

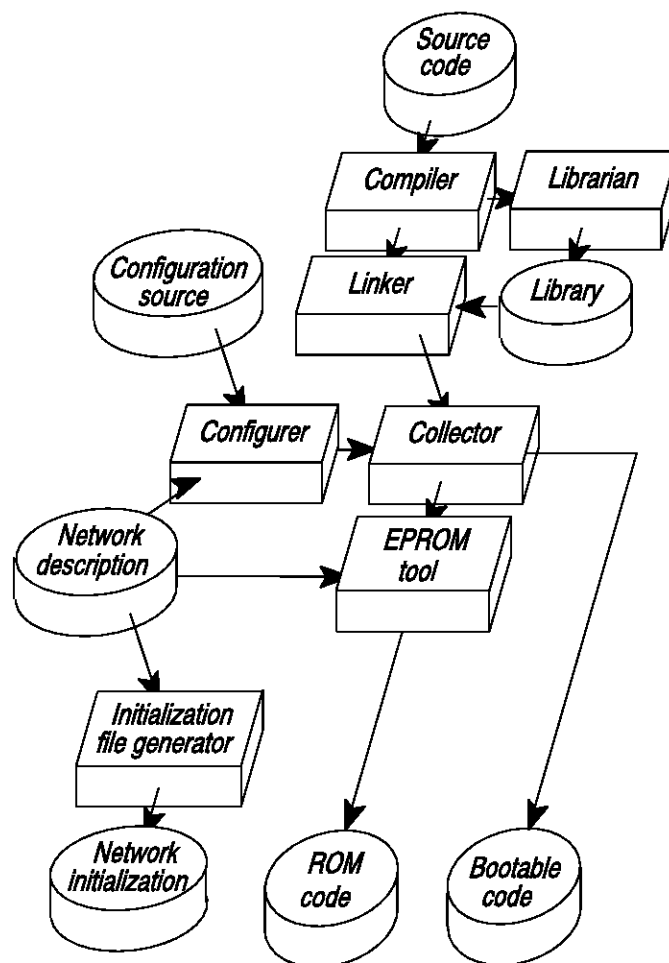
T9000 software development tools
FEATURES

- H Complete ANSI C Toolset for IMS T9000 transputer networks
- H Source compatible with T2/T4/T8 and ST20 development tools
- H Cross-development from Sun 4 and 386 PC platforms
- H ANSI C compiler for IMS T9000 fully compliant with X3.159-1989 standard
- H Global and local optimization
- H Code generation for IMS T9000 instruction set and pipelined CPU
- H Assembler
- H Configuration tools that exploit new communications architecture
- H Software routing used for networks without routing chips
- H Support for mixing ANSI C and occam 2
- H Support for communication between T2/T4/T8 networks and IMS T9000 networks
- H Support for assembler inserts
- H Support for EPROM programming

DESCRIPTION

The SGS-THOMSON T9000 ANSI C Toolset supports the construction of parallel C programs which may be loaded onto IMS T9000 networks via a link, or put into a form suitable for booting from ROM.

The Toolset also includes the hardware configuration tools required to initialize IMS T9000 networks, fully supporting the use of STC104 packet routing switches including initialization and labelling of networks.

PRODUCT INFORMATION


Contents

1	Introduction	3
	1.1 Key features	3
	1.2 Overview	4
	1.3 Hosts	5
2	IMS T9000 networks	5
	2.1 The IMS T9000 family of devices	6
3	Parallel programming	7
4	Compiler and compilation tools	9
	4.2 Optimization	11
	4.3 Use of IMS T9000 features	12
	4.4 TCOFF	13
	4.5 Separate compilation	13
	4.6 Mixed language programming	13
	4.7 Assembler code	13
5	Libraries	14
6	Configuration, initialization and loading	15
	6.1 Network description	17
	6.2 Memory configuration	17
	6.3 Initializing and loading	17
	6.4 Configuration language	18
	6.5 ROM support	20
7	Product components	21
	7.1 Tools	21
	7.2 Libraries	21
	7.3 Sources	21
	7.4 Documentation	21
8	Product variants	21
	8.1 Sun-4 product	21
	8.2 PC product	22
9	Problem reporting and field support	23
10	Ordering information	23

1 Introduction

This document contains product information for the T9000 ANSI C Toolset. Toolsets are available for IMS T2xx, T4xx and T8xx series transputers. Within this document the IMS T2xx, T4xx and T8xx transputers are called 'T2/T4/T8' transputers.

The T9000 ANSI C Toolset provides a complete ANSI C cross-development system for IMS T9000 transputer network targets. It can be used to build sequential or parallel programs for single IMS T9000 transputers or for multi-transputer IMS T9000 networks.

The Toolset supports the construction of parallel C programs, which may be loaded onto IMS T9000 networks via a link or put into a form suitable for booting from ROM. The Toolset also includes the hardware configuration tools required to initialize IMS T9000 networks. It fully supports the use of STC104 packet routing switches including labelling and initializing.

1.1 Key features

- S Full ANSI C cross-compiler for IMS T9000 target networks;
- S Compiler fully compliant with X3.159-1989 standard;
- S Code generation for IMS T9000 instruction set and pipelined CPU;
- S Excellent compile time diagnostics;
- S Global and local optimization;
- S Support for parallelism;
- S Assembler;
- S Support for assembler inserts;
- S Listings of where variables and functions reside in memory;
- S Small runtime overhead;
- S Separate compilation, using linker and librarian tools;
- S Tools for creating and loading multi-processor programs;
- S Exploitation of the new communications architecture by the configuration tools;
- S Software routing of channels for networks without routing chips;
- S Automatic makefile generator;
- S Mixed language programming support;
- S Tools to support preparation of programs for EPROM;
- S Consistent tools across PC and Sun-4 hosts;
- S Support for dynamically loading programs and functions.

1 Introduction

1.2 Overview

The IMS T9000 transputer range of devices is supported by the following software products:

- S IMS Dx394 T9000 ANSI C Toolset;
- S IMS Dx395 T9000 occam 2 Toolset;
- S IMS Dx390 T9000 INQUEST development environment, containing debugging and profiling tools;
- S IMS Dx397 interface software.

The T9000 ANSI C Toolset is source compatible with T2/T4/T8 and ST20 Toolset products. This has been achieved by using similar compiler, libraries, linker, configurer and other tools. In addition, code written for a single IMS T4xx or IMS T8xx can be ported directly to an IMS T9000 for performance enhancement.

The IMS T9000 offers two important enhancements; the new communications architecture and the new pipelined CPU. In order to take full advantage of these features, a new Toolset has been developed which will function alongside the existing Dx314 ANSI C Toolset.

The configuration tools are the main area in which the difference between the IMS T9000 and the T2/T4/T8 range is apparent to the programmer. Configuration is the process of mapping a multi-process network application to a transputer network. Tools are provided in the T9000 ANSI C Toolset allowing users to:

- S describe a hardware network made up of IMS T9000 and STC104 packet routing switch devices;
- S define values for the user-programmable attributes of the devices (such as the IMS T9000's programmable memory interface, and the STC104 packet routing switch's interval labelling registers);
- S define how processes in the application should be mapped to processors in the network.

The tools allow the full user-programmable functionality of the IMS T9000 and STC104 devices to be used with simple textual descriptions of attribute values, without the need to write any low-level configuration code.

The configuration tools can check the network description to ensure that the network is properly connected. Applications of arbitrary connectivity can be mapped onto the hardware network. Where the network includes STC104 packet routing switches, the headers required to connect the channels between processors are calculated automatically by the tools. If the network does not contain STC104 packet routing switches, the tools provide software through-routing so that channels may still be connected between processors which are not neighbors.

EPROM tools are provided to support the creation of initialization ROMs for IMS T9000 transputers, and the creation of system ROMs containing application code.

The debugger supplied in the INQUEST development environment supports source-level and low-level debugging of multi-process and multi-processor programs. The debugger runs on the host computer from which the IMS T9000 network has been loaded. A graphical user interface, with multiple windows, simplifies use of this sophisticated tool. INQUEST also includes profiling tools for detecting program 'hot-spots' and for evaluating how effectively parallel programs use a network of processors.

1.3 Hosts

Programs developed using the Toolset are both source and binary compatible across all host development machines. The T9000 ANSI C Toolset is available for the following development platforms:

IMS D4394 T9000 ANSI C Toolset for Sun-4 under SunOS and Solaris,

IMS D7394 T9000 ANSI C Toolset for IBM PC under MS-DOS.

2 IMS T9000 networks

The new generation transputer, the IMS T9000, provides major enhancements to the communications capabilities of transputer networks. The T9000 ANSI C Toolset provides support for these features in the form of new configuration and initialization tools.

Using the IMS T9000 transputer and the T9000 ANSI C Toolset, a large network of processes and channels may be programmed quite independently of the transputer network on which it will be implemented. The IMS T9000 can multiplex a large number of channels, called virtual channels, onto a single physical link. This allows a large network of channels between processes to be efficiently implemented on small IMS T9000 networks without routing or multiplexing software.

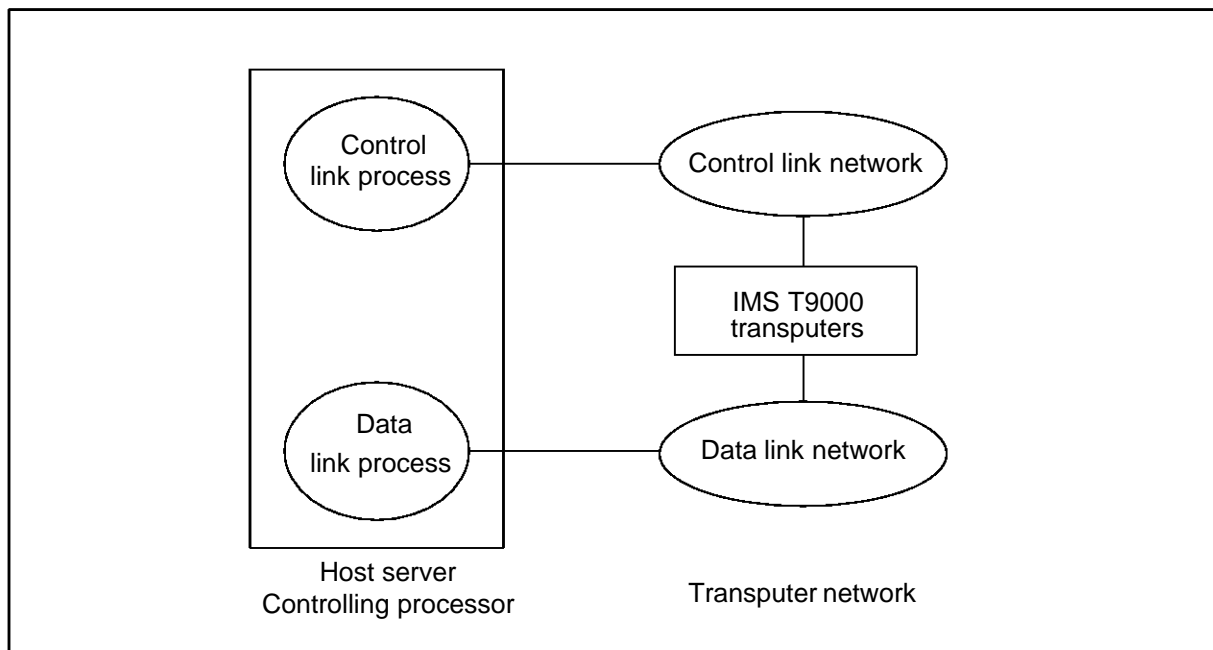


Figure 1 IMS T9000 network architecture

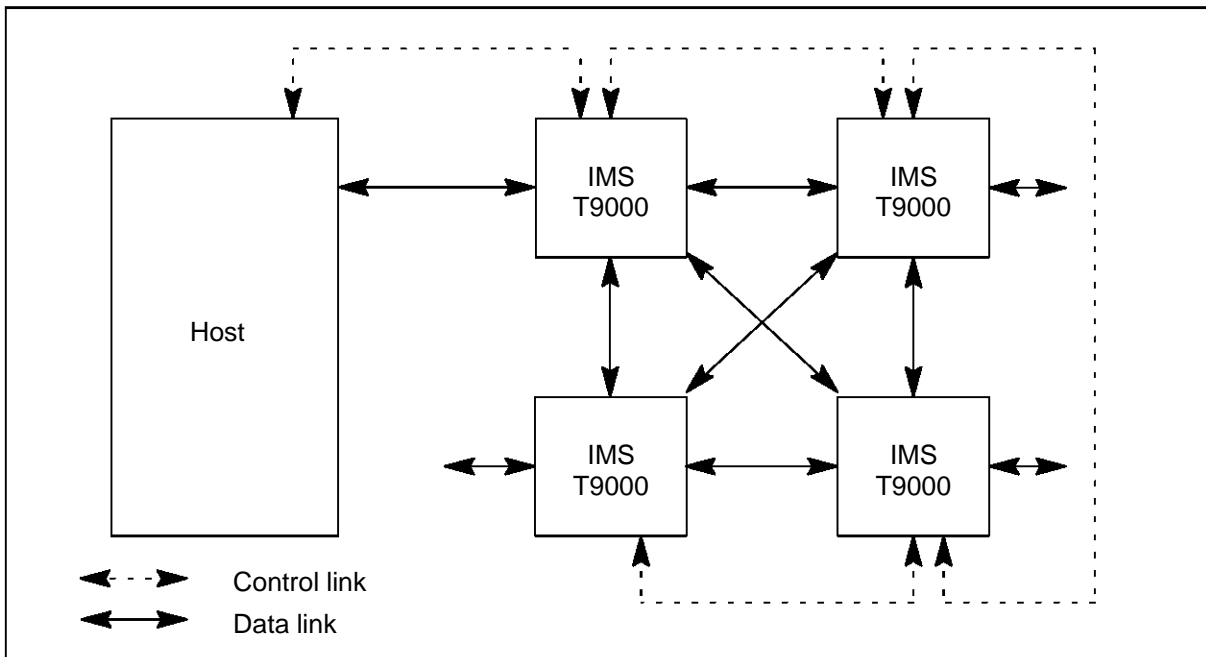


Figure 2 IMS T9000 network example

This hardware multiplexing support is complemented by a software through-routing mechanism provided by the T9000 ANSI C Toolset. The combination of hardware multiplexing and software through-routing allow any user-defined channel connections to be specified and implemented with optimal flexibility.

The STC104 packet routing switch gives higher communications performance in larger systems. It is a fast packet router which can route each incoming packet out along any one of its links, depending on the packet header added to the data by the sending transputer. It enables direct connections between the transputers in a large network and can remove the need for software through-routing. Large networks may also be implemented without packet routing switches, in which case each IMS T9000 may have a direct connection to up to four other devices.

An IMS T9000 transputer system has a network of links for data communications plus a separate network of control links for system initialization and debugging control. An IMS T9000 network can be viewed as a number of processors each of which is connected to both the data network and the control network. During system development, both networks should also be connected to the host or other controlling processor, as shown in Figure 1.

2.1 The IMS T9000 family of devices

The T9000 ANSI C Toolset supports the full family of the IMS T9000 transputer and the associated STC104 packet routing switch. The STC104 packet routing switch is a 32-way packet router.

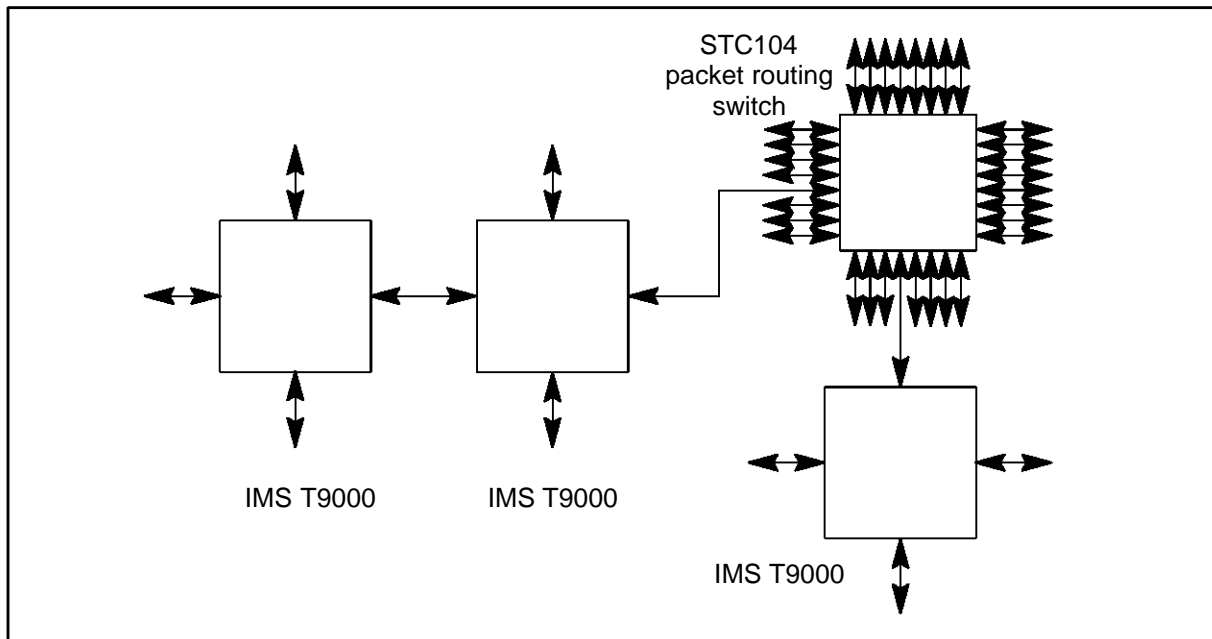


Figure 3 IMS T9000 family of devices

3 Parallel programming

The T9000 ANSI C Toolset supports parallelism on individual transputers and parallelism across networks of IMS T9000 transputers.

The transputer programming model consists of parallel processes communicating through channels. Channels connect pairs of processes and allow data to be passed between them. Each process can itself be built from a number of parallel sub-processes, so that an entire software system can be described as a process hierarchy. Processes may be created at high and low priority levels. Interrupt routines are typically implemented as high priority processes. This model is consistent with many modern software design methodologies.

By using library calls, parallel processes may be created dynamically. Processes may be created individually in which case the function call will return immediately with the called process executing concurrently with the calling process, or created as a group, in which case the function will return when all the processes within the group have completed. Processes communicate by message passing over channels.

Figure 5 shows a collection of four processes communicating through channels. The `mux` process communicates with a host computer and hands out work to be done to one of three `worker` processes. Results from the workers are then returned to the host by the `mux` process. The following example shows how this collection of processes can be described in C to run on a single IMS T9000 transputer. Section 6.4 shows how the parallel processes in the program can be mapped onto a network of processors using the configuration language.

The code in Figure 4 illustrates how to program the collection of parallel processes shown diagrammatically in Figure 5. The functions `fmux` and `fwkr` contain the executable code of the processes `mux` and `worker` respectively. These processes communicate using channels.

3 Parallel programming

```
/* Example program */

#include <process.h>
#include <misc.h>
#include <stdlib.h>

/* Define prototypes for process functions */

void fmux (Process*, Channel*, Channel*, Channel*[], Channel*[], int);
/* process, hostin, hostout, in, out, no_workers */

void fwkr (Process*, Channel*, Channel*, int);
/* process, in, out, worker_id */

/* Main program */

int main (int argc, char *argv[], char *envp[])
{
    int i;

    /* Declare processes and channels */
    Channel *hostin, *hostout, *wkrIn[3], *wkrOut[3];
    Process *mux, *worker[3];

    /* Allocate channels */
    hostin = get_param(1);
    hostout = get_param(2);
    for (i = 0; i < 3; ++i) {
        wkrIn[i] = ChanAlloc ();
        wkrOut[i] = ChanAlloc ();
    }

    /* Allocate processes */
    mux = ProcAlloc (fmux, 0, 5, hostin, hostout, wkrIn, wkrOut, 3);
    for (i = 0; i < 3; ++i) {
        worker[i] = ProcAlloc (fwkr, 0, 3, wkrIn[i], wkrOut[i], i);
    }
    /* Start the processes running in parallel */
    ProcPar (mux, worker[0], worker[1], worker[2], NULL);

    exit (0);
}
```

Figure 4 Parallel processes

The ANSI C compiler in the Toolset can be used to compile the program shown in Figure 4. The compiled code can then be linked, configured and collected to produce a code file to run on a single transputer. Alternatively it may be preferred to distribute the processes over a network of more than one processor, in which case the code for the `mux` and `worker` processes must be compiled and linked separately. The linker produces processes in the form of fully linked units. A network of fully linked processes can be distributed over a network of processors using the configuration tools. The code shown in Figure 4 would be replaced by a configuration description, as described in Section 6.4.

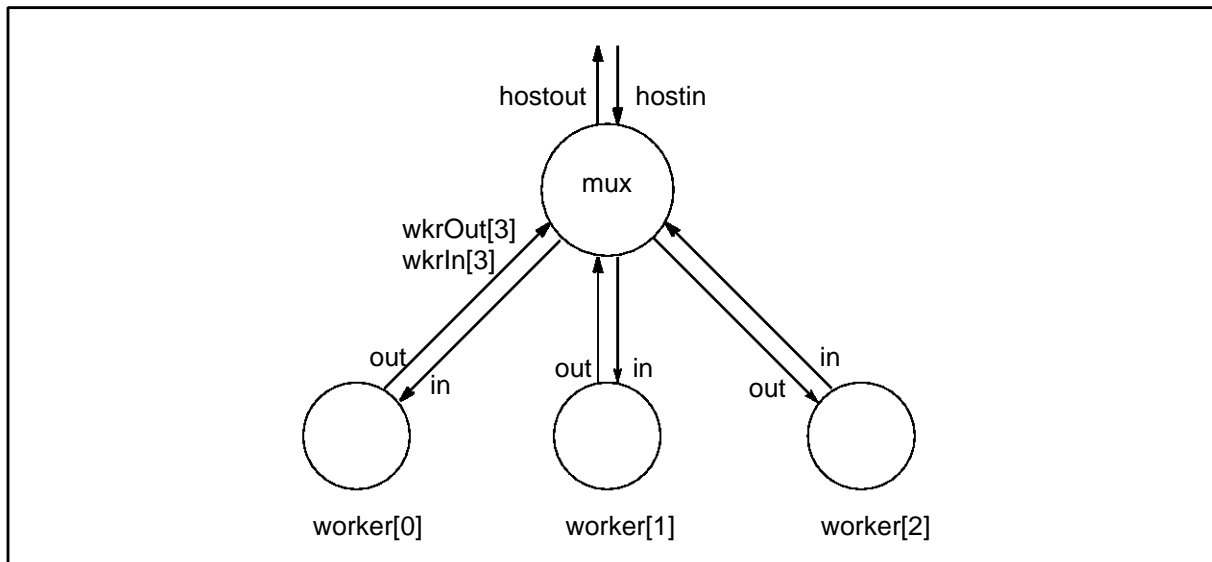


Figure 5 Software network

Each process has its own stack space (typically allocated from the heap of the main process) but the processes in a single linked unit share static space, heap space and code (including libraries). Processes created in this way can communicate by message-passing over channels or by shared data (optionally protected by semaphores or channels).

Functions are provided to read a message from one of a list of channels (implementing the OCCAM **ALT** construct), to time-out on channel input and to access the high and low resolution timers built into the IMS T9000 transputer. The transputer's hardware scheduler provides extremely efficient scheduling of these processes and efficiently implements many features which would normally require a real-time executive.

The compiler operates from a host command line interface. The pre-processor is integrated into the compiler for fast execution. The compile time diagnostics provided by the compiler are excellent. These include type checking in expressions and type checking of function arguments.

The tools integrate into the host operating system build utilities, allowing, for example, the use of standard editor, make and source code and configuration control utilities.

4 Compiler and compilation tools

4.1 ANSI C conformance

The T9000 ANSI C Toolset supports the full standard language as defined in X3.159-1989. The key extensions and functions which are in the ANSI standard but not in the original definition of C by Kernighan and Ritchie are:

- S Prototypes to define function parameters.
- S Better definition of the pre-processor.

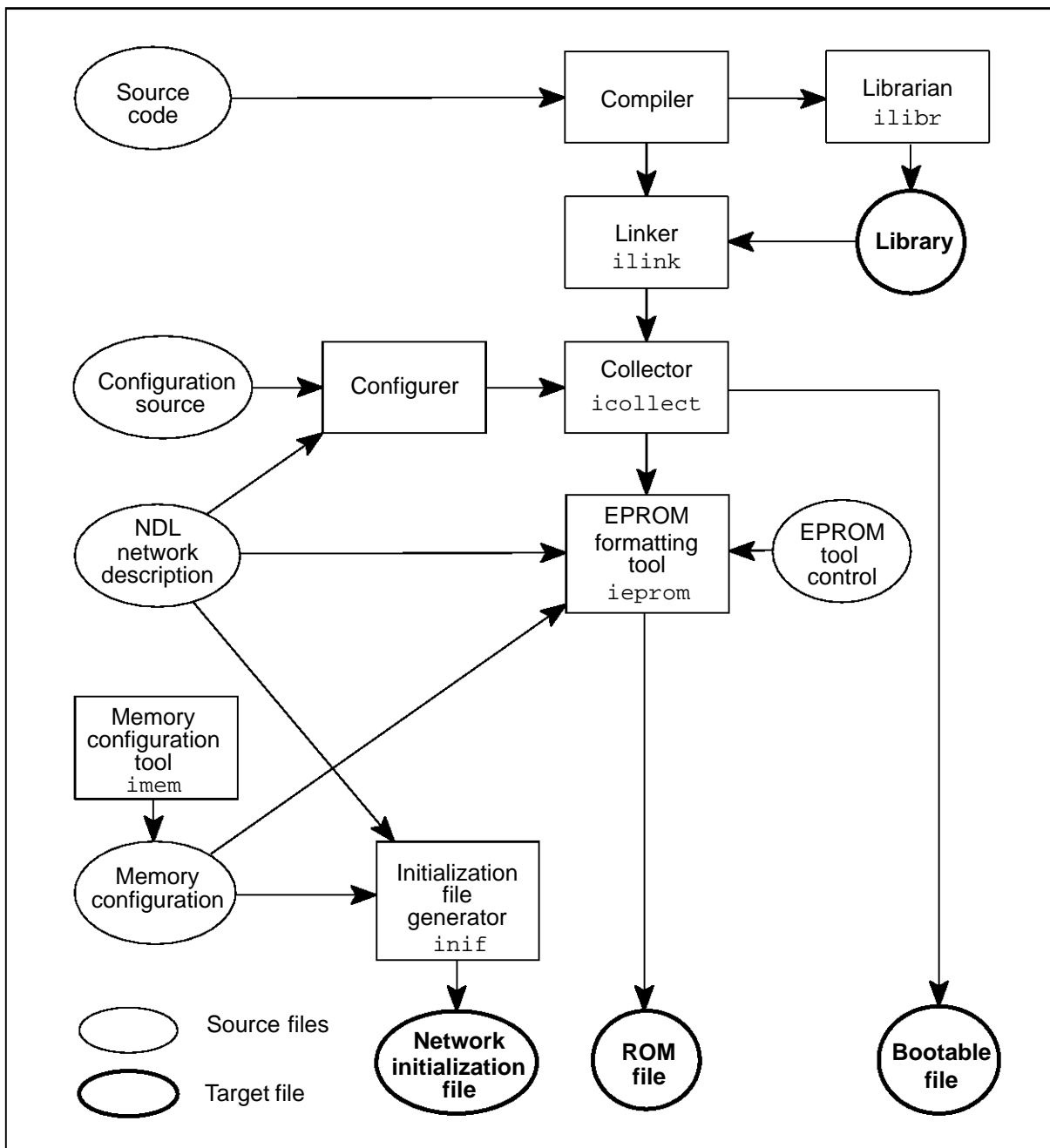


Figure 6 Program build model for the T9000 ANSI C Toolset

- S Longer identifiers. The standard specifies that at least 31 characters are significant in identifiers and six characters for external names. The ANSI C Toolset implementation allows arbitrarily long identifiers, the first 256 characters of which are significant.
- S Trigraphs introduced for use by some character sets which do not have some of the normal C characters.
- S Data items which can be declared as **const** or **volatile**.
- S Support for special character sets (including Asian ones).

- S Enumeration types.
- S Implementation limits which are accessible from header files.
- S Structures which can be assigned, passed to functions and returned from them.

The compiler passes all the tests in the validation suite from Plum Hall. The compiler will be validated by the British Standards Institution (BSI). The validation is recognized across Europe by the French (AFNOR) and Italian (IMQ) Standards Institutes. The standards authority in Japan (JMI) has also agreed to harmonize their C validation with Europe.

4.2 Optimization

The compiler implements a wide range of local code optimization techniques.

Constant folding. The compiler evaluates all integer and real constant expressions at compile time.

Advanced workspace allocation. Frequently used variables are placed at small offsets in workspace, thus reducing the size of the instructions needed to access them and ensuring that the workspace cache of the IMS T9000 is employed for frequently used variables. This therefore increases the speed of execution.

Dead-code elimination. Code that cannot be reached during the execution of the program is removed.

Peep-hole optimization. Code sequences are selected that are the fastest for the operation. For example, single precision floating variables are moved using the integer move operations.

Instruction scheduling. Where possible the compiler exploits the internal concurrency of the transputer.

Constant caching. Some constants have their load time reduced by placing them in a constant table.

Unnecessary jumps are eliminated.

Switch statements. The compiler can generate a number of different code sequences tailored to cover the dense ranges within the total range.

Special idioms that are better on transputers are chosen for some code sequences.

Globally optimized code generation

When global optimizations is selected, the compiler extends the types of optimizations it performs to global techniques. These have typically given a 15%-25% improvement in speed over the local optimizations, as measured by a suite of internal benchmarks.

Common sub-expression elimination removes the evaluation of an expression where it is known that the value has already been computed; the value is stored

4 Compiler and compilation tools

in a temporary local workspace. This improves the speed of a program and reduces code size.

Loop invariant code motion can move the position where an expression is evaluated from within a loop to outside it. If the expression contains no variables that are changed during the execution of a loop, then the expression can be evaluated just once before the loop is entered. By keeping the result in a temporary location, the speed of execution of the whole loop is increased.

Tail-call optimization reduces the number of calls and returns executed by a program. If the last operation of a function is to invoke another function and immediately return the value therefrom, then the compiler attempts to re-use the same workspace area by just jumping to (rather than calling) the lower level function. The called function then returns directly to where the upper level function was called from. In the case where the call is a recursive call to the same function, then the workspace is exactly the right size, and a saving is also made because the stack adjustment instructions are no longer needed either. This optimization saves time and total stack space.

Workspace allocation by coloring reduces the amount of workspace required by using a word for two variables when it can be determined that they are not both required at the same time. In addition the variables that are most frequently used are placed at lower offsets in workspace. This reduces the number of prefixes needed to access the variables and so reduces the total code size.

The optimizing compiler also implements a pragma, `IMS_nosideeffects`, whereby the user can indicate that a function does not have any side-effects on external or static variables. The optimizer can then use this information to make assumptions about what state can be changed by a function invocation and hence about the validity of any previously computed expressions.

To make the access to some of the transputer's instructions even more effective, a number of special library functions have been defined which the optimizing compiler can render as in-line code. This removes the overhead of a library call, but it also gives the optimizer more information on what the program is doing.

Normally, when the optimizer sees a function containing some assembler code, it must make very conservative assumptions about the effect the code has on its surroundings, e.g. on static variables and parameters. By using the functions defined to access the instructions, the optimizer knows exactly what the effects will be and can make the minimally correct assumptions for the side-effects of the code.

The transputer instructions that can be accessed in this way include block moves, channel input and output, bit manipulation, CRC computations and some scheduling operations.

4.3 Use of IMS T9000 features

Programs are compiled to run as L-processes (with a local trap handler) on the IMS T9000. A trap handler is set up to catch and handle program runtime errors. Signal handlers can be set up to handle certain run-time errors.

The code generated by the compiler and in the supporting libraries, makes full use of the new features introduced to the transputer instruction set in the IMS T9000. In particular:

- S The part-word support instructions are used to improve the handling of 8-bit and 16-bit integers.
- S The new floating point instructions, *fpsqrt* and *fprem* are used.
- S `volatile` variables are implemented using the device access instructions.
- S The semaphore support in the runtime library is implemented using the new semaphore instructions.
- S Care has been taken in the code generation and optimization to ensure that effective use is made of the IMS T9000's workspace cache and CPU pipeline.

Access to the full IMS T9000 instruction set is available through assembly inserts.

4.4 TCOFF

The binary code produced by T9000 ANSI C Toolset tools is in Transputer Common Object File Format (TCOFF). This allows integration with other TCOFF-based utilities, including the T9000 occam 2 Toolset and the INQUEST development environment.

4.5 Separate compilation

Collections of subprograms can be compiled separately using the ANSI C compiler and optionally combined into a library. The linker is used to combine separately compiled functions and procedures as a linked unit. A single copy of a linked unit cannot be distributed over more than one processor. The linker supports selective loading of library units.

4.6 Mixed language programming

The T9000 ANSI C and occam 2 compilers are fully compatible and allow simple mixing of languages. OCCam and C processes may be freely mixed when configuring a program for a single transputer or a network of transputers. Such processes will run in parallel and communicate using channels.

The T9000 ANSI C Toolset allows OCCam 2 procedures and single-valued occam 2 functions to be called from C just like other C functions. Pragmas are provided to tell the C compiler not to generate the hidden static link parameter (which is required by C but not by OCCam) and to change the external names of procedures and function, since OCCam names may not be legal C function names.

Similarly, the T9000 occam 2 Toolset supports calling C functions directly from OCCam. C functions which require access to static variables are supported.

4.7 Assembler code

The T9000 ANSI C Toolset provides a very powerful assembler insert facility. Assembler code can be written at any point in the source code to achieve direct access to transputer

5 Libraries

instructions for reasons of speed or code size. Full access is available to the IMS T9000 transputer instruction set and C program variables.

The assembler insert facility supports:

- S Access to the full instruction set of the IMS T9000 transputer
- S Symbolic access to automatic and static variables
- S Pseudo-operations to load multi-word values into registers
- S Loading results of expressions to registers using the pseudo-operations.
- S Labels and jumps
- S Directives for instruction sizing, stack access, return address access etc.
- S The ANSI C pre-processor may be used to provide macro assembly programming facilities.

If there is no other way to obtain the code required, for example when writing customized bootstrap mechanisms, then the final stage of the compiler may be invoked as an assembler to assemble user-written assembler code.

5 Libraries

The full set of ANSI C libraries is provided, as specified in the ANSI standard.

The standard mathematics library operates in double precision. Versions of the mathematical functions are provided that operate on float arguments and return float values. These libraries provide improved performance for applications where performance requirements override accuracy requirements.

The standard C mathematical functions provided use the same code as the OCCam libraries. This ensures identical results and accuracy for all T2/T4/T8, ST20 and T9000 compilers supported by SGS-THOMSON. A reduced C library is supplied to minimize code size for embedded systems applications and for processes which do not need to access host operating system facilities.

Extra libraries are provided to support parallel programming and to provide access to the special features of the transputer. Functions are provided to create parallel processes dynamically. Processes may be created individually in which case the function call will return immediately with the called process executing concurrently with the calling process, or created as a group, in which case the function will return when all the processes within the group have completed.

Functions are provided to support communications between processes by message passing over channels. Any form of data may be sent or received on a single channel. Input from a list of channels is also supported. Library functions provide access to the high and low resolution timers built into the transputer, allowing any process to read the current timer value or delay for a specified time. Channel inputs may also be timed out. Support is also provided for semaphores.

Support is given for loading code dynamically. Miscellaneous library functions are also provided, including functions to identify the type of host, the host link and the boot link.

6 Configuration, initialization and loading

Configuration is the process of defining how an application program is to be run on the available hardware. Given a description of the hardware network, the software network of an application and the mapping between them, the T9000 ANSI C Toolset produces a *bootable file* which can be sent to the network for execution. The hardware network description is also used to prepare a *network initialization file* which is used to initialize the network and bootstrap the processors into a state where they are ready to receive the bootable file.

A configuration description file is used as input to a tool known as the *configurer*. The description is written in a C-like language which is upwards compatible with the configuration language in the Dx314 Professional ANSI C Toolset. The configuration description describes the application as a network of processes and channels. It indicates which linked files should be used as the code for each process, and it also indicates on which processor each process is to be run.

The hardware description is separate from the rest of the configuration description. It is written in the Network Description Language (NDL). The configuration description refers to the NDL description of the hardware by means of a `#network` directive. The hardware description describes the processors and routing devices in the network and their connections.

The main features of the configuration tools are as follows:

- S To initialize, boot and load an IMS T9000 network, it is not necessary to write any low-level code; instead the attributes of devices are specified by simple textual statements in the network description, and the low level code is generated automatically by the tools.
- S The network description for a particular network can be written once, when the hardware is designed, and does not normally need to be changed between different applications.
- S The configuration tools will check the correctness of the network description; this includes checking the labelling of the routing chips for deadlock freedom.
- S Channels specified by the user in the configuration description are automatically mapped by the tools. In a system with routing devices, the configuration tools calculate the routing headers required for the user's channels. If there are no routing devices in the system, the configurer adds any software through-routing mechanisms required. Thus the software network is not constrained by the topology of the hardware network.
- S For a particular application, the tools can produce either a bootable file which can be used to load the network from a host, or a boot-from-ROM file which can be used to program a ROM on one of the processors in the network.

6 Configuration, initialization and loading

```
#INCLUDE "stdndl.inc"

-- Name of file with memory timing information, etc.
VAL memconfig IS "T9000.mem" :

VAL n          IS 4 :                -- Number of transputers
VAL SysMem     IS 2*K :              -- Size of system RAM
VAL UsrBase    IS #80000000 :        -- Start address of user RAM
VAL UsrMem     IS (2*M) - SysMem :    -- Size of user RAM
VAL SysBase    IS UsrBase+UsrMem :    -- Start of system RAM
                                           -- (= top of user RAM)
VAL memoryflags IS [[SysBase, SysMem, RAM + SYSTEM],
                    [UsrBase, UsrMem, RAM + USER]] :

[n]NODE p :                          -- n transputers
CONTROLPORT host :
ARC          hostlink :

NETWORK fourT9
DO
  -- set link speeds
  SET DEFAULT (link.speed.multiply := 10)
  SET DEFAULT (link.speed.divide   := [1])
  SET DEFAULT (control.speed.divide := [8])

  -- set processor types
  DO i=0 FOR n
    SET p[i] (type := "T9000")
  SET p[0] (root := TRUE)
  DO i=1 FOR n-1
    SET p[i] (root := FALSE)

  -- set memory information
  DO i=0 FOR n
    SET p[i] (memconfig, memory := memconfig, memoryflags)

  -- connect control network
  CONNECT host[control] TO p[0][control.up]
  DO i=0 FOR n-1
    CONNECT p[i][control.down] TO p[i+1][control.up]

  -- connect data links
  CONNECT host[data] TO p[0][link][0] WITH hostlink
  DO i=0 FOR n-1
    CONNECT p[i][link][1] TO p[i+1][link][2]
  CONNECT p[n-1][link][1] TO p[0][link][2]
  DO i=0 FOR n-2
    CONNECT p[i][link][3] TO p[i+2][link][0]
:
:
```

Figure 7 NDL network description example

- S Initialization of individual IMS T9000 devices in the network (for example, setting up the external memory interface) can be done over the control link, or from an initialization ROM local to each IMS T9000. Support for generation of local initialization ROMs is included.

6.1 Network description

The Network Description Language (NDL) is used to describe the available hardware – the types of processors, their attributes and how they are connected. Processor attributes include, for example, a description of its memory map and its link speeds. The NDL also describes any packet routing switch devices in the network and their attributes. Attributes of a routing device include the *labelling* of the routing device, which indicates how packets from processors should be routed through it.

The network description will not normally change unless the hardware is changed. This NDL description of the system is used by the tools for a variety of purposes, from initializing the hardware to mapping application code onto processors. These tools can either read and check the NDL source directly or read a binary version produced by the NDL compiler, `indl`.

Figure 7 shows the network description for the example network shown in Figure 10.

6.2 Memory configuration

The IMS T9000 programmable memory interface supports a wide variety of memory configurations. It can be set up to provide the appropriate signals and timing needed by the memory being used. The parameters to initialize the memory interface may be sent to the transputer using the control link or may be included in the bootstrap code in ROM.

The memory configuration tool, `imem`, is used to assess and define memory interface parameters for each IMS T9000. It can be run interactively or in batch mode. The output may be incorporated into the network initialization file or into a ROM. The memory configuration description will not normally change unless the hardware is changed. `imem` can also be used to generate memory interface timing documentation.

6.3 Initializing and loading

A hosted IMS T9000 network is initialized by first sending control commands to the transputers and routers connected to the control network and then loading code using the data network.

The correct sequence of control network commands is held in a network initialization file, which contains all the information needed to initialize a system through the control links prior to loading the application code. It can be automatically generated by the initialization file generator tool, `inif`, from the network description and memory configurations. To initialize the network, the network initialization file is used by the initialization software to generate the correct sequence of commands to send to the control link network.

The bootstrap code for each processor, loading code and user application code are all incorporated in the bootable file with the necessary routing information. The bootable

6 Configuration, initialization and loading

file is generated automatically by the collector, `icollect`, from the configuration information and linked application code files. To load the code onto the network, the bootable file is sent down the data link to the data link network.

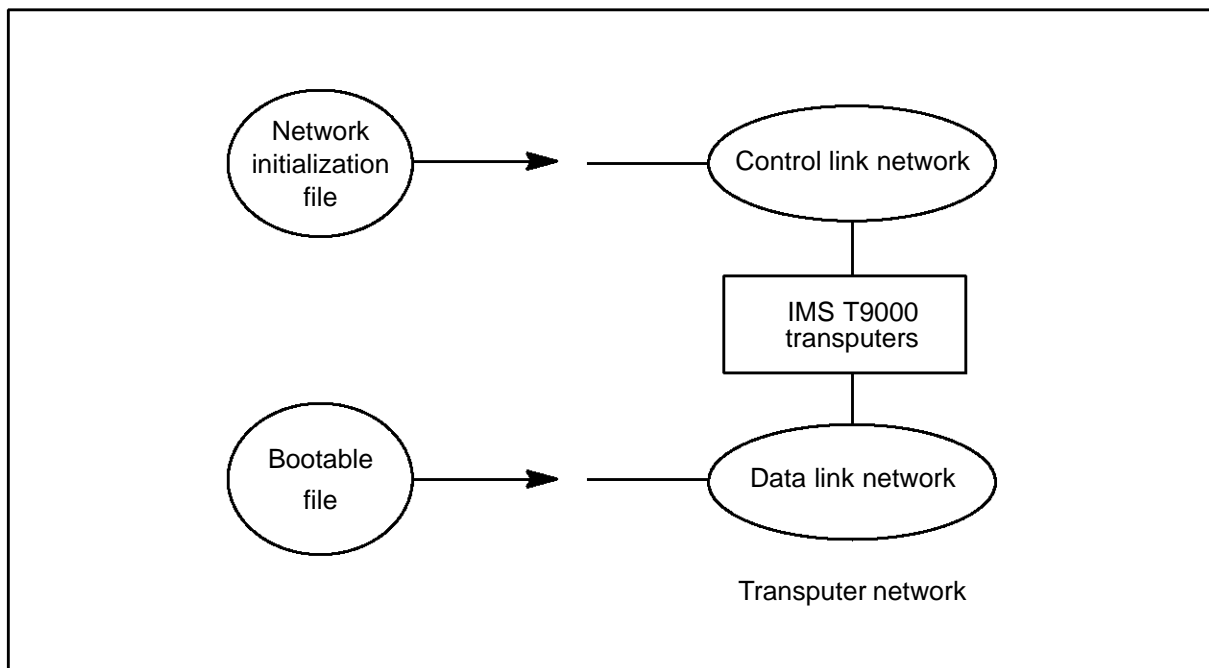


Figure 8 Initializing a hosted IMS T9000 network

6.4 Configuration language

A C-like configuration language is used to describe the network of processes and channels and the mapping of the processes onto the transputer network. The transputer network is described separately in the network description file. Multiple processes may be mapped onto the same transputer. The network description file is referred to by means of a `#network` configurer directive. This allows the user to map processes in the configuration description onto the processors named in the network description file. The routing of channels may be generated automatically or may be included in the configuration description.

```

/* Configuration example */

/* Hardware description */

#network "four9.ndl"

/* Software description */

/* Define process memory sizes and interfaces */
process (stacksize = 2K, heapsize = 16k); /* Define defaults */
rep i = 0 for 3
    process (interface (input in, output out, int id)) wkr[i];
process (interface (input hostin, output hostout,
                    input in[3], output out[3])) mux;

/* Define external channels, interconnections and parameters */
input hostinput; /* Host channel edges */
output hostoutput;
connect mux.hostin to hostinput; /* Host channel connections */
connect mux.hostout to hostoutput;
rep i = 0 for 3
{
    worker[i] (id = i); /* Set worker process id parameter*/
    connect mux.in[i] to wkr[i].out;
    connect mux.out[i] to wkr[i].in;
}

/* Mapping description */

/* Define linked file units for processes */
use "mux.lku" for mux;
rep i = 0 for 3
    use "wkr.lku" for worker[i];

/* Map processes to processors and external channels to edges */
rep i = 0 for 3
    place worker[i] on p[i+1];
place mux on p[0];
place hostinput on host;
place hostoutput on host;

```

Figure 9 Configuration

The following example illustrates just how easy it is to configure a program for transputers. Instead of using the top level single transputer code shown in Figure 4, it may be preferred to distribute the program over a network of processors, as shown in this example.

Figure 9 shows the configuration text for describing the software network of processes and channels and mapping the software onto the hardware shown in Figure 10 and described in NDL in Figure 7. The `mux` and `worker` processes in the software network have been compiled and linked into the files `mux.lku` and `wkr.lku` respectively. The NDL shown in Figure 7 is in the file `four9.ndl`. The software description in the configuration text replaces the top level of code shown in Figure 4. In this example the

6 Configuration, initialization and loading

mux process runs on the root transputer, called $p[0]$, while the individual worker processes run one on each of the transputers labelled $p[1]$ to $p[3]$ in Figure 10.

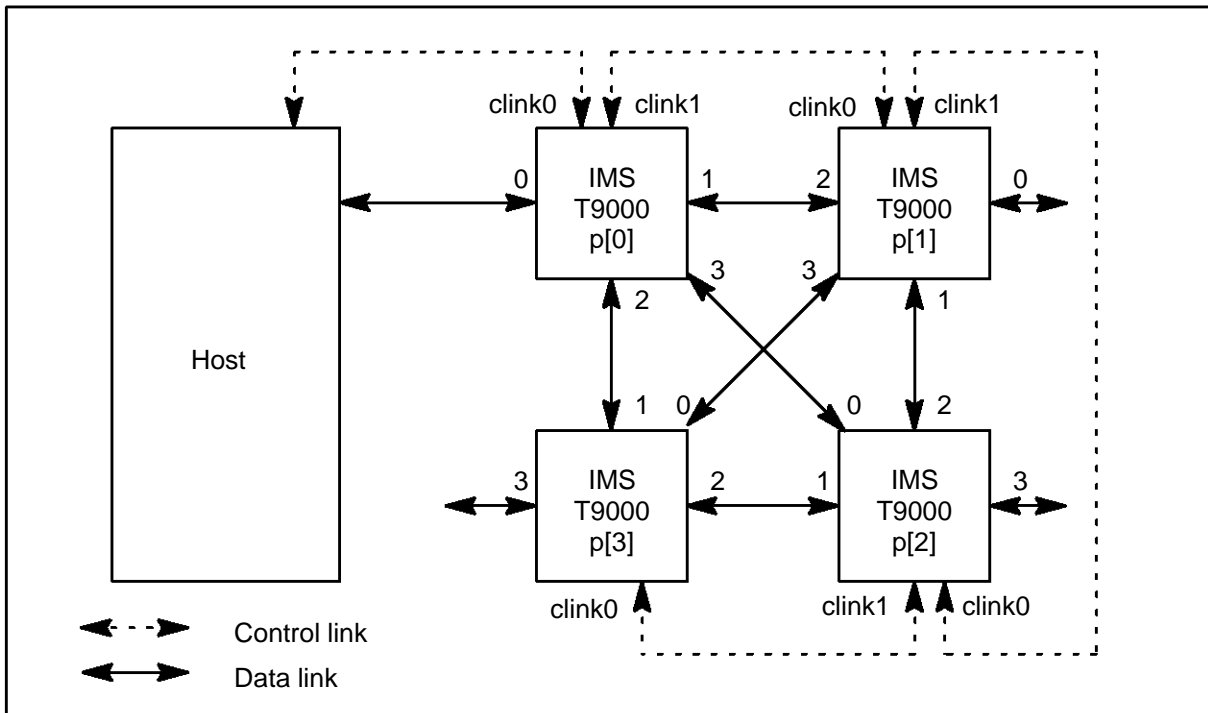


Figure 10 IMS T9000 network example showing link numbers

6.5 ROM support

The software can usually be developed and tested in a hosted development system without ROM by using host files loaded via transputer links. As a final stage, the completed application software and initialization data can be loaded into EPROMs using the EPROM program formatting tool, `ieprom`.

The EPROM formatting tool can produce a file suitable for programming a ROM; this may be either a system ROM or a local ROM. A system ROM holds network initialization data and application software for initializing and loading onto a network of transputers. Stand-alone systems will need to boot from a system ROM. A local ROM holds local initialization data for only one transputer in a network. One or more transputers in a network may be initialized from local ROMs, whether the application is loaded from a host file or from a system ROM.

The EPROM formatting tool may read the network description, memory configurations and the collected application software. From this data, the EPROM formatting tool can build a ROM file incorporating the network initialization data and application software. A control file allows the user to control how the data is arranged in the ROM. The ROM file may be produced in ASCII hexadecimal, Intel hexadecimal, extended Intel hexadecimal, Motorola S-record or binary format.

7 Product components

7.1 Tools

ANSI C compiler, linker and librarian – `icc`, `ilink`, `ilibr`

Makefile generator, binary lister program and memory map lister – `imakef`,
`ilist`, `imap`

Configuration tools – `indl`, `imem`, `inif`, `inconf`, `icollect`

EPROM programming tool – `ieprom`

7.2 Libraries

Full ANSI library plus parallel support – `libc.lib`

Reduced library for embedded systems – `libcred.lib`

Mixed language support library – `centry.lib`

Configuration support libraries

7.3 Sources

Programming examples

Configuration support library source files

Network description examples

C start-up source files

Network control software source files

7.4 Documentation

T9000 ANSI C Toolset User Guide

T9000 Toolset Reference Manual

T9000 ANSI C Language and Libraries Reference Manual

T9000 Toolset Hardware Configuration Manual

T9000 Toolset Delivery Manual

T9000 ANSI C Toolset Handbook

8 Product variants

8.1 Sun-4 product

S IMS D4394 T9000 ANSI C Toolset

8 Product variants

Operating requirements

For Sun-4 hosted cross-development the following will be required:

- S A Sun-4 workstation or server with 1/4 inch tape drive capable of reading QIC-24 format;
- S SunOS 4.1.3 or Solaris 2.4 or compatible
- S 9 Mbytes of free disk space.

For loading target systems, a suitable transputer network interface will be required, such as an IMS B103 Ethernet to DS-Link Interface board and IMS S4397 Sun 4 T9000 Network Interface Software.

Distribution media

Sun-4 software is distributed on DC600A data cartridges 60 Mbytes, QIC-24, tar format.

Licensing

The IMS D4394 T9000 ANSI C Toolset is supplied with a single-user license. Multiple copies can be purchased for larger project teams using volume discount curves.

No licence fee is charged for including libraries in customer products when linked with customer applications using the linker `ilink`. Example programs and other sources provided may be included in software products, but SGS-THOMSON Microelectronics Limited retain original copyright. Full licensing details are available from SGS-THOMSON Sales Offices, Regional Technology Centers and authorized distributors.

8.2 PC product

- S IMS D7394 T9000 ANSI C Toolset

Operating requirements

For PC hosted cross-development the following will be required:

- S IBM 386 PC (or compatible) with a minimum of 8 Mbytes memory
- S DOS 5.0 or later
- S 9 Mbytes of free disk space

For loading target systems, a suitable transputer network interface will be required, such as an IMS B108 add-in board or IMS B103 Ethernet to DS-Link Interface board and IMS S7397 PC T9000 Host Interface Software.

Distribution media

Software is distributed on 1.44 Mbytes 3.5 inch IBM format diskettes.

Licensing

The IMS D7394 T9000 ANSI C Toolset is a single-user product. Multiple copies can be purchased for larger project teams using volume discount curves.

No licence fee is charged for including libraries in customer products when linked with customer applications using the linker, `ilink`. Example programs and other sources provided may be included in software products, but SGS-THOMSON Microelectronics Limited retain original copyright. Full licensing details are available from SGS-THOMSON Sales Offices, Regional Technology Centers and authorized distributors.

9 Problem reporting and field support

A registration form is provided with each product. Return of the registration form will ensure you are eligible for future product updates. Software problem report forms are included with the software. SGS-THOMSON products are supported worldwide through SGS-THOMSON Sales Offices, Regional Technology Centers and authorized distributors.

10 Ordering information


Description	Order number
T9000 ANSI C Sun 4 Toolset.	IMS D4394
T9000 ANSI C PC Toolset.	IMS D7394

Table 1 Ordering information

Information furnished is believed to be accurate and reliable. However, SGS-THOMSON Microelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of SGS-THOMSON Microelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. SGS-THOMSON Microelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of SGS-THOMSON Microelectronics.

E 1995 SGS-THOMSON Microelectronics - All Rights Reserved

 **Imos**, IMS, occam and DS-Link are trademarks of SGS-THOMSON Microelectronics Limited.

 is a registered trademark of the SGS-THOMSON Microelectronics Group.

X Window System is a trademark of MIT.

OSF/Motif is a trademark of the Open Software Foundation, Inc.

Windows is a trademark of Microsoft Corporation.

SGS-THOMSON Microelectronics GROUP OF COMPANIES

Australia - Brazil - France - Germany - Hong Kong - Italy - Japan - Korea - Malaysia - Malta - Morocco -
The Netherlands - Singapore - Spain - Sweden - Switzerland - Taiwan - Thailand - United Kingdom - U.S.A.