Inside the TC2000™ Computer

Revision: First Release

Part No. A370014G10
Document Rev: A

BBN Advanced Computers Inc.

RESTRICTED RIGHTS LEGEND

BBN Advanced Computers Inc
10 Fawcett St.
Cambridge MA 02138

RELEASE LEVEL

This manual conforms to the February 1990 TC2000™ multiprocessor hardware..

NOTICE

TRADEMARKS

BBN ACI thanks the following contributors for their efforts in developing this manual:

Author:
Michael Beeler

Designers and Reviewers:
Dave Barach
Tom Downey
Jim Gibson
John Goodhue
Phil Herman
Kent Hoult
Larry Kaplan
Becky Mercuri
Rich Schaaf
Julie Tiao

Contents

List of Figures

How to Use This Manual

Purpose of the Manual

This manual explains the hardware structure, components and capabilities of the TC2000 computer, including how operating system and application software may make use of those capabilities. This manual is intended to suffice for understanding the concerns in programming the TC2000 computer, but does not contain the details of system calls, register bit allocations and the like necessary for actually writing the program. Other components of the TC2000 document set deal with those details.

Revision History

This edition is a revision and expansion of the earlier, "preliminary" edition dated August 14, 1989. No major changes have been made, but rather areas of less importance that were not fully covered in the earlier edition have been filled in. An error in the August 14 edition has been corrected:

- The "VME" in "VMEbus" originally stood for Versa Module Europe, (not Virtual Memory Extension).

Other Places to Find Answers

If you experience any problems with our product, or if you have questions or suggestions, please do one of the following:

- Send electronic mail from anywhere on the Internet to:

  aci–questions@bbn.com

- Send mail to:

ACI Bugs
BBN Advanced Computers Inc.
10 Fawcett St.
Cambridge, MA 02138

- If you are under warranty, or have a software maintenance contract, you can also call our hotline number:

  1-800-4AC-BFLY (1-800-422-2359) in the United States
  1-617-873-8660 from any other location

If you are reporting a problem, please include as much information as you can, as follows:

- The operating system **version** and multiprocessor **model name**
- The **size** of your multiprocessor (number of function cards and amount of memory)
- The **number of nodes** that were in the cluster when the problem occurred (if relevant)
- The **total number of people** using the system when the problem occurred
- An **example** that illustrates the problem
- A **record** of the sequence of events that led to the problem; especially a stack backtrace (see the system administration guide)

We are also interested in your evaluation of our documentation. We would appreciate it if you would fill out the form at the back of this manual and return it to us.

Audience Level

This manual is intended for any new TC2000 user with a basic knowledge of computers. Sections that begin with a basic introduction are so noted, allowing the more experienced reader to skip the introductory material.

Other References

References to other documentation, both within the TC2000 document set and other hardware manuals, are placed with the related subject in the text.

Organization

Chapter 1 is both an introduction and a condensation of the other chapters in the book; it may be read independently for a quick picture. Each following chapter examines a different aspect of the machine in detail.

Typographic Conventions

This manual uses the following conventions to present information:

bold	Text in **bold** indicates a key word or phrase.
italics	Text in *italics* indicates a specific name, such as the name of a bit in a register.
bold italics	Text in ***bold italics*** indicates an emphasized word or phrase.
`type`	Text in `typewriter` font is used when monospacing (rather than proportional spacing) is needed for a diagram or example.

1 Structure of the TC2000 Computer

This chapter names the major components of the TC2000 computer and describes their function and their relation to each other. You can read this chapter by itself for a quick understanding of the whole machine. The following chapters describe the same parts of the machine, in greater depth. Throughout, the emphasis is on understanding the internal structure so that you can use the TC2000 computer efficiently and effectively.

1.1 Basic Characteristics

The TC2000 computer is a powerful new multiprocessor. It builds upon BBN's experience in the design of parallel processors — the Pluribus and Butterfly computers — and extends and expands those concepts to a state of the art machine.

The TC2000 computer is a **multiprocessor** machine because it employs a number of microprocessors, each executing individually on the users' tasks in a controlled and coordinated way. The opposite is a uniprocessor, in which a single processor executes all tasks. Only by employing exotic and costly technology can a uniprocessor offer performance even close to the TC2000 multiprocessor.

The TC2000 computer employs **shared memory** to store information. All main memory of the machine is accessible to every processor. Access protection mechanisms are used to restrict access as needed by the software. In a machine without shared memory, a processor wishing to read or write data stored in another processor's memory must take extra steps. Usually these steps are cumbersome to program and slow in execution. Shared memory avoids these pitfalls.

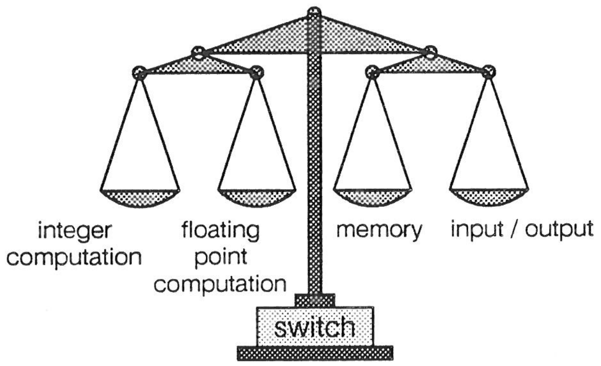
The TC2000 processors access the shared memory through an interconnection network called the **Butterfly switch**. The switch provides a fast, efficient and effective access path. Multiprocessor machines without a switch rely on a bus.

A bus has a fixed limit on how much data it transfers per unit of time, and when that bandwidth limit is reached, further demand for data transfer must wait, and the bus-based machine runs no faster. The TC2000 switch avoids saturation because its bandwidth, unlike that of a bus, increases as the machine is expanded.

The TC2000 design is **modular** and **scalable**. Processors, memory and I/O capacity can be added board by board, as needed by the application. A small, development system can be expanded as the user's requirements or budget evolve. The design permits installation of any number of boards, each containing both a processor and memory, from one to 512. Without the scalability offered by the TC2000 design, the user would be constrained to fit the application to a small set of machine sizes. The compromise of contorting the program into a too-small machine, or the high cost of a machine bigger than the application needs, are not choices the TC2000 user has to make.

The TC2000 computer has a **balanced** architecture. The integer computation, floating point computation, memory, and input/output capabilities are approximately equal in power. Each "processor" board can also contain memory and/or I/O capacity, so this balance can be maintained as boards are added. Further, the bandwidth of the switch expands to keep pace with the added processing, memory and I/O capacity. This makes the TC2000 well suited for a wide variety of applications. The architecture of some other machines is out of balance, making them suitable only for applications that place heavy demand on their strong points. The balanced architecture of the TC2000 avoids this constraint.

Figure 1–1 **Four–way balance of TC2000 architecture.**



1.2 Architecture

The TC2000 architecture consists of **function boards** interconnected by a high performance **Butterfly switch**. In addition, the **Test and Control System** (TCS) monitors the entire machine.

Each **function board** contains some or all of the following: a processor, cache and memory management unit, memory, a VMEbus interface, a switch interface, TCS circuitry and power supplies.

The **switch** connects to two ports on each function board, one by which the board accesses other boards, and one by which the board services access requests from other boards. The switch also provides clock signals to each function board.

The **TCS** initializes, loads, starts, monitors and resets all function boards in the machine. It also monitors and controls all other components — switch cards, clock cards, and power supplies.

Support components include the **midplane** — like a backplane, but with connections on both sides — that connects function boards and switch cards into modules of eight function boards each; power supplies; and cabinetry. VMEbus devices, typically for I/O, connect to function boards via another midplane.

Figure 1–2 shows the main components of the TC2000 architecture. While the design permits expansion to 512 function boards, the current implementation is limited to at most 64 function boards. To avoid cluttering Figure 1–2, the components typically interconnected by a single midplane are shown in a separate illustration, Figure 1–3.

Figure 1–2 **Main components of the architecture.**

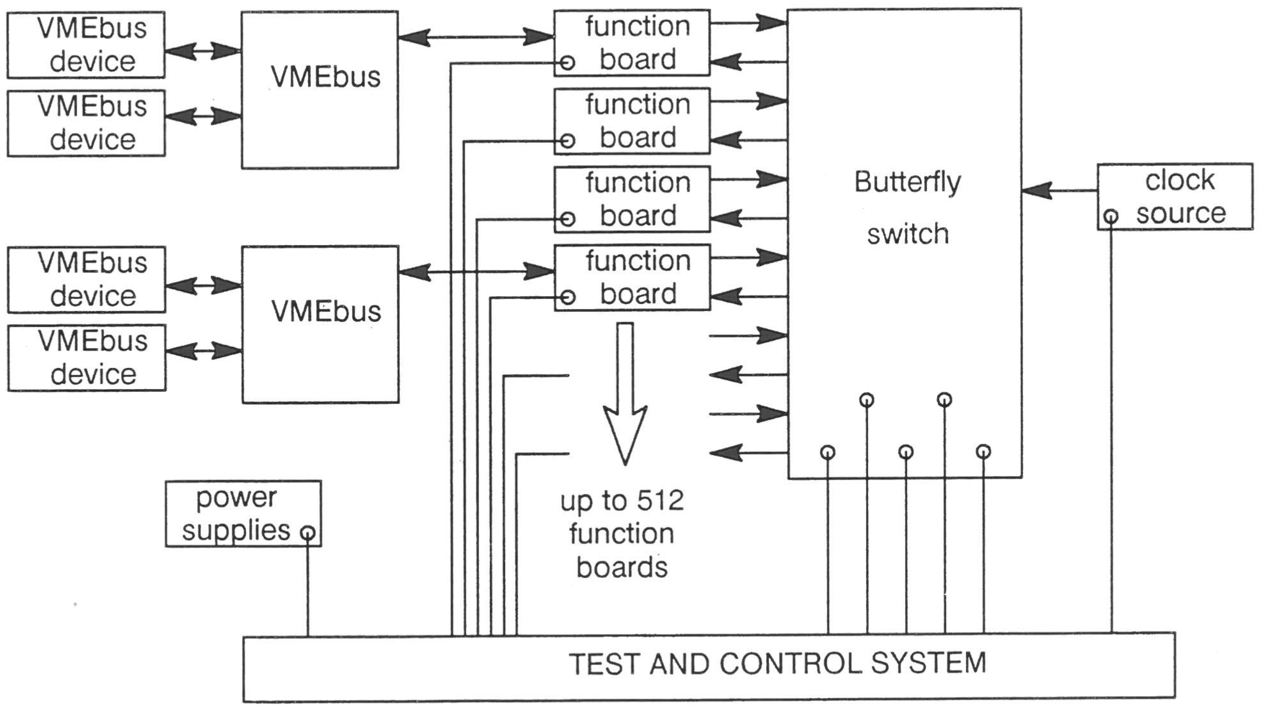
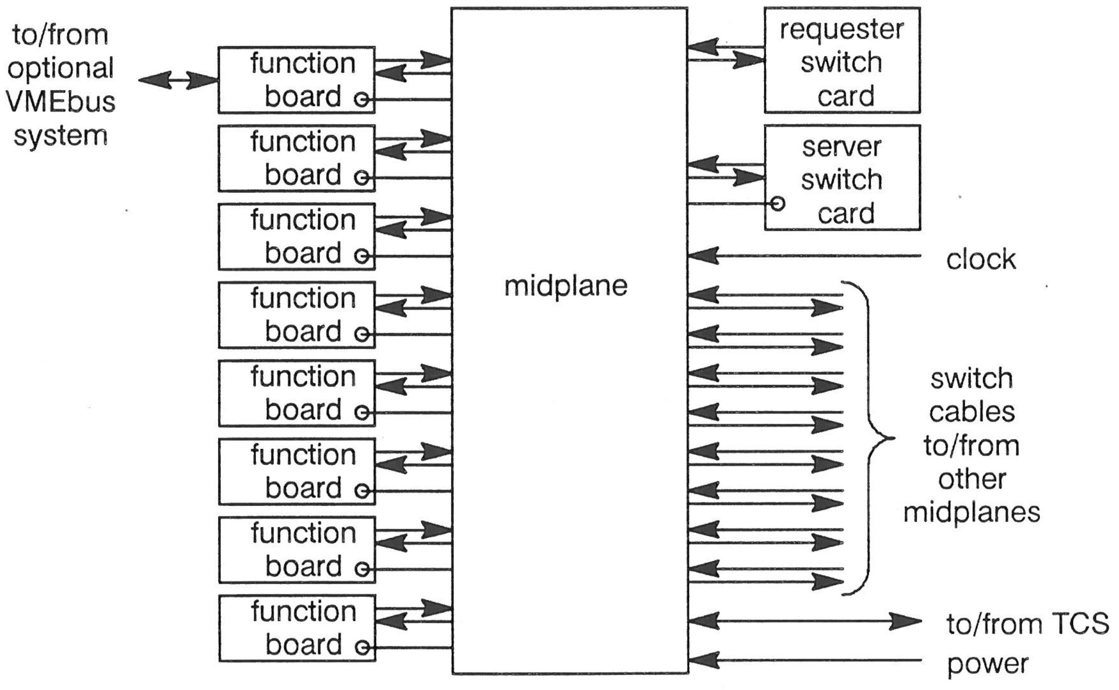


Figure 1–3 Components associated with one midplane.

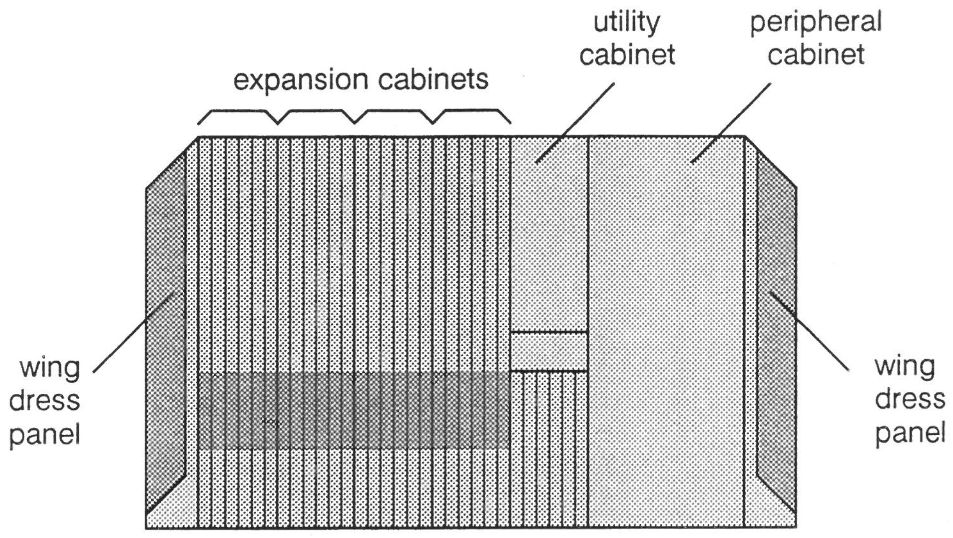


1.3 System Components

1.3.1 Physical Structure

The TC2000 computer is assembled from three kinds of cabinet, placed side to side, with a dress panel on each end. This is shown in Figure 1–4. A typical machine has one peripheral cabinet, one utility cabinet, and from one to eight expansion cabinets.

Figure 1–4 **Cabinets make the machine.**



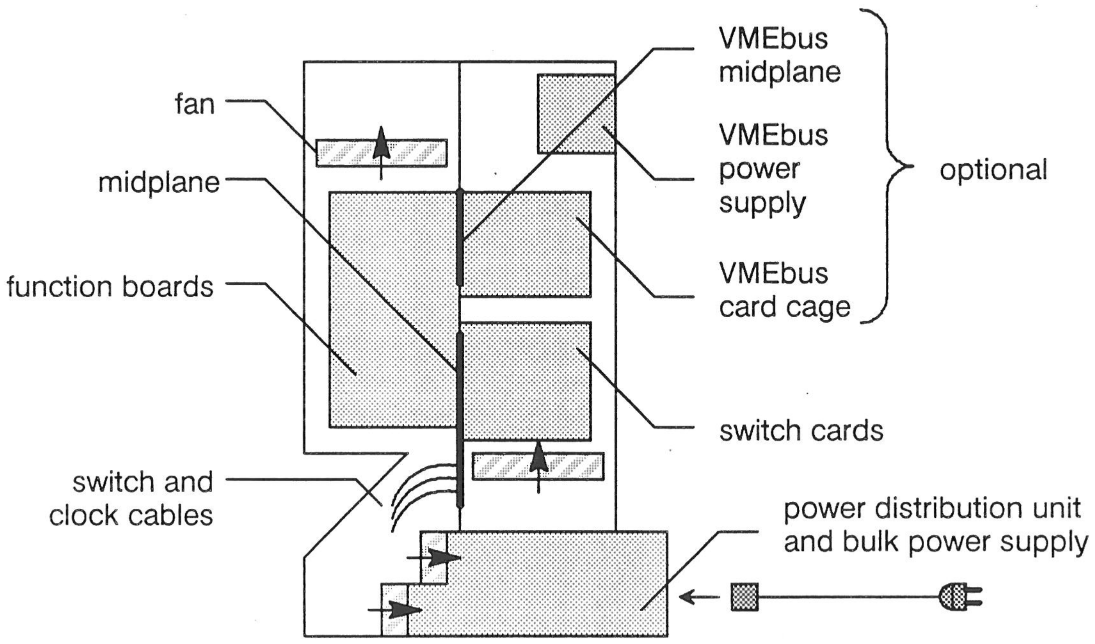
- Each **expansion** cabinet holds up to eight function boards, plus the switch cards, VMEbus cards (if any), and power supplies associated with those function boards.
- The **utility** cabinet houses the Test and Control System (TCS) master, its associated disk, tape drive and power supply, and the clock card for the entire machine.
- The **peripheral** cabinet contains a VMEbus card cage, standard and optional I/O devices, and power supplies.
- The **wing dress panels** on either end provide space for cable routing, and protect the stabilizer feet.

Each of the three kinds of cabinet is described below.

Expansion Cabinet

Figure 1–5 shows a side view of the TC2000 expansion cabinet. Expansion cabinets are added as needed to house the required number of function boards; hence the name "expansion" cabinet.

Figure 1–5 **Expansion cabinet side view.**



Function boards provide the processing, memory and VMEbus interface capabilities of the machine. Up to eight function boards can be installed in an expansion cabinet. All electrical connection to the function boards is via the midplane and the optional VMEbus midplane.

Switch cards interconnect the function boards in this and other expansion cabinets, and distribute the machine-wide clock and TCS communication signals. There are two types of switch card, a requester card and a server card. Each requester card is paired with a physically adjacent server card. An expansion cabinet contains one pair of switch cards.

The **midplane** connects function boards, switch cards, and switch and clock cables. The midplane serves a function similar to a backplane on more conventional machines. It is called a *mid*plane to emphasize that it has connectors on both sides. A backplane typically contains a global communication bus. Since the TC2000 computer uses a switch for global communication, not a bus, the midplane does not contain such a bus. The midplane also distributes power to function boards and switch cards.

The **switch cables** connect switch cards in this expansion cabinet, and thereby its function boards, to those in other expansion cabinets. The **clock cables** supply clock and other machine-wide signals to the switch cards in this expansion cabinet, from where the signals are further distributed to this cabinet's function boards.

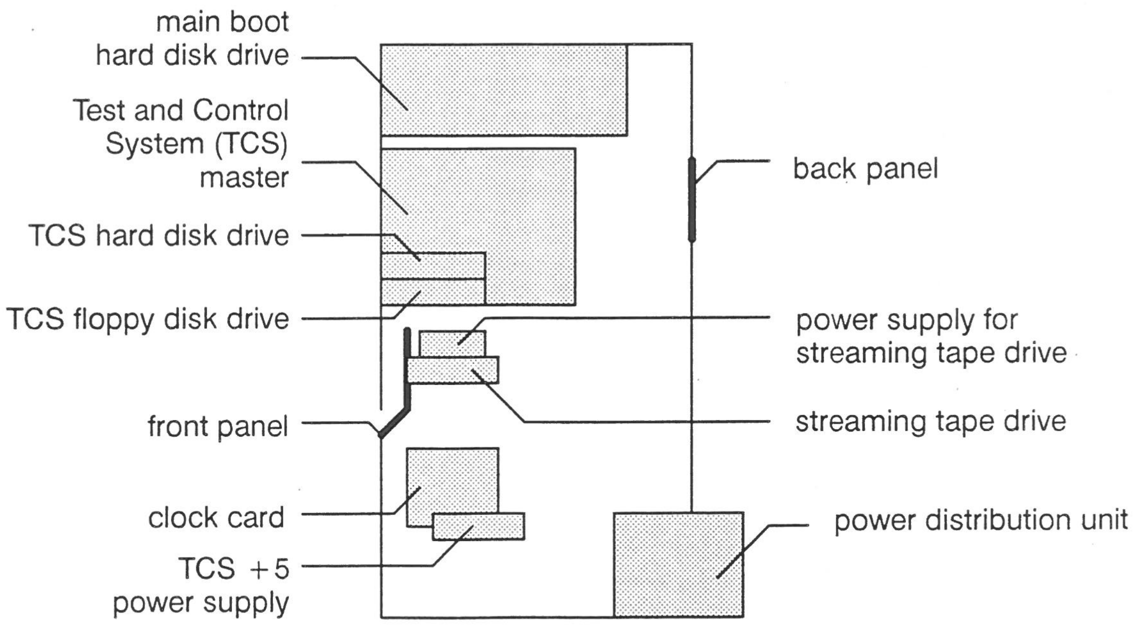
The **power distribution unit** (PDU) and **bulk power supply** receive AC power from outside the machine, convert it to ±24 volts, and distribute it to the function boards and switch cards in the expansion cabinet. Various **fans** force air in through the front of the expansion cabinet and through the components as shown in Figure 1–5.

Each expansion cabinet may contain a package of three optional VMEbus components: a VMEbus midplane, card cage and power supply. The **VMEbus midplane** connects function boards to VMEbus cards installed in the **VMEbus card cage** (size "6U"). The VMEbus midplane implements not one, but several small VMEbuses, each connecting certain function board slots and certain VMEbus card cage slots. VMEbus cards often have cables on their back edge, leading to peripheral devices mounted in the utility or peripheral cabinets. Power for the VMEbus card cage comes from the **VMEbus power supply**.

Utility Cabinet

The utility cabinet is shown in Figure 1–6. It houses assorted equipment specific to the TC2000 computer, and only one utility cabinet is required per machine. The equipment here generally serves to control the rest of the machine.

Figure 1–6 **Utility cabinet side view.**



The **front panel** consists of four indicator lights (LEDs), a keyswitch, a reset button, a tape drive, and a small amount of support circuitry behind the panel.

The LEDs indicate main power, TCS enabled, TCS power, and attention required. The keyswitch controls machine power and has three positions: off, on and secure. The TCS master can sense whether the keyswitch is in the "on" or the "secure" position, and decide accordingly whether to act on commands it receives. The reset button forces the TCS master processor to reset; the rest of the TC2000 hardware is reset by the TCS master upon command. The **streaming tape drive** is used to load and dump TC2000 system software. The keyswitch, reset button and tape drive are accessed by opening a door above the LEDs, which are always visible.

The **TCS master** controls power and operation of the machine as a whole. Power, resetting, loading, execution startup, and monitoring are all under the control of the TCS master. The TCS master is an IBM PC/AT compatible microcomputer and associated hardware. In particular, the physical structure houses a **TCS floppy disk drive** used to load TCS master processor software. The **TCS hard disk drive** stores this software and other files pertaining to the control, configuration and operation of the machine. The floppy disk drive is hidden by a panel during normal operation.

The **main boot hard disk drive** stores the operating system and its associated files. This drive is also available for user file storage.

The **clock card** supplies the master clock signals to the rest of the machine. Certain other signals are also generated in the clock card. The frequency of the clock signal is set by the TCS master.

The **TCS +5 power supply** provides power for the TCS slave microcomputers throughout the machine. These microcomputers appear on function boards, switch cards, and the clock card. They respond to commands from the TCS master, performing control actions and monitoring status on their respective boards. Because they receive power separate from the main bulk power of the machine, the slaves have control even when their boards' power is off. In fact, turning on board power is one of the slaves' responsibilities.

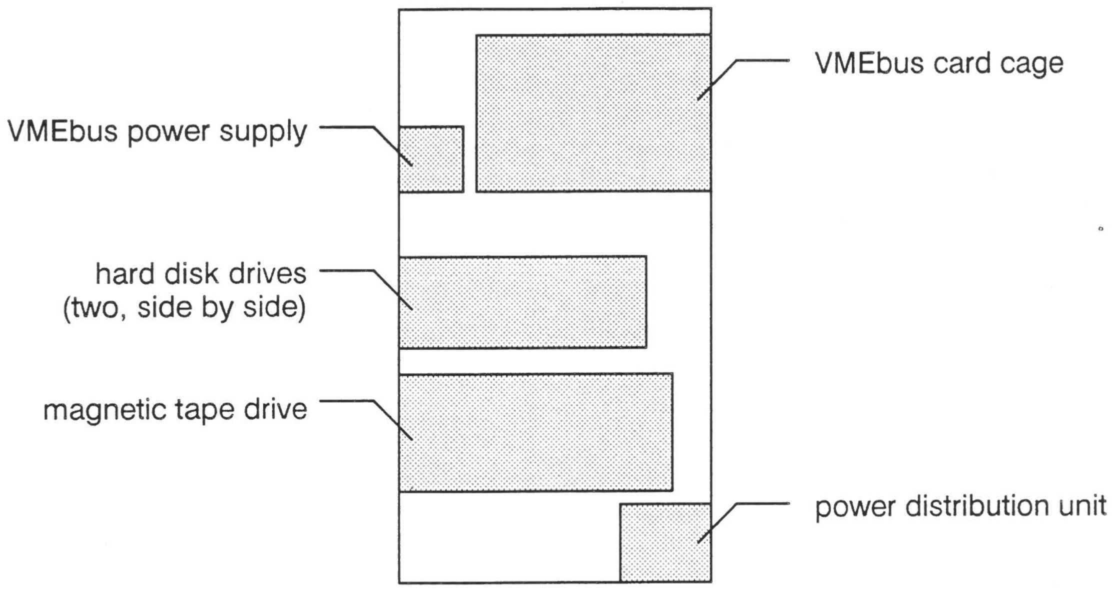
The **power distribution unit** in the utility cabinet differs from the PDU in an expansion cabinet in that it is controlled by the front panel keyswitch, while expansion cabinet PDUs are controlled by the TCS master.

The **back panel** contains five jacks. One jack accepts an RS-232C cable to the TCS console, typically a standard display terminal, used to communicate with the TCS master. A second jack is for the PDU control line, by which the TCS controls the PDUs in other cabinets. An optional telephone line, used for remote diagnostic procedures, connects to a third jack. When such a line is connected, an ordinary telephone can be connected to the fourth jack for voice communication. A fifth jack is reserved for future use.

Peripheral Cabinet

The peripheral cabinet houses whatever additional I/O equipment is needed at a particular site. If the I/O capability of the expansion and utility cabinets is sufficient, no peripheral cabinet is used. Usually, however, at least an additional VMEbus card cage is required, and often other gear as suggested by Figure 1–7.

Figure 1–7 **Peripheral cabinet side view (typical).**



In the example configuration of Figure 1–7, three I/O devices are mounted in the peripheral cabinet: two hard disk drives and a magnetic tape drive. The **VMEbus card cage** is typically a larger (9U) size than the VMEbus card cage in expansion cabinets (6U). VMEbus cards requiring a 9U cage must be mounted in the peripheral cabinet. The 9U cage in the peripheral cabinet is connected to a 6U cage in an expansion cabinet by a **VMEbus repeater**, making the two cages into one VMEbus system.

The peripheral cabinet's VMEbus card cage typically holds controller cards for the devices in the cabinet, such as the hard disk drives and magnetic tape drive shown in Figure 1–7. It may also hold the controller for the hard disk drive mounted in the utility cabinet. Further, it may hold communications interfaces, such as an **Ethernet interface** and a **multi–line terminal controller**.

The TCS master controls the peripheral cabinet PDU.

1.3.2 Logical Structure

Figure 1–2, above, shows the main components of the TC2000 architecture. This section describes the logical function and structure of each kind of major component, and its connections to other components.

The Function Board

The function board provides all processing and memory capacity used by application programs. A function board ***may*** contain a processor and associated circuitry; memory; a switch interface; a VMEbus I/O interface; and a bus linking these components. Each component is described below. Then the question of which components are required and which are optional is discussed. Only the TC/FPV function board contains all of them.

The **processor** is a Motorola 88000 chip group comprised of an 88100 CPU chip and at least two 88200 cache/memory management unit (CMMU) chips. One or two 88200 chip(s) handle instruction references, and one other 88200 handles data references. The processor chips connect with the rest of the function board through circuitry called the CPU interface. This interface provides three major functions. It translates addresses from the 32-bit addresses used in the Motorola chips to the 34-bit addresses used in the rest of the machine. Secondly, it supports extensions to the operations natively available from the 88000, such as locking. And third, it implements several registers that control the CPU's operation.

The 34–bit system physical address gives the machine a 16–gigabyte address space. Any one function board may contain up to 32 megabytes of shared memory, taking 25 of the address bits. The remaining nine address bits select a given function board from up to 512 possible function board slots in the machine.

The **memory** is dynamic RAM, with parity, addressable as byte, halfword (2 bytes) or word (4 bytes), aligned on boundaries of the size being addressed. The memory supports an important augmentation to the basic 88100 features: locking. **Locking** is a means to synchronize access to memory among multiple processors. During a sequence of locked accesses to a memory module — that is, to the memory on a given function board — no other CPU or I/O interface can access that memory module with a normal access request. Therefore, the CPU or interface making the locked accesses can, for example, read the contents of a location, compute a new value for the location, and write it back, **atomically**. A typical use for locking is to access a shared datum. The software can explicitly **bypass locks**, in which case the access is permitted despite the lock. Atomicity, locking and bypassing locks are discussed further in chapter 2.

Almost all I/O to and from the machine is via one or more VMEbus interfaces. A **VMEbus interface** is implemented on a function board, and permits high bandwidth access from the TC2000 machine to the VMEbus and vice versa. The VMEbus interface contains two major sections. One section is a VMEbus **master**, mapping references coming from the TC2000 hardware into references on the VMEbus. The other section is a VMEbus **slave**, responding to references on the VMEbus and mapping them into the TC2000 address space. The mapping capabilities in both directions are fully programmable and support several features of both the TC2000 and the VMEbus architecture, such as locking (on the TC2000 side) and interrupt generation and handling (on the VMEbus side).

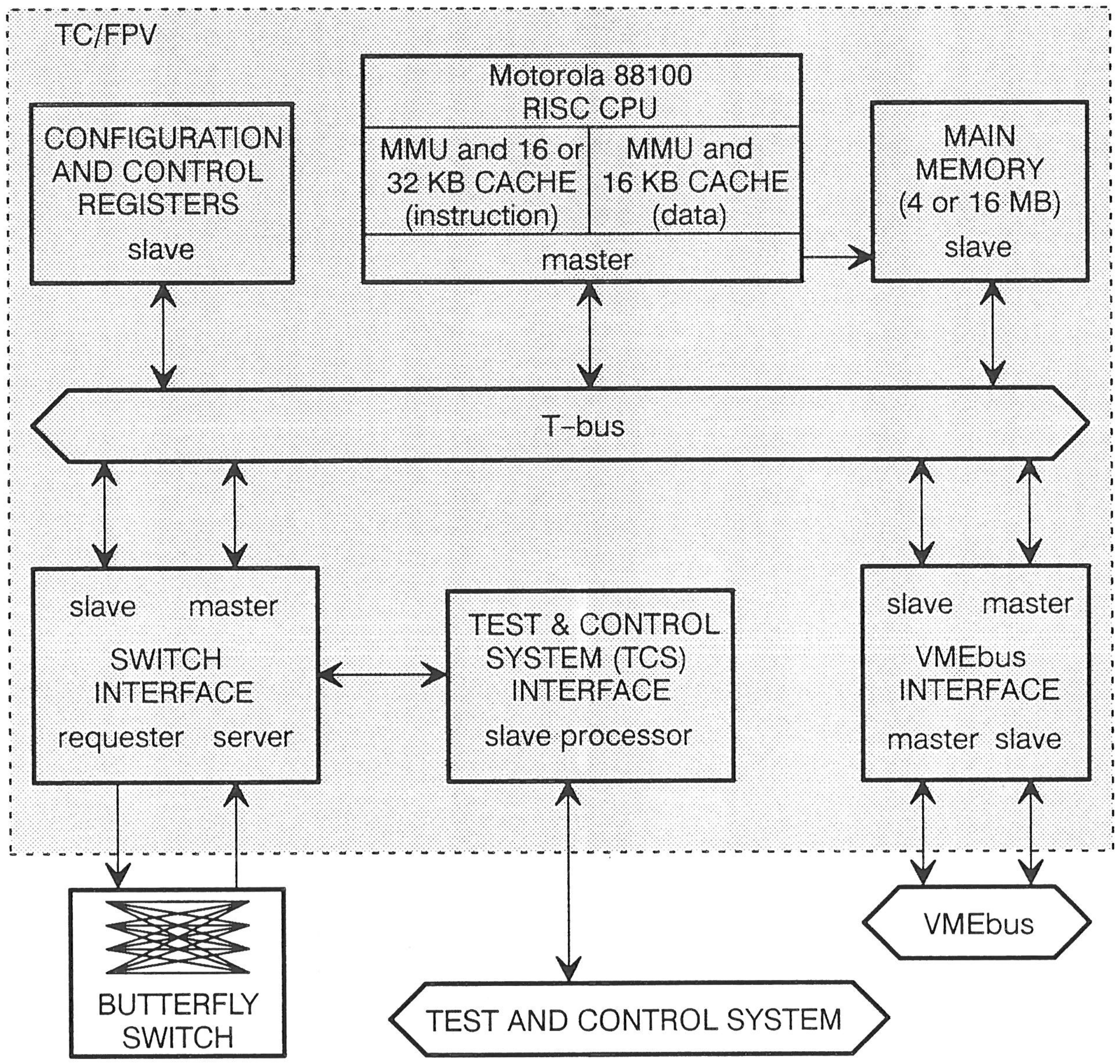
Every function board connects to the rest of the TC2000 computer via the switch. The only other connection is a low-speed line to the Test and Control System master, so the switch is critical for access to resources on remote function boards. The **switch interface** is implemented with two custom gate array chips. The Switch Interface Gate Array (**SIGA**) is a special purpose micromachine. Within the SIGA, one section transforms access requests arising on the local function board into messages, sends these into the switch, receives the reply message, and gives the reply data to the CPU or I/O interface that made the request. This section of the SIGA is called the **requester**. Similarly, another section of the SIGA, the **server**, accepts messages arriving through the switch from remote function boards, services the request by reading or writing the data in the local memory or I/O interface, and sending any reply data back through the switch to the requester. The other special chip is the Level Converter (**LCON**), which converts and conditions the signals between the electrical conventions used within the function board (TTL) and in the switch (ECL).

Connecting together all the logical modules on a function board is the transaction bus, or **T-bus**. The T-bus is a high-bandwidth bus confined to the function board.

To be a TC2000 function board, what components must a board contain? The simplest way to define a function board is that it connects to the TC2000 switch (and TCS). Therefore, it has a switch interface consisting of an LCON and a SIGA. The SIGA in turn communicates with a T-bus, which every function board must have. Thus, a minimum requirement is: switch interface, T-bus and TCS slave. To this skeleton can be added a processor, memory, and a VMEbus interface. Doing so results in the first function board implemented, and the only function board available in the original model of the machine, the TC/FPV board. The TC/FPV therefore provides an example and the focus for most discussion of function boards. The design permits, however, a function board with two processors, with memory and no processors, with specialized hardware such as an array processor, and so on. Whatever appears on a function board, it must conform to the architectural requirements such as support of locked accesses.

Figure 1-8 shows the block diagram of the TC/FPV. In the original model of the TC2000 machine, all function boards are TC/FPVs.

Figure 1–8 **TC/FPV function board.**



The Butterfly Switch

The switch interconnects all function boards, and provides signals such as clock and the TCS communication line.

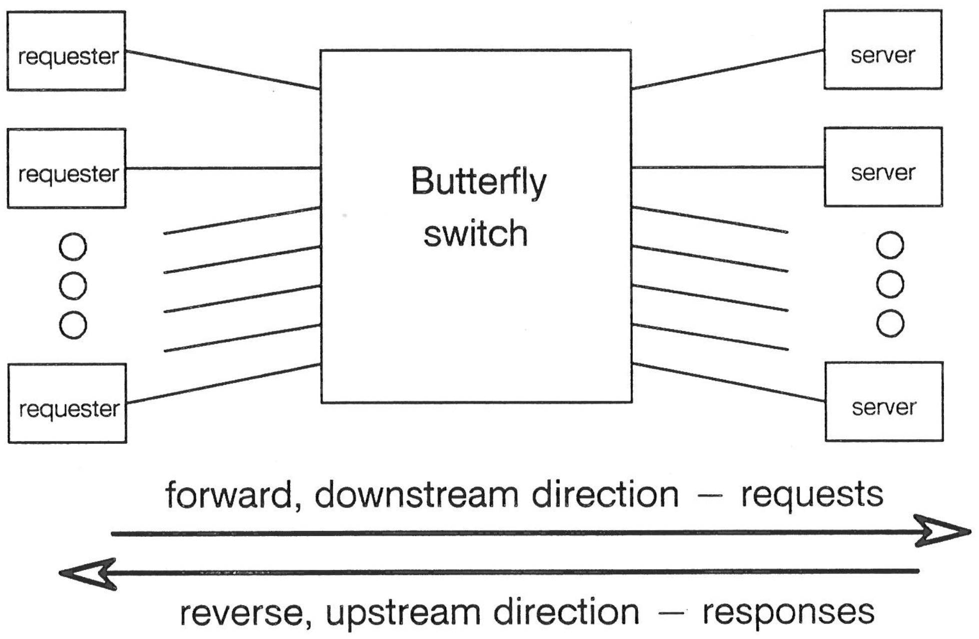
A connection through the switch from one function board to another is called a **switch path**. When the requester SIGA on a function board injects a message into the switch to set up a path, each switch element in turn along the path examines the **route** contained in the message, and determines whether it can complete the required link from itself to the next element on the path. If so, the message is forwarded on that link, and proceeds step by step through the switch, ultimately arriving at the end of the route, where it enters the destination function board. Memory or other resources accessed over the switch are called **remote**, because they are not local to the function board originating the access. Of course, those same resources are **local** to the processor on that other function board. A "remote" access is any access made over the switch; a "local" access is one that stays on the function board.

In the switching network of the TC2000 computer, the route a message takes is the same as the address of its destination. A message addressed to function board "C", injected by any function board in the machine, will arrive at board "C". In practice, certain configurations of the TC2000 switch permit more than one route to a destination slot. In such a configuration, the hardware automatically tries the alternate paths, resulting in improved performance when the switch handles very heavy traffic.

The individual switching element is implemented in a custom gate array chip called the Switch Gate Array (**SGA**). SGAs work in groups of four, and each quartet accepts eight input lines coming from the requester direction, and provides eight output lines going in the direction of the server. Any of the input lines can connect to any of the output lines, and up to eight connections can be carrying data at once. While connected, only one input connects to a given output, and only one output to any input; there is no fan–in or fan--out. In a simple analogy, the SGA acts as a telephone switchboard for single–subscriber lines; it does not do party lines or conference calls.

Once set up, the requester's message is delivered to the server function board, where it is acted upon. If a response is produced — namely, the data from a read operation — the server sends it back over the same switch path. This is possible because the switch path is **bidirectional**, as illustrated in Figure 1–9. A control signal ("reverse") tells each SGA along the path to listen on its server side, and transmit the data out its requester side. The switch path is held open until the requester releases it by use of another control signal ("frame"). This is like hanging up a telephone, because all switch components that were dedicated to the path are freed for use by other traffic.

Figure 1–9 Bidirectional switch concept.


requester
requester
requester
Butterfly switch
server
server
server
forward, downstream direction – requests
reverse, upstream direction – responses

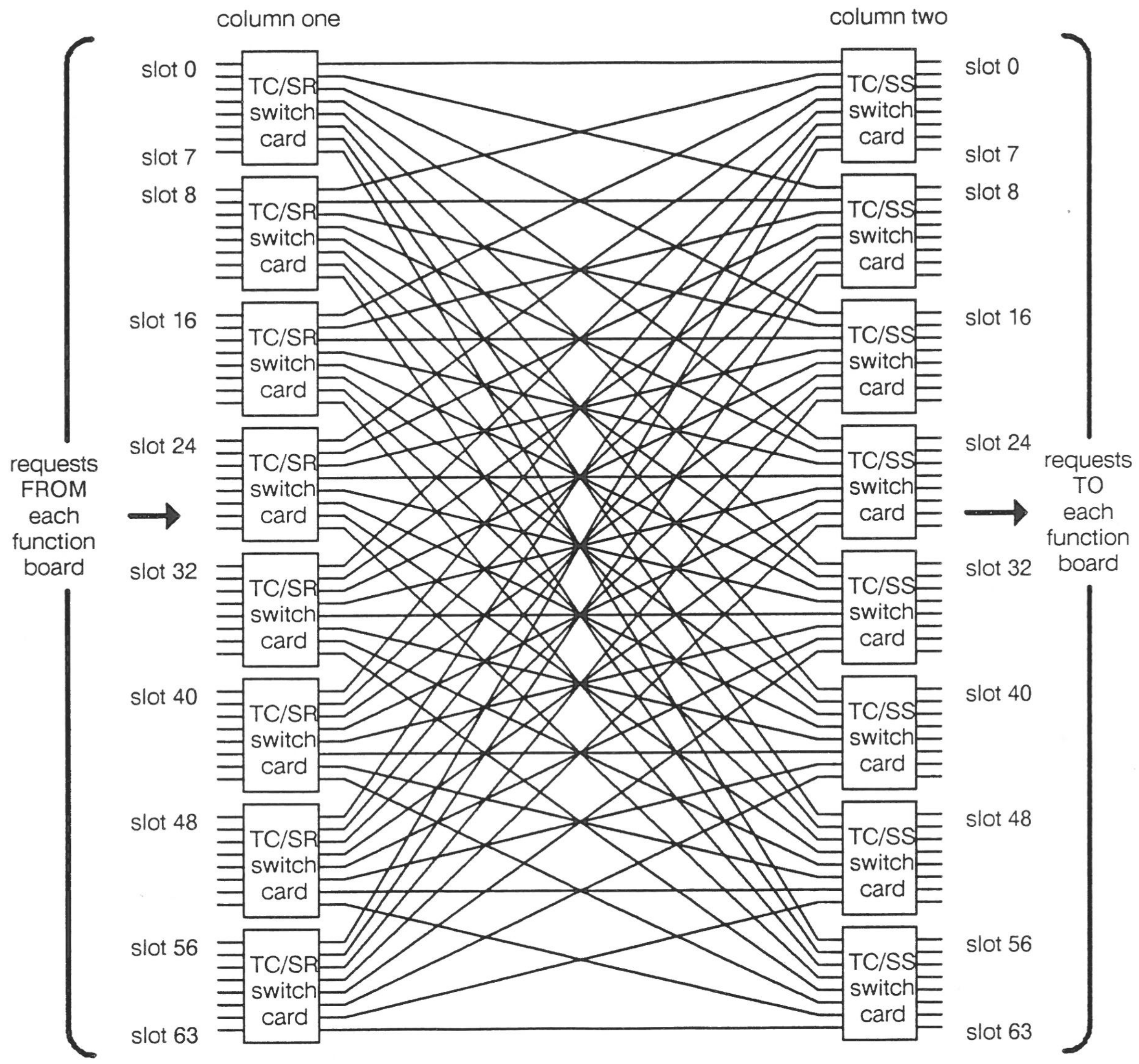
The switch supports **locking**. During a locked transaction, the requester SIGA keeps the switch path open until told to release it by the CPU or I/O interface that made the request. This allows several requests and replies to flow on the connection before it is released.

The switch hardware, in cooperation with the SIGAs on function boards, automatically detects and deals appropriately with several conflict or error conditions that can arise during operation. If an SGA finds a requested link is busy, it indicates a "reject" condition to the previous SGA, causing the SGA switching resources along the partially acquired route to be released. That is, a message encountering contention within the switch **backs out**, rather than holding switch resources while it waits for completion of its path. The requester SIGA holds a copy of the message, and **retransmits** a rejected message until it is serviced. Retransmission is controlled by a **backoff** algorithm that pauses longer and longer between retransmissions of the same message, easing congestion. If the hardware is broken, the retransmission count reaches a limit and triggers an error mechanism. Similarly, a connection that is held beyond a timeout limit causes an error condition, and the switch resources it was using are released.

Further, the hardware supports an **express** message facility to limit switch latency. In a switch with backoff and retransmission, there is a chance of very heavy traffic causing a message to be rejected many times. This delay is undesirable, and the express mechanism periodically promotes the priority of each retransmitted message so it will get through the congestion and achieve its connection. And finally, each message is protected from corruption by a checksum–like code.

Conceptually, one quartet of SGAs could connect up to eight function cards to each other. For a machine of more than eight function boards, a connection must travel through additional layers of switching. Each such layer is called a **switch column**. In practice, the minimum switch implemented is a 2–column switch. Fully populated, a 2–column switch interconnects up to 64 function boards ("slots"), as shown in Figure 1–10.

Figure 1–10 **Two–column, 64–slot switch.**



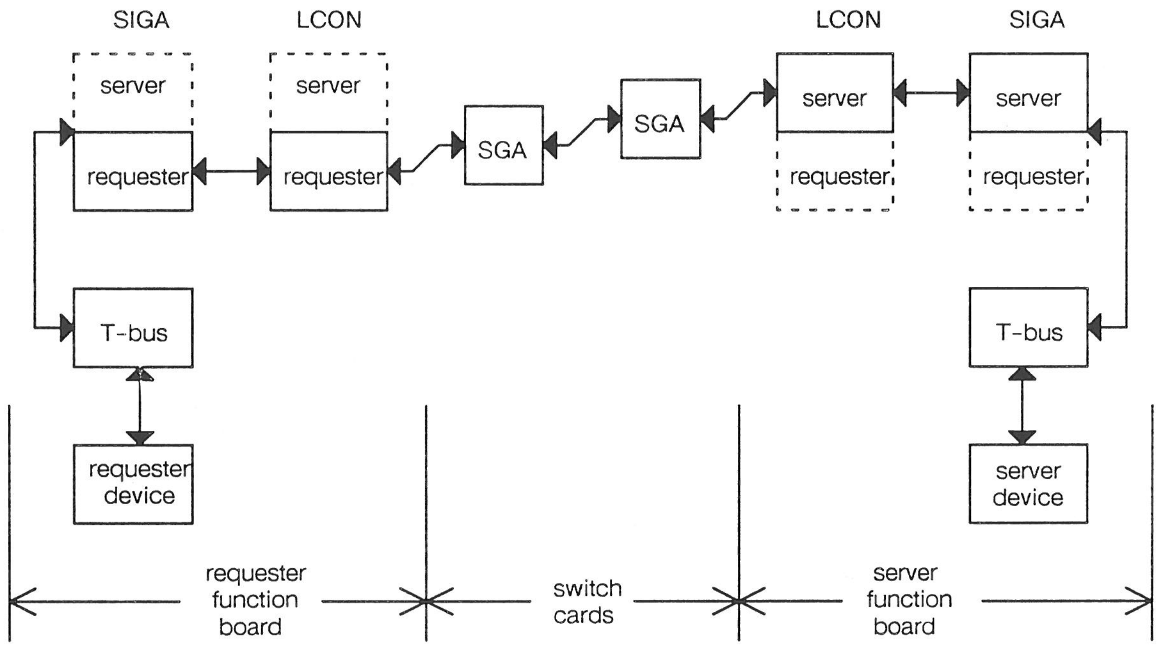
NOTE

63 OR 64 SLOTS — TECHNICAL AND HISTORICAL DETAIL
A 2–column switch supports 64 slots. An early version of the TC/FPV could not be used in slot 0, because that was used as one way to address devices on the local board. That led to the limit of 63 slots in the initial machine. The TC/FPV does not have this restriction, so a 64–slot machine is possible. The TC/FPV can be used in a backward–compatible, 63–slot mode.

A quartet of SGAs is implemented on one **switch card**. Thus, each box in Figure 1–10 represents one switch card. The switch cards on the requester side and server side are very similar in function but differ slightly in implementation, leading to two kinds of switch card. The requester side switch card is the TC/SR, and the server side switch card is the TC/SS. Each midplane, and thus each expansion cabinet, contains one TC/SR and one TC/SS. This **switch card pair** provides switch connections for the eight function boards on that midplane.

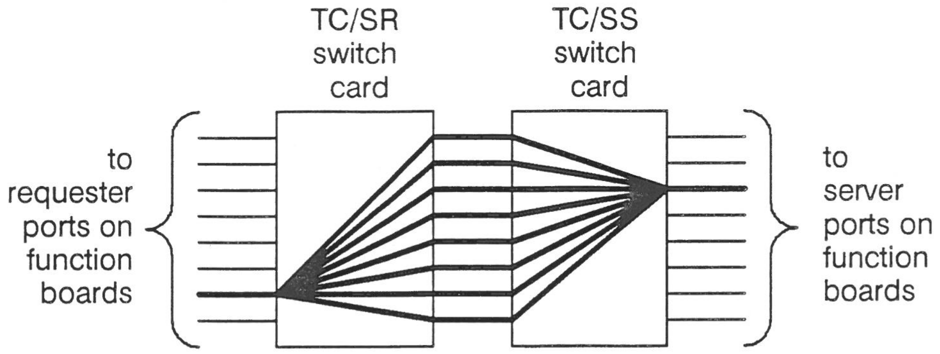
Figure 1–11 shows the various components that are involved in a connection from one function board, through the switch, to a remote function board. If the requesting function board addresses its own slot, the connection comes back into the same function board via its switch server interface.

Figure 1–11 **Resources used in a switch connection.**



In a machine with only one expansion cabinet, there is just one switch card pair, and each output of the TC/SR is connected to an input of the TC/SS. This is an example of the multiple switch paths mentioned earlier. No matter which of its eight outputs the TC/SR selects, the TC/SS can connect it to whichever of the eight function board server ports a message addresses. Figure 1–12 shows as heavy lines the eight alternate paths between one function board and another.

Figure 1–12 **A switch with alternate paths.**


TC/SR switch card
TC/SS switch card
to requester ports on function boards
to server ports on function boards

The Clock

The **clock card** (TC/CLK) provides the master clock signals for the switch and the switch interfaces in function boards. Each function board (specifically, the TC/FPV) has its own processor clock, and separate clocks are used in the TCS master and slave processors. Each VMEbus system is asynchronous to all of these.

Synchronous operation of the switch requires precision in distribution and use of the clock signals. The signal generated by the clock card is sent to the switch cards, where it controls the operation of the SGAs. From the switch cards, clock signals are further distributed to the function boards on the associated midplane. There, the clock signals are used in the LCON and SIGA.

In a large machine, the cables between the TC/SR cards in one expansion cabinet and the TC/SS cards in another cabinet may be relatively long — so long that the data may not reliably traverse the cable in one clock period at the clock rates used in the TC2000 switch. Therefore, the clock card generates two separate clock signals, one for the requester side of the switch and one for the server side. The TC/CLK can be configured to make these either matched or 180 degrees out of phase, providing more time for the data to traverse the switch cables between cabinets. Thus, each LCON and SIGA receive and use two clock signals — **requester clock** and **server clock**. This complexity is handled completely by the hardware, and is of interest to the programmer only to understand the design and implementation.

The switch clock rate is programmable, and is set by the TCS master during system startup.

In each SIGA is a register called the **Real Time Clock** (RTC). This register is incremented at a constant rate of 1 megahertz, derived from the switch requester clock. Because the switch clock frequency is programmable, the SIGA contains a programmable prescaler that appropriately divides down the switch clock to 1 megahertz. A machine–wide signal ("65 milliseconds", because it is a pulse once every 65.536 milliseconds) is produced by the TC/CLK and used in the SIGA in maintaining and synchronizing the RTCs across the machine. The RTC is used for timer–generated interrupts and general software purposes. Its accuracy is determined by the oscillator on the TC/CLK. For a long–term time and date clock, the operating system relies on a different source, such as the TCS.

Besides the switch clock and 65–millisecond signals, the TC/CLK generates a switch–wide signal ("hold") used in the implementation of express messages described above. The TC/CLK also connects directly to the TCS master, fans out the TCS communication line to all the switch card pairs, and fans in the TCS communication from them.

Test and Control System

The **Test and Control System** (TCS) oversees operation of the TC2000 machine. It monitors and controls all aspects of the hardware. The TCS is best thought of as connected only to the TC2000 hardware, and not to the software. Only through conventions established in software is there any communication between the TCS and the operating system or application program.

A master–slave design is used for the the TCS. A single microcomputer master communicates with several microprocessor slaves distributed throughout the machine, over a serial communication line. The **TCS master** is an IBM PC/AT compatible microcomputer with peripheral devices appropriate to its role in controlling the rest of the machine. Each **TCS slave** is a microcomputer chip well suited to monitoring and controlling a variety of signals. There is one TCS slave on the clock card, on every function board, and on every pair of switch cards. Through these, the TCS master has full access to the entire machine.

The TCS master contains a CPU, memory, hard disk and floppy disk drives, interface card, and optional modem. Connected to the TCS master proper are the front and back panels of the TC2000 machine, and a terminal. The **CPU and memory** are ordinary IBM PC/AT compatible components; noteworthy, however, is that it contains a real time clock with **calendar** and **battery back–up power**. This provides a non–volatile and more accurate date and time source than the real time clock feature of the SIGA chips in the switch.

The TCS master's **hard disk** drive is used to store machine configuration and operating parameters, TCS log data, TCS master software, and diagnostics. Some of the diagnostic test and exerciser programs are for the TCS master to test the Test and Control System itself; others are executed by the TCS master and slaves to test the TC2000 hardware; and yet others are executed on the function boards, under supervision of the TCS master, to test the TC2000 hardware. The TCS master's **floppy disk** drive is used primarily to load new TCS software.

The **interface card** (TC/TCS) connects the TCS master to the rest of the machine. It contains not only the TCS communication line interface, but also hardware to turn on and off the main power in all the expansion cabinets and the I/O power in peripheral cabinets, and connection to the machine's front and back panels. The interface card also houses read–only memory (ROM) that contains TCS master startup and error routines, and a watchdog timer that resets the TCS master if its software gets hung. The interface card is the only custom equipment in the TCS master.

The **modem** connects to a telephone line for remote test and diagnosis of the machine. This modem and line permit faster analysis of, and response to, problems with the hardware. At a site with security requirements in conflict with such facilities, no telephone line is hooked up.

Figure 1–13 shows the TC2000 **front panel**. Four light–emitting diodes (LEDs) indicate overall machine status. The "main power" LED is controlled by the TCS master, and is lit when the TCS master turns on main power to the rest of the machine. "TCS enabled" is lit when the TCS communication line between the master and slaves is ready for data transfer. "TCS power" is lit when power for the TCS — both master and slaves — is on. The "attention required" LED is under control of TCS software to bring the operator's awareness to conditions that need attention.

Above the front panel LEDs is a hinged door, behind which lie the keyswitch, the reset button and the streaming tape drive. The **keyswitch** "off" position turns off all power to the machine; the "on" position powers the TCS, which can then turn on power to the remainder of the machine. The "secure" position disables the reset button, and is otherwise the same as the "on" position. The TCS master can read whether the keyswitch is in its "secure" or "on" position, and can inhibit certain TCS commands, based on keyswitch position. The **reset button** forces a reset of the TCS master microcomputer. It does not directly affect operation of the rest of the machine. The **streaming tape drive** is used to load or dump TC2000 system software. A button at its upper right corner opens the drive door, and an LED in its lower left corner indicates the drive is in use (so the door should not be opened).

Figure 1–13 **Front panel.**

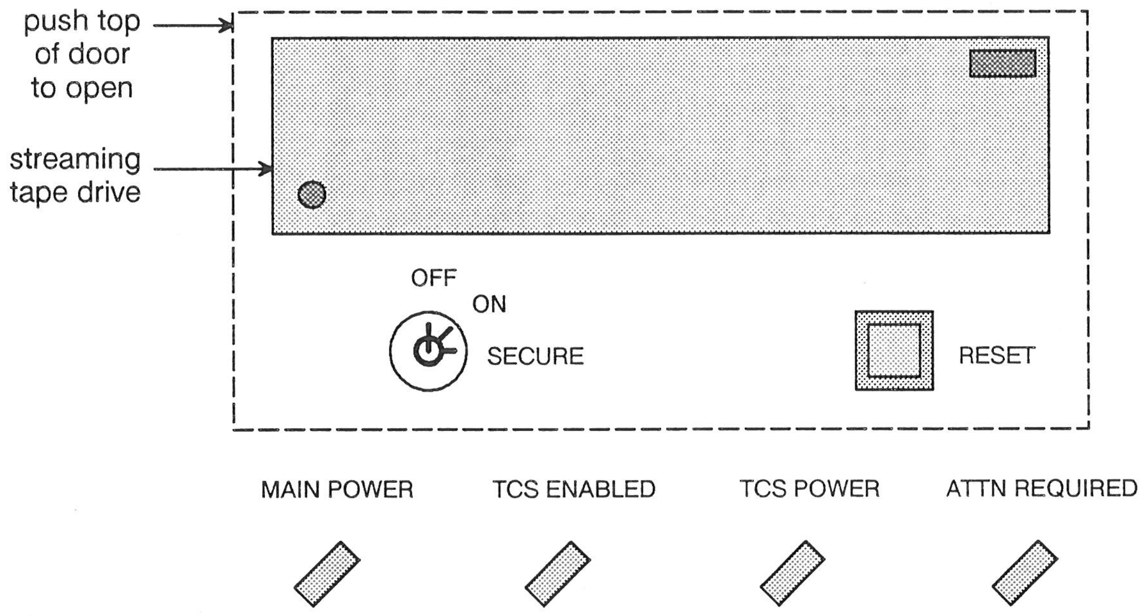
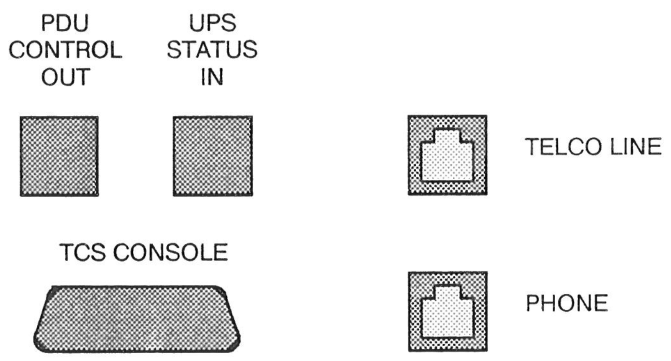


Figure 1–14 shows the TC2000 **back panel**, located at the rear of the utility cabinet. The TCS console jack connects to the TCS master **terminal**. The standard TCS master terminal is a DEC VT320. The "PDU control out" jack supplies control from the TCS master to power distribution units (PDUs) in other cabinets. The "UPS status in" jack is for a future, optional capability whereby the entire machine is powered by an Uninterruptible Power Supply (UPS), and the TCS senses its status. The two modular telephone jacks are for the optional diagnostic line, and a telephone for voice communication on that line.

Figure 1–14 **Back panel.**



The **TCS slave processor** is a microcomputer (68HC11) especially intended for monitor and control applications. It contains the serial line interface necessary to communicate with the TCS master, on–chip memory, outputs for digital control, and inputs for sensing status. The input capability includes both digital and analog channels; the latter are used for monitoring temperature and power supply voltages. The slave processor is responsible for the entire environment of the board on which it is mounted. (Or, in the case of switch card pairs, the *board pair*. The switch server card (TC/SS) contains a TCS slave that controls both the TC/SS and the paired TC/SR requester card.)

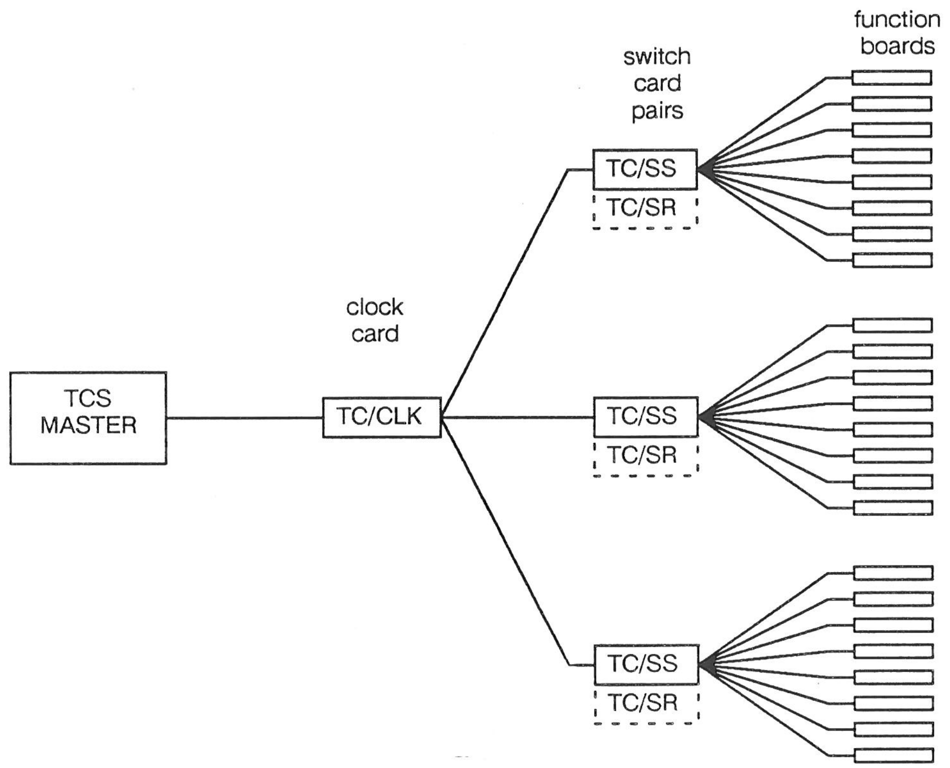
Each TCS slave receives its own power from the "TCS + 5" power supply. This supply is separate from the power for each cabinet, and the power for each board, so the TCS can operate even when the rest of the board is powered off. An expansion cabinet's main, or bulk, power is ±24 volts; once this is available, the TCS slave can enable the power supplies on its own board. These convert the bulk power into the voltages required on the board. Besides enabling board power, the TCS slave can also modify ("margin") the voltages of board power for testing purposes.

Besides temperature and power, the TCS slave monitors and controls digital signals and parameters of various sorts dependent on the nature of the board. On a TC/FPV function board, the TCS holds the 88000 processor reset while it initializes the board hardware. It loads appropriate bootstrap code into the memory and releases the reset condition, allowing the 88000 to begin executing. It provides a low-level communication path between the operating system and the TCS master software. On the TC/CLK clock card, the TCS slave sets parameters such as switch clock frequency. On the TC/SS and TC/SR switch card pair, the TCS slave can enable and disable individual switch paths for test and diagnostic purposes. These are merely examples; the full range of TCS slave capabilities is significantly greater.

The **TCS communication line** connects the TCS master to all TCS slaves. The protocol used is query–response, in which the slaves never volunteer information but rather transmit only in response to a command from the master. All slaves receive all commands, and recognize when the command applies to them. Conversely, all data transmitted by slaves is funnelled to the master. The simplest way to think of this is as a shared communication bus; so, although its implementation is more complicated as we shall see, it is called the **TCS bus**.

In implementation, the TCS bus is more of a tree, fanning data from the master out along its branches to the slaves, and fanning in data from the slaves. The master is attached directly only to the clock card, as shown in Figure 1–15. Data from the master goes directly to the clock card slave, and is fanned out (relayed, repeated, buffered) to the midplane in each expansion cabinet, where it goes to the TC/SS of each switch card pair and to the eight function boards. The slave on each TC/SS serves the paired TC/SR, which has no slave.

Figure 1–15 **TCS bus fan–out and fan–in.**



In the other direction, data from function board slaves is fanned in (collected, combined, multiplexed by OR'ing together) on its way to the master. Data from the function boards on a midplane is combined, at their TC/SS card, with data from the slave on the TC/SS. Data from every TC/SS is combined, at the clock card, with data from the TC/CLK's slave, and sent to the master. This fan–out and fan–in scheme achieves machine–wide communication with very simple wiring.

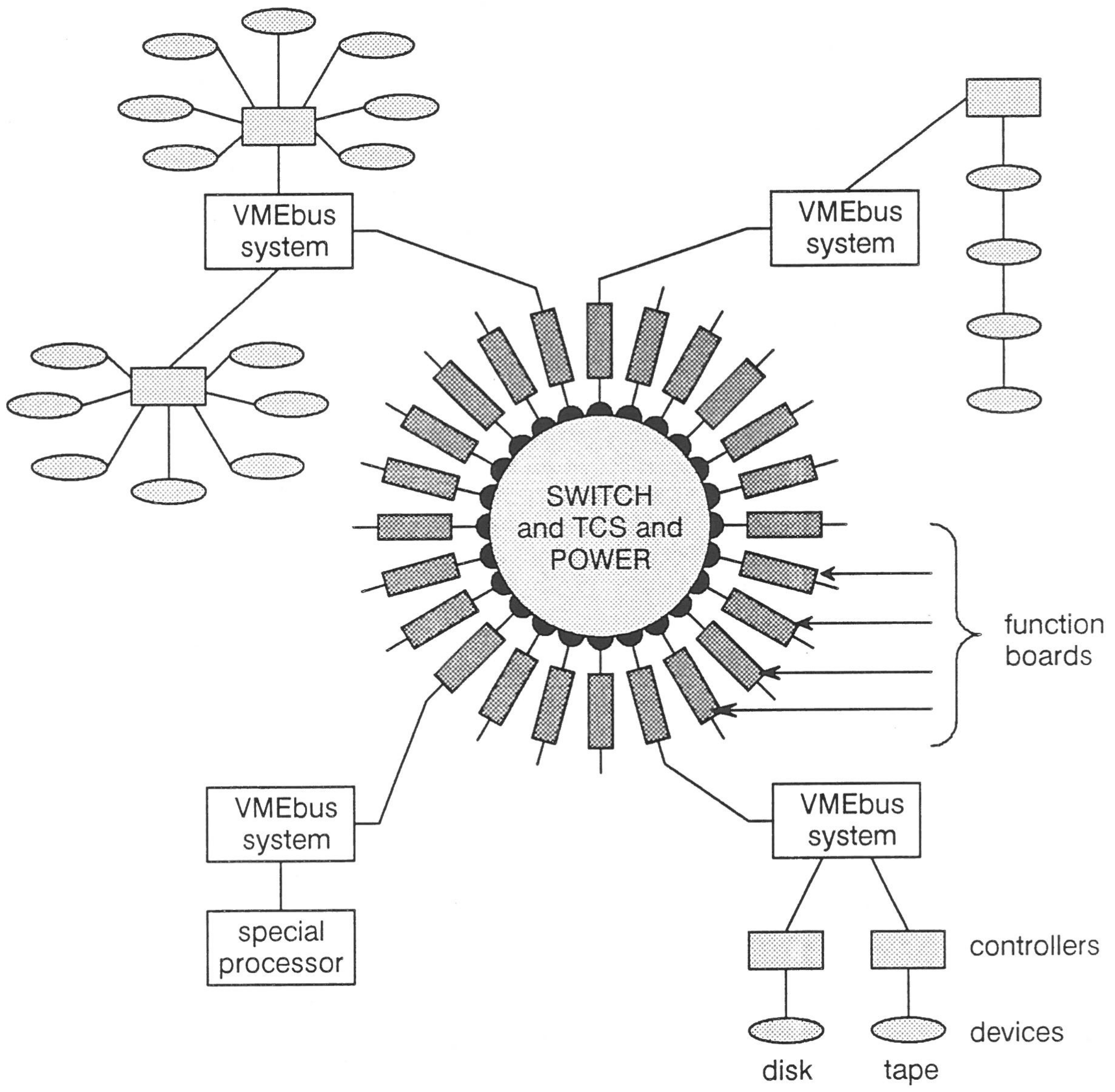
On the other hand, such a communication tree can be vulnerable to data corruption if components fail. To protect against this, the TCS bus design buffers each fan–out signal separately at the clock card, and contains programmable gating of fan–in signals. The TCS master can use this gating to selectively disable — amputate — any branch of the fan–in tree that is corrupting the TCS bus. This fault isolation mechanism permits the machine to operate in a reduced configuration, consistent with the machine design philosophy of reconfigurability to work around failed components.

I/O Capabilities

The vast majority of TC2000 I/O is via one or more VMEbus systems. Figure 1–16 presents the TC2000 architecture to emphasize the I/O. At the core is the switch, TCS and power supplies. Connected to this core via switch ports are function boards. Connected to some of the function boards are

VMEbus systems. Each of these VMEbus systems can be configured as required for the particular site.

Figure 1–16 **I/O is VMEbus systems via function boards.**



The devices normally available include:

- Terminal multiplexor
- Hard disk drives
- Magnetic tape drives (½-inch width tape, on reels)
- Streaming cartridge tape drive (¼-inch width tape, in cartridges)

- Ethernet interface
- Controllers for Small Computer System Interface (SCSI) devices

A given installation may have site-specific requirements for other VMEbus equipment. In general, adding or interfacing to other VMEbus equipment is straightforward. The TC/FPV function board's interface to the VMEbus is highly configurable under program control. This flexibility permits the TC/FPV to perform various roles, depending on the requirements of the particular system. For further information on support of specific equipment, please contact BBN ACI. The customer may need to provide device driver software if unusual or non-standard devices are to be supported.

For completeness, we briefly name the other I/O capabilities of the machine.

- Primary of these is the TCS, which can read and write floppy disks, and communicate via its terminal. The TCS terminal serves as the system console for the nX operating system on the TC2000 machine, and may be accessed from application programs if necessary. Access to the TCS floppy disk is currently restricted to the TCS master, although the software design includes, and the hardware permits, the TCS master to service requests from the nX operating system and application programs to do MS-DOS file system I/O.
- Various LED indicators show machine state. Besides the front panel LEDs described above, green LEDs on each switch card and each function board indicate the presence of power at various voltages. An amber LED on each of these boards is under control of the TCS slave, and can be used to identify a particular board during maintenance. The TC/FPV function boards also have green LEDs indicating the transmission and reception of switch messages. These can be useful as a rough indication of the level and location of switch activity.

For further information on the TC2000 I/O system, please refer to chapter 5.

1.4 Machine Specifications

1.4.1 Computational Specifications

The following items describe the first model of the TC2000 computer. In particular, they reflect the original TC/FPV function board.

- CPU clock rate: 20 megahertz
- Memory
    - Size: 4 megabytes per TC/FPV function board, expandable to 16 megabytes; TC/FP function board available with either 16 or 32 megabytes per board; $64 \times 16 = 1024$ megabytes maximum per ma-

chine, using any mix of the TC/FPV16 and the TC/FP16 (excluding any peripheral memory on the VMEbus)

  - Parity: one parity bit per byte, on main memory only; special feature for testing parity logic
  - Error rate: 3.17 years estimated mean time between detected soft errors per 4–megabyte TC/FPV
  - Dynamic RAM refresh cycle: automatic

- The design nominal time to access local memory with no contention is 0.55 microseconds for a read, and 0.60 microseconds for a write. Access times depend on several conditions: local or remote (over the switch); read or write; single word or multiple word; cache hit, cache miss or cache inhibited, and copyback or writethrough caching policy; use of fast path from CPU to local memory; within machine or to VMEbus address space; and amount of contention for the resource. Access times for many combinations of these conditions are discussed in chapter 2.
- Switch clock rate: 38 megahertz
- Switch bandwidth: 38 megabytes per second per path, peak
- VMEbus throughput: 8 megabytes per second per VMEbus interface, to a machine total of 320 megabytes per second (current architecture supports up to 40 VMEbuses)

1.4.2 Environmental Specifications

For the following and related information, please refer to the TC2000 *Site Planning Guide*.

- Physical dimensions, weight and servicing clearances
- Flooring recommendations, for both air flow and cables
- Power line requirements — voltage, amperage, phasing and grounding
- Power consumption, heat dissipation and cooling requirements
- Temperature and humidity specifications

CAUTION

Avoid placing objects on top of the cabinets, where they could impede the flow of cooling air. It is good practice to maintain the machine's operating temperature margins by not blocking the air flow.

1.5 Equivalent but Distinct Function Boards

This section describes how software conventions, or the equipment connected to function boards, or both, distinguish some function boards from others.

1.5.1 The nX Master Function Board

From a **hardware** viewpoint, *all TC/FPV function boards are equivalent*, any one may be installed in any slot, and there perform all functions that any other TC/FPV would perform.

From a **software** viewpoint, the nX operating system distinguishes one TC/FPV function board as the **master** function board (or master "processor node"). This distinction is based on three properties:

- The **design of the nX operating system** dictates that certain functions be performed or coordinated by one specific processor, and that certain data structures reside in the memory local to that processor.
- The TCS master must pick some TC/FPV function board on which to load and start the nX bootstrap. For the sake of using **a specific board during booting**, the TCS uses a particular function board named in a machine configuration file.
- Bootstrapping the nX operating system from the system hard disk is **simpler and faster** when the nX boot disk is on the same VMEbus system as the master node. Therefore, as a practical measure, the master node must be a TC/FPV function card so connected.

When the machine is booted, the TCS master software (TEX) reads a configuration file from the TCS hard disk. Among other parameters, this file specifies the slot number of the master. In a normal system this is 7.7.7, in the format "**bay.midplane.slot**" used to specify a particular function board. TEX loads a bootstrap program from the TCS hard disk onto the function board in this slot, via the TCS slave on that board, and starts the 88100 CPU executing it.

The bootstrap loads the nX system from the nX system disk, over the VMEbus attached to the function board. Once loaded, the nX software starts up.

When the nX operating system is running, certain software functions reside on the master. Data structures associated with those functions usually reside there also, merely as a performance issue, because of the speed advantage in accessing local memory. There is little or no absolute requirement about which functions reside on the master. Requiring that only the master execute a given function is a simple and easy way to achieve synchronization among the many processes, executing on many processors, that might invoke the function.

However, when a function must be performed only by the master, there is potential for the master becoming a bottleneck. The processing speed of the master, access to lock–controlled resources on the master, switch (interprocessor communication) bandwidth to and from the master, or I/O capacity of the master could limit overall machine performance. Considerable care has gone into programming and configuring the nX system to make such bottleneck effects small and infrequent.

For example, the following functions were once executed only on the master, but have been parallelized by recoding, and can now execute elsewhere:

- Most file system operations: read, write, lseek, etc.
- Most Ethernet I/O operations
- Process scheduling

1.5.2 Clusters

The nX operating system supports the concept of **a cluster of function boards**. Loosely, a cluster is a collection of function boards viewed as a computing resource available for some particular purpose. The **system cluster** contains all function boards in the machine. The **public cluster** contains one or more functions boards that run your shell when you first log in. The **I/O cluster** contains the nX master function board, and typically no others. Permission to run processes in the I/O cluster is restricted to groups "wheel" and "root", to prevent undue competition for its memory and CPU. The occasional nX system call that is serialized traps to the master, and a parallelized call may trap to the master to perform physical I/O to a device attached to the master. Certain I/O intensive activities, such as dumping to tape, are usually run on the master. Function boards in the **free cluster** are unallocated and may be allocated by system calls or by nX commands. A user invokes such a command, or his program makes such a system call, to allocate function boards into the user's own private cluster, in which the user may execute programs. A cluster of **bare nodes** is a resource set aside when the nX operating system starts up, from which clusters may be allocated to run programs under the pSOS^{+m} real–time operating system.

Clusters are an important, powerful and flexible capability in the way the TC2000 machine is used. Since clusters are an aspect of the software, not hardware, they are not discussed further in this document. For further information on clusters, please refer to nX documentation.

1.5.3 Physical Slots

When specific hardware — in particular, a VMEbus system — is connected to a particular function board, the application software must be able to refer to that function board specifically, and allocate it specifically. The system software supports this capability.

1.5.4 Slot Numbering

The physical card slots in the TC2000 machine are normally numbered in "dotted octal" format:

<bay>.<midplane>.<slot>

such as:

7.1.6

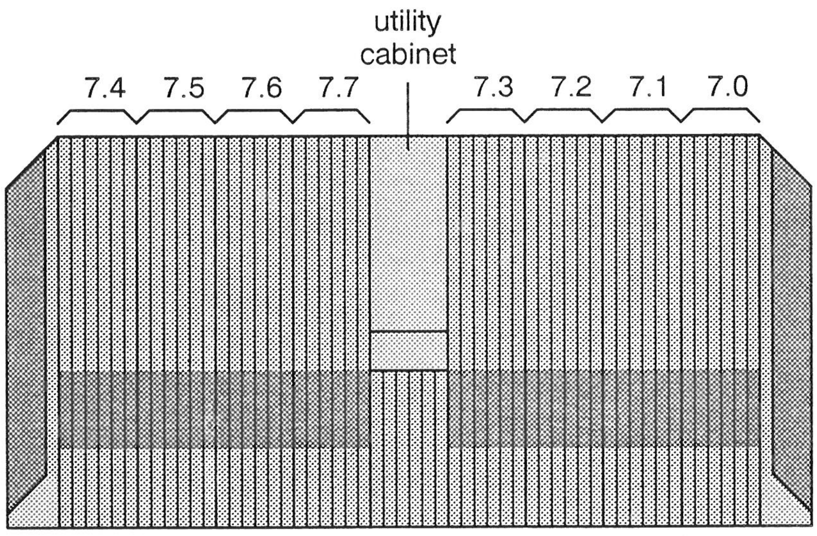
The first field in dotted octal format, here "7", specifies the **bay** in which the slot appears. Each bay holds up to 64 function boards. In a machine with 64 or fewer function boards, the entire machine is by convention bay number 7.

The second field, here "1", specifies the **midplane** to which the slot connects. Each midplane serves up to eight function boards.

The third field, here "6", specifies the particular **slot** on the midplane.

Figure 1–17 shows the standard midplane numbering in a 64–slot machine. If the machine has fewer midplanes, midplanes are omitted in the order: 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6. A one–midplane machine has only midplane 7.7.

Figure 1–17 Midplane numbering.



The above format applies in a clear way to all function boards in a machine. The bay, midplane and slot fields can each be octal digits 0 through 7. The TCS master, however, must refer to other cards as well — switch requester and server cards, and the clock card. In a machine larger than 64 (function board) slots, there are second–stage clock cards and middle–column switch cards. The TCS uses a natural extension of the dotted octal format to refer to these cards, by extending the "slot" field above the value 7. The TCS also provides ways for the operator to refer to classes of cards, such as "all switch requester cards". For details, please refer to the TCS documentation.

Aside from physical slot numbering, the operating system sometimes refers to function boards by logical number. The logical number is normally in decimal, and the mapping from logical number to physical slot varies depending on the current configuration of the machine (which physical slots are occupied) and how it is being used. Logical number may refer to a machine-wide set, an operating-system-wide set, or a cluster-wide set, depending on the context.

2 The TC/FPV Function Board

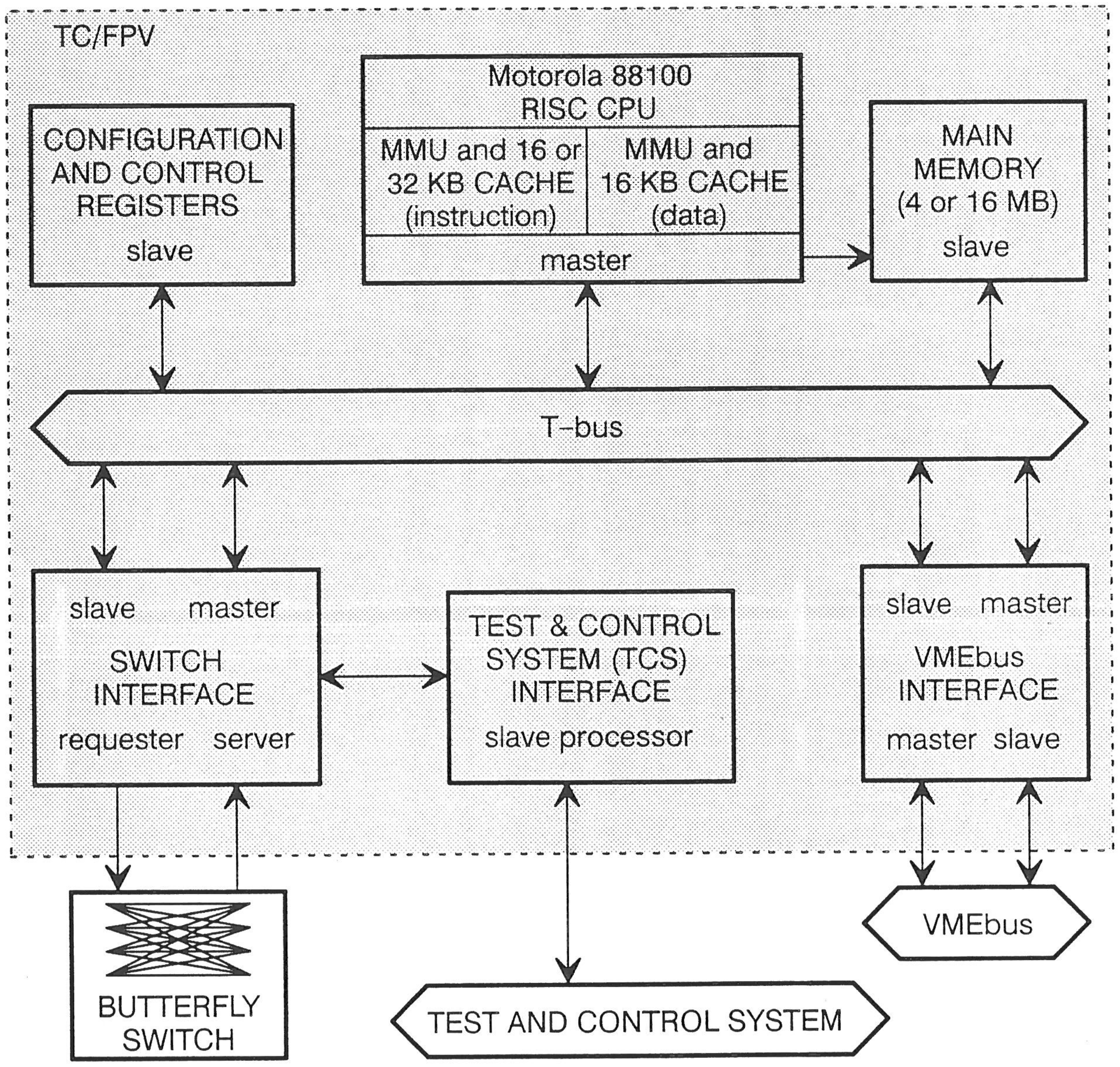
2.1 Organization

The TC/FPV function board is available in various configurations:

version number	memory (megabytes)	number of CMMU chips
TC/FPV4	4	2
TC/FPV4-1	4	3
TC/FPV16	16	2
TC/FPV16-1	16	3

The TC/FPV function board is a bus–based subsystem of its own. The transaction bus (**T–bus**) interconnects the functional blocks shown in Figure 2–1 (also shown in chapter 1, and reproduced here for ease of reference).

Figure 2–1 TC/FPV block diagram.



The T–bus itself is a high-performance bus with a master–slave protocol. Any given functional block may have a master interface, a slave interface, or both, on the T–bus. The T–bus does not extend beyond the function board. Addresses on the T–bus are System Physical Addresses (34 bits), described further below. Data transfer operations are read and write. A T–bus slave can refuse an access, and force a pause during an access, or can promise to return the requested data later. Data transfer size is byte, halfword (two bytes), word (four bytes), two words, three words or four words. (The current hardware does not use 2– or 3–word transfers, but 4–word transfers occur often, to fill and write back CMMU cache lines.) The T–bus has 32 data lines, and supports the machine's locking mechanism.

The other functional blocks of the TC/FPV are discussed in sections below.

2.2 Processor, CMMU and CPU Interface

The processor is a Motorola MC88100 chip. This microprocessor features a Reduced Instruction Set Computer (RISC) architecture, pipelining, a separate floating point execution module on the chip, and separate interfaces for data and instructions. Two Motorola MC88200 Cache/Memory Management Unit (CMMU) chips support the CPU; one CMMU services data transfers, while the second services instruction fetches, often simultaneously. The TC/FPV can be configured with a second, optional instruction CMMU chip. The CMMUs, and the nX operating system, support demand paging. The TC2000 page size is 8 kilobytes. Since the CMMU implements a 4–kilobyte page size, the operating system software allocates two adjacent CMMU pages whenever a page is required.

The 88100 CPU has an instruction, **xmem**, that exchanges the contents of a CPU register and a memory location. During this operation, the CPU and CMMU assert a signal intended to hold the memory bus, and therefore make the xmem an atomic operation. The TC2000 hardware preserves the atomicity of xmem by holding the path between the CPU and the referenced memory, even if that location is on another function board. This is a special case of the TC2000 locking mechanism described in section 2.9.

For a full description of the 88100 CPU and the 88200 CMMU chips, please refer to the Motorola literature:

MC88100 User's Manual

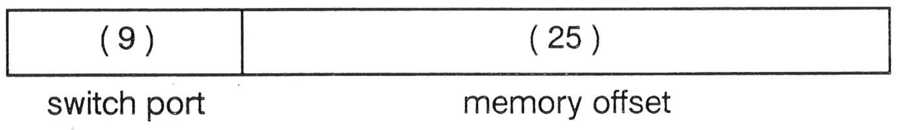
MC88200 User's Manual

The CPU and CMMU are based on 32–bit data and address words. Since the TC2000 address space is based on 34–bit addresses, address translation is necessary between the Motorola M–bus and the TC2000 T–bus. The CPU interface performs this translation, as well as generating other signals related to accesses the CPU makes. This translation and the other signals are performed by the CPU Mapping RAM, described in the following sections.

2.2.1 TC2000 Physical Address Space

The TC2000 architecture supports a global physical address space of 34 bits, for a maximum capacity of 16 gigabytes. At the lowest level, every byte in a TC2000 system has a unique System Physical Address with the format shown in Figure 2–2.

Figure 2–2 **System Physical Address.**



The size of the System Physical Address determines the maximum memory capacity of the TC2000 system. The structure of the system physical address determines the maximum number of function boards and the maximum memory capacity of a single function board. The size and structure of the system physical address were chosen to support the following system characteristics:

maximum system capacity	16 gigabytes
maximum number of switch ports (function boards)	512 ports
maximum function board capacity	32 megabytes

Increasing the maximum ***globally addressible*** memory of a function board beyond 32 megabytes requires changes to the T–bus specification and to the Switch Interface Gate Array. Memory that is ***only locally accessible*** may be added to a function board design with only on–board changes.

2.2.2 88000 Access to Global Memory

The series of transformations that convert an 88000 virtual address (Process Logical Address) to a System Physical Address is shown in Figure 2–3. The purpose of the CPU interface address transformation is to convert the 32–bit Physical Address generated by the 88200 CMMU to a 34–bit System Physical Address. The TC/FPV uses a mapping RAM to implement this transformation. The CPU Mapping RAM (CMR) is shown in Figure 2–4.

Figure 2–3 **Address transformation.**

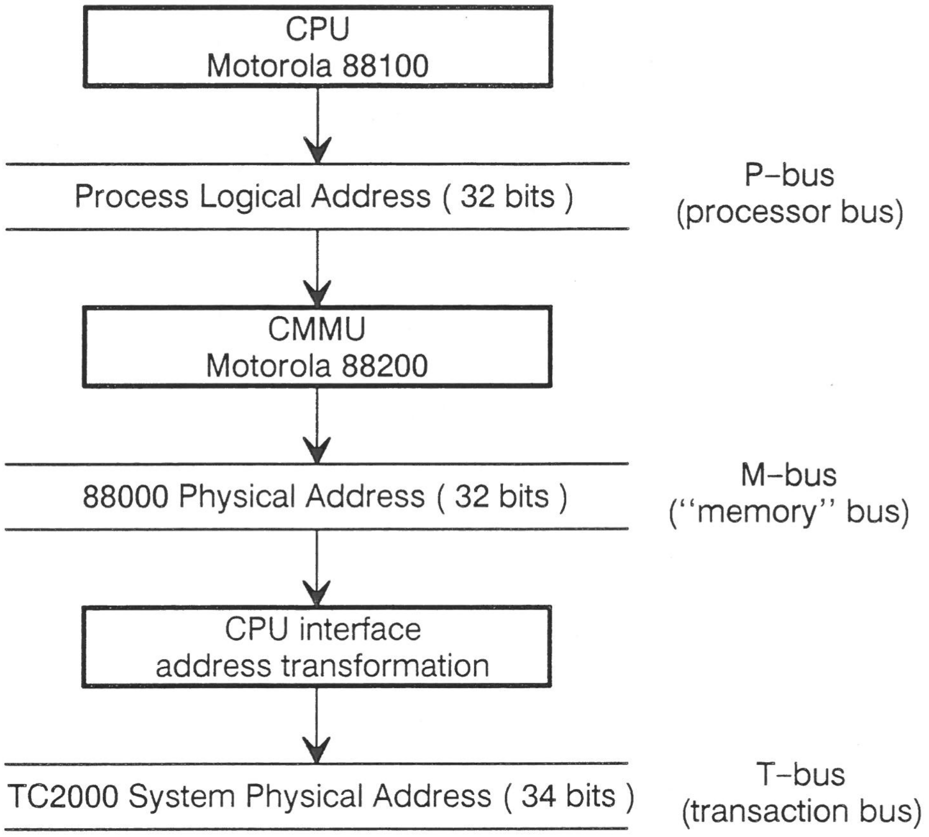
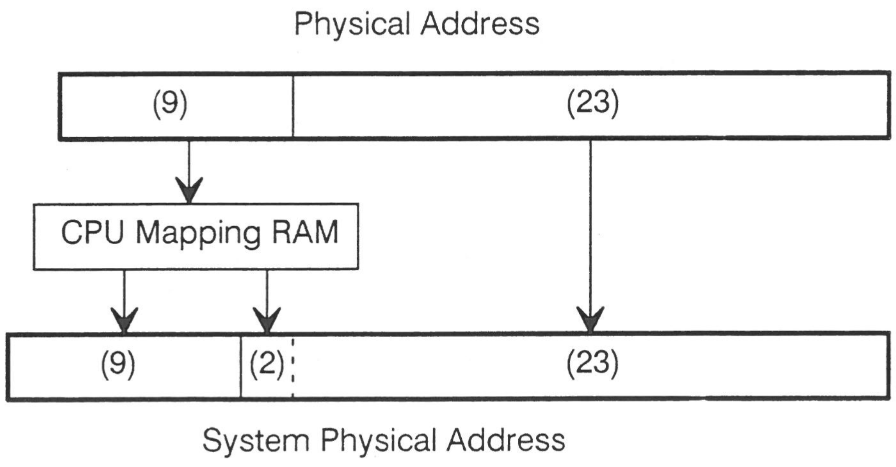


Figure 2–4 **Physical Address to System Physical Address.**



The CMR takes in the high nine bits of the Physical Address generated by the CMMU and uses it to generate the high eleven bits of the System Physical Address. These 11 bits select one of the 2048 8–megabyte "banks" of address space within the machine.

Note that even with this flexible mapping scheme, it is possible to build systems with more memory than the 88000 can address. A few observations should be made about the memory that lies outside the 4–gigabyte range that is directly accessible to all of the 88000s in the system.

1. There is no hardware restriction against setting the CPU Mapping RAM differently on different function boards. Thus it can be used to set up memory to store private copies of code and data. nX kernel code that must be replicated on every processor function board is one example. Programs with supervisor privileges can manipulate the mapping RAM to access private memory on remote function boards for diagnostics, initialization, debugging, and other purposes. This flexibility could also be used to allocate different subsets of physical memory to different clusters.
2. Some function boards may not have the 32–bit limitation of the 88000. For example, an array processor or high–throughput I/O function board could be built to access a 16–gigabyte memory space.
3. The 88000 addressing limitation might be removed in future generations of the part.

2.2.3 Block Transfer and the CMR Intercept Access Bit

A feature called **Intercept Access** is available in conjunction with the CPU Mapping RAM. The intended use of this feature is to reduce the number of switch accesses required when transferring blocks of data from one function board to another. Such use may enhance performance and reduce the asymmetrical nature of pushing versus pulling blocks of data around the TC2000 machine.

The MC88200 behaves asymmetrically around cache write misses when the page is marked copyback cacheable. This behavior is as follows. When the processor misses the cache on a cacheable write (in copyback mode), the 88200 selects a cache line. (If none are available, it selects one for replacement and if necessary, copies it back into memory.) It then reads in the new line with the intent–to–modify bit set. (This read is to force any snooping master with dirty data to flush it, a non–issue in the case of the TC/FPV.) Next, it writes the datum to memory, and finally it writes the datum to the newly–read cache line. See the *MC88200 User's Manual* for more detail.

The effect of the 88200's behavior in the TC2000 computer is as follows. For simplicity, the description assumes that the data is quad–word aligned. In the current model for copying blocks of data around the TC2000 machine, a block of data is read into the cache, and then written to another cacheable location. The process of cache line replacement, or explicit flushing, causes the data to be written to memory. In the case of copying data from local memory to remote memory (**pushing**), the behavior described above results in three ***switch transactions*** for each line of data to be copied:

1. The initial read with intent-to-modify of the cache line
2. The write-once
3. The ultimate burst write when the line is replaced in the cache

On the other hand, when copying data from a remote function board to local memory (**pulling**), the data is read into the cache across the switch, and the burst read, write-once, and burst write are performed to ***local*** memory. Hence, only one switch reference is required. In either case, the burst read and the write-once are extraneous. New data is immediately copied over the data that has been read in; further, the line is written to memory when it is selected for replacement, or when it is explicitly flushed, so the write-once is redundant also.

If the **Intercept Access** bit is set in a CMR entry, support circuitry in the CPU interface causes any cycle mapped through the entry to get intercepted. The processor is acknowledged as if the cycle had completed, but no T-bus cycle is actually generated. (On an intercepted read, the data returned is undefined and should not be used for computation.) The CMR has separate entries for read and write cycles, so reads and writes can be intercepted independently of each other.

Besides improving performance, the Intercept Access function can be used to make block copy behave in a symmetrical fashion. The Intercept Access bit should be set in the read entry for the destination function board. This reduces the number of switch references from three to two for each line in the pushing case, by intercepting the read-with-intent-to-modify cycle when the cached write occurs. In addition, if the destination line is read in (intercepted locally) before the data is copied to it, then a cache miss will not occur when the data is written to the destination. This will prevent the write-once from occurring, reducing the number of switch references to one per cache line, regardless of whether the data is pushed or pulled through the switch.

2.2.4 The Local Bit in the CMR

When the location being addressed is on the same function board, we say it is in ***local memory***. It is possible to access local memory across the switch, but almost always desirable to access it directly instead.

On the TC/FPV, bit 11 in each CMR entry indicates whether the memory referenced through that entry is local or remote. When the CMR asserts *local*, the T-bus T_PATH bits are set to specify a local access, so the value of the T-bus address bits T_AD <33..25> is irrelevant, but they are driven from the CMR onto the T-bus anyhow to supply valid electrical levels.

NOTE

HISTORICAL NOTES
In the B2VME function board, a predecessor to the TC/FPV, the CPU interface had no CPU Mapping RAM. There, the decision that an access was local was based on Physical Address bits 28..23 being zero. This zero-detection is *not* present in the TC/FPV, but can be emulated by appropriate setting of the CMR registers.

Also in the B2VME was detection of locality by "switch shortcut" logic that noticed when the switch port addressed was the same as the function board's own. The logic then caused the access to be serviced locally instead of using the switch. This logic could be disabled by a bit in the Machine Configuration Register. This logic and this bit are replaced in the TC/FPV by the *local* bit in CMR entries. The CMR can be set up for backwards compatibility.

2.2.5 The Interleave Enable Bit in the CMR

Bit 13 in each CMR entry indicates whether the memory referenced through that entry can be interleaved.

The Interleave Decision RAM also controls interleaving. Physical Address bits 31..26 and 22..15, a total of 14 bits, select one of 16,384 1-bit Interleave Decision RAM entries, each controlling interleaving in one 32-kilobyte *quad-page*.

In the TC/FPV:

```
        T-bus T_INTERLEAVED signal =
        ((Interleave Decision RAM register bit = = 1)
        AND (CMR interleave enable bit = = 1))
```

The intent is that software use the CMR and the Interleave Decision RAM so that references to any given System Physical Address are interleaved for both read and write, or non-interleaved for both read and write. However, the hardware does not enforce this, and it is possible (but confusing) to have reads interleaved and writes not, or vice versa.

NOTE

INTERLEAVE VERSUS LOCAL
Suppose an access has interleaving enabled in both the CMR and the Interleave Decision RAM; the T-bus T_INTERLEAVED signal is asserted, and the intent is that any interleaved access go over the switch. But the CMR *local* bit might also be asserted, trying to force the access to be serviced locally. In this case, the interleaving forces use of the switch.

NOTE

HISTORICAL NOTE
On the B2VME predecessor to the TC/FPV, interleaving was restricted to "bank zero", the low 8 megabytes of each function board. There is no such restriction on the TC/FPV.

2.2.6 The Bypass Bit in the CMR

The "bypass" bit, bit 12 in each CMR entry, tells the CPU interface that the access should bypass the TC2000 locking protocol.

The granularity of bypassing is 8-megabyte blocks. The design permits software control of how much of the virtual address space is given over to bypassing.

NOTE

HISTORICAL NOTE
In the B2VME predecessor to the TC/FPV, bypassing was controlled by Physical Address bit 31, both in a local reference and in a remote reference on a machine with two switch columns (64 or less function boards). On the TC/FPV, Physical Address bit 31 has no special meaning. The B2VME model can be emulated with the CMR.

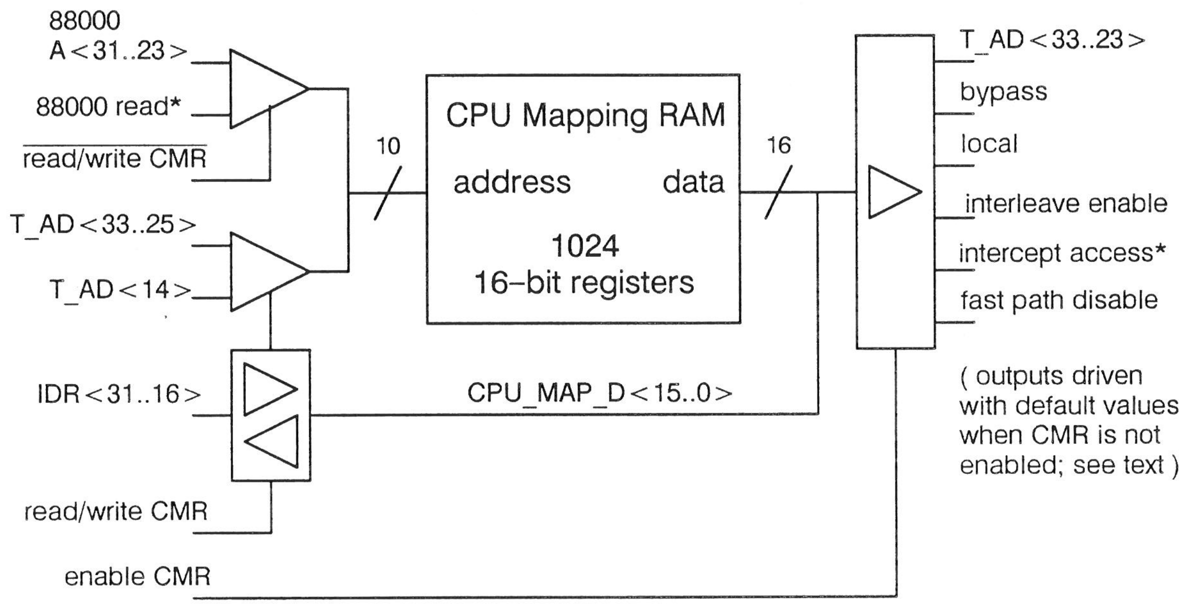
2.2.7 The Fast Path Disable Bit in the CMR

Normally, certain references from the CPU to local memory traverse a special path that is faster than a normal T-bus access. This **fast path** is used for non-interleaved reads to local memory, if enabled by a bit in the Machine Configuration register and not disabled by this bit in the CMR. The fast path speeds up local references, reduces T-bus contention, and is never detrimental, so normally it is enabled, and is disabled only for testing and diagnostics.

2.2.8 CMR Block Diagram

Figure 2–5 shows a block diagram of the CPU Mapping RAM.

Figure 2–5 CPU Mapping RAM block diagram.



Reads and writes of the CPU Mapping RAM are performed using the interleaver loader mechanism.

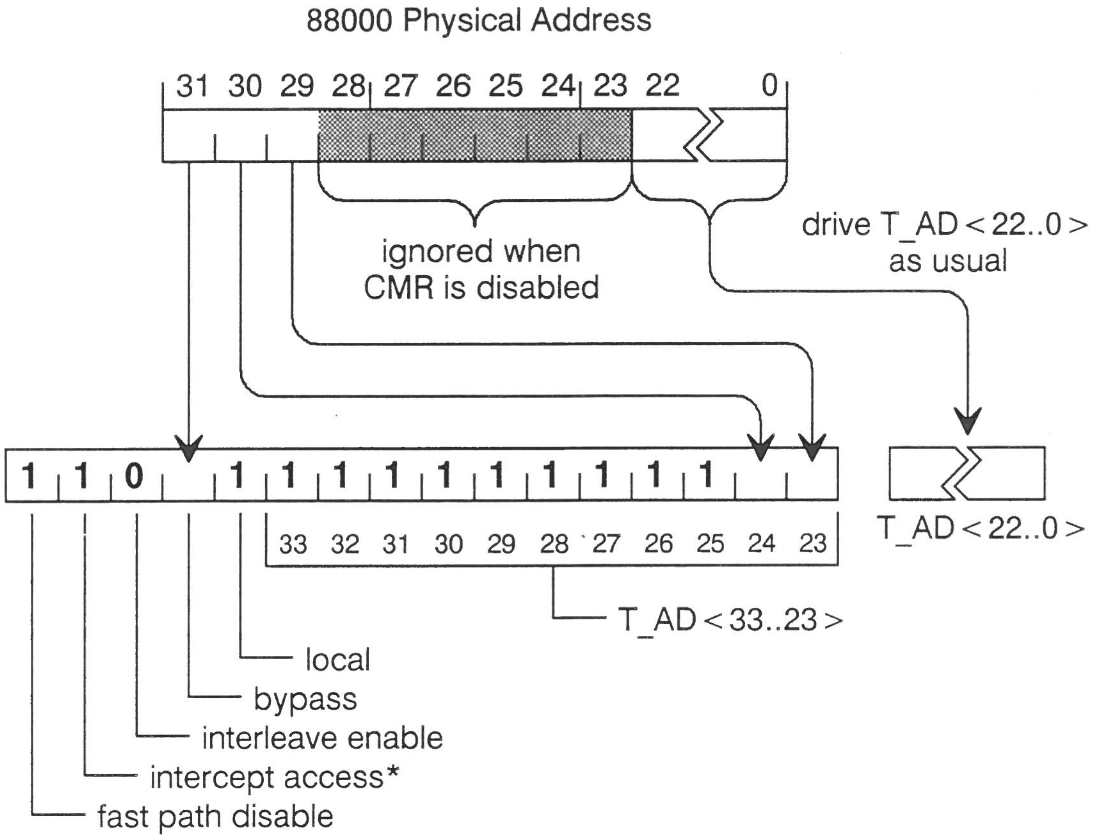
NOTE

Unless the Intercept Access mechanism is being used, the intended use of the CMR is to map ***read and write*** accesses of a given Physical Address into the ***same*** System Physical Address. To use the CMR this way, the software setting up the CMR must load two CMR entries for each block of addresses to be mapped, one for reading and one for writing.

2.2.9 CMR Power-up and Disabled State

Upon power up, the CMR is disabled. When it is disabled, the following transformation is performed on the 88000 Physical Address bits to generate the bits that the CMR drives when it is enabled.

Figure 2–6 **Power–up and Disabled CMR Operation.**



This transformation is similar to the fixed mapping for small (under 64 slots) machines used in the original B2VME design, in that bits 29 and 30 of the 88000 Physical Address are shifted to bits 23 and 24 of the System Physical Address, and bit 31 indicates a bypassed reference. However, the TC/FPV disabled CMR transformation causes all memory accesses from the 88000 to go to local memory. This lets the processor access local bootstrap code that can initialize and then enable the CMR. The CMR is enabled by setting the *CMR enable* bit in the Machine Configuration Register.

2.2.10 CMMU and CPU Interface Affect Address Use

Note that the CMMU, and the CPU interface's control and configuration registers, place certain restrictions on address usage, regardless of how the CMR is set up.

The 88200 CMMU has a fixed, one–to–one mapping for the top one megabyte of supervisor address space, called the *control memory address space*. This area is intended for memory–mapped peripherals and I/O devices. Within this top one megabyte, four kilobytes are diverted to address the CMMU's internal registers. Accesses to the remaining 1020 kilobytes are passed through the CMMU as a Physical Address. Thus, the top one megabyte has a 4–kilobyte "hole" that cannot be used to access locations elsewhere in the TC2000 machine in supervisor mode. When the machine is powered up, a hardwired circuit sets where this hole lies by initializing a register within the CMMU. Because there are two (or three) CMMUs per CPU, there are actually two (or three) 4–kilobyte holes in the top megabyte of supervisor mode Process Logical Address space. For more information on the Physical Address and operation of the CMMU, see the Motorola *MC88200 User's Manual.*

The configuration and control registers of the CPU interface on every TC/FPV occupy the top one megabyte of the 32–megabyte System Physical Address space associated with that function board.

2.3 Memory

The TC/FPV can be configured with either 4 megabytes or 16 megabytes of memory.

The memory on each function board is accessible to all other function boards, thus constituting the global memory of the machine. Memory management is used to map pages of memory conveniently and to control access permissions.

A parity bit with each memory byte protects against errors.

A function board can address its own address space over the switch instead of locally, when various parameters are set up appropriately. (The parameters affecting this are the mapping in the CMMU, the mapping in the CMR, the local bit in the CMR, the interleave enable bit in the CMR, and the bit in the selected Interleave Decision RAM entry.)

Locations accessed over the switch, typically memory, can be **interleaved**. Interleaving is a technique that distributes small chunks of address space that are normally contiguous, to chunks on different function boards. This is useful to avoid certain kinds of contention. The TC2000 interleaver is described in section 2.4.4.

2.4 Switch Interface

The switch interface, implemented by the Switch Interface Gate Array (SIGA) chip, automatically handles references to or from remote function boards.

2.4.1 References to Remote Function Boards

When a request appears on the T–bus from either the CPU or the VMEbus interface that references a remote location, the requester section reads the parameters of the request, packages them up into a small message, and sends the message into the switch. If the message is rejected, the SIGA delays for a short time according to the backoff algorithm (selected by system initialization code) and retries the message, repeatedly. Normally, the message quickly succeeds in traversing the switch to the destination function board, where it is serviced. If it is a read operation, the data read is returned over the same switch path to the requester section of the SIGA. There, the T–bus is acquired and the data is given to the CPU (or VMEbus interface). Ordinarily, the software — both user and system — never takes any different action for a remote reference than for a local reference. Only if an error, such as a timeout, occurs is additional software activity invoked.

2.4.2 References from Remote Function Boards

When the address space on the given function board is referenced by a remote function board, the server section of this board's SIGA serves the request. The received message is checked for correctness, and then interpreted as a read or write request. The server section acquires the local T–bus and makes the request. The local memory, VMEbus interface or registers respond. If the operation is a read, the data is packaged up by the server section and sent through the switch to the requester. Ordinarily, the software on the local function board is never aware of references to it made by other function boards, except for the following effects:

- If the reference writes a location, the new data is visible because memory is shared. CPU interface registers may also be accessed remotely; in particular, a remote processor requests an interprocessor interrupt by writing into a register dedicated to that purpose.
- If the reference locks the local memory module or local VMEbus interface, data references made by the local CPU are delayed until the lock is freed, unless the local reference is made bypassed.
- Remote references use some of the bandwidth of the T–bus and the referenced module. The remaining bandwidth, available to the local processor, is typically high but can be noticeably reduced by very heavy referencing from remote function boards.

The first and second effects arise from the shared memory architecture of the machine, and are important features. Memory management is used to limit the access processes have, so a faulty process does not overwrite or lock arbitrary remote locations.

The third effect, due to contention, is controlled by careful program design and development. Tools are available to detect and identify hot spots of contention, and they usually yield to known programming techniques.

2.4.3 More Switch Interface Features

The SIGA operating parameters are discussed in chapter 3.

Error conditions, including timeouts, arising from use of the switch are discussed in section 2.10.

Besides functioning as a switch interface, the SIGA provides three other functions.

- The SIGA implements a clock and timer interrupt feature, described in section 2.10.
- The SIGA provides a facility called the **interleaver loader**. This facility is used by nX (and in the future, pSOS^{+m}) system software to access registers in the memory interleaver (see section 2.4.4), in the CPU Mapping RAM (see section 2.2), and also in the VMEbus Master Mapper (see section 2.5).
- The SIGA is the avenue by which the function board TCS slave accesses the T-bus, and thereby the rest of function board logic circuits.

The Level Converter (LCON) chip, while a vital part of the switch interface, is invisible to operating system and and application software.

2.4.4 Interleaver

The interleaver applies a mapping to references made over the switch. It takes in several bits of the System Physical Address, performs a programmable translation on them, and presents the result to the switch interface as an alternate switch port number (the high nine bits of the System Physical Address). A separate signal related to the T-bus, T_INTERLEAVED, tells the switch interface whether to use this alternate switch port or the port specified in the address from the T-bus.

The interleaver is described in chapter 4, in conjunction with memory, because its intended use is to modify the way memory is addressed, whether the reference comes from a CPU or from a VMEbus.

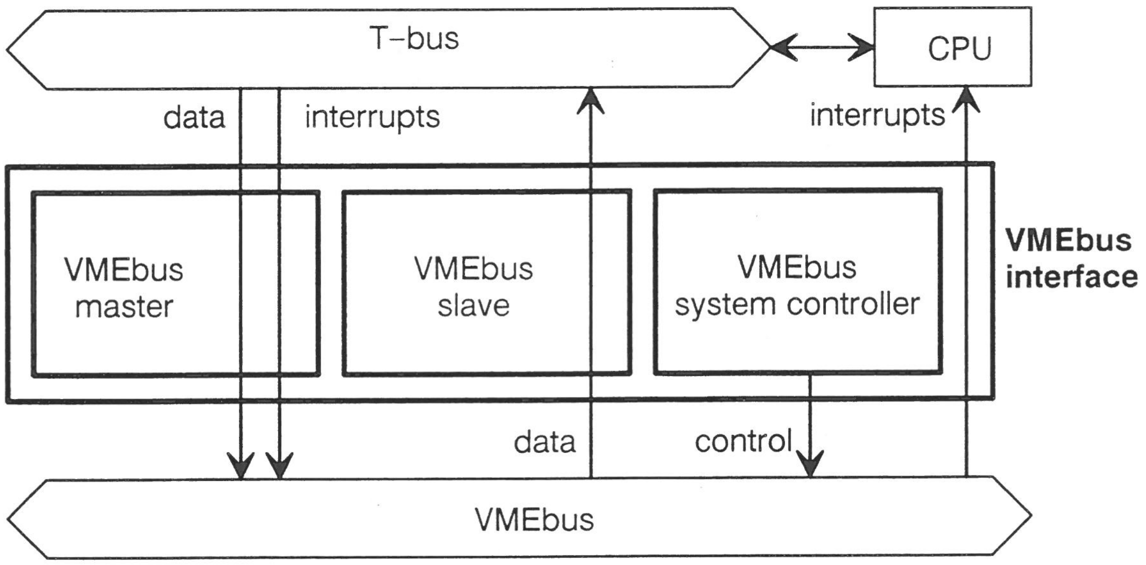
2.5 VMEbus Interface

The VMEbus interface connects the T–bus to a VMEbus, in each direction. Also, the VMEbus interface can be configured to perform certain duties ("system controller") that some device on a VMEbus system must provide.

The VMEbus interface is made of three functional blocks.

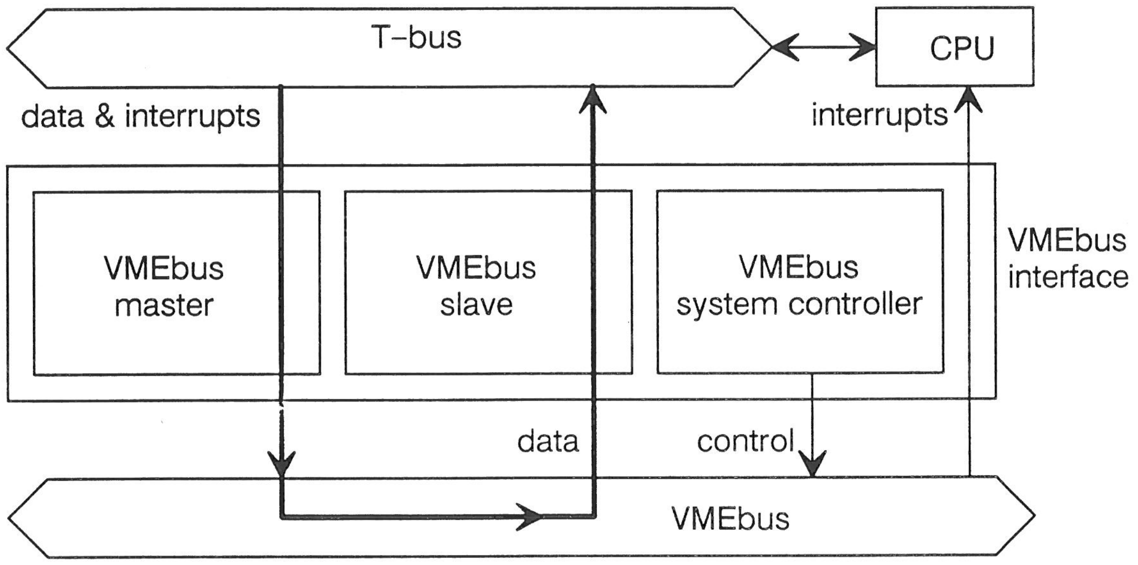
- The **VMEbus master mapper** takes requests on the T–bus that refer to address space mapped to the VMEbus, and translates them into VMEbus transactions as a master device on the VMEbus.
- The **VMEbus slave mapper** takes requests on the VMEbus that refer to address space in the TC2000 machine. Responding as a slave on the VMEbus, this functional block translates such requests into transactions on the T–bus.
- The **VMEbus system controller** permits the VMEbus interface to perform certain VMEbus control and management duties.

Figure 2–7 VMEbus interface components.



When the VMEbus interface acts as a VMEbus *master*, it is simultaneously acting as a T–bus *slave*. And conversely, when its VMEbus *slave* is active, it is acting as a T–bus *master*. The interface can be set up so a transaction is looped — from the T–bus, out onto the VMEbus, back in and onto the T–bus. Such looping would only be used in testing, but it dramatically illustrates the independence and flexibility of the interface's master and slave functions. Looping the other direction — originating on the VMEbus, through the interface onto the T–bus, back through the interface and onto the VMEbus again — is supported by the VMEbus interface, but will get a timeout because the VMEbus itself can't be doing both the original request and the looped reappearance of the request at the same time.

Figure 2–8 **Example of a looped–back reference.**



The sections below describe the three functional blocks in more detail. For a description of the VMEbus itself, please refer to:

> *The VMEbus Specification*, by Motorola (The TC/FPV conforms to revision C.1 of this specification.)

2.5.1 VMEbus Master Mapper

The VMEbus master mapper translates 1–, 2– or 4–byte read or write requests on the T–bus into requests on the VMEbus. This translation occurs whenever the request falls in any of the 2048 8–kilobyte pages in the upper 16 megabytes of the TC/FPV's address space. (However, the topmost megabyte is not usable for translation because the TC/FPV configuration and control registers reside there.)

Put another way, any T–bus local access in the range 0x1000000 to 0x1EFFFFF becomes an access to the VMEbus. The resulting page number (address bits 31 through 13 inclusive, denoted "31..13") placed on the VMEbus is given by one of 2048 mapping registers and may be whatever the application wishes. The offset within the page (address bits 12..0) is copied directly from the T–bus.

Each mapping register in the VMEbus master mapper also supplies the six **address modifier** bits used in the VMEbus transaction. These bits specify the kind of access being made, such as "standard" or "extended" addressing. Also, each mapping register supplies a interrupt acknowledge (**IACK**) bit. This is used when the TC2000 software is acknowledging an interrupt request made by some device on the VMEbus. The interrupt level acknowledged is determined by the address accessed within that register's page. Reads and writes cannot be made through a mapping register while it is set up to acknowledge interrupts, so programs typically set aside one mapping register just to acknowledge interrupts on the VMEbus. For details of both address modifiers and the VMEbus interrupt system, please refer to *The VMEbus Specification*.

When the VMEbus master mapper becomes master of the VMEbus, it can obey either of two mastership protocols: **release on request** or **release when done**, defined in *The VMEbus Specification*. The protocol used depends on the application. The TC/FPV can be configured by a jumper to employ either protocol.

The VMEbus master mapper registers are read and written through a special mechanism called the **interleaver loader**.

2.5.2 VMEbus Slave Mapper

The VMEbus slave mapper translates 1–, 2– or 4–byte read or write requests on the VMEbus into requests on the T-bus. Typically the requests address local memory, but they may address local registers or resources on remote function boards. This translation occurs whenever the request falls in any of several contiguous 8–kilobyte pages of VMEbus address space. The location and size of this **window** are determined by control registers in the VMEbus interface. The window *location* — where the window begins in the VMEbus address space — allows the TC2000 software to place the window conveniently for the design of the VMEbus system, or even to move the window around during execution.

The configurable *size* of the window is related to VMEbus address size. *The VMEbus Specification* defines two addressing sizes: "standard" addressing is 24–bit addresses, and "extended" addressing is 32–bit addresses. The VMEbus slave mapper is software configurable to respond to either standard or extended VMEbus addressing. Which kind of addressing is appropriate depends on the application and the other devices on the VMEbus.

- If the VMEbus slave mapper is set to respond to *standard* VMEbus addressing, the window is **4 megabytes** (512 8–kilobyte pages).
- If the VMEbus slave mapper is set to respond to *extended* VMEbus addressing, the window is **16 megabytes** (2048 8–kilobyte pages).

This flexibility in VMEbus slave mapper window size allows the window to be matched to the application. For example, a graphical display memory or a memory dual-ported to another computer might benefit from a large window. A small VMEbus system performing sensor data gathering, however, might not be able to afford more than 4 megabytes out of its address space.

As with the VMEbus master mapper, any access in the slave mapper's window becomes an access translated onto the T-bus. The new page number (address bits 31..13) is given by one of 2048 mapping registers and may be whatever the application software sets up. The offset within the page (address bits 12..0) is copied directly from the VMEbus.

Each mapping register in the VMEbus slave mapper also supplies several control signals. These specify the following:

- The **path** the request will take (local to the function board, or out over the switch)
- If the request goes over the switch, whether it references **interleaved** memory or non-interleaved memory
- Whether to **lock** the T-bus and the memory module, and the switch if it is used, as long as the VMEbus transaction is in progress (thus permitting atomicity of operations that originate on the VMEbus)
- If the referenced memory module is locked, whether the reference will **bypass** the lock

(The slave mapping register also specifies, if the request goes over the switch, its **priority** value. This capability should ordinarily not be used; priority should be left to the switch latency control mechanism in the SIGA hardware.)

The VMEbus slave mapper registers are accessed directly from the T-bus; they occupy specific locations in the global System Physical Address space. The VMEbus slave mapper registers in any TC/FPV function board can be loaded (and read) from any function board in the TC2000 machine, subject to protection via memory management on the board making the access. This global accessibility is an important part of parallel I/O functionality on the TC2000 machine.

2.5.3 VMEbus Interrupt Handling and Generation

The VMEbus interface can both handle interrupts generated by VMEbus devices and generate interrupts on the VMEbus. As mentioned in section 2.5.1, one (any one) of the VMEbus master mapper's mapping registers is typically set aside for generating the IACK cycle to acknowledge an interrupt.

2.5.4 VMEbus System Controller Functions

When in slot 1 of a VMEbus system, and configured by jumper to be system controller of the VMEbus system (as it normally is), the VMEbus interface will perform the following functions:

- Arbiter, using single level arbitration — grants mastership of the VME-bus to devices requesting it
- IACK daisy chain driver — repeats the interrupt acknowledge signal back along a daisy chain, where a VMEbus device requesting an interrupt will sense it
- System clock driver — provides a 16–megahertz clock
- Bus timer — provides two timeout functions:
    - VMEbus Arbiter Timer — limits how long the VMEbus may be granted without the device taking mastership by asserting the signal *bus busy*
    - VMEbus System Bus Timer — limits how long a VMEbus master may assert the signal *address strobe* without any slave responding with the signal *DTACK* (data acknowledge)

The two timer functions above are performed only as VMEbus system controller, and do not affect the remainder of the VMEbus interface, TC/FPV or TC2000 machine in any way unless it is making the VMEbus access that is timed out. A third timer, the VMEbus TC/FPV Master Bus Timer, is different from the above two timers and is discussed in section 2.10.

For further explanation of the system controller function, please refer to *The VMEbus Specification.*

2.6 TCS Slave

The TCS slave on the TC/FPV connects directly with the SIGA, through which it has access to the T–bus and thence to all memory and registers on the board.

This TCS slave performs the following functions:

- Monitor temperature
- Monitor +5 and -5 volt power, from on–board supplies
- Control the on–board power supplies (on, off and margining)
- Reset the board (resets the 88100 CPU and other hardware)
- Report board type to TCS master upon request
- Access the T–bus, to perform the following actions:

- Initialize registers as necessary before operation
- Load bootstrap code into memory
- After the nX operating system is running, monitor certain locations for data sent by the nX software to the TCS master, and return any results from the TCS master to the nX software

For more information on the TC/FPV TCS slave capabilities, please refer to the current documentation on TEX, the TCS master software, in the *System Administration Guide.*

2.7 Configuration and Control Registers

The CPU interface contains several registers that configure and control the TC/FPV. These are described briefly below, organized into nine functional groups.

2.7.1 User Registers

Registers in this group are intended to be accessible to the user's application program. The address of each is on its own page, so the operating system can easily permit or prohibit access to each independently, for any process, by modifying the process's memory map. Also, some fields in one of these registers, the PCR, are protected by **mask** bits in another register. When a mask bit is "1", attempting to set the masked field in the PCR results in a bus error. This provides even finer control over access to PCR functions.

NOTE

NO nX USER ACCESS
In the current release (2.0) of the nX operating system, none of these registers are accessible to the user software.

Interprocessor Interrupt Register

Setting the one defined bit in this register causes an interrupt request to the local CPU. Since this register may be accessed over the switch, it provides a way for one processor to interrupt another.

Process Configuration Register

This register has four fields. The **synchronized access** bit selects a particular switch access strategy in the SIGA, that can be set up to reduce congestion when many CPUs are all referencing the same function board. The **path** field contributes to the decision of whether a reference is local (to this board) or remote (goes out over the switch). The **default priority** and **priority scheme** fields contribute to switch message priority, and ordinarily should not be used; priority should be left to the switch latency control mechanism in the SIGA hardware.

Augmentation Register Block

There is one register, called the **Augmentation** register (AR), and a block of locations the user employs to modify its contents. In general, the Augmentation register extends (augments) the instruction and addressing capabilities of the 88000 CPU. Augmentation applies only to certain kinds of references. The **lock** bit controls whether references made by the CPU will employ the TC2000 locking protocol (described in section 2.9.7. The **exception action** field can be used by the operating system to remember what action it should take after an exception during an instruction sequence that uses the lock bit (continue, restart or abort). The **disable interrupts** bit inhibits the presentation to the CPU of interrupt requests from certain sources (illustrated in Figure 2–11).

2.7.2 Configuration Registers

These registers, in addition to the user–accessible registers in section 2.7.1, directly control the process execution environment. These, however, are intended for access by privileged processes only. Privileged processes, in referencing the Augmentation register, can access it directly as well as through the Augmentation Register Block locations noted above.

PCR Disable Mask Register

Bits in this register mask (***prohibit*** setting of) the *path*, *default priority* and *priority scheme* fields in the Process Configuration register.

Machine Configuration Register

This register controls certain basic operating characteristics of the TC/FPV. The **cache selection scheme** bit determines how the two code CMMU chips are shared, in a TC/FPV configured with two. The code CMMU to use is selected either by a bit in the Process Logical Address, or by the supervisor/user mode bit. The **fast path enable** bit allows certain references from the CPU to local memory to traverse a special path that is faster than a normal T–bus access. The **write wrong parity** bit causes incorrect parity to be written, and is used for diagnostic purposes. The **CMR enable** bit enables the CPU Mapping RAM in the CPU interface.

2.7.3 Interrupt System Registers

The TC/FPV's CPU interrupt facility extends the single–level interrupt capability of the 88100. These registers control that facility. Interrupts to and from the VMEbus are described in section 2.7.6. The interrupt facility is described in section 2.10.

Interprocessor Interrupt Register

This register requests a CPU interrupt, as described in section 2.7.1.

Non–maskable Interprocessor Interrupt Register

Setting the one defined bit in this register causes an interrupt request to the local CPU. This register is similar in function to the Interprocessor Interrupt register, except that this register's request cannot be suppressed by the Interrupt Enable Mask register.

Interrupt Source Register

This register informs the CPU which of several possible events is requesting interrupt service. The sources are: the VMEbus (seven levels), the non–maskable interprocessor interrupt, the maskable interprocessor interrupt, the real–time clock interrupt(s), and the "interrupts disabled too long" timeout.

Interrupt Enable Mask Register

This register allows the CPU to selectively enable interrupts from certain sources — namely, from the VMEbus (each level independently), and from the Interprocessor Interrupt register.

2.7.4 (T–bus) Bus Error Register

Bus Error Vector Register

This register, the only one in this functional group, indicates the reason that a bus error was generated. The error is reported on the T–bus, but may have originated elsewhere, such as in handling a switch connection.

2.7.5 Latency Control Registers

The latency of interrupt servicing and of access to a memory module may be controlled with these registers. Latency is also controlled by timer registers in the VMEbus interface for VMEbus transactions, and in the SIGA for switch transactions.

Interrupts Disabled Timer Register

This register limits how long the CPU may disable certain interrupts — namely, those from the VMEbus, real time clock timer(s), and the maskable interprocessor interrupt.

Interrupts Pending/Abort Retries Timer Register

This register limits how long the CPU will wait for the establishment of a switch connection while an interrupt is pending.

CPU Lock Timer Register

This register limits how long the CPU may hold a memory module locked.

2.7.6 VMEbus Interface Registers

The TC/FPV VMEbus interface contains three general types of registers.

VMEbus Configuration Register
VMEbus Master Map RAM Registers
VMEbus Slave Map RAM Registers

These registers (the RAM locations are loosely called registers) control the role played by the TC/FPV as a device on the VMEbus — the location, size and mapping of the window from VMEbus address space into TC2000 address space, the mapping of the window in the reverse direction, whether the TC/FPV is system controller, and related parameters. Figure 2–7 shows these functions.

VMEbus Interrupt Request Register
VMEbus Interrupt Vector/Control Register

These registers control the generation of interrupts on the VMEbus by the TC2000 software.

VMEbus Arbiter Timer Register
VMEbus System Bus Timer Register
VMEbus TC/FPV Master Bus Timer Register

These registers control timers that detect and abort conditions that persist too long. The first two are described in section 2.5.4, and the third in section 2.10.

2.7.7 SIGA Registers

The SIGA contains several registers accessible from the T–bus. These are described in four functional groups below. The SIGA also contains registers accessible only to the TCS slave, and registers not directly accessible from outside the chip; these are not discussed here.

Message Classification Register
Protocol Timer Configuration Register
Transmit Time Configuration Register
Priority Time Configuration Register
Requester Configuration A and B Registers
Requester Test A Register

These registers control the transmission of switch messages by the requester portion of the SIGA.

Server Configuration A and B Registers
Server Test A Register

These registers control the reception of switch messages by the server portion of the SIGA.

Real Time Clock (RTC) Registers

Time Of Next Interrupt (TONI) A and B Registers
TONI A and B Configuration Registers

These registers concern timekeeping. The RTC provides a constantly incrementing, nearly real time counter. Based on the RTC, two TONI registers provide programmable timer interrupts. These are described further in section 2.10.

Interleave Address Register (IAR)
Interleave Data Register (IDR)

These registers implement an interface, called the **interleaver loader**, between the T–bus and various high speed RAMs: the Interleaver RAM (in the switch interface), the Interleave Decision RAM and the CPU Mapping RAM (both in the CPU interface), and the VMEbus Master Map RAM (in the VMEbus interface). Accessing the RAM locations works as follows. A read access to certain locations (near that of the IAR) causes the contents of the IDR to be stored into, or to be loaded from, a location (selected by the IAR) in one of the special RAMs.

2.7.8 Interleaver Control Registers

The TC/FPV has two kinds of registers controlling interleaving — registers that determine whether a given access is to interleaved memory, and registers that produce modified bits of the interleaved address.

Interleave Decision RAM Registers
CPU Mapping RAM Registers

These registers determine whether a reference generated by the CPU is to interleaved or non–interleaved memory. For interleaving to occur, an enable bit must also be set in a SIGA configuration register.

VMEbus Slave Map RAM Registers

These registers determine whether a reference generated by the VMEbus interface slave mapper is to interleaved or non–interleaved memory.

Interleaver RAM Registers

These registers produce the modified address bits used in an interleaved access. The translation takes bits from the T–bus and supplies modified bits to the SIGA, for inclusion in the outgoing switch message.

2.7.9 CPU and CMMU Registers

The Motorola 88100 CPU and 88200 CMMU contain several internal registers for control of data processing and of caching and memory management, respectively. For example, the CPU Processor Status register permits disabling interrupts; disabling interrupts this way makes the CPU immune to all the TC/FPV interrupts, including the "non–maskable" interprocessor interrupt and the "interrupts disabled too long" timer. A more complete discussion of CPU and CMMU registers is beyond the scope of this document. For such a description, please refer to the *88100 User's Manual* and the *88200 User's Manual.*

2.8 Path and Speed of References

Since the TC2000 architecture contains a variety of mechanisms controlling and accelerating the access to addressable resources, the discussion of what path an access takes and how much time it takes are somewhat subtle.

2.8.1 What Path an Access Takes

This section describes the path taken by a memory access from the TC/FPV CPU. This description is both to clarify operation of the hardware, and to set context for memory access timing in section 2.8.2. More detailed information about the 88000 chip set can be found in the *MC88100 User's Manual* and the *MC88200 User's Manual.*

The possible outcomes of an access are:
— a cache hit
— a fast path access to local memory
— a T–bus access to local memory
— a switch access to remote memory (including back to this function board)
— a bus error
— a transaction fault detected during address translation in the CMMU.

The CPU's normal processing can be disrupted by any of the following exceptions:

- Internally, the CPU can detect and assert various exception conditions (see the *88100 User's Manual*).
- Externally:
  - Assertion of the reset pin resets the CPU.
  - The CPU may be interrupted by assertion of its interrupt (INT) pin.
  - A data transaction (read or write) may encounter an error, signalled to the CPU by a code on its Data Reply pins (11 = transaction fault).
  - A code transaction (instruction fetch) may encounter an error, signalled to the CPU by a code on its Code Reply pins (11 = transaction fault).

The four external causes have different exception vectors assigned to them. Reset is not of concern here, where we assume processing is in progress. Interrupts are signalled only by the interrupt logic on the TC/FPV described in section 2.10. The data and code transaction faults are signalled only by the CMMUs.

START:

- The 88100 generates a reference. It supplies Process Logical Address bits, supervisor/user mode bit, read/write bit, and instruction/data bit.
- One of the two (or three) CMMUs will respond, based on the instruction/data bit and, if the reference is an instruction fetch and two code CMMUs are present, the code cache selection logic, as follows:
  - If the Machine Configuration register *cache selection scheme* bit is zero, then Process Logical Address bit 12 selects the code cache.
  - If the Machine Configuration register *cache selection scheme* bit is one, then the CPU's supervisor/user mode bit selects the code cache.

  From here on, the CMMU that responds is called "the" CMMU.
- The CMMU attempts to locate a mapping for the Process Logical Address. The possible outcomes of this process are:
  - The CMMU may find a valid entry in its ATC with protection attributes that match those of the process making the reference.
  - The CMMU may have to search its translation tables to find the mapping. This involves memory references initiated by the CMMU. If a memory error occurs on one of these references, the CMMU returns a transaction fault to the CPU. Otherwise, it loads the appropriate translation information into its Address Translation Cache (ATC) and translates the address.
  - The CMMU may find that the transaction violates the protection specified in the selected translation register, either write protection or supervisor mode protection. A transaction fault is returned to the CPU.
  - The CMMU may find that a segment descriptor or page descriptor that it needs is invalid. A transaction fault is returned to the CPU.
- The CMMU attempts to make the requested access, using the Physical Address generated in the translation step. The possible outcomes are:
  - The page may be marked "cache inhibit", that is, non-cacheable. The access is passed on through the CMMU to the memory bus. If a bus error occurs on the memory bus, the CMMU passes the error back to the CPU as a transaction fault.
  - The CMMU may have the data cached (a cache hit). For a read, the data is returned quickly to the CPU. For a write, the cached

data is overwritten with the new data. If the page being referenced is in copyback mode and the cache line (four 32-bit words) had previously been modified, the transaction is done. If the page is in writethrough mode, or if it is in copyback mode and this is the first write since the line was loaded into the cache, the CMMU writes the cache line back to memory. If a bus error occurs on this write, the CMMU returns a transaction fault to the CPU.

- ○ The CMMU may not have the data cached (a cache miss). The CMMU will select a cache line to replace with the desired line. If the selected line is modified and is in copyback mode, the CMMU must write it back to memory before replacing it. If a bus error occurs on this write, the CMMU returns a transaction fault to the CPU. If the selected line is not modified, or is in writethrough mode, it is simply discarded. The CMMU then reads the new cache line from memory. If a bus error occurs on this read, the CMMU returns a transaction fault to the CPU. Having filled the cache line, the CMMU performs the requested access, as described above for a cache hit.

Whenever the CMMU initiates a memory reference, the CPU interface on the TC/FPV translates the CMMU address, data and control signals, which obey the M-bus protocol defined by Motorola, into signals that obey the TC2000 T-bus protocol. In addition to accounting for various timing differences, the CPU interface logic performs several operations:

- Decide whether the reference can be handled by the fast path, and issue the necessary control signals if it can.

  A reference from the CPU will use the fast path only if all of the following conditions are met:

  - ○ The operation is a read (either code or data), not a write.
  - ○ The Machine Configuration register "fast path enable" bit is 1.
  - ○ The "fast path disable" bit of the CPU Mapping RAM entry selected by the current reference is 0.
  - ○ The "local" bit of the CPU Mapping RAM entry selected by the current reference is 1.

- Decide whether the reference should be intercepted. If the *intercept access** bit (the * is part of the name of the bit, indicating it is low true) in the CMR entry selected by this reference is zero, the access is intercepted. That is, the CPU is acknowledged as if the reference has completed, but no T-bus cycle occurs. If the reference is a read, the data returned is undefined.

  The intercept access mechanism is used for speeding up certain block transfer operations involving the cache. It is described further in section 2.2.3.

- Generate the T–bus control signals needed to specify the path to be taken by the address and data for this reference (if not the fast path). These are:
  - T_INTERLEAVED — Generated by the Interleave Decision RAM, subject to the CMR *interleave enable* bit. If this is a remote reference, this signal indicates that the transformed address generated by the interleaver should be used by the SIGA. The CMR *local* bit is ignored if the T_INTERLEAVED bit is asserted.
  - T_PATH < 1..0 > — Generated by the Process Configuration register, subject to the CPU Mapping RAM *local* bit and interleaving. Indicates whether this reference should be handled by switch interface A, switch interface B, or local memory.
- Generate T–bus control signals that specify other attributes of the reference:
  - T_PRIORITY < 1..0 > — Generated by the priority scheme mechanism. Indicates the priority of this reference in the switch. (Normally, priority should be left to the automatic hardware mechanism used to bound switch latency.)
  - BYPASS — Derived from the CPU Mapping RAM *bypass* bit. When this signal is asserted, the TC2000 locking protocol is inhibited and T_LOCKOP is set to "bypass".
  - T_LOCKOP < 1..0 > — Derived from the state of the Augmentation register *lock* bit, the $\overline{\text{DLOCK}}$ signal asserted by the 88200 CMMU during XMEM operations, and the BYPASS signal. Opens, maintains and frees locked transactions, or bypasses locks, according to the TC2000 locking protocol.
  - T_SYNC — Generated by the *synchronized access* bit in the Process Configuration register. If this is a remote reference, this signal influences the time at which the request message is allowed to enter the switch.
  - T_AD < 33..23 > — Generated by the CPU Mapping RAM. These bits select an 8–megabyte "bank" of memory within the System Physical Address space global to the machine. As part of that selection, they specify the switch port that is addressed. The switch port, however, is subject to possible further modification by the interleaver. If the reference falls in a window to VMEbus memory, the address is subject to further modification by the VMEbus master mapper at the addressed switch port.
- If the request is not serviced by the fast path or by the intercept access mechanism, the CPU interface issues a T–bus request, where it may be serviced by any of several T–bus slaves, depending on the value of T_PATH < 1..0 >.

- ○ T_PATH = 11: The local memory, the VMEbus master, or the configuration and control registers will respond.
- ○ T_PATH = 10: The switch interface serving the "A" switch will issue a request message, retransmitting until the request reaches its destination or is timed out.
- ○ T_PATH = 01: The switch interface serving the "B" switch will issue a request message, retransmitting until the request reaches its destination or is timed out.
- ○ T_PATH = 00: Illegal value.

- If the reference is non–local, the SIGA at the remote end makes a request on its local T–bus. The parameters of this request are as follows:
  - ○ The T_PATH <1..0> bits are always 11.
  - ○ The T_AD <33..25> bits are driven to zero. For a local access (T_PATH = 11), the *T–bus Specification* requires that T_AD <33..25> be driven to valid binary levels, but their value is undefined.
  - ○ The remaining address and control bits are specified by the switch request message.

  The result of the T_PATH bits being 11 is that the the request cannot go back out either SIGA on the remote function board, but must be served (if at all) by a device local to that board. In the TC/FPV, this can be memory, local configuration and control registers, or the VMEbus master interface. In fact, the VMEbus interface can be set up so the request goes out onto the VMEbus and comes back in the VMEbus slave interface, so the request could proceed further. However, use of this facility (for other than testing) is highly contrived and risks having timers (that help control latency) go off, aborting the connection and returning a bus error.

- When the switch reply message comes back, the SIGA acquires the T–bus and places the requested data on it. The CPU interface takes the data from the T–bus and hands it to the CMMU, which retains a copy in its cache if it is a cacheable reference. The CMMU hands the data on to the CPU, and execution resumes.

2.8.2 CPU Memory Access Timing

Figure 2–9 shows the memory access time from a TC/FPV CPU under a variety of conditions. The accompanying notes are essential. Each value shown applies throughout its connected white space. These are generally best case (minimum) times, as detailed further in the notes.

Figure 2–9 **Memory access time (microseconds).**

Cache		Access Made by CPU			
Mode	**Activity**	**Read from Local**	**Write to Local**	**Read from Remote**	**Write to Remote**
inhibited	none	0.550	0.600	1.913	1.889
writethrough	hit	0.150	0.600	0.150	1.889
copyback	hit	0.150	0.150	0.150	0.150
writethrough	miss	0.850	1.200	2.529	4.168
copyback	miss with no writeback	0.850	1.200	2.529	4.168
copyback	miss with writeback to local	1.500	1.850	3.179	4.818
copyback	miss with writeback to remote	2.905	3.255	4.534	6.173

Notes for Figure 2–9.

1. The operating frequency of the TC/FPV characterized here is 20.0 MHz.
2. The timing shown is the full latency including the CPU's "execute" phase (a Motorola term for instruction decoding) and "address" phase through completion of the access. For example, a read with a cache hit takes three cycles (execute, address, and successful reply), a total of 0.150 microseconds. The *bandwidth* may be greater than the reciprocal of the timing shown, since the execute and/or address phases may be pipelined with the processing of other instructions, depending on the mix of instructions. The maximum bandwidth is one access per cycle. Pipeline stalls caused by recent instructions that have not yet finished are possible during the execute and/or address phases; the timing shown assumes such stalls are absent. For further details, see the Motorola *MC88100 User's Manual.*
3. The fast path is assumed used where possible, namely in reads (either a burst or a single word) from non-interleaved local memory. Using the fast path reduces the access time by 3 cycles (0.150 microseconds).

4. The timing shown assumes that no page table walking is performed. If the CMMU must load memory mapping information to service the access, the time seen by the CPU is increased.

5. The intercept access mechanism is assumed not used. Its effect on timing is discussed later.

6. If the access is augmented with the TC2000 locking protocol, the access time in some cases may be reduced because, if a remote function board is referenced, locking holds the switch path to it open. Therefore, if switch transmissions were not immediate or if there was contention in the switch or at the destination port, only the initial locked reference would be delayed and not the subsequent references during the sequence. However, the timing shown assumes immediate transmission and no contention, so there is no effect due to locking on the timing shown. (The timing of VMEbus accesses, not discussed here, is affected by locking.)

7. The chart is intended to show data access timing, although the entries relevant to an instruction fetch are also valid for that. (In an instruction fetch, the times shown include the 1-cycle "prefetch" phase rather than the 1-cycle "execute" phase.) The M-bus ("memory" bus, the common output of the CMMUs) is assumed not occupied by another CMMU. (That is, occupied by one of the two instruction CMMUs when a data access is made, or by the data CMMU when an instruction fetch is made.) If the M-bus is occupied, the access time shown is increased by the time needed for that access to release the M-bus. (Note that there are separate instruction and data P-buses, so instruction and data accesses do not contend for the P-bus.)

8. The T-bus is assumed not occupied. If the T-bus is occupied, the access time shown is increased by the time needed for that access (and any other accesses with higher T-bus priority than the CPU) to complete. In the TC/FPV, the CPU has the lowest T-bus priority. Note that accesses to the VMEbus master interface and to the switch interface split cycles on the T-bus, releasing the T-bus for other use while the requested operation is performed.

9. Any memory module used in servicing the access is assumed not locked and idle. If it is locked (via the TC2000 locking protocol), the access time shown is increased by the time needed for the locked transaction to complete and the access to be retried. The hardware performs the retry automatically. (If it is a bypassed access, the timing is unaffected by the locked status of the memory module.) If the memory module is busy — with a refresh cycle or completing a write cycle — the access time shown is increased by the time needed for a T-bus REFUSED reply and a new T-bus arbitration. This is typically 2 cycles (0.100 microseconds).

10. When the access includes one or more references to a remote function board, the timing shown assumes there is no contention for the T-bus and memory on the remote function board.

11. The timing shown assumes the access is to non–interleaved memory. If any reference involved in serving the access is to interleaved memory, that reference is forced to go over the switch. If the reference would have gone over the switch anyway, the access time is not changed. If the reference would have been serviced entirely on–board, the access time is increased.

12. The Butterfly switch included in this characterization is a 2–column switch and operates at a clock frequency of 38.0 MHz.

13. The TC/FPV contribution to access timing was calculated by accounting for individual cycles of the board clock. The Butterfly switch contribution was calculated by a program that models the switch parameters set to their fastest settings. In particular, immediate transmission strategy is assumed. The timing shown would be increased, for example, by a strategy that delays before the first transmission. Such a strategy may be used to pace accesses made to a software spin lock.

14. The Butterfly switch timing assumes no switch contention. That is, there is no contention for the local SIGA, for switch ports within the switch, or for the switch port at the destination function board. If contention is present, the access time shown is increased.

15. The times for specific switch transactions included in the access timing chart are as follows:

transaction	**microseconds**
1–word read	1.337
1–word write	1.363
4–word read	1.953
4–word write	1.729
synchronizer uncertainty	0.152 (see note)

16. Synchronizer uncertainty:
The TC/FPV is clocked at a different frequency than the Butterfly switch. Therefore, each time data enters or leaves the switch, it passes through a synchronizer, a circuit that re–clocks it to the new environment. The delay at a synchronizer varies from no delay up to one cycle at the new clock frequency. Each normal switch access passes through four synchronizers — the request goes into and out of the switch, and the reply does likewise. The "synchronizer uncertainty" shown above is the maximum total delay for all four synchronizations. The computation of access timing includes one half the maximum delay, assuming that the delay is uniformly distributed and therefore is, on the average, half the maximum. Each switch reference incurs this synchronizer delay. For example, if the CPU's access requires three switch references, the access time shown includes three times half the synchronizer uncertainty.

2.8.3 Intercept Access and Timing

The intended use of the intercept access mechanism is to speed up the copying of blocks of data. In this application, one typical case of its use is as follows:

- CPU access is a write to remote memory
- Cache mode is copyback
- Cache activity is a miss with writeback to remote memory
- CMR is set up to intercept the read (The data read in will be immediately overwritten, so it can safely be intercepted.)

In this case, the intercept access mechanism reduces the access time seen by the CPU from 6.173 microseconds to 4.294 microseconds, a savings of about 30 percent.

2.9 Atomicity and Locking

This section begins with general discussions of race conditions, and atomicity and locking. Readers familiar with these topics may skip this general discussion and proceed directly to section 2.9.3 without loss of continuity.

2.9.1 Race Conditions

The facilities described in the next section — atomicity, locking, exclusion and synchronization — are necessary only because multiprocessing is inherently susceptible to **race conditions**, also called multiprocessing hazards. A race condition is competition between two or more processes, over access to resources that each must use, but which use can corrupt each other's results.

For example, suppose processes A and B each need to increment a counter N that is accessible to them both. The steps each might take are:

1. Read the value of N from its location in memory into a register.
2. Add the increment to the value in the register.
3. Write the register's new value back into memory.

But suppose that the processes are executing on separate processors, or that an operating system's preemptive scheduler switches between the two processes in the following unfortunate way:

Process A reads N. Suppose the value it reads is 5.

Process B reads N, getting 5 just as A did.

Process B adds its increment, say 1, getting the new sum 6.

Process B writes 6 back into memory.

Process A adds its increment (1), getting its new sum 6.

Process A writes its 6 back into memory.

We now have 5 + 1 + 1 = (erroneously) 6.

Many machines support the operation of incrementing a value in memory (as in the above example) with a single instruction. Such an instruction is called a "read-modify-write" operation. Such an instruction usually cannot be interrupted (as B interrupted A above), or if it is aborted, it is later restarted at the read. However, on a multiprocessor machine, this is insufficient defense against the race condition. Also needed is a way to prevent B from accessing memory between the read and write phases of A's instruction. Such prevention is called **locking**. Locking the memory location guards against the race condition and guarantees a correct answer in this example.

Race conditions are typically a concern in multiprocessing, although they can also arise in a single processor, single process system due to interactions with I/O devices.

A characteristic of race conditions is that they may sometimes produce the correct answer, and sometimes an incorrect answer, depending on vagaries such as when an interrupt occurs, or the value of numerical data (which can affect how many instructions are needed to compute functions of the data). Therefore, race conditions are sometimes difficult to debug. It's important to program with care to avoid race conditions, by using the facilities described in the next section.

2.9.2 Two Sides of a Coin

Atomicity and **locking** are like the sides of a coin, because although they are different, you can't have one without the other.

We use the terms "atomicity" and "locking" to mean the following:

atomicity	The indivisibility of an operation. An operation is atomic if no other operation that might interfere with the validity or accuracy of its results can occur while the atomic operation is in progress. This notion derives from the Roman philosopher Lucretius, who described physical matter as made of atoms that cannot be divided, and a Greek word for "uncut".
locking	The act of acquiring rights to access a resource, such that no other operation can access the resource in a way that might interfere with the validity or accuracy of the locker's access. The access may be to read, write, or modify in more complex ways. The resource is usually an area of memory, but could also be an I/O device or other data processing circuits.

The notions of atomicity and locking are similar. Atomicity emphasizes that the operation is not interrupted. Locking emphasizes the way that possible interference is prevented. You can't have one without the other, in at least some form.

For example, consider what might happen when you make an airplane reservation. The travel agent keys in the parameters of the desired operation: flight number, date, your name, and so on. The reservations software obtains a lock on the data base entries it will access in performing the atomic operation of making your reservation. To obtain this lock, it may call a locking routine. The locking routine appears atomic to the calling software, but inside it makes repeated attempts to obtain the lock. Each attempt is an atomic reference to a memory location. The reference is atomic because the computer's hardware permits only one process to access the location at a time. This atomicity of reference to a location is enforced by circuitry that locks the data path to the location; only one requester circuit at a time is permitted to drive the request signal wires. The hardware implements the locking of access to the signal wire by the atomic decision to grant only one of possibly several access requests. The atomicity of choosing one request among several works by the physics of the electronic device — if a particular voltage is present, it locks out the effect of competing forces that would produce a different voltage. We see that atomicity and locking support each other. An atomic operation is atomic because it is protected by some form of lock, and it in turn may serve as a lock to secure the atomicity of a larger operation.

A **spin lock** is a way to program the obtaining of a lock: try to get the lock, and upon failure, try again right away, repeatedly until the lock is obtained. A spin lock has two advantages and a disadvantage.

+ Simplicity — It's easy to program, and takes little space.

+ Speed, in simple cases — In the absence of complicating factors noted below, a spin lock achieves the minimum delay between the lock becoming free and the requesting process obtaining it. This can be essential in some time critical applications.

— Congestion (a disadvantage) — If several processes are contending for the lock, many of the requests are either delayed, or aborted and must be retried. This can tie up machine resources such as the access path to the lock location, possibly slowing other processes that aren't even concerned with the lock. Further, the process trying to free the lock can experience this congestion, delaying the freeing.

The cure for congestion due to a spin lock is usually to include a small delay in the program loop. This is like the interruption of highway traffic at an intersection, by a stop light. Or like the pulsing of blood through capillaries. Neither traffic, nor blood, nor spin locks work very well without pulsing when the volume is high.

The delay may be constant, or may increase each time the lock request fails. The increasing delay is called ***backoff*** or pacing or throttling. To avoid repeated bursts of lock attempts from several processes that are backing off with the same schedule, the actual delay may be randomized. In fact, such a randomized, backoff strategy is one of the mechanisms used in the TC2000 switch hardware to reduce congestion, regardless of whether locking is involved or not.

A lock may prohibit all access, or only certain types of access. A common example of the latter is a lock that permits only one process to write new data into a location, but any number of processes to read data from the location. Such a **single writer, multiple reader** lock is useful when one process is responsible for computing the new value of the variable, and other processes can operate successfully with either the old or the new value. This works well if the writer can update the data all at once (atomically), so that no reader will see a mixture of partly old and partly new data.

Two other terms that often appear in conjunction with atomicity and locking are **exclusion** and **synchronization**. By these we mean:

exclusion Preventing another operation, typically performed by another process executing on the same computer, from interfering with an operation that the process at hand is going to perform. Exclusion is like locking, but exclusion emphasizes keeping the other processes away, while locking emphasizes obtaining one resource (the lock) in order to gain rights to further resources.

synchronization
Ensuring that separate processes interact with each other in a controlled manner, so that the overall computation is correct.

Mutual exclusion (or "mutex") is how any one of several competing processes may exclude the others, thus obtaining exclusive rights to a resource. The term, **mutual exclusion lock** is perhaps redundant but nevertheless commonly used.

As an example of **synchronization**, suppose the simulation of chemicals in a mixing vat is divided up among several processes, each simulating fluid in a different area of the vat. The processes interact, because the fluid moving out of one area moves into another area. The processes need to keep in step, or else one will start working on garbage data in memory locations that haven't yet been filled by the result from the process supplying it. Locking can be used to implement some forms of synchronization. In this example, the supplier process can lock the memory locations that will hold the result, unlocking them only after the result is placed there.

In the above example, processes synchronized in pairs. Each piece of data is supplied by one specific process, and is used by another specific process. The transfer of the data is controlled by synchronization between that pair. Another form of synchronization, **barrier synchronization**, is used when several processes must all reach a certain stage in their computations before any can continue. This is like a roadblock in a race; all participants must check in at the roadblock before the barrier is lifted and all are allowed to proceed. Examples of barrier synchronization often arise when each process's computation depends — or may depend — on results of the others. For instance, a simulation of particle motion in a hot, ionized gas may be programmed as discrete steps in simulated time. At each time step, a particle's motion is affected by the forces exerted on it by nearby particles. Perhaps only a fraction of all the particles are near, but which those are varies from step to step as they all move. Barrier synchronization is a convenient way to ensure that these forces are felt in a controlled way. Otherwise, the simulation of some particles' motion might get ahead of other particles' motion, and the proper interactions would not be computed.

The term **semaphore** is often used for a mechanism, such as a lock, employed to synchronize processes.

The following section describes atomicity and locking on the TC2000 computer. The emphasis is on the primitives available to the programmer. Lower level details are provided when they clarify or motivate the basic programming facilities. Higher level constructs are mainly the responsibility of the programmer, who uses the basic functions to build execution control mechanisms appropriate to the application.

2.9.3 The xmem Instruction

The MC88100 CPU has one read–modify–write instruction, **xmem**. xmem exchanges the contents of a CPU register with the contents of a memory location, so calling it a "load–then–store" operation is more accurate than the more general term "read–modify–write". The load and store memory accesses are indivisible; only an access error on the store access can prevent the two from occurring as a unit. The MC88100 supports this by asserting a special signal ($\overline{\text{DLOCK}}$), and the MC88200 CMMU honors this signal by holding on to the M bus. The Motorola design expects the M bus to lead directly to memory, thus ensuring that no interruption is possible and thereby guaranteeing that xmem is atomic.

In the TC2000 machine, however, the M bus does not lead directly to memory. It leads to the CPU interface, which in turn connects to the T–bus, and thence directly to local memory and indirectly over the switch to remote memory. The TC2000 architecture is designed to honor the $\overline{\text{DLOCK}}$ signal throughout these additional components. In particular, an xmem referencing remote memory holds the switch path open, and holds the memory module on the remote function board locked, until the store has completed. This explicit support of xmem in the TC2000 hardware ensures that ***xmem is atomic on the TC2000 computer***.

NOTE

Motorola's *MC88100 RISC Microprocessor User's Manual* contains, in section 8.4, a discussion of xmem as the basis for synchronization operations on the 88100, including examples of locking. The reader may find this a useful additional description of the topic, but should be aware that one aspect of that discussion does not apply to the TC2000 machine. Namely, TC2000 memory is not all on one bus. In fact, none is directly on the M bus. Therefore, the shared snooping caches described there do not exist in the TC2000 machine, so that method of alleviating the cost of a spin lock is not available.

2.9.4 Atomic Functions Based on xmem

The CPU atomic instruction xmem is available in C and Fortran via library functions that compile straightforwardly into the 4-byte and 1-byte versions of the assembly language instruction. For details of these calls, see the *TC2000 Programming Handbook*, or the manual pages in the *nX Programmer's Reference*.

Among its various uses, xmem may be used to obtain a lock on other data, or the lock and data may be combined in one word as follows. The programmer may construct an application-specific operation that is protected by a lock to ensure atomicity by setting aside one bit of a 32-bit value as a lock. Then the xmem instruction can be used efficiently to implement a 31-bit atomic operation. To obtain the lock, xmem is used to swap a value with its lock bit set, with the location in memory. If the result obtained also has the lock bit set, the location was locked and the code tries again (perhaps after a short delay). This is repeated until the value obtained has its lock bit clear. The application-specific operation is then performed, the flag bit is forced clear, and the result is written back into memory. The bit chosen as a lock might be the high order bit, the sign bit. If so, negative numbers could not appear in a location maintained with this mechanism. Normally, a location maintained with this mechanism should not be referenced in other ways (load, store, or simple xmem) because it is not atomic with respect to those. However, in some applications it may be acceptable to read the location normally. For instance, if the program is looking for a specific value of the (31-bit) counter, then the locked value is certain to not match because its high bit is set.

2.9.5 Atomic System Calls

The current nX operating system provides calls to perform certain frequently used operations atomically. These are implemented with the TC2000 locking protocol, described in section 2.9.7. They are somewhat slower than the xmem instruction because they trap to the operating system, whereas xmem is part of the 88100 instruction set. Therefore, if a construct (such as mutual exclusion) can be implemented either with xmem or with one of these calls, xmem should be used.

These calls perform two consecutive references to the target location, just as xmem does, so they do not load the memory or switch any more than xmem. In this respect they are preferable to user-constructed "31-bit" atomic operations described in section 2.9.4, that load the memory and switch more and are clumsier to use.

The atomic system calls (except atomcas) can be applied to 8–, 16– or 32-bit target locations, and the full name of the call has "8", "16" or "32" appended to the basic name, respectively. (For example, atomadd32 or atomior8.) Also, several are available in both a signed and an unsigned version. In these, a "u" is inserted between the basic name and the size suffix, to specify the unsigned version. For further information, please consult the corresponding manual pages.

atomadd ADD a quantity into a memory location, returning the original contents of the location. Signed and unsigned versions.

atomand AND a mask into a memory location, returning the original contents of the location.

atomcas compare the contents of a memory location against a comparison value. If the two are equal, store an update value into the location. If the two are not equal, do not change the location. In either case, return the original contents of the memory location. The "cas" stands for "compare and swap". 32-bit operation only, not 16 or 8.

atomcta clear the bits in a memory location as specified by a mask, and then add a specified value to the result, store the sum back in memory, and return the original value of the memory location. The "cta" stands for "clear then add". Signed and unsigned versions.

atomff0andset
find the first (most significant) "0" bit in a memory location ***and set*** that bit to "1", returning the bit number found, or 32 if there was no "0" bit.

atomff1andclear
find the first (most significant) "1" bit in a memory location ***and clear*** that bit to "0", returning the bit number found, or 32 if there was no "1" bit.

atomior inclusive OR a mask into a memory location, returning the original contents of the location.

atomload and **atomstore**

load a value from, or store a value into, a memory location. These calls are ***provided for portability*** to architectures that do not have hardware support for the other atomic operations. On the TC2000 machine, these translate into simple loads and stores. On a machine that emulated the atomic operations in software, the implementation would be more complicated. Signed and unsigned versions.

2.9.6 Atomicity of Memory Accesses

As one would expect, loading or storing a **byte** of data is atomic. If two processes are trying to write the same byte in parallel (that is, at the same time), the operations get serialized — one write is performed, then the other. The byte never ends up with some bits from one write and some from the other. Similarly, if one process is reading a byte while another is writing it, the reader always obtains a consistent value, either the eight bits before the write or the eight bits after the write. This may seem an obvious point, but the operations on larger quantities of data discussed next can be understood as analogous to this simple case.

A **halfword** is two bytes, aligned on a two-byte boundary (that is, at an even byte address). Reading or writing a halfword is atomic on the TC2000 computer.

A **word** is four bytes, aligned on a four-byte boundary. Reading or writing a word is atomic on the TC2000 computer.

A **cache line** is sixteen bytes, aligned on a 16-byte boundary. Data is stored in the instruction and data 88200 CMMU chips, in units whose size is one cache line. If any data in a cache line is cached, all 16 bytes are cached. When data is read into a CMMU to **fill** a cache line, or when it is written from the CMMU out to memory to **flush** a cache line, the four words comprising the cache line are read or written in a rapid **burst**. The TC2000 hardware supports the burst read and burst write of a cache line, by locking out any other access to the memory that is being accessed. Therefore, ***filling or flushing a cache line is atomic on the TC2000 computer.*** This is true whether the memory is local or remote, or even on an attached VMEbus device (so long as the VMEbus device is not dual ported and permitting simultaneous accesses).

Ordinarily, the CMMU hardware and the operating system software manage the filling and flushing of cache lines, so the user is neither concerned with nor able to control these operations. And indeed, the user never has complete control over them. But some system calls are available to permit sophisticated programmers to implement their own caching policy.

vm_cache_setup

Specifies how a specified region of the caller's memory is cached.

vm_cache_flush

Forces data in a specified region of the caller's cached memory to be marked invalid and/or written out into memory.

For example, by properly controlling the caching (and alignment) of a 4–word data structure, the user could implement application–specific atomic operations on the data, that would execute quite efficiently. There are hazards and pitfalls, however. For instance, in the middle of what is intended to work as an atomic sequence of instructions modifying the 4–word structure, the CMMU or the operating system might flush the cache line to memory. If another process could access it before it was re–loaded and the "atomic" sequence completed, that other process would see an inconsistent state of the data.

For further discussion of vm_cache_setup and vm_cache_flush, see the *TC2000 Programming Handbook*.

2.9.7 TC2000 Locking Protocol

The TC2000 hardware supports a locking protocol that may be used to achieve both speed and atomicity. The basic concept is the same as that used to support the 88100 instruction, xmem, described in section 2.9.3. The **resource** is locked to prevent any other access to the resource until the lock is freed. On a remote access, the **switch path** is locked also, both for efficiency of subsequent references and to support freeing the resource (it is freed when the switch path is torn down).

Locking is implemented as a side effect of data references to locations with a special bit set in the mapping register used by the reference. The data reference itself establishes (opens) the lock, so locking is sometimes described as an "augmentation" to normal instructions that load or store data.

NOTE

USE RESTRICTED TO SUPERVISOR MODE
The lock protocol is not accessible in user mode under the current nX operating system. The current pSOS^{+m} operating system cannot prohibit use of the lock protocol, since the user may enter supervisor mode, but pSOS^{+m} does not facilitate use of the lock protocol through any special features.

Lockable Target Devices

There are two lockable resources in the TC2000 architecture: memory modules and VMEbus interfaces. A **memory module** is all of the memory on a function board. While a memory module is locked, no normal access to it is possible except by the device holding the lock. (We say "normal" access because there is a special way to bypass locks, described later.) For example, a processor's access to data in local memory is delayed if a remote processor has the memory locked.

A **VMEbus interface** provides two windows, one in each direction, between the T–bus of a function board and a VMEbus system. (It also provides control and interrupt functions.) The TC2000 locking protocol is supported in both directions. The window from the T–bus (where it acts as a slave device) to the VMEbus system (where it acts as a master device) supports locking by holding mastership of the VMEbus until the lock is freed. This provides atomicity of operations arising in the TC2000 that access VMEbus address space, unless the VMEbus device is dual ported to allow separate access via another path.

Lockable Switch Paths

The access paths that support locking are the T–bus and the TC2000 switch. Each function board has a **T–bus** interconnecting the modules on the board; these modules may include a CPU interface, memory, and a VMEbus interface, and a switch interface is always included. The T–bus supports locking by transporting signals that describe each access as normal, opening (establishing) a lock, maintaining an already existing lock, or (as discussed later) bypassing a lock. The T–bus itself serves only one access at a time, but it does not stay locked. The target device remembers whether and by whom it is locked, and enforces any access restrictions while it is locked. For example, a remote CPU may lock the memory of a function board, and while it is locked may make several accesses to it. Between those accesses, the T–bus is inactive, and the local CPU may use the T–bus to access remote memory or its VMEbus interface.

The TC2000 **switch**, when involved in a locked access, supports locking by holding the switch path open between the function board doing the locking and the function board on which the locked target device resides. The switch path is kept open until the lock is freed. A locked switch path entails the following subtleties:

- All remote references made by the locker until the lock is freed must be to the same remote function board. Only one outgoing switch interface exists per function board, so to reference a different remote function board that interface would have to abandon the locked, held switch path. Doing so would violate the locking protocol, free the lock, and result in an error. Note that if multiple addresses in interleaved memory are accessed, their physical locations may be on different function boards even if their virtual addresses are contiguous and/or on the same page.
- No other function board in the machine can access the target function board until the lock is freed. Only one incoming switch interface exists per function board, so any attempt to access that board from another function board will be rejected (and automatically retried, eventually succeeding but suffering delay).

- In theory, the locker may hold multiple target devices on the target function board locked simultaneously, because the switch permits an "open lock" message even though a locked transaction is already in progress. When the locker terminates the locked sequence, all devices it had locked on the remote function board are freed. In theory, multiple devices on the local function board may be locked as well, but this is especially an issue in remote locked accesses because the switch interface must explicitly permit it. However, no lock–generating interface (CPU interface and VMEbus slave interface) is implemented to generate a second "open lock" message when it already has a lock open. Therefore, ***in practice the locker can hold only one target device locked at a time.***

Generation of Locked Accesses

A locked access can arise from only two sources: the CPU and the VMEbus interface. In the **CPU interface**, the *lock* bit in the Augmentation Register (AR) controls locking. When the CPU makes an access (to a non–cached location) while the lock bit of the AR is set to one, the access locks the resource addressed and, if it is a remote access, the switch path. Clearing the AR lock bit to zero generates a special "free locks" cycle on the local T–bus, unlocking any local locked resources. If remote resources were locked, they and the switch path to them are also freed.

Locking is a side effect of a ***data*** reference; an ***instruction*** fetch never generates a locked access.

NOTE

CACHING AND LOCKING
The xmem instruction never operates on cached data. Circuitry in the CMMU recognizes the xmem, and if the location is cached and is dirty (has been modified in the cache), the cached value is written out to memory. In any case, the xmem then operates directly on memory, not on data in the cache. However, locking using the Augmentation Register does not receive this special treatment. If the CPU loads or stores a cached location while the AR *lock* bit is set, the reference is serviced from the cache and the memory module is ***not*** locked. As a general principle, it is best to keep locations that will be locked non–cacheable.

When the **VMEbus interface** responds as a slave to an access on the VMEbus that falls within the slave map window, an access into TC2000 address space results. The VMEbus slave mapper translates the VMEbus address into a TC2000 System Physical Address. The VMEbus slave mapper also produces various control signals that accompany the access, one of which specifies locking. The mapping, locking and other access control signals are selected by the high 19 bits of the VMEbus address, while the low 13 bits are passed straight to the System Physical Address. Therefore, mapping, locking and the other controls have a granularity of 8–kilobyte pages. If the access from the VMEbus falls on a page set up for locking, the reference is a locked access; otherwise, it is not. The lock is held until the VMEbus master device originating the access drops mastership of the VMEbus; at that time, a "free locks" cycle is generated, freeing whatever local, remote and switch path resources were locked by the access.

Bypassing Locks

A locked target device (memory module or VMEbus interface master mapper) does not service normal access requests except from the locker. Instead, it responds with a "refused — locked" reply on the T–bus; the requester gets a bus error and may try again. However, the access can be accompanied by a special "bypass" signal that makes the locked device respond as if it were not locked. In effect, the access ***bypasses the lock***. Note that bypassing does not circumvent a switch path that is being held by a locked sequence. The purpose of bypassing is primarily so that the CPU can access instructions and page tables in its local memory, even if some device has the local memory locked.

In the CPU interface, the bypass signal is generated by the CPU Mapping RAM, and it controls 8–megabyte blocks of the address space. The software may set the bit before the locked operation and clear it afterward. Or, the software can maintain one or more 8–megabyte blocks as "bypass access windows", however is convenient.

In the VMEbus slave mapper, the bypass signal is generated independently for each 8–kilobyte page, just as the lock signal is.

All instruction fetches, and page descriptor fetches made by the CMMU, automatically bypass locks. These are the only references that automatically bypass locks.

Certain data references, such as those associated with exception processing in the CPU, should bypass locks. System software is responsible for setting up mapping so that the following data references are mapped bypassed:

- Memory management unit page table walks (namely, segment descriptor fetches made by the CMMU; page descriptor fetches made by the CMMU are automatically bypassed)
- Exception vector fetches
- Supervisor stack references
- Configuration and control register references (see note below)

NOTE

LOCK AND BYPASS — WHO WINS?

The most important effect of bypassing is to ignore locking held by someone else. However, it can also affect your own locking, as follows. A reference made by the CPU may be marked as **locking** by the Augmentation Register *lock* bit. Independently, it may be marked **bypass** by a bit in the selected CPU Mapping RAM register. If both bits are asserted, how does such a reference operate? The bypass property overrides the locking property; such a reference will not establish a lock and will circumvent a lock held by someone else.

NOTE

BYPASSED ACCESS TO CPU INTERFACE REGISTERS

The configuration and control registers in the CPU interface cannot be locked, and therefore it is not strictly necessary that they be accessed bypassed (by mapping them bypassed). However, an attempt to lock these registers will get a bus error, and referencing them with the xmem instruction or inside portions of lock-generating (AR lock bit set) code may be very convenient. One simple way to permit these references without getting an error is to map the registers bypassed.

CAUTION

BYPASSING AND XMEM

The **xmem** instruction normally makes a locked, read-modify-write reference. But if the target location of the xmem is mapped bypassed, ***bypass overrules the xmem*** and the xmem reference becomes a read reference followed by a separate write reference. An xmem reference to an address mapped bypassed is ***not atomic***.

CAUTION

BYPASSING TC2000 LOCKING — USE CARE
When the application program references certain areas of memory bypassed, the programmer should design the program with this in mind. **Writing** data structures with a bypassed access into an area normally protected by TC2000 locking is questionable and potentially dangerous, because it can corrupt the structure for all processes that access it. For instance, a bypassed write to clear a lock that is normally accessed using xmem can leave the lock locked by nobody. If such writing is necessary, the user should take great care, and protect the consistency of the data through higher level flags or execution control. **Reading** such data structures with a bypassed access may be less disastrous, because its immediate effect is limited to the reader. For instance, it may catch an inconsistent snapshot of the program's variables, if TC2000 locking is normally used to ensure consistency. If the data structure is updated all at once — with a single write — then reading it obtains a consistent result, either the old state or the new, which may be acceptable in some cases.

Considerations in Using the TC2000 Locking Mechanism

The points listed below should be considered when using the locking mechanism on the TC2000 computer.

- Use of locking is currently restricted to supervisor mode, under both nX and pSOS^{+m} operating systems.
- As noted earlier, access to remote resources during a locked sequence is restricted to one remote function board. Remember that interleaving spreads contiguous chunks of virtual address space among several function boards.
- A locked sequence cannot be arbitrarily long. The CPU Lock Timer prevents the CPU from holding a lock longer than its setting, with a maximum of 255 microseconds. Upon expiration, a "free locks" cycle is automatically generated. If a remote resource is locked, the Connection Timer tears down the switch path when it expires, also with a maximum of 255 microseconds. All the timers are discussed in section 2.10.
- Keeping a resource locked for a long time can adversely affect the execution speed and latency of other processors, and increases the risk of timeouts. For example, devices in an attached VMEbus system may be more sensitive to access latency than TC2000 function boards are. Consequently, it may be advisable to set up the VMEbus slave mapper so that its accesses bypass locks.
- If an exception occurs during a locked sequence, the exception handler needs to know what to do after processing the exception; should it continue normally, restart the sequence, or abort (the exception is fatal to the process)? To aid the software, the Augmentation Register contains an

exception action field. The code that uses locking can set this field to tell the exception handler what to do.

The TC2000 lock protocol provides an efficient means to implement a great variety of atomic operations. However, because of its current restriction to supervisor mode, the nX application programmer must leave its use to the operating system. The alternative to the hardware lock protocol is to implement software locking using one or more of the atomic operations provided by the system. Software locking gives the programmer a free rein to tailor the mechanism and the operations it protects to suit the needs of the application, without the time constraints of the hardware lock protocol.

2.10 Timers and Interrupts

Timers are one source of interrupts in the TC2000 computer. Figure 2–10 describes the timers in the TC/FPV function board, and is mostly self-explanatory. Two timers deserve further discussion here. The **Interrupts Disabled Timer** starts counting when interrupts to the CPU are disabled (by setting a bit in the Augmentation Register). It is used to ensure that interrupts are not disabled so long that the maximum interrupt latency desired cannot be guaranteed.

The **Interrupts Pending / Abort Retries Timer** helps limit the interrupt servicing latency. Interrupts are serviced only after instructions complete. So, if an interrupt arrives after an instruction begins a remote reference, and the setup of the switch path encounters delay, the interrupt servicing could be delayed undesirably. This timer starts counting when an interrupt request arrives. If it expires, the switch interface is told to abort all connection retries until all interrupt requests are gone.

Figure 2–10 Timers in the TC/FPV.

Name and Range	Purpose	Action on Expiration
	LOCK AND INTERRUPT TIMERS	
CPU Lock Timer 1 - 255 microseconds	Limit how long the CPU may hold a lock.	Generate a FREE_LOCKS cycle. CPU will later get a "maintain present" error.
Interrupts Disabled Timer 1 - 255 microseconds	Help guarantee maximum interrupt service latency.	Interrupt.
Interrupts Pending / Abort Retries Timer 1 - 255 microseconds *	Help guarantee maximum interrupt service latency.	Signal SIGA to abort retries in case connection establishment is in progress. CPU gets bus error if retries are aborted.
	SWITCH PROTOCOL TIMERS	
Reject Timer 1 microsecond - 0.49 seconds *	Prevent SIGA from trying too long to establish a connection.	Bus error.
Connection Timer 1 - 255 microseconds *	Prevent switch connection from being held open too long.	Tear down connection. CPU gets bus error — code and timing depend on when timer expires.
	REAL TIME CLOCK TIMERS	
Time Of Next Interrupt - A (TONI-A) 1 microsecond - 1 hour *	Allow software to ask for an interrupt at a specified time.	Interrupt.
Time Of Next Interrupt - B (TONI-B) 1 microsecond - 1 hour *	Allow software to ask for an interrupt at a specified time.	Interrupt.
	VMEbus INTERFACE TIMERS	
VMEbus Arbiter Timer 4 - 1020 microseconds	Limit how long VMEbus bus grant may be asserted without bus busy.	Arbiter removes bus grant.
VMEbus TC/FPV Master Bus Timer 1 - 255 microseconds	Limit how long the TC/FPV as VMEbus master may await a response from a slave.	Assert VMEbus signal BERR.
VMEbus System Bus Timer 4 - 1020 microseconds	Limit how long any VMEbus master may await a response from a slave.	Assert VMEbus signal BERR.

* Also, these timers can be disabled by software.

The 88100 CPU has only one interrupt line, so the TC/FPV function board circuitry combines various interrupt sources into one signal applied to the 88100.

Further, the TC/FPV interrupt system provides information to the CPU allowing it to determine what interrupt source(s) are currently asserted, and means to dismiss and/or disable certain interrupts. Figure 2–11 shows the logical derivation of the interrupt signal presented to the 88100 CPU in the TC/FPV. The gating is conceptual and does not necessarily reflect the gate level implementation in hardware. Note that if the 88100 has internally disabled interrupts (by setting a bit in its Processor Status Register), none of the interrupt sources shown in Figure 2–11 will produce an interrupt until that PSR bit is cleared.

Figure 2–11 TC/FPV interrupt derivation.

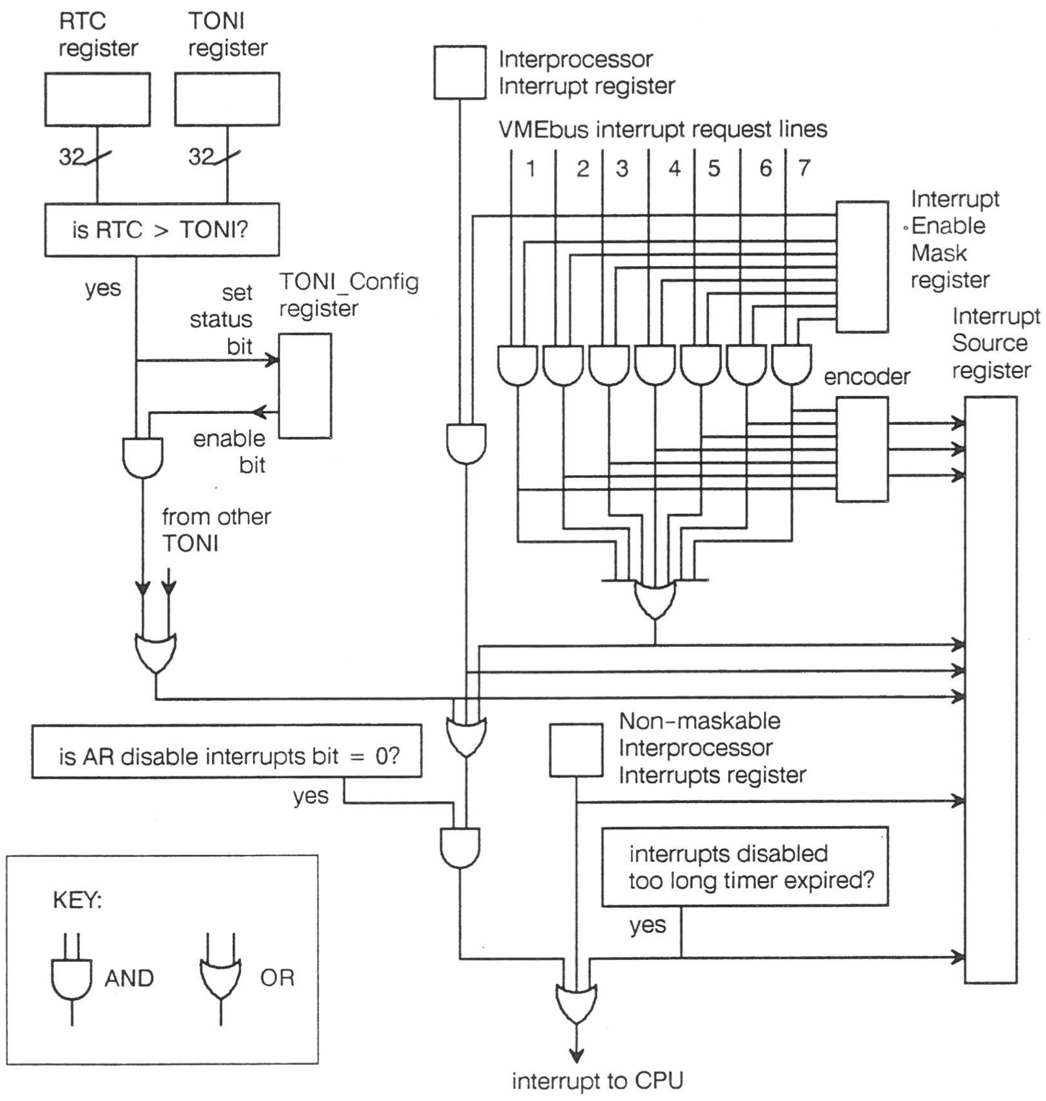
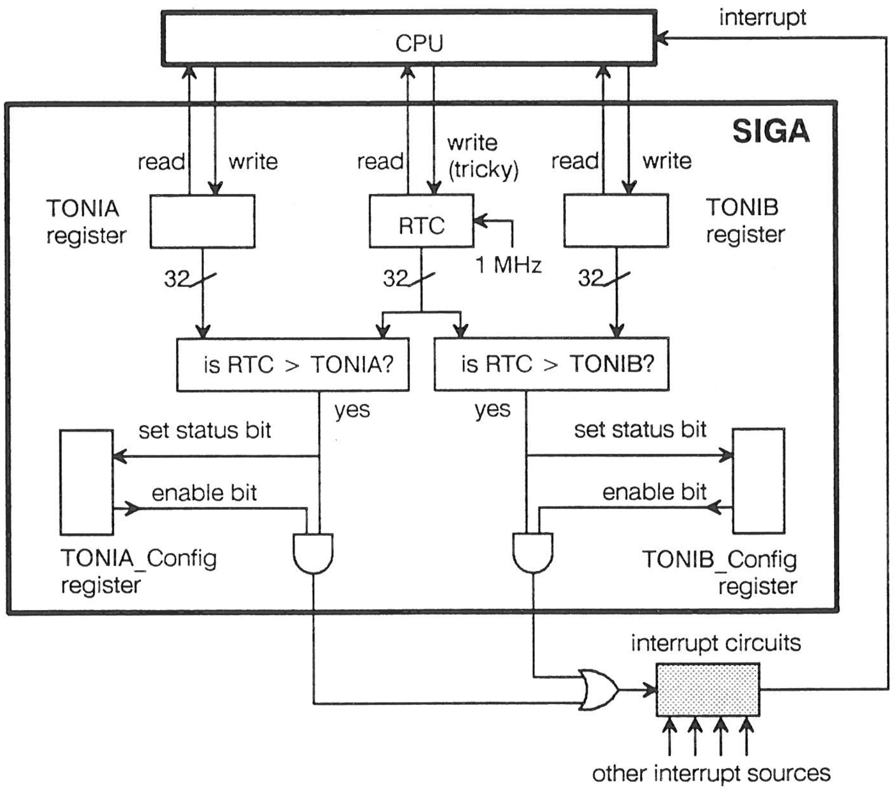

RTC register
TONI register
32
32
is RTC > TONI?
yes
set status bit
TONI_Config register
enable bit
from other TONI
Interprocessor Interrupt register
VMEbus interrupt request lines
1 2 3 4 5 6 7
Interrupt Enable Mask register
encoder
Interrupt Source register
is AR disable interrupts bit = 0?
yes
Non-maskable Interprocessor Interrupts register
interrupts disabled too long timer expired?
yes
interrupt to CPU
KEY:
AND
OR

Figure 2–11 shows the derivation of the real–time timer interrupt signal for one of the two TONI registers in the switch interface. Figure 2–12 shows this programmable timer mechanism in greater detail. Within the Switch Interface Gate Array (SIGA) are two independent Time Of Next Interrupt (TONI) systems. Each system consists of a TONI register, a comparator, a configuration register and an output signal. The SIGA compares the TONI register to its Real Time Clock (RTC) every microsecond. The comparison sets the status bit in the associated TONI configuration register, and also generates an interrupt request if enabled. The TONI mechanism is implemented in the SIGA for simplicity — the RTC is implemented there because it is driven by the switch clock signal and is used in other switch message processing.

Figure 2–12 **TC/FPV TONI mechanism.**



2.11 Bus Errors

There are a variety of conditions in the TC2000 machine that terminate CPU cycles with a bus error. Because the number of bus error conditions is rather large, bus error causes are prioritized and encoded. The CPU can read the encoded information from the Bus Error Vector register and use it as an offset into a dispatch table in the bus error handler. This mechanism is included to improve the bus error service latency.

3

The Butterfly Switch

3.1 Importance and Name

The Butterfly switch distinguishes the TC2000 computer from many other multiprocessor designs. Its importance is reflected in the fact that the custom VLSI chips in the TC2000 implement and support the switch; other portions of the machine are assembled using commonly available parts.

Figure 3–1 **Two–by–two crossbars.**

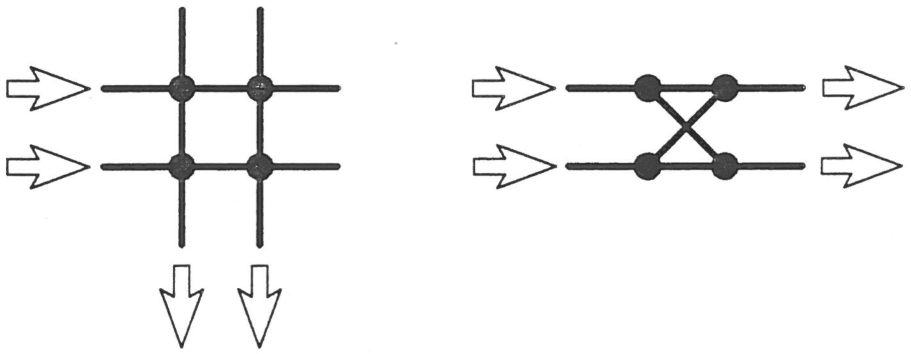


Figure 3–1 shows a two-by-two crossbar switch in two notations. The left diagram shows two horizontal wires and two vertical wires. Each of the four intersections of wires is a *crosspoint*. The wires at the crosspoint are normally insulated from each other, and closing the crosspoint connects them. This diagram resembles the physical construction of electromechanical crossbar switches once used in telephone exchanges. The diagram on the right shows two wires on the left side, each of which may be passed straight through to the wire on the right, or may be switched over to the other wire on the right. This diagram resembles railroad tracks, and the data flow in some Fast Fourier Transform algorithms. Its resemblance to a butterfly is the origin of the name "butterfly transform" in signal processing, and of the Butterfly switch.

3.2 Function and General Structure

3.2.1 Provide Access to Remote Boards

The primary function of the TC2000 switch is to **interconnect the function boards** so they can **access each other's address space**. The main reason to access a remote board is to access its **memory**. Also important, however, is access to **registers** in the remote board's CPU interface (such as the Interprocessor Interrupt register) and VMEbus Slave Map RAM. Access to devices on the remote board's VMEbus system may also be important in some applications.

A total of 20 signals to each function board support remote access. These are described in section 3.3.

3.2.2 Also Distribute Signals

A secondary function of the switch hardware is to distribute machine–wide signals. These fall into two categories, clock signals and TCS signals. They are described briefly now; the rest of this chapter deals only with the signals that support remote access.

- **Requester clock** — this signal runs at the switch clock frequency and is used by the requester section of the LCON and the SIGA. In the SIGA, it is divided down and runs the Real Time Clock.
- **Server clock** — this signal also runs at the switch clock frequency, but is either in phase or 180° out of phase with the requester clock, as selected on the machine's clock card, to adjust for switch data cable length. It is used by the server section of the LCON and the SIGA. No direct effects of the server clock are visible to the 88000. Both the requester clock and the server clock signals can be used as a CPU clock or other on–board functions, although they are not so used in the TC/FPV.
- **65 milliseconds pulse** — this signal is generated on the clock card and fanned out via each switch requester (TC/SR) card to each function board. It is asserted for one switch clock period every 65,536 microseconds. It is used in the SIGA's Real Time Clock circuit and helps to keep the RTC synchronized with those in other function boards.
- **TCS master to slave** — this signal originates in the TCS master and is fanned out by the clock card to each midplane, where it is delivered to the function boards and to the switch server (TC/SS) card. The TCS slave processor in each of these cards receives messages from the TCS master by receiving this signal.
- **TCS slave to master** — this signal carries response messages from the function board's TCS slave processor. The signal goes first to the switch

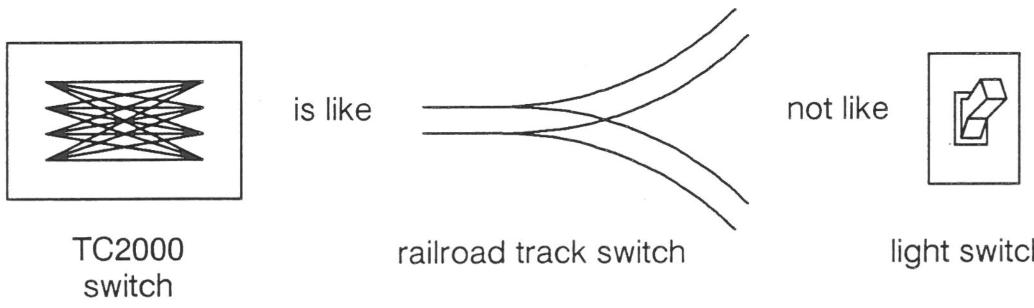
server (TC/SS) card serving the function board, where it is combined with the TCS slave to master signal from the other seven function boards, and with the TC/SS's own slave to master signal. That combined signal goes to the clock card for further fan–in, and then to the TCS master.

3.2.3 Structure of the Switch

The TC2000 switch is introduced in chapter 1, and its general structure is covered there. Here we examine it in more detail.

First, let's be sure we have the right image of "switch". The TC2000 switch is ***like a railroad switch***, not like a light switch. A switch in railroad track controls where the train goes; similarly, the TC2000 switch determines where the remote access message goes. The railroad switch and the TC2000 switch both determine a ***route***, a selection from alternatives. A light switch, on the other hand, is on or off; the power is present or absent. The light switch determines ***whether*** something happens (the lamp is lit), and the railroad and TC2000 switches determine ***how*** something happens (where the train or message goes).

Figure 3–2 Like a railroad switch, routes are made.

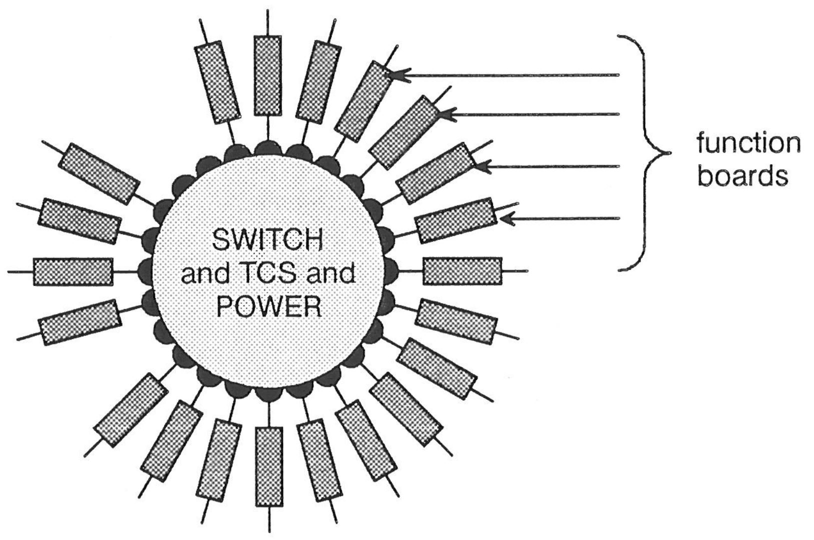


In chapter 1, we noted that the TC2000 switch (along with power and the TCS) is a central facility to which each function board connects. Let's look more closely now at that attachment. Figure 3–3(a) shows the TC2000 machine as in chapter 1, but without showing I/O systems. Also, here we show that some places where function boards could attach to the switch may not be filled.

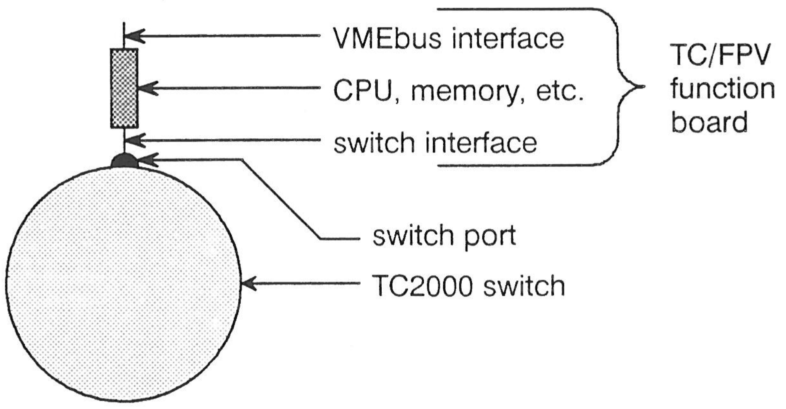
Figure 3–3(b) shows a close–up of how one function board attaches to the switch. The switch provides a connection point called a **switch port**, to which the **switch interface** portion of a function board connects.

Figure 3–3 **Switch ports in the TC2000 computer.**

(a)



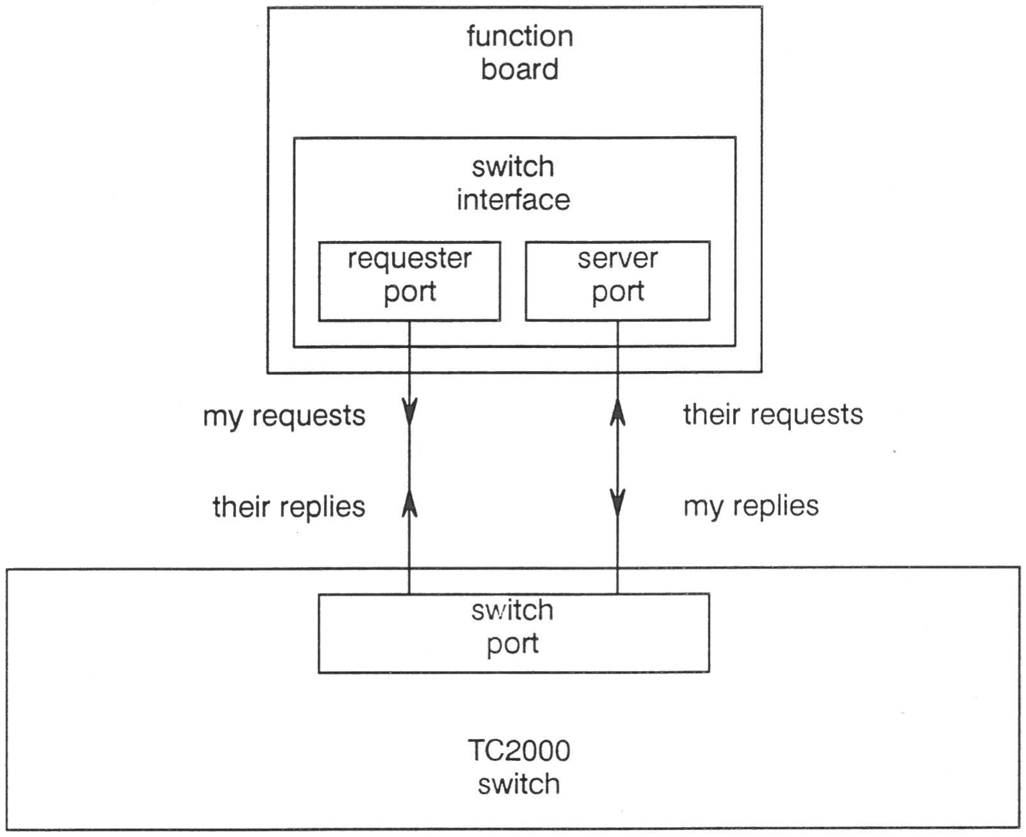
(b)



As we see in Figure 3–4, the switch port consists of two parts: a **requester** port and a **server** port.

- The function board uses the **requester port** to issue requests for access to remote function boards, and to receive replies to those requests.
- The function board supports the **server port** only for the benefit of other function boards in the machine. Their requests for access to this function board arrive on its server port, are serviced automatically by the function board hardware, and replies are sent back out the server port.

Figure 3–4 **Switch port = requester port + server port.**



Note that a reply to a given request is ***returned over the same path*** as that request. In fact, the very same wires carry the data; the TC2000 switch is **bidirectional**. This provides an important performance benefit to the user. Instead of having to establish a new path for the reply, the path is held open so the reply can return immediately. Holding the path does have the cost that other switch traffic can't use the individual switch elements in the path while the reply is being prepared, but this cost is small compared to the advantage gained.

NOTE

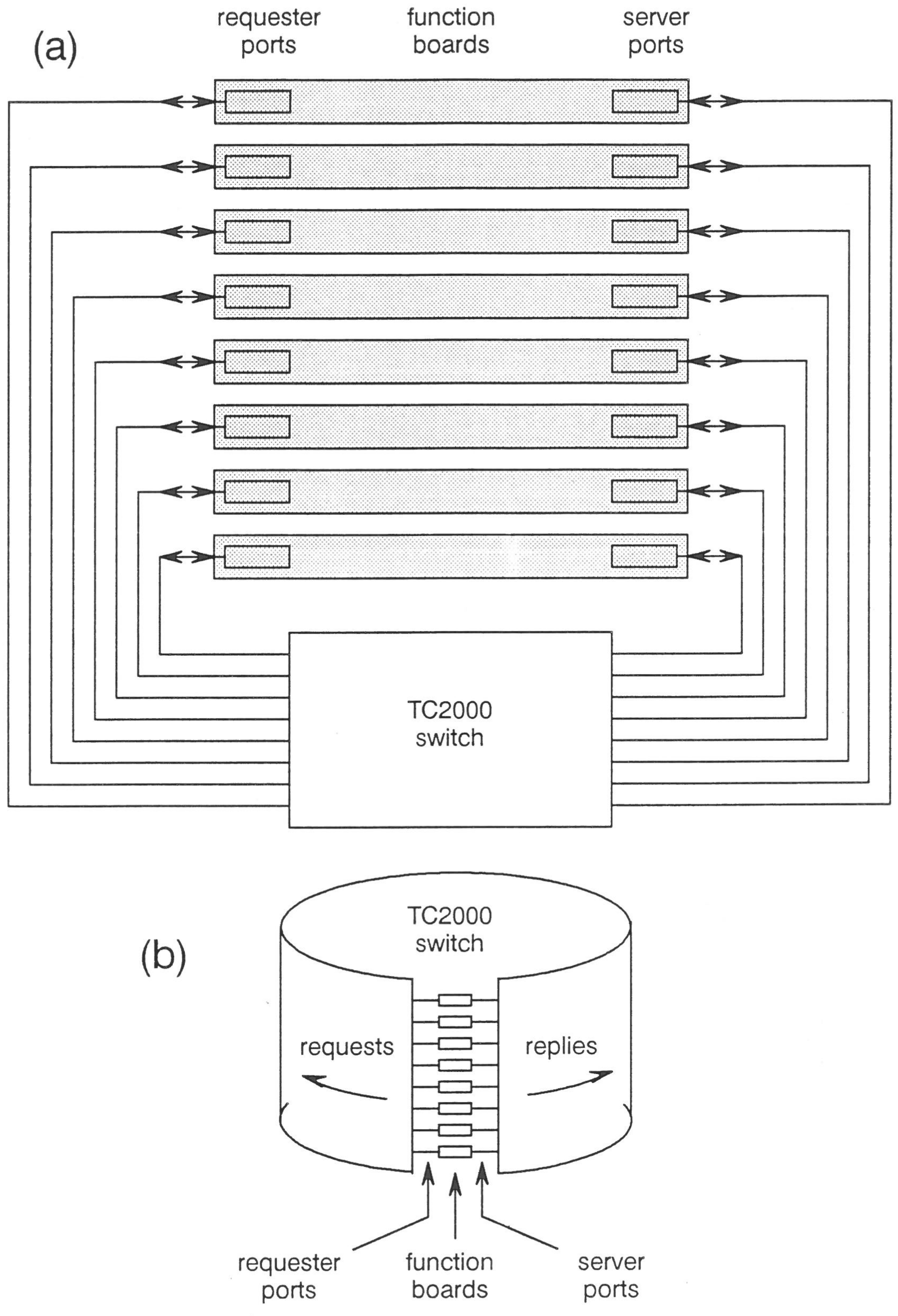
The GP1000 computer, a predecessor to the TC2000, did not have a bidirectional switch. In it, a new connection had to be established, over a new switch path, to return the reply. Users familiar with that earlier architecture should note this improvement in the TC2000 switch.

The switch hardware establishes a path through the switch by allocating, incrementally and locally, switch hardware resources to support the path. From the time a path through the switch is established, to the time it is torn down by freeing the resources, we say that a **connection** exists. The term "**path**" emphasizes the hardware resources and the physical route through the switch, and the term "**connection**" emphasizes the ability to communicate and thus to access a remote function board.

Normally, the requester port and server port on a function board are completely autonomous. They may, and often do, handle independent connections simultaneously. A function board may, if desired, access itself over the switch. In this case, both its requester port and its server port are involved in the same connection.

Figure 3–5 shows another way to think of the function boards and the switch. In (a), we have moved the requester and server ports to opposite sides of the function boards. Considering the switch as a piece of paper, we then lifted it out of the plane of the page and curled its sides forward almost into a cylinder. The requester port of each function board connects along one edge of the split cylinder, and the server port connects along the other edge, as in (b). This image emphasizes the separateness of the requester ports from the server ports; they are on opposite "sides" of the switch!

Figure 3–5 **Requester and server ports attach to switch "cylinder".**



NOTE

TERMINOLOGY: CROSSBAR UNIT = SWITCH NODE
In developing the conceptual description of the Butterfly switch in the following text, the term **crossbar unit** is used. The term **switch node** is commonly used to mean the same part of the machine.

Now we look inside the switch, to examine its internal structure. As Figure 3–6 shows, the TC2000 switch is composed of several linked crossbar units, arranged in columns. The requester side of the crossbar units in the first column connects to function board requester ports. The server side of the first column connects to the requester side of the second column. In principle, there could be several columns — the server side of each column connecting to the requester side of the next column. The server side of the final column connects to function board server ports. For machines with up to 64 function boards, only two columns are required. A 3–column switch is needed for machines with 65 function boards or more, up to the TC2000 architecture limit of 512 function boards.

The TC2000 switch is expanded by adding crossbar units, an equal number in both columns. This increases the total bandwidth of the switch as a whole, so that the bandwidth available per function board remains constant and does not become a bottleneck.

The crossbar unit is implemented as a printed circuit card: the TC/SR switch requester card (for the first column) and the TC/SS switch server card (for the second column). On each card, four identical Switch Gate Array (SGA) chips together implement the crossbar function; each SGA provides one quarter of the switching circuitry.

Figure 3–6 **Columns of crossbar units.**

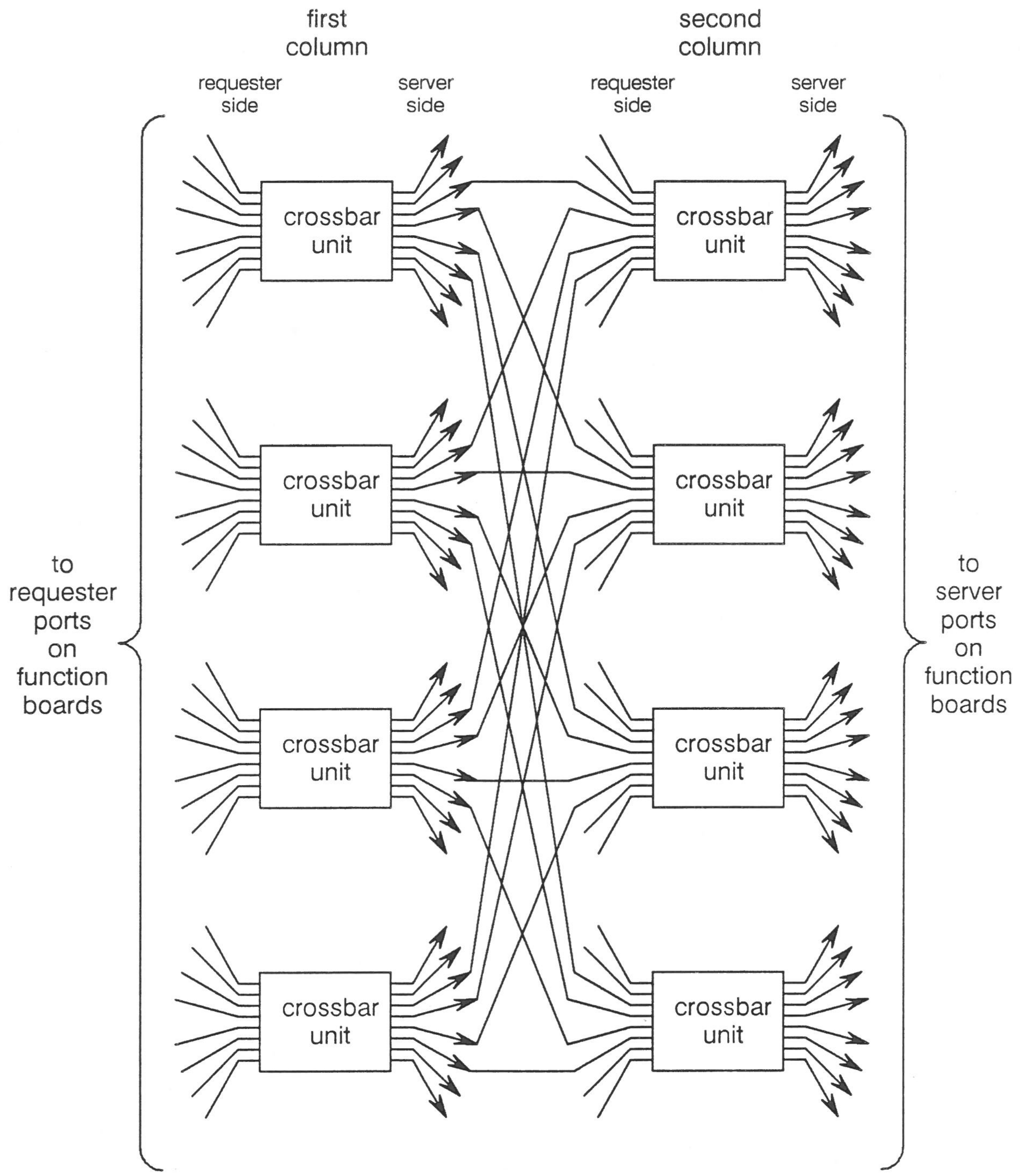
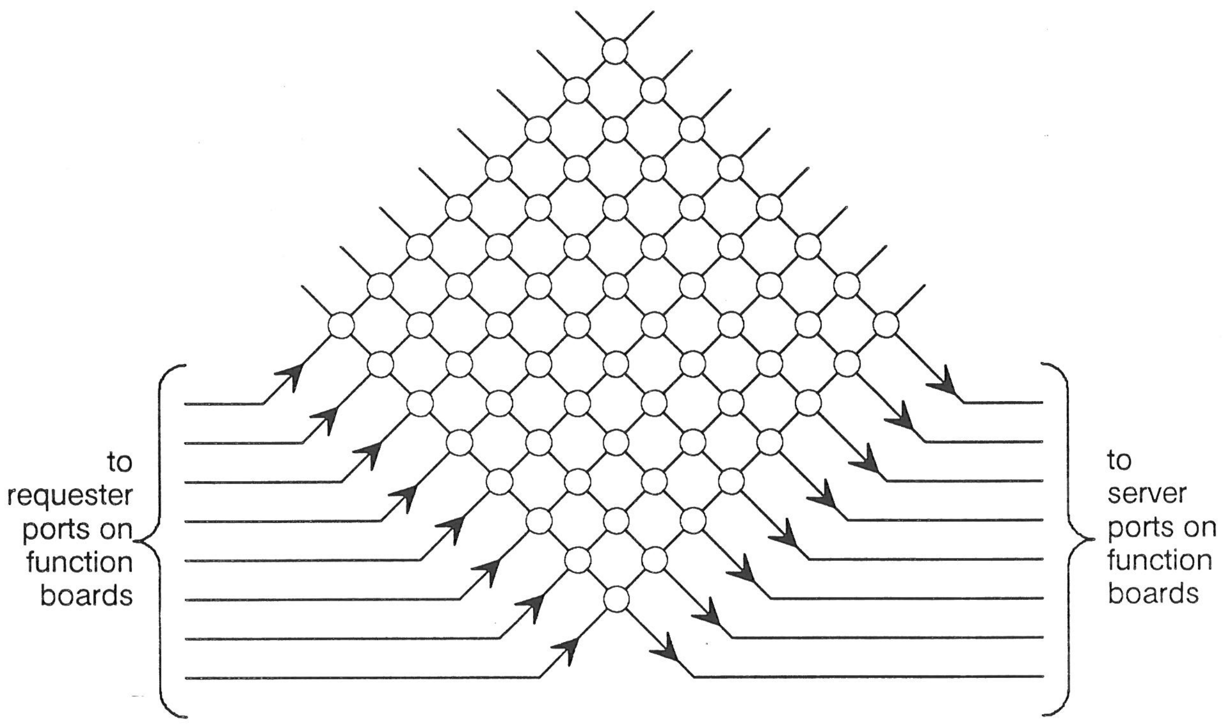


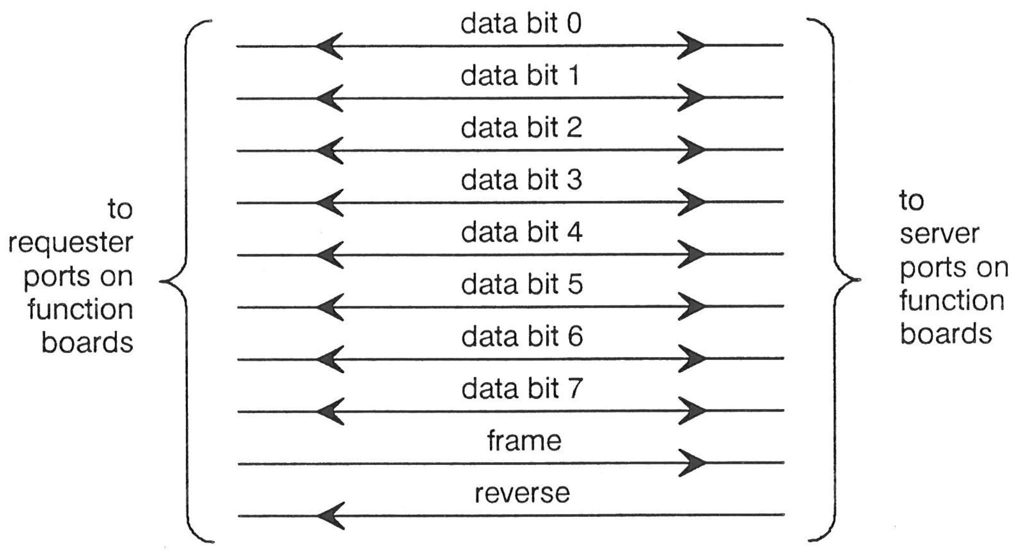
Figure 3–7 shows the insides of a crossbar unit. At each of the 64 circles, the switch can connect the two paths. The actual signals involved in each path are shown in Figure 3–8. The eight **data signals** are bidirectional, and the two **control signals** ("frame" and "reverse") always go in a single direction, as indicated.

Figure 3–7 **TC2000 switch crossbar unit.**


to requester ports on function boards
to server ports on function boards

Arrows show direction requests travel; replies travel in the reverse direction.

Figure 3–8 **The signals in one switch path.**



TC/SS and TC/SR Cards

A pair of switch cards — one TC/SR requester card and one TC/SS server card — provide the actual switching and interconnection in the TC2000 machine. The pair provides the switch connections for eight function boards. Those function boards access remote boards by sending a message into the requester side of the switch (TC/SR). Accesses coming from remote function boards are delivered by the server side of the switch (TC/SS).

The differences between the TC/SR and TC/SS are different clocking (noted above), different machine-wide signals (also noted above) and only a single TCS slave processor (on the TC/SS, serving both cards of the pair). Since these differences do not affect connections through the switch, we can ignore the TC/SR–TC/SS distinction in describing the structure of the switch.

The heart of the switch card is **four SGA chips**. These chips handle the switch connections. Other than the SGAs, the card contains a TCS slave and support circuitry.

SGA Chip

The SGA performs a highly specialized function: it establishes connections between its input ports and its output ports. In attempting to establish a connection, it detects conflicts and resolves them by rejecting connection requests. After a connection is established, it handles the reversal of data flow when replies come back from the server. The SGA detects the end of a connection, due either to normal termination or an error condition, and returns the circuitry that supported the connection to its idle state.

The SGA chip, like the LCON and SIGA described below, has a TCS interface. Through this interface, the TCS slave can test, monitor, and set parameters in the chip. This interface is important in testing and configuring the machine, but is far from the user and is not discussed further here.

LCON and SIGA Chips

The switch includes not only the switch requester (TC/SR) and server (TC/SS) cards, but also the LCON and SIGA chips on each function board. The **LCON** reclocks the signals to restore proper timing and a crisp waveform, and converts the signal voltage between TTL levels used in the SIGA (and the rest of the function board) for cost, power consumption and noise immunity, and ECL levels used in the rest of the switch for speed.

The **SIGA** performs many functions — indeed, it is essentially a dedicated, packet switching microprocessor. Among its functions are:

- SIGA requester section

  - Receive requests from the T-bus, turn them into switch messages, and transmit them into the switch
  - Store the message and retransmit it repeatedly until it gets through
  - Throttle the retransmissions of the message according to programmable parameters
  - Address the message differently to use alternate paths through the switch, if enabled to do so
  - At a regular interval, promote the message (if any) awaiting retransmission to be an express message, to bound switch latency
  - Receive any switch message returned from the remote end of the connection, extract the data, and send it on the T-bus to the module that asked for it
  - Detect error conditions and report them accordingly
- SIGA server section
  - Receive switch messages, extract the operation and data parameters, and perform a T-bus transaction as requested
  - If the operation was a read, obtain data from the T-bus, turn it into a switch message and send the reply back into the switch
  - Detect error conditions and report them accordingly
- RTC and TONI — Provide a clock, and a timer interrupt facility, for software
- Configuration and Status unit
  - Provide read and write access to programmable parameters of the SIGA
  - Support loading of certain fast mapping RAMs, including the interleaver RAM — the "interleaver loader" facility
- TCS interface unit
  - Provide access from the function board's TCS slave to the function board's T-bus, from where the TCS slave can access all modules on the T-bus
  - Provide the TCS slave direct access to certain functions in the SIGA

The only sections of the SIGA that are discussed in the rest of this chapter are the requester and server sections.

3.3 Theory of Operation

3.3.1 Switch Message Contents

Every switch message contains certain information, and the first message on any connection also contains further information used to establish the connection. These components are:

- **Header** — contains the **route**. The header is present only in the initial message on a connection, including any retransmissions thereof. The rest of the message is called the **body**.
- **Command** — specifies function (read or write), address, size, and locking. Present in every request message, and never in a reply message.
- **Data** — contains information needed to perform the function specified in the command component. The content of the data varies depending on the purpose of the message.
    - In a **write request**, the data component contains the data to be written.
    - In a reply to a successful write request, the data component contains a byte of unspecified value.
    - In a reply to an unsuccessful write request, the data component contains an error code.
    - In a **read request**, the data component is absent.
    - In a reply to a successful read request, the data component contains the data read.
    - In a reply to an unsuccessful read request, the data component contains an error code, and may contain some data read.
- **Checksum** — contains a "checksum" to detect corruption of the message while it traverses the switch, and a separate error indication to say an error occurred on the remote function board. The checksum component is present in every message.

3.3.2 Routing a Message, Making a Path

In the TC2000 switch, these actions are all the same thing:

Deliver the initial message to the switch port it addresses.

Deliver the initial message to the function board it addresses.

Find a route through the switch for the initial message to take.

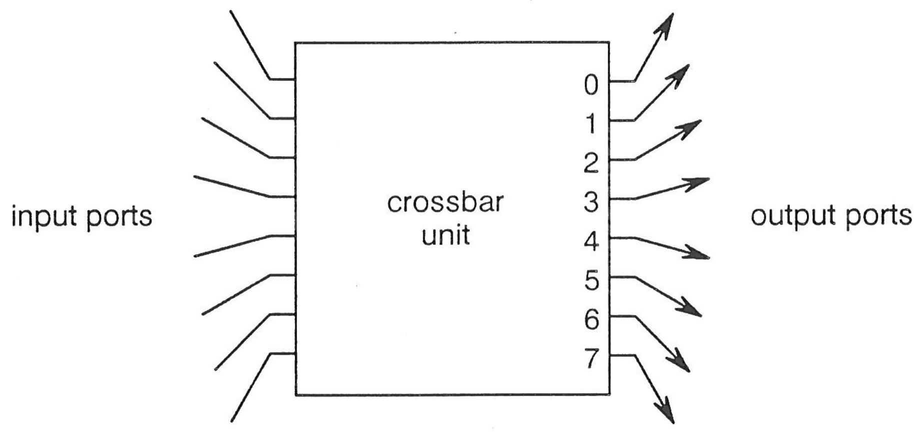
Establish a connection on which a series of requests and replies may flow.

The equivalence of these actions is a very important point in understanding the operation of the TC2000 switch. Not all computer interconnection switches have this equivalence, so ideas based on other architectures may not apply here.

The initial message contains a header component, and part of that header is the route. **The route specifies a complete and exact path through the switch.** The route is prepared by the requester SIGA. It is complete before the message leaves the SIGA, and is not changed. At each switching choice point (crossbar unit, quartet of SGAs) along the route, the route is either granted as specified, or is rejected. If the message is rejected, the SIGA will try again. Different messages the SIGA receives from the T-bus may use different (but specific) paths if there are alternate paths available in the machine, but once a message is received by the requester SIGA, the route in that message is not changed for retries.

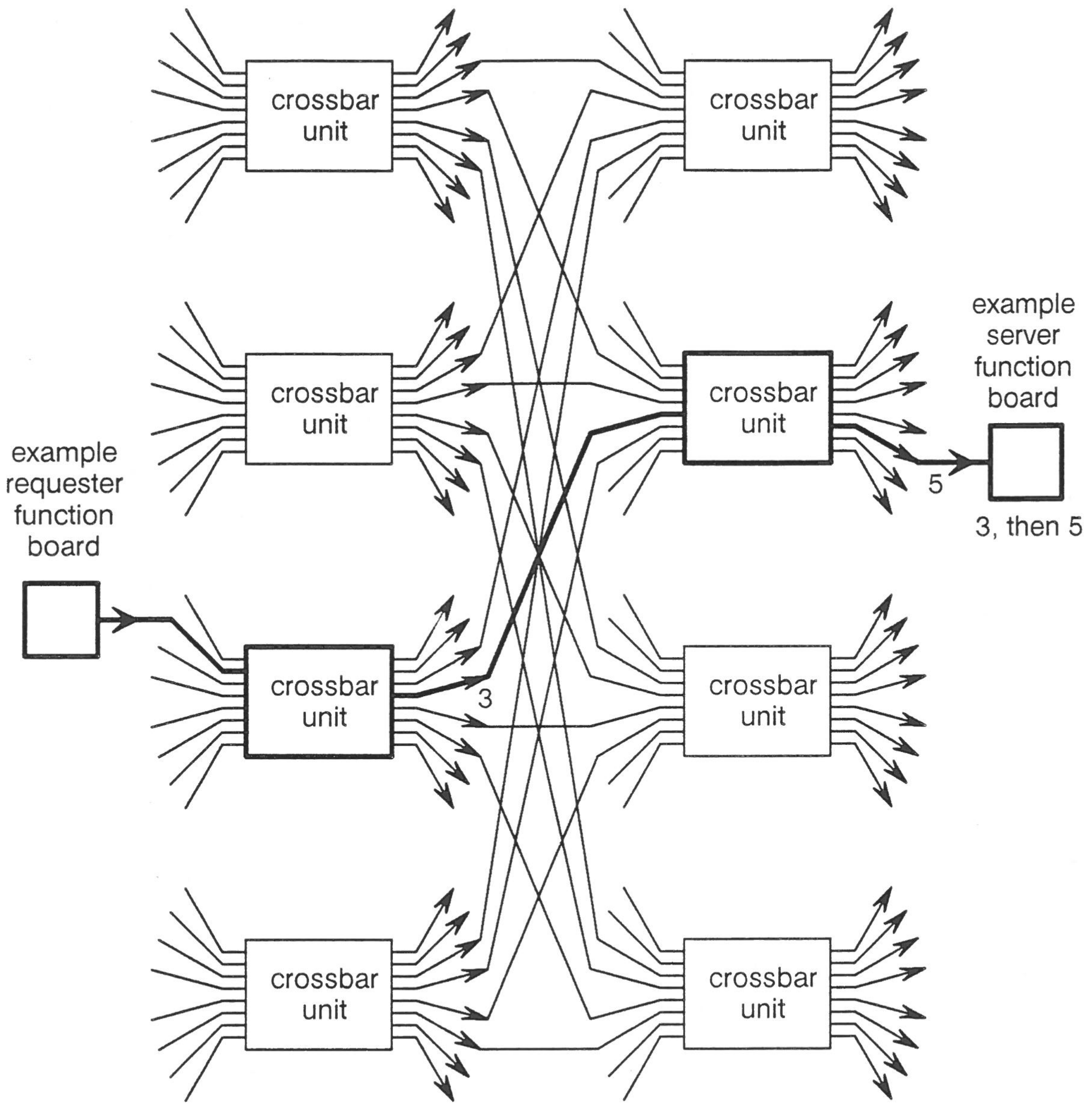
The message header specifies the route by telling each crossbar unit along the way what output port on that unit it should use. The unit's output port is specified in local, not global, terms. For example, "your output port number 5". In Figure 3–9, we see that each output port has a number associated with it. All crossbar units use the same numbering.

Figure 3–9 **Output ports are locally numbered.**



For example, suppose the route said, "in the first column, use output port 3, and in the second column, use output port 5". The message would travel a path shown in heavy lines in Figure 3–10.

Figure 3–10 Example path through switch.



The TC2000 switch routing has a very important property:

The message goes to the same destination, regardless of where it entered the switch.

You may wish to convince yourself of this by trying a few other cases. Pick any of the 32 requester side ports along the left; take the number 3 output in the first column; and take the number 5 output in the second column. You always arrive at the same port on the server side. We have emphasized this by labeling that function board "3, then 5" in Figure 3–10. Similarly, a message addressed to "4, then 2" would arrive at one particular server port, no matter where it entered the switch; and so on for every address.

It should now be clear why the four actions at the beginning of this section are equivalent. The initial message contains a header, and in the header is the complete route, so the switch itself does not have to do any *searching* to find a route for the message. If the canned route works, then the message reaches a unique switch server port. That server port connects to just one function board, so the route identifies a unique function board as well. And as the initial message traverses the path defined by its route, it acquires the switch hardware along that path, thus establishing a connection. These hardware resources are allocated to the path as long as the connection exists, so requests and replies may flow through them.

NOTE

You may wonder what would happen to a message with route "1, then 5" in Figure 3–10. Nothing is connected to output port 1, so the illustration doesn't tell us what would happen. In a real TC2000 machine, such auxiliary switch ports would be jumpered to auxiliary input ports in the second column, providing alternate paths through the switch. Can you see how to wire the switch so that route "1, then 5" will deliver any message to the same function board as "3, then 5"?

3.3.3 Route Format and Use

One further aspect of routing bears discussion. It is an optimization in how the route is located in the message, and how it is processed as it goes through the switch. The port numbers appear at the beginning of the message, with the first column's port choice sent into the switch first, followed by the second column's port choice. For example, the SIGA sends a message with route "3, then 5" into the switch by saying: "3", then "5", then the rest of the message. (If there were a third column in the switch, the port choice for it would be sent between the "5" and the rest of the message.)

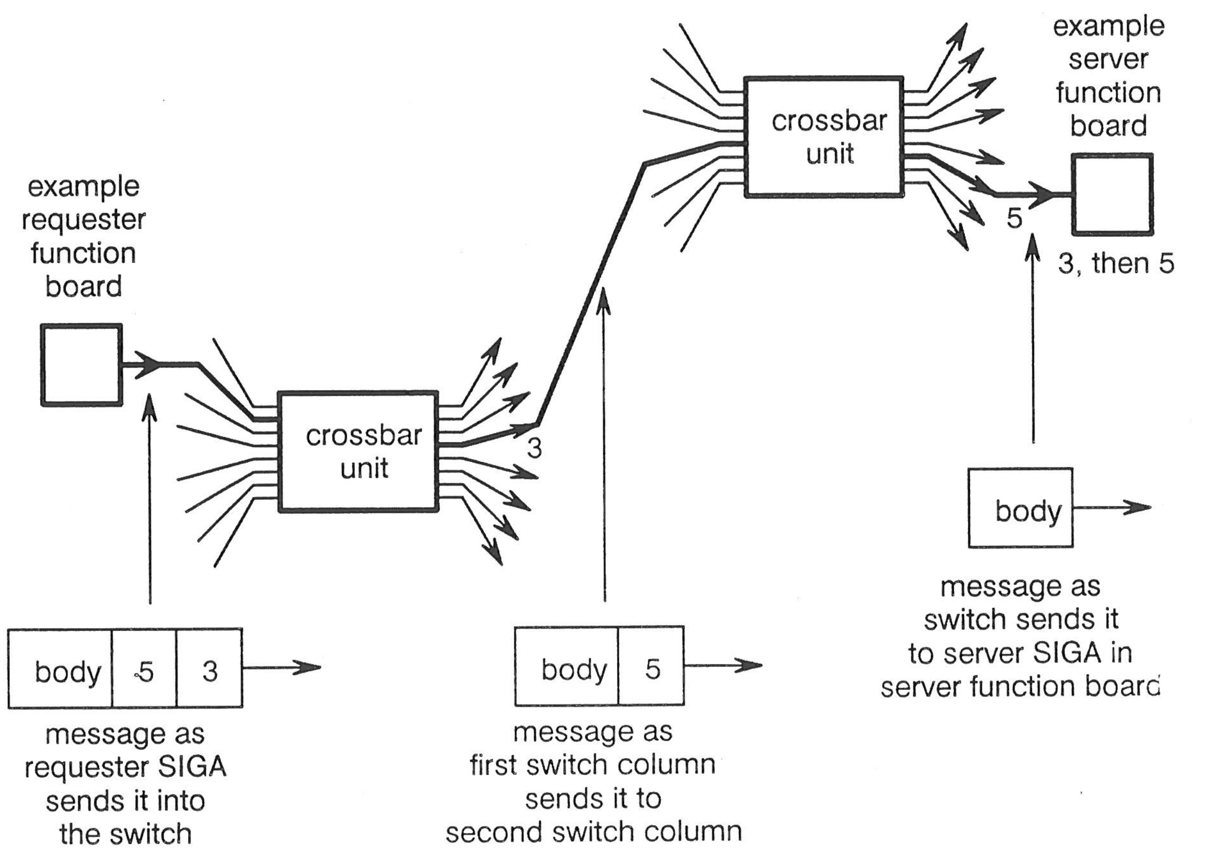
This route format allows each crossbar unit in turn to examine the first data it receives, use that data internally, and forward the rest of the message onward **without the port choice data it used internally**. This has several important advantages:

- The data needed to select an output port arrives first, so switching activity can begin immediately.
  - If the required output port is free, the resources needed can be allocated quickly and the rest of the message can be forwarded without delay.
  - If the output port is already in use, the message can be rejected right away, freeing up switch resources as soon as possible.

  - If the output port is at high priority, possibly due to an express message that is just now arriving, any normal (non–express) messages arriving and addressed to the port can be immediately rejected.
  - If the required output port is free but another message has arrived simultaneously that also requires the same port, the switch hardware performs an arbitration and immediately grants the port to one message (chosen randomly) and rejects the other, without delay.
- Because the routing data arrives first, and is either granted the output port or rejected very quickly, there is essentially no buffering in the switch. This simplifies the design and reduces delay.
- Each crossbar unit strips off the routing data it uses, leaving the next column's routing data at the head of the forwarded message. Therefore, each crossbar unit simply uses the first data it receives, as its routing data. This allows all crossbar units to be identical, regardless of which column (or where within the column) they are installed. Thus, there is **only one kind of SGA chip**, used in all switch cards. This reduces cost and enhances maintainability.

Figure 3–11 illustrates this route format and switching technique. When the message is delivered to the destination function board, it has no header.

Figure 3–11 **A message traverses the switch.**


example requester function board
crossbar unit
3
crossbar unit
example server function board
5
3, then 5
body
5
3
message as requester SIGA sends it into the switch
body
5
message as first switch column sends it to second switch column
body
message as switch sends it to server SIGA in server function board

Using a railroad analogy, a series of switchmen would line up at the front of the locomotive. As the train approaches each switch, the switchman at the head of the line jumps off and throws the switch to route the train on the proper track. The train proceeds on toward the next switch, with the switchman who was previously second now at the head of the line. In a railroad, the switchman who detrained might jump on the caboose as it passed; the analogy breaks down here, because the TC2000 switch simply discards the used route data. The message becomes slightly shorter as it travels through the switch.

3.3.4 Use of Alternate Paths

As discussed in chapter 1, the Butterfly switch in some sizes of TC2000 machine has multiple, or *alternate*, paths from each function board to each other function board. In such a machine, the hardware can be set up to automatically use these paths, helping to reduce congestion within the switch. In the TC2000, the SIGA selects a given route for an initial message before its first transmission into the switch, and does not change that route during any retries of the message. Different paths are used by separate initial messages, but not by separate retries. The intent of this is to spread out switch traffic on the time scale of connections, not on the scale of retries.

3.3.5 Reject and Retry

This section describes a mechanism that the switch hardware performs automatically. Under normal operation, the software is neither aware of nor affected by this automatic reject and retry mechanism. However, understanding the reject and retry mechanism is useful for these reasons:

1. Messages from the many function boards in the machine **contend** for switch resources. Inappropriate programming can lead to **congestion** beyond the normal contention. There are tools (described in software documentation) for measuring the performance of your application and detecting when that performance is degraded by congestion. Programming techniques are available to then restructure the program or its data to reduce the congestion.
2. In situations of hardware or software failure, or extreme congestion, the automatic retry mechanism will give up and notify the software of an error condition. If this occurs, you need to understand the retry mechanism to understand what the error report means.
3. The reject and retry mechanism is an interesting aspect of the machine architecture, if you like that sort of thing.

An initial message may be **rejected** as it travels through the switch. It is rejected by a crossbar unit if the output port required there is already in use (or is given to another initial message, by arbitration). Looking at Figure 3–10, we see there are two places the message could be rejected — one in the first switch column, and one in the second. If the message succeeds through these two places, it has succeeded in setting up a connection.

- An initial message is rejected if an output port it requires is busy. This is a natural occurrence, but when it happens a lot it is termed **congestion**.
- The rejection may occur at any switch column. Depending on where it occurs, it is symptomatic of congestion arising from different kinds of software activity.
    - If it occurs at the final switch column, that is because the port to the addressed function board is busy. Congestion of this kind happens when many function boards are accessing one particular function board (a "**hot spot**") at a high rate.
    - If it occurs before the final switch column — that is, within the switch — that is because other connections are using resources within the switch. The addressed function board's port may or may not be busy. Congestion of this nature happens when the software is making extremely heavy, but spread out, use of remote access. It can also arise as a secondary effect to hot spot congestion.
- Once an initial message makes it through the switch, the path is set up and no rejection will occur (except for timeouts or protocol errors, which do not occur in normal operation).

- Reply messages and request messages after the initial one are not rejected (except for errors as noted above).

The rejection is conveyed using the control signal "reverse". This tells the switch hardware in the requester direction to stop sending the message, free up the circuitry it had allocated to the connection, and forward the reject back along the path toward the requester. Thus a reject **tears down** the path, all the way back to the requester SIGA.

When the requester SIGA receives the reject, it waits a while and then sends the message into the switch again. The second attempt may also be rejected, in which case the SIGA waits and retries again. The SIGA will retry the message many times if necessary. The message is still considered an **initial message**, because it is still attempting to establish a connection, no matter which retry is going on.

Each retry is roughly like all others, but differs in these ways:

- The time interval for which the SIGA delays before making a retry will change. This is called **pacing** and is described below.
- Starting when the first attempt is made, and lasting until the connection is set up, a timer is counting. This timer limits how long the SIGA will wait for a successful connection. If the timer expires, an error condition is signalled to the software.
- Periodically, the SIGA is permitted to promote the priority of an initial message it retries. This causes the message to become an **express message**, which contends more strongly for switch resources, assuring that it will soon succeed in establishing a connection. This limits switch latency, and is described further later.

Selection of Pacing Strategy

The hardware implements four strategies for **pacing** the injection of initial messages into the switch. Two "**random**" strategies share the same algorithm and differ only in their parameters; and two "**slotted**" strategies use one algorithm and differ only in the parameters. System software sets the parameters for each strategy to be the same in all SIGAs throughout the machine.

Each T–bus request for a remote access contains information about the nature of the access, such as **read or write** and **locked or not locked**. The SIGA uses this to select one of the four pacing strategies to govern the initial transmission and any retransmissions of the message. (This selection is overridden, and use of the "slotted, number 0" strategy is forced, when the CPU makes an access with the **synchronized access** bit in the Process Control Register on.)

In each of the four strategies, one of its parameters can force the initial transmission to be **immediate** rather than random or slotted. If transmission is immediate, the remaining parameters are ignored for the initial transmission.

On retransmissions, the immediate parameter is ignored, the pacing strategy is either random or slotted, and the remaining parameters are used. The meaning of the remaining parameters is different for a random strategy than for a slotted strategy. Each is described below.

Slotted Strategy

The idea behind the slotted strategy is a sequence of **periodic time slots**. The next time a slot arrives, the message is transmitted or retransmitted. In this strategy, there is no dependence on which transmission is being made — initial, first retry, second retry, or whatever — except for the immediate mode on the initial transmission.

The arrival of the slot is detected by comparing time to a given value. The time used in the comparison is given by the Real Time Clock (RTC) and associated prescaler circuitry in the SIGA. All RTCs in the machine are synchronized.

The slotted strategy has two parameters, the slot **period** and slot **phase**. The slot **period** parameter selects ½, 1, 2 or 4 microseconds as the interval between slots. This governs how strongly the message contends against others for switch resources. The slot **phase** parameter specifies what value the low bits of time have when the slot occurs. This permits staggering of message injection among different SIGAs in the machine.

Two details of the slotted strategy may be of interest to some readers, and may be skipped without loss of continuity.

1. The slot phase can be varied only within the first half microsecond of the slot period. For example, if the slot period is two microseconds, the last 1½ microseconds of it cannot be selected by any SIGA.
2. The number of possible values for the slot phase parameter depends on the switch clock frequency; it is one half the switch clock frequency in megahertz. For example, a 38 megahertz switch clock permits 19 different values of slot phase.

Random Strategy

To understand the details of the random strategy, it helps to keep in mind the general intent. The random strategy implements **a modified version of "binary exponential backoff"** message transmission algorithm. The ***unmodified*** version of this algorithm says that before each (re)transmission of a message, a delay is imposed. The amount of this delay is doubled after each (re)transmission.

For example, suppose you are calling a friend on the telephone. You get a busy signal, so you wait one minute and try again. You get a busy signal again, so you wait twice as long — two minutes — and call again. Still busy! Wait four minutes and try again. Your retries are getting less and less frequent (backing off), at an exponential rate by doubling (base two; binary) each try.

Backoff is a common mechanism in communication networks, where it alleviates congestion. Binary exponential backoff is a frequently used backoff algorithm, because it behaves well and is easy to implement. The TC2000 switch implements a ***modified form of this algorithm***. The modification is to include a **randomization**. Using the telephone analogy, the modification is to wait a *random* amount of time: up to one minute before the first retry, up to two minutes after the first and before the second retry, up to four minutes between the second and third, and so on.

The binary exponential backoff telephone example above has no random component. The random strategy in the TC2000 switch employs randomness by selecting a uniformly distributed, random delay between zero and the strict binary exponential value. The random component helps to **stagger** retransmissions that otherwise might continue to collide with retransmissions of other traffic. Traffic that is coincidentally clumped gets **desynchronized**, so retransmissions from several function boards will seldom collide again and again with each other.

If you time the telephone calls trying to reach your friend the way the TC2000 switch does, then you don't actually double the waiting time after each call. Instead, you wait for a random time, with that doubling time as the upper limit. So after the first try, you wait some time between none and one minute; after the second try, a random time between none and two minutes; next, between no wait and four minutes; and so on. After each try, the chances of waiting any number of seconds, between zero and the current limit, is the same as waiting any other number of seconds. Figure 3–12 shows the telephone analogy graphically.

Figure 3–12 Random strategy – telephone analogy.

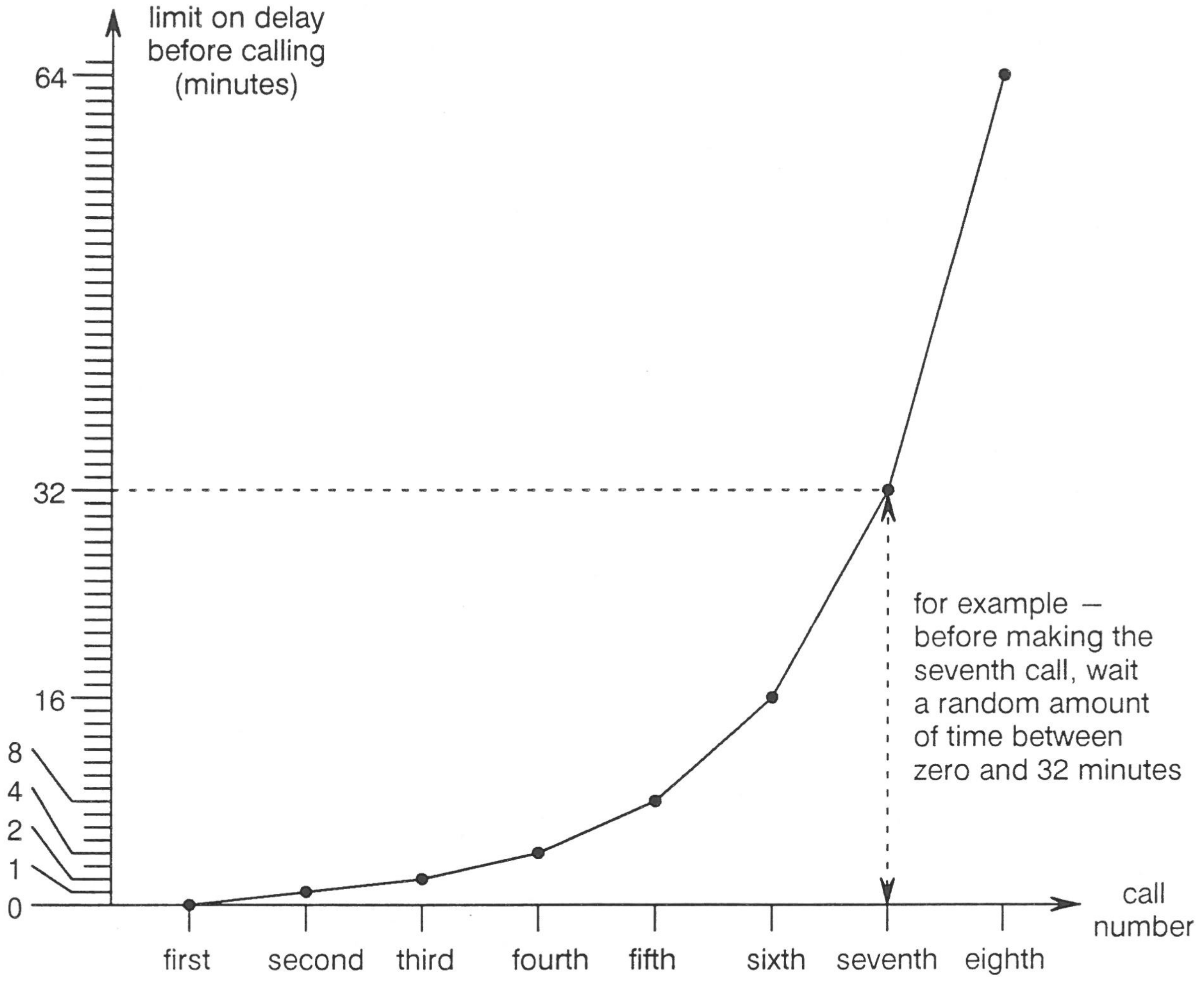
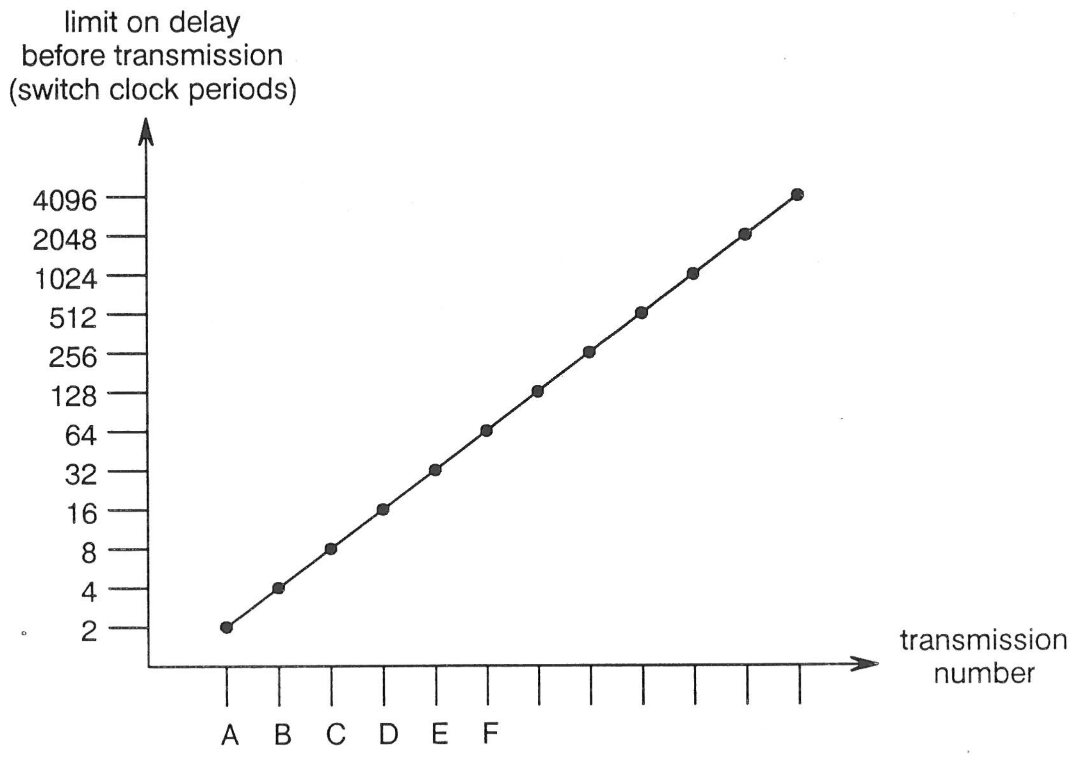


Figure 3–13 shows the TC2000 switch random pacing strategy graphically. To better illustrate the wide range of possible delays, a logarithmic scale is used in the figure. The random strategy algorithm has two parameters, set by system software. The first parameter is where along the "transmission number" axis the first transmission lies. The first transmission can be at any of the six values labeled A through F. Selecting a later value has the effect as if the message had already been rejected and retried some number of times; the exponential backoff still occurs at the same rate, but operation begins further along the curve.

Figure 3–13 **Random pacing strategy.**



The second parameter is how fast the algorithm proceeds along the "transmission number" axis. The system software selects one of four values for this rate. After the message has been rejected eight times, the algorithm will have jumped to successive marks in Figure 3–13 no times, once, twice, or three times, depending on the rate parameter.

Certain subtleties of the random strategy may be of interest, and are listed below. These can safely be skipped without loss to understanding the operation of the machine.

1. If the rate parameter is set to "no times", the delay limit never changes, resulting in no backoff; the delay is randomly chosen from a **constant range**.
2. The **minimum delay limit** is two, so — on the average — the random strategy always has some effect on transmission timing. (Except for the first transmission, if the "immediate" parameter is on as noted above.)
3. The **minimum delay** is actually one, but whether it is one or zero is unimportant for three reasons.
   - A. The unit of delay is a tick of the switch clock, and that is so short that a tick more or less is negligible.

   B. There is a few–tick overhead in getting the message started out of the SIGA anyhow.

   C. The process is (pseudo–) random, so its behavior varies anyhow. A tick more or less is swamped by the variation.

4. The choice of delay from within the current range is essentially **uniformly distributed**, but not exactly so. Its deviation from uniformity is negligible in practice.

5. There are only twelve operational points along the "transmission number" axis, indicating that the actual transmission number (first, second, third...) is mapped onto twelve values used in the algorithm. As the message is repeatedly rejected, operation jumps from point to point. If the message is rejected many times, operation would reach and later jump from the twelfth point back to the point labeled A. That would make the message retries suddenly become rapid, a peculiar but not damaging effect. It is also very unlikely, because after all the delays necessary to jump off the twelfth point, the switch reject timer (discussed later) would probably expire, aborting the message.

Using the Pacing Strategies

The TC2000 software is supplied with default pacing parameters that study and experience have shown provide good overall performance.

The **application** programmer has no control over the selection of pacing strategy or the parameters of the strategies, since these are set up by system software. Awareness of the retry mechanism, however, allows the application programmer to design his program and data structures to avoid severe contention, and to spot and fix congestion when it does appear.

The **system** programmer has complete control over the strategy selection and parameters. Simulation of the TC2000 switch, and experience with the actual hardware, indicates that the programmability provides a useful range of operating behaviors. Production oriented sites may find applications that place unusual stress on the switch, where a modification to the standard default parameters can improve performance. Academically oriented sites may study the results of varying the pacing parameters under experimental loads.

The intent of the pacing strategy design is that all traffic use the random strategies (to gain the statistical benefits of asynchrony), except that references to locations where contention is expected (such as spin locks) should use the slotted strategies (to reduce hot spot contention). The *synchronized access* facility helps make this intent easy to follow, by forcing use of the "slotted, number 0" strategy based on a bit in the Process Configuration register.

3.3.6 Reply Messages – Bidirectional Path

A connection's path through the switch is held open until it is explicitly released (or until an error or timeout occurs, which are abnormal). While the path is held, messages can flow on it in either direction; the TC2000 switch is **bidirectional**. In the simplest case, the data obtained in servicing a read request is returned to the requester over the path. After a write request, a reply is also returned, telling the requester that the write succeeded or, if it failed, what error occurred. After receiving the reply, the requester releases the connection and the switch path is torn down, freeing its switching resources for other messages to use.

3.3.7 Multiple Messages per Connection

Beyond the simple request–reply scenario, the requester may send another request instead of releasing the connection. This second request may be either a read or a write, independently of what the first request was. The server performs the requested action and returns a reply to it. This request–reply interchange may be repeated several times. Finally, the requester releases the connection as in the simple case.

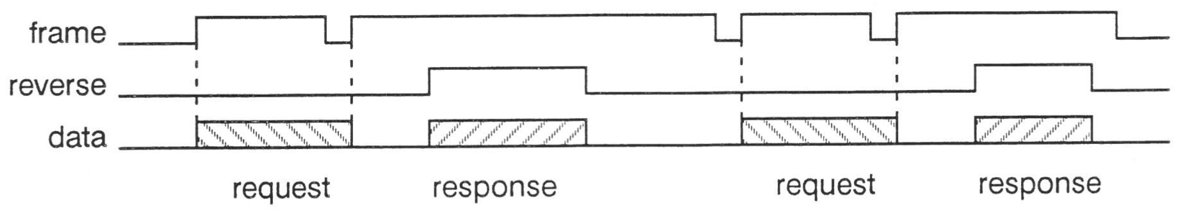
In any use of a switch connection, there is an alternation of requests and replies. Each request elicits a reply, and each reply is the result of an explicit request; the server does not "volunteer" messages.

The programmer causes a connection to be held open by using the **locking** facility. The lock bit in the **Augmentation Register** controls locking. Also, the 88000's **xmem** instruction, which exchanges the contents of a CPU register and a memory location, automatically locks the switch path for the duration of the instruction. Either form of locking achieves **atomicity** of the operations performed while the switch path — and remote memory — are locked. (The **bypass** augmentation is available to explicitly permit an access in spite of locking.)

The connection cannot be held open indefinitely, whether messages are being exchanged on it or not. When an initial message is transmitted, the **Connection Timer** begins running. A reject stops the timer, and it is started again (at zero) when the message is retried. If the timer expires and the connection is still established, the path is torn down and an error is returned to the requester module on the T-bus. The Connection Timer's timeout value is set by system software, and has a range of 1 to 255 microseconds.

The control signals **frame** and **reverse** establish a connection and delimit messages on it. The requester asserts frame during the connection, and releases the connection by de-asserting frame. During the connection, the requester marks the start and end of each request message by de-asserting frame for just one switch clock cycle. Reverse is asserted by the server during each response message. A **reject** is the assertion of the reverse signal for just one switch clock cycle, and originates either at a crossbar unit during path setup, or at the server SIGA if it is unable to accept the connection at this time. Reject occurs only while a connection is being set up, never after it is established. Figure 3–14 shows how frame and reverse function during a locked connection.

Figure 3–14 **Frame and reverse during example connection.**



3.3.8 Error Detection

The switch hardware detects and reports errors of various sorts, as described below. In each case, the software receives a bus error and can examine an error code to determine what error occurred. The errors detected include data corruption, protocol violations and timeouts. In the list below, the errors are grouped according to where along the connection's path they are detected.

1. If the T–bus request violates the protocol for making a remote access, the requester SIGA detects the error and no switch access is made.

   ○ **Maintain present** — the request asks to maintain a locked connection, but currently none exists.

   ○ **Maintain absent** — a locked connection exists, but the request does not ask for a locked transaction (as by opening, maintaining or bypassing one).

   ○ **Lock address violation** — a locked connection exists, and the request asks to access a switch port other than the one to which this connection goes.

   ○ **Miscellaneous CSU error** — the Configuration Status Unit (a part of the SIGA) detected an error in a request either to open a lock, or transfer multiple words.

2. The requester SIGA detects problems with the switch connection.

   ○ **Wait timeout** — the Connection Timer expired while waiting for a reply to a request message that had already been sent.

   ○ **Idle timeout** — the Connection Timer expired while no reply was being awaited; the requester SIGA was idle.

- **Reject abort** — the Interrupts Disabled Too Long / Abort Retries Timer in the CPU interface expired, forcing the SIGA to stop trying to establish a connection. This is described further in the TC/FPV chapter.
- **Reject timeout** — the Reject Timer expired, indicating that the attempt to establish a connection was unsuccessful.
- **Reverse** — the "reverse" signal had incorrect polarity while a response message was arriving. (The correct polarity for all cases is somewhat complicated, and is not discussed here.) This error indicates a hardware problem.
- **Checksum** — the checksum component of a response message was incorrect. This error indicates a hardware problem.

3. Errors detected by the remote (server) SIGA are reported back over the switch path to the local (requester) SIGA and thence to the T-bus module that made the request.

   - **Downstream write** — while the server was driving its T-bus, it detected a write error from the T-bus slave. Because the SIGA was driving the data lines, the slave's actual error code is not available.
   - **Downstream OTL** — the server made a request on its T-bus, but the T-bus slave did not respond. (The slave was Out To Lunch!)
   - **Downstream late** — the remote T-bus slave detected a parity error. (The name "late" is historical, and now means only parity.)
   - **Downstream refused** — the server thought it had established a locked connection, but the T-bus slave responded to its request with a refusal, claiming to be locked (as by another module).

   Note that if the server SIGA receives a request message with a bad checksum, no reply is sent, and therefore no error is sent by the server. Instead, the connection timer is expected to eventually time out, and that will cause an error condition. This behavior is chosen because the bad checksum probably means something is wrong with the switch path, and any reply sent back over it is likely to be garbled and ignored, or possibly misunderstood.

4. Errors signalled normally by the remote T-bus slave are received by the server SIGA, and the error code is conveyed back over the switch path to the requester SIGA and its T-bus to the originator of the request. These errors are not switch specific, and are not described here.

3.4 Features Important to the User

The features described in this section are performed automatically by the TC2000 switch hardware, and their programmable parameters are configured by the system software. The application programmer benefits from them but does not have to do anything to invoke, adjust or service them.

3.4.1 Locking

The need for locking in multiprocessor software is discussed in the section on **atomicity** in chapter 2, and the operation of the TC2000 switch in supporting locked operations is covered in section 3.3.7 above. The advantage to locking is that it makes separate actions, performed together under a lock, atomic. While a single read or write can be performed locked, there is no advantage in doing so, because they are atomic anyway. The locking capability of the TC2000 architecture is particularly powerful because any sequence of reads and writes needed by the application can be performed locked. The restrictions on the use of locking are slight:

- The duration of the switch connection cannot exceed the Connection Timer timeout, discussed in section 3.3.7.
- All locations referenced over a locked switch connection must be on the same remote function board; the switch connection cannot disconnect from one destination switch port and reconnect to another.
- The duration of the locked operation, whether or not it accesses remote (over the switch) resources, cannot exceed the CPU Lock Timer timeout, described in chapter 2.
- Considerations of latency and performance may make it preferable — or necessary — to make certain references **bypassed**. This depends on the application, and is described in chapter 4.

NOTE

The above restrictions indicate that preventing interrupts during a locked transaction is desirable (or writing the handler to not use the switch). A plausible implementation is to ensure that the pages to be accessed are mapped in and resident, and then trap to the operating system kernel (that is, supervisor mode) to perform the actual locked transaction.

For example, suppose a chemical processing system is being simulated by software running on several TC2000 processors in parallel. One of the processors is about to update the yield rate in a particular reaction tank. The calculation requires a consistent set of values for the temperature, pressure, mixture of reactants, and stirring rate. These parameters are stored in the memory of one function board, but not the function board that is about to do the calculation. The CPU makes a locked reference to the remote memory holding the parameters, and reads their values. It now releases the connection and proceeds with the calculation. If the calculation is quick, and the result is to be stored back on the same function board, the connection can be held open and the updated value written back. Writing back the results on the same locked connection ensures that the yield rate is always consistent with the values driving it, so long as the application software accesses them locked.

A second example shows the use of the 88000 **xmem** instruction to perform locking. Sometimes the operations you wish to perform under a lock do not fit the requirements listed above. In the chemical processing example above, the calculation of yield rate may be long and involved, exceeding the time a lock may be held or a connection kept open. Or, the data may be stored in the memories of more than one function board. Here, a standard technique is to designate a memory location as a lock on that part of the data. By convention, the application might use location "tank_lock" to lock all the data about this reaction tank. To update the yield rate, the routine first obtains the lock, then reads the temperature and so forth, computes the result and writes it to memory, and then releases the lock. In the simplest form, the lock location contains a particular value (such as zero) to mean "not locked", and another value (such as one) to mean "locked". A routine with steps such as these does the job:

1. Use the xmem instruction to exchange the value "1" with the contents of location tank_lock. If the result is a "1", some other processor already had the data locked, so try again. Executing a small delay before trying again can help avoid congestion that could arise from contention over access to the lock location. This is analogous to the pacing strategy performed automatically by the switch hardware during connection setup.

   Note that the atomicity of the xmem instruction, as supported by the TC2000 hardware and especially the switch, ensures that no other xmem (or any locked reference) can read the location as containing "0" after we have read it and before our "1" is written there. **This is the heart and the power of the xmem locking facility.**

2. If the result is a "0", the lock was free and you now have it. Read the variables, compute the result and write the result back to memory. During this time, any other processor trying to get the lock will see a "1" in the lock location, so they will not read or modify the tank variables.

3. Having finished the locked portion of the program, release the lock by using the xmem instruction to exchange the value "0" with the contents of location tank_lock. The result you get back from the xmem is not important, but can be checked for consistency (it should be a "1").

You do not have to write assembler language to use **xmem**; the higher level languages supported on the TC2000 computer have constructs to invoke it.

Often, there are more useful values to place in a lock location than just the symbolic "locked" and "free" used in the example above. For example, the identity of the processor or process can be used, and that aids debugging. In general, locking mechanisms on multiprocessors is a rich topic. The interested reader may turn to computer science literature for further information.

3.4.2 Automatic Retry

Because of the shared use of switch resources, connection attempts will sometimes temporarily fail. The hardware automatically remembers what connection is being requested, and retries the connection until it succeeds. This is described in section 3.3.5. The user is therefore freed from handling any of the reject and retry mechanism in the application program, nor is the operating system burdened with any of this activity.

The user does need to be aware of two aspects of remote access. First, the **time required** to make a remote access is longer than for a local access. Usually this is a small cost, but occasionally a remote access may take a noticeable time. This effect and how it is controlled are discussed in section 3.4.3.

Second, **error conditions** can arise during the automatic retry. These errors exist not because of the complex and helpful automatic retry mechanism; they are conditions that can arise in any interprocessor communication design. They indicate that the expected behavior of the hardware did not occur, and therefore the software should know that something is apparently broken. The cause could be a hardware malfunction, or it could be a software problem. Detecting these error conditions and informing the software allows appropriate action to be taken as quickly as possible. Section 3.3.8 describes the errors detected by the switch hardware.

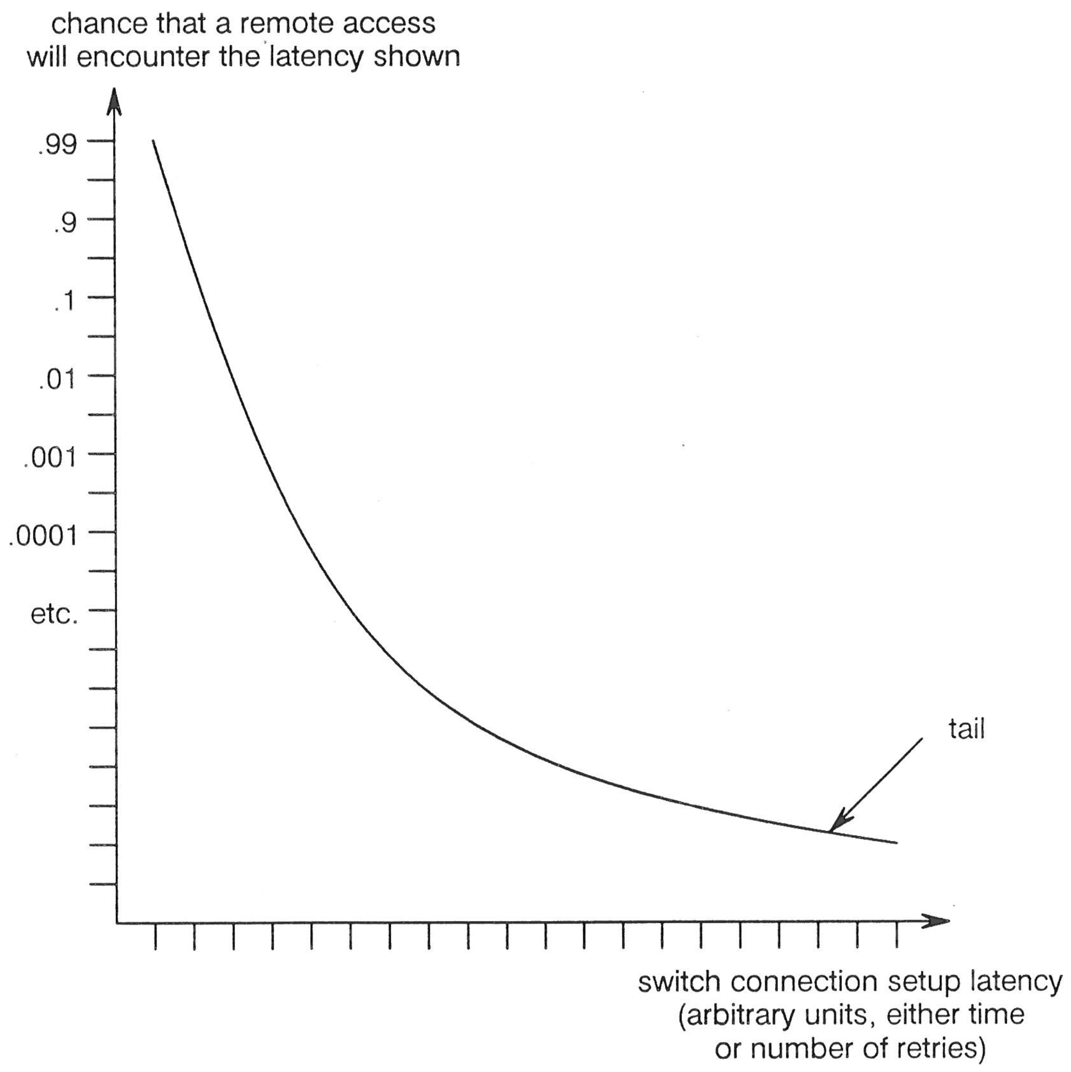
3.4.3 Latency Control

Latency is how long it takes for a requested action to take place. In regard to the TC2000 switch, latency is how long it takes to reference a remote function board. The major component of switch latency is the rejection and retry of the initial message setting up the connection. Other components are relatively small. The one exception to this is latency of accessing the resource on the remote function board; that also is usually negligible compared to connection setup, but could be significant in certain cases. For example, if the reference is to VMEbus devices on the remote function board, the latency of the VMEbus system must be considered. However, because connection setup is normally the dominant component, and because the topic of this chapter is the switch, we will not consider such effects further here.

Without a latency control mechanism, the switch latency would be unbounded. That is, if congestion were extremely heavy, it is possible that occasionally a remote access would be rejected indefinitely, and never get serviced. The level of congestion required to cause this is unlikely to arise except in a poorly structured program or in a test program written specifically to cause it. However, there is a more insidious effect than an access that never gets serviced: accesses that are serviced, but only after a long time.

To understand this effect of long service time, consider how long each of a large number of accesses take to be serviced. Suppose our program makes a billion accesses. Perhaps 900 million of them are serviced immediately; the first transmission of the initial message succeeds in setting up the connection. And suppose that congestion is moderately heavy, and ten percent of the first transmissions are rejected and have to try again. That is, 100 million accesses take at least two tries. Suppose that again 90 percent of these succeed. Ten percent of the 100 million, or 10 million, nevertheless require at least a third try. If the chance of success remains 90 percent on any transmission, we see that probably one of our billion accesses will require ten transmissions before it succeeds. This concept is shown in figure Figure 3–15. The **tail of the curve** illustrates that needing many transmissions is very unlikely, but the chance never actually goes to zero — in the absence of a latency control mechanism.

Figure 3–15 **Switch latency without controls – conceptual.**



And how long does it take to do, say, ten transmissions? That depends on the pacing parameters. Depending on how these are set up, it could take from a few microseconds (but such parameters could exacerbate congestion) to about a millisecond. In some applications, the longer end of this range is not acceptable. Some mechanism is needed to bound switch latency.

The TC2000 switch contains two mechanisms to bound switch latency: **timeouts**, and an **express message** facility. The timeouts are simplest in concept, operation and implementation. They provide a safety net, ensuring that broken hardware or unacceptable latency conditions will not go unnoticed. The express message facility, on the other hand, clips the tail off the latency curve, so that the timeouts should never occur unless something is really wrong. Each is discussed below.

Timeouts

Timeouts release the resources controlled by the timer that expires, and signal a bus error to the process that held those resources. Timeouts mainly benefit other processes than the one that receives the error. Those other processes — whether executing on the same function board or on another — can now use the freed resources for purposes of their own. Without the timeout, those processes would be slowed or stalled whenever they needed those resources.

For example, a process holding a locked connection to memory on a remote function board monopolizes several resources: the memory itself, the switch port to that function board, the switch card switching circuitry allocated to the path that is held open; and the switch interface on its own function board. Each of these resources may be needed by another activity.

There is also a benefit to the process that receives the timeout error. The error tells the process that a problem exists. The process can then make an informed decision to try the action again, perform cleanup activities (specific to the application), log or report the problem, or halt — or any combination of these.

The timeouts related to switch use are the Reject Timer and the Connection Timer. These are described in section 3.3.8. Expiration of the **Reject Timer** occurs when the connection was never established. Expiration of the **Connection Timer** occurs when the connection setup was successful, but the path is being held open too long. In the latter case, different error codes inform the process of the state of the connection when the timeout occurred: either a reply was being awaited, or each request sent so far had received a reply.

The **Interrupts Disabled Too Long / Abort Retries** timeout, also listed in section 3.3.8, is not strictly a switch timeout but does affect switch operation. This timer is part of the CPU interface, and limits how long interrupts may be disabled. This is to control the latency of interrupt servicing (as distinguished from the latency of remote accesses). If this timer expires, it automatically tells the SIGA in the local switch interface to abort any retry. That is, if the SIGA prepares to ***retry*** the transmission of an initial message that has already been ***rejected*** at least once but has not yet succeeded in setting up its connection, the SIGA notices the timeout condition, stops retrying, and signals an error condition.

Express Messages

The express message facility is a mechanism that helps the user by controlling switch latency. It operates automatically and invisibly because it is implemented in the switch hardware. Its parameters are initialized by system software to values in configuration files, and normally never changed after startup. Nevertheless, the facility is important to the user because it removes what would otherwise be a subtle, nagging concern in accessing remote resources, the tail of the latency curve (Figure 3–15).

The **express message** facility promotes the priority of a given message, to the occasional slight detriment of other switch traffic, to ensure that it will penetrate congestion in the switch that otherwise could keep it retrying until timed out.

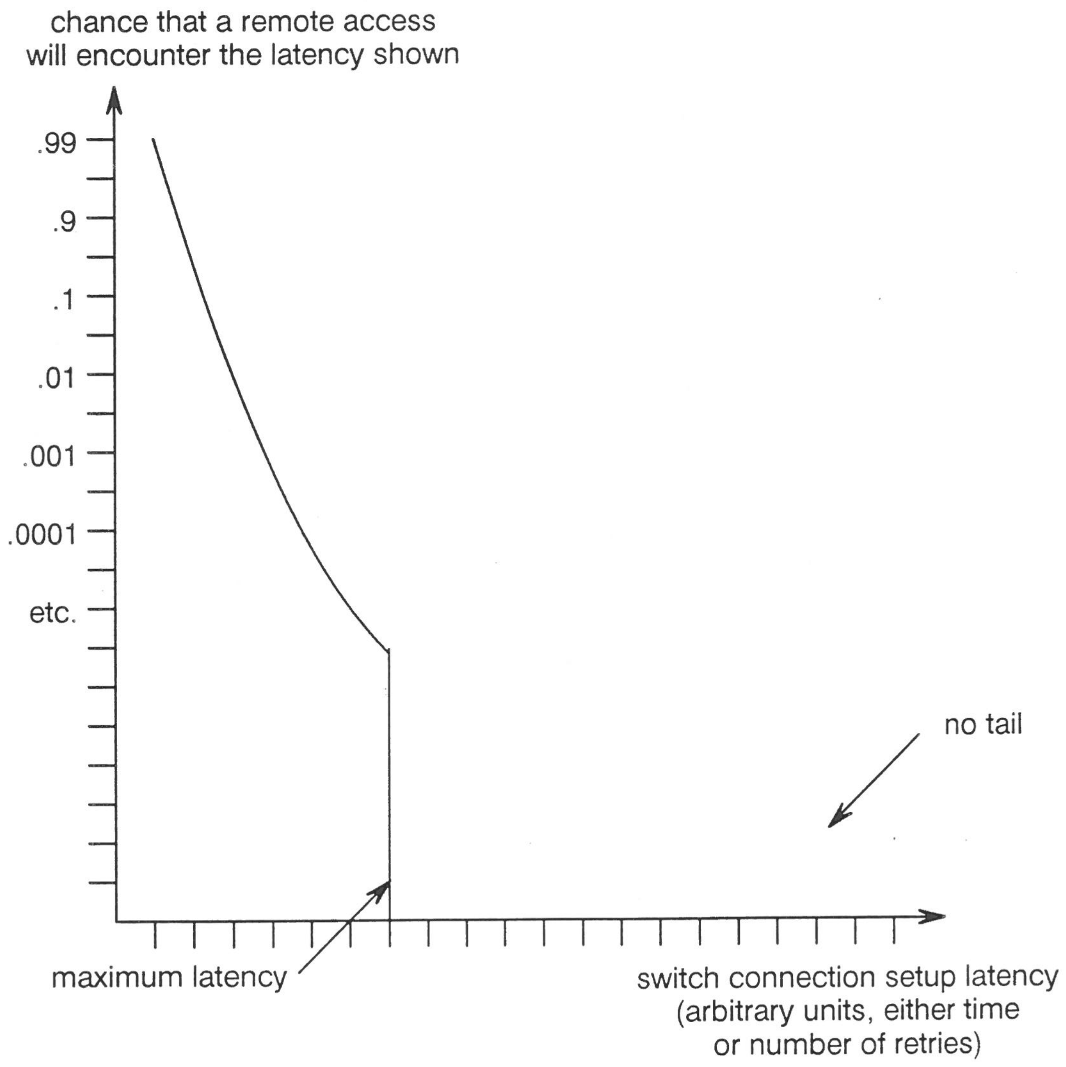
To understand the motivation and benefit of express messages, imagine you can watch all the connection setup activity in the TC2000 machine. Most of the time everything is fine — connections are made quickly. Once in a while, though, a remote access seems to have bad luck. Each time its initial message enters the switch, it is rejected, again and again. "If only," you say to yourself, "that message could be given a privileged status that would let it compete more strongly with other traffic. Giving that status to just any message would be unfair, but this message has already tried so long, it seems fair to give it an advantage." That is exactly what the express message facility does.

The express message facility is, in effect, a virtual token passed among the function boards. When a function board holds the token, any retry it sends into the switch is marked as an **express message**. Other than this marking, the contents of an express message is the same as a normal message. In the switch cards, the express message contends for switch node ports as normal messages do, but in addition, each port remembers that an express message has asked to use the port. We say that the **priority of the port** has been increased from normal to high. In the high priority state, the port will reject normal messages that try to set up connections through it. Only an express message can set up a connection through a port that is at high priority. Therefore, although the express message may get rejected on this try, the port will eventually become free and the express message will succeed on a later retry.

The express message facility places a maximum bound on the switch latency. This bound is the time required for the virtual token to come around to a SIGA, plus the time for the SIGA to retry the message (possibly more than once). The effect is shown in Figure 3–16, which shows the tail of the curve in Figure 3–15 has been cut off. Beyond a particular bound, there is no chance that a message will take longer than that amount of time to establish a connection. (Barring broken hardware, of course, which the Reject Timer catches.)

The numerical value of this bound on latency depends on how the parameters of the express message facility are set up by the system software.

Figure 3–16 Effect of express message facility — conceptual.



There are several subtleties to the express message facility. While we do not discuss the hardware implementation and its use by software in detail here, the interested reader may be left unsatisfied with the conceptual description above, so we summarize the details here. This summary may be skipped without sacrificing understanding of what the facility does for the user.

1. The **virtual token** is merely a concept; no actual token is passed. Rather, the Real Time Clocks, which are synchronized throughout the machine, are used in conjunction with a mask and a value in each SIGA. Using these parameters in its registers, each SIGA detects when its **priority time slot** arrives — that is, when it holds the virtual token.

2. The **mask** and **value** parameters in each SIGA are independent of those in other SIGAs, but the intent is that they be set consistently, to implement a regular progression of the virtual token from function board to function board, round-robin fashion.

3. The mask and value parameters can be set so the time for the virtual token to circulate once is up to 65,536 microseconds. The duration that each SIGA holds the virtual token can be from one microsecond to the full token circulation time. Specifically, the mask and value are 16-bit quantities, and the token is held whenever the low 16 bits of the RTC equals the value parameter in all bit positions that are zero in the mask.

4. The virtual token need not be held by some SIGA at every instant; there can be grace periods between one SIGA's priority time slot and that of the next SIGA.

5. The hardware does not ensure that the switch contains at most one express message at a time. Rather, there are ways to set up the parameters that ensure this, and others that do not. For example, it is possible to circulate more than one virtual token.

6. Only **retries** are promoted from normal to express messages. The first transmission of an initial message is never made an express message.

7. The express message facility concerns the priority of only the **initial message**, the one that sets up a connection. Subsequent messages have no priority value, which is marked in the message header that appears only in the initial message.

8. The intent is that all remote accesses be submitted to the T-bus as **normal priority**. The hardware permits a remote access to be marked "express", but only diagnostic software should use this capability. It causes the resulting initial message (including the first transmission) to be an express message, so use of this by either system or application software would compromise the latency bound for other traffic.

9. To be precise about which retries are promoted: After the header of an initial message has been sent, that message is said to be awaiting retransmission. If there is a message awaiting retransmission at any time during the priority time slot, subsequent retries of that message are promoted. This applies whether the message was already awaiting retransmission when the slot began, or was first transmitted during the slot. It also applies whether the actual retry occurs within the slot or after the slot has ended. Once promoted, the message retains its express priority.

10. A switch node output port is set to high priority by any express message attempting to use it, whether the attempt is granted or not. If the port is already in use, the existing connection is not affected in any way.

11. After becoming high priority, a switch node output port rejects all initial messages of normal priority, until the port is reset to normal priority.

12. A switch node output port drops from high to normal priority by the machine-wide signal **hold** being de-asserted twice (for one switch clock cycle each time), but only if during the first de-assertion the port is idle (no connection, of either priority, exists or is being set up), and between the first and second de-assertions no express message arrives asking to use the port.
13. The **hold** signal is generated by the clock card and distributed to all switch cards. It is periodically de-asserted, with a period programmable by the TCS. The appropriate period depends on SIGA retry rate and other factors.
14. Just as there are two switch clock signals (one for the requester side and one for the server side, either in phase or 180° out of phase), there are two hold signals, each synchronized to one of the switch clock signals.

4 Memory

Every resource in the TC2000 machine has a unique address within a 16–gigabyte global System Physical Address space. This address space is accessible to every CPU and I/O device in the machine. Thus, shared memory is the basic mechanism for communication among function boards in the TC2000 computer. By controlling access to the System Physical Address space in various ways, many different interprocessor communication paradigms can be built on top of this basic mechanism, ranging from strict message passing environments where no data is explicitly shared, to unrestricted sharing of global data.

4.1 Structure

The TC2000 memory is **distributed** among the function boards of the machine. In the original model of the machine, the TC/FPV is the only function board type, and each TC/FPV contains memory. The TC2000 design also permits function boards without memory, and function boards with only memory. The term **memory module** is often used for the memory and associated circuits on one function board. The TC/FPV board is configurable to have either 4 or 16 megabytes of memory.

From the point of hardware, the memory is **shared** among all function boards. Each function board can access memory on any other function board (and of course its own memory). Memory on another function board is called **remote memory** to emphasize that accessing it takes somewhat longer; memory on the CPU's function board is called **local memory** to emphasize the speed advantage in accessing it.

Memory is **paged** with a page size of 8 kilobytes in the nX operating system. This page size is especially convenient because the VMEbus interface windows, in both the master and the slave directions, have a granularity of 8 kilobytes. The nX demand paging is relatively standard, and employs the Motorola 88200 Cache / Memory Management Unit (CMMU) chip. Since the

CMMU's page size is 4 kilobytes, two adjacent CMMU pages are allocated for each nX page.

The pSOS^{+m} operating system never performs demand paging, and the pSOS^{+m} application is affected by page size only for memory allocation, particulary for memory to be shared with other pSOS^{+m} processes. A 4-kilobyte page size is used here, mainly for the simplicity of using the CMMU's page size.

NOTE

PAGE SIZE AND INTERLEAVING
Memory interleaving (described later) has various granularities. The **clump** size, within which all bytes map to the same function board, is 16 bytes. The pattern of mapping clumps to switch ports repeats after each 8-kilobyte boundary, but this is a minor effect. Within any 32-kilobyte **quad-page**, all interleaved accesses map to the same **pool** of function boards. The decision of whether a given access is interleaved or not has a 32-kilobyte (quad-page) granularity in the CPU interface, and an 8-kilobyte (one page) granularity in the VMEbus Slave Mapper. It is intended that the software interleave or not interleave memory with a 32-kilobyte granularity; the VMEbus Slave Mapper granularity is finer because its interleave decision bit is associated with its 8-kilobyte page mapping.

Paging permits the following advantages:

- Allocation of memory in small units (pages), reducing waste
- Access control over individual pages, providing protection and privacy
- Demand paging of program code and data into memory from disk, reducing the amount of memory required to execute programs (Under the nX operating system only; the pSOS^{+m} operating system does not swap pages to or from disk.)

TC2000 memory is **mapped** to achieve flexible allocation of available pages, and to place memory windows conveniently in the address space. Mapping occurs in the following places in the TC2000 architecture:

- References made by the CPU are mapped by one of the 88200 CMMU chips, and by the CPU Mapping RAM.
- References made from a VMEbus system into TC2000 address space are mapped by the VMEbus interface slave mapper.
- References made from the TC2000 address space into a VMEbus system are mapped by the VMEbus interface master mapper.
- References made either by the CPU or from a VMEbus system, that traverse the switch to access a remote function board, can be optionally mapped by the interleaver.

Access to the memory — and in general, the address space — of the TC2000 computer is **protected** by the CMMU. The CMMU provides both write protection and supervisor protection. Its operation is described in the *MC88200 User's Manual.* Briefly, the Processor Status Register in the CPU contains a bit defining the current mode as user or supervisor. This bit, and a bit indicating whether an access is a read or a write, are presented to the CMMU with every access. If the requested location is mapped read–only and this is a write access, or if it is mapped supervisor–only and this is a user mode access, the CMMU rejects the access and signals an error to the CPU. It is the responsibility of the nX operating system to assign the appropriate permissions for each page mapped. The nX system software further protects memory from inappropriate access by checking privileges before mapping in pages or granting other system calls that could compromise protection and privacy.

The memory is addressable in **byte**, **halfword** (16–bit) and **word** (32–bit) units. Halfword and word accesses must be aligned on halfword and word boundaries, respectively; an unaligned access causes a misaligned access exception trap. (Some application execution environments may handle this trap invisibly, or automatically but also tell the user.) Also, when the CMMU loads or flushes a cache line, it makes a **burst read or write** of four consecutive words, aligned on a 4–word boundary. The memory responds to a burst read or write efficiently. (In particular, the T–bus arbitration is performed only once for the burst. Also, if the reference is to remote memory, the data is contained in one switch message.)

The memory supports **read** and **write** operations. Further, the memory supports the TC2000 locking mechanism. A prime example of locking is the CPU's ***xmem*** instruction, which first reads and then writes a memory location while keeping the memory module locked.

Errors in memory are detected with a **parity** bit on each byte. A parity error causes a bus error, with an error code indicating parity error. The machine has a capability to intentionally write the wrong parity into memory, used in diagnostic programs to test the memory and parity logic.

The memory employed is **dynamic RAM**, with refresh provided automatically by the hardware. The impact of refresh cycles on memory latency is negligible, as the concerned reader can see from the details below.

> The memory subsystem performs a refresh cycle once every 12.8 microseconds. A refresh cycle requires five T–bus clock cycles (nominally 250 nanoseconds). During a refresh cycle, all T–bus requests to memory are responded to with "REFUSED", and the requesting module tries again. Refresh requests have the highest priority of any request to the memory. Refresh still occurs when the memory interface is locked.

4.2 Design

This section discusses how the memory structure, described in section 4.1, appears to the program and the programmer. These topics affect primarily the ***operating system*** programmer and the sophisticated pSOS^{+m} user, because the nX operating system handles most of these issues for the nX user. The nX programmer operates mainly in the virtual address space provided by the nX operating system.

4.2.1 Global Address Space

The address space of the TC2000 machine is **global**. This means two things: the address space is **accessible** to all CPUs and VMEbus interfaces, and the address space **uniquely addresses all resources** in the machine. As we shall see in section 4.3, we are talking here about the **System Physical Address** space. The software uses mapping to restrict application programs to appropriate portions of the global address space.

To the user, the importance of the address space being **accessible to all** is that the user is not concerned with which function boards run the program. Any function board is equally capable of executing the code and accessing the data. Of course, I/O devices in a VMEbus system attached to a given function board do make that function board more efficient for running programs that use the devices. And, although code stored on one function board can be executed over the switch by another function board, in practice code is stored locally for speed of access. The considerations of I/O and code placement, however, are much less restrictive than an architecture — unlike the TC2000 — in which parts of the address space are inaccessible from some of the processors.

To the user, the importance of the address space **uniquely addressing all resources** is twofold. The addressing of ***all*** resources means that there is no part of memory or control register or window into VMEbus space that can't be accessed. The programmer does not have to work around peculiar or patchy addressing structure; using the TC2000 address space is straightforward. The ***uniqueness*** of the address space means that there is one, machine–wide System Physical Address for any given location. This frees the programmer from concern about which CPU is making the access, or what mode a memory is in, or what type of operation is being performed; the TC2000 address space does not have such dependencies.

The system programmer receives the direct benefit of the System Physical Address space being global. The application programmer usually deals with virtual, not physical, addresses. But the application programmer does gain substantial indirect benefit from the global quality of the TC2000 address space. In particular, the simplicity of the system software design and operation, and the program execution efficiency and speed, are possible in part because of the global nature of the address space.

4.2.2 Mapping

This section describes aspects of mapping (address translation). Mapping in the CMMU, in the CPU interface, and in the VMEbus interface are discussed.

CMMU

Mapping is performed by the Motorola MC88200 Cache / Memory Management Unit (CMMU) chips. Two or three of these chips are wired to each MC88100 CPU; one CMMU services data references, while the other one or two service instruction (code, text) references. For a thorough description of the CMMU, please refer to Motorola's *MC88200 User's Manual*. Salient features of the CMMU include:

- Memory management unit features:
  - Two 4–gigabyte address spaces (one for user mode, one for supervisor mode)
  - Access protection (of supervisor memory from user mode access)
  - Write protection
  - Maintenance of "used" and "modified" page flags
  - Probe capability (for testing the status of a virtual address without actually referencing the location)
  - Page (4 kilobytes) — TC2000 software allocates two adjacent CMMU pages to make each system page
- Cache features (per chip)
  - Separate cache chips for data and instructions
  - 16–kilobyte, 4–way, set-associative physical cache
    - There are **16 kilobytes** of storage for cached data/instructions
    - The unit cached (one **line**) is four 32–bit words
    - **Four** lines with the same offset in their pages can be cached simultaneously
    - Each of 256 **sets of four cache lines** is associated with an offset from the base of their pages; address bits 11..4 select the set
    - Entries are organized by **physical** address
  - Concurrent address translation and cache access, gaining speed
  - Writethrough / copyback with area (4 gigabytes), segment (4 megabytes) or page (4 kilobytes, paired by system) granularity
  - Cache inhibit with area, segment, page or block (512 kilobytes) granularity

    - Cache flush and invalidate initiated selectively by software or automatically by hardware

- Multiprocessor support: locking during the **xmem** instruction is passed on from the CPU, through the CMMU, to the rest of the CPU interface

Largely, the memory management and caching performed by the CMMU is invisible to the application programmer. The nX operating system in particular handles all details of the CMMU operation. Programmers using the pSOS^{+m} operating system have greater ability to modify hardware parameters, although direct access to the CMMU is discouraged.

Protecting Access to Registers

Under the nX operating system, memory management is used to limit access to configuration and control registers in the TC/FPV, some on an individual basis. The prime example of this is the **Interprocessor Interrupt register**. Since the register resides on a page of its own, the nX system can control access to it simply by controlling the access protection of the virtual page(s) mapping the register. More explicitly, each TC/FPV contains an Interprocessor Interrupt register, so the nX software can permit or prevent interrupting remote processors by controlling the application program's mapping of those registers. (However, the implementation of nX does not permit direct user access to any of the CPU interface registers.)

Mapping References Bypassed

The CPU interface translates the Physical Address produced by the CMMU into the System Physical Address placed on the T–bus. As part of this translation, the CPU interface also produces certain auxiliary control signals that describe the access. One of these is the **bypass** signal. When the CPU interface determines the access is to a part of memory that is mapped "bypassed", it asserts the bypass signal, which in turn ***suppresses the TC2000 locking protocol***. Normally, the Augmentation Register (AR) determines whether an instruction will be performed under the TC2000 locking protocol, described in section 4.4.1 and in chapters 2 and 3. Accessing certain locations and certain kinds of data with locking would be undesirable. The hardware automatically suppresses locking on all instruction fetches, and on all page (but not segment) descriptor fetches made by a CMMU. In addition, the system software maps certain data references, especially those associated with exception processing, in bypassed address space. For further details, see chapter 2.

While we have concentrated here on bypassed accesses made by the CPU, the VMEbus slave also generates bypassed accesses, under control of a bit in each of its mapping registers.

Memory Mapping under the nX Operating System

The nX application programmer deals almost exclusively with the virtual address space provided by the nX system. The following nX system calls allow the user to manipulate that space, indirectly affecting the mapping to physical memory.

getphysaddr	return the physical address corresponding to a virtual address if its page is resident in memory
getmmuinfo	return the physical address and memory management unit (MMU) bits corresponding to a virtual address
vm_allocate	allocate virtual memory
vm_allocate_and_bind	allocate virtual memory on a specific function board
vm_cache_flush	invalidate cached data or write it out to memory
vm_cache_setup	specify caching attributes for pages of virtual memory
vm_deallocate	release access to an area of virtual memory
vm_inherit	specify how to pass pages of virtual memory to child processes
vm_mapmem	allocate virtual memory, or map a file for shared access
vm_mapstat **vm_mapstat_pid**	get status of process's virtual memory
vm_protect	change protection of virtual memory pages
vm_statistics	obtain status of system's use of virtual memory
vm_sync	write modified virtual memory pages to buffer cache or to disk (use only with memory allocated by vm_mapmem)
vm_transfer	copy virtual memory between processes

VMEbus Mapping

The VMEbus interface contains two address mapping RAMs. When a VMEbus device accesses a location in the window into TC2000 address space, the VMEbus Slave Map RAM translates the VMEbus address into an TC2000 address. When an access within the TC2000 machine falls within the interface's window into VMEbus address space, the VMEbus Master Map RAM translates the TC2000 System Physical Address into a VMEbus address.

The point above about **bypassed access** bears further discussion in the context of the VMEbus interface. For example, accesses from VMEbus devices may have more stringent latency requirements than the TC2000 computer needs, and mapping accesses from the VMEbus into TC2000 memory as bypassed can accommodate such requirements of the VMEbus system.

Interleaver Mapping

The interleaver optionally maps addresses from the T-bus used by the SIGA requester in accessing remote locations. Its use is described further in section 4.5. Interleaving is not supported in the current release of the nX operating system.

4.2.3 Demand Paging

The nX operating system, supported by the TC2000 hardware, performs demand paging. This is invisible to the process, except for delay while the page is made available. A description of the process is beyond the scope of this document, but the user should be aware of the following points.

- Under the nX operating system, when a user program references a remote data page, that page is copied into local memory. (Text pages are always allocated in local memory only.)
- The nX operating system has proper memory sharing semantics for creating child processes, so ***vfork* is identical to *fork***. Programs that rely on the temporary *vfork* implementation on other systems, where execution of the parent was suspended until the child exited or executed an *execve*, will not work under nX. See the nX *fork(2)* manual page for further details.
- Under the nX operating system, a page is "cleaned" (old contents paged out to disk if necessary) only when a user needs it; the system does not do predictive page-outs.
- The nX operating system pre-zeros free pages in the idle process, so that when a new page is needed, it is often available immediately and does not have to be cleared while the user process waits.
- The nX operating system will retrieve a page from the disk buffer cache if it still resides there, avoiding the time penalty of reading the page in from disk.

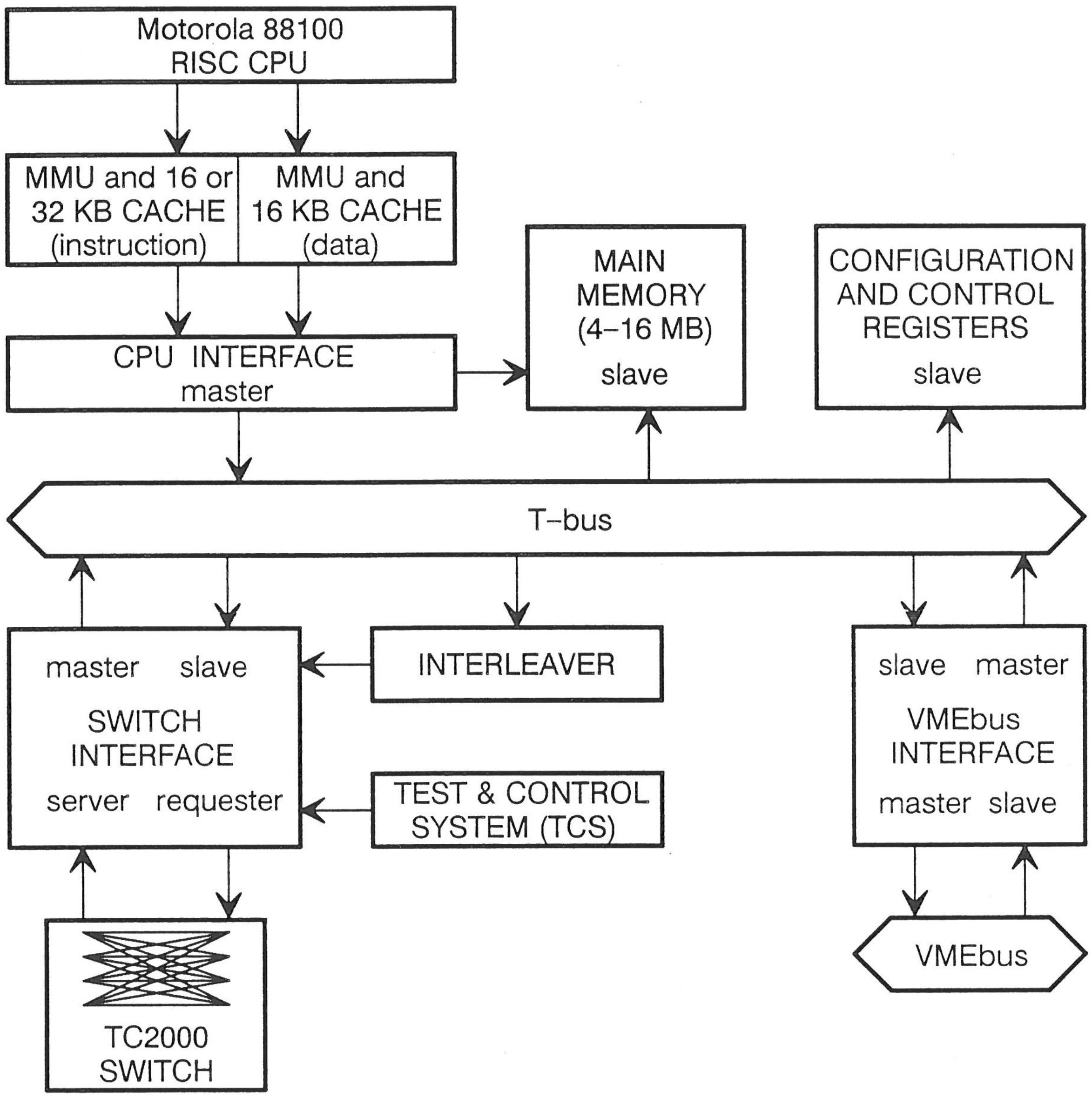
4.2.4 Interleaving

Interleaving is not supported in the current release of the nX operating system. However, it is a capability of the TC2000 hardware. The TC2000 interleaving mechanism is described in section 4.5.

4.3 Addressing

Figure 4–1 shows components of the TC2000 machine that generate, transform or respond to addresses. The figure uses the TC/FPV as an example. The principal originator of addresses is the CPU. VMEbus master devices may also generate addresses sent into the TC2000 machine, and the Test and Control System generates addresses as part of its control and monitoring activity. The component that responds to the most addresses is the memory. Registers, except those internal to the CPU, are accessed by their address. VMEbus slave devices may also occupy addresses and respond to TC2000 requests.

Figure 4–1 **Address flow.**



Addresses are transformed in five places. Addresses generated by a CPU are first mapped by either its instruction CMMU or its data CMMU, depending on the type of reference the CPU is making. Then the address is transformed by the CPU Mapping RAM in the CPU interface, and placed on the T–bus or sent via the fast path to local memory. An address to which the requester SIGA responds may be taken directly from the T–bus, or part of it may be transformed by the Interleaver. The other two address transformations occur in the VMEbus interface, where addresses on the T–bus are mapped to addresses on the VMEbus, and vice versa.

The remainder of this chapter discusses how the CPU accesses memory, and how the memory system responds to accesses from any source.

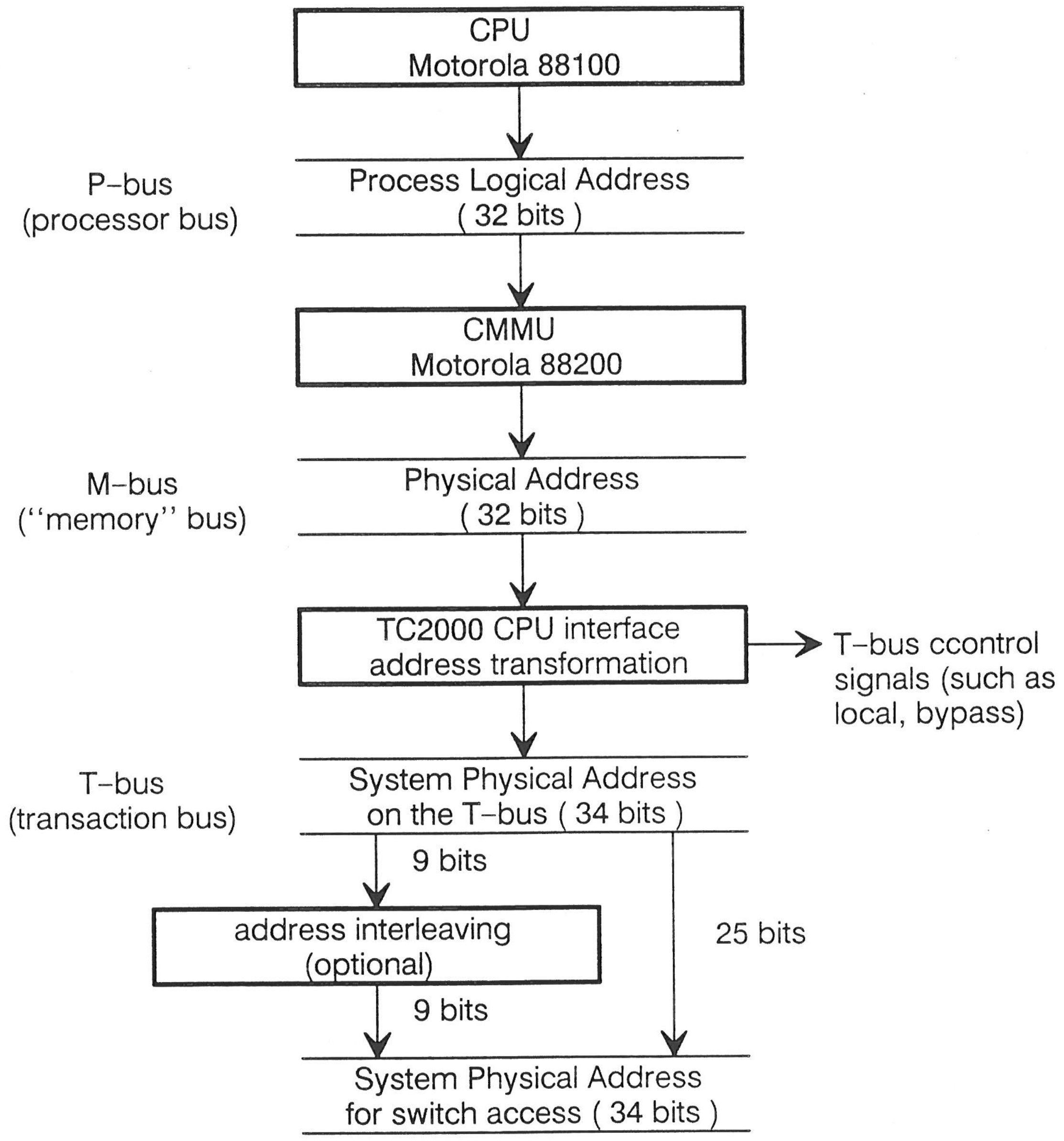
4.3.1 Address Formats

The path from CPU to memory uses three address formats:

- Process Logical Address
- Physical Address
- System Physical Address

Figure 4–2 summarizes the path of address transformation for every address generated by the CPU.

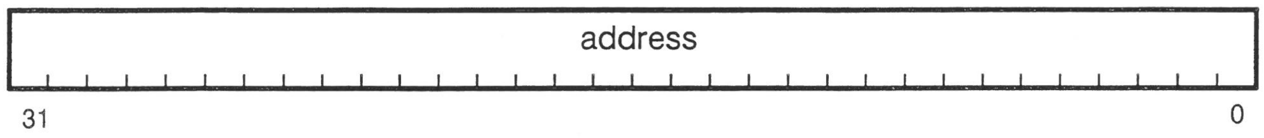
Figure 4–2 **Addressing from CPU to T–bus and switch.**



The Motorola MC88100 CPU operates in the virtual address space of 32–bit ***Process Logical Addresses***. Examples of a Process Logical Address are an address stored in a CPU register, or a pointer in the C language. The user's program sees only Process Logical Addresses, and hence a 32-bit virtual address space. The nX operating system's use of the page translation mechanisms in the CMMU hides many of the details of the other two address formats and address translation from the application–level programmer.

The Process Logical Address is a 32-bit field with no defined internal structure. See the Motorola *MC88100 User's Manual* for further discussion of the CPU operation and its use of addresses.

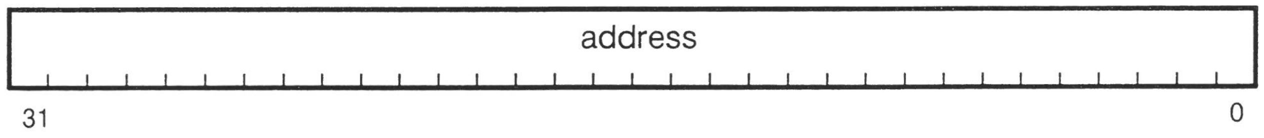
Figure 4–3 **Process Logical Address format.**



The 32-bit Process Logical Address generated by the CPU is transformed by a Motorola MC88200 CMMU into a 32-bit ***Physical Address***. One CMMU transforms data addresses, and instruction addresses are transformed by another CMMU (either a single one, or one of a pair). For more information on the Physical Address and operation of the CMMU, see the Motorola *MC88200 User's Manual.*

The CMMU places little constraint on how the Physical Address it produces is interpreted. The CMMU is designed so the Physical Address could be presented directly to a memory system. The Physical Address (Figure 4–4) is 32 bits without any field definitions, because it is transformed by the CPU Mapping RAM.

Figure 4–4 **Physical Address format.**

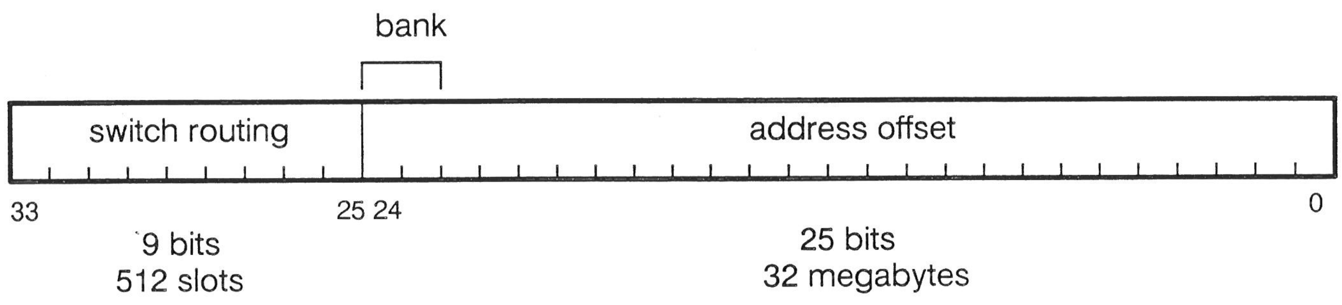


The ***System Physical Address*** is unique to the TC2000 architecture, not a part of the Motorola CPU or CMMU. It is 34 bits. The CPU interface transforms the 32-bit Physical Address into the 34-bit System Physical Address, and in the process also produces other address-related signals. The System Physical Address from the CPU interface is placed on the T-bus. It is important not to confuse the Physical Address with the *System* Physical Address.

- The **Physical Address** exists only going from a CMMU to the attached CPU interface. "Physical Address" is a term employed by Motorola in their CMMU literature. There it is sometimes called the M-bus address, because the CMMU was designed to drive a memory bus. It is 32 bits.
- The **System Physical Address** exists on the T-bus and in the switch. It is the common language by which all T-bus master devices address all T-bus slave devices. "System Physical Address" is a term defined by TC2000 designers. It is 34 bits.

Figure 4–5 shows the System Physical Address format and its fields. The switch routing field specifies the path through the switch, and therefore one of 512 slots. The address offset field addresses four 8–megabyte banks of memory, for a total of 32 megabytes addressable per switch port. The address space of the System Physical Address is 512 × 32 megabytes, or 16 gigabytes. The top two bits of the address offset are a subfield called the bank bits.

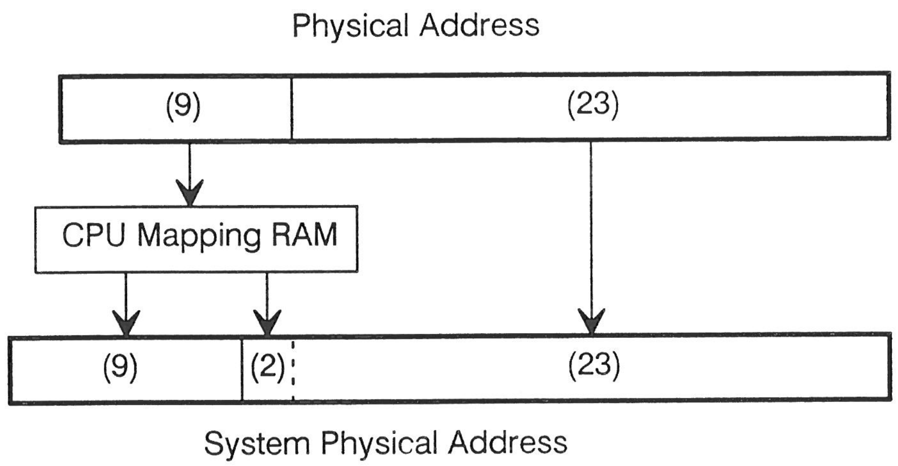
Figure 4–5 **System Physical Address format.**



4.3.2 Address Translation

The CPU interface receives a Physical Address from one of the CMMUs, and produces a System Physical Address that is placed on the T-bus. Figure 4–6 shows this transformation.

Figure 4–6 **Physical Address to System Physical Address.**



Besides the transformation of Physical Address to System Physical Address, the CPU interface indicates whether the access is local or remote, whether it bypasses locks, whether it may access interleaved memory, whether it may use the fast path to local memory, and whether it is intercepted without actually accessing any resource. These are conveyed on separate signals and are not strictly a part of the address.

4.3.3 Banks

Banks are the four 8–megabyte sections of the 32–megabyte address space at each switch port (and therefore on each function board). The concept of banks is necessary because the 88100 CPU and the 88200 CMMU deal only with 32-bit addresses, while the TC2000 architecture uses 34–bit System Physical Addresses composed of 9 bits of switch port and 25 bits of address space at each switch port. Going from 32 bits to 34 bits requires the insertion of two bits. The design inserts these as the top two bits of the address space at each switch port, thus identifying four 8–megabyte banks there.

Banks, being an aspect of physical addressing, are invisible to the nX application programmer, whose code executes in virtual address space.

4.4 Features Important to the User

This section covers features of the TC2000 memory system that are of particular interest or use to the user. The previous sections present the memory more from a design and hardware operation viewpoint, while this section is geared more to the programmer.

4.4.1 Locking

The TC2000 locking protocol is supported by the memory hardware. The processor support for TC2000 locking (and the overall concept of locking) is described in chapter 2, the switch support in chapter 3, and the memory support here. The memory's interface to the T–bus implements the locking mechanism, so the entire memory on the function board is locked at once, and unlocked at once.

When a T–bus reference to memory specifies locking, and the memory interface is not already locked, the memory interface performs the reference and remembers that the memory is locked by the module making the reference. That module may be the CPU interface, the VMEbus slave interface, or the switch interface (the server side). The switch interface makes references only on behalf of a remote CPU or VMEbus slave, not on its own. During a locked transaction from a remote function board, the memory is held locked because the first memory access message on the connection locked the memory, not because the switch path is held open. (Holding the switch path does, however, prevent any other remote function board from accessing any locations on this function board, for the duration of the locked connection.)

While the memory is locked, only the following accesses are accepted:

- Accesses from the same module that has the lock
- Accesses from the CPU interface via the fast path
- Bypassed accesses, including all instruction fetches

All other references to memory are refused, and are retried by the requesting module.

The memory is unlocked when the module holding it locked releases it. In the CPU interface, this happens automatically at the end of an **xmem** instruction, and explicitly by clearing the lock bit of the AR at the end of a sequence locked by using that bit. The memory interface itself contains no timeout on being locked; the design depends on the requesting module timing out a locked condition. The CPU interface has a lock timer, and the VMEbus has an implicit timer described in the following detail note.

> The VMEbus slave holds a lock only as long as the VMEbus signal "address strobe" is asserted. The longest this signal is normally asserted is during a read–modify–write operation, which is fast. If the signal were asserted too long, such as by broken hardware in a VMEbus device, the VMEbus system bus timer (optionally the one in the TC/FPV) will expire, aborting the operation.

4.4.2 Error Detection

Each byte is protected by its own parity bit. When an access (of any size) is made that includes a byte with a parity error, the hardware responds to the requesting module with a bus error. If the access was made by the CPU, the Bus Error Vector register in the CPU interface can be read to determine that the error was specifically a parity error. The return of a bus error occurs whether the access is local or remote; in the latter case, the switch conveys the error code back to the requesting function board.

The CPU interface can be set to write incorrect parity into memory. This is used only by test and diagnostic programs, never in normal operation. Nevertheless, it is indirectly helpful to the user because it increases confidence that the machine is detecting any errors that occur.

Detected Soft Error Rate

The estimated, detected soft error rate, is about once in three years per four megabytes of memory. (This uses a chip error rate of 1000 errors per billion hours of operation, an appropriate figure for the high (90%) confidence level we use in design.) This is sufficient for the original model. Later models, especially ones with more than 64 function boards, may employ error correction circuitry on their memory. There is no difficulty accommodating error correction in the TC2000 architecture.

Undetected Soft Error Rate

Using the above conservative, design chip error rate, and other plausible assumptions about memory use, the undetected soft error rate is less often than once in a billion years per four megabytes of memory. This is negligible compared to other sources of error, including human error.

Hard Errors

After a soft error, the machine will reboot and all hardware can still be used. After a hard error, the memory subsystem with the error must be replaced or configured out of the system. The hard error rate is of importance in large machines such as the TC2000 computer, but of less importance here than in many other machines because of the ability to configure around failed function boards. Experience with TC2000 machines indicates that the hard error rate is low.

4.4.3 Synchronized Access to Memory

NOTE

NOT IN CURRENT nX OPERATING SYSTEM
User access to the synchronized access feature is not supported in the current nX release. The reader may skip this section without sacrifice in understanding of the features available to the nX user. The nX software does, however, use synchronized access for kernel xmem instructions and locked sequences.

The topic of this section is implemented in the CPU interface and the switch interface, and concerns congestion at a server switch port, but is discussed here in the memory chapter because of its impact is on the user's access to memory.

The **synchronized access** facility is a mechanism to help reduce contention for **access to remote memory**. When many CPUs attempt to access the memory module on one function board at a high rate, the switch port to that function board is often busy. This contention results in frequent switch message rejects and automatic retries, congestion in the switch, and slowed execution of programs. The memory module contended for is called a **hot spot**. The contention may be over a single location (such as a lock or other shared variable), a particular area of memory (such as a shared page containing heavily referenced variables), or a large portion of the memory on that function board (such as a large hash table). Accesses from VMEbus interfaces can also contribute to making a hot spot, but CPU accesses are typically the cause.

Whatever the contended resource in the memory module (locations, pages, or a large area), all remote references must pass through the one switch port, making the switch port and T-bus and memory module interface a bottleneck through which the accesses are serialized. Several approaches are available for alleviating hot spots; one is the synchronized access mechanism. To see this, consider a "**spin lock**". A spin lock is the heavily referenced, shared data that processes repeatedly test, as rapidly as possible, until it takes on some awaited value. A spin lock is one of the simplest forms of synchronization among processes, and therefore is often used. While "spinning" on the lock, the processes heavily load the switch port to the memory module containing the lock, creating a hot spot. Under such heavy contention, the CPU on the same function board may also be slowed.

Hot spots can arise either through straightforward heavy loading or through fluctuation into a region of instability. As an analogy, consider the flow of logs down a river. If the logs are a foot in diameter, and the river is 100 feet wide, it can hold at most 100 logs side by side. Suppose a huge log holder dumps 200 logs into the river all at once; a jam is nearly inevitable. On the other hand, consider a smooth flow of 20 or 30 logs side by side, smoothly flowing down the river. Suppose there is some disturbance — a log catches on a rock, for instance. This disturbs the flow, and some logs turn sideways across the river. Other logs catch on these, and a jam builds up. Similarly, a hot spot can arise in smoothly operating communications if the load is heavy and a pileup is triggered by some event — such as momentary heavy traffic through a switch node. This logjam effect can build a "warm spot" into a hot spot. And once the contention is heavy, the hot spot may stay hot until operating conditions change. Another reason that a spin lock hot spot stays hot is that the process holding the lock and now ready to free the lock, also encounters heavy contention, needlessly delaying the freeing of the lock.

The TC2000 software environment provides profiling and diagnostic tools the user may employ to detect and diagnose hot spot behavior (the tool ***gist*** is an example), and programming techniques to reduce or avoid it. Sometimes, even the switch transmit and receive lights on function boards will show the user an unintended switch traffic volume to a particular board or between particular boards.

The **synchronized access** facility operates by forcing the switch interface to use a particular one of the four possible pacing strategies explained in the switch chapter. (In particular, the "slotted, number 0" strategy is used.) The facility is enabled by a bit in the Process Control register in the CPU interface. When the facility is enabled, any remote access the CPU makes will use that one pacing strategy. The parameters of that strategy may be initialized by system software to pace switch transmissions appropriately for references that are expected to encounter (and therefore add to) contention. In this application, the delay before initial transmission can be quite important, not just the pacing of retries. Slightly delaying every access to a spin lock, for instance, reduces or avoids the logjam behavior. This frees the programmer from putting explicit pacing in his program. It is appropriate whenever the program accesses shared data that is likely to be referenced concurrently by several processors.

CAUTION

A TREATMENT, NOT A CURE
Software backoff is ***much*** more important in reducing **spin lock** load than the synchronized access facility is. Using synchronized access makes spinning "not as bad", but it still isn't a good programming practice when several processors may be contending for the lock.

In the current release, the nX operating system uses the synchronized access feature for kernel xmem instructions and locked sequences. It is possible that it may be used in the future in additional ways:

- The operating system can use it for its own references (besides xmem and locked sequences) to remote, shared data for which contention is likely
- The operating system can grant explicit or implicit requests from application code, surrounding references to heavily referenced, shared, remote data
- The operating system can permit the application process to enable and disable this facility itself, providing the above functionality but without the overhead of a system call

The pSOS^{+m} user is not prevented from using the synchronized access facility, since the pSOS^{+m} user may enter kernel (supervisor) mode.

4.4.4 Optimizing CMMU Use

The nX philosophy is that memory management, caching and demand paging are invisible to the application program and generally support the user's needs well. For most applications this is entirely true. Occasionally, however, an application has particular needs that can be better met by taking some cognizance of how these operations work in the hardware, and how the nX software employs them.

For example, a program might be analyzing the different physical shapes (conformations) that a drug molecule can assume, looking for shapes that minimize total energy (because those are most likely). Such a program might compute intensely, with a relatively small number of data pages. If the data resides in the cache, the calculation will proceed significantly faster than if each reference is to main memory.

There are different optimizations to consider. Some apply to one application, others to another. The following discussions provide a bridge for the programmer to cross the gap between the raw hardware design and the purely software descriptions of system calls. The analysis of which optimizations could benefit a given application is left to the programmer. The topics are:

- Distributing data structures
- Localizing data references
- Reducing page fault rate
- Wiring down data pages
- Reducing page table walks
- Increasing cache hit rate

The topics chosen for discussion below apply especially to the TC2000 machine. Other, more general optimization techniques are described adequately in computer science literature and are not covered here. (That includes compiler options for optimizing space or run time, restructuring algorithms for parallel execution, reducing subroutine calling, coding critical routines in assembly language, trading off memory use and execution speed, avoiding the packing and unpacking of bit fields, and so on.)

Distributing Data Structures

If a data structure frequently referenced by several processors resides entirely on one function board, then a **hot spot** can develop involving the switch path to that function board, the switch server interface there, the T–bus, and the memory module. This phenomenon is described in section 4.4.3. Besides modifying the access strategy to that one function board (as the synchronized access mechanism described there does), the data structure may be spread out over several function boards. This can dramatically improve the program performance. The **Uniform System library**, for example, provides the **scatter matrix** facility for distributing the rows or columns of a matrix among a group of function boards, for applications using the Uniform System. For further information on **scatter matrix**, please refer to Uniform System documentation. Distributing parts of a data structure that is conceptually one whole is an idea that any application program may implement in its own way, even if not using the Uniform System.

Localizing Data References

Access over the TC2000 switch to data stored on a remote function board is slower than access to data stored locally. When the cost of remote access limits program performance, performance improves when data resides in the memory of the function board that most often references it. Another technique to localize data references is to make a local copy of data that will be heavily referenced. The local copy may be only temporary, if the heavy reference is only temporary. For example, a program that multiplies matrices might copy into local memory the row or column that is heavily used in each execution of the inner loop.

Reducing Page Fault Rate

When the nX operating system removes a page from the working set of a process, and the process later refers to a location in that page, the reference is suspended while the page is made available. If this page–out activity is prevented, program performance improves.

A simple way to reduce page–out is to reduce the demand on virtual memory by restructuring the process and/or its data. The following programming practices work toward this end.

- Use compact data structures; don't waste space
- Consider running fewer processes, or processes that use less memory, on the affected function boards
- If memory on an affected function board is allocated to processes running on other function boards, redirect this allocation

One can imagine a process knowing that soon it will reference a page that is not currently resident, and asking the nX system to page it into memory while the process continues execution instead of being suspended. This is contrary to the nX design philosophy, and no direct way to do this is provided. An indirect means toward this end is to have another process (such as a child that shares the given page with its parent) reference the page. This causes the nX software to bring the page into memory, so that when the parent references it, the delay will be substantially smaller. This is rather contrived, and is more useful as a thought exercise than as a general technique in practice.

Wiring Down Data Pages

The nX operating system provides a means to insist that specified data pages remain in memory. Data pages can be **wired down** to prevent the nX system from paging them out. Application software can tell the nX system to wire down pages of virtual memory by using the **vm_mapmem** system call. In the current release of the nX operating system, text pages of user space cannot be wired down.

Wiring down pages is more drastic than the good, general programming practices that simply reduce demand for memory. Wiring down pages has the power to cripple the operation of the nX system, and should be used only when necessary and only with care. To wire down pages, user group "wheel" access privilege is required.

Reducing Page Table Walks

Even if the nX system removes no pages from the process's working set, performance may be reduced by another aspect of memory management — page table walks. Compared to page–out, page table walking is usually a minor effect. To the programmer needing a small improvement in performance, however, the effect could make the difference.

The memory management implemented by the CMMU includes on–chip Address Translation Caches (ATCs). If the ATCs do not contain the information needed to translate a virtual address, the CMMU automatically fetches that information from memory and places it in the ATCs. Therefore, the programmer needs to understand the use of ATCs in order to know what restructuring of the program would improve performance. Below are the basics of this use.

Each CMMU contains two ATCs: the Block Address Translation Cache (**BATC**) and the Page Address Translation Cache (**PATC**). There are ten entries in the BATC, each translating a 512–kilobyte block. Eight are programmable, and two are hard–wired to map control memory (the top megabyte of supervisor space). The eight programmable BATC entries are maintained by software. However, the nX operating system allocates user memory on the basis of pages, and has no provision for allocating such a large (half megabyte, aligned on a half–megabyte boundary) block to users. The nX system uses all the BATC entries for system purposes; ***no user process address space is mapped through the BATC***.

The PATC, on the other hand, is maintained entirely by the CMMU. Its 56 entries are dynamically shared between supervisor and user page mapping. The PATC contains the 56 most recently used page translation entries. Therefore, an application process may use all 56 entries. However, after the operating system (including interrupt routines) runs, some or all of those entries will be replaced, and page table walks will occur as they are needed. Our interest here is to minimize page table walks while our process is running — there is nothing we can do in the application program to keep PATC entries from other processes when they need them.

Each PATC entry maps one 4–kilobyte CMMU page (under the nX system, pages are 8 kilobytes, two adjacent CMMU pages). So the 56 PATC entries map $56 \times 4 = 224$ kilobytes. Therefore, as long as the process references at most 224 kilobytes, it will suffer no additional page table walks. Those 224 kilobytes need not be contiguous; they may be in any 56 different 4–kilobyte CMMU pages (each 4–kilobyte aligned).

Note that the above discussion applies ***independently to each CMMU*** — the text being executed may fall in up to 56 different 4–kilobyte CMMU pages, and the data may reside in up to 56 different 4–kilobyte CMMU pages. (Under the nX system, text and data are never stored in the same page.) If the function board is configured with two code CMMUs, executed text may occupy up to 112 different 4–kilobyte CMMU pages without causing more page table walks.

Increasing Cache Hit Rate

As with page table walks, some cache misses are unavoidable, but the rate can be affected by the structure of the program and its data.

The CMMU's "data cache" (to distinguish it from the Address Translation Caches), like the PATC, is managed entirely by the CMMU. Its entries cannot be sequestered by the operating system or by user processes.

The cache in each CMMU is 16 kilobytes, organized as described in section 4.2.2. To incur no cache misses beyond those unavoidably resulting from execution of nX system software and other user processes, our process must restrict its accesses to 16 kilobytes. This area may be 16 kilobytes of contiguous memory, but other distributions are also possible. Specifically, the 16 kilobytes referenced have to contain at most four 4–word lines at each offset within their pages. For details, please refer to the *MC88200 User's Manual*, or to the summary in section 4.2.2.

Thus, it may help to pack code and data tightly, so each fits into 16 kilobytes. (Or into 32 kilobytes of code, if the function board has two code CMMUs.) The nX system and C and Fortran provide limited means to control the contiguity of text routines or data structures. Some techniques that can help are:

- Routines that are combined into a single routine occupy contiguous locations. (However, if routines are called from more than one place, putting them in–line requires multiple copies of them. This will expand the overall code size, working against a high cache hit rate.)
- Routines that appear next to each other in the source, and are in the same module/file, are more likely to be next to each other in the executable image than routines that are separated in the source. ✻
- Data structures that are combined into one structure, or stored in one array, occupy contiguous locations.
- Data structures that appear next to each other in the source, and are in the same module/file, are more likely to be next to each other in the executable image than data structures that are separate in the source. ✻

CAUTION

The techniques marked ✻ are heuristic only and are not guaranteed.

NOTE

MAPPING DATA NON-CACHEABLE
Occasionally, an application may benefit from inhibiting the caching of data structures referenced in certain ways. For example, a large hash table or bit array, referenced often but at random addresses, will place cache entries in the CMMU that are rarely used again before they are dropped from the cache, possibly displacing entries that will be used again soon. Another example is rapid references to single locations on successive pages of a large array. In such situations, the pages holding the data structure can be marked **non-cacheable** by using the **vm_cache_setup** nX system call. This can reduce the number of cache misses. (In contrast, vm_cache_setup is typically used to mark shared data ***cacheable***.)

There is no analogous way to inhibit the page table walks in such situations, but page table walking is less of a problem because the PATC replaces the least recently used of its ***56*** entries. The cache replaces the least recently used of the ***four*** lines that match the within-page offset of the reference.

4.4.5 Interleaving

Interleaving is not supported in the current release of the nX operating system. When and if it is supported, it could aid the user by reducing the contention for data stored in a given memory module and accessed by several function boards. The TC2000 interleaving mechanism is described in section 4.5.

4.5 Interleaving

The discussion of interleaving begins with introductory discussion that the advanced reader may wish to skip.

Please note that the current release of the nX operating system does not support interleaving. The following discussion describes the general concept of interleaving, how the TC2000 interleaving hardware works, and how it could be used by software.

4.5.1 Overview of Interleaving

Interleaving is a form of address mapping. A range of addresses that the CPU, and therefore the program, sees as contiguous is mapped into several sub-ranges that lie in separate memory subsystems. The first several addresses go to one memory subsystem, the next several addresses to a different memory subsystem, the next several to yet another, and so on. Often there are more sub-ranges than there are interleaved memory subsystems, so the mapping eventually cycles back through the memory subsystems and maps many sub-ranges to each memory subsystem.

4.5.2 Motivation for Interleaving

Why Interleave At All?

The programmer does not see any direct effect of interleaving. The purpose of interleaving is to improve efficiency, thus speeding computation. This improvement depends on the pattern of memory references being somewhat sequential, a hypothesis which is very well met by the instruction fetches of almost all programs, and is usually moderately well met by their data fetches as well. This is called *locality of memory access*. Programs often refer to elements of a data structure that resides in contiguous locations of virtual address space. For instance, indexing through array elements is a common operation.

As a counterexample, a program segment that fits entirely in the cache, making memory references at random (such as to a hash table), derives no direct benefit from interleaving. In a multiprocessor environment such as the TC2000 machine, however, even this program segment will probably execute faster due to the *indirect* benefit of reduced switch and memory contention and congestion.

Historically, an early use of interleaving was to improve the effective speed of memory. Large memories, such as banks of magnetic core memory, were relatively slow compared to the CPU speed achievable. Interleaving allowed the next memory reference to begin while the previous reference, to a different memory subsystem, was completing. This application used fine grain interleaving; typically, each successive word address was mapped to a different memory subsystem. This was often implemented by taking some of the low bits of the virtual address and moving them to a high position in the physical address.

Why Interleave This Way?

The TC2000 interleaving is motivated by other concerns and is implemented in a different way. Several processes may be accessing data structures that are stored near each other in a shared, virtual address space. Without interleaving, these data structures would usually reside in nearby physical locations, usually in the same memory subsystem. When the memory subsystem is servicing an access request from one process, other access requests to it will be delayed. The memory subsystem becomes a bottleneck. Also, switch paths converging on the memory subsystem's server port will become heavily loaded as access requests are rejected and retried.

The crux of this bottleneck is that several popular data structures all lie in one memory subsystem. The solution is to spread out the data among several memory subsystems. The access pattern is then spread out, and the performance approximates average behavior instead of "hot spot" behavior.

The amount of data mapped to each memory subsystem in turn — the interleaving granularity — need bear no relation to the actual size of the data structures the processes are referencing. The TC2000 interleaver maps each clump of 16 bytes to a different switch port, and therefore to a different memory subsystem. This scatters the data across many memory subsystems.

The 16–byte clump size is well matched to the maximum switch message data size, also 16 bytes. If a single switch message could access more than the interleaving clump size, then such an access to interleaved memory would necessarily refer to more than one switch port. Sixteen bytes is also the size of a cache line, so filling or flushing a line accesses only one switch port.

In the historical interleaving, it was common for the entire virtual address space to be interleaved. On the TC2000 machine, interleaving is performed on a *quad–page* basis. Each quad–page (32 kilobytes) may be entirely interleaved or entirely non–interleaved. The quad–page–based interleaving permits flexibility in the use of interleaving. TC2000 interleaving is more like memory mapping than the historical interleaving, both in implementation and use.

In the TC2000 machine, some memory subsystems may be busy with other tasks. Some may be physically not installed, or be configured out of service. The number of memory subsystems may be different than when the system was previously brought up, and may often not be a power of two. All of these variations are accommodated well by the quad–page–based, mapping style of interleaving, and would be very difficult to accommodate with interleaving that relied simply on a shuffling of bits within the address.

4.5.3 Uniform Use of TC2000 Interleaving

The TC2000 interleaving design permits considerable flexibility in use. Each switch interface has associated with it its own copy of the interleaving hardware, and each T–bus master has its own capacity for deciding whether a given access is to an interleaved page.

The hardware was designed with machine–wide uniformity of interleaving in mind. The idea is that ***all interleavers will be set up identically***. Also, that all hardware that decides whether a given page number is interleaved or non–interleaved will be set up identically. We suggest that the reader approach TC2000 interleaving with this model in mind, and that the programmer consider using interleaving this way to avoid complexities in coding and debugging.

Various deviations from this uniform model are possible. Some would produce very peculiar effects and should rarely if ever be used, while others may be valuable in certain obscure applications. An example of very strange use of interleaving is to access a given physical page sometimes as interleaved and sometimes as non–interleaved. This has the effect of reordering and/or hiding portions of the data. While one might imagine a sorting or hash–coding algorithm based on this "use" of interleaving, it's more of a misuse.

A second example illustrates a potentially realistic deviation from uniform interleaving. In an application where the TC2000 is serving several users, each user is probably allocated a number of processors for the user's exclusive use. The operating system might restrict the user's access to memory by preventing access to physical pages outside the user's cluster of processors. If the machine is thus partitioned, the operating system could set up a given quad–page as interleaved within that cluster, but non–interleaved in the rest of the machine.

Again, we suggest that the standard use of TC2000 interleaving is uniform. If a given quad–page is interleaved for any access, we suggest it be interleaved for all accesses in the machine.

4.5.4 Implementation of TC2000 Interleaving

Overview as a "Black Box"

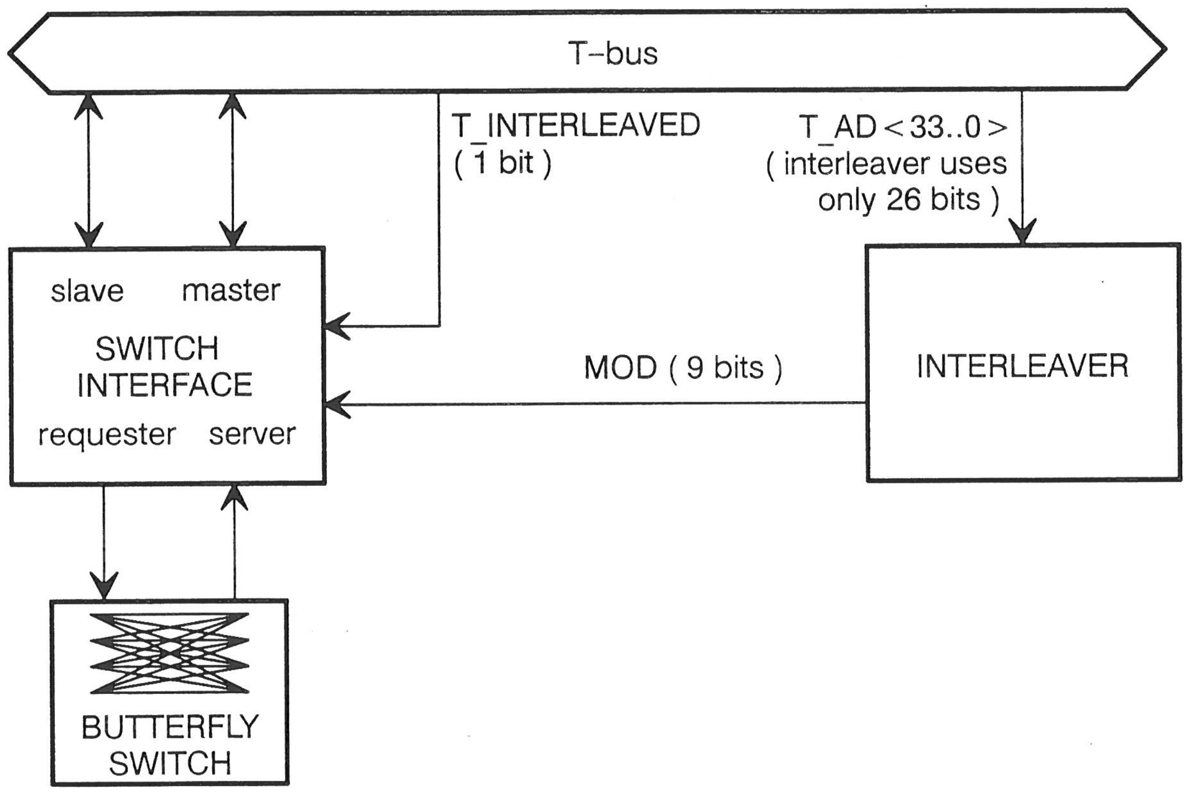
The interleaver input is most of the address bits on the T–bus, therefore a 34–bit System Physical Address. The interleaver output is nine bits, all of which are inputs to the SIGA. For the switch route portion of the address, the SIGA uses either T–bus bits T_AD < 33..25 > or the nine bits from the interleaver. A 1–bit control signal associated with the T–bus, T_INTERLEAVED, tells the SIGA which source to use.

Figure 4–7 shows how the interleaver fits into the TC2000 design. Its only action is to change the switch routing used by the SIGA, based only on the T–bus address. Therefore, the following points are true.

- Interleaving applies to remote (over the switch) references only. Local references are not interleaved. References to memory on the same function board may be interleaved only if they access that memory over the switch.
- The effect of interleaving is to change the switch port to which an access is directed. The 25 bits of address offset are unchanged; the access will have the same address offset at the new switch port as it would have had at the other switch port without interleaving.
- T–bus masters that originate accesses over the switch, such as the CPU interface and the VMEbus interface in the TC/FPV, should drive the T_INTERLEAVED signal consistently. Unusual — and probably unde-

sired — results would ensue if one T–bus master accessed a quad–page as interleaved, and another T–bus master accessed the same quad–page as non–interleaved.

Figure 4–7 **Overview of the interleaver.**



Internal Workings of the Interleaver

The interleaver contains two RAMs and an adder. Its organization is shown in Figure 4–8. T_AD<33..25> and T_AD<12..4> provide the two 9–bit inputs to an adder. T_AD<33..28> supply the high six address bits, and T_AD<22..15> the low eight address bits, to the pool RAM. The address supplied to the pool RAM selects one of 16,384 3–bit locations. The three bits from the selected location drive the high three address bits of the modulus RAM, and the low ten bits of its address come from the adder. This selects one of 8,192 9–bit locations in the modulus RAM. The nine bits from that location are the output of the interleaver. They go to the SIGA.

Figure 4–8 **Interleaver internal processing.**

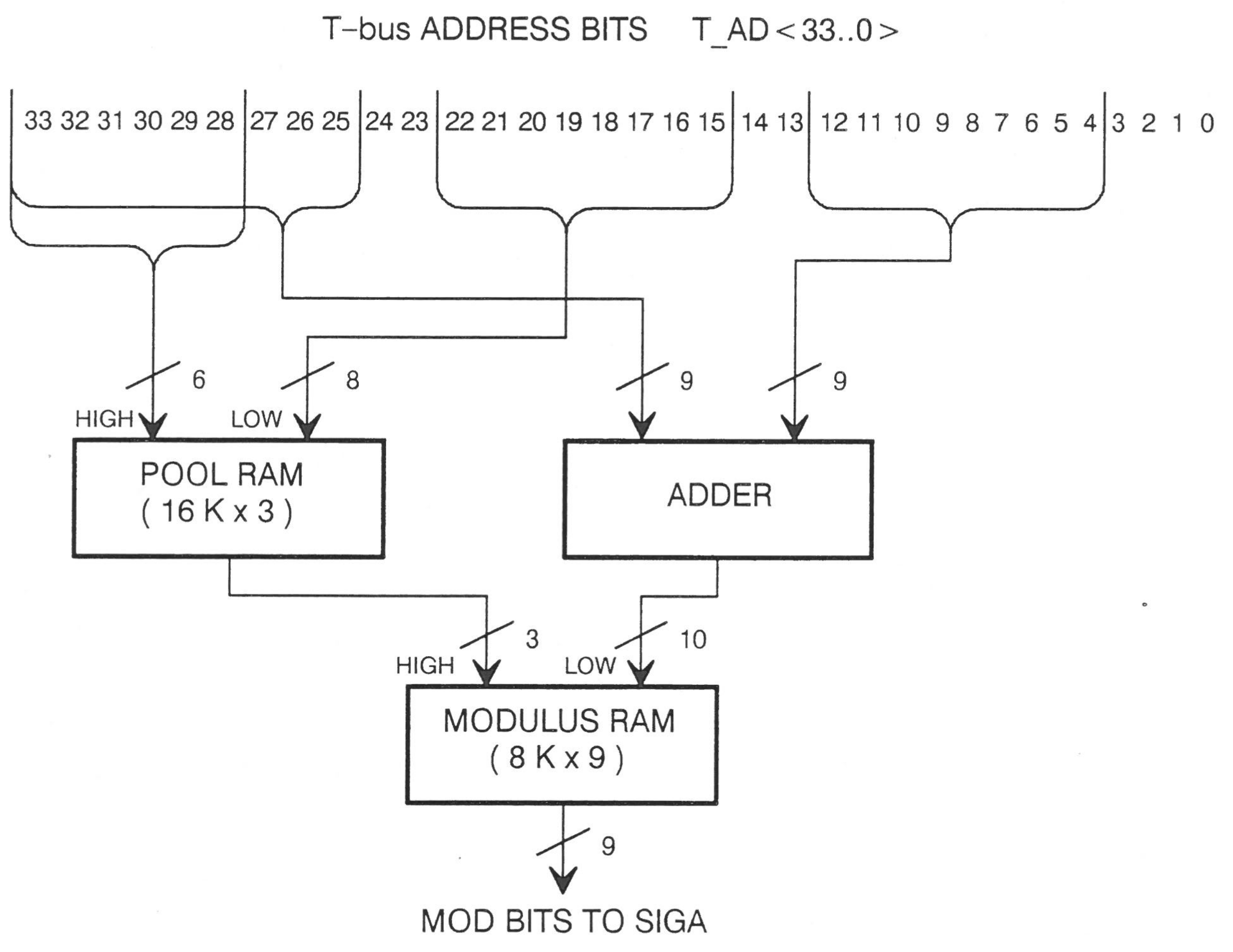
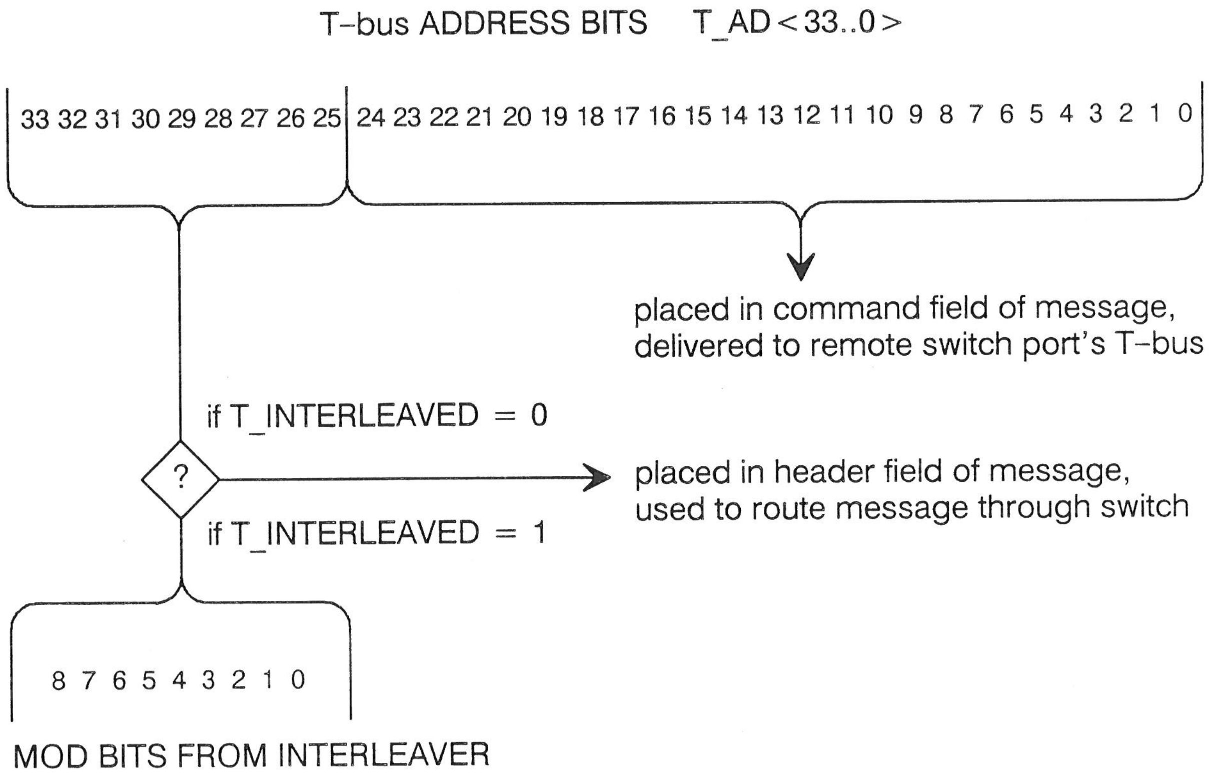


Figure 4–9 illustrates how the SIGA uses the T–bus address bits, the MOD bits from the interleaver, and the T_INTERLEAVED bit.

Figure 4–9 **Interleaving processing in the SIGA.**


T–bus ADDRESS BITS T_AD<33..0>
33 32 31 30 29 28 27 26 25
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
placed in command field of message, delivered to remote switch port's T–bus
if T_INTERLEAVED = 0
?
placed in header field of message, used to route message through switch
if T_INTERLEAVED = 1
8 7 6 5 4 3 2 1 0
MOD BITS FROM INTERLEAVER

The CPU interface derives T_INTERLEAVED from its Interleave Decision RAM, gated with the CMR *interleave enable* bit; unless both RAMs say to interleave, T_INTERLEAVED is not asserted.

The VMEbus interface derives the T_INTERLEAVED bit from its selected VMEbus Slave Map RAM register. The intended use is that software will set up this RAM such that T_INTERLEAVED is asserted consistently with the CPU interface's use, and that the individual page–by–page control available in the VMEbus Slave Map RAM be used to support quad–page interleaving.

4.5.5 Conceptual Operation of the Interleaver

The interleaver hardware was designed with a particular model of interleaving in mind, although the design is relatively general and could be used in various ways. This section describes the intended use of the interleaver and motivates the hardware design. We examine each task the interleaver performs, identifying the hardware that implements that task.

Mapping Clumps to Switch Ports

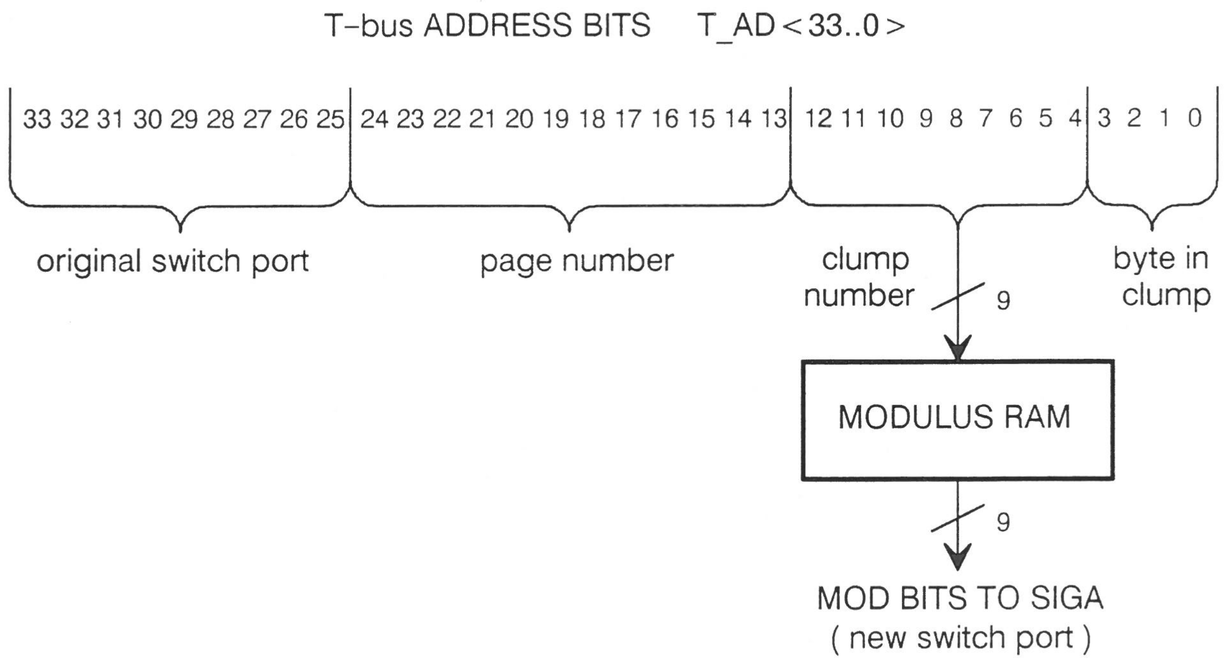
The primary task of the interleaver is to distribute different clumps of bytes within a page (or quad–page; see note below) to different memory subsystems. The switch port identifies the memory subsystem, so the interleaver maps each successive clump to a different switch port.

The interleaver operates on the System Physical Address. For all addresses within a page (eight kilobytes), the high 21 bits of the System Physical Address are the same. Therefore, at this most basic level, the high 21 bits are not relevant to the interleaving operation. Only the bottom 13 bits change, so only they can contribute to the interleaver's mapping of address to new switch port.

Further, all 16 bytes of each clump should be mapped to the same memory subsystem, so the low 4 bits of the System Physical Address do not contribute to the interleaving. *Only System Physical Address bits 12..4 drive the* ***basic*** *interleaver mapping.* These bits identify the clump within the given page, and therefore are called the *clump number.*

The basic task of mapping the clump number to a new switch port is performed by the modulus RAM. The clump number supplies the address to this RAM, and the contents in the selected location are the new switch port where that clump of addresses in the interleaved page resides. Figure 4–10 shows this basic task.

Figure 4–10 Mapping clumps to switch ports.



The modulus RAM could be programmed to map each of the 512 clumps in the page to any "random" switch port. Indeed, this flexibility is important, because only certain of the possible 512 switch ports may be available when interleaving is set up, typically during system initialization. The modulus RAM is intended for use in a more structured way, however. When interleaving is set up, the software should decide which switch ports are available to supply clumps of memory in the interleaved page. These switch ports constitute a *pool* of switch ports. The modulus RAM assigns clumps to each of the switch ports in turn, round robin fashion, recycling through the pool as many times as necessary.

Usually there will be fewer than 512 switch ports in the pool; suppose there are six. The first six clumps are mapped to different switch ports. The seventh clump is mapped to the same switch port as the first clump. The eighth clump is mapped to the same as the second clump. And so on. Figure 4–11 shows this use of the modulus RAM. In this example, the modulus RAM computes the function, port = clump modulo 6. Hence the name, *modulus* RAM.

Figure 4–11 **Modulus RAM use – example 1.**

```
clump 0 ⇒ port 0     clump 6 ⇒ port 0      clump 12 ⇒ port 0     ...
clump 1 ⇒ port 1     clump 7 ⇒ port 1      clump 13 ⇒ port 1     ...
clump 2 ⇒ port 2     clump 8 ⇒ port 2      clump 14 ⇒ port 2     ...
clump 3 ⇒ port 3     clump 9 ⇒ port 3      clump 15 ⇒ port 3     ...
clump 4 ⇒ port 4     clump 10 ⇒ port 4     clump 16 ⇒ port 4     ...
clump 5 ⇒ port 5     clump 11 ⇒ port 5     clump 17 ⇒ port 5     etc.
```

In fact, the switch port numbers produced by the modulus RAM need not be the numbers 0 through 5. Suppose that the switch ports with available memory were every third port starting with port 10: 10, 13, 16, 19 and so on. Figure 4–12 shows how the modulus RAM would be set up.

Figure 4–12 **Modulus RAM use – example 2.**

```
clump 0 ⇒ port 10    clump 6 ⇒ port 10     clump 12 ⇒ port 10    ...
clump 1 ⇒ port 13    clump 7 ⇒ port 13     clump 13 ⇒ port 13    ...
clump 2 ⇒ port 16    clump 8 ⇒ port 16     clump 14 ⇒ port 16    ...
clump 3 ⇒ port 19    clump 9 ⇒ port 19     clump 15 ⇒ port 19    ...
clump 4 ⇒ port 22    clump 10 ⇒ port 22    clump 16 ⇒ port 22    ...
clump 5 ⇒ port 25    clump 11 ⇒ port 25    clump 17 ⇒ port 25    etc.
```

The available switch ports need not be in any particular order. Figure 4–13 shows the same switch ports as in Figure 4–12, but in a scrambled order. This will work fine. In other words, any permutation of the switch ports in the pool may be used in the round-robin assignment of clumps to ports.

Figure 4–13 Modulus RAM use – example 3.

clump 0 ⇒ port 19	clump 6 ⇒ port 19	clump 12 ⇒ port 19	...
clump 1 ⇒ port 25	clump 7 ⇒ port 25	clump 13 ⇒ port 25	...
clump 2 ⇒ port 16	clump 8 ⇒ port 16	clump 14 ⇒ port 16	...
clump 3 ⇒ port 13	clump 9 ⇒ port 13	clump 15 ⇒ port 13	...
clump 4 ⇒ port 22	clump 10 ⇒ port 22	clump 16 ⇒ port 22	...
clump 5 ⇒ port 10	clump 11 ⇒ port 10	clump 17 ⇒ port 10	etc.

In the above examples we have seen considerable flexibility in use of the modulus RAM. What is important is that *all interleavers that will access the interleaved page must use the same mapping*. The software may restrict this access to certain processors, such as one cluster of processors allocated to a particular user. This would mean the interleavers for other processors would not have to contain the same mapping for this page. On the other hand, the software implementation may take the far less complex route of mandating, by convention, that all interleavers in the entire machine hold the same mapping.

NOTE

PAGES AND QUAD–PAGES
The above discussion of mapping clumps to switch ports describes 8–kilobyte ***pages*** of interleaved memory, for two reasons: simplicity, and because the modulus RAM is in fact unaffected by System Physical Address bits 14..13. However, in practice, software would use interleaving based on 32–kilobyte ***quad–pages***, because the granularity is 32 kilobytes in both the Interleave Decision RAM (in the CPU interface) and the pool RAM (in the interleaver). The effect of using quad–pages but not having System Physical Address bits 14..13 affect the modulus RAM, is that *the mapping of clumps to switch ports is exactly repeated for each page in the quad–page.*

Efficient Use of Interleaving

Figure 4–14 shows the same mapping as Figure 4–12, but in graphic form. Each "A" indicates that the clump number shown to the left is in the memory subsystem at the switch port shown below. The bottom "A" says clump 0 is mapped to port 10, the next "A" maps clump 1 to port 13, etc.

This diagram pertains to one interleaved quad–page at each of the six ports. Interleaving this quad–page has no effect on any other pages in the machine. The software, however, may implement a convention that the same-numbered quad–page at every port also be interleaved.

Figure 4–14 **Modulus RAM use – one stripe.**

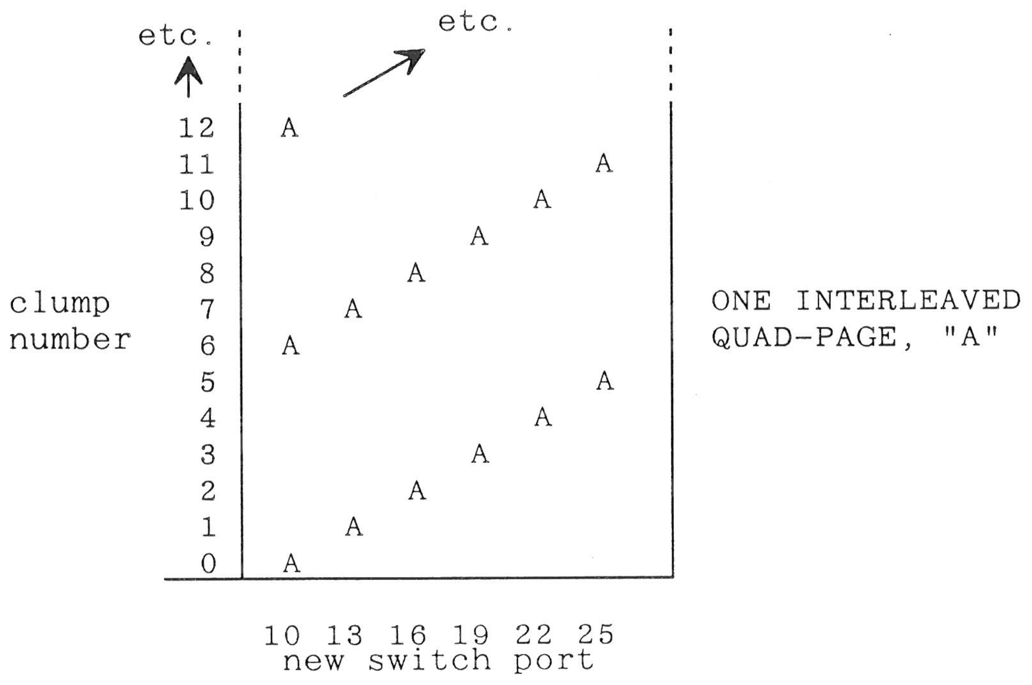
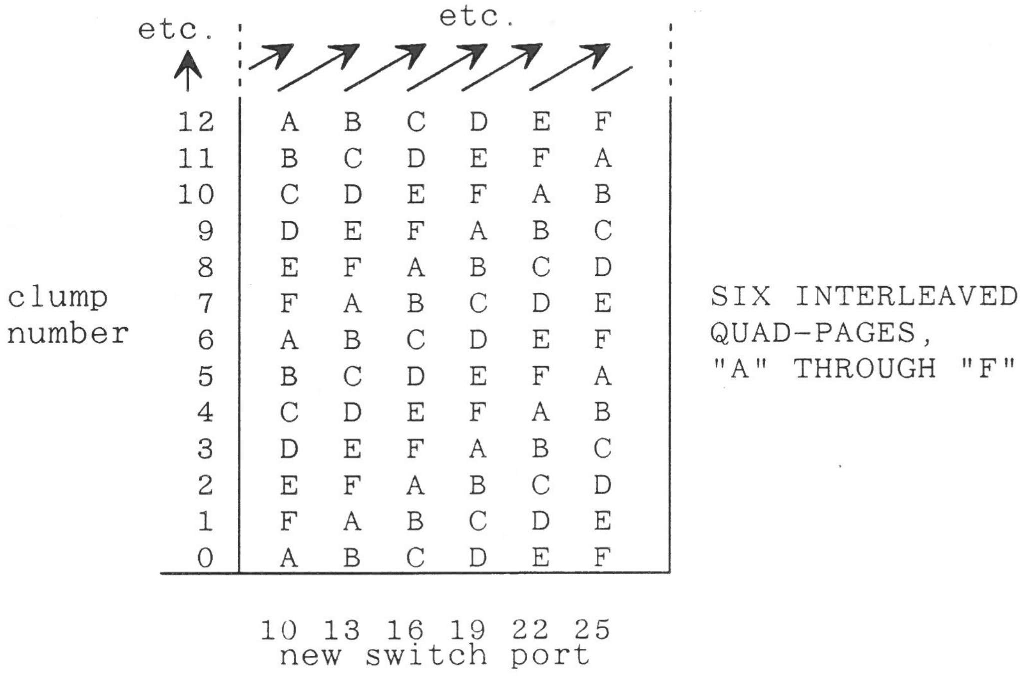


Figure 4–14 shows that successive clumps are mapped to higher and higher areas in the interleaved quad-page on each function board. We call this pattern of interleaved addresses a *stripe*, like stripes on a barber pole. This stripe is a result of cycling through the switch ports in a repetitive fashion. Although we allocated a quad-page of physical memory at each of the six ports in the pool, we have achieved only one interleaved quad-page of System Physical Address space. This would be a waste of system resources except for an additional feature of the interleaver.

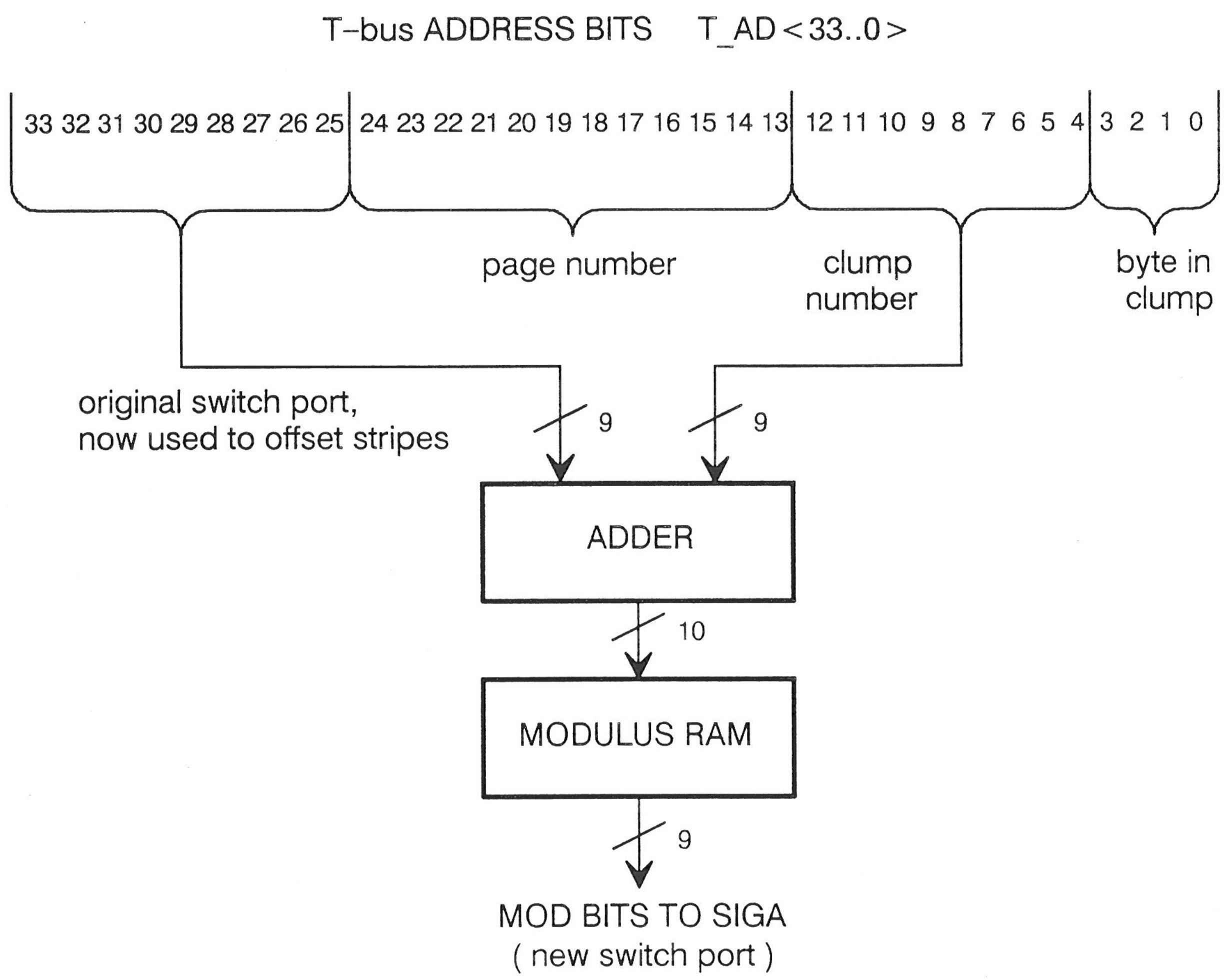
The unused clumps — blank in Figure 4–14 — can be used by other stripes, each offset from the others. Figure 4–15 shows how six interleaved quad-pages "A" through "F" fill up the six physical quad-pages of the pool. Quad-page "B" starts at the next switch port in the pool after where quad-page "A" starts, and thereafter has the same cyclic striping obtained by the modulo function as "A" has. Quad-page "C" starts at the next available quad-page in the pool, and so on. The number of stripes that this algorithm makes is the same as the number of switch ports in the pool, in this example six. This uses all the physical memory allocated to interleaved quad-pages very efficiently.

Figure 4–15 **Modulus RAM use – six stripes.**



To map different interleaved quad-pages to different stripes, the address supplied to the interleaver's modulus RAM is given an offset. In the example of Figure 4–15, quad-page "A" would have no offset, quad-page "B" an offset of one, "C" an offset of two, and so on. The interleaver provides this capability by adding T-bus address bits 33..25 to the clump number. In an address in a non-interleaved page, bits 33..25 specify the switch port; but in an interleaved quad-page these bits are replaced by the MOD bits from the interleaver. Therefore, these bits are available and provide a convenient place to specify this offset. Figure 4–16 shows the interleaver hardware to implement this.

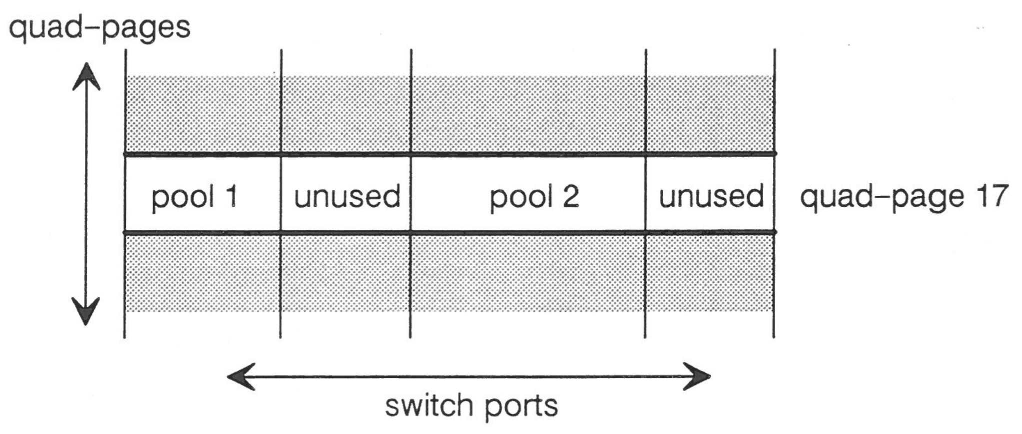
Figure 4–16 **Using an offset to pack stripes.**



Several Interleave Pools

The section above uses an example pool of six function boards (and thus six switch ports). The suggested uniform use of interleaving requires that a quad–page that is interleaved on one function board be interleaved on all function boards. Nevertheless, that quad–page on various function boards could be in different interleave pools, as illustrated in Figure 4–17. There, quad–page 17 on some function boards is in pool 1, on other boards it is in pool 2, and on yet other boards it is unused.

Figure 4–17 Multiple interleave pools.



There are several reasons why it would be convenient to have several interleave pools.

- Efficient use of resources

  The suggested model for TC2000 interleaving is uniformity of mapping. This implies that if a given quad–page number (a 32–kilobyte range of System Physical Address space) is referenced as interleaved at one switch port, it will be so at all switch ports. Therefore it is necessary to allocate that physical quad–page, at every port in the machine, to interleaving — even if only a few switch ports are actually used in the interleave pool. A mechanism to relax this requirement, while still using the uniform model, allows tailoring the allocation to the application.

- Independent, redundant copies for reliability

  One way to achieve robustness against errors or component failures is to store critical data structures in duplicate, in separate places. One way to do this is to use separate stripe offsets for the two copies. This has the disadvantage that failure of a switch port loses part of the data in each copy, so neither copy is usable; the data must be reconstructed by culling parts from each stripe. A simpler solution uses two interleave pools that have no switch ports in common. One copy is stored in each pool. Now one or the other copy is still intact after an error or failure.

- Performance control by reducing interactions

  The execution speed of any program is affected by contention for access to memory. Usually this interaction is negligible. Benchmark programs and some intensive applications, however, can be sensitive to the contention. For these, it can be useful to give the program its own interleaved pool. This isolates its memory accesses from those of the other processors. The benchmark's results are then more repeatable and meaningful, and the intensive application performs reliably. It could be that an exerciser test or diagnostic program would make such heavy use of its memory that isolating it from the rest of the machine would avoid degrading the service other users receive.

- Matching pool size to contention reduction needs

  With separate interleave pools, each pool can contain a different number of switch ports. Data structures that would suffer only a moderate amount of access contention at a single switch port can be placed in a small interleave pool. Data structures that would engender significant contention even in a small pool can be placed in a larger pool.

The TC2000 interleaver provides for eight interleave pools. (The hardware supports eight pools per interleaver, but the uniform use of interleaving, as section 4.5.3 recommends, limits this to eight pools per TC2000 machine.) Each pool operates as described above, with clumps mapped to stripes distributed among the memory subsystems of the pool. The multiple pool capability is implemented by having eight times as much modulus RAM as would otherwise be needed. By selecting a given pool, one of the eighths of the modulus RAM is selected.

The three additional address bits driving the modulus RAM come from the interleaver's pool RAM. The selection of a pool, that is, the address driving the pool RAM, is based on two fields of T-bus address bits.

- The first component of pool selection is T_AD<22..15>. These bits specify a 32-kilobyte *quad-page*.
  - Because T_AD<14..13> are not included in pool selection, individual pages cannot be independently assigned to interleave pools. Rather, all interleaved pages in a quad-page belong to the same pool. This is consistent with the CPU interface design — its Interleave Decision RAM decides whether a page is interleaved or non-interleaved based not on the entire page number, but on the quad-page in which the page lies.
  - Because T_AD<24..23> are not included in pool selection, the given quad-page *in each of the four 8-megabyte banks* belongs to the same interleave pool.
- The other component of pool selection is T_AD<33..28>. In an non-interleaved remote access these bits are part of the switch routing; in an interleaved access they are part of the stripe offset.
  - Because T_AD<27..25> are not included in pool selection, individual stripes cannot be independently assigned to pools. Rather, groups of eight stripes all belong to the same pool.

This concludes the conceptual operation of the interleaver. Figure 4–8 (above) shows the complete hardware implementation.

4.5.6 The Interleaver Loader

The modulus RAM and pool RAM are not read or written from the T–bus, but through the Switch Interface Gate Array (SIGA). The SIGA mechanism that performs this is called the *interleaver loader*, and is described in chapter 2.

5

I/O and the VMEbus Interface

5.1 The VMEbus

The VMEbus, designed mainly by Motorola, is now an industry standard. Many kinds of VMEbus compatible equipment are available from a wide variety of vendors.

Originally, the "VME" in "VMEbus" stood for Versa Module Europe. Now the name simply specifies a particular hardware architecture.

5.1.1 The VMEbus Specification

The definitive reference for VMEbus characteristics is *The VMEbus Specification*, by Motorola. The reader is advised to consult it whenever more detail is needed than is given here. *The VMEbus Specification* covers the following topics:

- Terminology, mechanical structure and functional structure
- Data transfer bus signals, addressing, operation and timing
- Arbitration for use of the data transfer bus
- Priority interrupt signals, functional modules and operation
- VMEbus system utilities, such as the system clock
- Electrical specifications, including timing, driving and loading
- Mechanical specifications, including boards, backplanes and connectors

TC2000 VMEbus equipment conforms to revision C.1 of *The VMEbus Specification*.

5.1.2 VMEbus Card Cages

The user of TC2000 I/O equipment needs some familiarity with basic VMEbus terms and requirements. A few basic ones are summarized here.

Boards may be either of two heights: **single height** (100 millimeters) or **double height** (233.35 millimeters). Each board typically occupies one **slot** of width, but multiple-slot boards are permitted. Boards are 160 millimeters deep.

VMEbus board dimensions are a subset of dimensions permitted in another standard, called **Eurocard**. A single-height VMEbus card is a "3U" Eurocard, and a double-height VMEbus card is a "6U" Eurocard. This leaves an easy path beyond the formal VMEbus specification. For example, the Sun Microsystems VMEbus-to-Multibus adapter card is a "9U" Eurocard, and is commonly described as a VMEbus card even though it is technically higher than allowed in *The VMEbus Specification*.

The **backplane** interconnects boards within a cage ("subrack"). Single height and double height boards may be mixed. All boards are required to contain a "P1" connector that mates with the backplane. Double height boards may also contain a "P2" connector. Besides the signals used for signalling, the backplane provides **power** and a **system reset** signal.

The **slots** of a VMEbus cage are numbered from the left side (when viewed from the front), ***starting with number 1***. We emphasize this because function board slot numbering on the TC2000 midplane starts with slot 0, not 1. (Function board slots are numbered from the left, when viewed from the function board side of the midplane.) A VMEbus cage cannot have more than 21 slots.

The board in **slot 1** of a VMEbus system must perform certain VMEbus system functions. Collectively, these functions are called the **system controller**. The system controller must provide:

system clock driver
data transfer bus arbiter
interrupt acknowledge (IACK) daisy chain driver
bus timer

The system controller may or may not also provide:

serial clock driver
power monitor

5.1.3 The TC/FPV's VMEbus Interface

Parameters of the TC/FPV VMEbus interface configuration and operation are listed and described briefly below.

- Standard or extended addressing

  The VMEbus Specification defines two kinds of addressing, standard and extended. The TC/FPV VMEbus interface is software configurable to respond to either kind. **Standard** addressing uses 24-bit addresses, for a VMEbus address space of 16 megabytes. **Extended** addressing employs 32-bit addresses, giving a VMEbus address space of 4 gigabytes.

- VMEbus master window

  The window from the TC2000 address space into VMEbus space is 16 megabytes, at a fixed location (the high 16 megabytes) in the local address space of the TC/FPV function board. References falling within the window are mapped onto the VMEbus at an address and with access parameters described by a VMEbus Master Map RAM register selected by the reference.

- VMEbus slave window

  The window from the VMEbus address space into TC2000 address space is configurable in both size and position. When using standard VMEbus addressing, the window is 4 megabytes. When using extended VMEbus addressing, the window is 16 megabytes. The window is software configurable to fall at any position in the VMEbus address space that is 4– or 16–megabyte aligned, respectively. References falling within the window are mapped onto the TC2000 global address space at an address and with access parameters described by a VMEbus Slave Map RAM register selected by the reference.

- Mapping granularity

  The mapping in both the master and the slave direction has 8–kilobyte page granularity. The bottom 13 address bits are passed through unmapped in each direction.

- Transfer size

  The interface services 1–, 2– or 4–byte read or write requests in either direction. Halfword (2–byte) and word (4–byte) requests must be halfword and word aligned, respectively.

- Locking

  The interface supports locking in each direction. A lock request on the T–bus results in holding the VMEbus until the lock is freed. A VMEbus transaction that falls in the slave window selects one of the slave mapping registers, each of which has a lock bit. If the lock bit is set, the slave interface opens and maintains a lock for the duration of the transaction.

- Auxiliary transaction control information

  In each direction, the interface provides auxiliary information to control the transaction. The VMEbus master mapper provides the six **address modifier** bits and the **interrupt acknowledge** bit required by *The VMEbus Specification*. The VMEbus slave mapper provides various T–bus control signals such as bypass, priority, path, interleaved, and lock.

- VMEbus mastership protocol

The TC/FPV VMEbus master interface is software configurable to follow either the **release–when–done** or the **release–on–request** bus mastership protocol defined in The VMEbus Specification.

- VMEbus system reset and system fail

  The VMEbus interface can assert the VMEbus **system reset** signal under software control. Both of the VMEbus signals **system reset** and **system fail** are monitored by the interface, permitting software to determine whether either has been asserted.

- VMEbus master request level

  The level on which the VMEbus master interface requests the VMEbus is selectable by jumpers on the TC/FPV board. (See the *TC2000 Maintenance Manual* for jumper settings.)

- VMEbus system controller functions

  The TC/FPV VMEbus interface can provide the **system controller** functions required of any board in slot 1 of a VMEbus system. This capability is enabled by jumpers on the TC/FPV board. (See the *TC2000 Maintenance Manual* for jumper settings.) The system controller functions are VMEbus arbitration, IACK daisy chain driving, and VMEbus system clock driving. The TC/FPV implements only single level arbitration.

- Timers

  The VMEbus interface contains three timers. The VMEbus Arbiter Timer and the VMEbus System Bus Timer are part of the arbitration function and operate only when the interface is VMEbus system controller. The VMEbus TC/FPV Master Bus Timer limits latency of references made by TC2000 software to VMEbus devices.

- Interrupts

  The VMEbus interface generates interrupt requests to the VMEbus on any of seven levels, and handles interrupt requests from the VMEbus on any of seven levels.

5.1.4 The TC/VMP VMEbus Midplane

TC/FPV boards may connect to a VMEbus via an additional midplane called the **TC/VMP**. One side of the TC/VMP has sockets for the VMEbus connectors of TC/FPVs, and the other side has sockets for standard VMEbus cards. In any given 8–slot module, a TC/VMP may be installed or absent. If absent, a piece of sheet metal is put in its place. The TC2000 architecture permits various TC/VMP designs, each connecting function boards and VMEbus slots in a different combination. Figure 5–1 shows the physical position of the TC/VMP and related components. Revision 1 of the TC/VMP, shown in Figure 5–2, implements ***five*** small VMEbuses. Sockets for eight (double height, "6U") VMEbus cards are mounted on the VMEbus card cage side.

On the function board side, only five of the eight slot positions are provided with sockets.

Figure 5–1 **Position of TC/VMP and related components.**

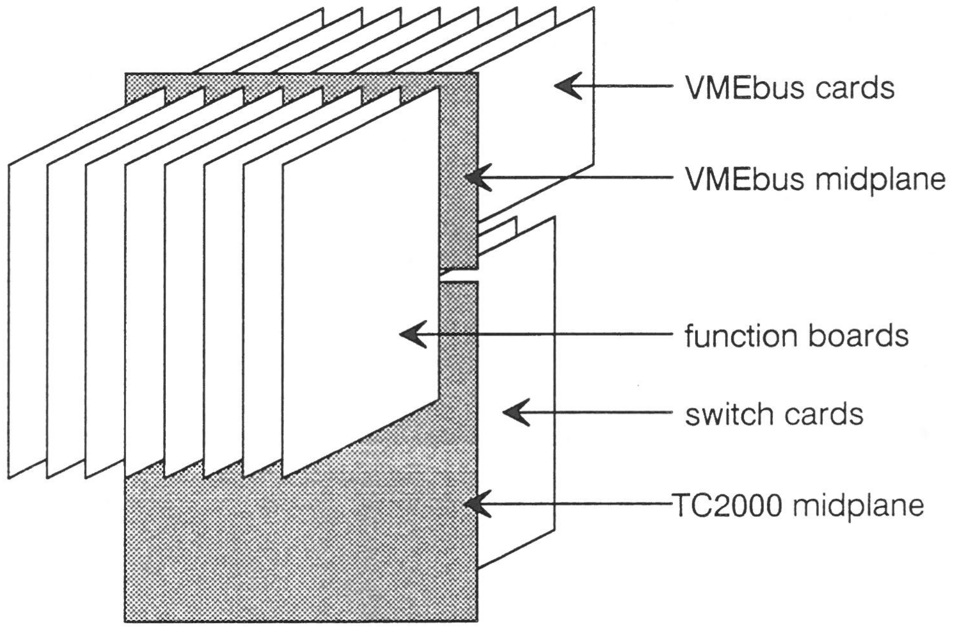
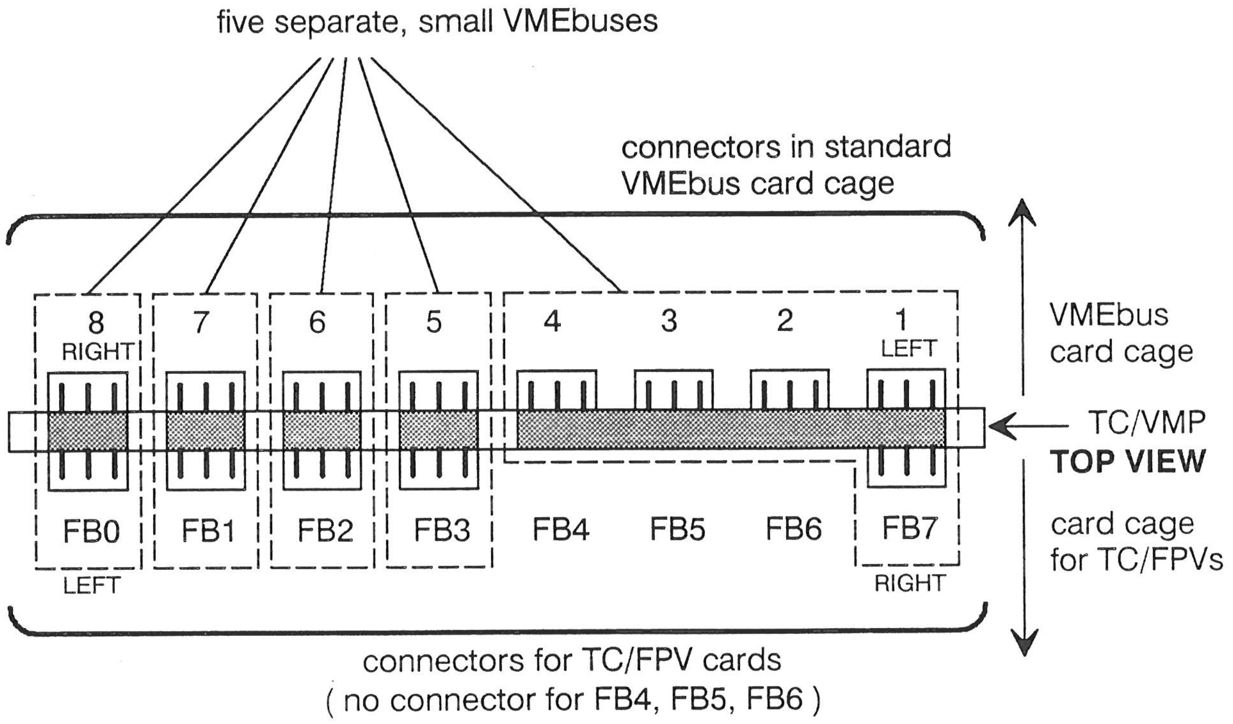


Figure 5–2 **TC/VMP connects function boards and VMEbus cards.**


five separate, small VMEbuses
connectors in standard VMEbus card cage
8
RIGHT
7
6
5
4
3
2
1
LEFT
VMEbus card cage
TC/VMP
TOP VIEW
card cage for TC/FPVs
FB0
FB1
FB2
FB3
FB4
FB5
FB6
FB7
LEFT
RIGHT
connectors for TC/FPV cards
(no connector for FB4, FB5, FB6)

The **system controller** function on a VMEbus is normally performed by the board in slot 1 ***of the VMEbus card cage***. Because of the novel 5–VMEbus, 2–sided structure of the TC/VMP, the system controller position is not identical with VMEbus card cage slot 1, and also is not identical with TC/FPV card cage slot 1. Rather, on each small VMEbus, either the TC/FPV or the directly opposite VMEbus card may be system controller. Typically, the TC/FPV is system controller.

Any given TC/FPV may be connected to VMEbus devices or not connected to VMEbus devices. It can be ***not connected*** in any of three ways: in an 8–slot module with no TC/VMP installed; in slot FB4, FB5 or FB6 of a TC/VMP; or in one of the other five slots of a TC/VMP but with no VMEbus device plugged into its small VMEbus on the other side of the TC/VMP.

A given TC/FPV may be ***connected*** to VMEbus device(s) plugged into its small VMEbus on the other side of the TC/VMP. In this case, the TC/FPV is often used as VMEbus system controller. Alternatively, the TC/FPV jumpers permit some modes of operation where a VMEbus device — plugged into the connector directly opposite the TC/FPV across the TC/VMP — can be system controller.

As in any VMEbus system, any card may assert the **system reset** signal. Also, the TC/VMP contains a circuit that, during power–up of the VMEbus cage power supply, asserts the **system reset** signal on each of the five small VME-buses.

5.1.5 Multiple VMEbus Systems

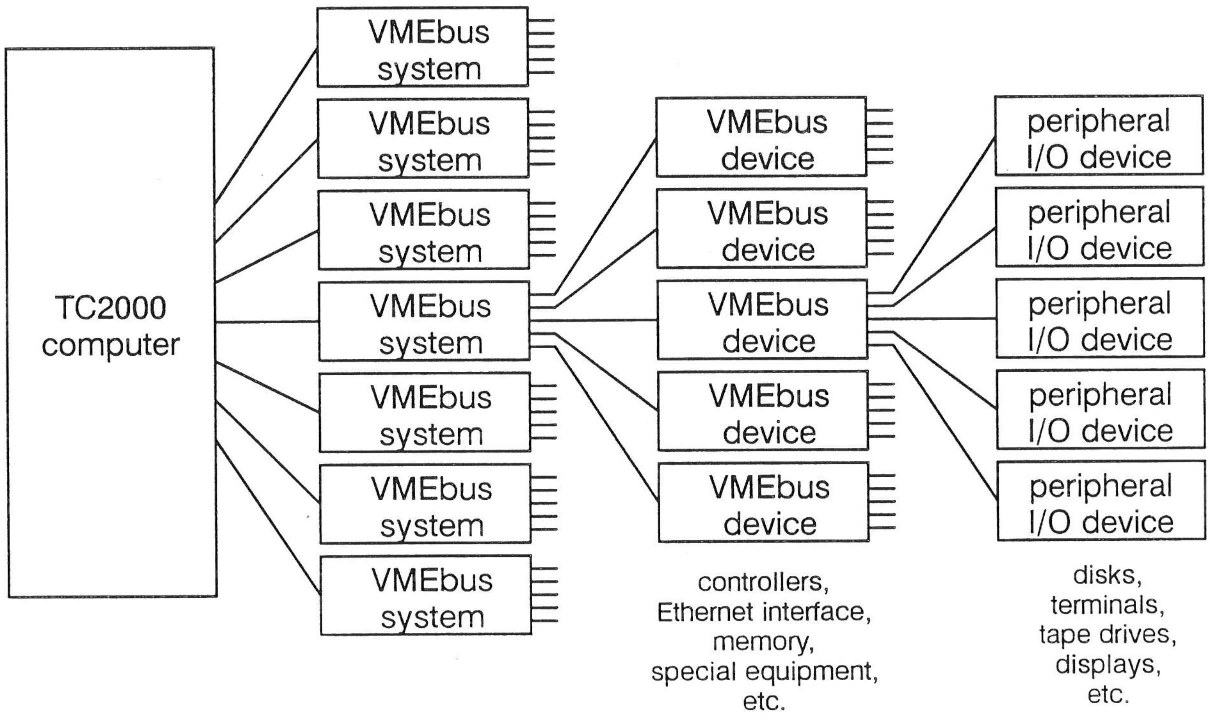
The TC2000 computer must connect to at least one VMEbus system, to access the disk from which the nX operating system boots. Since each TC/FPV in the TC2000 machine is capable of connecting to a VMEbus, there may be several VMEbus systems connected to the machine as a whole.

The TC2000 nX operating system supports multiple VMEbus systems. A separate file system can be installed on the disks of each VMEbus system. The current release of the nX system does not support "striping", the splitting of files into "stripes" across different disks or different VMEbus systems. The application program can — and when devices other than the default device are referenced, must — specify the VMEbus system for operations such as "open". Section 5.5 discusses device names and identification of their VME-bus system.

A pSOS^{+m} application program may directly access any VMEbus systems on function boards in the pSOS^{+m} cluster. Use of multiple VMEbus systems is more expected and more visible under the pSOS^{+m} system than under the nX system, for two reasons. First, the user controls I/O more directly under the pSOS^{+m} system, compared with the nX file system interface that most nX application programs use. Second, the time-critical application domain often handles a variety of I/O from different sources, compared to the time sharing (nX operating system) application domain.

Note that the TC2000 computer can connect to multiple VMEbus systems, and each VMEbus system can have several devices on it. Further, in the case of some VMEbus devices such as disk controllers and terminal controllers, several peripheral I/O devices (disk drives, terminal lines) can be connected to each VMEbus device. Therefore, the overall system capacity is quite large. This hierarchy is shown in Figure 5-3.

Figure 5–3 **Hierarchy of the I/O system.**


TC2000 computer
VMEbus system
VMEbus device
peripheral I/O device
controllers, Ethernet interface, memory, special equipment, etc.
disks, terminals, tape drives, displays, etc.

5.2 SCSI Adapter and SCSI Devices

The **SCSI bus** is an industry standard for interconnecting data processing equipment. "SCSI" originally stood for Small Computer Systems Interface, and has become popular in a variety of applications beyond its original "small computer" namesake. "SCSI" is pronounced to rhyme with "fuzzy".

For devices manufactured to operate on a SCSI bus, an adapter is used between these devices and the VMEbus in the TC2000 machine.

5.2.1 The SCSI Bus

The SCSI bus standard specifies the mechanical, electrical and functional requirements for an I/O bus interface, and command sets for peripheral device types — particularly storage devices — commonly used with small computers. The standard was developed by the American National Standards Institute (ANSI), originally based on a commercial small system parallel bus, the Shugart Associates System Interface (SASI). Some characteristics of the SCSI interface are:

- Physically, SCSI devices are cabled together, each device to the next. This results in a physical daisy chain, while electrically the bus is continuous.

All bus signals are common among all SCSI devices. Devices are mounted where physically convenient, not in a card cage as with the VMEbus. Because of the daisy-chaining, a SCSI bus is sometimes referred to as a *chain*.

- Electrically, the SCSI interface may be either single-ended (with a bus length up to 6 meters) or differential (up to 25 meters). The cable and devices on a bus must be all single-ended or all differential.
- Data transfer is 8-bit parallel and may be either asynchronous (up to 1.5 megabytes per second) or synchronous (up to 4 megabytes per second).
- The address space may be either the basic (2**21 blocks) or extended (2**32 blocks).
- Command sets for a variety of generic types of device are specified, including magnetic disks (fixed-medium, rigid removable-medium and Bernoulli-effect flexible "floppy" disks), magnetic tape (both start/stop and streaming), optical disks, printers and processors. The command set for any one device type is device independent; it hides the internal structure of the device, such as cylinders, heads and sectors of a disk.

For further information, please refer to the documents listed below.

- *Small Computer System Interface (SCSI)*, by the American National Standards Institute, Inc. (American National Standard X3.131–1986).
- *Common Command Set (CCS) of the Small Computer System Interface (SCSI)*, by the American National Standards Institute, Inc. (X3.T9.2/85–52 Rev 4.B, Addendum 4.B to Revision Four), June 23, 1986.
- *Standard for Electrical Characteristics of Generators and Receivers for use in Balanced Digital Multipoint Systems*, EIA RS–485–1983 (describes electrical characteristics used on SCSI bus).

5.2.2 The VMEbus-to-SCSI Bus Adapter

In the TC2000 computer, SCSI devices are supported by a VMEbus-to-SCSI-bus adapter, the Interphase V/SCSI 4210-BBN-SE Jaguar. The basic model of this adapter provides a single SCSI bus, and a second SCSI bus is supported by addition of a daughter board. In the TC2000 machine, each adapter has a daughter board installed. Up to seven SCSI devices may be placed on each bus. Jaguar features that enhance throughput are:

- The Jaguar can receive and queue **multiple commands** from the TC2000 computer.
- The Jaguar can **overlap activity** on the different SCSI devices, if the devices support overlapping via SCSI bus Disconnect/Reconnect.
- The Jaguar supports **scatter/gather**: data in contiguous areas in a SCSI device can be scattered into non-contiguous areas in TC2000 memory,

and data in non-contiguous areas of TC2000 memory can be gathered into a contiguous area in a SCSI device.

In operation, the SCSI bus adapter is invisible to the nX application programmer. In the current release of the TC2000 system, SCSI bus adapter(s) support the hard disk(s), the half-inch magnetic tape drive(s), and the quarter-inch streaming tape drive(s).

The following points describe varieties of SCSI bus adapters available:

- The 4210-BBN-SE card (standard in the TC2000 machine) is a mother card with a *single-ended* SCSI bus used as bus number 0, and a daughter card with a *differential* SCSI bus used as bus 1. The boot disk(s) are on chain 1, since they are differential.
- The 4210-BBN-DE card (optional) is a mother card with a *differential* SCSI bus and a daughter card also with a *differential* SCSI bus. This is used in systems with hard disks in addition to the standard boot disk. **Each differential SCSI bus can support a maximum of two hard disks.**
- The 4210-BBN-SC card (optional) is a mother card with a *single-ended* SCSI bus and a daughter card also with a *single-ended* SCSI bus. This is used in systems with the classified/removable disk, since it requires a single-ended SCSI bus.

The current nX device drivers require that there be a maximum of one SCSI bus adapter card per VMEbus system.

The following technical data may be of interest for interoperability with other user equipment.

For further information, please consult the documents listed below.

- For details of the SCSI bus, see the *Standard for Small Computer Systems Interface*, ANSI X9.3131 (1968).
- For details of the Jaguar SCSI adapter, see the *V/SCSI 4210 Jaguar User's Guide*, by Interphase Corporation.

NOTE

Available for compatibility with older, release 1 systems is the Interphase V/SCSI 4210-3-SE Jaguar VMEbus-to-SCSI-bus adapter. This is a mother board with a single-ended SCSI bus and ***no daughter board***.

5.3 VMEbus Bus Repeater

A VMEbus bus repeater is not part of the current, standard TC2000 base system. However, sometimes VMEbus devices that would logically be placed all in one card cage are instead placed in separate cages. For example, this situation may arise to connect a "9U" card to a "6U" card cage (such as on the TC/VMP); or to accommodate cabinet limitations (such as placing the card closer to hardware it controls); or to provide more VMEbus slots (such as for the small VMEbuses on the TC/VMP). The solution is to use a VMEbus **bus repeater**.

VMEbus bus repeaters are commercially available. When a bus repeater is needed in the TC2000 computer, the "VMEbus Repeater 2000" manufactured by HVE Engineering Inc. is normally used. The bus repeater consists of two VMEbus cards connected by two cables called the "external bus". The cards plug into separate VMEbus cages and couple the two VMEbuses. In the TC2000 computer, a VMEbus bus repeater is typically used to couple one of the small VMEbuses on a TC/VMP to a VMEbus card cage in the TC2000 peripheral cabinet, where there is room for a VMEbus cage with more and higher ("9U") slots.

The VMEbus bus repeater adds a relatively small amount of delay to VMEbus transactions, so its impact on transfer rate and latency is slight. The "external bus" cables are made by HVE Engineering and are six feet long.

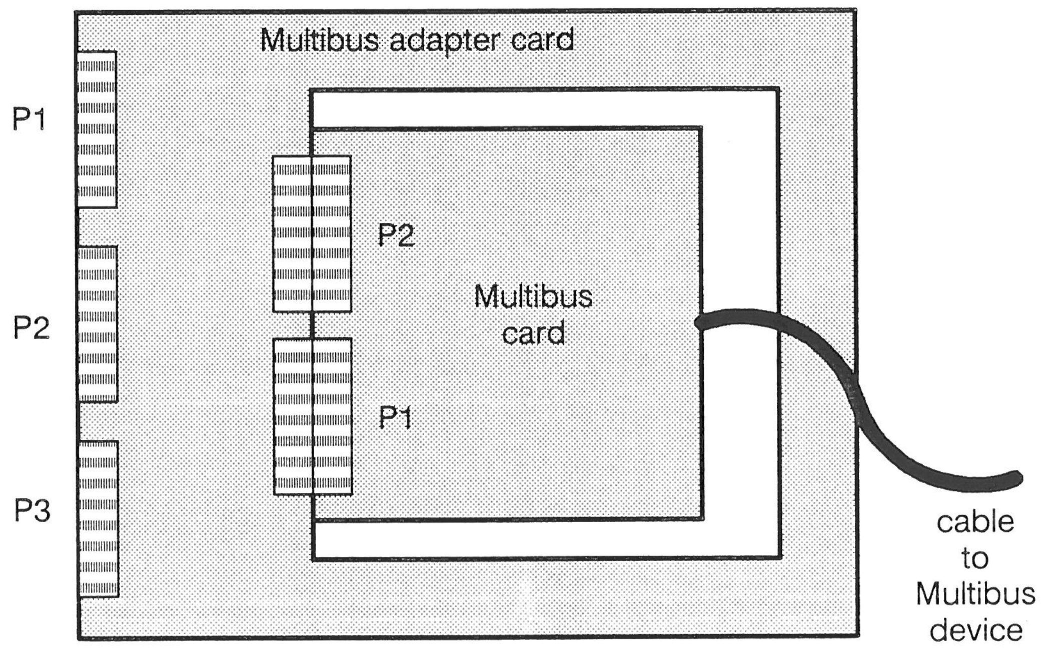
For further information, please consult the document listed below.

- *Product Specification AS90172000A, VMEbus Repeater 2000, VMEbus High Speed Repeater System*, by HVE Engineering, Inc.

5.4 Multibus Adapter

The Multibus is another industry standard for interconnecting computing equipment. There are no Multibus-based devices in the current, standard TC2000 machine. When the user's application requires interfacing to Multibus devices, a **Multibus adapter** card made by Sun Microsystems and modified by our factory is normally used in the TC2000 machine. This adapter, illustrated in Figure 5-4, plugs into a "9U" VMEbus slot. A large cutout space with connectors along one edge accepts the adapted Multibus card.

Figure 5–4 **Multibus adapter card.**



For further information, please consult the documents listed below.

- For a description of the Multibus, see the *Intel Multibus Specifications Manual*, by Intel Corporation.
- For details of the Multibus adapter, see the *Sun VMEbus–Multibus Adapter Board User's Manual*, by Sun Microsystems, Inc.

5.5 Device Naming and I/O Bus Specification

When an nX application program refers to a specific device, the operating system must be able to identify which VMEbus system the device is on. This specification is given by a **logical I/O bus number** that identifies the slot containing the TC/FPV function board connected to the VMEbus system. This logical I/O bus number appears in the nX device special file major device number, and by convention in the name of the device special file.

Entries in the **system boot configuration file**, maintained by the system administrator, define the correspondence between logical I/O bus numbers and the physical slot numbers in the machine. Application programs use the logical I/O bus numbers, which the nX software translates into physical slot numbers when it services the program's calls. In the example entry below, the function board (node) in slot 7.1.7 is declared to implement logical I/O bus number 1.

```
node 7.1.7 iobus 1
```

An example device name is shown below and each of its fields is described.

```
/dev/rsd0a
```

/dev/	nX convention for the file name of devices
r	Access mode: raw (r) for a device without a file system, nil (no character) for a device holding a file system
sd	SCSI device identifier — the VMEbus device is a VMEbus-to-SCSI-bus interface (especially the Interphase Jaguar), and the target device on that SCSI bus is a disk (sd) or tape (st)
0	Unit number — 0, 1, 2, 3... (No special significance, typically assigned in the order the device names were created.)
a	Partition (SCSI disk drive partitions are labeled a, b, c...)

Another example is the format for a magnetic tape drive on a SCSI bus:

```
/dev/rsmt<unit><rewind or nil><density (h or l)>
```

The tool program *showdev* (found in /etc/showdev) interprets device names, telling what SCSI bus they are on, their unit number, etc. See the *showdev* manual page for further details.

5.6 TC2000 I/O Devices Summary

The nX operating system contains device drivers for a variety of standard devices. These drivers take care of device-specific requirements, buffering, and error handling. An application program running under the nX system has standard, high-level system calls to access each such device. These calls provide, for example: read, write, open, close, and status manipulation depending on the nature of the device. The general nature of nX device drivers, and the system calls to access them, are essentially standard UNIX.

A program running under the pSOS^{+m} system has access to the nX file system for I/O. Also, pSOS^{+m} programs may include device drivers to access I/O devices on function boards in their pSOS^{+m} cluster. The application programmer may have to supply such drivers, depending on the device and the application requirements.

For the maximum numbers and combinations of various I/O devices supported, either as standard or as special order configurations, please consult the sales literature or contact a sales representative. These sources should also be consulted for constraints on mixing native VMEbus devices, native SCSI bus devices, and native Multibus devices in a system.

5.7 Hard Disk

The standard TC2000 hard disk controller and drive is the Northern Telecom model NT 8514–SCSI. (Its SCSI interface is *differential*, not single-ended.) The disk's capacity is shown in Figure 5–5. The capacity shown when formatted for nX is space usable for data, and does not include spare tracks or spare sectors.

Figure 5–5 Hard disk drive.

drive model and manufacturer	capacity (megabytes) unformatted	capacity (megabytes) formatted
NT 8514–SCSI Northern Telecom	1066	940

This disk is used for the nX file system. It stores the nX system software from which the machine is booted, as well as utility programs and user files. Demand paging is performed on this disk.

For further information, please consult the documents listed below.

- For discussion of formatting the disk, repartitioning the allocations of disk space, and file system setup, see the *System Administration Guide*.
- For details of the Northern Telecom hard disk drives, see the *Product Specification* and the *Installation Guide*, both by Northern Telecom Inc.

NOTE

Available for compatibility with older, release 1 systems is the Multibus-based Xylogics 451 controller and "SMD–E"-based disk interface and Northern Telecom NT 8414 hard disk drive, with capacity 887 (unformatted) and 700 (formatted) megabytes.

5.8 Half–Inch Magnetic Tape

The standard TC2000 half-inch magnetic tape device is a Cipher M990 GCR CacheTape drive with an integral Pertec interface. Physically attached to the tape drive is an NCR/CSC–100 SCSI adapter that converts between the Pertec interface and the SCSI bus.

The half-inch tape device is available for general purpose use by application programs, for disk backup, for data transfer among systems, and so forth.

The Cipher tape drive is compatible with IBM and ANSI standard 9-track reel-to-reel tape drives. It holds the tape horizontally rather than the vertical position used in many earlier magnetic tape drives. This saves rack space; it occupies only 14 inches height in the standard 19-inch-wide TC2000 peripheral cabinet. It is self-threading and easy to use. In normal operation, the user need only insert the tape and use the "load/rewind", "unload" and "on line" front panel buttons.

WARNING

REMOVING TAPE AFTER POWER CYCLING
If power is turned off while a tape is loaded, and later power is turned on, the drive thinks that no tape is loaded. ***But fingers holding the tape reel are still engaged.*** Forcefully removing the tape reel can break these fingers. To release the tape, press the "unload" button.

The characteristics of the half-inch tape device are summarized below.

- The current release of nX software supports two writing **densities**, 1600 and 6250 bits per inch (BPI), although the Cipher drive is also capable of 3200 BPI operation. The application software selects the density via the device name — a final "l" or "h" for low or high density, respectively. This selection by software overrides the density selected by the front panel "density select" button. On reading, all three densities are supported.
- The drive performs automatic **error correction** in all modes, and automatic read and write **retry**.
- The drive accepts 7-, 8.5- and 10.5-inch **reels** (for 600-, 1200- and 2400-foot tape, respectively). If recording is to be at 6250 BPI density, we recommend using tape certified for zero defects through 6250 BPI, to reduce errors (both recoverable and unrecoverable) and to permit recording as much data on the tape as possible.
- The drive contains an **internal cache memory** that serves as a tape buffering system for high performance. This allows read and write commands to be serviced quickly. Data is spooled between the cache and the physical tape when the tape position and motion are correct. The intelligent buffering achieves good tape utilization (streaming), even when significant intervals separate the write commands.
- At 6250 BPI, Group Code Recording (GCR) **recording technique** is used, and at 1600 BPI, Phase Encoding (PE) is used. Physical **tape speed** is 70 inches per second (IPS) at 6250 BPI, and 100 IPS at 1600 BPI.
- The drive has several diagnostic features; one that some users find helpful is an optional display of the approximate number of feet of tape remaining before the End Of Tape (EOT).

The **capacity of a reel of tape** is affected by many factors, but a rough estimate is provided below. Some of the factors are: tape length (actual may vary slightly from nominal); recording density; record size; number of faulty areas (writing automatically passes over such areas); 15–foot leader and trailer at the beginning and end of tape; number and size of inter–record gaps; number and size of end–of–file marks; and variation of tape speed from nominal.

Figure 5–6 provides a rough estimate of reel capacity, assuming 64–kilobyte records, all one file, no faulty areas, and nominal inter–record gaps (0.6 inches at 1600 BPI, 0.3 inches at 6250 BPI).

Figure 5–6 **Rough estimate of tape reel data capacity.**

	recording density (BPI)	
tape length (feet)	1600	6250
600	10.3	37.8
1200	21.1	77.7
2400	42.7	157.

Capacity estimate is in megabytes per reel.

For further information, please consult the documents listed below.

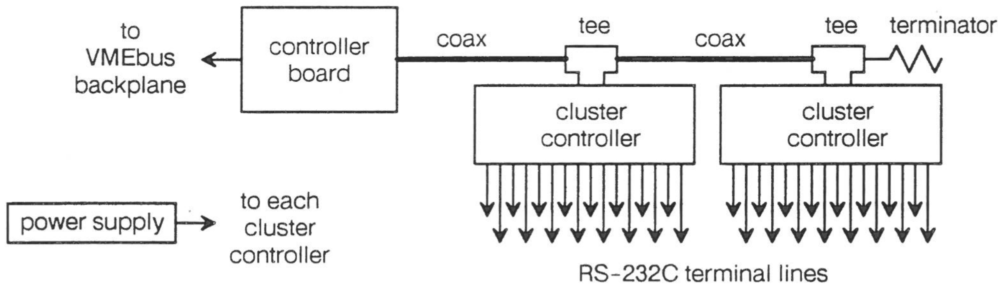
- For discussion of **head cleaning** and correct setting of the firmware configuration parameters in the Cipher drive, see the *Maintenance Manual.*
- For details of the Cipher tape drive, see the *Series M990 GCR CacheTape Tape Drive Product Description*, the *M990 GCR CacheTape Unit Maintenance Manual*, and the *M990 GCR CacheTape Unit Engineering Drawings*, all by Cipher Data Products, Inc.
- For details of the NCR SCSI adapter, see the *Functional Specification, NCR/CSC–100 SCSI Adapter*, by NCR.

5.9 Terminal Concentrator/Controller

The standard TC2000 terminal concentrator device is a Systech model HPS–6245 host adapter and one or more HPS–7080–030 cluster controllers. The Systech unit is a VMEbus device. "HPS" stands for "high performance serial".

The Systech terminal concentrator is comprised of a **controller board** and one or more **cluster controllers**. The controller board plugs into the VMEbus, and is connected to the cluster controllers by a coaxial cable bus. The coax bus may be up to 1000 feet long. Two models of cluster controller support 8 or 16 terminal lines each; the standard TC2000 equipment supports 16 lines. Figure 5–7 shows the system schematically.

Figure 5–7 **Terminal concentrator system — example.**



Each terminal line conforms to Electronic Industries Association (EIA) specification RS–232C, wired as Data Communication Equipment (DCE).

The lines supported by the terminal concentrator are available for user terminals and as devices upon which nX application programs may perform I/O.

The default hardware line characteristics are summarized below. An nX user program may change these characteristics of a line opened as a device, by making *ioctl* system calls modifying the special file /dev/tty. Several other line characteristics, relating to how the device driver software treats input and output, are also modifiable with *ioctl* calls. This is essentially the same as the standard UNIX tty device driver user interface.

- **Data rate** — 75 to 19200 baud; default is **9600 baud** (use "external A" rate to get 19200 baud)
- **Parity** — odd, even, or none; default is **none** (the hardware also supports mark and space parity, but these are not available through the nX software)
- **Character length** — 7 or 8 bits; default is **7 bits**
- **Stop bits** — **1** (the hardware also supports 2 stop bits, but this is not available through the nX software)

The detailed operation of the Systech terminal concentrator depends on microcode loaded into it by nX, but supplied by and proprietary to Systech. For further information, please consult the documents listed below.

- For configuring the terminal lines (via the system file "hps.conf"), see the *System Administration Guide*.
- For *ioctl* system calls, see the *nX Programmer's Reference* — section 2 for system call *ioctl* and section 4 for special files *hps* and *tty*.
- For details of the Systech terminal concentrator, see the *HPS–6200 Series Application Installation User Manual*, the *HPS Terminal Control Software User Manual*, the *HPS Cluster Controller Technical Manual*, and the *HPS VMEbus Advanced Host Adapter (HPS–6245) Technical Manual*, all by Systech Corporation.

5.10 Ethernet Interface

The standard TC2000 Ethernet interface device is the Excelan model 302. The Excelan interface is a VMEbus device and therefore connects directly to the VMEbus.

Application programs do not normally access the ethernet interface directly. Rather, they execute nX system calls to perform various Ethernet activities such as opening connections to remote hosts. The nX operating system supports the Network File System (NFS), permitting access to file systems on remote machines. Remote login to and from the TC2000 machine are supported, as is electronic mail. Some or all of these capabilities may be administratively prohibited at high security sites.

The current nX device drivers require that there be a maximum of one Ethernet interface card per VMEbus system.

For further information, please consult the documents listed below.

- For discussion of configuring the Ethernet parameters, see the *System Administration Guide*.
- For a detailed description of the Ethernet, see:
    - *The Ethernet: A Local Area Network: Data Link Layer and Physical Layer Specifications*, by DEC, Intel and Xerox Corporations (1980).
    - *The Ethernet: A Local Area Network: Data Link Layer and Physical Layer Specifications, Version 2.0*, by DEC, Intel and Xerox Corporations (1982).
    - *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications (Standard 802.3–1985/International Standard 8802/3)*, by The Institute of Electrical and Electronics Engineers, Inc. (1985).
- For details of the Excelan interface, see the *EXOS 302 Reference Manual*, by Excelan, Inc.

NOTE

Available for compatibility with older, release 1 systems is the Multibus-based Excelan 301 Ethernet interface.

5.11 Quarter-inch Streaming Tape

The standard TC2000 quarter-inch streaming magnetic tape drive is the Tandberg TDC 3640. The Tandberg drive is a SCSI bus device and therefore connects to the VMEbus via a SCSI controller, described in section 5.2.

Streaming is both an activity and a property of a device. The activity of streaming is maintaining a constant stream of data to or from a device, so that the maximum storage capacity or throughput is achieved. A streaming device is one designed to be used with an uninterrupted stream of data. The Tandberg drive uses a ¼-inch width tape in a cartridge. It writes data serially on a single track, starting at the beginning of the tape and recording continuously to the end. It then reverses the tape motion, and records data serially on a second track back to the beginning. It continues this serial, end-to-end recording until either the data stream terminates, or all 15 tracks have been written.

On a 600-foot (183-meter) cartridge, the TDC 3640 can record 125 megabytes of data. This is a maximum capacity, and is decreased by any intervals during which the data stream is interrupted. During short interruptions, the drive continues tape motion and records a filler pattern that is not returned to the user when read. A long interruption of data causes the drive to terminate the write operation.

Unnecessary start and stop operations in the middle of the tape, or large filler regions, will slow down the operation considerably. Therefore, to write the most data per cartridge and take the least time to write it (and later to read it), it is important that the data stream be interrupted as little as possible. This impacts the user, for example, if an application program is spooling data to the streaming tape drive. If the function board executing that program is also executing several other programs, the stream may be interrupted because the program may not get enough CPU time to keep up with the tape drive. Also, the program itself should be designed to provide the data stream quickly.

The streaming tape drive is ideal for backing up hard disk file systems, archival storage, and data interchange among machines. In some applications, the drive may also be appropriate for data logging. (For backing up the entire disk file system of a TC2000 machine, however, the larger capacity of the half-inch tape medium may be appropriate.)

The **tape** used should be of high quality to avoid unrecoverable errors. Tandberg recommends use of only 3M DC600A or 3M DC600XTD tape cartridges, or their equivalent. For critical applications, the user may wish to test the entire length of the tape before trusting it with real data.

For further information, please consult the document listed below.

- For details on the drive, see the *TDC 3600 Series Streaming Tape Cartridge Drives — TDC 3620 / 3640 / 3660 Reference Manual*, by Tandberg Data A/S, Data Storage Division.

5.12 Removable Disk Drive

The standard removable disk drive, intended for use where the TC2000 machine is used in a classified operation, is based on the Artecon RSU-304 removable package. The removable package is designed for 10,000 insertions and extractions. Into this package, BBN integrates the Hewlett-Packard HP97548S 5¼-inch disk drive. This is a single-ended, synchronous SCSI device with capacity 795 (unformatted) and 663.8 (formatted) megabytes. The standard configuration is two drives per controller.

For further information, please consult the document listed below.

- For details on the drive, see the *HP9754XS/D SCSI Disk Drives OEM Product Manual*, by Hewlett-Packard (HP manual order number HP 19530).

5.13 Special Site Peripherals

The applications at a given TC2000 site may involve specialized peripheral devices and controllers. Examples of this are graphics displays, array processors, video disk memories, sensors and actuators, and additional TC2000 computers or other general-purpose computing equipment. The VMEbus interface provides a flexible and powerful means to connect a limitless variety of such equipment. The TC/FPV is highly configurable and will operate well in a wide variety of VMEbus environments. This provides the capability to accommodate and interoperate with site-specific equipment. Normally, the site programming staff supplies the appropriate device drivers for such equipment.

6 The Supporting Modules

This chapter describes modules that support the function boards, the main computational elements of the TC2000 computer. These supporting modules are the Test and Control System (TCS), the clock card, the midplane, and the power supply and distribution system.

6.1 TCS

This section provides a general description of the TC2000 Test and Control System (TCS). For a further description of the TCS, and particularly the commands to the TCS Executive (TEX), please refer to the *TC2000 System Administration Guide*.

6.1.1 TCS Tasks

The TCS performs several tasks either automatically or under the direction of a person at the console terminal. In many cases, the automatic execution of these tasks ensures their rapid completion, avoids error prone manual specification of steps and parameters, and frees the operator from the tedium involved. The TCS is capable of the following categories of task:

- Testing — running both simple tests (called Power On Self Tests, or POSTs) and more complex exercise and diagnostic programs, including via remote telephone connection (installation is optional), isolating boards for testing, changing voltage margins during maintenance
- Bootstrapping — loading sufficient bootstrap code into TC2000 function boards so that the operating system(s) may be brought up
- Monitoring — machine temperatures and voltages can be monitored during operation, and appropriate action taken if these exceed predefined limits

- Control — powering on and off the Power Distribution Units (PDUs) and thereby the bulk power to the machine, powering on and off individual boards, enabling switch paths, setting the switch clock rate, and similar machine hardware initialization and control functions
- Configuration management — configure the machine under direction from TCS disk files and/or console terminal commands, using specifiec function boards to run the nX operating system (including a designated "king" or "master" function board), others to run the pSOS^{+m} operating system, others to be powered on but run neither system, and others to be left off
- Operator interface — informing the operator about conditions in the machine, executing commands entered on the console, and providing a communication link to the operating system(s) running on the function boards
- System software interface — providing system configuration and status information to the operating system(s), responding to system inquiries and commands

6.1.2 TCS Hardware Components

The TCS consists of the following hardware components:

- TCS master

  The TCS master controls and coordinates all TCS activity, and thereby oversees the operation of the entire machine. The master is an IBM PC/AT compatible microcomputer. In the master are the following:

  - A CPU card, containing the CPU, a SCSI port, two RS-232 serial ports, a real time clock / calendar with battery backup, and 512 kilobytes of memory.
  - A TCS interface card (TC/TCS), containing an interface to the TCS bus (described below), an interface to the front panel switches and indicators, control for the machine bulk power, a PC/AT bus interface, a TCS master watchdog timer, and PROM for bootstrap code specific to the TC2000 TCS master.
  - A modem card that connects to a standard modular telephone jack on the back panel, allowing optional connection to a telephone line for remote diagnostic and testing by field service personnel.
  - A hard disk, that holds TCS-related information such as the TCS master operating system (MS-DOS or PC-DOS), the TCS master executive (TEX), a TCS system diagnostic, machine configuration file(s), function board Power-On Self Test (POST) code, diagnostic code (runs in function boards and/or the TCS master), machine bootstrap code, and may hold a machine log file.

  - ○ A floppy disk, primarily used to load new versions of the TCS software and to load special diagnostics during field servicing.

- TCS slaves

  Each TCS slave controls and monitors the individual curcuit board of which it is a part. Each TCS slave is a Motorola 68HC11 microcomputer chip, with a small amount of support circuitry. Slave processors use a common power supply and individual clocks that are independent of the cards they monitor and control. Therefore, they can function even when their board is powered off. Aspects of the slave's environment that it monitors or controls depends on the type of board housing the slave.

- TCS bus

  The TCS "bus" is the communication system between the TCS master and the TCS slaves. Although it is basically a multi-drop, asynchronous, serial bus, in implementation it also contains fan-in and fan-out (described later).

- TCS front panel

  The TCS front panel contails a keyswitch, a button that resets the TCS master, and four indicator LEDs. An illustration of the front panel appears in chapter 1.

- TCS back panel

  The TCS back panel contains various connectors, including jacks for the TCS console terminal, the control line to the Power Distribution Units (PDUs), the (optional) telephone line, and a standard telephone (used with the optional telephone line). Provision is made for an Uninterruptible Power Supply (UPS) status sense connector as a future, optional feature.

- TCS power supplies

  One supply powers the TCS master, while another supply powers all TCS slaves in the machine. (Additional TCS slave power supplies are required for machines with over 64 function boards.) The slave power is 5 volts, often called "TCS +5" or simply "TCS 5".

- Machine bulk power control

  The TC2000 bulk power, ±24 volts DC, is applied to each function board, switch card and clock card. Each of these has its own, on-board power supply that converts the bulk power to the voltages needed locally. The bulk power supply system can be turned on and off by the TCS master. The master controls the Power Distribution Unit (PDU) in each cabinet, which then applies the external AC power to the bulk power supplies within the PDU's cabinet. (The utility cabinet PDU, however, is controlled by the front panel keyswitch.)

- TCS terminal

The standard TCS terminal is a DEC VT320. If hard copy is needed, a VT320 with a printer connected can be used.

Figure 6–1 shows the basic structure of the TCS, and Figure 6–2 shows the TCS master.

Figure 6–1 **TCS block diagram – overview.**

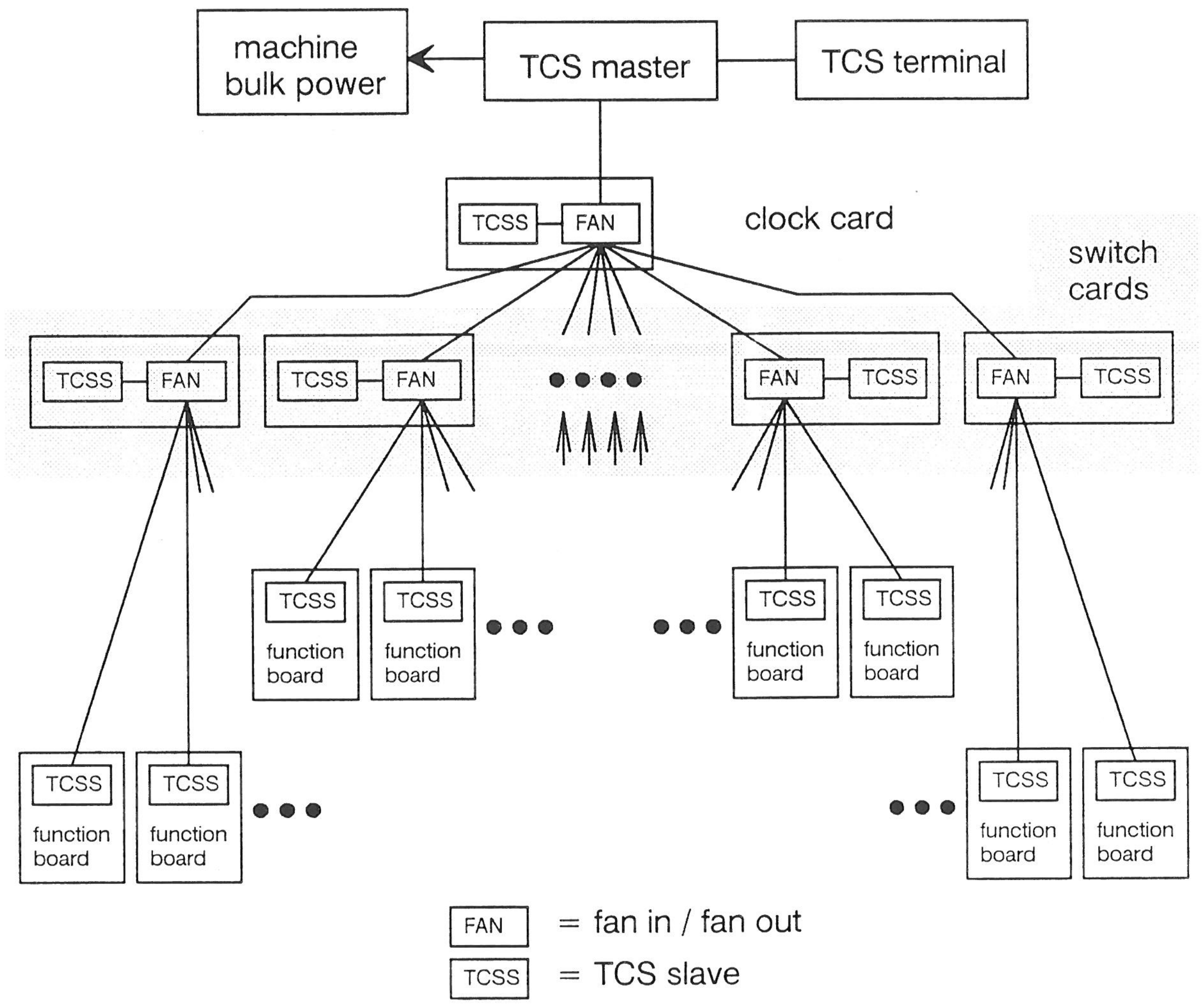
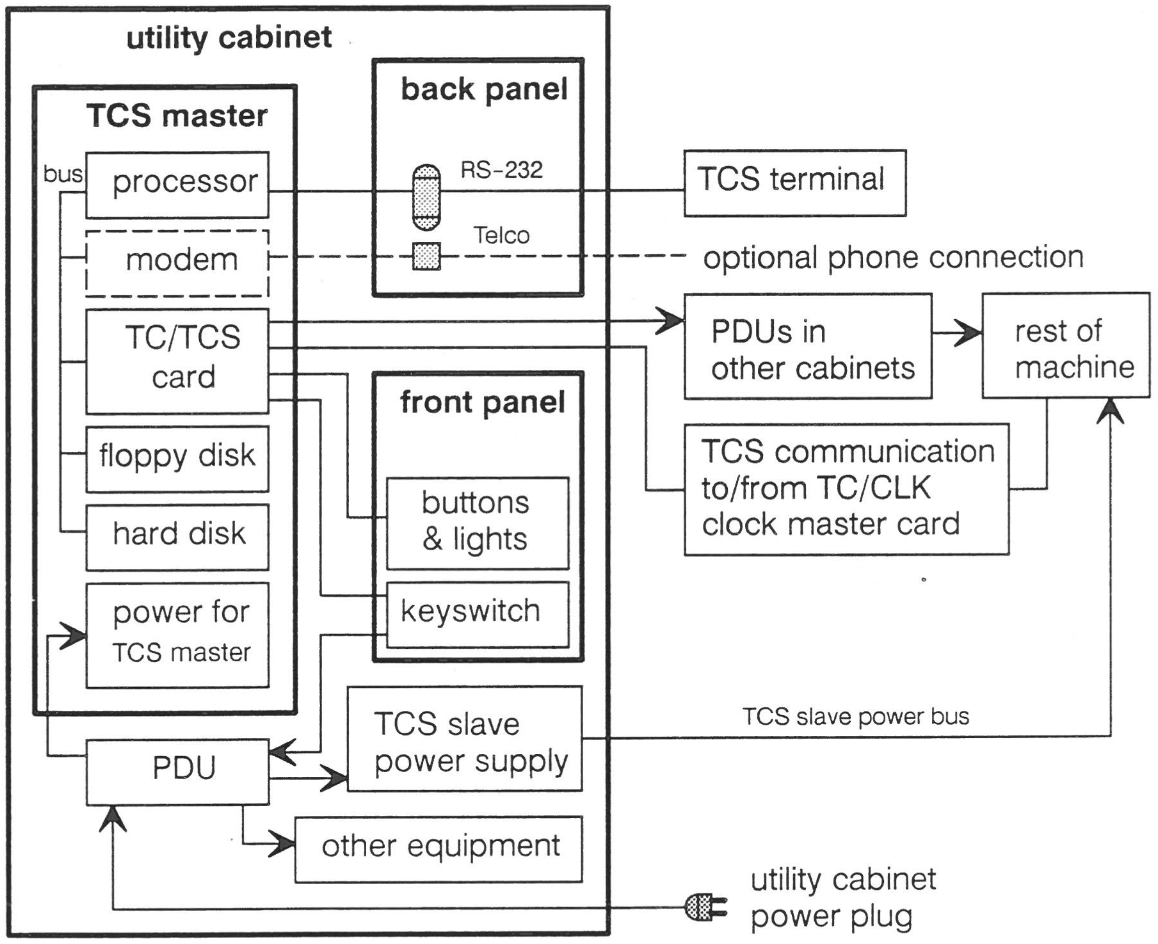


Figure 6–2 **TCS master and associated equipment block diagram.**



6.1.3 TCS Bus

The TCS communication bus, or "TCS bus" for short, is a cable that originates at the TCS interface card (TC/TCS) in the TCS master. The following four wires comprise the cable:

TCS transmit–direction data (master to slave)
TCS receive–direction data (slave to master)
TCS master identity (signal named A/B*)
ground

(The *TCS master identity* is provision for a future capability of dual TCS masters. This, and/or dual Butterfly switches, could be used to achieve a short Mean Time To Repair (MTTR) for certain applications.)

The TCS bus runs at 125 kilobits per second. With framing, start and stop bits, the result is 11,363 bytes per second. The same rate is used for both the master to slave direction and the slave to master direction.

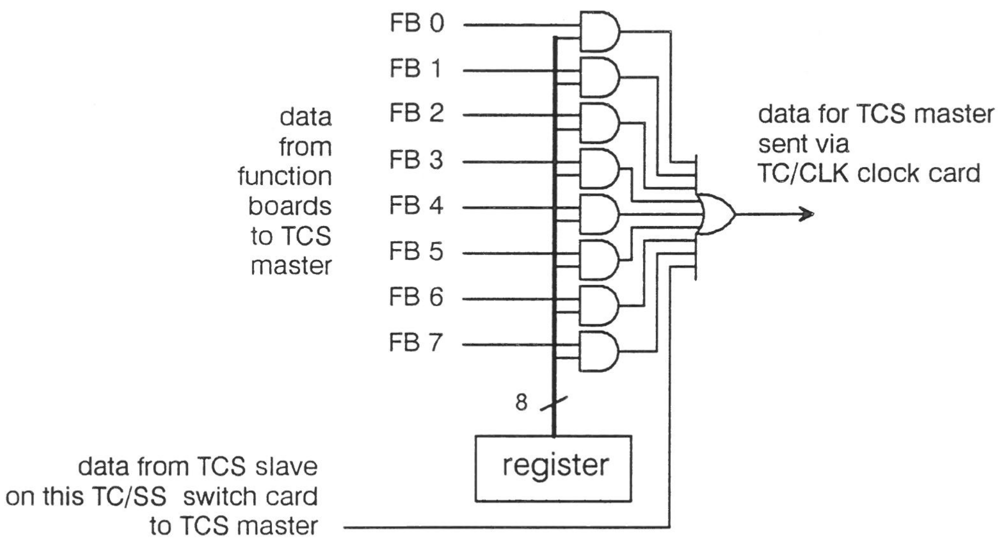
Receive (Slave to Master) Direction

In the slave to master direction, the TCS bus connects all the slaves to the master. On each midplane, the slaves on the eight function boards are connected to the TC/SS switch card. The wire from each function board is bused over the midplane to the switch card, where it connects to a schmitt trigger input AND gate. The other input to that AND gate comes from a register that is used to amputate function board slaves from the slave–to–master bus. This means that there are eight AND gates on each TC/SS switch card that receive data from eight function boards. These gates, and the TC/SS card's own data to the TCS master, are OR'ed together and drive a serial line connected to the TC/CLK clock card. The register that controls the enabling of the function board slaves is controlled by the slave on the TC/SS switch card. A TC/SS slave can amputate any of the eight function boards from the slave–to–master bus.

When the data from a TC/SS switch card arrives at the TC/CLK clock card, it is AND'ed with a control bit and OR'ed with data from other TC/SS cards before being sent on to the TCS master. The enable signals on these AND gates are controlled by the TCS slave on the TC/CLK.

Figure 6–3 illustrates the AND and OR fan–in of the TCS bus in the receive (slave to master) direction, using a TC/SS card as an example. At each stage of TCS bus fan–in, such a circuit is used. In Figure 6–1 above, this fan–in goes from the function boards at the bottom to the TCS master at the top.

Figure 6–3 **Slave to master TCS bus fan–in.**


FB 0
FB 1
FB 2
FB 3
FB 4
FB 5
FB 6
FB 7
data from function boards to TCS master
data for TCS master sent via TC/CLK clock card
8
register
data from TCS slave on this TC/SS switch card to TCS master

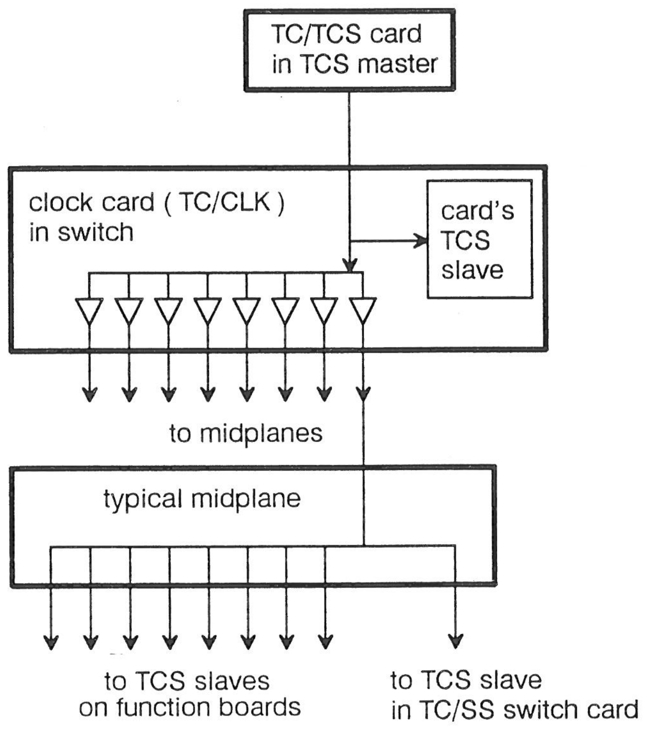
In normal operation, each TCS slave sends data to the master only in response to a command from the master, and the master is careful to let only one such command be outstanding at a time. Therefore, data collected by the fan–in circuitry is normally never garbled by other, interfering data. However, if a part of the hardware malfunctions, the TCS master amputates it by disabling data from the failed hardware. In this way the TCS amputates the failed component and continues to use the remaining, enabled portions of the fan–in circuitry.

Transmit (Master to Slave) Direction

In the master to slave direction, eight copies of the TCS bus are driven by the TC/CLK clock card and sent out over the clock cables to the TC2000 midplanes, where the signal is distributed to eight function boards and a switch card pair. In each TC/SS–TC/SR pair, the TC/SS card holds the TCS slave that services the pair.

Figure 6–4 shows the fan–out of the TCS bus data from the master to the slaves. In a machine with more than eight midplanes, an additional layer of clock cards (that are necessary to fan out the clock signals) is used to fan out TCS bus data.

Figure 6–4 **Master to slave TCS bus fan–out.**


TC/TCS card in TCS master
clock card (TC/CLK) in switch
card's TCS slave
to midplanes
typical midplane
to TCS slaves on function boards
to TCS slave in TC/SS switch card

TCS Bus Transactions and Protocol

TCS communications refers to how the master and slave talk to each other, and what commands are implemented. The protocol supports monitoring and control functions of the general nature described here, but whose details are presented in TCS software and operations documentation.

All transactions on the TCS are initiated by the TCS master. A TCS slave never sends an unsolicited message. Each TCS message has an address and a command in the message header. All the slaves receive all the messages that pass over the TCS bus. Each slave examines each message to extract the address information. If the address of a message matches the slave's address, then the slave carries out whatever command the message indicates. If the message's address does not match the slave's, the slave processor commands its TCS bus receiver chip to ignore further characters until a new start of message arrives. This way all slaves constantly resynchronize their reception on the start of each message.

There are two types of address, one that addresses a particular slave, and another that addresses multiple slaves. A message that addresses more than one slave is called a broadcast message. Broadcast messages can be addressed to all circuit cards, or to all of the cards of a given type (such as all TC/FPV function boards). Broadcast messages are used during the power-up sequence for loading power-up and bootstrap code into boards of the same type simultaneously.

During normal operation, the TCS master periodically polls the slaves for status information. That status (one byte) indicates whether there are error conditions.

6.1.4 Fault Recovery

Two kinds of fault are protected against in the TCS — the TCS master hanging, and a TCS slave corrupting the TCS bus.

If the TCS master hangs, a watchdog timer will reset the TCS master processor, and it will subsequently reboot. TCS knowledge of the current dynamic state of the TC2000 machine is lost, so it must be reconstructed from the TCS master's configuration files, discovery through querying the TCS slaves, and/or operator input. Even without a full reconstruction, the TCS can be used for many of its control and monitoring functions.

If a TCS slave fails such that the TCS bus (in the slave-to-master direction) is corrupted, the TCS master commands the TCS slave above it in the fan-in tree to amputate the failed slave. This is described in section 6.1.3. Failure in the master-to-slave direction is less serious, because of two effects — the fan-out buffering limits (to one midplane) how much of the bus can be corrupted; and the input circuitry is unlikely to fail in a way that corrupts the common signal bus (but rather, to corrupt the value received by that slave).

6.1.5 TCS Slave Functions

The functions described here, performed by TCS slave processors, support the TCS tasks listed in section 6.1.1. The slave functions fall into two broad categories:

- Aspects of the slave's environment that it monitors or controls
  - Gate array chip interfaces
  - Card reset
  - Card temperature monitoring
  - Card voltage monitoring
  - Card voltage control
  - Card LED control
- The nature of the slave itself and its use of the TCS bus

- Addressing (identity)
- Configuration parameters
- Amputation from the bus

Gate Array Chip Interfaces

Each special gate array chip used to implement the Butterfly switch in the TC2000 computer has a simple interface to the TCS slave on its circuit card. The general nature of the interface is the same for the SIGA, LCON and SGA chips, while the details of what is controlled within the chip vary according to the chip's function.

The simple interface uses four signals: clock, data in, data out, and execute. The TCS slave toggles the *clock* line to shift data from the *data in* line into a shift register in the gate array chip, and to shift data from another register out onto the *data out* line. When the input register has been loaded with a command or parameter that the gate array should act upon, the *execute* line causes the action to be taken. A response is obtained by shifting out the contents of the output register.

Actions specific to each gate array include the following:

- Switch Interface Gate Array (SIGA), on function boards — set switch message transmission and reception parameters, set scaling of switch clock to Real Time Clock (RTC), set SIGA address on T–bus, read or write T–bus data, load special RAM registers (the "interleaver loader" mechanism)
- Level CONverter (LCON), on function boards — enable the function board's switch port and clock, amputate the function board from the Butterfly switch by disabling the board's switch port and clock, assert or sense any of the LCON pins connecting the board to the switch (for testing and diagnostics)
- Switch Gate Array (SGA), on switch cards — enable or disable any of the four input or output ports, set any bit on any output port (for testing), read any bit on any input port (for testing), read whether any input or output port is busy

Processor Reset Control

The TCS can reset a function board's processor by writing a register designated as the processor reset register. (For example, in the TC/FPV, the TCS slave can assert the 88100 CPU's reset line. It can also independently reset each SIGA and the board as a whole, though these are usually asserted simultaneously.) For example, the reset function is used to stop the processor while code is loaded into its memory.

Temperature Monitoring

The function board's temperature is monitored using a temperature transducer and support circuitry connected to one of the slave's on-chip analog to digital converters.

Voltage Monitoring

The slave monitors board voltages by connecting them, scaled and offset, to the slave's analog to digital converters. Each slave monitors +5, -4.5 and -2 VDC, and other voltages appropriate to the particular card type. Bulk power is not monitored directly, but is inferred from the other voltages.

Voltage Control

The slave controls the voltage converter block that converts bulk power to board power. For example, on the TC/FPV board, the slave can control the power block output voltage of the +5 VDC and the -4.5 VDC supplies. Each of these two supply voltages can be set to one of six levels:

Off
On (nominal voltage)
+5% nominal voltage
—5% nominal voltage
+10% nominal voltage
—10% nominal voltage

Circuit Card LED Control

An assortment of LEDs are visible on each circuit card while it is installed in the card cage, typically indicating:

- TCS VCC present (green)
- Bulk 48 VDC (±24 VDC) present (green)
- Board VCC present (green)
- Board VEE present (green)
- TCS flag (amber)
- Board-specific data

Of these, two are directly controlled by the TCS. One indicates "card TCS power on" and is connected with a resistor to the board's TCS VCC power. The other is controlled by the TCS slave and can be set on, off, or blinking. Blinking the LED on and off is one of the TCS slave's tasks, and does not require further intervention from the TCS master. Two blink rates are defined, fast (about 3 Hz) and slow (about 1 Hz).

The TCS flag amber LED is intended to point out boards that fail diagnostics, and as an aid to a service person in locating a particular board in a large system. The convention for use of this LED is as follows:

on	=	not yet initialized or cannot be initialized
slow blink	=	diagnostic in progress, or failure
off	=	operational
fast blink	=	board locater signal

A hardware reset causes the LED to turn on. The slave leaves the LED on at startup, so that an uninitialized or totally broken board will have its light on continuously.

The TCS master starts the LED blinking at the slow rate when it begins testing or configuration discovery. This indicates that the TCS has discovered the board, but has not yet approved it for use.

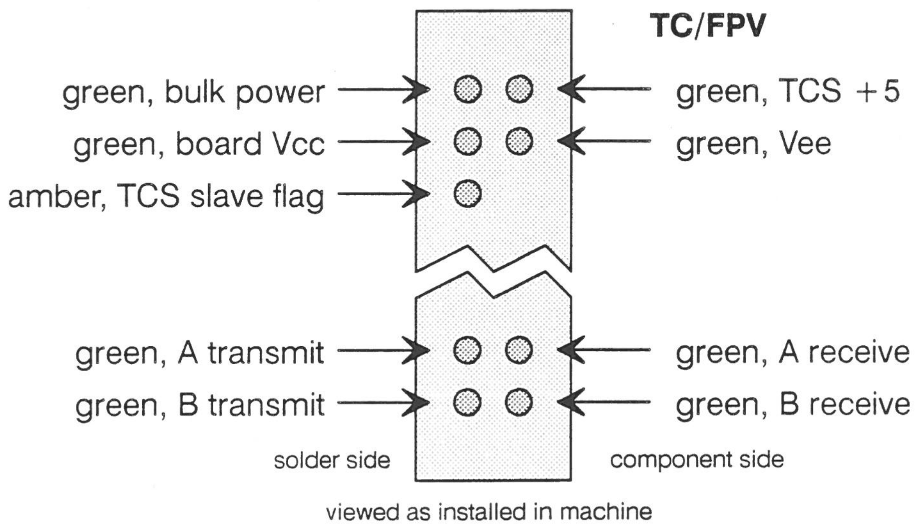
Once the TCS has completed diagnostics, the LEDs on boards that pass are turned off. Boards that fail or become non–communicative will continue to blink at the slow rate.

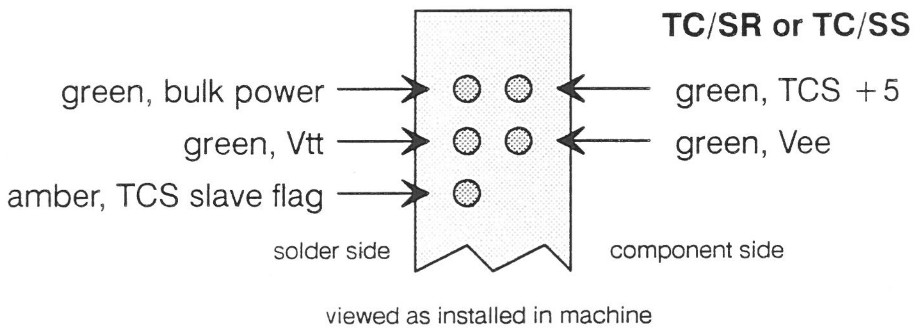
The TCS master provides a command that causes the LEDs on one or more slaves to blink at the fast rate to help a service person locate a particular board.

Note that the LEDs on each TC/SS–TC/SR pair of switch cards are both controlled by the same slave, on the TC/SS.

Figure 6–5 shows the LED indicators on the outside edge of TC/FPV function boards, and TC/SR and TC/SS switch cards.

Figure 6–5 **Function board and switch card indicators.**





The **transmit** (TX) and **receive** (RX) indicators are driven from the outgoing (into the switch) and incoming (from the switch) *frame* signals, and indicate that a message is being sent to or received from the switch. These indicators, therefore, give a crude measure of switch activity. For example, suppose none of these lights throughout the machine were brightly lit, except transmit on slot 3 and receive on slot 5, and the machine was not responding normally. A good guess is that something — either hardware or software — has gotten stuck sending lots of switch traffic from the function board in slot 3 to the one in slot 5. The **A** and **B** designations on these LEDs distinguish among two separate Butterfly switches, a provision for possible future capability. In the original model, only one switch is installed, so either all the A lights or all the B lights should be dark.

Slave Address Sensing

When a TC2000 function board or switch card is installed, eleven wires in the TCS slave interface stay high or are pulled low by the wiring on the card slot connector. This encodes that slot's TCS slave address. The slave processor compares this address to the address in TCS messages, in determining whether the message addresses this slave.

The TCS slave address is similar to the 9–bit *processor node number* that each function board obtains from the midplane: three bits each of bay, midplane and slot identity. For function boards, the bay, midplane and slot fields of its *TCS slave address* have the same values as those fields of its processor node number. The 9-bit scheme permits up to 512 function boards, the TC2000 design limit; but it leaves no addresses for switch or clock cards. Slot field values above the 0–7 range are used to address these cards in the TCS slave address format.

Slave Configuration Information

Several slave configuration values are written into the slave's EEPROM at the factory during final assembly and test. These values are listed below. The list below reflects the initial implementation. Because the definition of EEPROM contents is tied to TCS firmware and software, not to hardware, this list should be taken as highly suggestive but not definitive. For precise details, please refer to TCS software and operation documentation.

- **Board type** — Describes what kind of board this slave is on. This value is used to implement broadcast messages and to select board–specific slave routines.
- **Circuit card serial number**
- **Artwork revision level**
- **Electrical revision level**
- **TCS slave code (EEPROM) revision level**
- **Analog to digital converter calibration** — This is A/D converter calibration information for the 68HC11's analog to digital converters. This data calibrates the transducer systems that read the board voltages and temperatures. Having calibration values eliminates the need for precision voltage references and components.
- **Temperature alarm setpoint** — Temperatures above this value will signal an error condition in the slave status byte during polling.
- **Voltage within specifications setpoint** — Voltages deviating from nominal by more than this value, either above or below, will signal an error condition in the slave status byte during polling.

- **Timers** — Timeout values used by the slave program, including a T-bus access timeout.

Amputating a Slave from the TCS Bus

The slave on each TC/SS switch card can individually turn off the TCS slave-to-master serial signals from the eight function board slaves that are associated with that switch card. An 8-bit register accessible by the TC/SS slave has a bit corresponding to each of the function card slaves. This is described further in section 6.1.3.

6.2 Clock Card

The TC2000 machine contains one master clock card. Only if the machine is larger than 64 slots are slave clock cards required, so the description here concerns only the master clock card. It provides the following signals.

- Switch clock signals

  The clock signals are used by the SGA chips on switch requester and switch server cards, and also by the LCON and SIGA chips on function boards. The user sees their effect because they drive the Real Time Clock (RTC) in the SIGA. The clock signals may optionally be used as the board clock on function boards, although the TC/FPV uses its own clock instead.

  The clock signal for the requester half of the switch is separate from that for the server half; they are synchronized, but may be either in phase or 180 degrees our of phase, to adjust for switch data cable length.

  The source of the clock signal can be provided either by a fixed crystal, or by a programmable frequency synthesizer with a range of about 30 to 44 megahertz. The clock card's board impedance, routing and logic are carefully designed to reduce skew. Clock signals are distributed differentially.

- 65-millisecond pulse

  The 65-millisecond pulse, so called because its period is 65,536 microseconds, is used in synchronizing the RTCs throughout the machine. When this pulse occurs, the low half of the RTC is cleared, and the high half is incremented. This ensures that the low half of all RTCs is identical, so the system software has the comparatively simple task of synchronizing the high half.

- Hold

  The "hold" signal is used in the express message mechanism for bounding switch latency. It is distributed to all SGA chips in the machine, where it controls the SGA's returning of output ports from high priority status back to normal priority. This mechanism is described further in chapter 3.

- TCS bus

  The TCS bus provides communication between the TCS master and the TCS slaves on circuit cards throughout the machine. The clock card is at the head of the hierarchical tree that fans out data from the TCS master, and fans in data from the TCS slaves — the clock card itself connects directly to the TCS master. The TCS bus is described further in section 6.1.3.

The clock card TCS slave, besides normal temperature and power control/monitoring, can select the clock source, control and monitor the synthesizer, control and monitor the 65–millisecond and hold signals, and sense the position of on-board jumpers that control clock phase and termination.

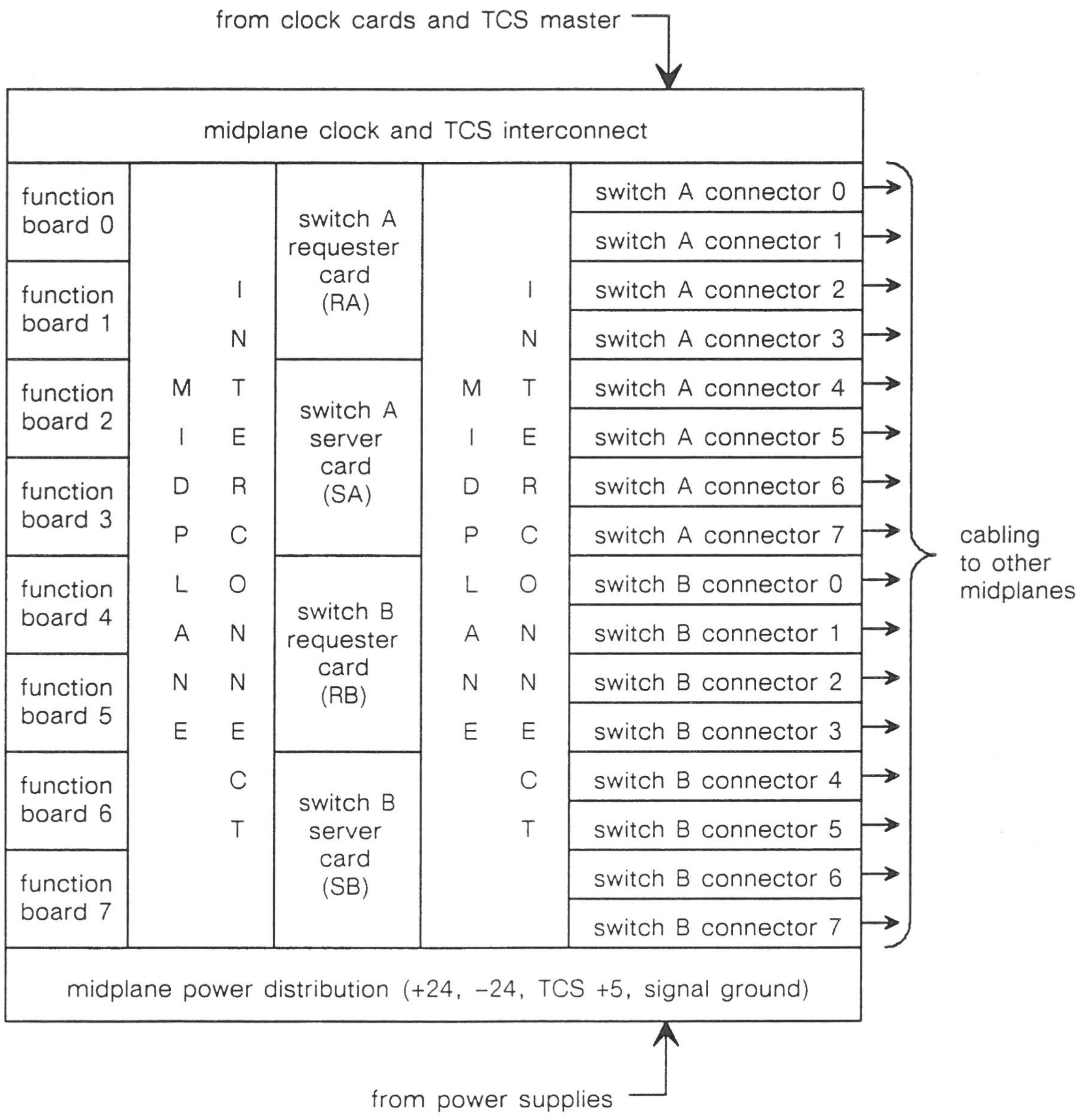
6.3 Midplane

The TC2000 midplane interconnects the following components:

- Eight function boards, such as the TC/FPV function board.
- Four switch cards: one requester—server pair, plus a second pair for the redundant switch.
- Two switch clock cables: the first cable for the primary switch clock, and a second cable for a redundant switch.
- Sixteen switch–to–switch data cables for connection to the rest of the machine: eight cables to switch cards throughout the machine, and a second set of eight for a redundant switch. In specific instances, some cables might be absent or replaced with TC/LOOP loop-back connectors.
- Power: +24 and -24 volts main power, +5 volts TCS power, and ground.
- Three ground straps to ensure a low–impedance, system-wide ground: one to the midplane to the left, another to the midplane to the right, and a third to an I/O midplane above the midplane (if any, such as the TC/VMP VMEbus midplane).
- Midplane ID DIP switch, uniquely identifying a particular midplane in a machine.

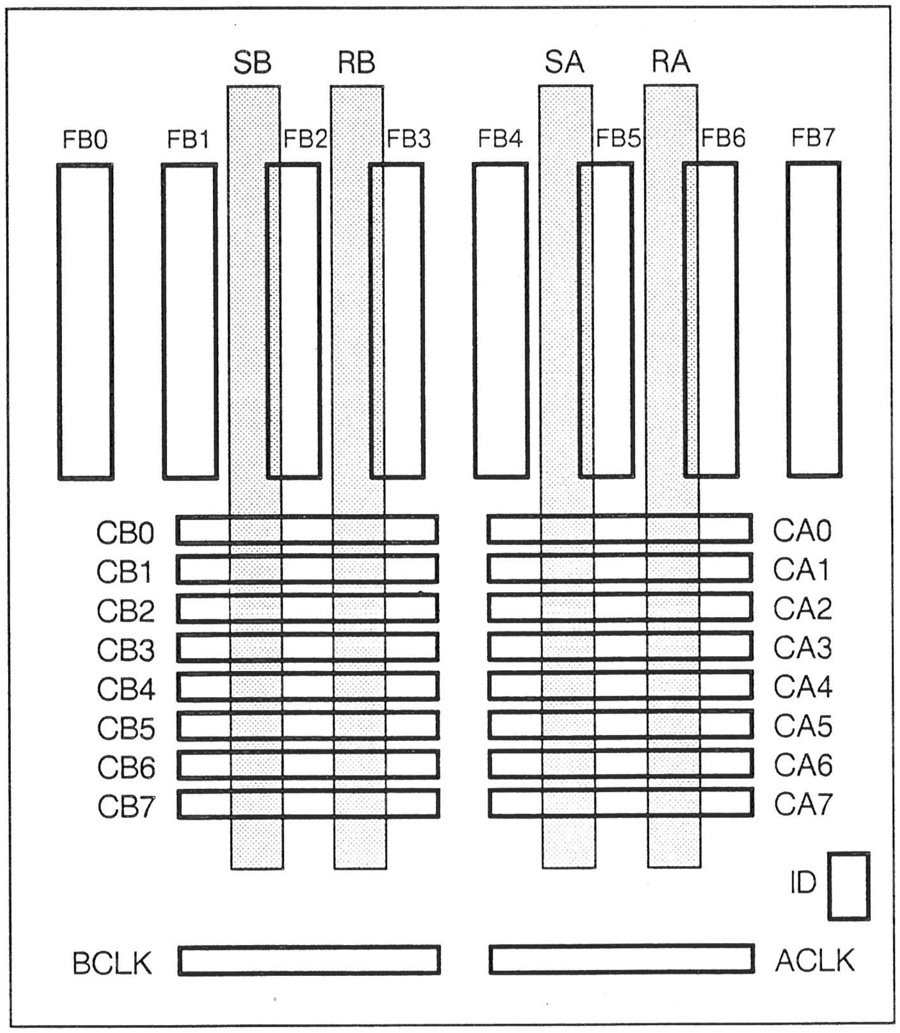
Figure 6–6 shows these interconnections in an abstract graphical form, illustrating how the midplane fits into the machine architecture.

Figure 6–6 Midplane interconnections.



The TC2000 midplane has connectors on both sides (hence the name "midplane"), controlled impedance signal lines (50 ohms), and no active components. Some of the complexity in the midplane wiring is because it supports a second Butterfly switch that, although not configured in the current version of the machine, may be provided as an option in the future. Figure 6–7 shows the layout of connectors on the midplane. The switch card connectors actually overlap the function board connectors in places.

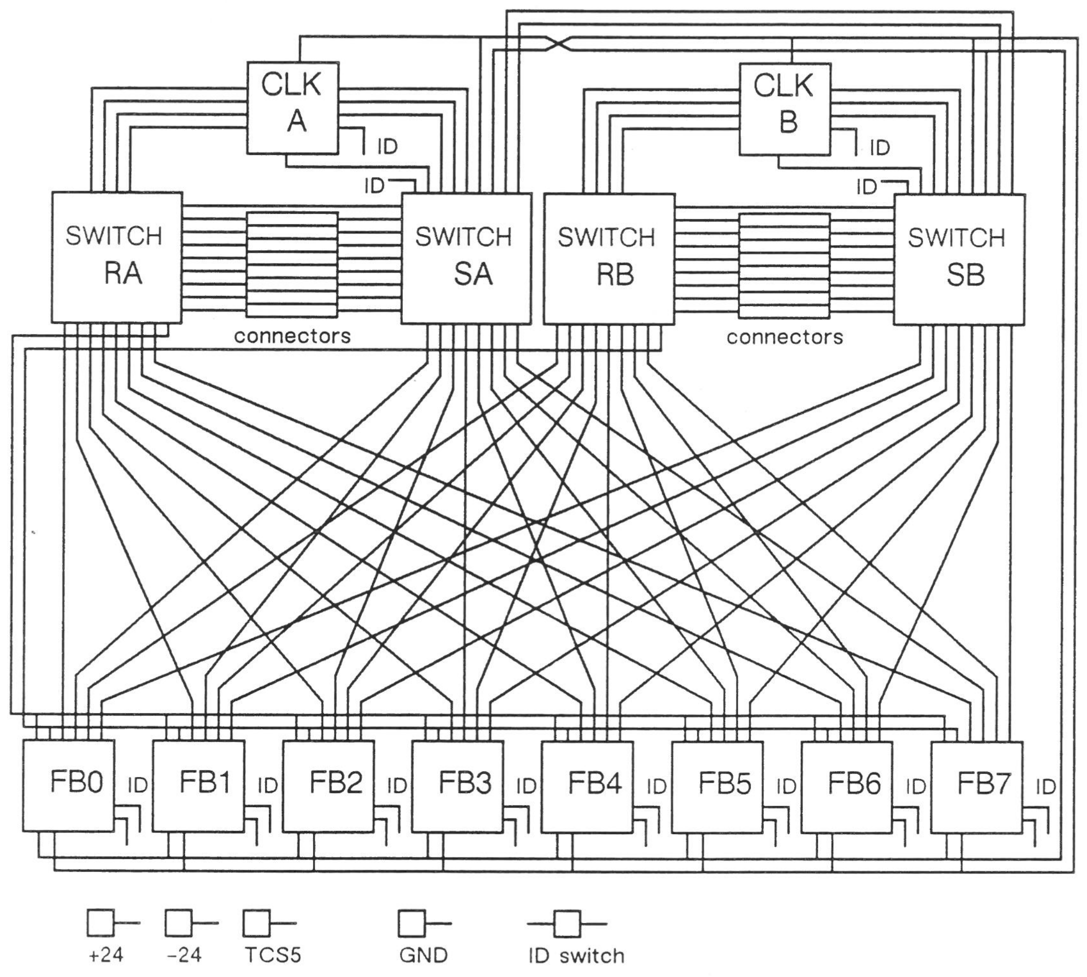
Figure 6–7 **Midplane connector layout.**



view from FUNCTION BOARD side

And Figure 6–8 is a block diagram of the midplane wiring. In this diagram, most of the connection lines represent several wires. Distribution of the ID switch wires to all cards, and of midplane slot ID wires to function boards, is indicated with the label "ID". Distribution of power and ground is merely suggested by the symbols at the bottom.

Figure 6–8 **Midplane block diagram.**



6.4 Power Supplies and Distribution

Power for the TC2000 machine is turned on in the following steps.

1. Assume that power cords are connected, circuit breakers are turned on, and the machine is configured for power control in the standard way. With the front panel keyswitch in the "off" position, power line power is going only so far as the Power Distribution Unit (PDU) in the each cabinet of the machine. (Low voltage power control signal is present on the power control wires, such as those to the keyswitch.)
2. The front panel keyswitch is turned to the "on" position. This turns on the utility cabinet PDU, which sends power to the TCS master, the TCS slave power supply, and any other utility cabinet equipment (such as I/O devices).

3. The TCS master turns on the PDUs in the other cabinets. In expansion cabinets, this applies power to the bulk power supply, which in turn applies ± 24 volts bulk power to the midplane and thus to all cards in the cabinet. In peripheral cabinets, this applies power to I/O devices and to VMEbus card cage power supplies.
4. The TCS master commands the TCS slaves on the various circuit cards to turn on their local card power supplies. This completes power-up of the machine.

The following sections discuss these steps and the equipment involved. For further details and instructions on the exact physical and electrical requirements, please refer to the *Site Installation Guide*.

6.4.1 Power Line

Each TC2000 cabinet has its own power cord. In the domestic (United States) systems, this is 208 VAC, 3 phase, Δ–wired, 30 amperes for the utility cabinet and for each expansion cabinet, and 20 amperes for each peripheral cabinet. International machines are wired differently (for example, they are Y–wired); consult the *Site Installation Guide* for such systems.

6.4.2 Keyswitch

The front panel keyswitch has positions "off", "on" and "secure". The "off" position disables all power to the entire machine when configured normally. (For servicing and diagnostic purposes, individual cabinets may be powered up separately.) The keyswitch "on" position allows the utility cabinet PDU to power up the utility cabinet equipment, including the TCS. The TCS can then power up the rest of the machine.

The keyswitch "secure" position is, for the purpose of power control, identical to the "on" position. The TCS master can sense which of these positions the keyswitch is in, and take appropriate software action. For example, certain operational or administrative commands may be ignored when the keyswitch is in the "secure" position.

6.4.3 Power Distribution Unit (PDU)

Each cabinet has one PDU. The purpose of the PDU is to accept the power cord and various power control signals (in the form of contact closures), and, based on the control signals, apply power line power to further equipment. Each PDU contains filters to reduce electromagnetic interference (EMI), circuit breakers, fuses, etc. The PDU's power control logic makes an all–on / all–off decision regarding powering the rest of the equipment in the cabinet; either no such power is applied, or (except if a branch circuit breaker is tripped or a branch fuse is blown) all such equipment is powered.

There are two kinds of PDU. One kind is used in the utility cabinet and in expansion cabinets; the other kind is used in peripheral cabinets. (In fact, there are differently wired PDUs for international machines. Counting this difference there are four kinds of PDU.)

Peripheral Cabinet PDU

In this kind of PDU, the power from the power cord first goes through an EMI filter and then a master circuit breaker, and then to three places.

- To unswitched 220 VAC outlets, protected by 10 A fuses.
- To the power supply for the power control logic, protected by a ⅛ A fuse.
- To a contactor (a relay for large amounts of power), controlled by the power control logic. The contactor supplies power to switched 220 VAC outlets, protected by circuit breakers.

The power control logic is driven by the following:

- The PDU box thermal protection cutout sensor.
- The local / off / remote toggle switch on the PDU. The "local" position tells the PDU to turn on, if the thermal protection sensor permits it. The "off" position keeps the PDU off. The "remote" position puts the PDU under control of the power control daisy chain, pending permission of the thermal protection sensor.
- The power control daisy chain.

The various equipment in the peripheral cabinet is plugged into the outlets on the PDU. This may include a VMEbus card cage power supply, tape drives, disk drives, etc.

Utility or Expansion Cabinet PDU

In this kind of PDU, the power from the power cord first goes through an EMI filter and then a master circuit breaker, and then to two places:

- To the power supply for the power control logic, protected by a ⅛ A fuse.
- To a contactor, controlled by the power control logic. The contactor supplies power to the following:
    - In expansion cabinets, the bulk power supply.
    - In expansion cabinets, the VMEbus card cage power supply (if installed).

  - In the utility cabinet, the ¼–inch tape drive.
  - Cabinet fans.

The power control logic is driven by the following:

- The cabinet thermal protection cutout sensors.
- The power control daisy chain.
- The drip screen interlock. The drip screen is a substance that, if the cabinet equipment were to catch fire and drip burning pieces, catches and extinguishes those pieces. If the drip screen is removed, an interlock detects this and prevents the PDU from turning on.

In this PDU, provision is also made to support a capacitor dump circuit in a future power supply system.

6.4.4 Power Control Daisy Chain

Each PDU has two power control connectors, so that it can not only receive a power control signal but also pass that signal on to another PDU. When a string of PDUs is thus connected, they are all controlled together.

The TCS master provides relay contact closures that drive the power control daisy chains in the machine. There are three daisy chains in a 64–slot machine:

- One chain controls peripheral cabinets. Many systems have only one peripheral cabinet, but if many I/O devices are required, additional peripheral cabinets are added to house the devices.
- Another chain controls expansion cabinets to the left of the utility cabinet (as viewed from the front). The control wires go from the TCS out to the farthest expansion cabinet (midplane 7.4), then from it to the one on its right, and so on. (Slot and midplane numbering is described in chapter 1.)

  The reason for this is so that the expansion cabinet at the end of the chain, namely midplane 7.7, is the one where the nX "master" or "king" function board resides. This function board is connected to the system boot disk, and the VMEbus system with that disk is in this expansion cabinet. Cabling in this way is a fail–safe procedure, since any disconnection along the chain will disable the machine from booting. Further, any powering down will be sure to remove power from the disk interface, ensuring that no erroneous disk I/O commands are issued.

  If there are fewer than four expansion cabinets to the left of the utility cabinet, the chain goes from the TCS out to the farthest expansion cabinet on the left, whatever that may be.

- The final chain controls expansion cabinets to the right of the utility cabinet. As with the second chain, this chain goes from the TCS out to the farthest expansion cabinet, and then back to the left, cabinet by cabinet.

  If there are four or less expansion cabinets in the machine, none are on the right of the utility cabinet, so there is no third power control chain in such a machine.

6.4.5 Bulk Power

Each expansion cabinet contains a bulk power supply. This supply obtains power from the cabinet's PDU and produces ±24 volts DC. Voltage regulation is not critical, since the bulk power is used only to power additional power supplies on each circuit card.

Bulk power from the supply is wired to the midplane, that distributes it to the switch cards and function boards in the cabinet.

6.4.6 Circuit Card Power

Each switch card and function board receives ±24 volts DC bulk power from the midplane. This is applied to on-board power supplies that produce the voltages required on the board. These board power supplies are under control of the TCS slave on the board, which can turn the power off, turn it on, and margin (adjust) it for testing. The margin voltages are +10%, +5%, −5% and −10% of nominal. (Certain supply voltages cannot be margined.)

Besides automatic protection mechanisms in the board power supplies, the TCS monitors board voltages and temperatures. If these exceed limits, the TCS will turn off the board power supplies.

Power for the TCS slave on each board is produced by a supply in the utility cabinet (+5 volts), and distributed to each midplane and thence to each TCS slave.

Board power for the switch requester and switch server cards is slightly complicated, since there is only one TCS slave for the card pair, and one of the voltages is supplied only on one of the cards and connected via the midplane to the other. However, this can be viewed as an implementation detail.

A

Floating Point Exception Handling

Floating point exception handling is of great interest to a few users — namely, those with programs that do intensive floating point computation with quantities that cause exceptions. Most users' programs either do not generate floating point exceptions, or do so so rarely that the cost to handle them is negligible. In those cases where it is a concern, however, the discussion below can clarify what the cost is and when it is incurred, and thereby aid the user in restructuring his algorithms to minimize the cost.

Probably the most common floating point exception relates to **gradual underflow**. As specified in the IEEE standard cited below, if the result of a floating point operation is too small to be represented by a normalized number with the smallest exponent value, then zero bits start filling the high end of the mantissa to create a denormalized number with less precision.

Because floating point exceptions, particularly the IEEE gradual underflow mechanism, are handled not in hardware but in software, the execution performance can be significantly affected. In most cases, most of the cost is trapping into and exiting from the kernel, with a relatively small amount due to handling the floating point exception itself. However, once a denormalized number has been created, any ***further*** processing of it also goes through the exception handler.

Floating point exception handling is more a software issue than most topics in this book, but is discussed here because it lies in the grey area between how the hardware works and what the software does.

The pSOS^{+m} floating point exception handler is the same as that under the nX operating system, and the pSOS^{+m} user may optionally replace it with his own.

A.1 Introduction

This document presents the design of the floating point exception handling routines for the TC2000 nX kernel. The Motorola 88100 requires special attention to floating point because it generates exceptions rather than completing all operations itself.

The following documents contain detailed information relevant to this discussion:

Motorola *MC88100 RISC Microprocessor Unser's Manual*

IEEE *Standard for Binary Floating–Point Arithmetic* (ANSI / IEEE Standard 754–1985)

nX operating system source code — files reg.h, locore.s, trap.c and fpe.s

Motorola *MC68881 Floating–Point Coprocessor User's Manual*, pages 2–19 and following (for discussion of floating point number formats)

A.2 Floating Point Numbers

A.2.1 Formats

There are two floating point number formats on the 88100, 32–bit single precision and 64–bit double precision. Each consists of a sign bit, a biased exponent and a mantissa. For "normalized" numbers, the mantissa is considered to have a hidden one bit which is not explicitly represented.

The highest and lowest values of the biased exponent in each format represent special cases. The highest is either infinity, if the mantissa is zero, or Not–a–Number otherwise. A zero exponent represents zero if the mantissa is zero, or else a denormalized number. These special cases cause most of the complexity in handling floating point exceptions; the 88100 does not deal with them itself.

For complete details of both formats, see Motorola's 68881 manual, pages 2–19 and following.

A.2.2 Operations

The floating point unit executes floating point FADD, FSUB, FMUL, FDIV, FCMP, FLT, INT, NINT and TRNC instructions, as well as integer MUL, DIV and DIVU instructions. All the floating instructions can result in exceptions for certain values of their operands. The integer instructions can generate separate exceptions, but we will ignore those here.

The results of floating point operations can depend on the rounding mode specified in the floating point control register.

The IEEE Standard requires implementation of four modes — round to nearest, round to zero, round to plus infinity, and round to minus infinity. Rounding of normal operations is done by the 88100, but correct exception processing must also consider the rounding mode to produce the proper result.

A.3 Floating Point Exceptions

The floating point unit generates precise and imprecise exceptions. These are indicated by the Floating Point Exception Cause Register and the Operation Type Registers.

Precise exceptions for the Floating Point Unit are:

```
. Unit Disabled
. Unimplemented Opcode
. Privilege Violation
. Integer Conversion Overflow
. Reserved Operand
. Divide by Zero
```

Imprecise exceptions are:

```
. Underflow
. Overflow
. Inexact
```

The first three, FPU disabled, unimplemented opcode and privilege violation, require no work, and result in a signal to the offending process. The inexact exception only occurs when a process has enabled it explicitly; it is handled by the process directly. The remaining five are the subject of what follows.

For each exception, the handler saves the state of the processor. The exception frame contains, among other things:

```
. General Purpose Registers 1 - 31
. Floating Point Control Unit Registers
  Exception Cause Register
  Precise Operation Type Register
  Source 1 Operand Registers
  Source 2 Operand Registers
  Imprecise Operation Type Register
  Result Registers
  User Status Register
  User Control Register
```

Each exception processing routine modifies the stored value(s) in the appropriate register(s) to reflect the result of the operation which caused the exception, and returns a bit mask to indicate the exception status. Not all events handled by these routines are "exceptions", according to the IEEE Standard.

A.3.1 Overflow Exception

Overflows produce a result of infinity or the largest non–infinite number, depending on the rounding mode and the sign. The correct value is loaded from a table and stored into the saved image of the users registers. The overflow exception bit is always returned.

A.3.2 Underflow Exception

Underflows result in either zero or a denormalized number. The exception handler first needs to undo any rounding already done by the Floating Point Unit, right shift the mantissa to denormalize it, and round the result. Rounding depends on the sign, the rounding mode, the LSB of the mantissa and the bits shifted off the right end of the number.

A.3.3 Divide by Zero Exception

Dividing zero by zero results in a NaN ("not a number", defined as the exponent field being zero) and returns the invalid operation bit. Dividing anything else by zero results either in infinity or the largest number (with the proper sign) and returns the divide by zero bit.

A.3.4 Convert to Integer Exception

The 88100 signals a Convert to Integer exception when one of the conversion instructions, INT, NINT or TRNC, encounters an operand with an unbiased exponent of 30 or greater. A positive number with an exponent of 30 will overflow only if it is equal to 0x7FFFFFFF (the largest positive number) and rounding adds one to it. All larger positive numbers overflow. No negative number with exponent of 30 will overflow, but an exponent of 31 or greater will overflow unless the number equals 0x80000000 and rounding doesn't add to it. Overflows result in the largest number of the proper sign, and return the invalid operation bit.

A.3.5 Reserved Operand Exception

All the reserved operand exceptions take significantly longer to process than the other Floating Point Unit exceptions. Some reserved operand exceptions get a second exception (typically underflow, rarely overflow) during processing.

When the floating point unit attempts to execute any instruction whose source operands are infinity, NaN or denormalized, it causes a reserved operand exception. Each type of operand, and each different instruction, requires separate handling.

In general, except for INT, NINT and TRNC instructions, processing begins by checking for NaNs. FCMP results in the "nc" bit ("not comparable") being set; all other instructions propagate the NaN as their result. Signalling NaNs return the operand exception bit while other NaNs do not. (Signalling NaNs are defined as NaNs with the high bit of the mantissa zero. For these, the exception handler generates a UNIX–style signal to the process.)

Once NaNs are processed, any single precision operands are converted to double precision. Finally, each instruction is handled separately. The general strategy is to modify the operands so that a reserved operand exception can no longer occur and then use the floating point unit to produce the proper result.

INT, NINT and TRNC Instructions

If the number is denormalized the result is usually zero. However, if the rounding mode is toward infinity and a denormalized number has the right sign, an INT instruction will round it to 1 (with the same sign). Therefore, for INT, we make a small normalized number and let the FPU do the rounding. NINT and TRNC always result in zero for denormalized numbers. No exception bit is returned.

For NaN or infinity the result is the largest integer of the same sign (technically, the sign of a NaN is meaningless). The reserved operand exception bit is returned.

FCMP Instructions

Since NaNs have been intercepted already, the FCMP must always produce the result of the comparison without returning an exception bit. This is done by modifying the operands. Infinities are changed to large finite numbers and denormalized numbers are converted to small normalized numbers. In each case the other operand may also be made smaller/larger to ensure that the sense of the compare remains the same.

FADD and FSUB Instructions

FADD and FSUB are handled together by inverting the sign of the second operand for subtraction. If either operand is zero the other operand results. If either operand is infinity it is the result unless we are adding infinities of opposite sign. In that case, the result is a NaN and the invalid operand bit is returned.

For denormalized numbers, we make the (biased) exponent 64, giving us a (larger) normalized number. Unfortunately, this also gives us a hidden bit which we then subtract away. All this has the effect of multiplying the original number by 2**63. We compensate for this by multiplying the other operand by 2**63 also, unless it is greater than 1. If the other operand is already greater than 1 the addition of a very small number will only affect rounding, and we need to avoid the possibility that multiplying by 2**63 will cause an overflow.

Once this has been done, we issue a FADD instruction. [NOTE: This could overflow, generating another exception which is processed transparently as far as the FADD routine is concerned.] Finally, we scale back down, if necessary, by multiplying by 2**–63. [This could underflow.] The result is stored in the copy of the users register(s), and the exception bits, set by any overflow or underflow exception, are returned.

FMUL Instructions

FMUL first deals with zeros and infinities. Zero times infinity results in a NaN and returns the invalid operand bit. Infinity times anything else results in a properly signed infinity.

Again, denormalized numbers are converted to normalized numbers, effectively multiplying them by 2**63. The other operand is multiplied by 2**–63 unless it is already very small. Finally, we do the FMUL, store the result in the users register(s), and capture the exception bits which may have been set if the FMUL underflowed.

FDIV Instructions

FDIV is processed very much like FMUL, with minor differences. Infinity divided by infinity results in a NaN. We need not check for zero divisor because that would have resulted in a different exception. We compensate for modifying a denormalized number by multiplying the other operand by 2**63. Finally, we do the FDIV, store the results, and capture any exception bits set by an underflow.

Index

Dear Customer:

To do a better job of serving you by providing improved documentation, we are asking you to help us. Please take a few minutes to answer the following questions and return them to us by folding up and taping this questionnaire. We value your comments and suggestions and appreciate the time you take to send them to us. Thank you.

1. What is your position/function? ☐ application user ☐ system manager/administrator ☐ programmer ☐ service engineer ☐ other

2. How many years have you been working with computer systems? ____________

3. What system are you currently working on? (please check both **hardware** and *software*) ☐ **GP1000** ☐ **TC2000** ☐ *Mach 1000* ☐ *pSOS 1000* ☐ *nX* ☐ *pSOS^{+m}*

4. What application are you using the product for? ______________________________

5. How much of your time do you spend reading or referring to documents? ☐ 10% ☐ 20% ☐ 30% ☐ 40% ☐ 50%

- -

Please answer the following questions about: Inside the TC2000 Computer

6. Is the information accurate?	☐ always	☐ mostly	☐ seldom	☐ never
7. Is the material clear and logical?	☐ always	☐ mostly	☐ seldom	☐ never
8. Is it easy to find the information you need?	☐ always	☐ mostly	☐ seldom	☐ never
9. Does the index contain the words you look up?	☐ always	☐ mostly	☐ seldom	☐ never
10. Is the information located where you expect it in the book?	☐ always	☐ mostly	☐ seldom	☐ never
11. Are the illustrations and examples adequate?	☐ always	☐ mostly	☐ seldom	☐ never
12. Did you need information not available in this book or set of books?	☐ always	☐ mostly	☐ seldom	☐ never

13. What sections of the book did you use the most?

14. What areas need more or better examples? (Please list page numbers.)

May we contact you for more information? If so, please give your name, address, and telephone number.

Do you have additional comments? If so, please write them on the reverse side.

After completing the form, **fold this end up to the dotted line**, fold down and tape the top, stamp, and mail. Thank you.

Additional Comments:

Documentation Department
BBN Advanced Computers Inc.
10 Fawcett St.
Cambridge, MA 02138

Tape here