inmos®

Transputer Development System Delivery manual

1 Introduction

This manual is the “delivery manual” for the Transputer Development System (TDS), describing how to install the software, and giving an overview of the contents of the release. This release of the TDS is the IMS D700D, for the IBM PC and compatibles, and the IMS D800D, for the NEC PC-9801. For the rest of this manual the release will be referred to as the ‘D700D’.

The software of the TDS runs on a transputer board attached to the host computer. This may be an IMS B004 or IMS B008 for the IBM PC, an IMS B010 for the NEC PC, or a board compatible with one of these. The binary code of the TDS will run either on an IMS T414 processor (although not a revision A T414) or on an IMS T800. A program called a *server* runs on the host computer and provides the TDS with access to the keyboard, screen and filing system of the host computer. The TDS can be ported to a different host/transputer board combination by porting the server, and a version of the server written in C is provided for this purpose.

The release consists of nine 360Kbyte floppy disks and a number of manuals. This manual includes the information needed to upgrade from the previous release of the software (the IMS D700C). In addition it gives a list of known bugs and problems with the system, at the time of release.

The other manuals provided with the system are as follows:-

1 occam 2 reference manual.

2 A tutorial introduction to occam programming.

3 The transputer development system manual.

2 Installation instructions

The system is supplied on nine 360Kbyte disks for the IBM PC AT and compatibles. The files are provided in a compacted form produced by an archive utility called `arc`. The installation procedure assumes that you are going to install all of the supplied system onto the same volume. You will need about 6 Mbytes of space on your volume for the whole system, but you can be more selective about what you install, as described below.

2.1 Deleting the D700C release

The D700D release of the TDS replaces the previous release of the TDS, the D700C. The directory names for the D700D are different from those of the D700C, so if you wish you can have the two releases co-existing on your computer over the period of transition to the new release. Alternatively you may wish to delete the D700C before installing the D700D. A batch file is provided on the first disk of the D700D to aid in this; it assumes that the files are installed as in the D700C installation procedure. The batch file is called `deld700c.bat`. Proceed as follows:

1 First of all make sure that there are no files in any of the D700C directories which you wish to keep.

2 Set your default drive to the drive containing the TDS (for example, `c:`).

3 Place the first D700D disk into the floppy drive (for example, drive `a:`), and type:

```
a:deld700c
```

If the floppy drive has a different name from `a`, use that in the line above instead.

4 The program will start up, give a message and ask for you to press a key to proceed. Do that (you can press ctrl-C here if you have changed your mind!).

5 As the program runs you will have to keep entering 'y' to confirm the deletions in each directory (again you can press ctrl-C to terminate the batch file).

2.2 Full installation

To install the whole system proceed as follows:

1 Set your default drive to the drive you wish to install onto (for example, `c:`).

2 Put disk 1 into the floppy drive (for example drive `a:`) and type

```
a:install a
```

If you have used a different floppy drive, replace `a` in the above line with the name of that drive (e.g. `b:install b`).

3 The installation procedure will then instruct you to put in each of the subsequent disks, and will then proceed to install the archived files. This will take up to 30 minutes, depending on the speed of your disks.

The following directories are created during installation:

Directory	Contents
`\tds2\system`	TDS system and utilities
`\tds2\complibs`	Compiler libraries
`\tds2\tutor`	Some simple tutorial programs
`\tds2\examples`	Some example programs
`\tds2\tools`	Code for tools
`\tds2\tools\src`	Source for tools
`\tds2\iolibs`	I/O library header files
`\tds2\iolibs\src`	I/O library sources
`\tds2\iolibs\ht2`	I/O libraries for the T2 (HALT mode)
`\tds2\iolibs\ht4`	I/O libraries for the T4 (HALT mode)
`\tds2\iolibs\ht8`	I/O libraries for the T8 (HALT mode)
`\tds2\mathlibs`	Maths library header files
`\tds2\mathlibs\src`	Maths library sources
`\tds2\mathlibs\ht2`	Maths libraries for the T2 (HALT mode)
`\tds2\mathlibs\rt2`	Maths libraries for the T2 (REDUCED mode)
`\tds2\mathlibs\ht4`	Maths libraries for the T4 (HALT mode)
`\tds2\mathlibs\rt4`	Maths libraries for the T4 (REDUCED mode)
`\tds2\mathlibs\ht8`	Maths libraries for the T8 (HALT mode)
`\tds2\mathlibs\rt8`	Maths libraries for the T8 (REDUCED mode)
`\tds2\server\server14`	Server sources
`\tds2\server\afserver`	Aferver sources
`\tds2\server\cserver`	Cserver sources

In addition to the TDS directories, the installation procedure also creates the directory `\arcd700d` which at the end of installation includes the `arc` program, and a number of batch files. These include a file called `deld700d.bat` which removes the TDS files and directories (this may be useful to remove the system before installing another release of it). In addition there are a number of batch files for selectively installing parts of the system; see the next section. You can delete the contents of `\arcd700d` if you wish, as it is not required to run the TDS.

2.3 Selective installation

You can be more selective about which bits you install by giving parameters to the `install` program. The parameters are:

`/ne`	No examples; the `tutor` and `examples` directories are not made.
`/nt`	No tools source; the `tools\src` directory is not made.
`/nl`	No library sources; no `iolibs\src` and `mathlibs\src` directories.
`/ns`	No server source; the `server` directories are not made.

For example, to omit the examples, the tools source and the server sources, type

```
a:install a /ne /nt /ns
```

If you have asked for part of the system to be omitted, the system will not ask you to load unnecessary disks.

Omitting the examples will save about 200 Kbytes, the tools source will save about 900 Kbytes, the server sources will save about 900 Kbytes, and the library source will save about 600 Kbytes.

If you subsequently decide that you need to install one of the parts you have omitted, you can use one of the following batch files left in `\arcd700d` after installation:

`getparte.bat`	The `tutor` and `examples` directories.
`getpartt.bat`	The `tools\src` directory.
`getpartl.bat`	The `iolibs\src` and `mathlibs\src` directories.
`getparts.bat`	The `server` directories.

When running one of these, supply as a parameter the name of the floppy drive you will use to load the files; for example:

```
getparte a
```

The batch file will then run, telling you which disks to insert, and then de-archiving the files into the appropriate directories.

2.4 Using the system on an NEC PC

If you are using an NEC PC, it is suggested that you set up the files in the `\tds2\system` directory so that the NEC version is picked up by the standard commands. To do this, rename the file `nectds2.bat` in the `\tds2\system` directory to `tds2.bat` and the file `necafser.exe` to `afserver.exe` (you will first have to delete or rename the existing files with these names).

The IMS B010, the transputer board for the NEC PC, is supplied with 1 MByte of memory, rather than 2 Mbytes as on the IMS B004. When running the TDS on the IMS B010, there is less memory available to load and run utilities. Utility sets should be loaded one at a time and cleared when another one is required; otherwise there will be very little space for running the compiler. The `Autoload` fold in the toolkit fold should be changed so that it only includes the compiler utility set. The file handling utility set and the debugger should be placed in the `Tools` fold and only loaded when needed,

The 'interrupt' key for the NEC, which causes the TDS to be interrupted and rebooted, is 'ctrl-V'.

2.5 Setting up the system for use

WARNING: This release will not work on a revision A IMS T414. If you have a T414A in your transputer board you must upgrade to a later revision of the T414 before starting to use the system. The system will give an error message if an attempt is made to start it on a T414A.

Once you have installed the system, you need to be sure that you can run it. The directory `\tds2\system` contains a `.BAT` file for running the TDS. Put `\tds2\system` into your DOS `PATH` command in `AUTOEXEC.BAT`.

You will need to edit the `TDS2.BAT` file under the following circumstances:

- If you are running the TDS on a different drive from the drive on which the TDS is installed, you will need to edit all the file names in the batch file to include the drive name.
- If you are running the TDS on an IMS T800, include the parameter `-p T800` in the server call.
- If you are running the TDS on a board which is larger or smaller than 2

Mbytes, change the value after the `-s` parameter to match the memory size.

- If for some reason you have installed the transputer board at a different link adaptor address value than the standard one (see the board manual for details of this), you will need to supply a `-l` parameter to the server giving the link adaptor address (for example, to specify a link adaptor at base address hexadecimal `#310`, include the server parameter `-l #310`).

You can try out the system in the directory `\tds2\tutor`. To use the system in a different directory you will need a "toolkit" fold. Copy the file `TOPLEVEL.TKT` from `\tds2\system` into the directory you want to work in.

As supplied, the system is intended to be used within the same drive as it is installed on. If you wish to use the system on a different drive, you will need to make a different toolkit fold which will refer to utility and library files in a manner including the drive name. To do this, follow the instructions below. You will need to use the TDS to complete this task, so have a keyboard map with you (see appendix A of the TDS manual).

1 Go to the directory `\tds2\system` and type `tds2` to start the system. Press ENTER FOLD to enter the top level file.

2 Use ENTER TOOLKIT (ALT-F1) to enter the toolkit fold. Edit the directory names in the `Library logical names` fold to include a drive name.

3 Exit the toolkit fold.

4 Follow the instructions in the top level file to make new `Autoload` and `Tools` folds.

5 Exit the toolkit fold, exit to top level, and finish the session (FINISH is CTRL-1 on the numeric keypad).

The `TOPLEVEL.TKT` file in `\tds2\system` will now be of the required form.

Once you are satisfied that the system is ready for use, turn to the TDS manual for instructions on how to start using the system.

3 Software contents of the TDS release

This chapter gives an overview of the contents of this release of the TDS. The components are split up as follows:

1 TDS system and utilities (in `\tds2\system`).

2 Compiler libraries (in `\tds2\complibs`).

3 Software tools (in `\tds2\tools`).

4 I/O libraries (in `\tds2\iolibs`).

5 Maths libraries (in `\tds2\mathlibs`).

6 Tutorial material (in `\tds2\tutor`).

7 Example programs (in `\tds2\examples`).

8 Server sources (in `\tds2\server`).

3.1 TDS system

The TDS system comprises:

1 The `TDS2.BAT` DOS command file.

2 The TDS server object (`SERVER14.EXE`).

3 The TDS loader (`TDSLOAD.B4`)

4 The TDS `.XSC` file.

5 The compiler/configurer utility set (`OCCAM2.CUT`).

6 The file handling utility set (`FILEHAND.CUT`).

7 The debugger program (`DEBUGGER.CEX`).

8 The toolkit file (`TOPLEVEL.TKT`).

9 The host file server object. (`AFSERVER.EXE`).

10 The NEC server object (**NECSERVE.EXE**).

11 The NEC host file server object (**NECAFSER.EXE**).

12 The ANSI server object compiled for the IBM (**ANSISERV.EXE**).

13 The C server object compiled for the IBM (**CSERVER.EXE**).

14 The **NECTDS2.BAT** file to call the NEC server.

15 A number of **.LD** and **.TBL** files for the NEC PC.

3.2 Compiler libraries

The compiler libraries are as follows:

```
reals
realpds
ints
intpds
r64utils
t2utils
```

The compiler libraries are used by the occam compiler to support extended data types in occam and the implicitly defined libraries described in the TDS manual. The compiler libraries are provided in the directory **\tds2\complibs**, compiled for all transputer types and error modes (see below). The sources are not provided.

Transputer types supported are:

- **T8**: The IMS T800.
- **T4**: The IMS T414.
- **T2**: The IMS T212.

Error modes supported are:

- **HALT** mode.
- **STOP** mode.
- **REDUCED** mode.

These terms are explained in the TDS manual.

The extraordinary link handling library (`reinit`) and the block CRC library (`blockcrc`) are provided compiled for all transputer types and error modes, in the same directory as the compiler libraries. These libraries are described in the TDS manual. The sources are not provided.

3.3 Software tools

The software tools comprise the following principal tools (referred to in the `Tools` fold of the toolkit)

1 The lister program `LIST.CEX`.

2 The unlister program `UNLIST.CEX`.

3 The link transfer program `LINKCOPY.CEX`.

4 The transputer network tester program `NETTEST.CEX`.

5 The memory interface program `MEMINT.CEX`.

6 The EPROM hex program `EPROMHEX.CEX`.

7 The hex-to-programmer program, `HEXTOPRG.CEX`.

These are all referred to or described in detail in the TDS manual.

The tools source directory contains:

1 The link transfer program sources.

2 The lister program sources.

3 The unlister program sources.

4 The EPROM hex program sources.

5 The hex-to-programmer sources.

6 Example ROM sources.

7 Extractor program sources.

8 Network memory browser sources.

9 An “add preamble” program sources.

10 Example transputer loaders and analyse worms.

11 A simple transputer network exploration worm.

12 Sources of a disassembler program.

Tools which are not described in the TDS manual are accompanied by some explanatory text in the tools source directory.

3.4 I/O libraries

The input/output libraries comprise:

```
ioconv
extrio
strings
userio
interf
slice
ufiler
msdos
derivio
afio
afiler
afserver
t4board
t2board
```

These are all described in the TDS manual.

The I/O libraries are provided compiled in `HALT` mode only, for the `T8`, the `T4` and the `T2` (where relevant). The sources of these libraries are supplied.

The I/O library sources are provided in the directory:

```
\tds2\iolibs\src
```

The compiled and compacted versions are supplied in the directories:

```
\tds2\iolibs\ht8
\tds2\iolibs\ht4
\tds2\iolibs\ht2
```

In addition there are the following header libraries, supplied in the directory `\tds2\iolibs`, and listed in an appendix to the TDS manual:

```
uservals
userhdr
filerhdr
krnlhdr
afhdr
```

3.5 Maths libraries

The maths libraries comprise:

```
snglmath
dblmath
t4math
```

These are described in the TDS manual. The sources of these libraries are available.

The elementary function libraries are provided compiled in **HALT** and in **REDUCED** mode.

There are two basic versions of the elementary function library: a version using floating point arithmetic, and a version using fixed point arithmetic. The floating point function library is compiled for the **T8**, the **T4** and the **T2** (the **T2** version contains some functions specially optimised for the **T2**). The fixed point function library is compiled for the **T4**, and referred to using the name **t4math**. On the IMS T414, this library should be used in preference to the floating point function library, as it is much faster. The floating point function library should only be used on the IMS T414 if for some reason it is required to behave exactly the same way as on an IMS T800 (for example, if a program intended for a T800 is being tested on a T414).

The sources are in the directory:

```
\tds2\mathlibs\src
```

The compiled and compacted libraries are placed in the directories:

```
\tds2\mathlibs\ht2
\tds2\mathlibs\ht4
\tds2\mathlibs\ht8

\tds2\mathlibs\rt2
\tds2\mathlibs\rt4
\tds2\mathlibs\rt8
```

In addition there are the header libraries **mathvals** and **mathhdr**, placed in the following directory:

```
\tds2\mathlibs
```

The header **mathvals** is listed in an appendix to the TDS manual.

3.6 Tutorial

The tutorial directory contains the following:

1 The editor tutorial, in **TUTORIAL.TOP**. This is described in chapter 4 of the TDS manual.

2 The user guide examples, in **EXAMPLES.TOP**. These are a pipeline sorter example, in various configurations, and the debugger example, both referred to in the 'User guide' chapters of the TDS manual.

3.7 Example programs

The examples contain:

1 Simple input/output examples.

2 User filer interface examples.

3 Simple transputer network examples.

3.8 Server sources

The server sources contain:

1 The source of the TDS server (in assembler).

2 The source of a TDS server (in C).

3 The source of the host file server (in C).

Directories are:

```
\tds2\server\server14
\tds2\server\cserver
\tds2\server\afserver
```

These are described in more detail in the next chapter.

4 The servers

This release of the TDS contains a number of different servers. A *server* is a program which runs on the host computer, such as the IBM PC, and boots a transputer program into its transputer board. After it has booted the program it communicates with the program running on the transputer and provides it with the ability to access the terminal and filing system of the host computer.

There are two basic types of server provided with this release of the TDS:

- The "TDS file server" which supports the TDS running on a transputer board.
- The "host file server" which provides programs running independently of the TDS with the ability to access the terminal and filing system of the host computer.

Two entirely different versions of the TDS file server are provided: one written in Intel 8086 assembler, and one written in C.

The 8086 assembler version of the TDS file server is contained in the file **`server14.exe`** in **`\tds2\system`**, and is the one normally used to support the TDS on the IBM PC. The sources of this are provided in case the user wishes to make minor modifications to it. In addition to the standard version, there is also a version in the file **`ansiserv.exe`** which uses ANSI commands for screen driving. Although slower than the standard screen driver, this may be useful if you are running the TDS on a PC which is connected to an ANSI terminal over a network. The PC must have an ANSI terminal driver installed for this server to work. There is also an NEC version of the assembler server, in the file **`necserve.exe`**; this differs in the terminal handling, and in the address of the link adaptor on the transputer board. These different versions of the server can be built from the same sources by setting different compilation flags in the sources.

The C version of the TDS file server may be used to port the TDS to a different host computer with an attached transputer board. The C version supports the same facilities as the assembler version, apart from the ability to search within the current directory for files with the extension **`.TOP`**. This facility may be added by the user, if desired, when porting the C server to a different operating system. A compiled version of the C server, for the IBM PC, is in the file **`cserver.exe`** in the directory **`\tds2\system`**. When running the C server, it requires that all TDS files be contained in the file **`toplevel.top`** in the directory.

The host file server is provided in a compiled form for the IBM PC in the file **`afserver.exe`**, and for the NEC PC in the file **`necafser.exe`**. The host file server is written in C, with a small (optional) section in assembler, To summarise, the following executable versions of the servers are provided in the directory `\tds2\system`:

`server14.exe`	Assembler version of TDS server for IBM PC
`necserve.exe`	Assembler version of TDS server for NEC PC
`ansiserv.exe`	Assembler version of TDS server for IBM PC (ANSI screen)
`cserver.exe`	C version of TDS server for IBM PC
`afserver.exe`	Host file server for IBM PC
`necafser.exe`	Host file server for NEC PC

The sources are provided in the following directories:

`\tds2\server\server14`	Assembler TDS server sources
`\tds2\server\afserver`	Host file server sources
`\tds2\server\cserver`	C TDS server sources

Server sources are provided in TDS file format, and may be viewed using the TDS. To produce executable versions, the source files must be exported from the TDS into standard DOS files, and then compiled using the Microsoft assembler or C compiler. Each of the source directories contains instructions on how to do this.

5 Changes since D700C

This section is intended for users of the D700C release, and outlines the main changes to the system since that release. It may be omitted by new users of the software.

5.1 Functional changes to the TDS

The TDS programming environment has been improved substantially since the D700C release. The main additions are as follows:

- The TDS can run on a board with more memory than 2 Mbytes, and can use the extra memory itself or supply it to a user program running within the TDS.
- A "toolkit" fold containing utilities and their default parameters can be accessed easily from anywhere in the fold structure. The toolkit is preserved between sessions using the TDS.
- Multiple utilities and programs can be kept in memory at the same time.
- A set of standard working utilities can be loaded with a single keypress.
- Selection of parameters to utilities can be done more easily.
- Before running utilities or programs within the TDS, they are moved to make optimum use of the on-chip RAM.
- There are extra editing functions, such as word move/delete, move to top or bottom of fold, picking and copying of lines into an accumulating buffer.
- It is possible to "suspend" the TDS temporarily to issue some DOS commands.
- A "macro" key equivalent to a commonly used sequence of keys may be defined for use during a session.

A number of the functions have changed their positions on the keyboard, to accomodate the new functions in the system. See Appendix A of the TDS manual for the new keyboard layout.

This release of the TDS contains a compiler which conforms to the language as described in the occam 2 Reference Manual, with a few minor exceptions described in the next section. The main additions to the compiler in the D700C release are as follows:

- Implementation of functions.
- Full checking of alias and usage rules.
- Compilation for different error modes.
- Use of separate workspaces for scalars and vectors.
- Logical names for libraries.
- occam 2 language at configuration level.
- Recompilation of programs using parameters from the previous compilation.
- Multiple program compilation.
- Multiple library compaction.

These are all described in detail in the TDS manual.

This release also includes a post-mortem source-level debugger which can be used on programs run within the TDS, programs on a network loaded from the TDS, or programs booted from DOS with a server. A hardware debugging tool known as the "Transputer network tester" is also included.

5.2 Changes to existing D700C programs

This section describes a number of minor changes which will be required in existing programs to make them compile with the compiler in the D700D. Most of these changes were described in the note supplied with the D700C entitled "Writing occam programs to minimise future changes".

5.2.1 EXE programs

The way that **EXE** programs should be written has changed since the D700C release (this has been done for technical reasons associated with the separate vector space). It is now not necessary to start a program with a procedure heading indicating the channel interface between the **EXE** and the TDS. This interface is now supplied by the compiler. So where previously an **EXE** might have been written as:

```
{{{  EXE myprog
{{{F myprog.tsr
PROC user.program(CHAN OF ANY keyboard, screen,
       [max.files]CHAN OF ANY from.user.filer,
                              to.user.filer)
  ...  Declarations
  SEQ
    ...  Program
:
}}}
}}}
```

Now it has the form:

```
{{{  EXE myprog
{{{F myprog.tsr
...  Declarations
SEQ
  ...  Program
}}}
}}}
```

The channel names (for example, **keyboard** and **screen**) are now automatically supplied by the compiler, so they have fixed names and can be used directly without declaration. The channel parameters supplied are listed in chapter 6 of the TDS manual.

The keyboard channel is now an **INT** protocol; apart from this, no changes have been made to the behaviour of these channels since the D700C release. Ideally these should be proper occam protocols instead of **ANY**, but they have been preserved to allow existing programs to continue to run unchanged.

The easiest way to adapt existing programs is to put in a procedure call at the end, as follows:

```
{{{  EXE myprog
{{{F myprog.tsr
PROC user.program(CHAN OF INT kbd,
                  CHAN OF ANY scr,
                  [max.files]CHAN OF ANY from.uf,
                                         to.uf)
  ...  Declarations
  SEQ
    ...  Program
:
user.program(keyboard, screen, from.user.filer,
                               to.user.filer)
}}}
}}}
```

Note that this allows the supplied channels to be renamed to any names used to identify them in an existing program.

5.2.2 Configuration language

The configuration language is now a proper subset of occam 2. This may require a few minor changes to expressions at configuration level.

5.2.3 Lists of abbreviations, retypes and placements

The D700C compiler allowed lists of names in abbreviations, retypes and placements. This is not supported in the current release.

So instead of writing:

```
VAL one IS 1, two IS 2:
PLACE var0 AT #0, var1 AT #1:
VAL INT64 fp64.zero RETYPES 0.0(REAL64),
          fp64.one RETYPES 1.0(REAL64):
```

write the following:

```
VAL one IS 1:
VAL two IS 2:
PLACE var0 AT #0:
PLACE var1 AT #1:
VAL INT64 fp64.zero RETYPES 0.0(REAL64):
VAL INT64 fp64.one RETYPES 1.0(REAL64):
```

5.2.4 Protocols

There is a minor change to the handling of **CHAN OF ANY**. In this release a channel declared to be **CHAN OF ANY** may be passed as an actual parameter to a procedure with a formal channel parameter of any protocol (for example, **CHAN OF INT**, or a user-defined protocol). This is the same as in the D700C. However, a channel declared with a protocol may not be passed as an actual parameter to a procedure whose formal parameter is declared as a **CHAN OF ANY**, as was allowed in the D700C.

5.2.5 Program checking rules

This release now supports usage and alias checking, as decribed in the occam 2 reference manual. The TDS manual (chapter 5) describes some details about the implementation of these checks, particularly in relation to array elements. It is recommended that existing programs be adapted to pass these checks as soon as possible; however, the checks can be switched off in the meantime, using the compiler's **alias.check** and **usage.check** parameters.

5.2.6 Functions

Functions were not supported in the D700C compiler, but are supported in this release, and are used in the supplied libraries. This has the following implications for existing programs written without functions.

- Some of the standard library procedures are now functions, so programs containing calls to these will have to be changed to make them into function calls. The standard libraries are listed in Appendix J of the occam 2 Reference Manual.
- The elementary function library provided (**SIN** etc.) contains functions rather than procedures. It is still possible to recompile the D700C versions of these with the current release, but users are advised to move to the new versions as soon as possible.

- The alias checking rules have the implication that it is not possible to pass the same variable more than once as a parameter to a procedure (except as a **VAL** parameter). However, it is possible to return a value into a variable from a function call, where that variable is also one of the function parameters. It may therefore be necessary to turn some procedures into functions before it will be possible to fully check the program with usage and alias checking turned on.

5.2.7 Program layout

The rules for program layout have been tightened up since the D700C release. The new rules for program layout are given at the start of the occam 2 Reference Manual. The main implications for existing programs are as follows:

- The compiler in the D700C was more lenient about where a program statement could be broken across lines, so some lines may have to be edited to bring them into line with the language rules.
- A text string must be broken by ending the first line of the string with *, and starting the following line with *, instead of using double quotes, as in the D700C.
- A comment may not be less indented than the following statement, which may require some editing of existing comment lines.

5.2.8 Library references

The system of "library logical names" in this release gives a much more convenient method for identifying libraries. For libraries provided by INMOS, such as the I/O libraries and maths libraries, it is suggested that users edit their programs to pick up the new versions of these libraries using their logical names. For user-defined libraries, the method of identifying libraries by their actual file names may be retained, if desired, or replaced with logical name references. Users may wish to re-examine how their libraries are organised, in the light of the facilities provided by the logical name system.

5.2.9 The libraries

The elementary function libraries supplied with the D700C have been replaced with new libraries using functions. The functions supported in the libraries are the same as in the D700C.

The I/O libraries have been given logical names which relate to the file names of the libraries in D700C. For example, instead of writing:

```
#USE "\directory\userio.tsr"
```

write:

```
#USE userio
```

The I/O libraries have been extended with new libraries, such as a string handling library `strings` (which incorporates some of the string handling procedures from `userio`) and with new procedures in some of the libraries. Some library procedures have had their names changed, principally in the libraries `afiler` (to make their style consistent with the other libraries) and `msdos` (to ensure that the names are distinct from the names of procedures in other libraries). There have been some changes to the parameter lists of some library procedures. Apart from these superficial changes, the facilities provided by the I/O libraries in D700D are a superset of those in D700C, and moving to the new libraries should be fairly straightforward.

6 The implementation of occam

This release of the TDS contains a compiler which supports the occam language as described in the occam 2 Reference Manual. There are some implementation restrictions governing how particular language features may be combined, which are listed here. Most of these restrictions are due to space limitations within the compiler, and there are no current plans to fix them. In addition, this section lists some of the upper limits on program size imposed by the compiler's internal data structures.

6.1 Implementation restrictions

- `VALOF`s may not appear within expressions.
- Multiple assignments may not include assignments of arrays.
- A specification may not appear immediately before an option on a `CASE` statement.
- A specification may not appear immediately before an `ALT` process appearing as a guard within another `ALT`.
- An expression within a table may not include a function call. (even an implicit function call generated by the compiler, e.g. for long arithmetic).
- An expression within an `ALT` guard may not include a function call (even an implicit function call generated by the compiler, e.g. for long arithmetic).
- At the outermost level of an `SC` (i.e. outside the procedures of the `SC`), a constant declaration may not include a function call (even an implicit function call generated by the compiler, e.g. for long arithmetic).
- A constant may not be retyped into an array.
- A table may not be placed at a location in memory.
- An `ALT` guard may not include an input from a `PORT`. Since `PORT`s are always ready, this can always be replaced by a `SKIP` in the guard and a `PORT` input at the start of the process.
- Replicated `PRI PAR` is not implemented.

6.2 Size limitations

In the following list 'procedure' should be taken to mean 'procedure or function'. The term 'compilation unit' is defined in the TDS manual.

- The maximum number of identifiers in a compilation unit is 3000.
- The maximum number of dimensions of an array is 10.
- The maximum depth to which replicated PARs can be nested, and be usage checked, is 10.
- The maximum number of tags in a variant protocol is 256.
- The maximum number of functions which may be in scope at any point in a compilation unit is 1000.
- The maximum number of parameters to a procedure or function for which debugging information can be generated is 50. Without debugging information, the number of parameters allowed depends on their complexity, but is subject to a maximum of about 75.
- The maximum number of results which may be returned from a function is 20.
- The maximum number of options in a CASE statement is 256.
- The maximum number different library procedures which may be called within a procedure declaration is 200.
- The maximum number of procedures in an SC is 200.
- The maximum number of library entry points in a program being linked is 1000.
- The maximum number of libraries in a program being linked is 100.
- The maximum number of libraries which may be used within an SC is 50.

In addition to these fixed limits, some of the compiler's data structures are dependent on the amount of freespace available when the compiler is run. More space can be made available for these by clearing other code items from memory before running the compiler, or by running the TDS on a larger board. These are as follows:

- The size of an array which may be alias or usage checked.

- The size of a channel array which may be declared at configuration level.
- The total code size of a system which may be linked.

7 Known problems

This list contains the list of problems with the TDS documentation and software known at the time of release. This includes software bugs, and problems which users have encountered in using the system. The list of known problems with the system is divided up into the following categories:

1 The documentation

2 The development environment

3 The servers

4 The compiler utility set

5 The file handling utility set

6 The debugger

7 The tools (excluding the debugger)

8 The libraries

9 The examples and tutorial

Each software problem in the list below is followed by a number, which is an INMOS internal number for the problem reports on D700D. The number should be quoted in any correspondence with INMOS about the problem.

7.1 The documentation

1 **Section 8.3.** The parameter should be called:

`first.processor.is.boot.from.link`.

2 **CAUSEERROR.** The documentation does not describe the implicitly defined occam procedure `CAUSEERROR()` which sets the error flag, and which may be useful for debugging purposes.

3 **Control keys on the IBM.** The documentation does not state that the control keys of the NEC PC keyboard layout (for example, ctrl-Z for SUSPEND TDS) are also available on the IBM PC.

4 **CHAN OF ANY.** The documentation does not state that a channel declared to be `CHAN OF ANY` may be passed as an actual parameter to

a procedure with a formal channel parameter of any protocol (for example, `CHAN OF INT`, or a user-defined protocol). However, a channel declared with a protocol may not be passed as an actual parameter to a procedure whose formal parameter is declared as a `CHAN OF ANY`.

5 **Debugger example.** The worked example for the debugger in chapter 9 does not give exactly the same values for addresses as the software does when the example is worked through.

7.2 The development environment

1 **No disk space.** If disk space runs out when the compiler, or other utility, is running, the utility can deadlock. (2)

2 **Repeated parameter pop-up.** If a utility reads an already existing parameter fold from the toolkit, and one of the parameters is missing, it keeps on popping up the parameter fold, instead of supplying the missing parameter itself. Solution: type it in yourself, or abort the utility and delete the old parameter fold from the toolkit. (14)

3 **Server side effect on BREAK**. Running the TDS server has the side effect of setting the DOS `BREAK` state to `OFF`. (15)

4 **User filer uf.derive.file signal.** The user filer `uf.derive.file` command, when sent for a fold that is already filed causes the TDS to set error. (18)

5 **User filer uf.test.filed command.** The user filer `uf.test.filed` command should return an error value if it is used for an item which is not a fold; instead it returns `fsd.result` followed by `fi.not.filed`, just as it does on an unfiled fold. (33)

6 **User filer uf.unfile command.** The user filer command `uf.unfile` should return an error when used on an unfiled fold; instead it sets error. (43)

7 **SELECT PARAMETER behaviour.** The SELECT PARAMETER function may have unexpected behaviour on a badly-formed parameter line. (60)

8 **Failure to write toolkit.** There is no warning message given if the TDS fails to write out the toolkit file `TOPLEVEL.TKT` (for example, this will happen if the file is write-protected). (63)

9 **Disk full errors.** The TDS does not always recover correctly if it runs out of disk space. The system may deadlock, or the first filing system error may cause any further filing operations to fail giving messages such as

`File does not exist` or `File has incorrect format`. To recover from this, exit the TDS (do not suspend it), delete some files to make more space, and then reboot the TDS. (64)

10 **Autoload failure.** If an error occurs during autoloading (for example, if one of the code files does not exist), the error message only appears briefly, and it is not usually apparent that an error has occurred. (65)

11 **Core dump out of disk space.** If the TDS runs out of disk space while writing the core dump file, it gives a message but the name of the file is corrupted. (94)

7.3 The servers

No known problems.

7.4 The compiler utility set

1 **Library headers not checked.** When a library is validated, the compiler does not check the syntax of the text folds containing constant and protocol definitions, so errors in this text are not detected until the library is used in a program. (6)

2 **SEQ in a PROCESSOR statement.** Using `SEQ` in a `PROCESSOR` statement, followed by a sequence of procedure calls, gives the misleading error message `Processor number` *number* `already used`. (13)

3 **Text lines in library folds.** A library may not contain any text lines between the folds. (19)

4 **RECOMPILE parameters.** The $\boxed{\text{RECOMPILE}}$ utility displays a full compiler parameter fold, which may be confusing, since it normally only uses the `compile.all` and `force.pop.up` parameters, taking the other parameters from existing descriptor folds. (22)

5 **Configuration level libraries.** If using a header library at configuration level, there must be a logical library name translation for the name with the target type of `T4` (along with the error mode set in the parameter fold) even if the rest of the system contains only `T8` or `T2` processor types. (29)

6 **RECOMPILE on a PROGRAM** When a `PROGRAM` is compiled, the compiler parameters are not recorded in the descriptor, so on $\boxed{\text{RECOMPILE}}$ the parameters are taken from the parameter fold. (41)

7 **Non-existent library.** If a logical library name points to library files, but not for the required target type, the error message `Cannot open library file` is given rather than informing you that the logical name has not been set up correctly. (42)

8 **Incompatible protocols from multiple USEs.** If a compilation unit includes a number of `PROC` declarations, each preceded by a `#USE` of the same library, the compiler says that the protocols are incompatible when the procedures are called. (52)

9 **Using uncompacted libraries.** Using uncompacted libraries from another directory can lead to very obscure errors. The names of nested filed folds are interpreted as if they applied to the directory in which the library is being used, rather than the directory containing the source. If the current directory happens to include files with these names, they can be picked up instead of the library files, with unpredictable effects. (74)

10 **Compiler sets error again.** If the compiler sets error during a compilation, it may lead to a badly formed descriptor in a foldset which appears to be valid, leading the compiler to set error again when it is applied to the program. The data folds should be removed from the foldset before applying the compiler to the program again. (91)

11 **Functions in replicators.** Using a function call in a replicator base or count expression gives incorrect values for the replication. Solution: put the function call in an abbreviation before the replicator. (101)

12 **Assign to protocol tag sets error.** If a compilation unit includes an assignment to a protocol tag name (a meaningless operation) the compiler does not report it, but sets error instead. (103)

13 **More than 50 parameters sets error.** If a procedure declaration includes more than 50 parameters, then if `create.debugging.info` is `TRUE`, the compiler will set error. (104)

7.5 The file handling utility set

1 **Disk full.** When using the COPY IN, COPY OUT or COMPACT LIBRARIES utilities, if the disk becomes full, it is not possible to suspend the TDS, delete some files, return to the TDS and repeat the operation, as the TDS continues to be unable to write any files. It is necessary to reboot the TDS before the new disk space can be used. (38)

7.6 The debugger

1 **Debugger read-only mode.** Using the debugger, when located into a library, pressing R/W gives the misleading message `Read only mode is now set` instead of informing you that you can only be in read-only mode while viewing the source of a library. (20)

2 **Debugger handling of non-local arrays.** The debugger is unable to deduce the size of some abbreviations of non-local arrays, where the size of the abbreviated array is not specified either in the local abbreviation or in the previous specification. (106)

7.7 The tools

1 **Disassembler output too big.** The disassembler program can produce a file which is too big to be read with the TDS. It must be listed to a host file using the lister program before it can be read. (5)

2 **Lister filename handling.** The lister program is unable to deal with a DOS filename beginning `..` (meaning the parent directory). It is truncated to a zero-length name. (16)

7.8 The libraries

1 **Functions exp and Dexp.** The elementary function libraries include the functions `exp` and `Dexp` for internal use, but the names are not listed in the descriptions of these libraries.

7.9 The examples and tutorial

1 **Example 3 STOPs.** Example 3 has a `STOP` on error which makes it likely to set error when not run on a fold bundle. (44)