Networks, Routers and Transputers:

Function, Performance and applications

Edited by: M.D. May, P.W. Thompson, and P.H. Welch

IOS Press
Van Diemenstraat 94
1013 CN Amsterdam
Netherlands

IOS Press, Inc.
P.O. Box 10558
Burke, VA 22009–0558
U.S.A.

IOS Press/Lavis Marketing
73 Lime Walk
Headington
Oxford OX3 7AD
England

Kaigai Publications, Ltd.
21 Kanda Tsukasa–Cho 2–Chome
Chiyoda–Ka
Tokyo 101
Japan

This chapter was written by J.M. Wilson.

5 Using Links for System Control

5.1 Introduction

The T9000 family of devices includes processors and routers which have subsystems and interfaces which are highly flexible to match the requirements of a wide range of applications. In addition to the static configuration requirements of subsystems such as the memory interface of the T9000, the more dynamic aspects of a network of devices must be configured before application software is loaded. These more dynamic items include:

- cache organization;
- data link bit–rates;
- virtual link control blocks;

If T9000 processors are configured as stand-alone devices, the configurable subsystems will be initialized by instructions contained in a local ROM. When the devices are integrated as part of a network with a static configuration every processor in the network could also initialize these subsystems independently by executing code contained in a local ROM. Typically, however, networks of T9000 family devices contain routers as well as processors and executing code from a ROM is not an option for a routing device. As a consequence, routing devices must be configured under external control. During system development or for systems which are used for multiple applications a flexible configuration mechanism for processors is also required.

Debugging of software and hardware on networks consisting of many devices is not a simple problem. The major difficulty is in monitoring the behavior of the system as an integrated whole rather than observing the individual behavior of the separate components. A flexible mechanism which allows monitoring tools to observe and manage every device in a network in a simple manner is essential in designing a system-wide debugging environment.

5.1.1 Virtual channels

Connecting processors together with point-to-point serial links overcomes many of the problems of shared memory multi-processor systems. Point-to-point links, however, introduce a different set of problems. Of these problems, two of the most critical for system design are, firstly, the difficulty of mapping a software structure on to an arbitrary hardware topology and, secondly, routing messages between processes running on processors which are not adjacent. A great deal of effort has gone in to seeking solutions to these problems and the most flexible and readily implementable technique for overcoming the difficulties is the concept of virtual links. Processes in a network communicate via channels and so the collection of processes and channels define the software topology of a system. The IMS T9000 has multiplexing hardware (the *Virtual Channel Processor*) which allows any number of channels to share the available physical links in such a manner that processes communicating via the channels are unaware of the sharing. Virtual channels are naturally paired to form virtual links, as described in chapter 2. The use of virtual channels allows the software structure of a system to be developed independently from the hardware on which it is to be executed.

Control virtual channels

An ideal way of configuring and monitoring a network of T9000 family devices would be to create a control network in which a master control process running on a host is connected to a client control process on every configurable device in the network. Using virtual links to implement this control network gives exactly the level of control and flexibility required. The remote end of the control virtual link must be managed by an autonomous process which is active and able to obey the instructions of the control process even if the device itself is in a completely unconfigured or stopped state. To achieve this, this process is implemented by an independent hardware module called a *control unit*.

Figure 5.1 illustrates how control virtual channels appear to the control processes involved.

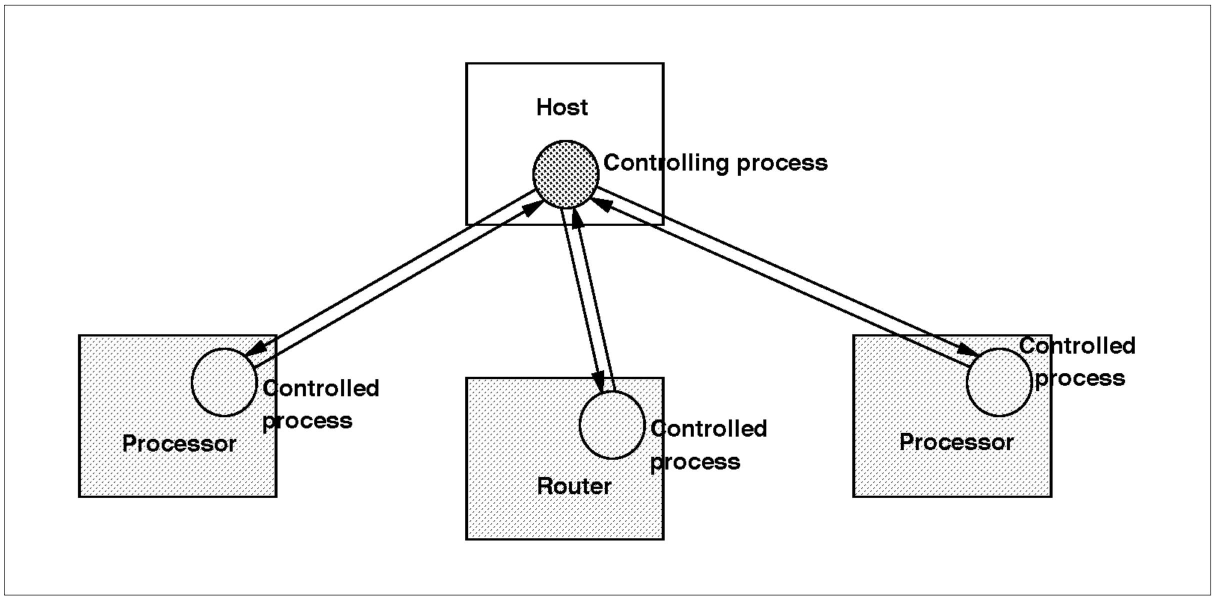


Figure 5.1 Control virtual channels

Providing all device types with an identical control unit allows:

- host system control software to be consistent for every member of the product family;
- the control network on a mixture of devices to be explored and the device types determined;
- processor–free routing networks to be initialized and monitored for error;

A virtual channel from a system control process to every device in a network means that each device can be controlled and monitored as if it were the only device in the network. The ability to control and monitor routing devices is an important capability especially in networks containing no processing devices. Facilities provided by the control system must include the ability to:

- start the device;
- stop the device;
- reset the device;
- identify the device;
- configure the device;
- examine and modify memory (if any);
- load boot code (if the device uses loadable code);
- monitor the device for error;
- re-initialize the control system after an error.

Control links

Because of the critical function of the control system in system initialization and error recovery it is vital that is highly reliable. To guarantee the integrity and reliability of the control system it is essential that it exists in an entirely different domain from the normal operation of the communication system. This separation is achieved by providing each device in the T9000 family with two dedicated control links (`CLink0` and `CLink1`, also called the '*up*' and '*down*' links) and a dedicated control unit. Implementing the control links with a data link is not desirable because it adds complexity to the implementation (by mixing functions which can otherwise be implemented separately) and reduces security, since for example an error in the data network might be impossible to report.

The control links of every device in a network are connected to form a control network. The communication links of the devices will be connected to form a data network. The control network is kept completely separate from the data network and is intended for use by the control system exclusively. It is an important feature that the control links are not accessible to software running on a T9000 processor; the control system is a mechanism designed exclusively for initializing and monitoring the various *hardware* subsystems of the T9000 family of devices. The type of error which would be reported via the control system includes system crashes such as link failure. The control system could not be used for run-time system messages to report failure of a user application. In the latter case the failure messages would be routed via established virtual channels across the data network but in the former case these channels may no longer be reliable. The control network may be run at a lower speed or use different interconnect technology from the data network for increased reliability if necessary.

5.2 Control networks

In a network of T9000 family devices, the control system of each device will have a virtual link to a process running on the processor being used to manage the initialization and monitoring of the system (typically a host). The managing processor, referred to as the control processor, is connected to the network via the *control port*, which consists entirely of one or more standard DS-Links. If the control processor is a T9000, one of its serial links could be used as the control port and the *Virtual Channel Processor* would then implement the virtual channels to the controlled devices. If the control processor is not a T9000, the control port would need to be implemented by a device such as a DS-Link adapter and the virtual channel handling would need to be implemented by software.

Within the control network every control unit obeys a simple protocol on its virtual link. Each message from the control process to a device is acknowledged by a handshake message back to the control process. Each unsolicited message from a device to the control process is acknowledged by a handshake message from the control process to the device as shown in figure 5.2.

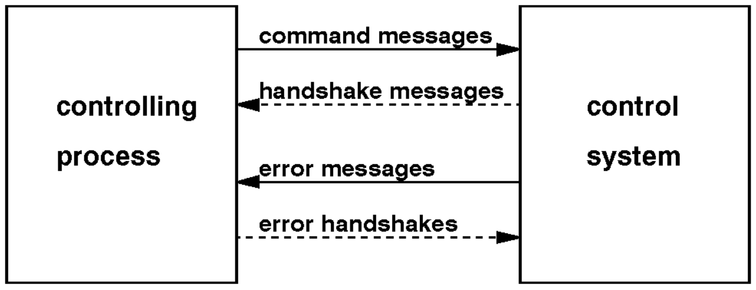


Figure 5.3 Communication between control process and control system

This strict exchange of a handshake message for each command or error message means that the controlling process can be implemented entirely sequentially without danger of deadlock. Even if the control system sends an error message at the same time as the controlling process sends a command, the controlling process subsequently performs an input in any case in order to receive the handshake for its command. When it receives an error message instead it knows that a further pair of messages must be exchanged.

The messages received by a control unit have the form of a command byte followed by parameters specific to that command. Of the thirteen commands in the protocol some are common to all device types and some are specific to particular device types. The physical implementation of the part of the control unit which handles the common commands is generic to all device types. The commands common to all device types are those to start, reset, and identify the device, and to recover from an error in the control network. Other commands are specific to particular devices. The meaning of the commands is detailed in section 5.8.

5.2.1 Implementation

After hard (power-on) reset the virtual links between the control process and the control unit of all the devices in the network must be established. The virtual link to a device is established by the first message received by the network device on `CLink0`; this must be a *Start* command. The *Start* command will be used to set the device 'label' as well as the return header used by the device on every packet sent back to the control process. The label is the header which identifies the virtual link to this device; all packets received from `CLink0` with this label are directed to the device control unit and all those with a different label are passed to `CLink1`. Packets received on `CLink1` are passed directly to `CLink0`. By connecting the control links of all devices into the control network and establishing a virtual link to every device, the control process can initialize and monitor every processor and router in the network independently of the behavior and topology of the data network.

Each device has a single control link pair so in a network consisting entirely of processors these must be daisy-chained as shown in figure 5.4.

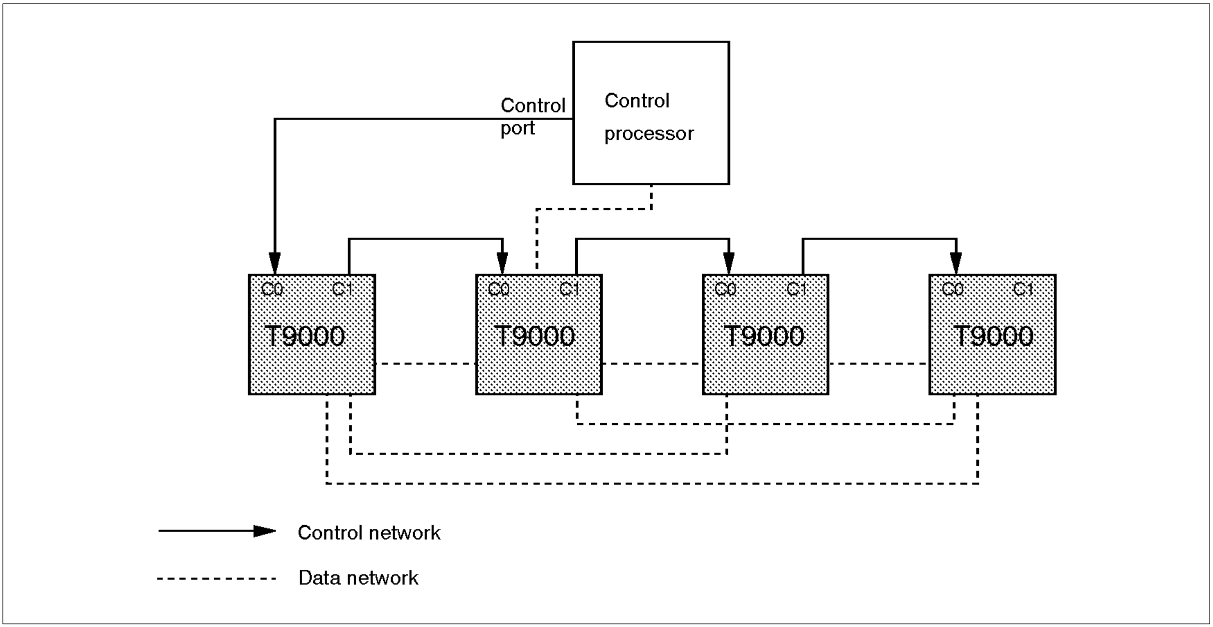


Figure 5.4 Daisy chained control links

For large networks containing IMS C104 devices daisy-chaining is undesirable because of command latency and possible physical routing constraints. In these networks it is better to route the control network via C104s as shown in figure 5.5.

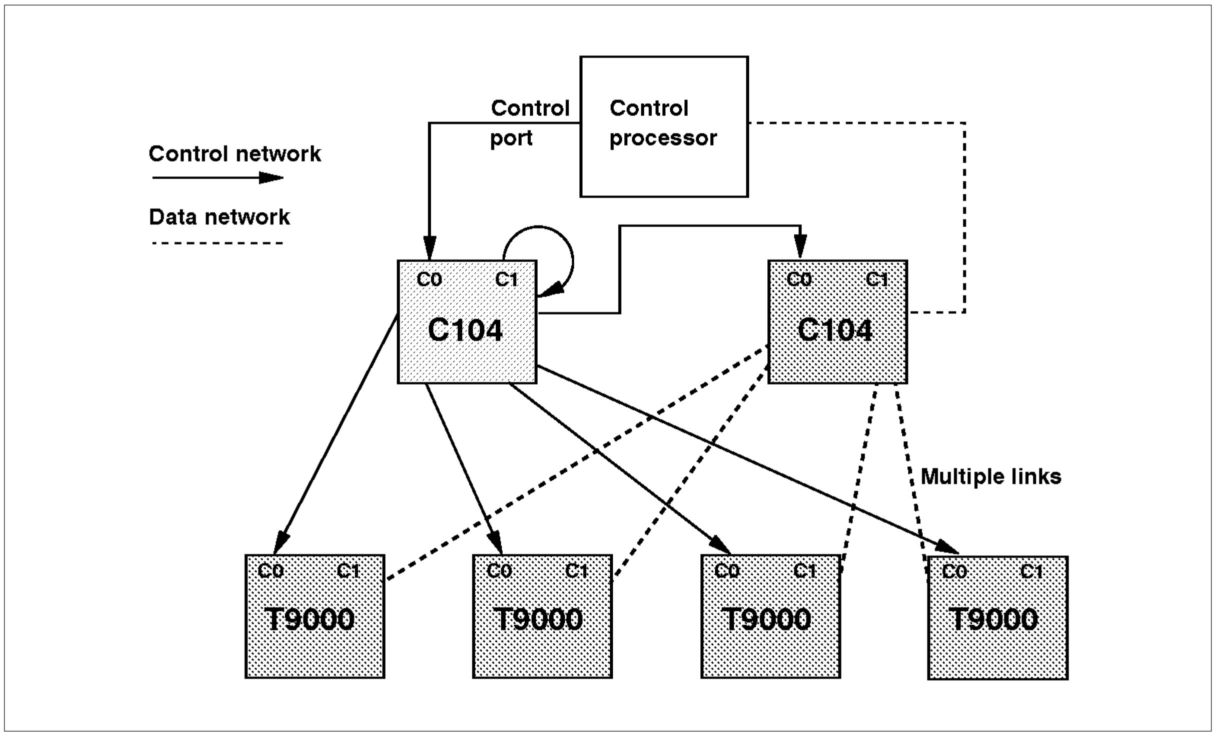


Figure 5.5   Routing control links through an IMS C104

It is possible to use C104s for control network routing because control links use the same electrical and packet–level protocols as the standard data links. When data links on a C104 are used to route the control network, its *down* control link, `CLink1`, can be connected into one of its own data links and thus the control network can fan out in a similar manner to the data network. It is strongly recommended that C104 devices which are part of the control network are used exclusively for the control network and are not part of the data network. If it is unavoidable that a C104 is part of the data network as well as part of the control network it must be partitioned into separate logical devices so that no link can be in both networks (as described in section 3.6.4 of chapter 3). In this case special actions must be taken during reset sequences to avoid losing the control network when resetting the data network. When the control network includes C104 devices the routing tables of the C104 must be initialized using *CPoke* commands before the control network can be fully established.

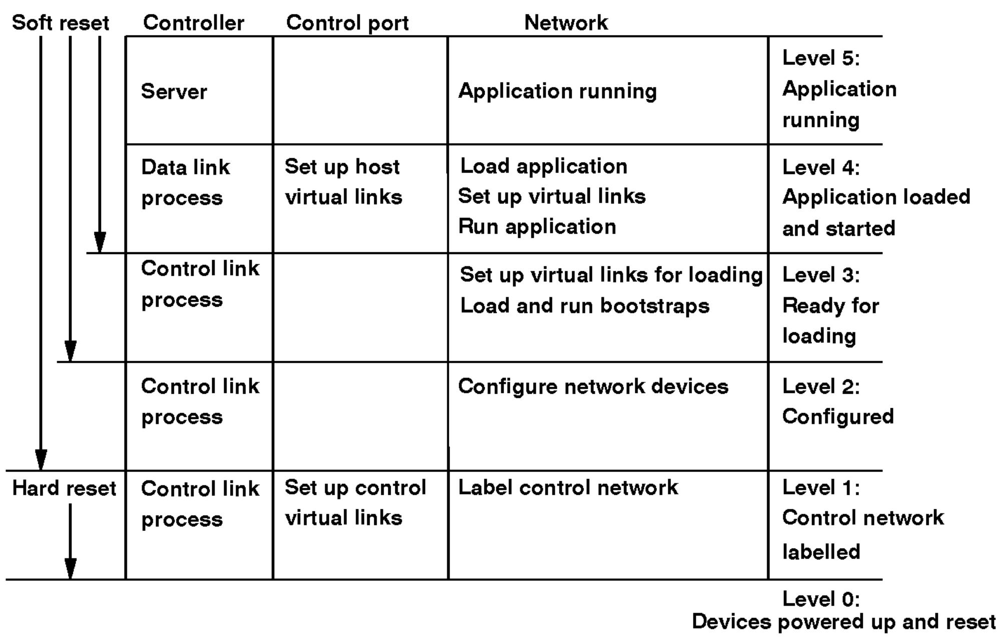
`CLink0` is started automatically by the arrival of the first token. `CLink1` must be started explicitly via a *CPoke* command received by the control process. If a message is received for downward transmission and `CLink1` has not been started a protocol error will be reported.

5.3 System initialization

System initialization is the sequence of actions from receipt of a hard reset (i.e. assertion of the reset pin) until the devices in the system are ready to perform the application for which the system is intended. In a network containing processors, the application may be an operating system ready to run user software or an embedded application ready to start receiving its control data. In a network consisting entirely of routers the system is fully initialized when all of the routing information of the network is established. A possible sequence of actions for a network containing processors and a host, referred to as *levels of reset* and shown in figure 5.6, is as follows:

- Label the control network (including configuring any C104s in the control network) - the network is now at level 1.
- Configure the devices in the network using the control network - level 2.

- Set up virtual links over which to load the network and then run boot code in each processor -level 3.
- Load the network with the application and then set up the virtual links required by the application software -level 4;
- Start the application on the network and a server on the control process - level 5.



Soft reset
Controller
Control port
Network
Server
Application running
Level 5: Application running
Data link process
Set up host virtual links
Load application
Set up virtual links
Run application
Level 4: Application loaded and started
Control link process
Set up virtual links for loading
Load and run bootstraps
Level 3: Ready for loading
Control link process
Configure network devices
Level 2: Configured
Hard reset
Control link process
Set up control virtual links
Label control network
Level 1: Control network labelled
Level 0: Devices powered up and reset

Figure 5.6 Levels of reset

The sequence can be performed one device at a time or network wide one level at a time. For a processor some of the configuration actions can be performed either across the control network or by local software. Because a richer protocol with higher data transfer rates and (possibly) shorter paths can be implemented across the data network than exists on the control network it is generally desirable to establish the data network as early as possible in the initialization cycle.

Using the sequence outlined, application software is loaded onto processors in the network via virtual channels established within the data network. A loader must first be loaded and connected to the virtual channels to load the application at the desired locations. This loader must be loaded and started using the control network and the control channel protocol contains the commands *Boot*, *BootData* and *Run* to facilitate this. The *Run* command provides a workspace pointer and an instruction pointer to start the T9000 CPU. The *Run* command and the *Stop* command are the two commands by which the control system can modify the behavior of the T9000 CPU.

The control network can only be re-labelled after a hard reset so no packet corruption can result in control messages re-configuring the control network. The control process can, however, issue a *Reset* command to any device in the network. The *Reset* command directs the device to reset to level 1, 2 or 3 so the control process can restore individual processors to a known state ready for re-loading an application or, perhaps, to load a debugging kernel.

Some or all of the processors in a network may be set to boot from ROM. Boot-from-ROM devices might be used simply to the configure the local environment or, alternatively, in embedded applications they can be used to configure and then load the whole system.

5.3.1 Local ROM

In many networks it is desirable to localize configuration information. For example it is often useful to program the memory interface locally with the the characteristics of the memory system connected to that processor. Suppliers of special purpose interface boards can build a ROM onto the board which sets up all of the specific characteristics without having to worry about the environment in which the board is going to be used. While a network processor is executing code from its local ROM it is important that the control process does not attempt to load and configure the device. A simple convention to prevent this from happening is for the code in the ROM to set error when it has completed its local configuration, and thereby cause the processor to halt and transmit an *Error* message. The receipt of the *Error* message then signals to the control process that the device is now ready to receive the rest of the initialization sequence.

The local ROM could contain code to take the device to a higher reset level. It might be desirable to bootstrap the device to level 3 ready for the application to be loaded. The same convention as above could be adopted to indicate to the control process that the ROM has completed its initialization sequence.

Boot-from-ROM will only occur automatically after a hard reset. The control process can, however, instruct a T9000 to boot from ROM by sending a *Reboot* message. This allows the control process to be in complete control of the system initialization sequence.

5.3.2 System ROM

A T9000 network may be configured to boot from ROM. The processor which is the root of the network will have access to the system ROM, and will be connected so that one of its data links is the control port at the ‘top’ of the control network. Its own control links will not be connected as part of that same network. This processor will be the control processor as well as the root processor for the system initialization. Configuration information, bootstraps and application code will be drawn from the system ROM rather than from a local file store which would typically be the case if the network was booting from link. After booting the network, the root processor can execute its own application from RAM or continue executing from the ROM. All processors in the network, other than the root processor, are initialized and configured across the control network as shown in figure 5.7. These processors could boot from local ROMs for local configuration if necessary.

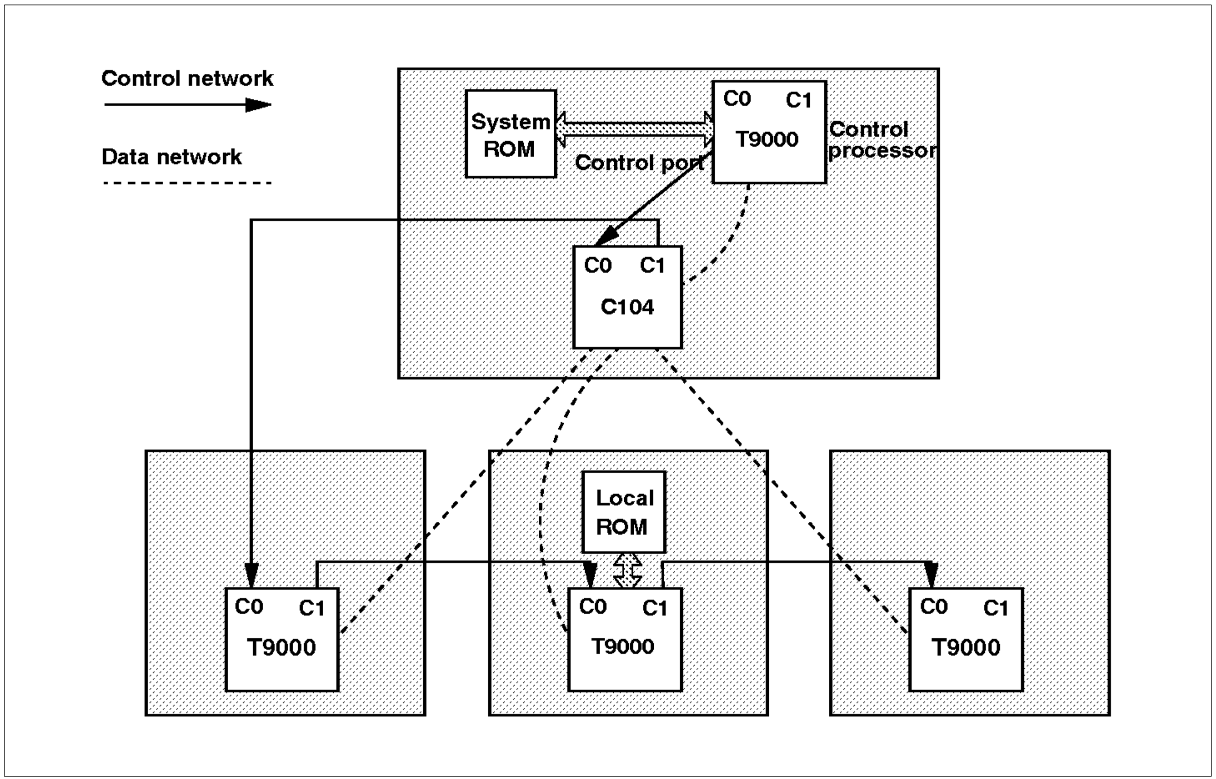


Figure 5.7 Booting from system ROM

A similar mechanism could be employed for a network consisting entirely of routing devices; a single (cheap) processor could initialize the routing tables for the whole network. The processor could then monitor the control system for errors taking appropriate recovery actions and logging information for later analysis.

5.4 Debugging

The normal mechanism for dealing with errors on a working T9000 processor is to execute a trap handler which takes recovery and repair actions to restore the processor to a known valid state. The trap handler may report its actions via the data network to a supervisory process in the system. During development of software and hardware, however, it may be desirable to halt the processor which has caused the error and examine the system state in some detail.

Errors generated by a T9000 subsystem (other than those detected in the CPU and caught by a trap handler) will result in an *Error* message being generated on the virtual channel back to the controlling process and the CPU being halted. The control process can then bring the whole system to a quiescent state by sending a *Stop* command to every T9000 in the network. The *Stop* command stops the processor cleanly, preserving register values and allowing a debugging kernel to retrieve processor state and thus trace the cause of the error. If a processor initiated the situation because of an error, that processor will have halted at the point of error. On all other processors the CPU will continue until the next deschedule point or timeslice. The links are unaffected, and the timers continue to run until a *Reset3* command is received, but no processes will be scheduled.

After the control process has received handshakes for all of its *Stop* messages it must allow time for the system to become quiescent and then issue a *Reset3* command to every T9000. When every device has received a *Reset3* command, all of the CPUs will be halted and the system is guaranteed to be static. At this stage the control process can make certain that the configuration is correct by using configuration ‘peek’ and ‘poke’ (*CPeek* and *CPoke*) commands.

If a debugging kernel is to be loaded into the network it may be necessary to save the area where it is to be loaded to guarantee that no processor state is lost to the analysis tools. This space can be retrieved across the control network using the *Peek* command and stored on the host processor. The debugging kernel can be loaded and started using a *BootData* and *Run* sequence which will not interfere with the preserved state of the data network.

The debugging kernel now has access to all of the previous processor state and can be directed by the debugging tools running on the control processor to retrieve information on all of the processor’s subsystems. The network is thus a distributed data base containing the memory state, register contents and call history of the whole system rather than of just a single processor. The debugging tools can piece together the cause of the system failure and observe the interaction between the different processes and processors. The combination of access to the state of every processor, access to the sources from which the system was built and knowledge of the compiling, linking and loading strategies enables debugging tools to produce an integrated picture of the behavior of the whole system at a symbolic level rather than at an instruction stream level. Once the debugging kernel is loaded onto the network, the debugging tools would, typically, establish virtual channels across the data network to communicate with the individual kernels.

The mechanism described above is called *post-mortem* debugging. *Interactive* debugging can be accomplished by running a debugging kernel on every processor in the system in parallel with the application. In this way breakpoints, watchpoints, single stepping and many of the other facilities delivered by ICE systems are provided without using expensive and intrusive additional hardware. An additional benefit of using links to assist in debugging is the ability to monitor the behavior of a complete multi-processor system observing the interactions across processor boundaries at source level. The debugger running on the control processor communicates with the debugging kernels through virtual channels additional to those established for the data network so that the applications are entirely unaware of the presence of the debugging system.

Much of what has been described in this section is familiar to developers of software for multiprocessor systems. The T9000 family of devices introduce many features to decouple software and hardware development and as a consequence access to the state of routing devices is a vital requirement in system debugging. Access to the state of routing devices is particularly important for networks which contain no processors. The post-mortem mechanisms described earlier are equally relevant for routers. A control process can examine the configuration of a routing device and proceed to access the state of every serial link and thus locate the point of failure and determine what recovery action must be taken. When a data link disconnect error is detected on a router it will cause an error message to be generated on the virtual link to the monitoring process running on the system control processor. As a consequence networks of routers do not require special hardware monitoring devices, a significant amount of fault detection and isolation can be built into the system by the addition of a single monitoring device.

5.5 Errors

The control system provides an error reporting mechanism for all errors, other than those detected by a CPU and caught by trap handlers. The reporting of errors by the control system to the control process is the only time that the controlled device is the initiator of a communication on the control network. The controlling process must acknowledge receipt of the *Error* message by sending an *ErrorHandShake* message back to the device generating the *Error* message. The *Error* message includes a field to indicate the source of the error. The control system will not send an error message if a handshake has not yet been received for a previously sent error message.

The control system handles three distinct classes of error, as listed below.

1. Errors on the control links, which include:
    - parity/disconnect on `CLink1`;
    - unexpected acknowledge;
    - invalid messages;
    - handshake protocol error;
2. System errors - errors from one of the subsystems when stand alone mode is not set.
3. Stand alone mode errors

The effects of the errors are given in table 5.1. The `ErrorSinceReset` flag is a flag in the IMS T9000 which is provided to assist self–analysis of stand–alone systems.

Table 5.1 Error effects

Error class	**Result of error**		
	Stops CPU	**ErrorSinceReset flag set**	**Error message sent on CLink0**
Control link error	No	No	Yes
System error	Yes	Yes	Yes
Stand alone mode error	Yes	Yes	No

The control unit will record a single error which is cleared by the error handshake from the control process. A hard reset, reset 1 or reset 2 will cause the record of untransmitted errors to be cleared.

5.5.1 Control link errors

The basic reliability of DS-Links used within their specifications, as discussed in chapter 4, is very high, and this reliability can be further enhanced for the purposes of the control network by reduced the operating speed somewhat and by paying particular attention to the connection of links. However since an error – however unlikely – in the control network is potentially very serious for the whole system, extra mechanisms are provided to report and recover from such errors.

A parity or disconnect error on `CLink1` will be reported by the control system to the control process via `CLink0`. A parity or disconnect error on `CLink0` will cause the link to halt. This halt will be detected by the device connected to the other end of the link which will in turn report the error.

After an error has occurred some virtual links in the control network may be in an invalid state. The controlling process ends of the virtual links must be reset and then the process can restore the control network to a valid state by sending *RecoverError* commands (which can be sent in violation of the normal protocol). A *RecoverError* command will reset the remote end of a control virtual link and cause any un-handshaken error message (which may have been lost) to be resent. A sequence of *RecoverError* messages sent by the control process to each of the devices in turn can thus systematically restore the control network and at the same time recover information which may help to determine the cause of the failure.

5.5.2 Stand alone mode

When a T9000 processor is operating in stand-alone mode, errors are handled in a distinct way. If an unmasked/untrapped error occurs the control system will reset all of the subsystems on the T9000 and then cause a boot from ROM. The `ErrorSinceReset` flag will be set so that the ROM code can determine that an error has occurred.

5.6 Embedded applications

The root processor in an embedded application which has booted from ROM takes over the role of the control processor on a system which has booted from a host. The control process can monitor and log errors, restarting and re-configuring processors after failure and recovering from errors in the control system. As described in section 5.5.2 above, errors in the control processor result in the processor rebooting. The control process can determine that an error occurred since the last reset and can recover and log information from the previous processor state for later analysis.

5.7 Control system

The control system of each device consists of a pair of control links, a packet handler, a control unit and system services as shown in figure 5.8. The functionality within each unit of the control system is described in more detail below.

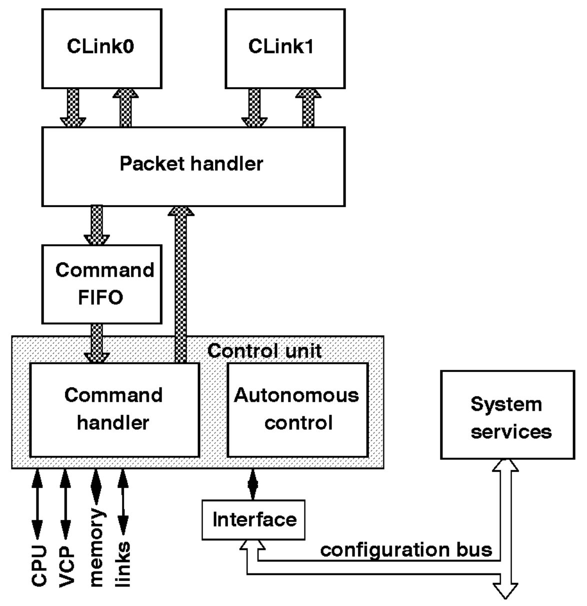


Figure 5.8 Control system components

5.7.1 Control links

A network of devices is controlled by a set of virtual links, one for every device in the network. A simple physical implementation of these virtual links can be achieved by connecting together the control links of a number of devices into a pipeline. The virtual links are multiplexed down this control link pipeline so that, as far as the network is concerned, each device has a single virtual link to the control process which is carried by `CLink0`. `CLink1` carries virtual links for devices further down the pipeline.

The virtual link is established by the first message received on `CLink0` after a hard reset. The physical management of the virtual links by routing packets received on `CLink0` to the correct destination is performed by the packet handler.

5.7.2 Packet handler

The packet handler manages the packet stream performing the following functions.

- Records the first header received on `CLink0` after hard reset as the device label.
- Records the return header from a received *Start* command.
- Checks incoming `CLink0` packet headers. Any with a different label from the one recorded after reset are forwarded to `CLink1`.
- Adds the return header to outgoing `CLink0` packets.
- Forwards incoming `CLink1` packets to `CLink0`.
- Detects and handles acknowledge packets received on `CLink0`.
- Validates that commands are correctly formed and forwards correctly formed commands to the control unit.
- Detects the commands *Reset*, *RecoverError* and *ErrorHandshake*.
- Rejects a command, other than the previous three, if another is already in progress.

. The format of the packets is shown in figure 5.9.

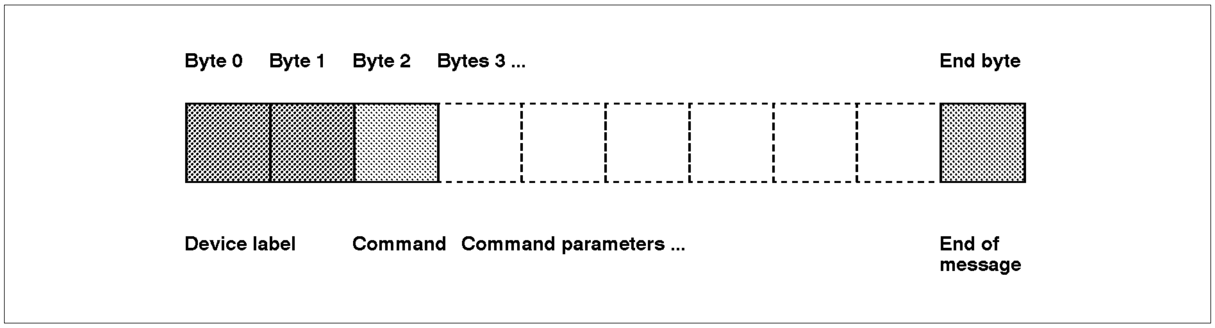


Figure 5.9 Command packet structure

5.7.3 Control unit

The control unit includes a command handler for acting on messages received from the control network and an autonomous control block which controls the behavior of the device when it is operating independently of a control network.

Command handler

The command handler:

- captures errors from error inputs and forwards them to the control process via `CLink0`;
- responds to errors with appropriate stop/halt to sub-systems;
- arbitrates between command responses and errors, and forwards via `CLink0` to the control process;
- filters illegal and inappropriate commands as errors;
- Controls sub-system reset after receipt of a *Reset* command;
- handles access to the configuration bus after receipt of *CPeek*, and *CPoke* commands;
- handles access to the memory system after receipt of *Peek*, *Poke*, *Boot* and *BootData* commands;
- stops the processor cleanly after receipt of a *Stop* command;
- starts the processor with with a given workspace and instruction pointer after receipt of a *Run* command;

- starts the processor with with a workspace and instruction pointer read from a ROM after receipt of a *Reboot* command.

5.7.4 System services

The system services is a block of registers in the configuration space containing control and general device information.

5.8 Commands

The commands to which the control unit responds are as follows.

5.8.1 Commands applicable to a variety of devices

Start

This must be the first command received by a device after a hard reset. It is used to program the return header of the device. After a hard reset it will also set the label of the device.

Identify

The *Identify* command causes the device to respond with a handshake containing an identifier unique to that device type.

CPeek

CPeek commands are used to examine registers in the configuration space. The handshake message contains the contents of the selected register.

CPoke

CPoke commands are used to initialize registers in the configuration space.

Reset

Reset is used to reset the device to a chosen state specified by a parameter. The parameter can typically have the values *1*, *2* or *3*.

1. Equivalent to hard reset but the control system is unaffected.
2. Resets all subsystems except the control system, and leaves the configuration unchanged.
3. Just halts the processor.

RecoverError

RecoverError is used to restore the protocol after a link error in the control link system.

5.8.2 Commands applicable to processors

Peek

Peek commands are used to read the normal address space of a T9000. The handshake message contains the contents of the selected address.

Poke

Poke commands are used to write data to memory locations in the normal address space of the T9000.

Boot

This command initiates a booting sequence. Parameters to the command specify the length of code to be loaded and where it is to be loaded in memory. The *Boot* and *BootData* allow code to be loaded much more efficiently than it would be by using a sequence of *Poke* commands.

BootData

A sequence of *BootData* commands follow a *Boot* command. Each *BootData* command will contain 16 bytes of code which will be loaded into consecutive locations starting from the address specified in the *Boot* command until the length specified by the *Boot* command has been reached.

Run

The *Run* command specifies a workspace pointer and an instruction pointer and causes the processor to start executing with these values.

Stop

This command causes the processor to come to a ‘clean’ stop ready for post-mortem debugging.

ReBoot

The *ReBoot* command re-initiates a boot-from-ROM sequence.

5.9 Conclusions

Using the same electrical and packet protocols for system control as for data transfer allows large concurrent systems to be programmed, monitored and debugged in a very straightforward way using virtual links. A small set of commands, supported directly in hardware, provides precise control over individual devices and the whole system. A simple handshaking protocol at the message level ensures that a simple, sequential control process can be used without any difficulty. Using a separate network for system functions improves the reliability and security of the system.

The provision of a two links and a basic through–routing function on each device allows a low–cost daisy–chain topology to be used for small systems. Larger systems can employ C104 routers in the control network to improve fan–out.

Facilities have been added to recover use of the control network even after the temporary disconnection of one of its links. The *RecoverError* command provides a ‘remote channel reset’ function to enable the control virtual links to be restored to a known state. Error information which might have been lost is re–transmitted.