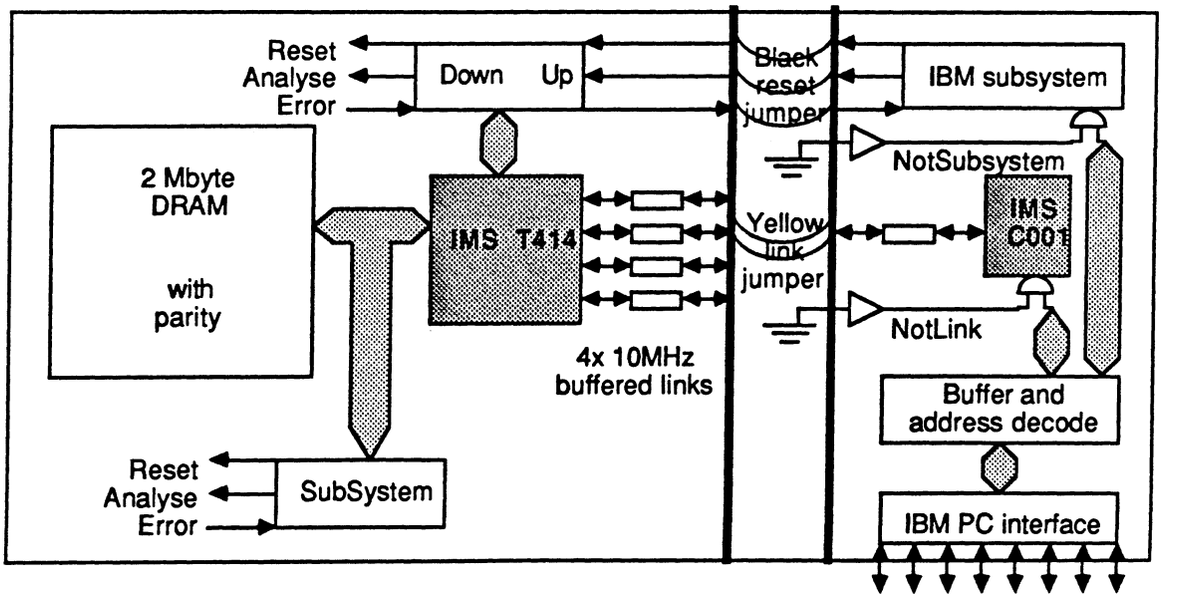
inmos

IMS B404 2Mbyte transputer module

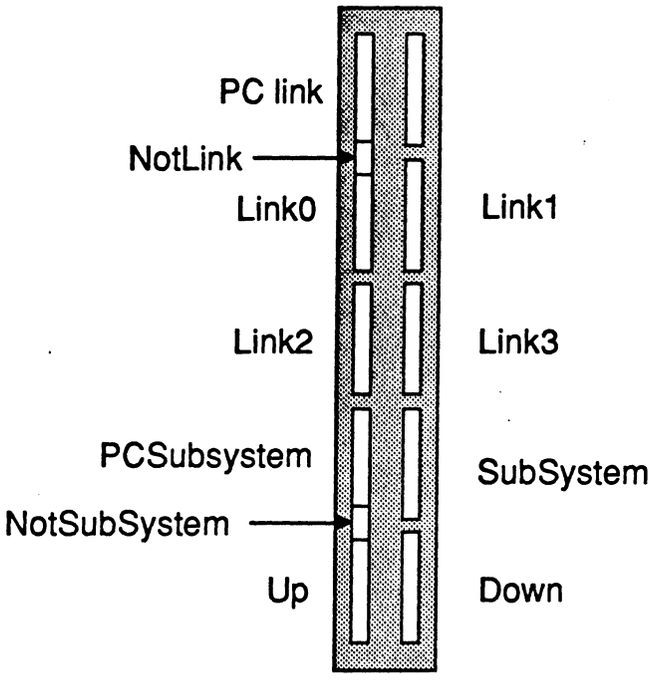


inmos

IMS B004 PC add-in board



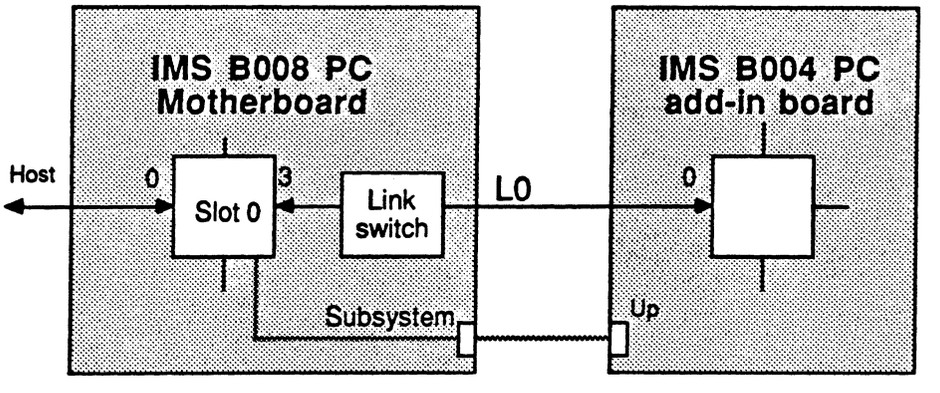
PC add-in board pinout



Change case practical - part 3

Run your change case code on two transputers.

Connect the PC motherboard to the PC add-in board as shown:



Use setc004 to set the link switch. Check the wiring with check.
Copy and change your configuration file to two transputers.
Change your makefile to compile the case changer for an IMS T414. Make and run your code.

inmos

Priority and time

SGS-THOMSON MICROELECTRONICS

inmos

Priority

SGS-THOMSON MICROELECTRONICS

Priority

The transputer has two levels of priority:

0 high priority

- no timeslicing.

1 low priority (default)

- possible timeslice after 1 to 2 msec.
- possible interrupt by high priority.

Priority can affect performance.

Bit zero of the workspace pointer shows priority.

Defining priority

Priority of a task is fixed at configuration time.

Low priority is default.

High priority is URGENT.

```
task app ins=2 outs=2
task io  ins=4 outs=4 data=10k urgent
```

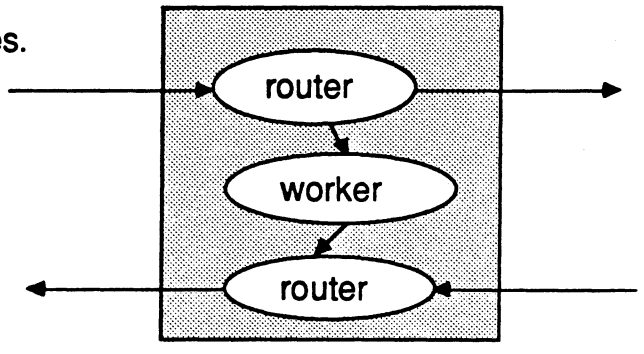
Uses of priority

Use high priority for:

handling link communications;

handling interrupts;

timing fast processes.



Timers

Transputer time

Two registers, clock0 and clock1.

Incremented regularly ± 200ppm.

Used for:

- benchmarking;
- delays;
- timeouts;
- timed events.

Clock rates

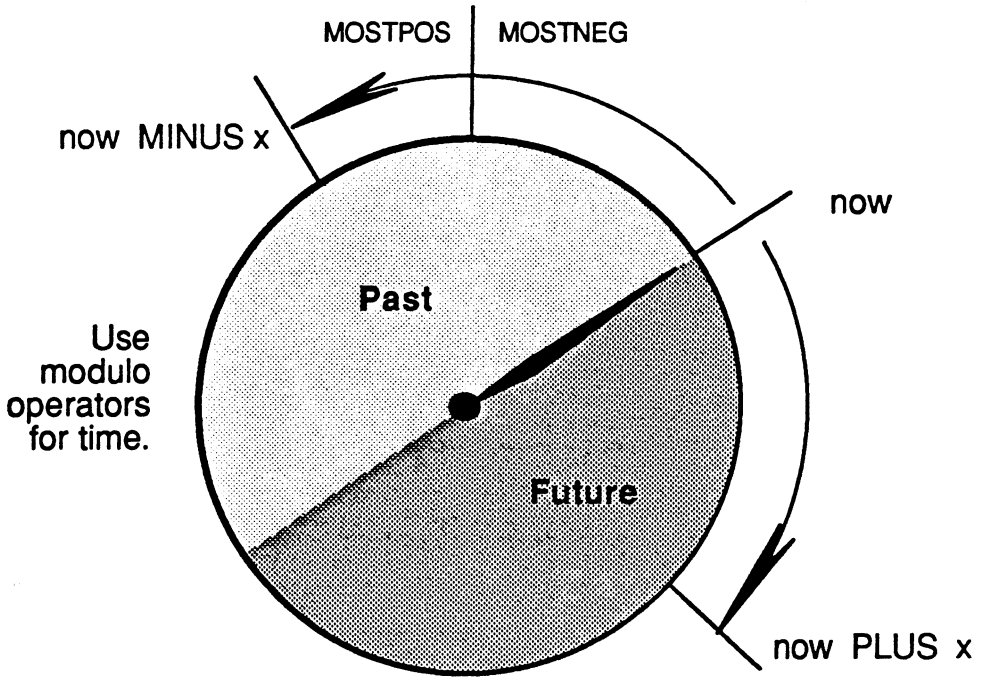
priority	low	high
time between ticks	64 microsecs	1 microsec
ticks per second	15625	1000000
approx time between resets - 16 bit	4 secs	1/16 sec
approx time between resets - 32 bit	76 hours	1 hour

Timing a process

Timer_now gives current time in ticks.

```
#include <timer.h>

benchmark()
{
    int start, end;
    start = timer_now ();
    marathon_process ();
    end = timer_now ();
}
```


Time arithmetic



Comparing times

Use timer_after to compare times.

```
#define FALSE 0
#define RUN_TIME 1000
#include <timer.h>

timed_out_loop()
{
    int now, stop_time;
    now = timer_now();
    stop_time = timer_plus(now, RUN_TIME);
    while (timer_after(stop_time, now) != FALSE)
    {
       main_routine();
       now = timer_now();
    }
}
```


Modulo arithmetic

Use t-code diff, sum and times.

```
int timer_minus (x, y)
    int x, y;
{
    int z;
    asm {ldl x; ldl y; diff; stl z;}
    return (z);
}

int timer_plus (x, y)
    int x, y;
{
    int z;
    asm {ldl x; ldl y; sum; stl z;}
    return (z);
}
```

Timer delay

Use timer_delay.

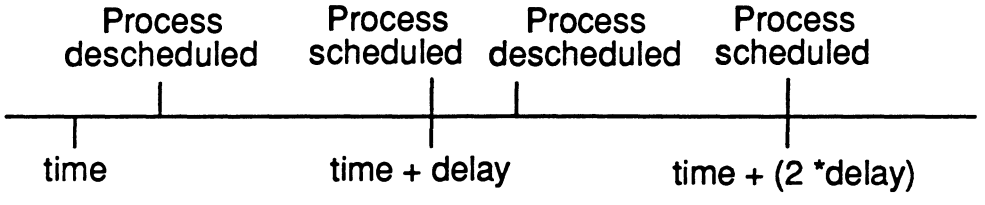
Current task or thread is descheduled

and uses no processor time.

Delay is in ticks.

```
#include <timer.h>

reset_subsystem()
{
    analyse_signal (0);
    reset_signal (1);
    timer_delay (1);
    reset_signal (0);
    timer_delay (1);
}
```


Regular pulses




```
#include <timer.h>
ticker (delay, out)
    int delay;
    CHAN *out;
{
    int time;
    time = timer_now();
    for (;;)
    {
       time = timer_plus (time, delay);
       timer_wait (time);
       chan_out_word (0, out);
    }
}
```

Timer functions

```
#include <timer.h>
```

`timer_now ()`	read current timer value.
`timer_wait (wake_time)`	deschedule until wake_time.
`timer_delay (delay)`	deschedule for delay ticks.
`timer_after (t1, t2)`	non-zero if t1 after t2, zero otherwise.

Channel input with timeout

#include <chan.h>

Function	Parameters	Parameter type
chan_in_byte_t	buffer channel timeout	pointer CHAN pointer int
chan_in_word_t	buffer channel timeout	pointer CHAN pointer int
chan_in_message_t	no of bytes buffer channel timeout	int pointer CHAN pointer int

Similarly output.

Timeout is in timer ticks.

Host time

#include <time.h>

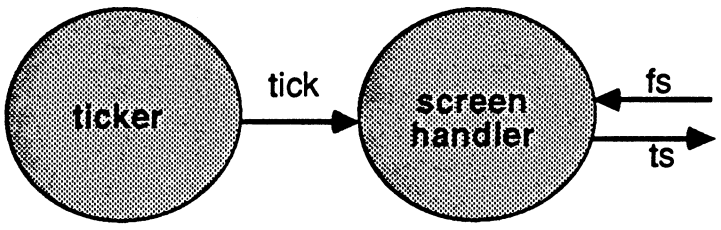
	Params	Result Type	Result
clock	none	int	Elapsed time in seconds
time	none	int	Elapsed time in seconds since 00:00 GMT 1/1/1970

Provided only in i/o run-time library.

Clock practical

Write a process to display a clock that starts from zero and runs forever.

Write two parallel processes as shown.



Ticker sends a signal every second.

The screen handler stores the time, incremented every second,

and displays it.

Threads

Overview

Thread functions

Semaphores

Multiplexor

Thread overview

Parallel code within a task.

Used for parallel output and input in a single task.

Must run on the same transputer.

Urgent or not urgent priority.

Shared static, extern and heap data.

Each thread has own stack for auto variables.

Use semaphores for allocation of shared resources.

Extensive library support.

Thread functions

Starting a simple thread

```
#include <thread.h>

thread_create (fun, ws_size,
               n, arg1, arg2, ..., argn);
   void (*function)();
   int ws_size;
  int n, arg1, arg2, ..., argn;
```

The thread will execute the function fun with:

arguments arg1, arg2, ..., argn;

workspace of size ws_size bytes from the heap;

current priority.

Starting a general thread

```
#include <thread.h>

thread_start (fun, worksp, ws_size,
              priority,
              n, arg1, arg2, ..., argn);
   void (*function)();
   char *worksp;
   int ws_size;
  int n, arg1, arg2, ..., argn;
```

The thread will execute the function fun with:

arguments arg1, arg2, ..., argn;

workspace worksp of size ws_size bytes;

priority THREAD_URGENT or THREAD_NOTURG.

Terminating a thread

Every thread created remains as a parallel process until:

the function it executes returns or

the function executes thread_stop.

```
#include <thread.h>
thread_stop ();
```

Other thread functions

`thread_priority`	Returns priority.
`thread_deschedule`	Deschedule briefly.
`thread_restart`	Restart after resetting channel.

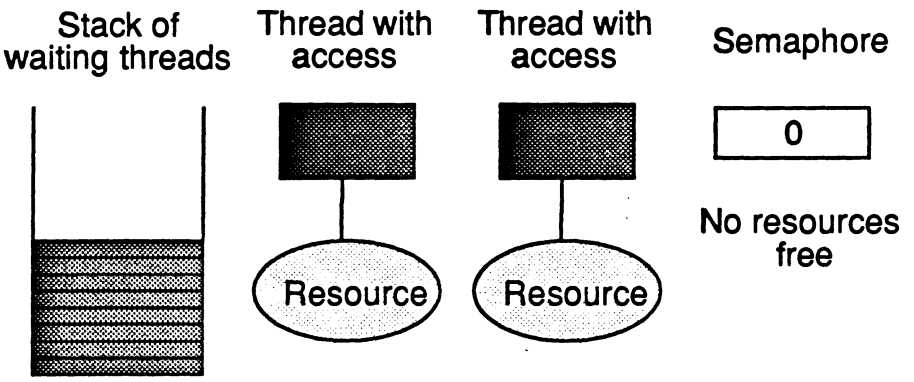
Semaphores

Purpose

To share resources, eg variables or channels,

between threads of the same priority.

Semaphore holds number of resources free.

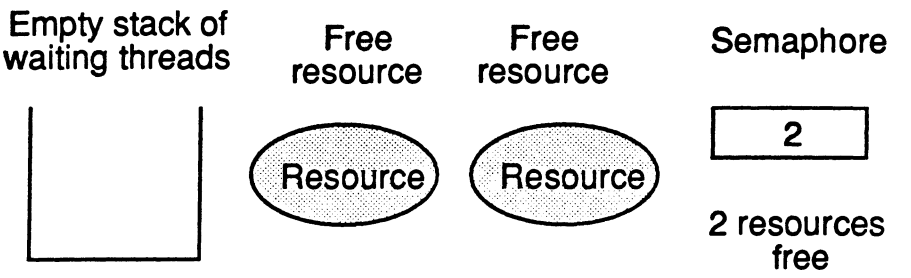


Setting up a semaphore

```
#include <sema.h>
SEMA *resource;
sema_init (resource, n);
```

Initialise semaphore to number of resources.

Type SEMA declared in sema.h.



Requesting resources

```
sema_wait  (resource);
sema_wait_n (resource, n);
```

If resources available (ie semaphore > 0)

grabs resource (ie semaphore = semaphore - n)

else waits until resource available.

Releasing resource

```
sema_signal  (resource);
sema_signal_n (resource, n);
```

Frees resource (ie semaphore = semaphore + n).

Gives it to a thread if any waiting.

Sharing run-time libraries

Only one thread may perform a library operation at once.

Use pre-defined semaphore par_sema to allocate.

Otherwise use interlocked functions:

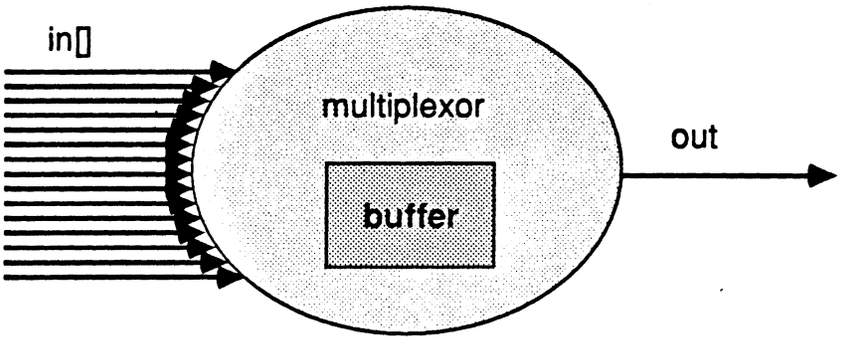
```
#include <par.h>

par_printf
par_fprintf

par_malloc
par_free
```


Multiplexor

The task

One task for each input channel.

Output channel and buffer are shared resources.



Multiplexor

```
#include <chan.h>
#include <thread.h>
#include <sema.h>
char buffer[1024];
SEMA buffer_free;

multiplexor (in, no_ins, out)
    CHAN *in[], *out;
    int no_ins;
{
    extern void receive();
    int i;
    sema_init (&buffer_free, 1);
    for (i=0; i<no_ins; i++)
       thread_create (receive, 50*size_of(int),
                      2, in[i], out);
}
```

Semaphore

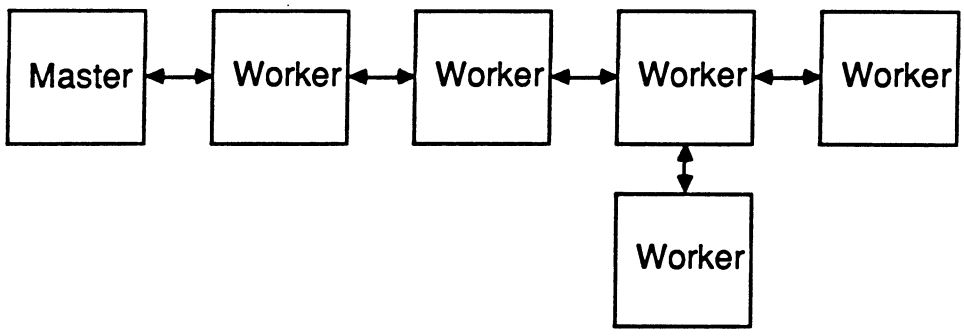
```
void receive(in, out)
    CHAN *in, *out;
{
    int msglen;
    for (;;)
    {
      chan_in_word (&msglen, in);
      sema_wait (&buffer_free);
        chan_in_message (msglen, &buffer[0], in);
        chan_out_word (msglen, out);
        chan_out_message (msglen, &buffer[0], out);
      sema_signal (&buffer_free);
    }
}
```

Flood configuring

Processor farms

Configuring

Processor farm



Advantages and disadvantages

Advantages

Uses all available processors.

Automatically balances load.

Can be used on mixed networks (IMS T4 and T8).

Disadvantages

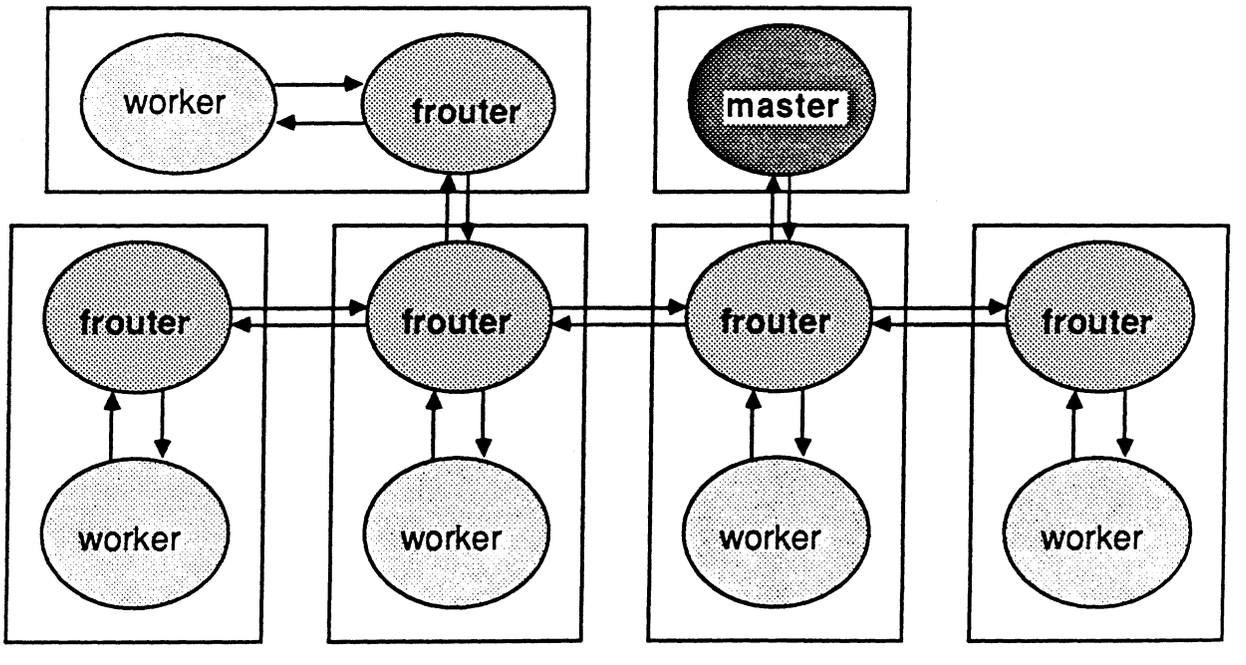
20kbytes of hidden code per node.

Master may be bottleneck.

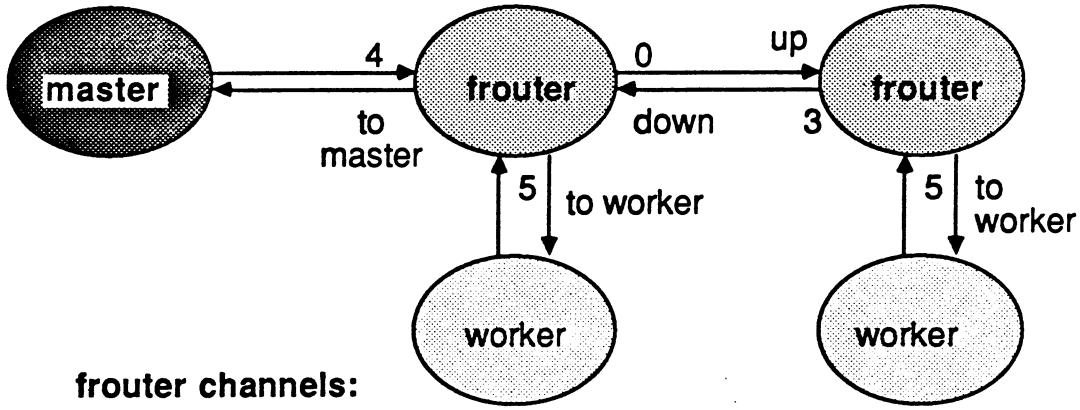
Constrains task layout.

Workers must only compute.

Tasks



Task frouter



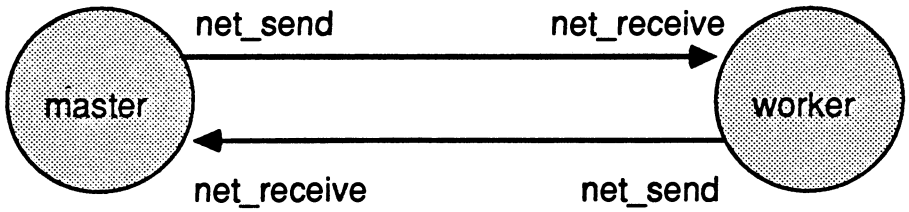
frouter channels:

0 - 2	down
3	up
4	to master
5	to worker

frouter is a standard task provided with Parallel C.

Communications

Communications between master and worker use the library functions net_send and net_receive.



Master communicates with next ready worker.

Messages may be complete tasks or results.

Workers cannot communicate with each other.

Reading messages

Messages are chopped into packets.

```
read_message (buffer, length)
    char buffer[];
    int length;
{
    int complete;
    length = 0;
    complete = 0;
    while (complete != 1)
    {
       length = length +
            net_receive (&buffer[length], complete);
    }
}
```


Sending messages

Maximum packet size is NET_MAX_PACKET_LENGTH bytes (1024).

```
#define PACKET_SIZE 256
send_message (message, length)
    char message[];
    int length;
{
    int complete, start, to_send;
    to_send = length;
    start = 0;
    while (to_send > PACKET_SIZE)
    {
       start = start +
         net_send (PACKET_SIZE, &message[start], 0);
       to_send = length - start
    }
    start = start +
       net_send (to_send, &message[start], 1);
}
```

Worker

Worker is usually sequential.

```
#include <net.h>

main ()

{
    char data[MAX_DATA], results[MAX_RESULTS];
    int data_len, results_len;
    for (;;)
    {
       read_message (data, data_len);
       do_task (data, data_len, results, results_len);
       send_message (results, results_len);
    }
}
```


Master

Master uses threads to send and receive in parallel.

```
#include <thread.h>
#include <sema.h>

main ()

{
    SEMA io_ready;
    sema_init (&io_ready, 1);
    thread_create (send_data, 10000,
                   3, io_ready, fs, ts);
    display_results (io_ready, fs, ts);
}
```

Configuring

Configuration file

Flood configuration needs only:

description of master task;

description of worker task.

```
task master file=mymaster
task worker file=myworker data=10k
```

The names master and worker are understood

by the configurer.

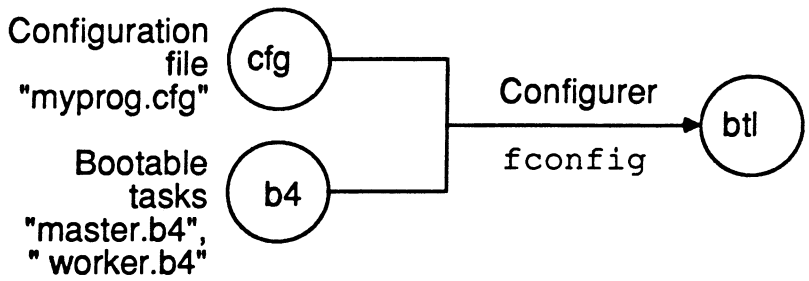
Mixed transputer networks

```
task t4master file=master4
task t4worker file=worker4 data=10k
task t8master file=master8
task t8worker file=worker8 data=10k
```

The names t4master, t8master, t4worker and t8worker

are understood by the configurer.

Configuring




```
t8c master
ilink taskharn.t8x master.bin crtl.lib /o io.c8x
iboot io.c8x /o io.b4 /c
t8c app
ilink taskharn.t8x app.bin sacrtl.lib /o app.c8x
iboot app.c8x /o app.b4 /c
config myprog.cfg myprog.btl
```

Makefile

```
myprog.btl:◊master.b4 worker.b4 myprog.cfg
◊fconfig myprog.cfg myprog.btl

master.b4:◊master.c8x
◊iboot master.c8x /c /o master.b4

master.c8x:◊master.l8x master.bin
◊ilink /f master.l8x

master.bin:◊master.c
◊t8c master

worker.b4:◊worker.c8x
◊iboot worker.c8x /c /o worker.b4

worker.c8x:◊worker.l8x worker.bin
◊ilink /f worker.l8x

worker.bin:◊worker.c
◊t8c worker
```

inmos

Other tools

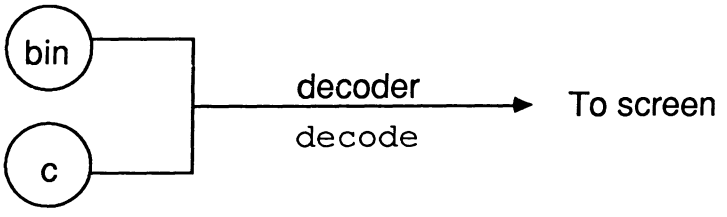
Binary decoder

Libraries

inmos

Binary decoder

Decoder use



To interpret compiled binary files, ie .bin files.

Disassembles and matches with source.

Gives assorted other information.

Outputs to screen.

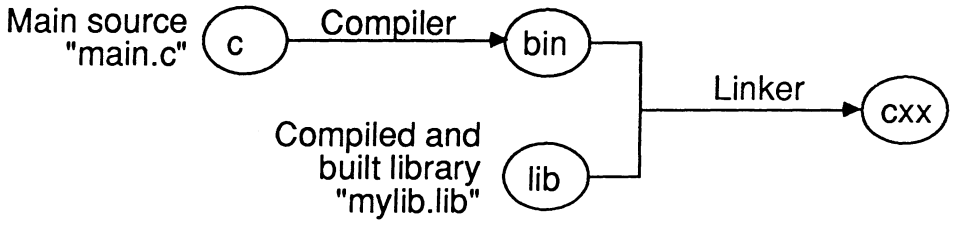
```
decode myprog > myprog.dcd
```


Decoder output

```
Transputer DECODE (V1.2) of hello.bin
ID T4 "occam 2 V2.1" "CC_transputer V2.0.1"
SC  0
TOTALCODE 144  0
STATIC  1
REF  #0,  "printf"
                          20   0008F                    | |
                    00000000   00058                    |____|
  432E4F4C 4C454807 00000001   0007C                    |.___.HELLO>C|
     1  main ()
.............
     2  {
     3    printf ("Hello World");
  646C726F 57206F6C 6C65480C   00000                    |.Hello World|
                          00   0000C                    |_|
                       46 64   00047   ldc      -74
                       FB 21   00049   ldpi
                          70   0004B   ldl      0
           20 20 20 20 20 20   0004C   call     printf
                          B1   00052   ajw      1
                       F0 22   00053   ret
.............
     4  }
```

Libraries

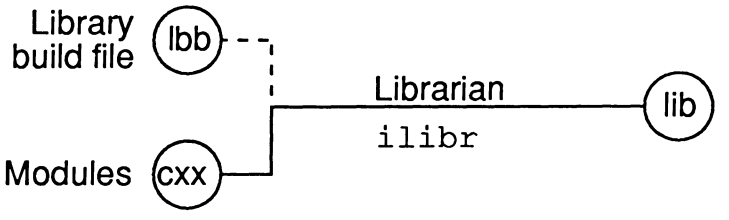
Using libraries




```
#include <mylib.lib>
```

```
t8c  main
ilink mainharn.c8x main.bin mylib.lib crtl.lib
```

Creating a library



Compile all the code in modules - ie the smallest unit that can be loaded.

For multiple targets and modes, compile for each.

Create a library build file.

Run the librarian, ilibr, to join the modules.

```
ilibr /f mylib.lbb
```


Library build files

Lists the files used to build a library.

Used by the librarian and make.

File: mylib.lbb

```
myproc1.c4x
myproc1.c8x
myproc2.c4x
```

```
ilibr /f mylib.lbb
```

Makefiles

Libraries may have separate makefiles

or be included in makefiles of user programs.

```
mylib.lib:◊mylib.lbb myproc1.c4x \
myproc1.c8x myproc2.c4x
◊ilibr /f mylib.lbb
```

Include user libraries in dependency lists.

```
main.bin:◊main.c mylib.lib
◊t8c main
```

inmos

C language implementation

Standards and extensions

Data type implementation

Run time libraries

inmos

Standards and extensions

Standards

The 3L compiler complies with:

Kernighan & Ritchie - 1978

ANSI standard X3J11.

Sequential extensions

Assignment to whole struct and union variables.

Functions with structures as arguments or results.

No restrictions on common struct member names.

31 significant characters in identifiers.

$ allowed anywhere in identifiers.

Escape sequences of unlimited length in strings, eg \dddddd.

Library for access to DOS host system.

Multiple type-specifiers in type-names.

Other features

Register type implemented as integers.

enum and void are keywords. entry is not.

void data types allowed, but not enum.

All bit fields implemented as unsigned int.

>> gives a logical shift, rather than arithmetic.

Loose type checking of . and -> operators.

White space is not allowed within compound operators.

sizeof is not allowed in array size expressions.

#line is accepted but ignored.

=op and int x 3 are not allowed.

Assembling T-code

```
int a=123, b;
asm { ldl a; ldl b; add; stl b;}
```

Assembler works out prefixes and operations.

Symbolic labels and identifiers.

Macros

Compiler switch d defines a macro.

Default value is 1.

```
t8c /dDEBUG /dhelp=3 /dJOE=Jim cats
```

is equivalent to:

```
#define DEBUG 1
#define help  3
#define JOE   Jim
```

at the top of cats.c.

Data type implementation

Simple C types

Sizes of variables and constants in bytes.		32 bit transputer
char	character	1
register	32 bit integer	4
int	32 bit integer	4
unsigned int	16 bit unsigned integer	4
short	32 bit integer	4
long	32 bit integer	4
float	32 bit floating point	4
double	64 bit floating point	8
pointer	address	4

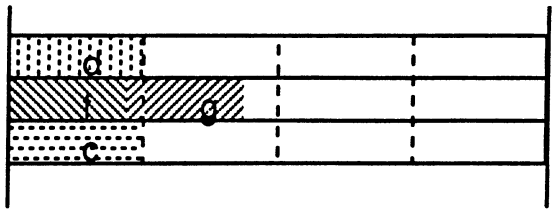
All types except char are word aligned.

Other C types

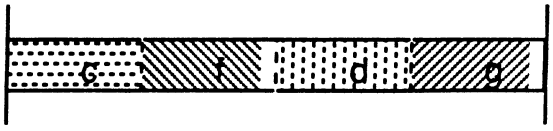
struct and union types are rounded up to whole words.

Each sequence in a struct is byte aligned. Sequences of more than one byte are word aligned.

```
struct s {char c; int f:7, g:7; char d}
```



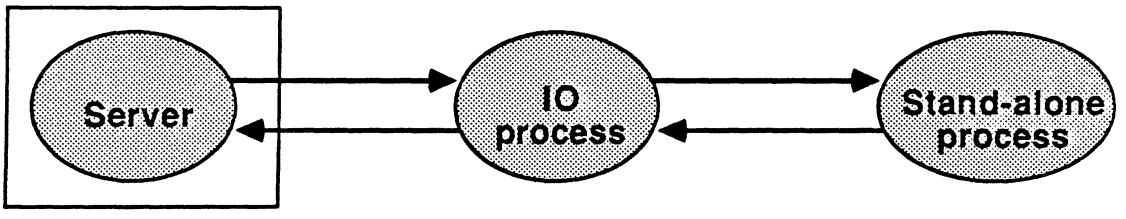

```
struct s {char c; int f:7; char d; int g:7}
```



enum is not implemented.

Run time libraries

Library files



Directory	IO process library	Stand-alone library
\tc2v0\libs	crtl.lib	sacrtl.lib

Library contents

IO library	Stand-alone library
Input and output routines	
Channel i/o routines	Channel i/o routines
Parallel processing	Parallel processing
Transputer routines	Transputer routines
Mathematical routines	Mathematical routines
String handling routines	String handling routines
Date and time routines	

Standard run-time libraries

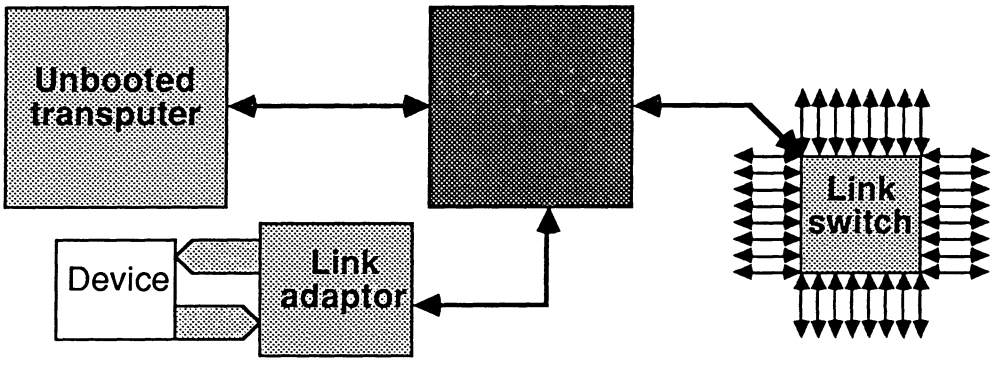
Library	Header	Stand-alone
Standard i/o Stream i/o Binary i/o Text i/o	#include <stdio.h>	
Mathematical	#include <math.h>	√
String handling	#include <string.h>	√
Character classification		√
Conversion	#include <ctype.h>	√
Memory allocation		√
Miscellaneous	#include <assert.h>	√
	#include <setjmp.h>	√
	#include <ascii.h>	√
	#include <errno.h>	√

Non-standard run-time libraries

Library	Header	Stand-alone
Channel i/o	#include <chan.h>	√
	#include <chanio.h>	√
Booting	#include <boot.h>	√
Timers	#include <timer.h>	√
Threads	#include <thread.h>	√
Semaphores	#include <sema.h>	√
Parallel processing	#include <par.h>	√
Processor farm	#include <net.h>	√
Date and time	#include <time.h>	
DOS	#include <dos.h>	

Links

Link input and output



Links may be used to talk to external devices

eg a linkswitch;

a link adaptor;

an unbooted transputer.

Driving a link adaptor

Use channels LinknInput and LinknOutput

(defined in <chan.h>) to talk down links.

```
#include <chan.h>

send_to_device (data)
    int data;
{
    chan_out_word (data, Link2Output);
}

read_from_device (result)
    int result;
{
    chan_in_word (&result, Link2Input);
}
```

Bind statement

Channels may be bound to i/o links

with the BIND statement in the configuration.

```
task link_adaptor ins=3 outs=3 data=10k

bind input  link_adaptor[2] value=&80000010
bind output link_adaptor[2] value=&80000000
```


Link addresses

Link0Output	&80000000
Link1Output	&80000004
Link2Output	&80000008
Link3Output	&8000000C
Link0Input	&80000010
Link1Input	&80000014
Link2Input	&80000018
Link3Input	&8000001C

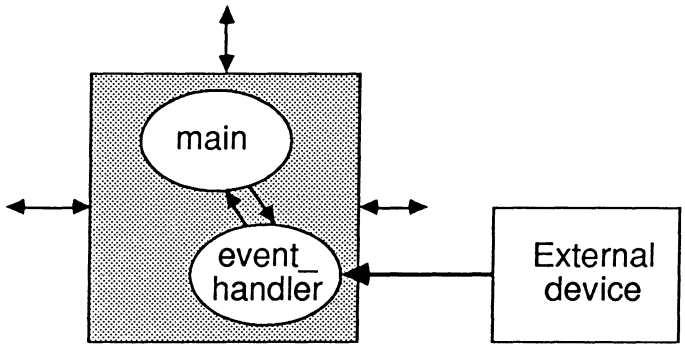
Events

Event handling

Event is like a link but only synchronises.

Generally used for interrupts.

Should be handled by a separate task.



Programming events

Event is like an input only link.

No data is received.

Use high priority for interrupts.

EventReq is defined in <chan.h>.

```
event_handler ()
{
    for (;;)
    {
       int trash;
       chan_in_word (trash, EventReq);
       deal_with_event();
    }
}
```


Binding events

Channels may be bound to the event hardware

with the BIND statement in the configuration.

```
task event_handler ins=3 outs=3 data=10k

bind input event_handler[2] value=&80000020
```

Hello World - possible solution page 1

File: hello.c

```
main  ()
{
   printf  ("Hello World");
}
```

File: hello.l8x -- linker file

```
mainent.c8x
hello.bin
crtl.lib
/o hello.c8x
```

File: hello -- makefile

```
hello.b8x:◊hello.c8x
◊iboot hello.c8x

hello.c8x:◊hello.l8x hello.bin
◊ilink /f hello.l8x

hello.bin:◊hello.c
◊t4c hello
```


Hello World - possible solution page 2

File: run.bat

```
iserver /se /sb %1.b8x
```

To compile and run hello:

```
make -f hello
run hello
```

Change case - possible solution page 1

File: casech.c

```
#include <chan.h>
#include <ctype.h>
#define keys        in[1]
#define results     out[1]
#define data        (char) 0
#define terminate   (char) 1

main (argc, argv, envp, in, inlen, out, outlen)
  char *argv[], *envp[];
  int argc, inlen, outlen;
  CHAN *in[], *out[];
{
  char tag, ch;
  int going = 1;
  while (going == 1)
     {
       chan_in_byte (&tag, keys);
       if (tag == terminate)
         going = 0;
       else
         {
            chan_in_byte (&ch, keys);
            chan_out_byte (chcase(ch), results);
         }
     }
}
```

```
char chcase (ch)
  char ch;
{
  if (isupper(ch))
    return (tolower(ch));
  else if (islower(ch))
    return (toupper(ch));
  else
    return (ch);
}
```


Change case - possible solution page 2

File: termhand.c

```
#include <chan.h>
#include <stdio.h>
#define keys          out[2]
#define results       in[2]
#define data          (char) 0
#define terminate     (char) 1
#define terminatech '%'

main (argc, argv, envp, in, inlen, out, outlen)
  char *argv[], *envp[];
  int argc, inlen, outlen;
  CHAN *in[], *out[];
{
  char ch;
  int going = 1;
  while (going == 1)
   {
     ch = getchar();
     if (ch == terminatech)
        {
          going = 0;
          chan_out_byte (terminate, keys);
        }
     else
        {
          chan_out_byte(data, keys)
          chan_out_byte(ch, keys);
          chan_in_byte(&ch, results);
          putchar(ch);
        }
   }
}
```

Change case 2 possible solution page 1

File: chcase.cfg

```
processor  host
processor  root

wire link root[0] host[0]

task termhand  ins=3 outs=3
task casech    ins=2 outs=2 data=10k
task filter    ins=2 outs=2 data=10k
task iserver   ins=1 outs=1

place iserver   host
place termhand  root
place casech    root
place filter    root

connect ?          filter[0]    iserver[0]
connect ?          iserver[0]   filter[0]

connect fs         filter[1]    termhand[1]
connect ts         termhand[1]  filter[1]

connect keys       termhand[2]  casech[1]
connect results    casech[1]    termhand[2]
```


Change case 2 possible solution page 2

File: chcase

```
chcase.btl:◊termhand.b4 casech.b4 chcase.cfg
◊config chcase.cfg chcase.btl

termhand.b4:◊termhand.c8x
◊iboot termhand.c8x /c /o termhand.b4

termhand.c8x:◊termhand.18x termhand.bin
◊ilink /f termhand.18x

termhand.bin:◊termhand.c
◊t8c termhand

casech.b4:◊casech.c8x
◊iboot casech.c8x /c /o casech.b4

casech.c8x:◊casech.18x casech.bin
◊ilink /f casech.18x

casech.bin:◊casech.c
◊t8c casech
```

Change case 2 - possible solution page 3

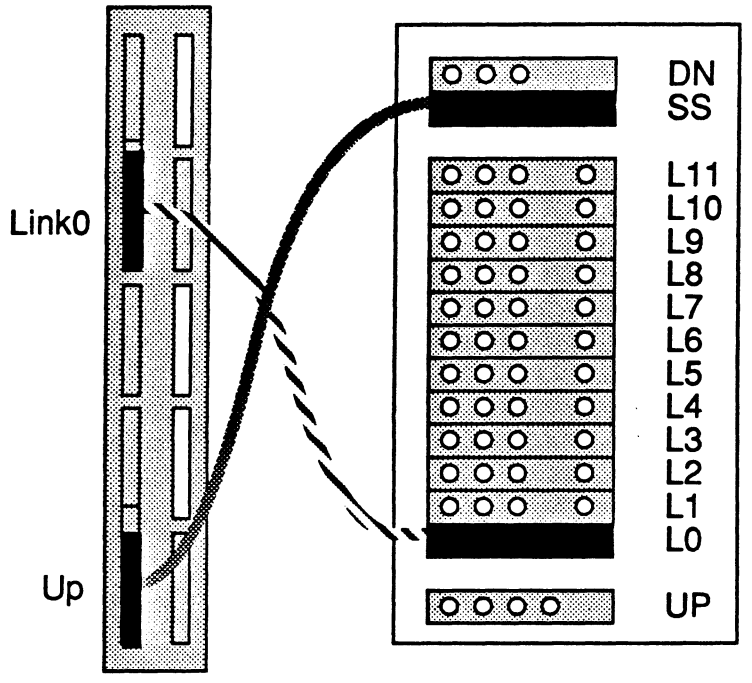
File: termhand.l8x

```
taskharn.t8x
termhand.bin
crtl.lib
/o termhand.b4
```

File: casech.l8x

```
taskharn.t8x
casech.bin
sacrtl.lib
/o casech.b4
```

Change case 3 - possible solution page 1



Change case - 3 possible solution page 2

File: chcase.cfg

```
processor  host
processor  root
processor  target
wire  link1  host[0]  root[0]
wire  link2  root[3]  target[0]

task  termhand   ins=3  outs=3
task  casech     ins=2  outs=2  data=10k
task  filter     ins=2  outs=2  data=10k
task  iserver    ins=1  outs=1

place  iserver   host
place  termhand  root
place  filter    root
place  casech    target

connect  ?          filter[0]    iserver[0]
connect  ?          iserver[0]   filter[0]
connect  fs         filter[1]    termhand[1]
connect  ts         termhand[1]  filter[1]
connect  keys       termhand[2]  casech[1]
connect  results    casech[1]    termhand[2]
```

Change case - 3 possible solution page 3

```
chcase.btl:◊termhand.b4 casech.b4 chcase.cfg
◊config chcase.cfg chcase.btl

termhand.b4:◊termhand.c8x
◊iboot termhand.c8x

termhand.c8x:◊termhand.l8x termhand.bin
◊ilink /f termhand.l8x

termhand.bin:◊termhand.c
◊t8c termhand

casech.b4:◊casech.c4x
◊iboot casech.c4x

casech.c4x:◊casech.l4x casech.bin
◊ilink /f casech.l4x

casech.bin:◊casech.c
◊t4c casech
```

Clock practical possible solution - page 1

File: ticker.c

```
#define tick out[1]
#define delay 15625
#include <timer.h>
#include <chan.h>

main (argc, argv, envp, in, inlen, out, outlen)
   char *argv[], *envp[];
   int argc, inlen, outlen;
   CHAN *in[], *out[];
{
    int time;
    time = timer_now();
    for (;;)
    {
       time = timer_plus (time, delay);
       timer_wait (time);
       chan_out_word (0, tick);
    }
}
...  timer_plus
```

File: ticker.l8x

```
taskharn.t8x
ticker.bin
sacrtl.lib
/o ticker.c8x
```


Clock practical possible solution - page 2

File: scrnhand.c

```
#include <chan.h>
#include <stdio.h>
#define tick       in[2]

main (argc, argv, envp, in, inlen, out, outlen)
   char *argv[], *envp[];
   int argc, inlen, outlen;
   CHAN *in[], *out[];
{
    int signal, secs;
    for (secs=0; secs>=0; secs++)
    {
       printf ("%d\n", secs);
       chan_in_word (&signal, tick);
    }
}
```

File: scrnhand.l8x

```
taskharn.t8x
scrnhand.bin
crtl.lib
/o scrnhand.c8x
```

Clock possible solution page 3

File: clock.cfg

```
processor host
processor root

wire link root[0] host[0]

task scrnhand  ins=3 outs=3
task ticker    ins=2 outs=2 data=10k
task filter    ins=2 outs=2 data=10k
task iserver   ins=1 outs=1

place iserver  host
place scrnhand root
place ticker   root
place filter   root

connect ?      filter[0]   iserver[0]
connect ?      iserver[0]  filter[0]

connect fs     filter[1]   scrnhand[1]
connect ts     scrnhand[1] filter[1]

connect tick  ticker[1]   scrnhand[2]
```


Clock possible solution page 4

File: clock - makefile

```
clock.btl:◊scrnhand.b4 ticker.b4 clock.cfg
◊config clock.cfg clock.btl

scrnhand.b4:◊scrnhand.c8x
◊iboot scrnhand.c8x /c /o scrnhand.b4

scrnhand.c8x:◊scrnhand.18x scrnhand.bin
◊ilink /f scrnhand.18x

scrnhand.bin:◊scrnhand.c
◊t8c scrnhand

ticker.b4:◊ticker.c8x
◊iboot ticker.c8x /c /o ticker.b4

ticker.c8x:◊ticker.18x ticker.bin
◊ilink /f ticker.18x

ticker.bin:◊ticker.c
◊t8c ticker
```