VAX 9000 Family System Technical Description

Order Number EK-KA90S-TD-001

digital equipment corporation
maynard, massachusetts

First Edition, May 1990

BI	KDM	RSTS	VAX FORTRAN
CI	KLESI	RSX	VAX MACRO
DEC	MASSBUS	RT	VAXBI
DECmate	MicroVAX	RV20	VAXcluster
DECUS	NI	RV64	VAXELN
DECwriter	PDP	TA	VMS
DHB32	P/OS	TK	VT
DIBOL	Professional	ULTRIX	Work Processor
DRB32	RA	UNIBUS	XMI
EDT	Rainbow	VAX	digital™
KDB50	RD	VAX C	

This document was prepared and published by Educational Services Development and Publishing, Digital Equipment Corporation.

Contents

2 Technology and System Packaging

3 CPU Subsystem

5 SPU and Scan Subsystem

6 Power and Control Subsystem

Tables

About This Manual

This manual provides an overview of the VAX 9000 family of computer systems. It is a reference for Customer Services personnel as well as a training resource for Educational Services.

Intended Audience

The content, scope, and level of detail in this manual assume that the reader:

- Is familiar with the VAX architecture and VMS operating system at the user level
- Has experience maintaining midrange and large VAX systems

Manual Structure

This manual is divided into eight chapters, an appendix listing related documents, a glossary of terms that are introduced in this manual, and an index.

- Chapter 1, Introduction, is a high-level overview of the basic functions of each major subsystem in the VAX 9000. Chapter 1 introduces the major system design, reliability, and maintainability features, and it describes basic physical and functional characteristics. This chapter also explains the basic system configurations, including memory and I/O.
- Chapter 2, Technology and System Packaging, describes the new system technologies, including system packaging and cooling. This chapter also describes the system and I/O cabinet configurations, including cabinet contents and major component locations.
- Chapter 3, CPU Subsystem, describes the CPU at a functional unit level, and it includes the functional units comprising the CPU. This chapter introduces each functional unit — the clock subsystem, IBox, MBox, EBox, and VBox — with a description of the major functions and an operational overview.
- Chapter 4, System Control Subsystem, introduces the system control unit (SCU) and the main memory subsystem. The SCU introduction describes the logical partitioning of the unit. It also describes the functional blocks of the SCU: the junction box (JBox), the I/O control unit (ICU), and the array control unit (ACU). Chapter 4 also describes the SCU planar module and the physical layout of the MCU/MCA.
- Chapter 5, SPU and Scan Subsystem, describes the basic functions of the service processor unit (SPU). It provides an operational overview of the SPU with an introduction to the scan system. Chapter 5 describes the hardware and software features of the SPU, and it provides a physical and functional description of the SPU.

- Chapter 6, Power and Control Subsystem, provides an introduction to, basic specifications for, and a functional overview of the power subsystem. The overview includes descriptions of power distribution and control, battery backup operation, and environmental monitoring and control. This chapter also describes the operator control panel (OCP).
- Chapter 7, I/O Subsystem, provides a functional overview of the I/O subsystem hardware components. This chapter includes a summary of typical I/O configurations, configuration rules, supported devices, adapters, and controllers.
- Chapter 8, System Exception and Interrupts, describes how the system handles interrupts and exceptions. This chapter provides an introduction to the functional units that handle interrupts and exceptions, memory faults, reserved opcodes, and reserved addressing faults.
- Appendix A, Related Documents, provides a list of documents, and order numbers, related to the VAX 9000 family of systems.
- The index provides an alphabetical list of topics described in this manual. An entry with an *f* appended to the page number (for example: MCUs, location of MCAs, 3–15f) indicates a figure reference. An entry with a *t* appended to the page number (for example: Tag STRAMs, for translation buffer, 3–16t) indicates a table reference.

1 Introduction

This chapter provides an introduction to the basic functions of each major subsystem in the VAX 9000 model 400 and 200 systems. Topics include:

- Major system design, reliability, and maintainability features
- New technologies
- Basic physical and functional characteristics
- Basic system configurations, including memory and I/O

1.1 System Overview

The VAX 9000 model 200 systems and the VAX 9000 model 400 systems are built from the same or similar components. The differences in the systems are mainly in packaging. The VAX 9000 family extends the range of the VAX family into the high end of performance by offering different configurations that provide performance levels between 30 VUPs and 108 VUPs.

NOTE
A VUP, VAX unit of processing, is equivalent to the performance of one VAX-11/780 system. Therefore, 21 VUPs is the performance equivalent of 21 VAX-11/780 systems.

The high performance is achieved through high-density packaging, pipelined instruction execution, and enhanced characteristics of the memory subsystem and the I/O subsystem.

All systems use a system control unit (SCU) to interconnect single or multiple processors, memory, and I/O subsystems. The I/O subsystem uses the XMI bus to support the corporate interconnect architectures: BI, CI, and NI.

Both systems support the VMS operating system, which includes symmetric multiprocessing (SMP) functionality, as their primary operating system. Both systems also support the ULTRIX operating system. The VAXELN operating system is supported on the service processor (console) subsystem.

1.1.1 System Features

Maximum system availability was one of the major design objectives of the VAX 9000 family of systems. Extensive data integrity checking, error checking, reporting, and logging mechanisms are found throughout the CPU. The goal is to detect intermittent failures, report them to the operating system and, whenever possible, invoke recovery procedures to minimize system crashes.

The system design allows error analysis, recovery, and maintenance to be performed on one CPU while other CPUs of the system continue to function under the operating system. Logic and power partitioning for the model 440 allows one pair of CPUs to be deselected for maintenance, while the remaining CPU pair continues to operate.

The system contains a service processor unit (SPU) that acts as the console front-end processor. The SPU consists of a combination of hardware, software, and firmware that provides maintenance and error functions. The SPU is the focal point of all system errors and includes extensive on-line error recovery and fault isolation capabilities. The SPU has scan paths into the SCU and CPUs that allow it to read the state of over 20,000 key internal logic signals to determine faulty operation. The SPU is also the service engineer's primary tool for system test diagnostics and fault isolation.

1.1.2 Common System Technologies

The VAX 9000 model 400 and 200 systems use the same basic hardware technologies. The new technologies used to implement the systems include:

- Third-generation ECL gate arrays called macrocell arrays (MCA III)
- Advanced multichip packaging called a multichip unit (MCU)
- Self-timed random access memory devices (STRAMs)
- Self-timed register file devices (STREGs)
- High density signal carrier (HDSC)

The logic is implemented in ECL gate arrays (MCA III) that offer twice the speed and eight times the density of those used in the VAX 8000 family, but using fewer chips. With the reduction of the number of chips, propagation time for logic signals is reduced, resulting in higher processing speed.

Self-timed storage devices (STRAMs and STREGs) are similar to standard RAM and register storage devices in operation, but they contain on-chip latches on the inputs and outputs and on-chip clock pulse generation circuits. The design of these storage devices reduces access time and their impact on clock skew.

The MCA III, STRAMs, and STREGs are packaged in MCUs. Up to eight MCAs or a combination of MCAs, STRAMs, and STREGs are packaged in each MCU. The interconnect of the logic elements in each MCU is provided by a high density signal carrier (HDSC). The HDSC is similar in concept to a printed circuit board but is a wafer-based interconnect that uses advanced semiconductor fabrication techniques. With over 600 signal lines per inch, the HDSC has over 30 times the density of most advanced printed circuit boards. The HDSC allows logic components to be packaged very closely, shortening interconnects and minimizing propagation delays between components.

The minimum configuration for each system requires a single scalar processor. Each processor is implemented on a single 24 × 24-inch planar module. The planar module accommodates up to 16 MCUs. The scalar processor occupies 13 MCUs. The remaining three MCU slots are used for adding an optional vector processor.

1.2 Performance Characteristics

In the VAX 9000 family, high performance is accomplished by using several levels of parallelism. At the system level, there is tightly coupled multiprocessing. Through problem decomposition, up to four processors can work on a task and communicate efficiently through shared memory.

Multiple I/O buses provide parallel paths to mass storage and other devices, resulting in high speed, extensive connectivity, and redundancy. Performance is further enhanced by using quadword wide data interfaces at critical points throughout the system.

Parallelism is accomplished in the processor by pipelining. Pipelining separates the execution of a macroinstruction into small, individual operations. These individual operations are then performed by dedicated and independent functional units that have been optimized for a particular operation. The objective is to keep the computing resources as busy as possible, with minimal wait time while information is being retrieved from memory or the I/O subsystem.

As a result, the execution operations can be overlapped, increasing total instruction throughput. Figure 1–1 compares the relative instruction time of a nonpipelined processor and a pipelined processor.

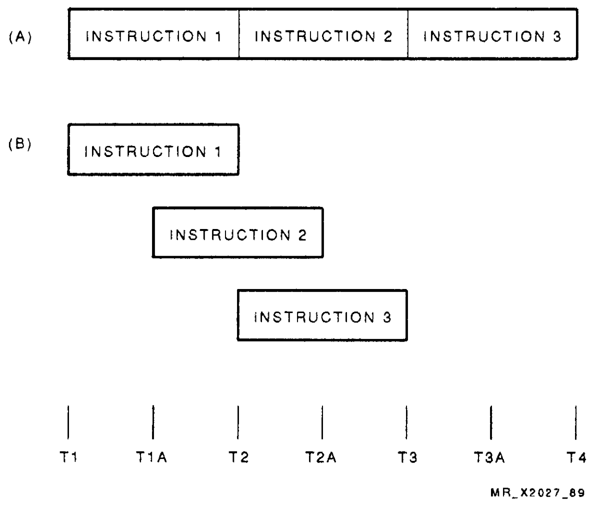


Figure 1–1 Nonpipelined and Pipelined Instruction Execution

The system contains hardware that allows the pipelines of the functional units to operate independently of one another. It also has hardware that detects and resolves most pipeline hazards.

To allow independent operation of the pipelines, queues are placed between the major functional units. The result of one functional unit is placed in a queue so that the next functional unit can access the result when it is available. After placing the result in the queue, the unit is free to continue operation. It does not have to wait for the second functional unit to complete an operation.

For example, the instruction unit (IBox) fetches instructions, decodes instructions, and evaluates the specifiers of the instructions. When the evaluation of the specifiers is complete, they are passed to the execution unit (EBox) so that the EBox can execute the instructions. If the EBox is busy, the IBox waits to pass the specifiers. But by placing queues or buffers between the pipelines of the IBox and EBox, the IBox can continue operating.

The pipelines are relatively short, helping to reduce conflicts (stalls). Conflicts occur when:

- The EBox ALUs execute floating-point instructions, integer multiplication, and division. These instructions require multiple cycles. Therefore, instructions requiring their results cannot be issued until the cycles are complete.
- One instruction writes to a memory location or general-purpose register (GPR) and a following instruction attempts to read that location. The read is stalled until the write is complete.
- The result of one instruction is used to form an address for a subsequent instruction. The address cannot be formed until that result is available.

To increase speed and signal integrity, most connections in the systems have single sources, and all destinations are on a single substrate. Unlike other VAX systems, there are no internal buses in the systems.

1.3 Major Subsystems

The VAX 9000 family has several major subsystems:

CPU
Clock
Service processor and scan subsystem (SPU)
System control unit (SCU)
Memory
I/O
Power

The following sections each provide a brief description of a subsystem.

1.3.1 CPU Subsystem

The CPU has four functional units: memory unit (MBox), IBox, EBox, and vector unit (VBox). These four functional units are connected to main memory and the I/O system through the SCU. Figure 1–2 shows a simplified block diagram of the system and Figure 1–3 shows a simplified block diagram of the system at maximum configuration.

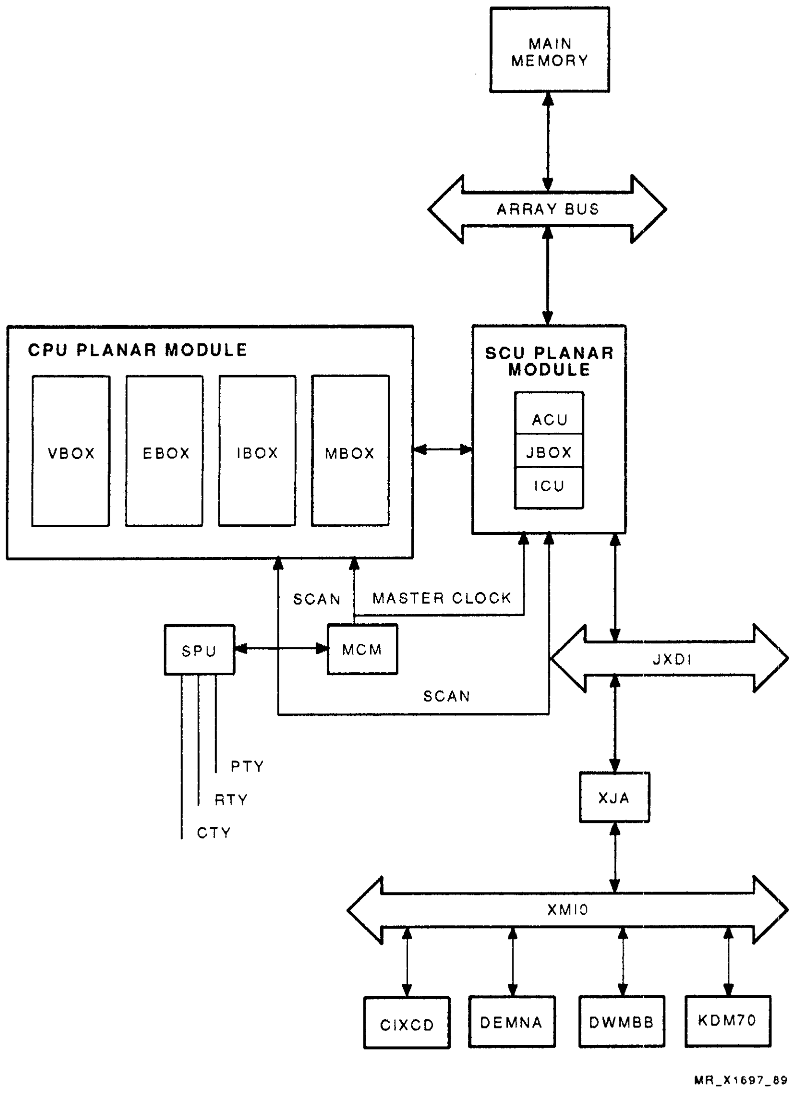


Figure 1–2 Basic System Block Diagram

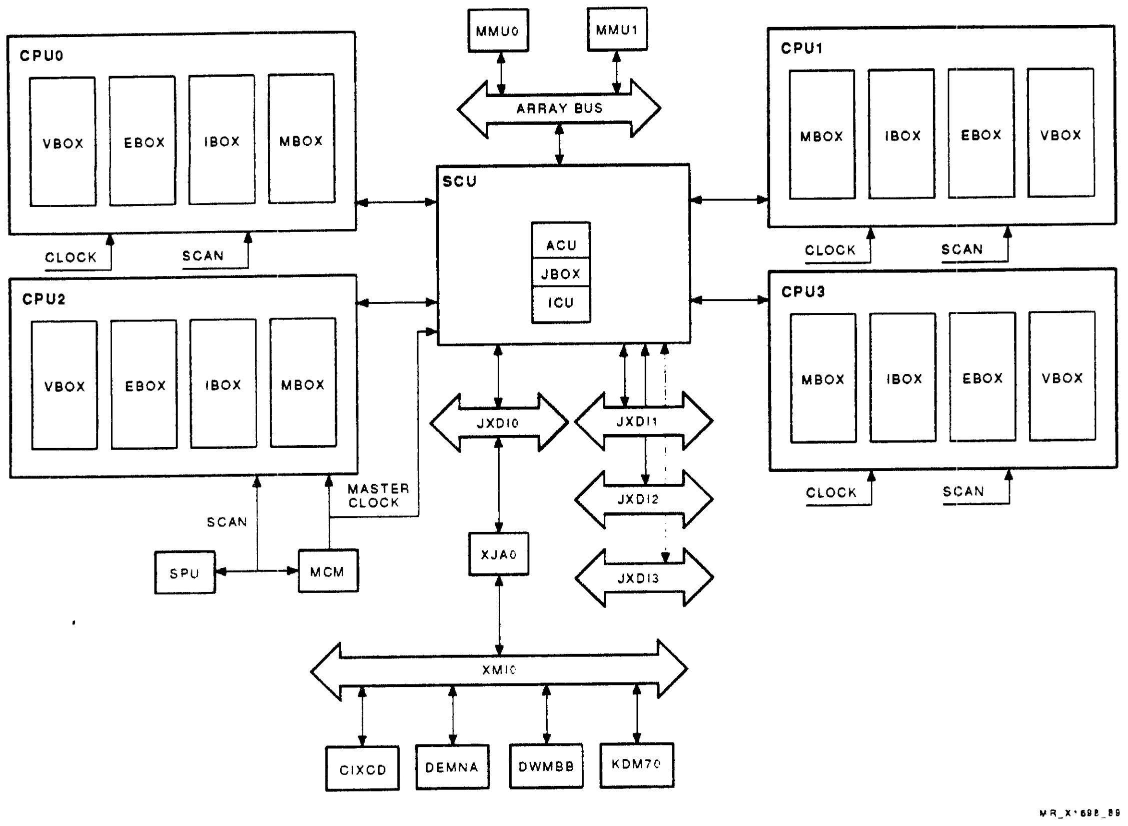


Figure 1–3 Quad CPU System Block Diagram

1.3.1.1 MBox

The MBox interfaces to the SCU, the EBox, and the IBox. The SCU interface allows communication between the CPU and main memory, the I/O subsystem, and other CPUs (in a multiple CPU configuration). The MBox contains a translation buffer, a 128-Kbyte data cache, and a translation buffer fixup unit. The translation buffer fixup unit resolves translation buffer misses by accessing memory management registers and fetching valid page table entries.

The MBox accepts virtual memory references and translates the virtual addresses to physical addresses. The MBox accesses memory data either locally in the data cache or indirectly through the SCU to main memory.

1.3.1.2 IBox

The IBox prefetches instructions, decodes opcodes and operand specifiers, fetches operands, and updates the program counter. The IBox performance is increased with the implementation of a virtual cache for storing prefetched instructions, a large instruction buffer, and by a decode unit that can decode up to three operand specifiers in a single cycle.

The IBox also contains a unique scheme for handling branch instructions. A cache is used to store the history of recently executed branches. When the same branch instruction is encountered, this history cache is used to determine whether to take the branch. When the branch instruction is not stored in the cache, an opcode-dependent bias determines which way to branch and then the prediction is loaded into the cache.

1.3.1.3 EBox

The EBox serves as the CPU execution unit, and it accepts instruction source and destination data from the IBox and MBox. It also provides integer, floating-point, divide, and multiply operations. The EBox contains four processing units capable of parallel operation. Most of the processing units are pipelined and can accept inputs while processing a previous operation.

The EBox passes result data to internal GPRs or to the MBox for subsequent transfer to the data cache, memory, or I/O.

1.3.1.4 VBox

The VBox is an optional vector processor that connects to the scalar processor. The unit accepts source and destination data from the IBox through a 32-bit load path of the EBox.

The VBox contains an adder, multiplier, and a mask unit for performing logical functions. The unit can produce a double-precision result every cycle, and it passes result data to the MBox through a 64-bit result data path.

1.3.2 Clock Subsystem

The clock subsystem generates and distributes system clock signals to the CPU and SCU planar modules and to the I/O subsystem. Clock generation is performed by a microwave frequency synthesizer that provides two clock outputs: master clock and reference clock.

The master clock operates at a programmable frequency between 332 MHz and 580 MHz. All system timing is referenced to the rising edge of the master clock. The reference clock is generated at a frequency of one-eighth of the master clock. The reference clock defines the beginning of a machine cycle and provides overall synchronization.

Clock signals are distributed to the planar modules with semiflexible coaxial cables. The HDSC of the planar module contains a dedicated layer for clock signals. The clock signals are delivered to clock distribution chips (CDCs) on each MCU.

The clock interfaces with the SPU, providing status information and receiving control signals. The SPU can change the operation frequency of the clock, stop and start the clock, and perform other clock maintenance functions.

1.3.3 Service Processor and Scan Subsystem

The service processor unit (SPU) contains four MicroVAX II processors located in the IOA cabinet of the model 200 systems and in the CPA cabinet of the model 400 systems. The SPU is a subsystem based on the BI adapter and contains an RD54 disk drive and TK50 tape drive for console loading and storage. The SPU is responsible for loading CPU microcode and initializing the system. The SPU also contains the interfaces that support local and remote user access. Figure 1–4 shows a simplified block diagram of the SPU.

In addition to providing the normal console features, the SPU also plays a major role in error detection and recovery for the system. The scan system of the SPU has access to over 20,000 points internal to the CPU and SCU and, when errors occur, uses these access points to test and diagnose the failing CPU or SCU. When an error occurs, the SPU can stop the system clocks, read the state of the system, and determine if the error is correctable. If so, the SPU can invoke error handling routines and then restart the system, allowing normal system operation to continue without operator intervention.

The SPU also has an interface to the power and environmental monitor (PEM). The PEM provides status and monitors information from the power and environmental system.

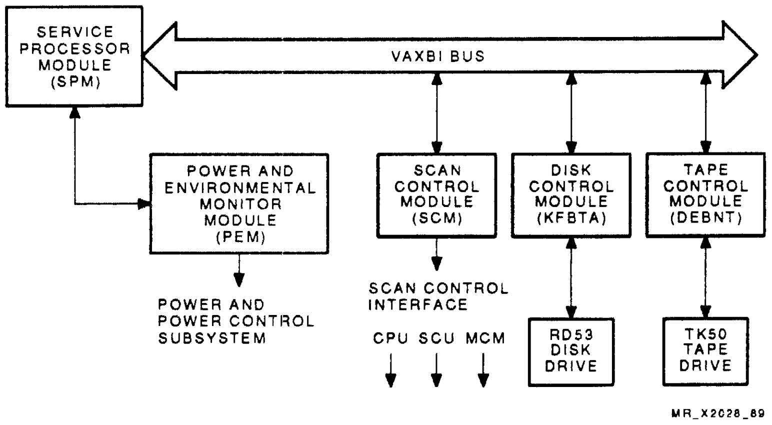

SERVICE PROCESSOR MODULE (SPM)
VAXBI BUS
POWER AND ENVIRONMENTAL MONITOR MODULE (PEM)
SCAN CONTROL MODULE (SCM)
DISK CONTROL MODULE (KFBTA)
TAPE CONTROL MODULE (DEBNT)
POWER AND POWER CONTROL SUBSYSTEM
SCAN CONTROL INTERFACE
CPU SCU MCM
RD53 DISK DRIVE
TK50 TAPE DRIVE
MR_X2028_89

Figure 1–4 SPU Simplified Block Diagram

1.3.4 System Control Subsystem

The system control unit (SCU) is comprised of three functional units: the I/O control unit (ICU) that manages the I/O subsystem, the array control unit (ACU) that manages the main memory system, and the JBox that manages the flow of data between the CPUs and I/O and memory. Figure 1–5 shows a simplified block diagram of the SCU.

The JBox provides up to four ports to interface with as many CPUs. It provides up to two ports to interface with the memory subsystem and up to four ports to interface with the I/O subsystem. The SCU can maintain simultaneous activity in all 10 ports.

Another major SCU function is to manage memory access. Because memory data is distributed in the CPU caches, the data in main memory may not be valid. The SCU tracks the location of valid data through a cache consistency unit, and ensures that a memory port request results in a valid read or write operation.

Interrupts from I/O devices, the SPU, and between CPUs are controlled by the I/O control unit of the SCU. I/O interrupts arbitrate for interrupt handling and can be distributed to CPUs in a round-robin fashion or directed to a single CPU.

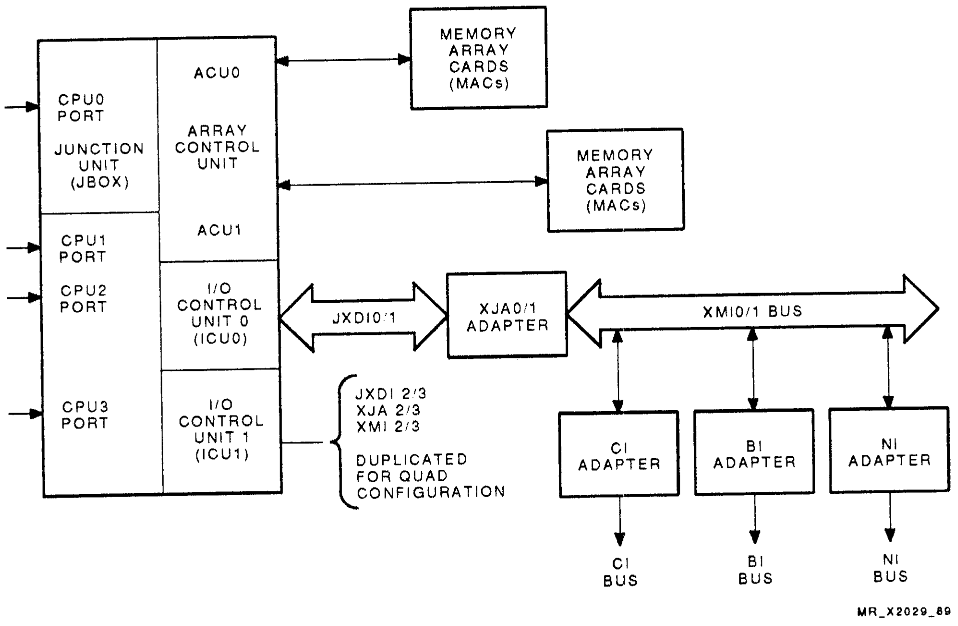


Figure 1–5 SCU Simplified Block Diagram

1.3.5 Main Memory Subsystem

The main memory subsystem consists of the memory control functions that are handled by the SCU and the main memory modules. The system supports a minimum of 256 Mbytes of main memory and can be expanded in increments of 256 Mbytes. The maximum supported configuration is 512 Mbytes using 1-Mbyte DRAMs. In addition, the system can support up to 2 Gbytes of main memory using 4-Mbyte DRAMs.

Memory is two-way interleaved with 256 Mbytes and four-way interleaved with 512 Mbytes. Each 256-Mbyte memory option connects to the CPU across a memory port that can read and write at 500 Mbytes/s.

1.3.6 I/O Subsystem

The I/O subsystem is based on the XMI bus. The XMI bus is a dedicated I/O bus that can operate at 125 Mbytes/s at peak performance. Figure 1–6 shows a simplified block diagram of the I/O system.

Up to four XMI buses can be added to the model 400 systems for a combined I/O bandwidth of 500 Mbytes/s at peak performance. Each XMI has a 12-slot backplane to accommodate multiple adapters in the I/O subsystem.

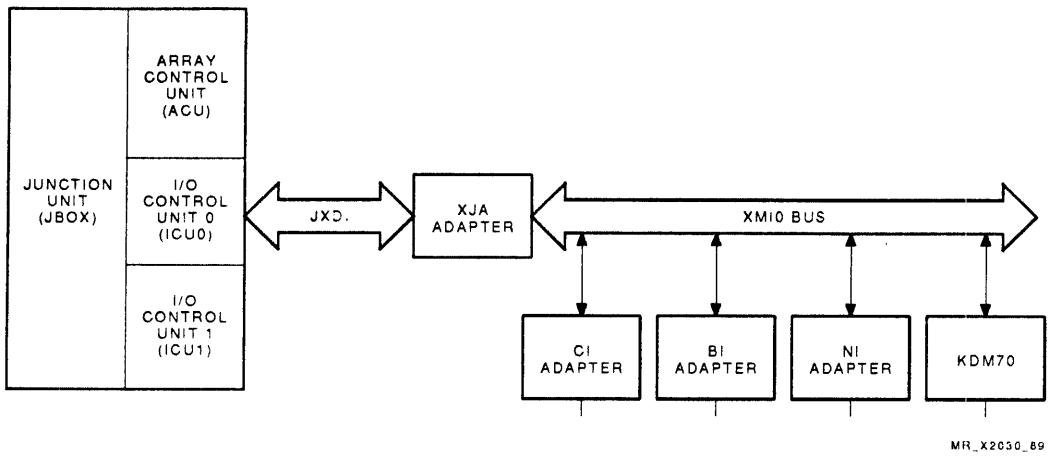


Figure 1–6 I/O System Simplified Block Diagram

1.3.6.1 I/O Adapters

Four XMI adapters are available for connection to the XMI bus:

- CIXCD adapter for connection to VAXcluster systems
- DEMNA controller for connection to Ethernet
- KDM70 controller for connection to Digital storage architecture (DSA) disks and tapes
- DWMBB interface for connection to the BI bus

The CIXCD adapter is a high-performance, intelligent I/O interface that connects to the Computer Interconnect (CI) VAXcluster systems. The CIXCD adapter allows communication over both CI paths simultaneously and in any order. The CIXCD adapter provides up to a four-fold increase in I/O performance over previous CI interfaces.

The DEMNA controller is a high-performance, XMI compatible, Ethernet controller. The DEMNA controller contains a CVAX processor for port command and data flow control and 512 Kbytes of on-board memory for command and data packet buffering.

The KDM70 controller is an XMI controller for RA disks, TA tapes, and ESE20 disks. The controller provides eight ports, any two of which can be used for tape interconnect. It achieves sustained data rates of 4.0 Mbytes/s with two ESE20s and 3.4 Mbytes/s with two RA90s, at speeds of up to 700 I/O requests per second.

The DWMBB interface is the VAXBI-to-XMI interconnect and allows connection of the VAXBI bus to the VAX 9000.

1.3.7 Power and Power Control Subsystem

Two power conditioners may be configured into VAX 9000 systems:

H7392 utility port conditioner (UPC)
H7390 power front end (PFE)

The H7392 UPC converts the 3-phase ac input to a regulated 280 Vdc output, with the capability to power a 20 kW load. It is a standalone unit and may be placed up to 50 feet from the system. The UPC can operate over a wide range of input voltages while maintaining its regulated output. Using the ride-through capability, the UPC can maintain the dc output for a relatively long period in the event of an input power line sag or interruption.

The H7390 PFE is the low-cost alternative to the H7392 and is integrated into the IOA cabinet. It is offered only on model 200 systems in the U.S. market. The PFE converts the 3-phase ac input to a 280 Vdc output. It can operate over a wide range of input voltages and can power an 18.5 kW load. Using the ride-through capability, the PFE can maintain the dc output for a short period in the event of an input power line sag or interruption.

Both conditioners can be operated remotely from the operator control panel or locally at the unit. The dc output of both conditioners are distributed to sets of dc/dc converters and air movers.

Power converters are configured into converter groups that have n + 1 redundancy. Each converter group comprises a specific power bus and is monitored and controlled by a regulator intelligence card (RIC). The RICs communicate through the power and environmental monitor (PEM) to the SPU. The SPU is notified of abnormal conditions and error conditions by the PEM.

Model 400 systems provide n + 1 redundancy for increased reliability. For example, the CPU logic has ten 240 A converter modules. Five converter modules supply -5.2 Vdc, while the remaining five modules supply -3.4 Vdc. Each set constitutes a regulator group, but only four converters in each group are needed to supply the entire load for the group. The fifth converter in each group provides the n + 1 redundancy by providing the balance of the required load when one of the converters in the group fails.

Figure 1–7 shows a simplified block diagram of the power system.

1.4 Diagnostic Test Strategies

Two diagnostic strategies, test-directed diagnosis (TDD) and symptom-directed diagnosis (SDD), are used to isolate hardware failures.

The system is designed with built-in testing features. These features include scan latches in the CPU and SCU, and built-in self-tests (BISTs) in the memory, the XMI, BI, and SPU controllers, and the power system. The diagnostics use these features to provide fault detection and isolation to a field replaceable unit (FRU).

The system uses TDD, including scan pattern generated diagnostics, and BISTs to detect and isolate "stuck at" fault conditions. Intermittent faults are detected and isolated using SDD. Dynamic faults in the system are detected by macro-level functional diagnostics and SDD.

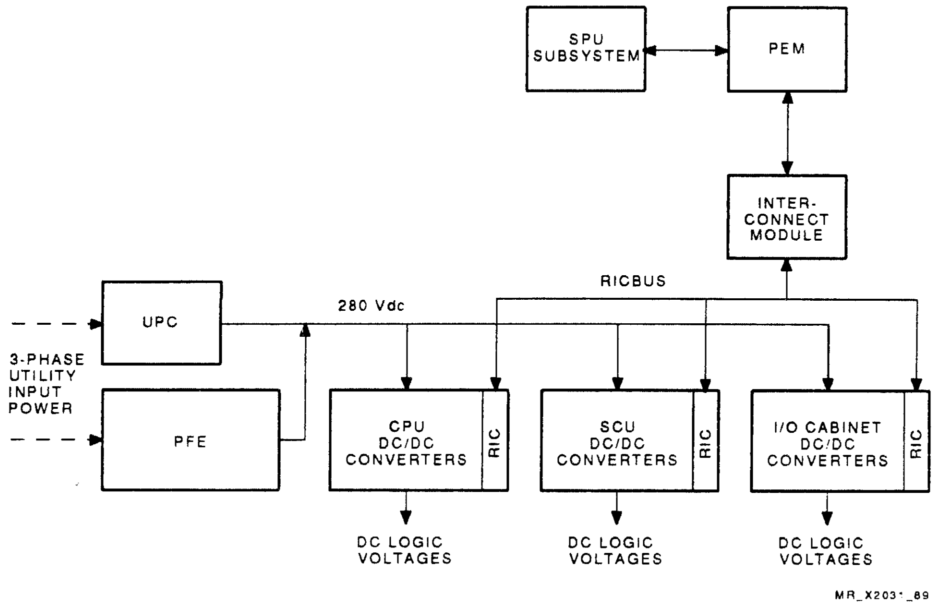


Figure 1–7 Power System Simplified Block Diagram

1.4.1 Test-Directed Diagnosis

TDD represents the traditional approach to fault isolation. TDD in the VAX 9000 uses four major types:

- ROM-based diagnostics (RBDs) and built-in self-tests (BISTs)
- SPU-based diagnostics
- System memory-based diagnostics
- Standalone and VDS-supported macrodiagnostics

The diagnostics are written at various levels of complexity to exercise the hardware to recreate a fault condition. Once the fault has been recreated and is recognized by the test routine, an FRU or function callout is produced.

Generally, the bottom-up and building-block approaches are used in TDD testing. This allows previously tested logic to be used as an aid in testing and isolating logic faults at a higher level.

The TDD diagnostic programs and routines are used to verify correct machine operation, starting with the SPU and scan path, progressing through the CPU and SCU logic, macroinstruction execution, and I/O subsystem interaction. TDD also includes hardware-based self-test routines that execute during system power-up and initialization. Typically, these diagnostics are used for acceptance testing during initial installation or system upgrade.

TDD routines are also used selectively to provide repair verification testing after FRU replacement. Additionally, in those cases where SDD cannot provide FRU callout, the TDD diagnostics are used for fault isolation.

1.4.2 Symptom-Directed Diagnosis

SDD is a diagnostic software product that runs on the SPU. The diagnostic performs fault isolation for any hardware errors that are reported to the SPU. SDD diagnostics analyze symptom information, saved when the fault is detected, to determine the cause of the error. The symptom information includes error latch values and key signal values that can be used to isolate the fault.

SDD diagnostics perform a vital role in the VAX 9000 family diagnostic strategy by providing instant and reliable fault analysis information to Customer Services. SDD diagnostics offer more reliable fault isolation for intermittent faults because error analysis is performed on symptoms that are saved at the time of the error, rather than on symptoms of a recreated error. SDD also offers faster error analysis because the diagnostics are automatically initiated by the SPU when a hardware error is reported.

1.4.3 Diagnostic Test Techniques

The system uses several distinct diagnostic techniques for fault detection and isolation. The general diagnostic techniques and fault coverage goals are as follows:

- **Built-in self-test** — Used for quick verification of devices when power is applied and used to isolate faulty devices.

  Fault coverage of 95% to 99%, depending on the unit under test, with an average isolation to 1.5 FRUs.

- **Scan patterns** — Used for quick verification when power is applied and used to isolate hard logic failures.

  Greater than 98% coverage, depending on the unit under test, with an average isolation to 1.1 MCUs and two components within an MCU.

- **Standalone macro-coded diagnostics (level 4)** — Used for quick verification of the XJA and XBI devices when power is applied. Also used for VAX macro hard-core instruction test before executing the VAX diagnostic supervisor (VDS).

- **Standalone VDS diagnostics (level 3)** — Used for functional verification of device operation when the system is installed or a device is replaced.

  Fault coverage of 95%, depending on the unit under test, with an average isolation to 1.5 MCUs.

- **VDS diagnostics in a VMS environment (levels 2 and 2R)** — Used for functional verification of device operation in the VMS environment.

- **Non-VDS diagnostics in a VMS environment (level 1)** — Used to simulate the operation of the system in a user application environment, and to isolate XJA and XBI failures in a VMS environment.

- **Symptom-directed diagnosis (SDD)** — Used to isolate hardware-detected faults that occur during normal system operation.

  Fault coverage of 85% to 90%, depending on the unit under test, with an average isolation to 1.2 MCUs.

1.5 System Configurations

Depending on the model, the VAX 9000 may include the following:

- 1 to 4 VAX 9000 CPUs, each with 128 Kbytes of cache memory
- Space for an optional vector processor in each CPU
- 256 to 512 Mbytes of main memory
- 1 to 4 XMI I/O buses
- Service processor
- Power system

The following sections describe the configurations of each of the models of the VAX 9000 family of systems. The descriptions also include the options available and the configuration limits of each model.

1.5.1 Model 200 Configurations

The VAX 9000 model 200 system is a uniprocessor, base-level system that has limited available options and expansion capabilities.

1.5.1.1 Model 210 Configuration

The VAX 9000 model 210 system is packaged in three cabinets and includes the following standard equipment:

- 1 CPU
- Service processor
- 256-Mbyte memory
- 1 XMI I/O bus
- 1 KDM70 storage controller or 1 CIXCD VAXcluster interface
- 1 DEMNA Ethernet controller
- 1 H7390 power front end (PFE) or 1 H7392 utility port conditioner (UPC) for the non-U.S. market

The available options are:

- Vector accelerator (VBox)
- H7392 UPC

1.5.2 Model 400 Configurations

The VAX 9000 model 400 systems are available in four configurations. The four models differ mainly in the number of CPUs included in the system. The four models are as follows:

Model 410 is a single CPU system.
Model 420 is a dual CPU system.
Model 430 is a triple CPU system.
Model 440 is a quad CPU system.

1.5.2.1 Model 410 Configuration

Model 410 is packaged in three cabinets and includes the following standard equipment:

- 1 CPU
- Service processor
- 256-Mbyte memory
- 2 XMI I/O buses
- 1 CIXCD VAXcluster interface
- 1 DEMNA Ethernet controller
- 1 H7392 UPC

The available options are:

- Vector accelerator (VBox)
- Additional 256 Mbytes of memory

See Table 1–1 for the XMI bus configuration limits of the model 410.

Table 1–1 XMI Bus Configuration Limits for Models 410 and 420

Device	Minimum	Maximum
DEMNA	1	8
CIXCD	1	8
VAXBI	0	8
KDM70	0	11

1.5.2.2 Model 420 Configuration

The model 420 system is packaged in four cabinets and contains the following standard equipment:

- 2 CPUs
- Service processor
- 256-Mbyte memory
- 2 XMI I/O buses
- 1 CIXCD VAXcluster interface
- 1 DEMNA Ethernet controller
- 1 H7392 UPC

The available options are:

- 1 vector accelerator (VBox) for each CPU
- Additional 256 Mbytes of memory

The XMI bus configuration limits of the model 420 are listed in Table 1–1.

1.5.2.3 Model 430 Configuration

Model 430 is packaged in five cabinets and contains the following standard equipment:

- 3 CPUs
- Service processor
- 512-Mbyte memory
- 4 XMI I/O buses
- 2 CIXCD VAXcluster interfaces
- 2 DEMNA Ethernet controllers
- 2 H7392 UPCs

Each CPU can be configured with an optional vector accelerator (VBox).

See Table 1–2 for the XMI bus configuration limits of the model 430.

Table 1–2 XMI Bus Configuration Limits for Models 430 and 440

Device	Minimum	Maximum
DEMNA	2	16
CIXCD	2	16
VAXBI	0	14
KDM70	0	23

1.5.2.4 Model 440 Configuration

The model 440 kernel system is packaged in five cabinets and contains the following standard equipment:

- 4 CPUs
- Service processor
- 512-Mbyte memory
- 4 XMI I/O buses
- 2 CIXCD VAXcluster interfaces
- 2 DEMNA Ethernet controllers
- 2 H7392 UPCs

Each CPU can be configured with an optional vector accelerator (VBox).

The XMI bus configuration limits of the model 440 are listed in Table 1–2.

2 Technology and System Packaging

This chapter introduces and describes the new system technologies, including system packaging and cooling. The system and I/O cabinet configurations, including cabinet contents and major component locations, are also described.

2.1 Technology Overview

The CPU and SCU use the following technologies:

- Third-generation ECL MCA IIIs
- Self-timed RAMs (STRAMs)
- Self-timed registers (STREGs)
- Vector registers (VRGs)
- Advanced multichip packaging unit (MCU)
- High density signal carrier (HDSC) integrated circuit interconnect
- Planar module assembly

2.1.1 Macrocell Array III

The macrocell array III (MCA III) is the basic logic building block for the CPU and SCU. It is implemented in third-generation ECL gate arrays with approximately eight times the density of MCA I devices.

2.1.1.1 Physical and Functional Structure

Each MCA has a 360-pin array with 200 output cells, 224 input cells, 2 clock driver cells, and 256 I/O ports. The MCA occupies a die size of 385 × 385 mils and has a floor plan as shown in Figure 2–1.

An MCA has 414 major cells that each contain 76 transistors and 76 resistors. The 414 major cells can be subdivided into quarter cells, producing a total of 1656 quarter cells. Each MCA can accommodate a maximum of 224 inputs and 200 outputs, but the total input and output capacity cannot exceed 256. (For example, if there are 224 input cells in an MCA, then the number of output cells is limited to 32.) A detailed MCA layout is shown in Figure 2–2.

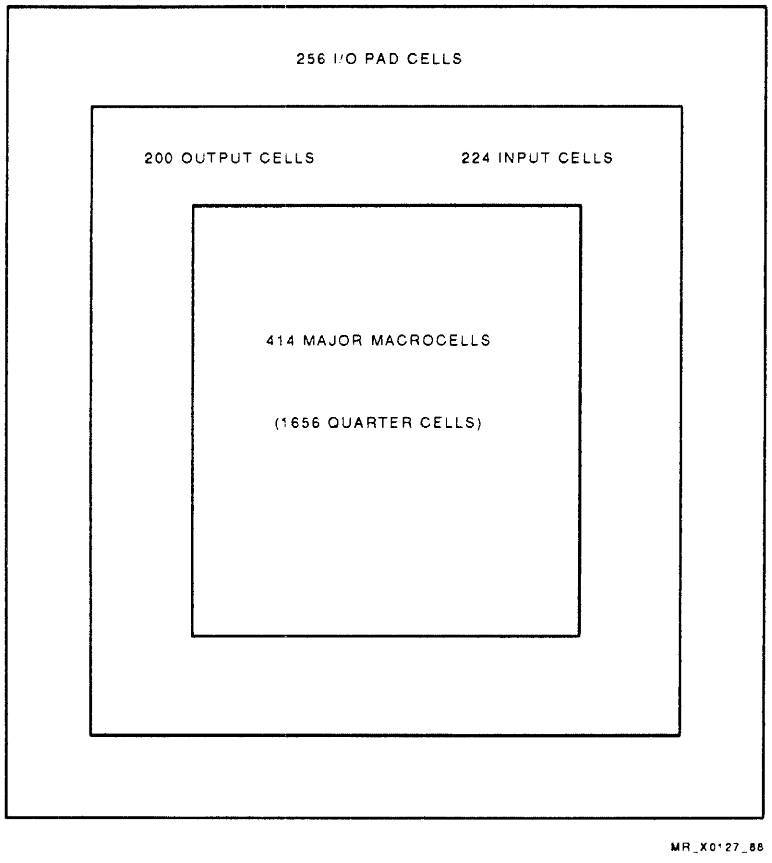


Figure 2–1 MCA III Floor Plan

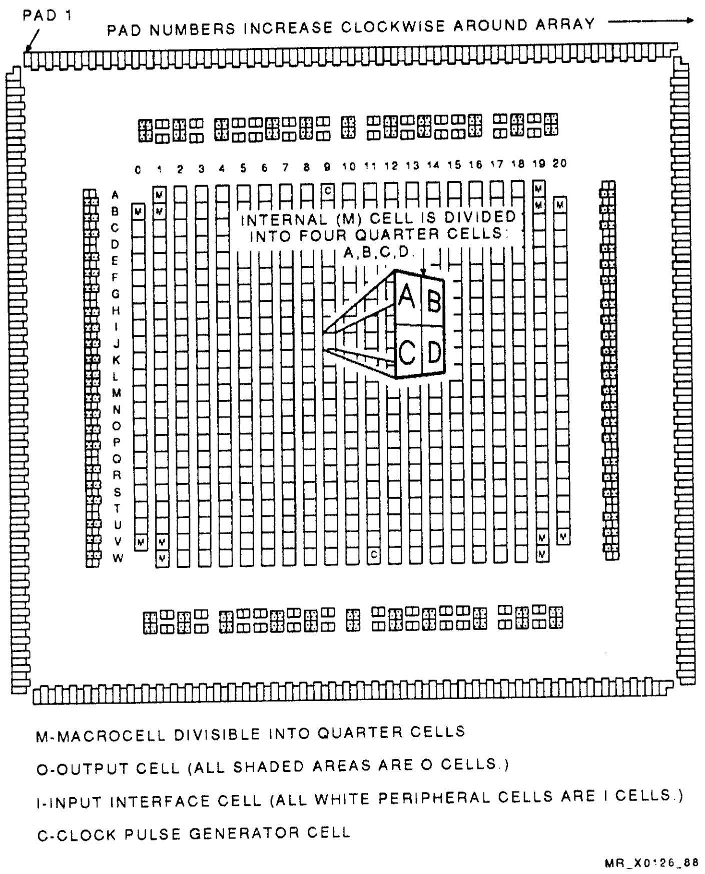

PAD 1
PAD NUMBERS INCREASE CLOCKWISE AROUND ARRAY
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
A B C D E F G H I J K L M N O P Q R S T U V W
INTERNAL (M) CELL IS DIVIDED INTO FOUR QUARTER CELLS: A,B,C,D
A B C D
M-MACROCELL DIVISIBLE INTO QUARTER CELLS
O-OUTPUT CELL (ALL SHADED AREAS ARE O CELLS.)
I-INPUT INTERFACE CELL (ALL WHITE PERIPHERAL CELLS ARE I CELLS.)
C-CLOCK PULSE GENERATOR CELL
MR_X0126_88

Figure 2–2 Detailed MCA Layout

The following MCA gate counts depend on the type of logic implemented:

- Over 10,000 equivalent gates are achieved if full adders, output latches, and input ORs are used in all cells.
- Over 7,000 equivalent gates are achieved if flip-flops, output latches, and input ORs are used in all cells.

The power dissipation of each MCA is approximately 15 to 30 W.

2.1.2 Self-Timed RAMs

STRAMs are storage arrays that are used throughout the CPU and SCU. In the CPU, the data cache, EBox control store, virtual instruction cache, and branch prediction cache are implemented in STRAMs.

2.1.2.1 STRAM Functional Overview

The STRAM is a synchronous, self-timed, static RAM device. The STRAM is similar to a traditional RAM except that it requires write, differential clock, and reference voltage inputs.

Two different STRAM chips are used in the system: the 1K × 4 and the 4K × 4. The major differences in these STRAM chips are the number of address bits required and the chip propagation delays. A STRAM requires the following inputs:

- Address: A [11:00] (4K), or A [09:00] (1K)
- Data input: DIN [03:00]
- Data output: DO [03:00]
- Chip select
- Write enable
- Differential clocks

Figure 2–3 shows the logic symbol for the 1K STRAM.

The logic symbol for the 1K and 4K STRAM differs only in the number of address bit inputs. (There are 9 address bits for the 1K STRAM and 11 address bits for the 4K STRAM.)

The core structure of the STRAM contains a 1K or 4K memory cell and decode and read/write logic. The core is surrounded by latches that store address, input data, control signals, and output data. During a write operation, an internal write pulse generator, driven by the external differential clocks causes data stored in the data latches to be written into the array and the output latches. During a read operation, the selected array data is stored in the output latches.

Address, data, and write and chip select functions (Figure 2–4) are passed through the input latches when the internal clock is a logic low and are stored when the clock is high. The output latch stores data when the clock is low and allows data to pass when the clock is high.

2.1.2.1.1 Read Operation

A read operation is initiated when the clock is low. The address, chip select, and write input latches open and allow the signals to propagate to the array (Figure 2–4). Note that the output latch is closed, and data from the previous cycle is stored. When the clock is high, the new data from the array is latched into the output latch, and the previous data is overwritten. If a read operation is attempted when the chip select function is not asserted, the STRAM loads the output latches with logic lows.

2.1.2.1.2 Write Operation

A write operation is also initiated when the clock is low. The address, data, chip select, and write input latches open and allow the signals to propagate to the array. When the clock is high, the new data is written into the array and the output data latch (Figure 2–4). If a write operation is attempted when the chip select function is not asserted, the STRAM prevents new data from being written to the array. However, the STRAM writes a logic low to the output latch.

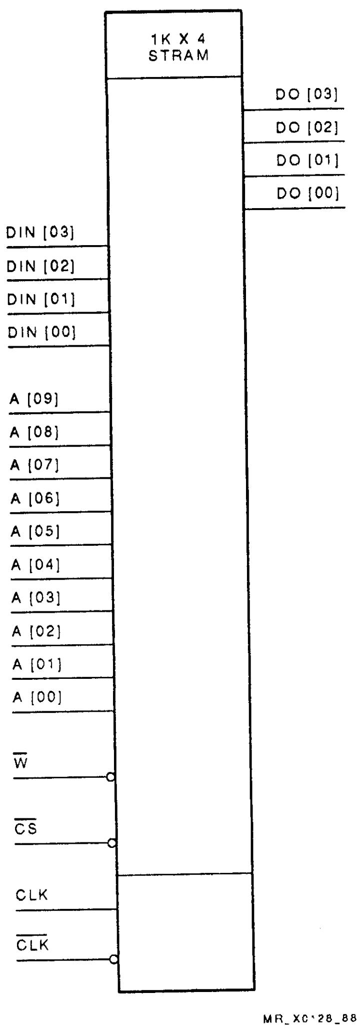


Figure 2–3 1K STRAM Logic Symbol

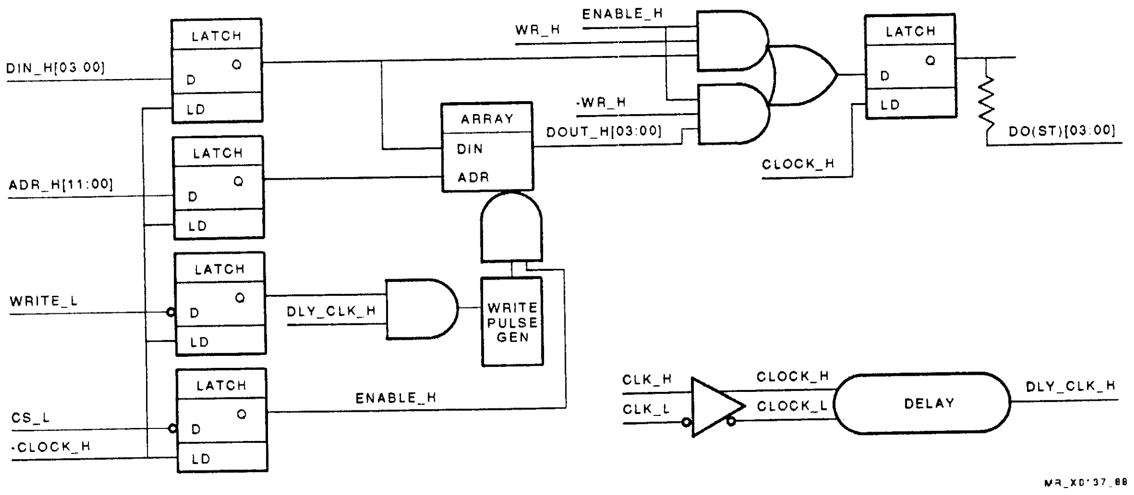


Figure 2–4 STRAM Cell Block Diagram

2.1.3 Self-Timed Registers

STREGs, like STRAMs, are self-timed storage arrays. STREGs are used as the storage devices that implement the GPR files in both the EBox and IBox. In the EBox, the STREG is also used to implement the microcode temporary registers.

2.1.3.1 STREG Functional Overview

The STREG is a 64 × 18-bit register file containing three write ports and two read ports (Figure 2–5). Figure 2–6 shows the 64 locations separated into four 16-location storage array sections (banks). The two read ports have independent access to all 64 locations. Simultaneous reads to the same location are allowed.

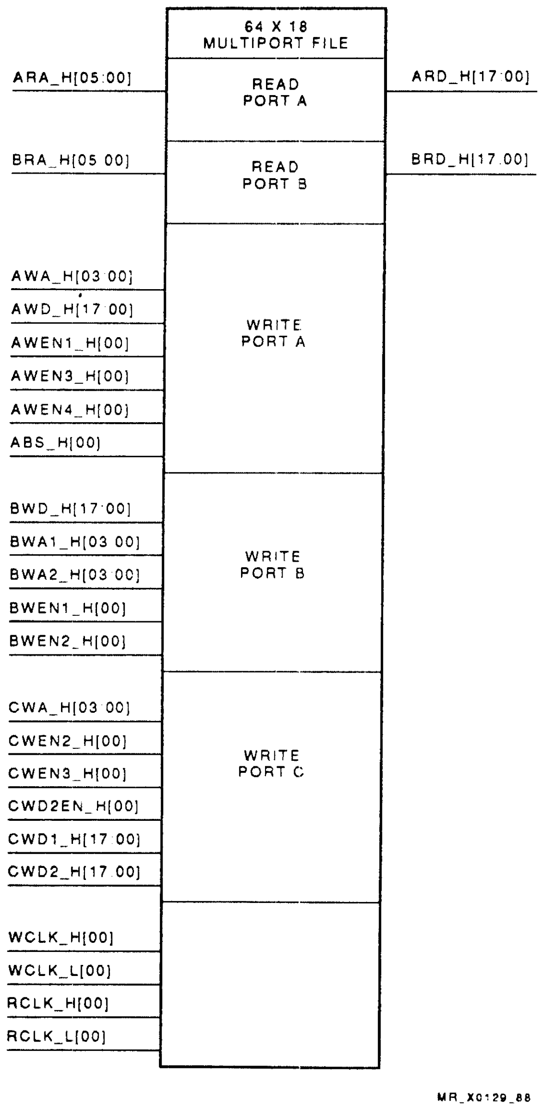


Figure 2–5 STREG Logic Symbol

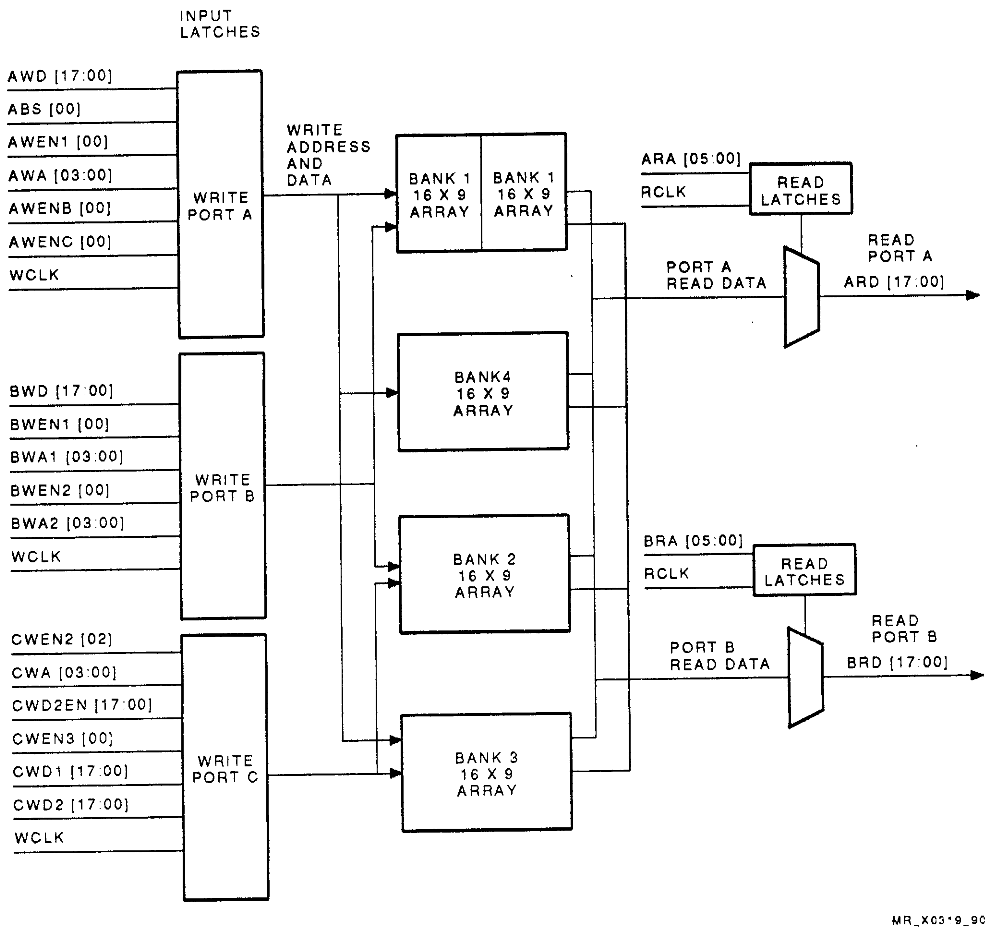


Figure 2–6 Basic STREG Block Diagram

Simultaneous write operations are possible through the three write ports, but there is no provision to detect writes to the same location from multiple ports. That is, data integrity is not guaranteed for simultaneous writes to the same address.

There are two clock inputs to the STREG: the read clock (RCLK) and the write clock (WCLK). The clocks control latching of the address, write enable, byte select, and data inputs, but the array timing is internally generated and controlled.

2.1.3.1.1 Read Operation

The read ports have a 6-bit address and an 18-bit read data field.

- Port A input: ARA [05:00]
  Data: ARD [17:00]
- Port B input: BRA [05:00]
  Data: BRD [17:00]

Read addresses are latched on the failing edge of the read clock (RCLK) (Figure 2–6). (Table 2–1 specifies the bank and location.)

Table 2–1 Read Port Addressing

Address	Array	Location
00–0F	Bank 1	0–15
10–1F	Bank 2	0–15
20–2F	Bank 3	0–15
30–3F	Bank 4	0–15

2.1.3.1.2 Write Operation

Address, data, write enable, and clock inputs are required for a write operation. Each bank has a write enable bit (xWENy) in which x is the port and y is the bank. All write port input is latched on the falling edge of the write clock (WCLK). The three ports write to different combinations of banks, but one port does not write to all 64 locations. In addition, each port has a different set of input requirements and write capabilities.

Write port A has one set of address and data lines used as input. The address and data lines are applied to banks 1, 3, and 4. To write into a bank, or a combination of banks, its write enable bit (AWENy) must be asserted.

Write port A is the only port with a byte write ability. Byte selection is provided by the byte select input (ABS) (Figure 2–6). ABS selects a write into the low-order byte or the entire word at the location specified in Table 2–2.

Table 2–2 Write Port A Addressing

AWEN1	ABS	Bits Selected
0	0	None
0	1	None
1	0	08:00
1	1	17:00

Write port B has two sets of bank address lines and two corresponding bank write enable bits (BWA1 [03:00] and BWEN1 for bank 1; BWA2 [03:00] and BWEN2 for bank 2). The port has the ability to write to different locations in both banks, provided the corresponding BWENx is set (Figure 2–6). The inputs in write port B, like those in write port A, are latched on the falling edge of the write clock (WCLK).

Write port C has one set of lines that addresses banks 2 and 3 (CWA [03:00]), and each bank has a corresponding write enable input (CWEN2 and CWEN3). The port also has two sets of data lines (CWD1 [17:00] and CWD2 [17:00]) and CWD2 has a path enable bit (CWD2EN). The data line sets are used to write two locations simultaneously. That is, CWD1 data is written to banks 2 or 3, or both. CWD2 data is written only if CWD2EN is asserted and it is written only to location CWA + 1. The input data is latched on the falling edge of WCLK.

The following example describes port operation (Figure 2–6).

Assume the following input conditions:

Address 1 (CWA [0001])
Bank 2 disabled (CWEN2_L)
Bank 3 enabled (CWEN3_H)
Data path 1 122 (CWD1 [112])
Data path 2 3F09 (CWD2 [3F09])
Data path 2 enabled (CWD2EN_H)

Bank 2 is not involved in the write operation (CWD2EN_L). CWD1 data (112) is written into location 1 of bank 3. At the same time, CWD2 data (3F09) is written into location 2 of bank 3.

If the first set of data had been written into location 15, the second set of data would have been written into location 0 (wrapped around).

2.1.4 Vector Registers

The VRG chip is a bit-sliced custom VLSI storage module that is used to implement the vector register file functions and port interfaces for the VBox option of the CPU. Sixteen 64 × 9-bit VRGs are used to implement the vector register file. This configuration yields a storage module containing 1024 storage locations or vector locations.

2.1.4.1 Vector Register Functional Overview

The VRG is a 64 × 9-bit register that contains three write ports and five read ports and is capable of five read and three write accesses in a single cycle.

Each read or write port has independent access to all 1024 vector elements. Simultaneous read and write access to all elements (locations) of the VRG is possible through the read or write ports with the correct result occurring, except for simultaneous write access of multiple ports to a particular VRG, which is an undefined operation.

Figure 2–7 shows the logic symbol for the VRG.

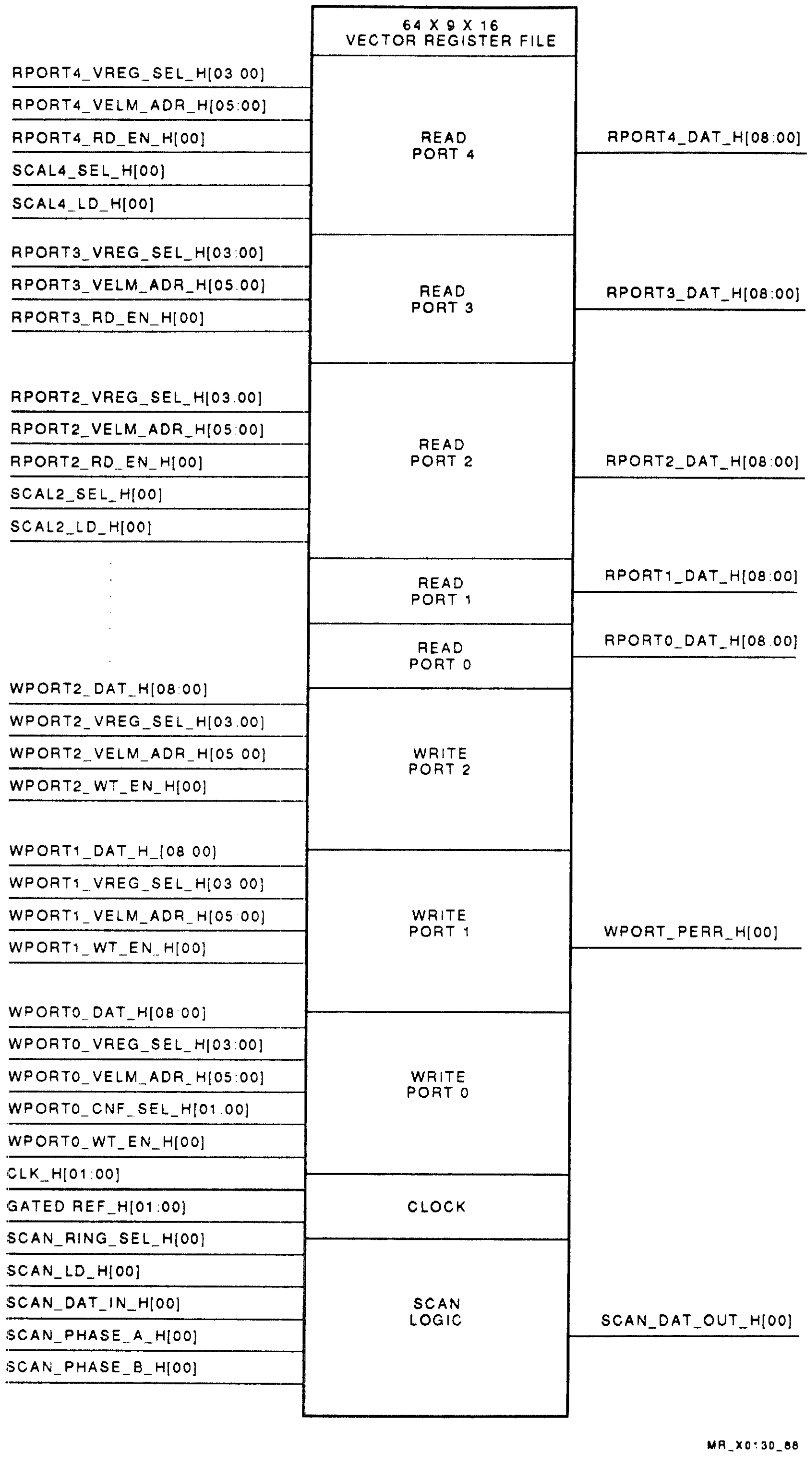


Figure 2–7 VRG Logic Symbol

2.1.5 Series-Terminated ECL

The MCA III logic elements of the CPU and SCU are configured with series terminated transmission lines. Series termination controls overshoot and ringing on transmission lines and requires a lower overall power requirement. The power required is supplied by the logic power supplies, as opposed to additional power supplies.

Figure 2–8 shows a circuit containing a series terminated transmission line. The operation of this circuit is described in the following sections.

STECL contains a small resistor (RS) in series with the output of the driving gate (❶). The resistor and the circuit output impedance is equal to the output impedance of the transmission line (Z_0).

When the circuit switches states, the voltage at the resistor (RS) is one-half the voltage of the internal voltage of the circuit (❷). After the propagation delay time of the transmission line, the waveform reaches the end of the transmission line (❸). At the end of the transmission line, the voltage doubles due to the reflection from the load at the end of the transmission line.

The reflection voltage (which is equal to the source voltage) returns to the source after the propagation delay time of the transmission line and no more reflection should occur.

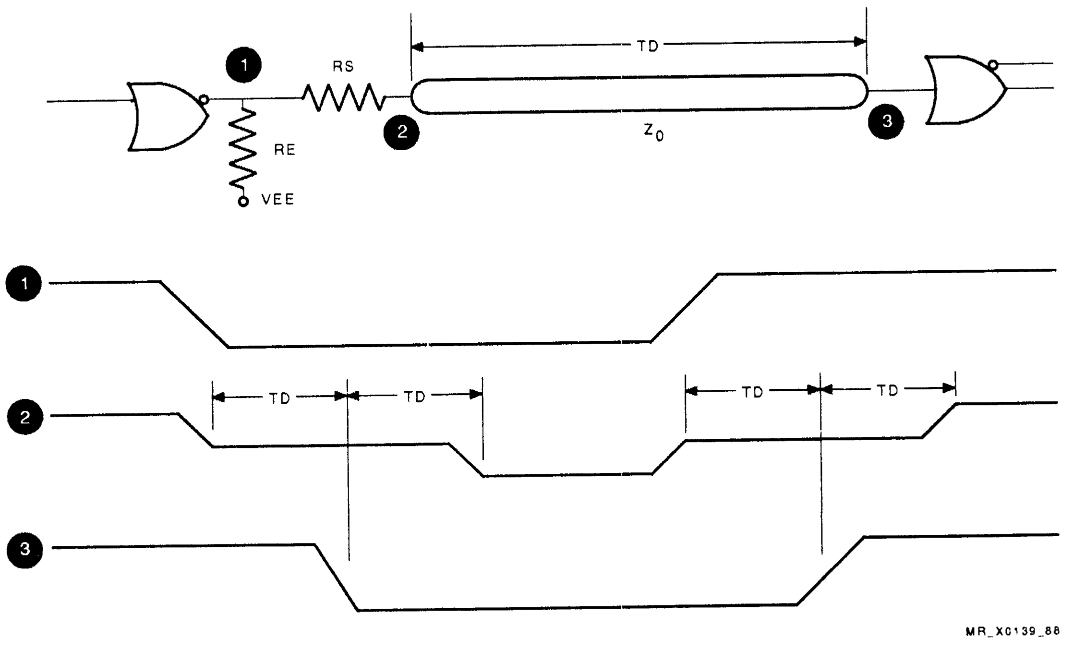


Figure 2–8 STECL

2.1.6 Multichip Unit

The MCAs, STRAMs, STREGs, and VRGs are assembled in an MCU. The MCU is the CPU and SCU field replaceable unit (FRU). The MCU is 10.2 cm (4.0 in) square and can accommodate 8 MCAs, 48 STRAMs, or a combination of both. Each MCU also contains one clock distribution chip (CDC).

The MCU is assembled by mounting the MCAs and STRAMs on a high density signal carrier (HDSC) (Figure 2–9). The HDSC is then mounted to the MCU housing. The MCU housing consists of a housing, a copper baseplate for heat dissipation, and a lid to enclose the HDSC. Signal flex connectors are connected from the HDSC to the planar module PWB. Power flex connectors are connected from the HDSC to the bus bars that provide power from the power subsystem.

The HDSC consists of nine layers and provides the I/O, signal, and power interconnects for all MCU components. It consists of three components: a baseplate, and power and signal cores.

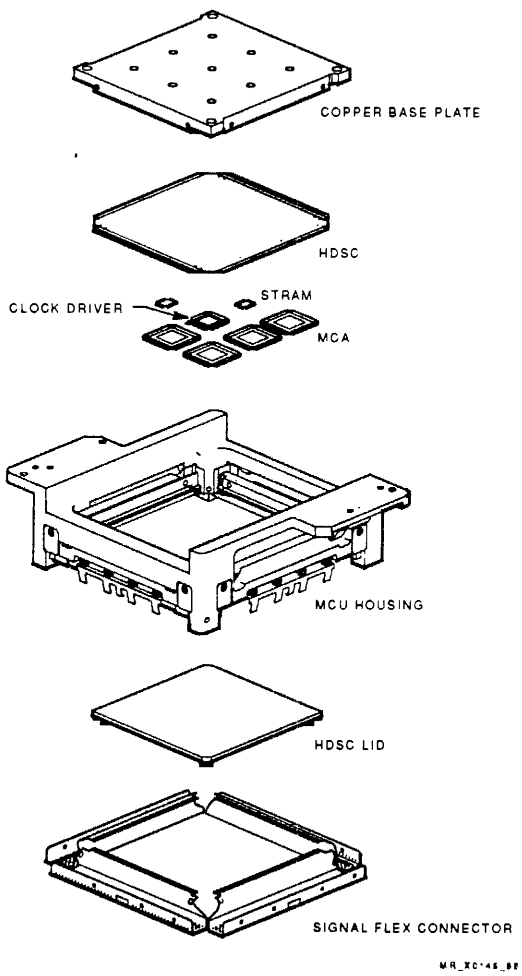


Figure 2–9 MCU Exploded View

The baseplate is chrome copper and is fastened to the cold plate to dissipate MCU component heat into the cooling subsystem. The signal and power cores consist of the following layers (Figure 2–10):

One footprint
Two reference
Two controlled impedance signal lines
Four power distribution planes

Signal interconnects from the MCU are provided by four, 201-pad, signal flex connectors (Figure 2–11). The power planes are connected through two power flex connectors (Figure 2–12).

Figure 2–13 shows a typical MCU layout. Note that the illustration orientation is the opposite of Figure 2–9.

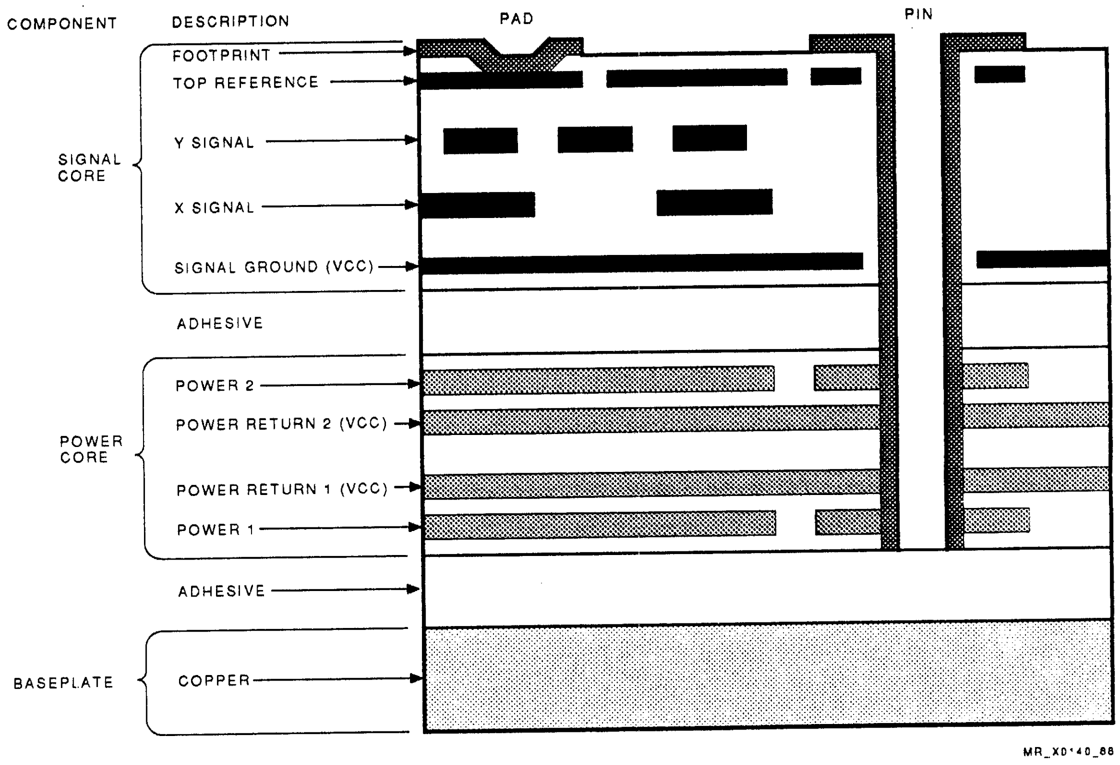


Figure 2–10 HDSC Layers (Side View)

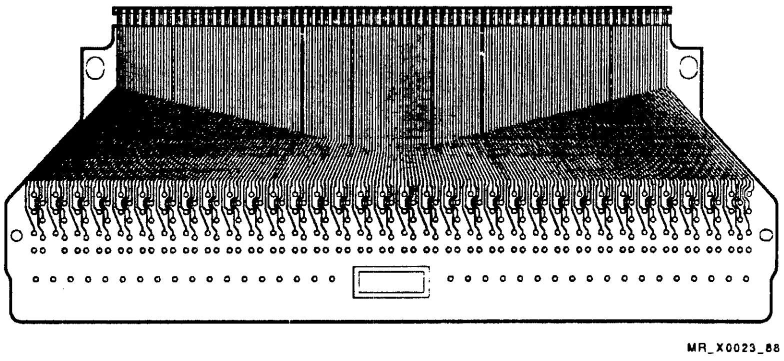


Figure 2–11 MCU Signal Flex Connector

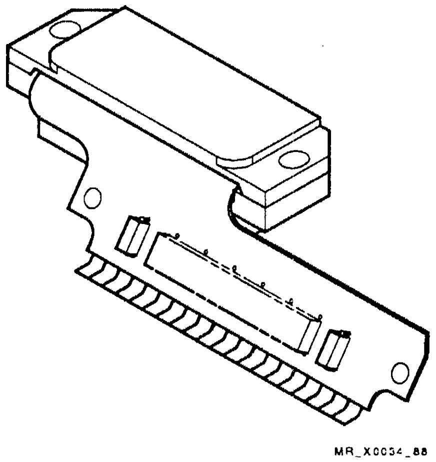


Figure 2–12 MCU Power Flex Connector

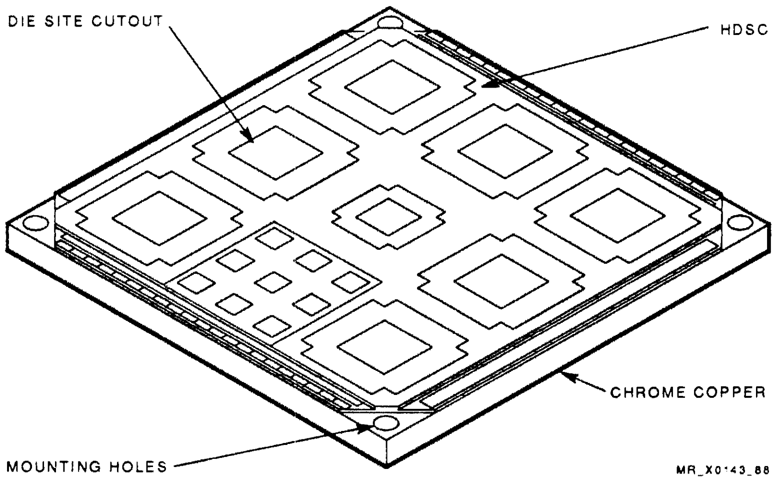


Figure 2–13 MCU Layout

2.1.7 Planar Module

The CPU and SCU components are assembled on planar modules. Figure 2–14 shows the physical layout of the CPU planar module.

Each planar module contains the following six basic parts (Figure 2–14):

- **MCU housing** — The MCUs are fastened to the housing in this layer.
- **Bus bar** — The bus bar assembly provides power to the MCUs.
- **MCU planar module** — This 26-layer advanced printed wiring board provides signal and power and ground interconnects to the MCUs.
- **Insulator** — This portion of the assembly provides an electrical insulation between the printed wiring board and the support module.
- **Support module** — The components of the planar module are attached to the support module. The support module is then attached to the specific cabinet.
- **Power dividers** — Four 1:4 power dividers supply clock signals to the CPU planar module from the clock subsystem and two 1:3 power dividers supply the clock to the SCU planar module.

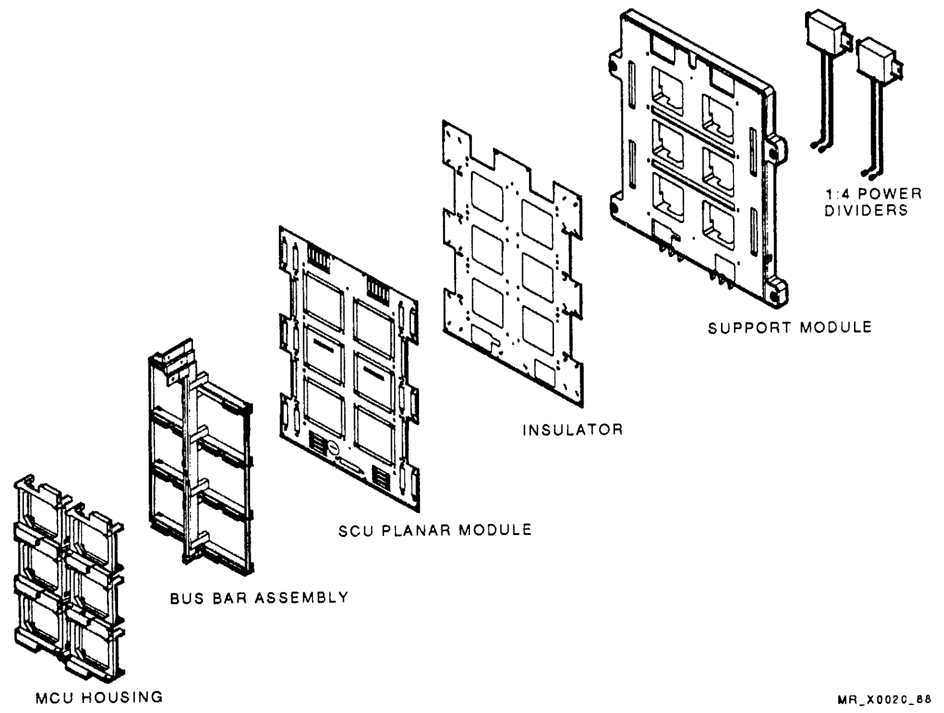


Figure 2–14 Planar Module Assembly

2.1.7.1 CPU Planar Module

The CPU planar module accommodates 16 MCUs, which provides 13 MCU dies for a scalar processor and 3 MCU dies for an optional vector processor. Figure 2–15 shows the layout of the CPU planar module.

The CPU planar module is approximately 63.5 cm (25.0 in) square and weighs approximately 81.6 kg (180.0 lb).

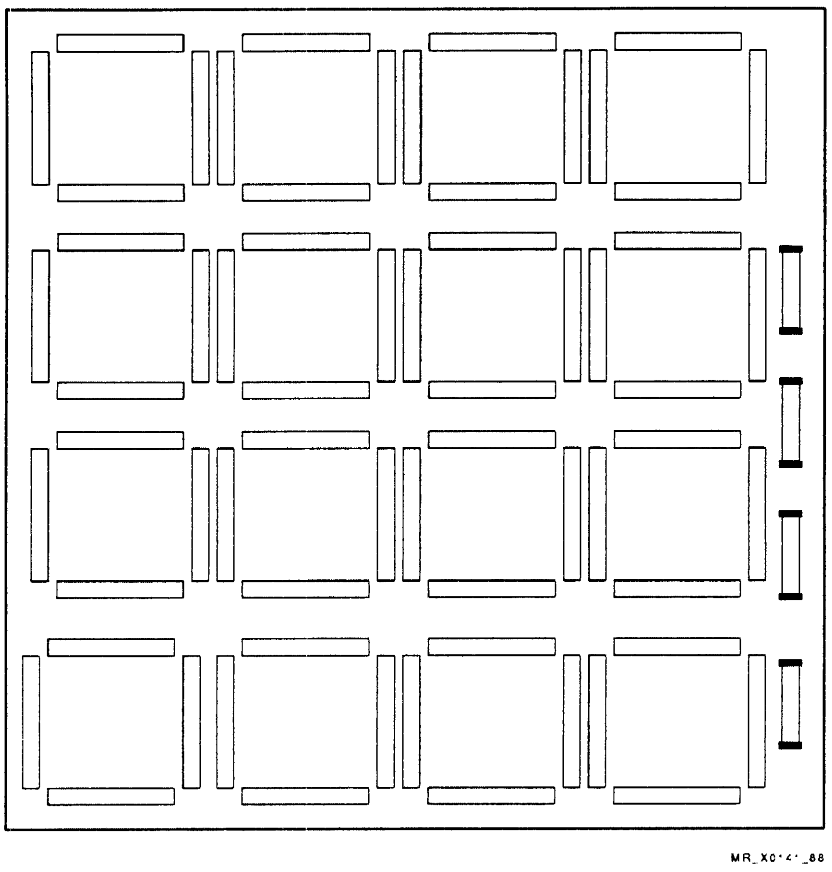


Figure 2–15 CPU Planar Module

2.1.7.2 SCU Planar Module

The SCU planar module accommodates six MCUs that provide four MCU dies for the JBox, the I/O subsystem control, and the memory subsystem control. Two additional MCUs can be installed to allow expansion of the memory and I/O subsystems.

The SCU planar module is 63.5 cm (25.0 in) high and 48.3 cm (19.0 in) wide and weighs approximately 45.4 kg (100.0 lb). Figure 2–16 shows the layout of the SCU planar module.

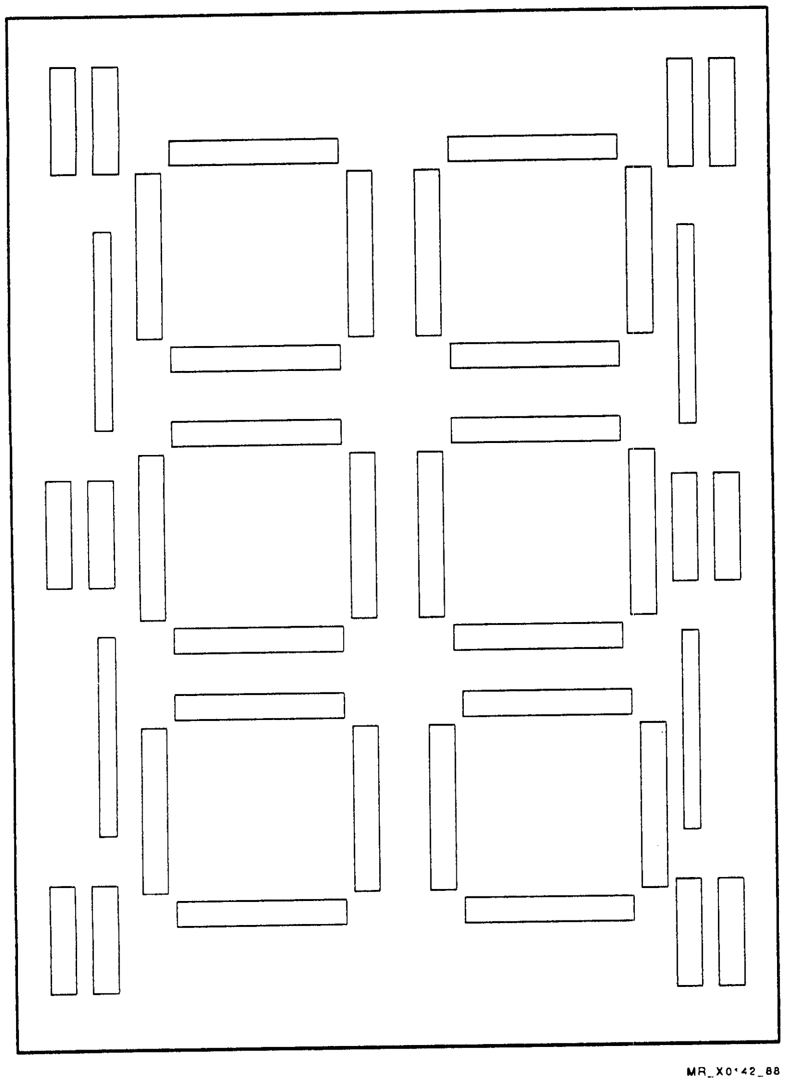


Figure 2–16 SCU Planar Module

2.2 VAX 9000 Cabinet Descriptions

This section provides a physical description of the system cabinets in the model 200 systems. The major cabinet components and the cooling system of each cabinet are described, including air movers, cooling component locations, and air flow. The main topics of this section describe only the kernel configurations.

2.2.1 Model 200 Cabinets

This section describes the cabinet configurations for model 210. Depending on the system configuration, the kernel of model 200 systems is packaged in three or four cabinets and may contain a number of expansion cabinets.

2.2.1.1 CPA Cabinet

The model 200 CPA cabinet contains the CPU planar module, power system components, and two blowers for cooling the logic components. Figure 2–17 shows the front and side views of the CPA cabinet, and Figure 2–18 shows the rear view.

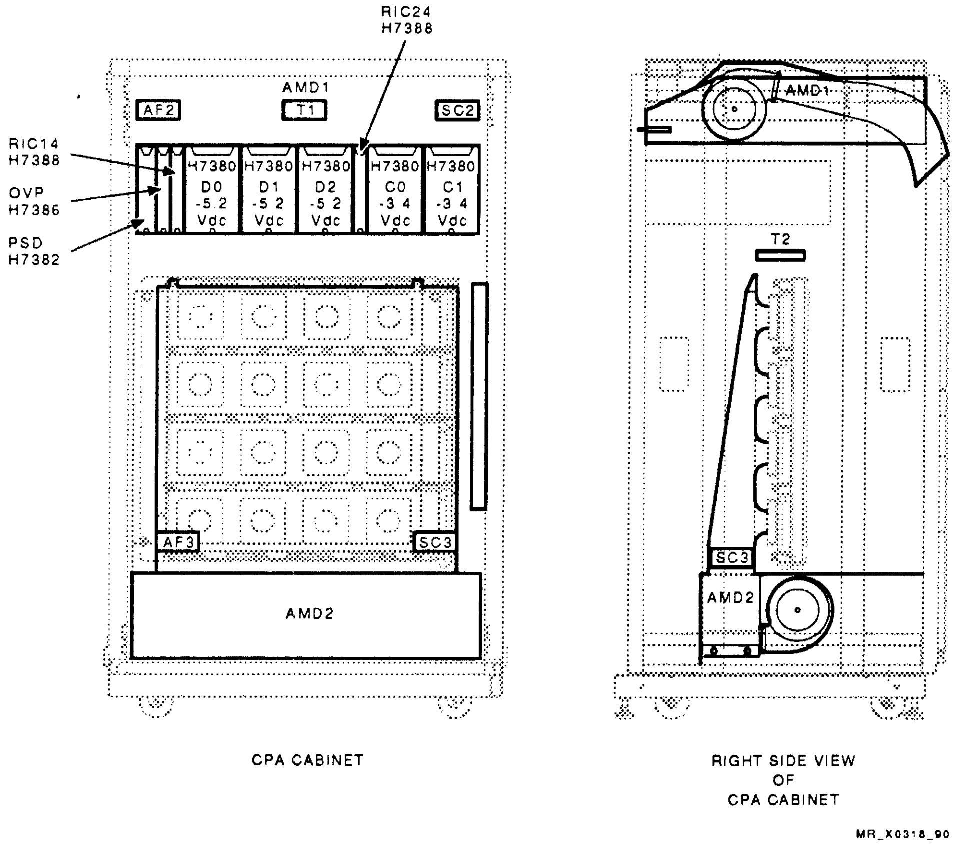


Figure 2–17 CPA Cabinet (Front and Side Views)

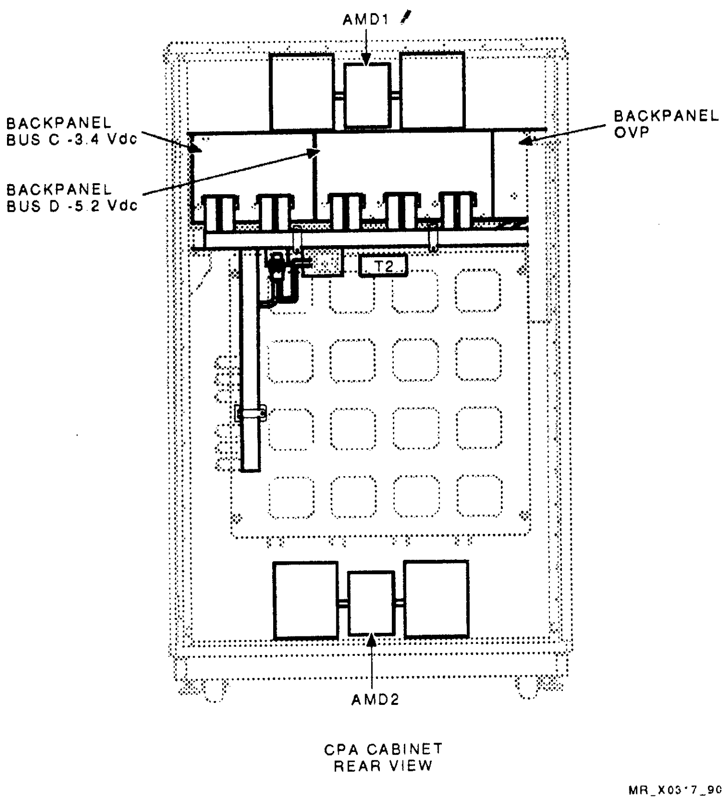


Figure 2–18 CPA Cabinet (Rear View)

2.2.1.1.1 CPU0 Planar Module

The CPU0 planar module is mounted vertically near the bottom center of the CPA cabinet with the active side of the components facing the rear of the cabinet. A plastic shroud encloses the front of the planar module to provide a plenum for efficient cooling.

2.2.1.1.2 Power Components

The power components in the CPA cabinet are configured into power buses and support logic in the three system cabinets. The following power components in the CPA cabinet are listed in their bus configuration groups:

- Two H7380 power converters provide -3.4 Vdc to the CPU and SCU planar modules. These two converters (two of three converters) are configured in the bus C power bus.
- The H7388 register intelligence card (RIC) monitors the status of several H7380 converters.

The remaining H7380 power converters are in the SCU cabinet.

- Three H7380 power converters provide -5.2 Vdc to the CPU and SCU planar modules.
- Another H7388 RIC monitors the status of the remaining H7380 converters.

The CPA cabinet also contains an H7386 overvoltage protection (OVP) module and an H7382 bias supply. The OVP module monitors for overvoltage conditions and notifies the RIC when this condition occurs. The H7382 bias supply provides bias power to the bus B regulators, the H7214/H7215 bias supplies, the SIP in the SPU, and the OVP module.

2.2.1.1.3 CPA Cabinet Cooling

The CPA cabinet contains two 550-CFM blowers that cool the logic components (Figure 2–18). One blower is installed near the top of the cabinet and cools the power system components. The second blower is installed at the bottom of the cabinet and cools the CPU planar module.

Each blower assembly contains a speed control sensor, a temperature sensing thermistor, and an air flow sensor. The power system monitors the output of the air flow sensors and reports air flow and temperature problems.

2.2.1.2 SCU Cabinet

In model 210, the SCU cabinet contains the SCU planar module, the memory subsystem, the clock subsystem, the power system components, and three blowers for component cooling. Figure 2–19 shows the front and side views of the SCU cabinet, and Figure 2–20 shows the rear view.

2.2.1.2.1 SCU Planar Module

The SCU planar module is mounted vertically in the rear of the SCU cabinet (Figure 2–20) with the active side of the components facing the front of the cabinet. The planar module is enclosed in a shroud that provides efficient cooling of the planar module components.

2.2.1.2.2 Master Clock Module

The master clock module (MCM) is mounted in the front of the SCU cabinet (Figure 2–19). The cables that deliver the clock signals to the SCU and CPU planar modules are connected to the front of the MCM.

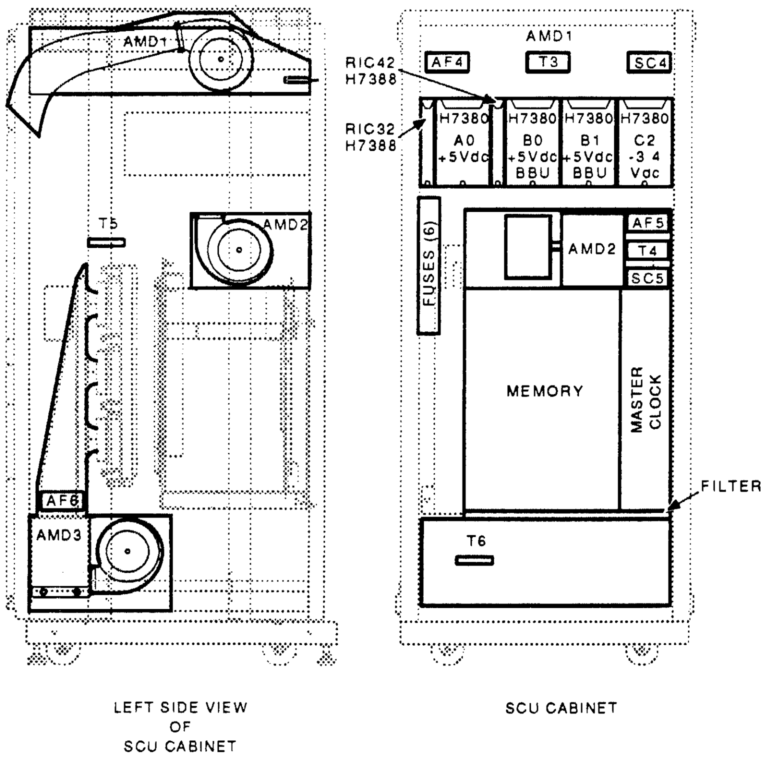


Figure 2–19 SCU Cabinet (Front and Side Views)

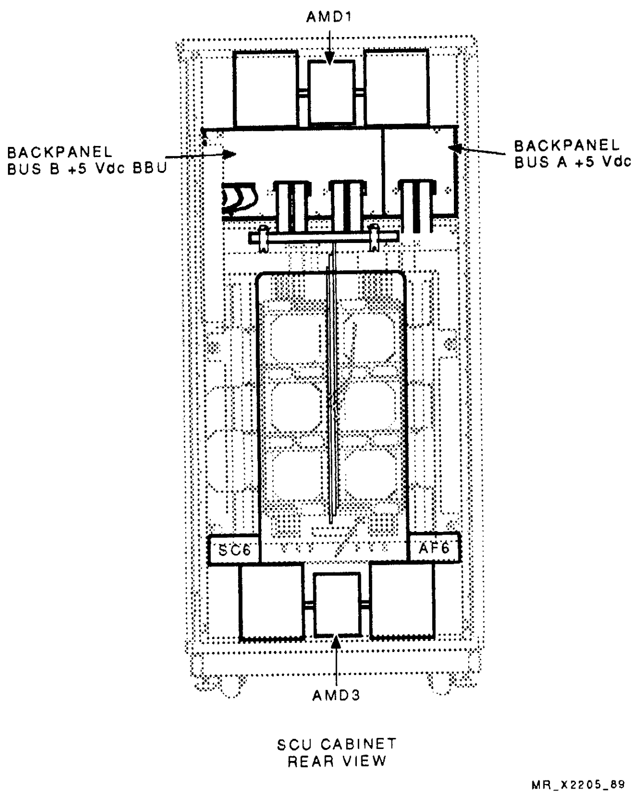


Figure 2–20 SCU Cabinet (Rear View)

2.2.1.2.3 Memory Subsystem

A maximum of two memory management units (MMUs) can be installed in model 200 systems. Each MMU contains four extended hex memory modules mounted in a card cage and connected by cables to the SCU planar module. The two card cages are in the front of the SCU cabinet (Figure 2–19) next to the MCM. The card cage closest to the MCM contains MMU0, and the card cage to the left of MMU0 contains MMU1.

2.2.1.2.4 Power Components

The SCU cabinet contains four H7380 power regulators and two RICs. The following list describes the power components and their associated buses.

- **H7380** — This converter, with its CPA groups, supplies -3.4 Vdc to the SCU and CPU planar modules.
- **H7380 BBU** — Two H7380 converters are part of the battery backup system. These two converters provide +5 Vdc for memory refresh during an ac power loss.
- **H7380 bus A** — This converter provides +5 Vdc on bus A for memory and clock modules.
- **H7388 (RIC)** — Two RICs monitor power subsystem status.

2.2.1.2.5 SCU Cabinet Cooling

The SCU cabinet contains three blowers that cool the logic components in this cabinet. All blower assemblies contain air flow sensors, temperature sensors, and speed sensors.

The SCU logic is cooled by a 550-CFM blower that is mounted below the planar module. Air is drawn from the bottom of the cabinet, passes through the shroud enclosing the planar module components, and exits through the rear of the cabinet.

The MMUs and the MCM are cooled by a 350-CFM blower that is mounted above the MMU card cages. Air is drawn through the card cages and the MMU, and it exits through the rear of the cabinet.

The SCU power components are cooled by a 550-CFM blower that is mounted above the power components at the top of the cabinet. This blower draws air through the power components and exhausts through the rear of the cabinet.

2.2.1.3 IOA Cabinet

The IOA cabinet in model 210 contains the SPU and the XMI card cage. The XMI card cage contains the I/O subsystem and power components that include the H7390 power front end (PFE), two battery backup units (BBUs), power regulators, and power converters. Figures 2–21 and 2–22 show front and rear views of the IOA cabinet.

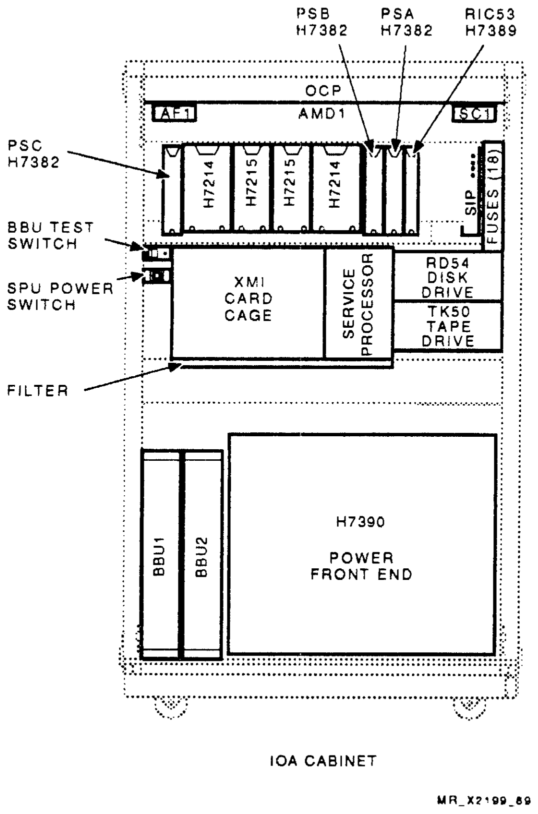


Figure 2–21 IOA Cabinet (Front View)

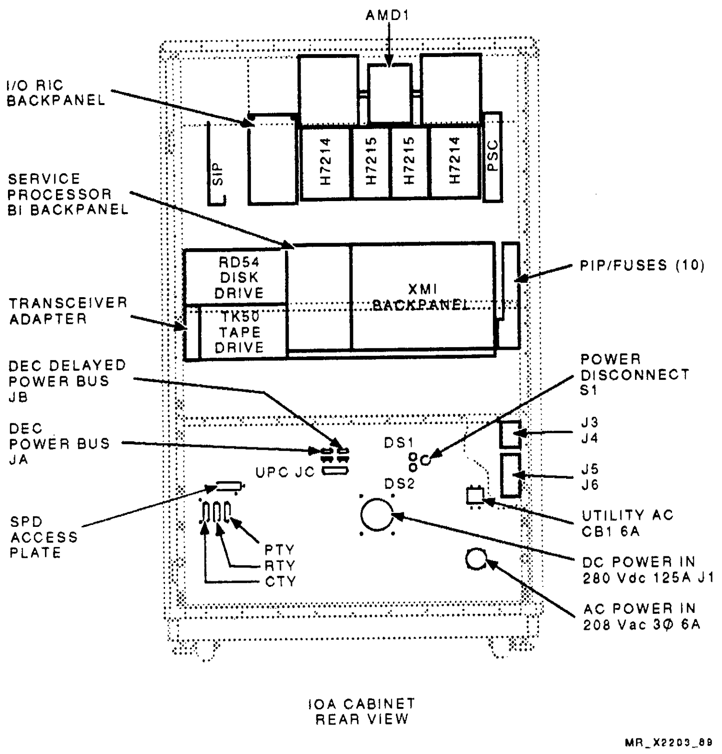


Figure 2–22 IOA Cabinet (Rear View)

2.2.1.3.1 Service Processor

The service processor modules are housed in a BI backplane that is located in the center of the cabinet (Figure 2–21). An RD54 disk drive and a TK50 tape drive are installed in this cabinet to the right of the service processor backplane and serve as the service processor load devices.

2.2.1.3.2 XMI Card Cage

The IOA cabinet contains a 14-slot XMI card cage that contains an I/O subsystem. Two XMI slots are dedicated to the XJA and the CCAR (timing and arbitration) modules. The remaining slots are populated with I/O adapters.

2.2.1.3.3 Power Components

The IOA contains four power converters that are different from the converters in the SCU and CPA cabinets. These converters are H7214 and H7215 converters and are used to power the XMI and VAXBI card cages.

Three H7382 bias supplies are in the IOA cabinet. The following list summarizes their functions:

- **H7382 PSA** — Supplies bias power to the XMI H7214/H7215 power converters and the H7390 PFE.
- **H7382 PSB** — Provides bias power to the RICs and power converters in bus A through bus D.
- **H7382 PSX** — Provides -15 Vdc to the Ethernet H4000 transceivers in the rear of the cabinet.

One RIC is in the IOA cabinet and monitors the H7390 or H7392, and the H7214/H7215 power converters that power the XMI card cage.

Two H7321 BBUs are in the lower left front of the IOA cabinet (Figure 2–21). The BBUs provide memory refresh power during power loss and also provide power to the time of year (TOY) clock and the diagnostic display (DD) LEDs of the operator control panel (OCP).

The H7390 PFE is in the bottom of the cabinet to the right of the BBUs (Figure 2–21). The PFE receives input ac power and converts it to high-voltage dc that is distributed to the power components.

The high-voltage dc that is output by the H7390 is connected to the power input panel (PIP). The PIP is mounted above the BBUs at the front left side of the cabinet (Figure 2–21) and provides a distribution point for high-voltage dc to the converters, bias supplies, and air movers in the three system cabinets.

Another distribution panel, the signal input panel (SIP), is in the IOA cabinet. The SIP provides an interface between the RICs and the power and environmental monitor (PEM) of the SPU. The SIP is in the bottom right corner of the IOA cabinet (Figure 2–22).

2.2.1.3.4 IOA Cabinet Cooling

The IOA cabinet contains a single 550-CFM blower for component cooling. The blower is mounted in the top of the cabinet above the power components. Air is drawn through the XMI and VAXBI card cages and the power components, and it exhausts through the top rear of the cabinet.

3 CPU Subsystem

This chapter defines each functional unit — the clock subsystem, IBox, MBox, EBox, and VBox — in the CPU subsystem. It introduces each functional unit, describes its major functions, and gives an operational overview.

The following topics are discussed in this chapter:

- Pipeline organizations
- Functional units of the CPU subsystem
- Physical layout, defined by multichip unit (MCU) and multichip unit/macrocell array (MCU/MCA) layout drawings
- Functional layout, defined by basic functional block diagrams
- Identification of MCUs and their major functions
- Data and control interfaces
- Functional unit operations and interactions

3.1 Clock Subsystem Introduction

The VAX 9000 clock subsystem generates and distributes system clock signals to the CPU and the system control unit (SCU) planar modules and to the XJAs of the I/O subsystem. The center of the clock generation is a microwave frequency synthesizer that provides two clock outputs: the master clock and the reference clock.

The service processor unit (SPU) provides clock control and receives error and status information from the clock. At power-up, the SPU initializes the clock subsystem.

The clock subsystem contains the master clock module (MCM) cables that deliver the clock outputs to the planar modules and the XJAs and cables that connect the control and status functions to the SPU.

3.1.1 Clock Functional Overview

Figure 3–1 shows a block diagram of the MCM and the distribution of the clock outputs throughout the system. The following sections provide a functional overview of the clock subsystem.

The MCM generates and distributes a master clock, a reference clock, and the clock control signals to the system. The MCM contains an interface to the SPU that transfers clock control and status information between the SPU and MCM. A single MCM provides the system clocks for any VAX 9000 configuration.

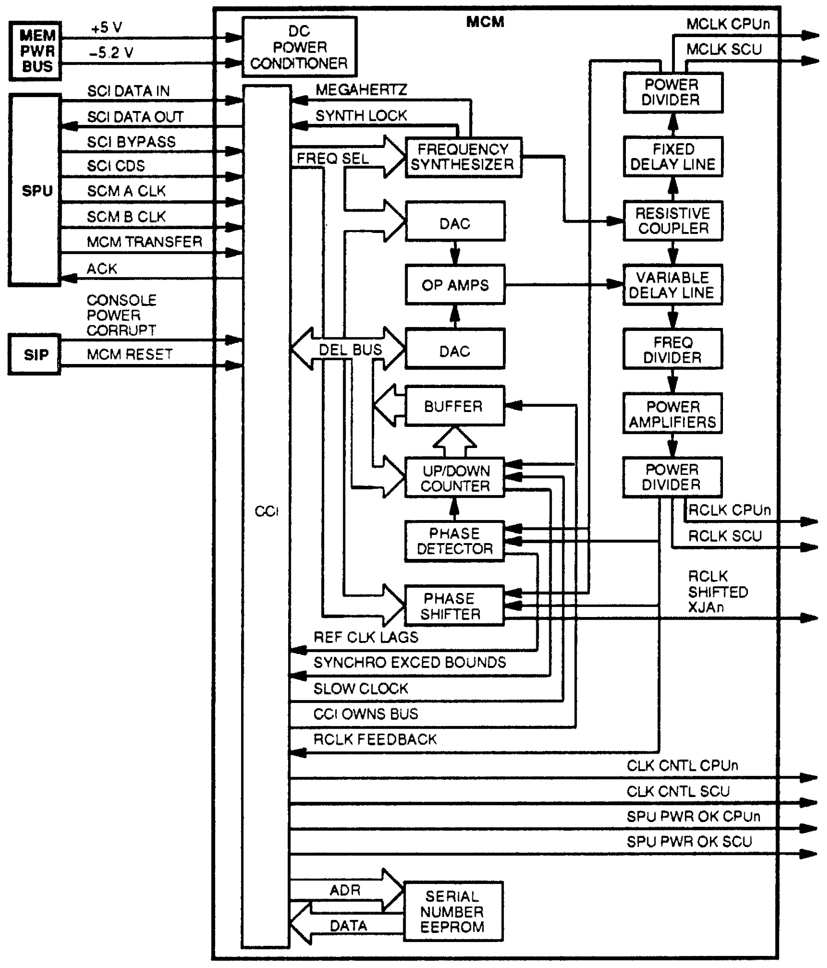


Figure 3–1 VAX 9000 Clock Subsystem

3.1.1.1 Clock Signal Overview

The following list describes the clock signals:

- **Master clock** — The master clock is sinusoidal and operates at a programmable frequency between 332 and 580 MHz. All system timing is referenced to the rising edge of the master clock.
- **Reference clock** — The reference clock is a square wave and operates at one-eighth of the master clock frequency. Each cycle of the reference clock is equal to one machine cycle.
- **XJA clocks** — The MCM generates phase-shifted reference clocks to compensate for data and clock transmission time differences between the SCU and the XMI-to-JBox adapter.
- **STRAM clock**— Master and reference clocks are used in the CDXX chips of the MCU to generated programmable-phase and eight-phase clocks for STRAM timing.

Figure 3–2 is a timing diagram that shows the relationship of the clocks.

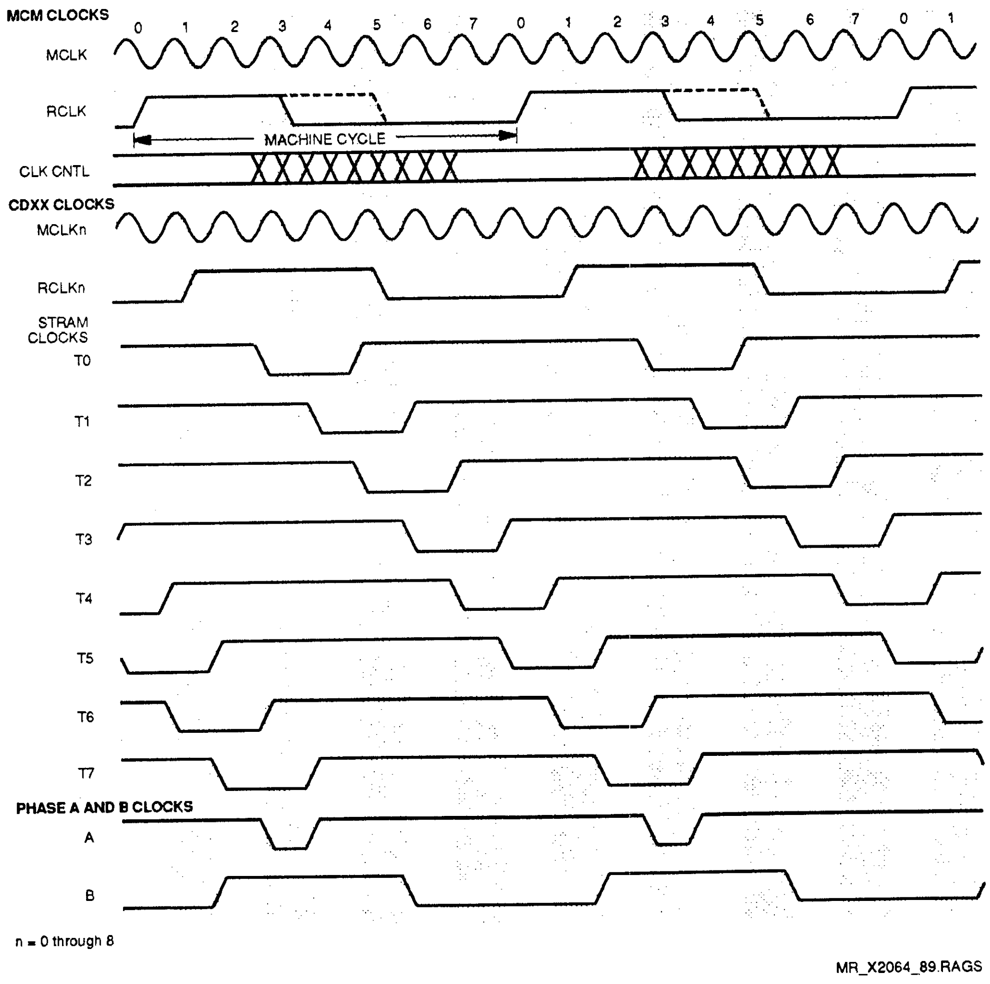


Figure 3–2 Basic Clock Relationships

3.1.1.2 Master Clock Module

The MCM contains a frequency synthesizer that outputs the master clock at an operating frequency between 320 and 580 MHz. The master clock usually runs at 500 MHz and provides the system clock timing.

The master clock generates a reference clock for the system. Dividing the master clock by eight generates the reference clock and it generates the system phase A, phase B, and STRAM clocks. The reference clock defines a machine cycle. Each cycle of the reference clock is equal to one machine cycle.

The MCM generates a clock for the XJAs. This clock is a phase-shifted reference clock that synchronizes the data transmitted from the XJA to the SCU.

3.1.1.3 Clock Control

The SPU clock control provides and sends signals that control the initialization and the operating modes of the clocks. Status information and error conditions are sent to the SPU by the MCM.

The MCM is initialized by a reset signal provided by the SPU. The reset signal is asserted when the power system becomes fully operational.

The control signals that set the clock subsystem to run in the operating modes are initiated as console commands. These control functions are passed to the MCM and stored in registers. The registers are read by the MCM, which performs the function that has been input to the system console.

Each planar module receives clock control signals that allow independent clock operations to be performed on that planar module. The following control functions can be performed:

- Stop the clocks.
- Run the clock on every machine cycle (normal operation).
- Run the clocks for a burst of machine cycles.
- Run the clocks at intervals for a burst of machine cycles.
- Run the clocks at intervals.

3.1.1.4 Clock Distribution

The master and reference clocks are distributed from the MCM power dividers to power dividers on the planar modules (1:4 for CPU and 1:3 for SCU). The planar module power dividers distribute the clock signals to the clock distribution (CDXX) chip of each MCU.

The CDXX chip outputs nine copies of the master clock and nine copies of the reference clock, and it uses these copies to generate 24 copies of the STRAM clock. The CDXX chip outputs are passed to the MCAs and STRAMs of the MCU.

The MCM clock control output is distributed to the CPU and SCU planar modules by connecting the signal to an MCU and then fanning out the signal to each CDXX on the planar module. In the CPU, an EBox MCU receives and distributes the control signal. In the SCU, a JBox MCU distributes the control signal.

Clock signals are distributed to the planar modules with semiflexible coax cable. The HDSC of the planar module contains a dedicated layer for clock signals. The clock signals are delivered to the CDXX chips by the signal flex connectors.

3.2 IBox Introduction

In the VAX 9000, the IBox is an independent functional unit that fetches and decodes instructions and their specifiers from the MBox and passes them to the EBox for execution. The EBox contains data and control functions that execute several instructions in one cycle.

The IBox provides instruction data (source, destination, program counter [PC], fork address, and source data and destination pointers) to the EBox. The instruction data is stored in the EBox source and destination queues. In a memory source operand, the IBox generates the operand address, passes it to the MBox, and requests a memory read operation. The MBox prefetches the data and then writes it into the EBox source list.

Using the source pointers provided by the IBox, the EBox accesses its source list and executes the instruction. Depending on the destination, the EBox writes the results to a general-purpose register (GPR) or transfers the results to the MBox to be later written to memory. In effect, the EBox deals only with data. (Opcodes or operand specifiers are not passed to the EBox by the IBox.)

The IBox can decode and store several instructions ahead of the EBox.

3.2.1 IBox Hardware Implementation

This section describes the major new hardware implementations in the IBox. Figure 3–3 shows the basic IBox block diagram.

3.2.1.1 Virtual Instruction Cache

An 8-Kbyte, direct-mapped, virtual instruction cache (VIC) reduces the number of I-stream requests issued to the MBox. A virtually addressed cache eliminates the need for address translation and translation logic. The VIC is also flushed on every REI so that writes to the MBox data cache are ignored by the IBox.

3.2.1.2 Extended Instruction Buffer

The IBox incorporates a 25-byte instruction buffer that presents 9 bytes of I-stream to the decode units of the IBox and contains an additional 16 bytes of prefetched I-stream that replenishes I-stream bytes as they are decoded. The instruction buffer is divided into three buffers: the IBUF, which contains the 9 bytes of I-stream to be decoded, and the IBEX and IBEX2, each of which contain 8 bytes of prefetched I-stream.

3.2.1.3 Branch Prediction

When branch instructions are encountered in the I-stream, the IBox predicts the direction of the I-stream. The IBox redirects the I-stream by taking a branch or it continues decoding sequential I-stream by not taking a branch.

To minimize the time spent flushing and refilling the pipeline after every branch instruction, the instruction decode stage includes a branch prediction cache (BPC). This 1K virtual cache increases performance by storing information about the branch validity and the target address. When a branch instruction is decoded, it is referenced in parallel in the BPC and a prediction is made whether to take the branch. The IBox uses the cached target address to redirect the instruction fetch stage to the new I-stream if the branch is taken. For performance reasons, the BPC is not flushed.

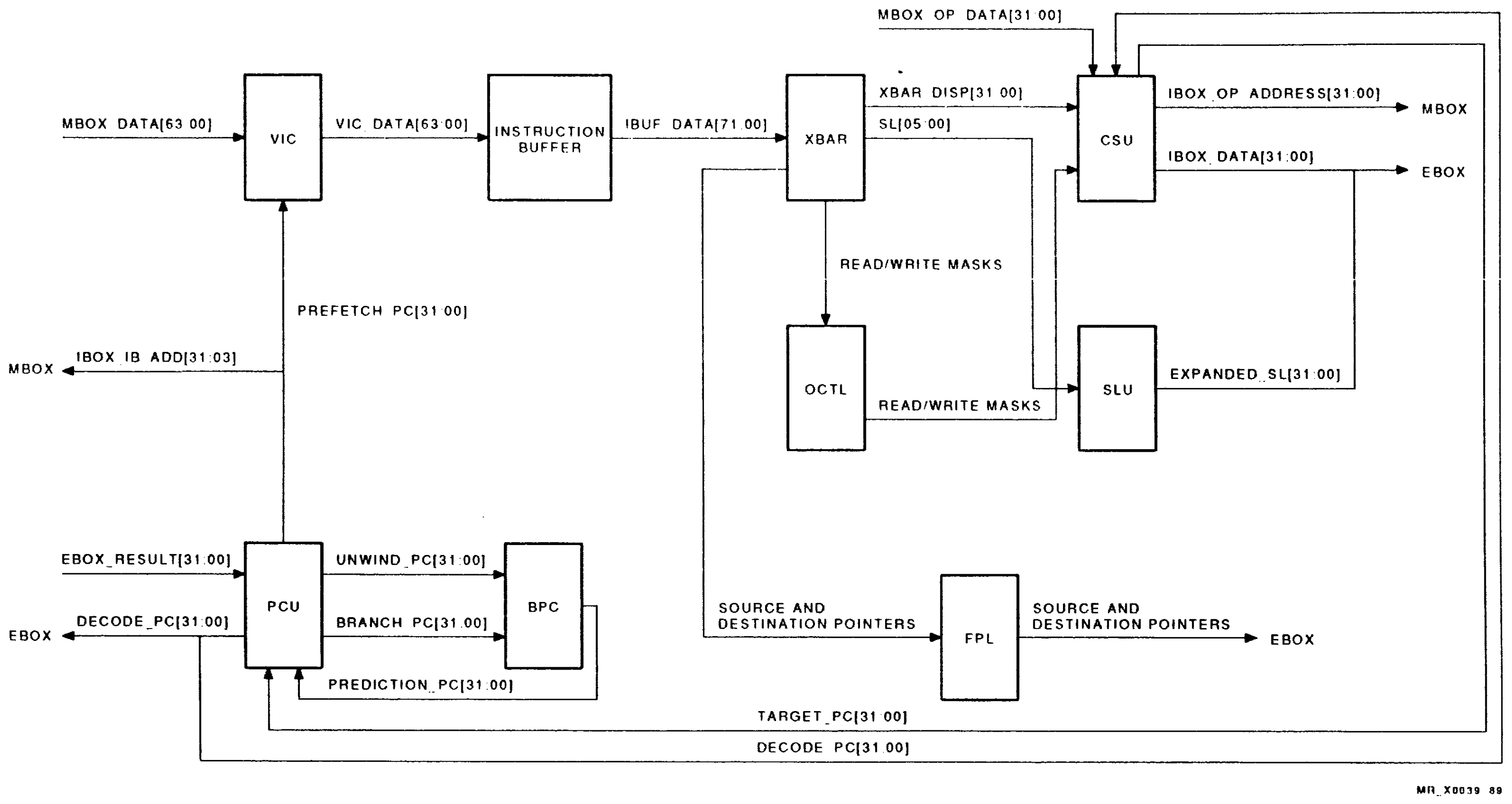


Figure 3–3 Basic IBox Block Diagram

3.2.1.4 Multiple Specifier Decode Unit

The multiple specifier decode unit (XBAR or crossbar) is a set of multiplexers that simultaneously decodes specifiers.

The actual number of specifiers decoded depends on the specifier type. The XBAR can decode up to three specifiers (for example, two simple specifiers and one complex specifier, or three simple specifiers). Simple specifiers are register mode or short literal, while all other specifiers are complex.

The XBAR determines the number of specifiers and the number of bytes the specifiers occupy in the instruction buffer. The XBAR also controls the shifting and validating of the I-stream bytes in the instruction buffer.

The XBAR contains logic that detects interinstruction read conflicts (IRC). These conflicts occur when an instruction directs the EBox to read a register that has been updated by a previous specifier in the same instruction. The XBAR directs the IBox to special instruction handling routines when these conflicts occur.

The XBAR contains a decode RAM (DRAM) that decodes the opcodes of instructions and produces the specifier counts, the specifier access type, and the specifier data type. These outputs provide the basis for the number of specifiers that are decoded in each cycle and they determine when the instruction decoding is complete.

3.2.1.5 Specifier Handlers

The IBox contains three dedicated specifier handlers:

Complex specifier unit (CSU)
Short literal unit (SLU)
Free pointer logic (FPL)

The three specifier handlers operate in parallel and produce pointers for two source operands and one destination operand in a single cycle (depending on the instruction).

3.2.1.5.1 Complex Specifier Unit

The complex specifier unit (CSU) is responsible for evaluating complex specifiers. The XBAR supplies the CSU with the register, register index, and up to 32 bits of displacement. The CSU calculates branch target addresses, immediate operands, and memory addresses of operands supplied to the EBox from the MBox.

The CSU contains the operand processing unit (OPU) port interface to the MBox and the interface to the EBox. Virtual addresses of the operands are sent to the MBox, while the EBox receives the immediate operands from the CSU.

The CSU also contains the IBox GPRs. These GPRs are read and written by the CSU, and are written by the EBox.

3.2.1.5.2 Short Literal Unit

The short literal unit (SLU) receives decoded short literal specifiers from the XBAR and expands them for entry in the EBox source list. The SLU produces a single longword of expansion for each cycle. The literal expansion depends on the specifier data type.

3.2.1.5.3 Free Pointer Logic

The free pointer logic (FPL) manages the EBox source list pointer. The FPL tracks the free source list addresses and associated pointers. It establishes the correct source 1 and source 2 pointers for operands the EBox uses to execute an instruction. The FPL also generates the correct destination pointer for an instruction result.

3.2.2 IBox Physical Organization

The IBox logic is physically located in three MCUs (Figure 3–4). This section introduces each MCU and its related MCAs.

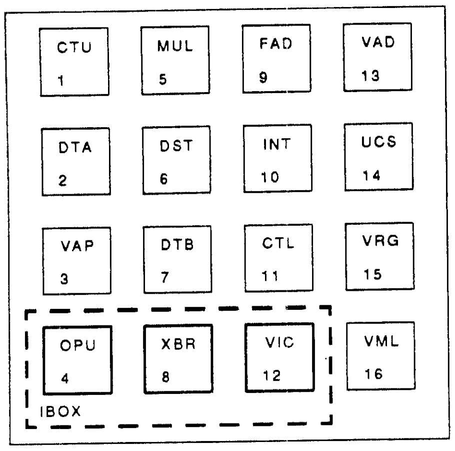


Figure 3–4 Planar Module Layout

3.2.2.1 VIC MCU

The virtual instruction cache (VIC) MCU contains two MCAs and forty-two 1K × 4-bit STRAMs. Two bit-slices of the program counter and the data STRAMs for branch prediction and VIC are resident in this MCU.

The following list introduces the MCA and STRAM functions of the VIC MCU:

- **PCBP MCA** — The program counter and branch prediction control MCA contains bits [07:00] of the PC and provides branch prediction control.
- **PCVC MCA** — The program counter and VIC control MCA contains bits [15:05] of the PC and provides VIC control.
- **VIC data STRAMs** — These eighteen 1K × 4-bit STRAMs are dedicated to the VIC data and associated byte parity.
- **Branch prediction STRAMs** — These twenty-four 1K × 4-bit STRAMs are dedicated to the branch prediction function. The STRAMs store the branch PC tag, branch instruction length, prediction PC, and prediction bit.

3.2.2.2 XBR MCU

The crossbar (XBR) MCU contains the IBUF, the XBAR, two of the four PC slices, and the tag STRAMs for the VIC.

The following list introduces the MCA and STRAM functions of the XBR MCU:

- **PCLO MCA** — The program counter low MCA contains bits [23:13] of the PC.
- **PCHI MCA** — The program counter high MCA contains bits [31:24] of the PC.
- **IBFA MCA** — The instruction buffer A MCA contains the low-order nibble of the instruction buffer. The IBFA also provides the VIC hit logic.
- **IBFB MCA** — The instruction buffer B MCA contains the high-order nibble of the instruction buffer. Parity checking is performed by combining IBFB parity with a partial parity from the IBFA.
- **XDTA MCA** — The XBAR data A MCA contains the low-order nibble of the XBAR. The major MCA outputs are displacements for the OPU.
- **XDTB MCA** — The XBAR data B MCA contains the high-order nibble of the XBAR data path.
- **XSCA MCA** — The XBAR control MCA is the XBAR control unit. It receives I-stream data from the IBUF and performs simple instruction decoding. The instruction buffer shift control is generated from the number of specifiers decoded and the number of specifiers the instruction contains.
- **VICT STRAMs** — Five of the nine 1K × 4-bit STRAMs contain the 19-bit VIC tags. Two of the STRAMs provide the 4-bit VIC quadword valid fields and associated parity bits and the parity for the VIC quadword valid bits. The remaining two STRAMs provide the VIC block valid field.

3.2.2.3 OPU MCU

The operand processing unit (OPU) MCU contains the logic responsible for the specifier decode process. The operand port interface to the MBox resides in this MCU. The MCU also contains a pair of STREGs that provide the IBox GPRs.

The following list introduces the MCA and STRAM functions of the OPU MCU:

- **OPUX MCA** — The OPUA and OPUB MCAs provide the data path for the SCU. OPUA provides the low-order word, and OPUB provides the high-order word. The SCU receives up to 32 bits of displacement from the XBAR and directs the outputs to the MBox, the EBox, or a loopback to the SCU.
- **OSQA MCA** — This MCA provides control for the GPR STREGs and the OPUA and OPUB multiplexers (AMUX and BMUX).
- **OSQB MCA** — This MCA receives short literal, source, and destination data from the XBAR. Short literal specifiers are expanded into the correct context and passed to the EBox. The source and destination pointers are also passed to the EBox.
- **OCTL MCA** — The operand control MCA provides control for the scoreboard function, as well as flush signals, to other IBox functional units.

3.2.3 IBox Interfaces

The IBox interfaces to the EBox, MBox, VBox and the SPU. This section provides a brief description of the functions of each interface.

3.2.3.1 MBox Interface

The IBox interfaces to the MBox through two ports in the MBox:

- **IBUF port** — Requests data from the MBox when a miss is encountered in the VIC.
- **OPU port** — Requests memory-related operands from the MBox and passes virtual addresses of operands, which specify memory destinations to the MBox.

3.2.3.1.1 Instruction Buffer Port

The IBUF port is a read-only port to the MBox. The IBox uses the IBUF port to issue requests to the MBox for I-stream data. The I-stream is retrieved from the MBox cache. In the case of a cache miss, the request is forwarded to memory through the SCU. A request is normally issued for four (aligned) quadwords to fill the VIC. I-stream requests are initiated by the IBUF on a VIC miss.

3.2.3.1.2 OPU Port

The OPU port is a read/write operand access port to the MBox. The port has a 32-bit wide data path with byte parity. Rotation of the data (justification) is performed by the MBox. The OPU port provides the following functions:

- Operand prefetch from cache or memory on behalf of the EBox and the VBox
- Queuing of addresses for operands destined to cache or memory
- Prefetching operands that are address-deferred (indirect) from cache or memory

3.2.3.2 EBox Interface

The EBox-to-IBox interface transfers the following data and control signals:

- Result data
- Starting or flushing PC
- RLOG unwind data
- Control information

  Branch valid
  Queue full
  Keep masks

This interface transfers 32-bit EBox result data (including byte parity) to the IBox. The result data is usually an operand that has a GPR destination. The data, including the byte parity, is written to the IBox GPR set. The EBox result data may also be a controlling function (for example, an address passed to the IBox to initiate an instruction fetch).

3.2.3.3 Service Processor Unit Interface

The SPU interface connects the IBox to the SPU through the scan latches in the IBox data and control paths and the error logic. The SPU can retrieve and store the state of the IBox scan logic for error reporting and possible recovery. Customer Services can then perform analysis on the stored error symptom data for identification of the failing FRU.

The scan paths can implement symptom-directed diagnosis (SDD) and test-directed diagnosis (TDD). The scan latches have scan data and clock inputs and the normal system data and clock inputs. The scan inputs are controlled by the SPU and the diagnostics to shift in test patterns and to test for the correct pattern that is later retrieved.

3.2.3.4 VBox Interface

The VBox issues requests for operands through the IBox. It sends the address (32 bits) with byte parity and control data. Control data is the reference data size and the type of reference (read or write), and it indicates whether the data is a block read.

3.2.4 IBox Pipeline Overview

The IBox consists of three main pipeline stages that closely relate to the MCU physical structure. Each stage generates indicators to aid in isolating errors to an MCU FRU. The three pipeline stages are defined as follows:

- **Instruction fetch** — This stage consists of the logic involved in fetching the I-stream before it is latched in the instruction buffer and includes the following components:

  VIC
  Program counter unit (PCU)
  IBEX, IBEX2, and part of the IBUF
  Instruction buffer-to-MBox interface

- **Instruction decode and branch prediction** — This stage consists of the logic involved in decoding the instruction in the IBUF and includes the following components:

  XBAR
  Branch prediction unit (BPU)

- **Specifier decode** — This stage consists of the OPU logic involved in the evaluation of operand specifiers and includes the following components:

  SCU
  Branch prediction logic
  FPL
  SLU
  Operand control unit (OCTL)
  OPU-to-MBox interface
  OPU-to-EBox interface

When a stage cannot complete its operation, previous stages must be stalled because the IBox is pipelined. (That is, previous stage operations are halted.)

Figure 3–5 shows the IBox pipeline decoding sequential instructions where stalls are not encountered in the pipeline stages. The following events occur in each of the machine cycles:

- **First cycle** — The IBUF is loaded with the I-stream to be presented to the decode units of the IBox.
- **Second cycle** — The logic representing the decode pipeline stage decodes the opcode and the specifier bytes in the IBUF, and passes them to the specifier handlers. The logic representing the fetch stage replenishes the decoded bytes of the IBUF.
- **Third cycle** — The decoded specifiers are processed by the specifier handlers and are passed to the EBox. The decode logic and the fetch logic continue to perform their operations simultaneously.

The parallel operations of the three pipeline stages continue until one of the units stalls or the IBox is flushed to a new I-stream. A stall in one of the units is caused by instruction specifiers that require more than one cycle to be processed or by conflicts in the instructions. For example, an instruction that contains two sequential complex specifiers and the first specifier requires more than one cycle to be processed in the specifier pipeline stage.

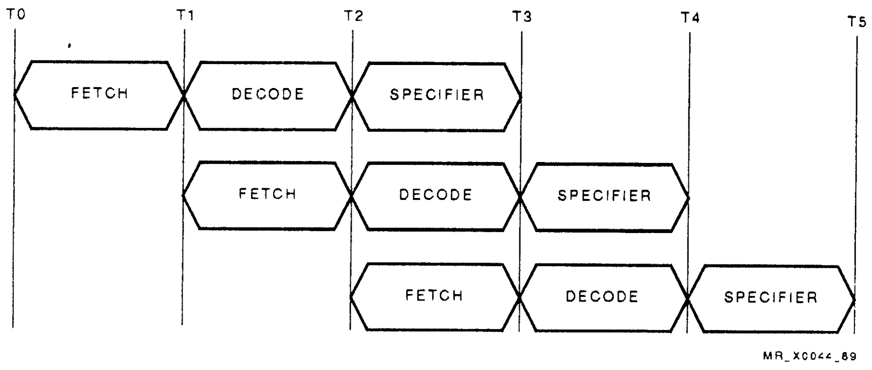


Figure 3–5 IBox Pipeline with No Stalls

3.3 MBox Introduction

In the VAX 9000, the MBox serves as the CPU interface to its local cache, to other CPU caches, and to main memory. The MBox is a multiport unit that receives requests from three external ports. The MBox consists of a translation buffer (TB) and a data cache. The MBox accepts virtual memory references and translates virtual addresses to physical addresses. The MBox accesses the memory data either directly in local cache or indirectly through the system control unit (SCU). The SCU interfaces to main memory, to the I/O, and to other processors.

The MBox includes the following features:

- A 128-Kbyte, 2-way set associative (2 separate sets) write back data cache. The cache block size is 64 bytes (8 quadwords). The cache tag store has an entry per block, or a total of 2K entries. Each longword has a valid bit, or a total of 16 valid bits per block. The maximum access width for cache reads and writes is 8 bytes, since each cache set is 8 bytes wide.

  The cache block is only copied back to main memory if:

  - The cache block is needed for other data.
  - Another CPU requests the cache block and the block has been modified and does not match the data in physical memory.
  - The CPU requests a cache sweep.
- Error correction codes (ECCs) for single-bit error correction and double-bit error detection in the data cache only.
- Cache tag STRAM parity.
- A 1-Kbyte TB that is direct mapped. The TB has 512 locations for process space and 512 locations for system space.
- A TB fixup unit that accesses memory management registers, fetches valid page table entries (PTEs), and resolves TB misses.

Figure 3–6 shows the following:

❶ TB I/Os

❷ Data cache address sources and data I/Os

❸ TB fixup unit I/Os

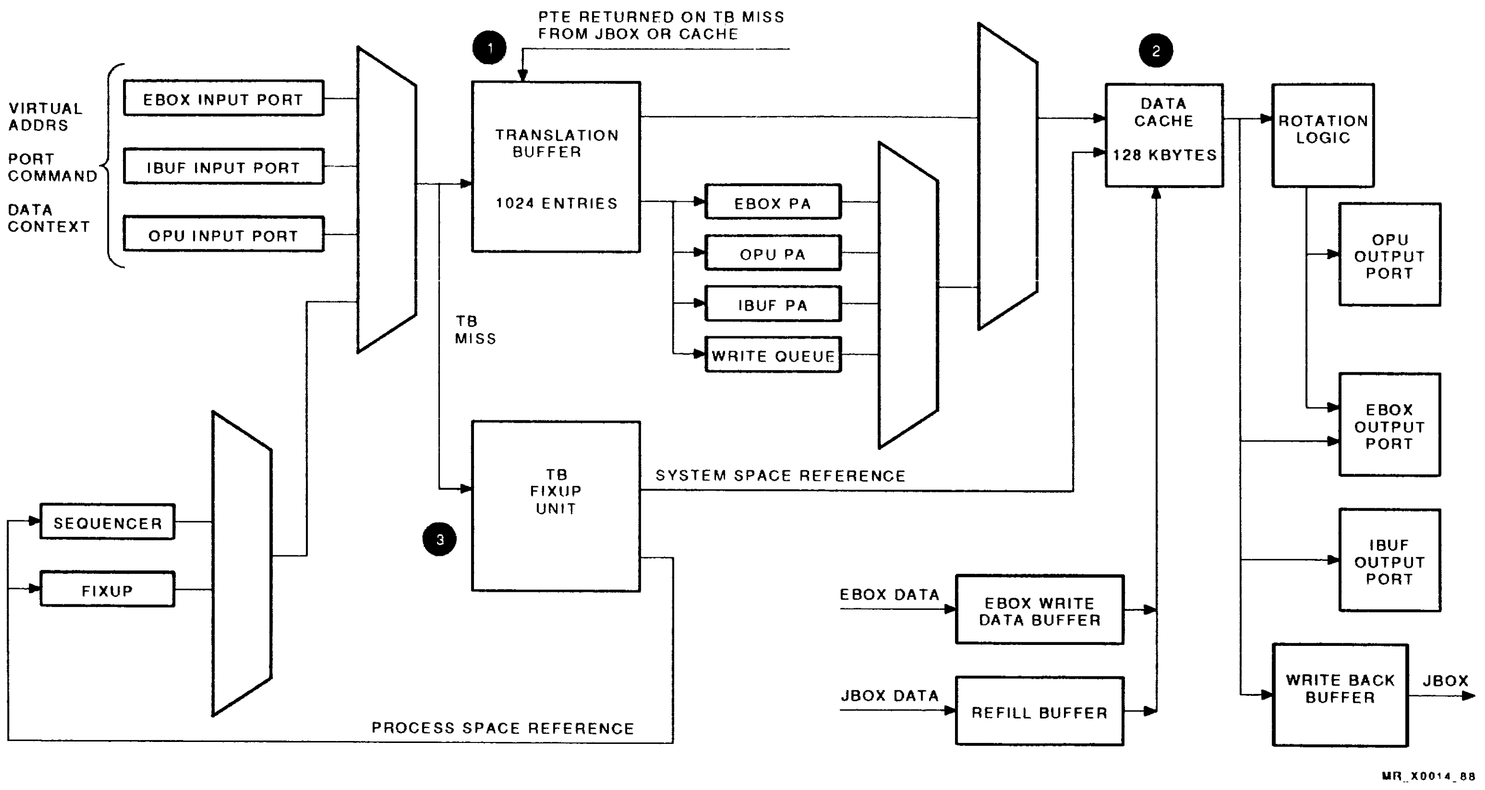


Figure 3–6 MBox Block Diagram

3.3.1 MBox Physical Organization

The MBox is partitioned across the following four MCUs: (Figure 3–7 shows the location of the four MBox MCUs in the CPU planar module.)

- **Virtual address port (VAP)** — This MCU contains the TB and the port arbitration decode and control logic. The VAP has 5 MCAs and 26 STRAMs.
- **Data cache (DTA and DTB)** — These two MCUs contain the data cache and the cache I/O buffers, including the refill buffer and the rotation logic. Each MCU has 3 MCAs and 40 STRAMs.
- **Cache tag unit (CTU)** — This MCU contains the cache tag control that determines whether the requested data is valid and is located in local cache (cache hit). The CTU also contains the JBox command encode logic and write back buffer and has four MCAs (nineteen 1K × 4 STRAMs and eight 4K × 4 STRAMs).

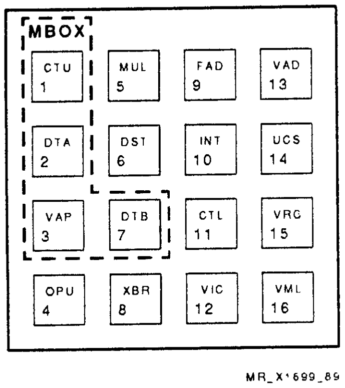


Figure 3–7 MBox MCUs

3.3.1.1 Virtual Address Port

The VAP MCU contains the TB and performs port arbitration and command decode. The VAP consists of the following five MCAs:

- **VAPO** — This MCA is the VAP of the MCU and performs arbitration for the TB. VAPO latches VA bits [11:00] and decodes the commands received from the two IBox ports, IBUF and the OPU; the EBox port; the fixup port; and the sequencer port.
- **FXUP** — This MCA receives VA bits [31:12] from all the ports and contains the TB fixup unit. The TB fixup unit accesses memory management registers and fetches valid PTEs for a VAP that the TB could not translate.
- **FALT** — This MCA performs the translation match and control and handles memory management faults. FALT controls writing and invalidating TB STRAMs. This MCA receives cache data when accessing PTEs from memory.
- **WRTQ** — This MCA contains the write queue, which has eight address entries and their corresponding status information.
- **CCSQ** — This MCA controls the cache operations and sends control signals to the CTU, DTA, and DTB.

The VAP STRAMs are as follows:

- **TB STRAMs** — The TB has twenty-one 1K × 4 STRAMs, which include the four copies for bits [15:09]. Four STRAMs are a second copy of bits [31:16]. Table 3–1 lists the STRAMs and their corresponding bits and definitions.

Table 3–1 VAP STRAMs

Name	Bit	Number of STRAMs
Valid	15:00, P	5
PFN	23:00	6
CTMA lookup	39:09	6
PAD0 lookup	15:09, P	2
PAD1 lookup	15:09, P	2
Tag, tag parity, modify, PFN parity	12:00	4
Protection	03:00	1

3.3.1.2 Data Cache

The following list describes the MCAs on the DTA MCU and the DTB MCU:

- **PADX** — These MCAs (PAD0 and PAD1) drive the address lines and the write enables for the cache data STRAMs. PAD0 is on the DTB, and PAD1 is on the DTA.
- **DTMX** — These MCAs (DTM0, DTM1, DTM2, and DTM3) buffer and control writes and reads for the cache data STRAMs. DTM0 and DTM1 are on the DTB, and DTM2 and DTM3 are on the DTA. Each DTMX deals with two byte-slices of the quadword interface. These MCAs contain byte-slices of the refill buffer and the rotator.

Table 3–2 lists each DTMX and its corresponding bytes.

Table 3–2 DTMX Byte-Slices

DTMX	Bytes
DTM0	0, 4
DTM1	1, 5
DTM2	2, 6
DTM3	3, 7

The 128-Kbyte cache has eighty 4K × 4 STRAMs with forty STRAMs on the DTB and forty STRAMs on the DTA.

3.3.1.3 Cache Tag Unit

The following list describes the MCAs on the CTU MCU:

- **CTMA** — This MCA, along with the CTMV, controls the cache tag STRAMs, performs address comparison, and interfaces to the JBox. CTMA receives PA [32:06] and drives the cache tag STRAM address lines. CTMA is partially responsible for tag matching, for generating a cache hit, and for assembling a command and address to send to the JBox.
- **CTMV** — This MCA controls the 16 valid bits (1 valid bit per longword in a 64-byte cache block) and generates the port responses. CTMV is partially responsible for generating cache hit and assembling the command and address to send to the JBox.
- **WBEM** — This MCA contains word-slices of the write back buffer. WBEM generates ECC check bits by using the partial ECC from the WBES, compares stored ECC against recalculated ECC, and generates syndrome bits for bit correction. WBEM also generates ECC control bits and sends them to the WBES. WBEM differentially drives data to the JBox, receives the command and address from the JBox, and forwards this information to the CTMA and CTMV.
- **WBES** — This MCA contains word-slices of the write back buffer. WBES generates partial ECC, which is sent to the WBEM, and receives ECC control signals from WBEM for bit correction. This MCA differentially drives data to the JBox, receives the command and address from the JBox, and forwards this information to the CTMA and CTMV.

The following list describes the CTU STRAMs:

- **Tag STRAMs** — There are nineteen 1K × 4 tag STRAMs: nine STRAMs for cache SET0, nine STRAMs for cache SET1, and one STRAM for the LRU bit. The format for each entry in the tag store consists of physical address bits [32:16], valid bits [15:00], a written bit, and two parity bits.
- **ECC STRAMs** — There are eight 4K × 4 STRAMs, four STRAMs for the ECC SET0 and four STRAMs for the ECC SET1.

3.3.2 MBox Interfaces

The MBox contains interfaces to the EBox, the IBox, and the SCU. The EBox interface sends virtual and physical references to the MBox. The IBox uses two ports to access memory data from the MBox. The IBUF port requests I-stream for the VIC. The OPU port requests reads of indirect addresses and operands, or receives addresses for writes. The SCU interface is used for cache refill and write back and get data operations. The MBox sends commands and addresses to the SCU for these data operations.

3.3.2.1 MBox-to-EBox Interface

The EBox interface functions include the following:

Read and write MBox registers
Perform memory management functions
Read and write physical addresses for I/O reads and writes

The EBox interface contains a 32-bit address, a 5-bit control field, and a 4-bit tag field. The 32-bit address path is also the data path for transferring the register contents to the memory management registers during a move to processor register (MTPR) instruction. The control field defines whether the EBox is making physical or virtual references. The EBox uses the tag to select memory management registers.

3.3.3 MBox Data Cache

Each CPU has a two-way set associative write back cache. The two sets are 0 and 1 with each containing 64 Kbytes of data. Each cache block is 64 bytes long and contains the following:

- One valid bit for each longword, for a total of 16 valid bits
- One written bit for the entire block

The valid and written bits are stored in a cache tag in the cache tag store. Figure 3–8 shows the format of the cache data found at each location in the data cache. Each cache set is 8K lines deep and 8 bytes wide. Each location consists of 8 bytes of cache data, a parity byte, and an ECC byte.

Figure 3–9 shows the relationship between the cache tags and the data cache in each CPU. Each cache reference results in the MBox addressing the cache tag to determine if the physical address that is being referenced is in cache set 0 or cache set 1. When a miss occurs in both cache sets, a cache refill is initiated.

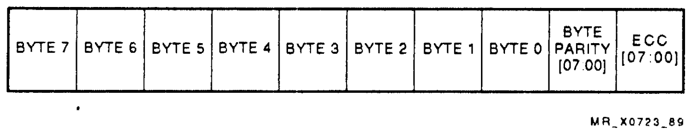


Figure 3–8 Cache Data (Quadword)

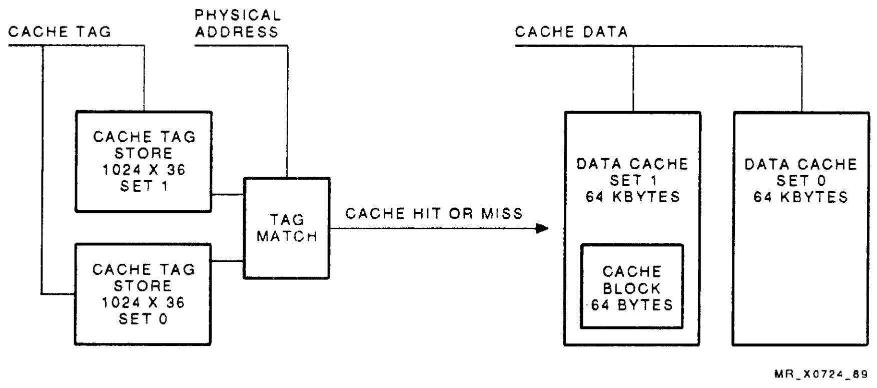

CACHE TAG
PHYSICAL ADDRESS
CACHE DATA
CACHE TAG STORE 1024 X 36 SET 1
CACHE TAG STORE 1024 X 36 SET 0
TAG MATCH
CACHE HIT OR MISS
DATA CACHE SET 1 64 KBYTES
CACHE BLOCK 64 BYTES
DATA CACHE SET 0 64 KBYTES
MR_X0724_89

Figure 3–9 Cache Tag Store and Cache Sets

3.3.4 MBox Data Traffic Manager

The data traffic manager manages the flow of data within the MBox. The data traffic manager routes the data received from the JBox to the ports and to the cache STRAMs and routes the data from the EBox, IBox, and cache to the JBox.

When performing unaligned data transfers, the data must be routed through rotation, shift, and merge logic. When aligned writes are performed, the data is addressed and sent on quadword boundaries directly to the merge logic and to the requesting port.

The data traffic manager controls selection of rotation and merge logic based on the starting address and the context of the requested data. The data traffic manager also detects requests that cross quadword boundaries and buffers and merges data for these unaligned operands.

The instruction buffer and fixup ports do not require data alignment. The instruction buffer port receives aligned quadwords and the fixup port can request aligned reads from cache to fetch PTEs. OPU and EBox reads and EBox writes may require data alignment.

3.3.5 Hardcoded Microwords

The MBox logic contains two hardcoded microwords: CCSQ microword and FXUP microword. Hardcoded microwords are hardwired in the associated MCAs and are not loaded into STRAMs.

The CCSQ microword controls refill, write back, and instruction buffer prefetches. It also determines the cache function as read or write. The FXUP microword controls memory management register selection, fault detection, request types, and fixup ALU length check functions.

3.3.6 MBox Operational Summary

This section provides an operational summary of some of the functions of the MBox.

3.3.6.1 Cache Refill

A cache refill is required when a requester generates a read or write request that results in a cache miss. To supply the correct data to the requester, the MBox must request a read refill or a write refill request for a block of data (64 bytes) to the SCU. The following steps summarize a cache refill operation:

1. The cache tag unit detects a cache miss and assembles a command and address to request a block of data from main memory.
2. The SCU accesses main memory, obtains the requested data, and assembles a corresponding command and address to return the data to the MBox.
3. The SCU loads the data into the refill buffers of the MBox 8 bytes per cycle in 8 consecutive cycles.
4. The cache tag unit reads the valid bits if the refill is to replace a partially valid or written block and the refill data is selected as cache write data.
5. The cache tags are updated when the last cycle of the write is complete.

3.3.6.2 Translation Buffer Operations

The translation buffer performs two operations, clearing the translation buffer and handling TB misses. The translation buffer is cleared to invalidate entries as the result of a context switch. Handling a TB miss requires fetching a PTE from cache and writing it to the translation buffer.

For the MBox to address memory or cache, the virtual address must be translated to a physical address. Because the translation buffer uses a PTE for every memory reference, the recently used PTEs are stored in the translation buffer. The translation buffer cannot store the complete copy of the page tables and a translation may occasionally miss in the translation buffer. Before the MBox can complete the request, the TB miss needs to be resolved by fetching the PTE from the page tables in cache or main memory.

The MBox initiates a write TB PTE request to service a TB miss. This request requires fetching a PTE from the system page tables in the array, formatting the PTE, and writing it into the translation buffer so that the translation can be retried.

The write PTE request does not need to distinguish between per-process or system space. At the start of the write PTE cycle, the virtual address port logic and the fixup unit select the address of the request that caused the TB miss.

In the cycle after the TB miss, the fixup unit checks bits [31:30] to determine which space the address is in and switches to one of three microcode flows (P0, P1, or system). Each has all the memory management registers hardcoded into the flow.

The TB fixup unit fetches PTEs from cache whenever the translation buffer detects a miss. PTEs are returned to the translation buffer, which inserts the new entry and tests for accessibility and translation-not-valid faults. The process of fixing misses in the translation buffer is not interruptible; no other ports have access to the translation buffer during miss processing.

3.3.6.3 OPU Port Operations

The OPU port prefetches memory-related operands for the IBox and does not have any data input to the MBox. The OPU port asserts a request and sends it with a 32-bit address to the MBox. The request also contains a function code selecting one of three codes:

Operand fetch for the EBox
Indirect fetch of a read operand
Indirect fetch of a write operand

The OPU port can send byte addresses and will receive the data in the correctly aligned format.

The operand request is gated into the TB and initiates a request pending state in the MBox port logic. The request pending enables latching of the fields associated to the request (context, indirect, tag, and so on) into the TB and latch a portion of the address into the fixup unit.

The MBox responds to the IBox to acknowledge that the request has been received.

The port request arbitrates for TB control. In the TB, the virtual address is compared in the valid, tag, and PTE STRAMs and a TB hit or miss is determined. A hit indicates that the PTE needed to translate the virtual address to a physical address is in the TB PTE store. When a miss occurs, the fixup unit fetches the new PTE and writes it to the TB STRAMs.

After the address is translated, a cache lookup is performed. The lookup requires checking cache sets 0 and 1 for the data referenced by the physical address. The cache STRAMs and tags are addressed to determine a cache hit or to determine if a cache refill is required.

The cache output is handled by the data traffic manager. Cache output data is latched and rotated when appropriate and then sent to an appropriate output port. Data returning to the OPU port is sent through the rotate data port.

3.4 EBox Introduction

The instruction fetching and decoding operations performed by the IBox and the MBox allows the EBox to be dedicated to the execution stage of an instruction. The EBox can execute several instructions simultaneously. While most VAX processing units have dedicated hardware for integer, floating-point, and multiply operations, this system can operate all processing units in parallel.

3.4.1 EBox Hardware Implementation

As shown in Figure 3–10, the EBox is organized into several independent functional units. The major functional units are introduced in the following sections.

3.4.1.1 Instruction Data Queues

Control information and operands are transferred from the IBox (and in some cases the MBox) to a set of FIFO buffers (queues) in the EBox. The queues decouple (buffer) the EBox from the IBox. With each instruction, the IBox supplies the following:

- A microcode dispatch address (fork address), stored in the fork queue
- A set of source operand pointers, stored in the source pointer queue
- A set of result store pointers, stored in the destination pointer queue

Four queues are maintained in the queue logic: fork, source, destination, and PC. The queues are loaded at the location designated by the associated queue load pointer. Each queued item contains associated valid bits.

- **Fork queue** — The fork queue is a 17-bit wide × 8-location queue. The queue locations contain a 16-bit fork address field, and an IBox prediction bit. The bit fields are separated and distributed to the appropriate EBox logic.
- **Source queue** — The source queue is a 5-bit wide (field) × 16-location queue. The high-order field bit specifies a GPR location if set, and a source list location if clear. The low-order four bits specify the GPR or source list location.
- **Destination queue** — The destination queue is a 5-bit wide (field) × 8-bit location queue. The high-order field bit specifies a destination GPR or memory location. The low-order four bits specify the GPR or memory location.
- **PC queue** — The PC queue is a 32-bit wide × 8-location queue that stores the macro PCs. The microcode can access either the current PC or the backup PC.

All queues are cyclic and receive load and remove pointers from the queue logic. The queue logic informs the IBox when the queues become full and controls flushing the queues for exceptions and bad branch predictions.

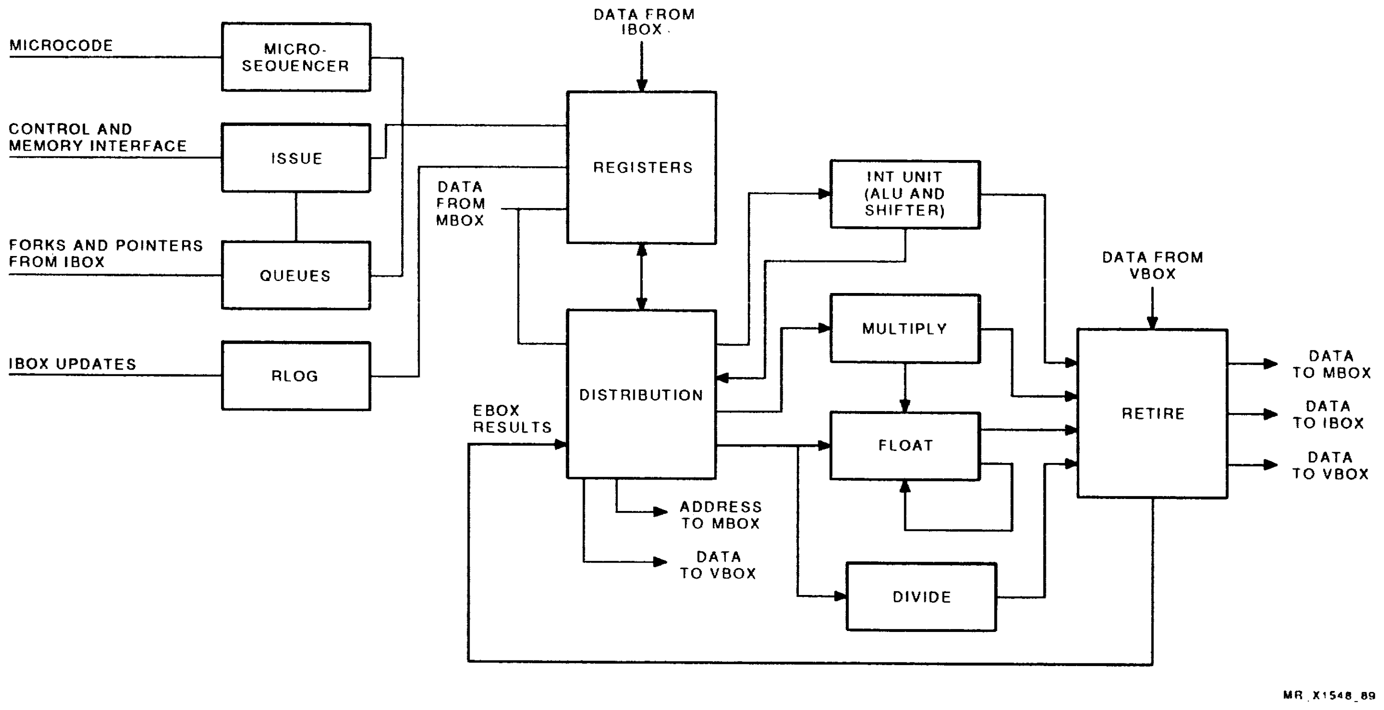


Figure 3–10 Basic EBox Functional Block Diagram

3.4.1.2 Issue Functional Unit

The issue unit (ISSUE) performs overall control of the EBox pipeline. That is, it controls the issuing of an instruction to one of the functional execution units (FEUs: integer unit, multiply unit, and so on), and the completion of the instruction through the retire stage. Control is maintained in conjunction with the microsequencer and microcode, which direct the specific execution functions.

Instructions are issued when the source operands are valid and available from the STREG, and the required FEU is available. Register operands are checked for pending writes in the destination queue before the instruction is issued. At any one time, several instructions may be executing in the EBox in various stages of completion.

The issue control logic consists of two tightly coupled units:

- Issue, which controls the starting of an instruction execution
- Retire, which controls the completion of an instruction execution

Both issue and retire can be initiated on every cycle.

The control logic also has a memory interface to the MBox. The interface implements two types of read and write operations to the MBox.

3.4.1.2.1 Issue Logic

Instruction issue is determined while the microword and the source data are being read. If for some reason the instruction cannot be issued, the microword is saved and used in the next cycle, and the source data is read again. A number of factors can inhibit an issue on every cycle, for example, source pointers not valid, source data not valid, memory write path busy.

When an issue is initiated, the result destination data is written into the result queue, together with the condition codes and branch checking. Where an instruction requires multiple computation cycles, other operations can be issued.

3.4.1.2.2 Bypass Control

Bypass control is also performed by the issue logic. This is done by comparing where data will be next cycle, with where it will be required in the next cycle. If a data path directly connects the two functional units, bypass enabling functions steer a copy of the data where it is needed. The normal flow of the data through the pipeline continues in parallel. Therefore, the required data can be obtained earlier through a bypass than by waiting for it to be written into a register file and then read out.

Four bypasses are implemented in the EBox: issue, register, memory, and data.

- **Issue bypass** — This bypass satisfies the requirements for an issue cycle. For example, a valid fork address is available, the target FEU is available, and so on.
- **Register bypass** — The temporary registers (temps) and GPRs have bypasses. The temporary registers have no bypass restrictions. However, several restrictions apply to the GPRs.
- **Memory bypass** — The EBox can bypass data from memory directly to an FEU. MBox data is passed into the DIST unit and immediately distributed to the specified FEU.
- **Data bypass** — This bypass allows the transfer of source and result data between EBox functional units (including FEUs). However, the divide unit has no bypass and is the exception.

3.4.1.3 Retire Functional Unit

The computational results from the FEUs are assembled in the retire functional unit (RETIRE). The retire unit then distributes the results back to the STREG, or in some cases, to the IBox. The retire unit can also direct data between the FEUs.

The retire unit contains the result queue. When the instruction is issued, its result write destination is written into the result queue. The clocking of condition codes and branch checking are also written into the queue.

Retirement specifies that the FEU is not required to hold its results. The retire unit informs the FEU that the results have been distributed to the appropriate destination (IBox, MBox, or register file). To do a retirement, the microcode specifies the function result destination to the control logic.

Instructions are retired in the same order they were initially received from the IBox. For example, a divide is issued and is followed by a multiply operation issue. The multiply may finish before the divide; the multiply is queued but not retired. When the divide completes, it is retired, followed by the multiply retirement.

Some macroinstructions require several retire cycles, while others require only one cycle. The retire unit always performs at least one retire cycle to flag macroinstruction completion.

3.4.1.3.1 Result Queue Logic

The result queue stores result destination pointers. The result queue provides two major functions. It allows issuing FEUs before a previous FEU operation is completed, and it maintains the function retire order. In addition to the destination pointer and memory destination, each entry in the queue contains: the retire unit, context, condition codes, the number of destinations, and last retire.

A result destination is required to retire an operation. When the FEU has indicated that it has completed its operation, the destination is popped from the top of the result queue, the results written, and the operation retired.

At times, it may be necessary to stall retiring results when writing to memory. Stalls may be encountered because a memory write might cause a page fault, or the memory pipe is full due to a number of previous memory write operations.

3.4.1.4 Functional Execution Units

All instruction processing is handled by the four FEUs. The FEUs are designated as the: integer unit, multiply unit, floating-point unit, and divider unit.

The FEUs are interconnected through the bypass data paths to allow data to be moved from one unit to another as quickly as possible. DIST distributes the source data to the specified FEU.

The FEUs are independent units that receive two 32-bit operands per cycle from the register file, through DIST. Only one FEU can be initiated per cycle, and only one 32-bit result can be handled in a cycle. The arbitrating logic of the retire unit selects which FEU result to write.

3.4.1.4.1 Integer Unit

The integer unit (INT) performs general integer arithmetic, logic functions, and other specific functions to increase simple instruction execution times. The integer unit contains a 32-bit ALU and a 64-bit barrel shifter. In addition, it contains an address generation unit capable of producing a memory address each cycle.

The integer unit executes simple instructions (for example: MOVL, ADDL) at a rate of one per cycle with little microcode control. Complex instructions (CALLS, MOVC, and so on) are executed by repeated passes through the data path. For those instructions, the microcode controls access to EBox resources.

3.4.1.4.2 Multiply Unit

The multiply unit (MUL) performs integer and floating-point multiplication and performs the following tasks:

Unpacking
Exponent handling
Sign handling
Multiplication
Condition code generation
Subproduct accumulation
Floating-point rounding
Packing
Control
Error handling

The multiply unit is a variable length pipe unit. If the context is integer, the multiply is performed in one cycle. If the context is floating point, the multiplication requires two or three cycles to perform, depending on format. A three-cycle multiply will pass through three MUL pipe stages: unpack, save and accumulate, and round and pack.

3.4.1.4.3 Floating-Point Unit

The floating-point unit (FLOAT) performs floating-point addition, subtraction, and certain data format conversions. It executes floating-point operations for the ADD, SUB, CMP, CVT, and MOV instructions in F, G, and D floating-point formats.

FLOAT is pipelined and can accept instructions on every cycle as the issue logic issues and retires them. Although it receives 32-bit source operands, it operates on an internal 64-bit data path.

The unit is implemented in hardware, following a basic flow of unpacking the sources, aligning, adding and/or subtracting the fractions, normalizing and rounding the fraction, packing, and transferring the final result. It produces a complete 64-bit result of which 32 bits can be immediately transferred, and the other 32 bits are saved to be sent in the following cycle.

3.4.1.4.4 Divide Unit

The divide unit (DIV) performs integer and floating-point division. Because the divide unit is not pipelined, only one divide instruction can be in progress at any one time. While instructions may be issued to other FEUs while the divide unit is busy, they cannot be retired until the division has completed. Divisions can be completed in 12 cycles, including D and G formats.

Context information is supplied from the microsequencer unit. The issue unit provides issue and retire control. The functional tasks performed include the following:

Unpacking
Exponent handling
Sign handling
Iterative division
Condition code generation
Quotient accumulation
Floating-point rounding
Packing
Control
Error handling

3.4.1.5 Condition Codes

The condition code logic performs two functions: set the PSL condition codes, and perform macrobranch checks to inform the IBox if the branch prediction was correct. The PSL condition codes can be set by:

- Being clocked by the microcode (usually at the end of a macroinstruction) based on the results of an instruction
- Being written directly when returning from an exception or interrupt handling routine
- Executing a BSPSW or BSPSL instruction

The condition codes are the four low-order bits of the PSL. With the appropriate logic functions enabled, the low-order ALU result is written to the CC bits.

3.4.2 EBox Microcode Overview

The EBox microcode provides the control required to execute the instruction set, to process interrupts and exceptions, and to control the entire system. EBox microcode is a synchronization point for the pipeline as it handles errors, interrupts, exceptions, traps, and so on. The functions executed by the EBox are controlled by generating specific sequences of microwords. Each microword within the sequence is encoded to perform the data routing and manipulation required.

The EBox microword is 146 bits wide and is contained in a 4K word control store. However, only 145 bits are used. The wide microword allows simultaneous control of multiple operations to achieve parallel execution.

The microword is comprised of 57 fields, some of which are overlapped. The microword fields are grouped according to function, and are laid out in a logical order rather than a physical order.

The control store is implemented in 1K × 4 and 4K × 4 STRAMs. Because the 1K STRAMs have a faster access time (7 ns), they are used for certain fields of the microword (microbranching, next address). That is, due to the 1K functionality, a new microword can be accessed each cycle.

3.4.3 EBox Interfaces

The EBox interfaces to the IBox, MBox, VBox, JBox, and to the SPU. This section provides a brief description of the functions of each interface.

3.4.3.1 IBox Interface

The IBox and EBox communicate across two interfaces. The IBox sends specifier data and pointers to the EBox, and the EBox sends control signals and GPR update data to the IBox.

3.4.3.1.1 IBox-to-EBox Interface

The IBox-to-EBox interface provides the means for the IBox to deliver decoded specifier data and specifier information to the EBox. Across this interface, the EBox receives opcodes, source pointers, destination pointers, and specifier data. The pointers are accompanied with valid signals and are loaded into the EBox source queue and destination queue.

Specifier data is delivered to the EBox across a dedicated 32-bit data path and loaded into the source list. Short literal and immediate mode specifiers are passed to the EBox across this data path. The EBox also receives GPR data from the IBox. This data is sent to the EBox when the IBox performs autoincrements and autodecrements to GPRs.

3.4.3.1.2 EBox-to-IBox Interface

The EBox-to-IBox interface transfers control signals and data to the IBox. The control signals include EBox flush signals, signals that indicate if a branch has been correctly or incorrectly predicted, and signals that direct the IBox to unwind autoincremented or autodecremented GPRs.

The EBox passes GPR data to the IBox copy of the GPRs and can pass new target addresses to the PC unit of the IBox. A new target address initiates an instruction fetch for new I-stream.

3.4.3.2 MBox Interface

The EBox and MBox communicate across two interfaces. The MBox sends specifier data or EBox-requested data to the EBox and the MBox receives result data from retired instructions and register data, during register write operations, from the EBox.

3.4.3.2.1 MBox-to-EBox Interface

The MBox-to-EBox interface contains a 64-bit data path to the EBox. This data path is used to place operand data requested by the IBox into the source list. It is also used to place EBox-requested data into the memory temporary registers in the register file. The data path can be used to send either 32 bits or 64 bits. Control signals are provided to indicate when data is valid, the size of the data, and where the data should be stored in the EBox.

The MBox also uses this interface to notify the EBox of any internal errors.

3.4.3.2.2 EBox-to-MBox Interface

The EBox-to-MBox interface includes a 32-bit address path, 32-bit data path, and numerous control and status signals.

The 32-bit address path provides an address to the MBox for explicit reads and writes initiated by the EBox. The same 32-bit path is used to provide data to the MBox during MBox register write operations.

The 32-bit data path provides data for IBox requested op writes and and for EBox explicit writes. Control signals are provided to indicate the context and origination of the write request (op write or EBox explicit write).

Two sets of control signals differentiate between EBox-initiated and IBox-initiated (op write) writes to the MBox. Because the destination of the op write has already been stored in the MBox write queue, the op write control signals only consist of an op write flag and indicate the context of the data being written. The EBox read and write control signals contain request, control, context, and tag signals. When a request is asserted, the tag indicates the address to be written or read, and the control signal indicates the function of the request (read, write, and so on).

3.4.3.3 VBox Interface

EBox transmission of data to the VBox is initiated by detection of a vector instruction by the EBox micromachine. The EBox sends commands and data to the VBox and, when the vector operation is completed by the VBox, the data is passed through the EBox to memory.

3.4.3.3.1 EBox-to-VBox Interface

The EBox-to-VBox interface contains a 64-bit data path and associated control signals. The data path delivers data to the VBox register file or supplies a command to the VBox control logic. The control signals of this interface indicate if a transaction is data or command.

3.4.3.3.2 VBox-to-EBox Interface

The VBox sends data to the EBox for vector register read operations and passes the data through the EBox to the MBox for vector store operations. The VBox also passes control information to the EBox to indicate the data is valid and to indicate the destination of the data.

3.4.3.4 JBox Interface

The JBox contains an interface to the EBox that is dedicated to passing interrupt codes. This interface consists of a pair of differential signals that are used to pass a serial I/O interrupt code.

3.4.3.5 SPU Interface

The EBox interrupts the SPU across two dedicated serial lines. The two interrupts sent by the EBox are EBox halt and keep alive. EBox halt is transmitted to the SPU when an EBox halt has occurred. Keep alive is transmitted to the SPU to indicate that the EBox is still executing instructions. The keep-alive signal is part of the keep-alive mechanism that the SPU uses to detect a hung CPU.

3.4.4 EBox Pipeline Stages

The microcoded EBox has a three-stage pipeline:

- The first is the issue stage and is a cycle used to access source data and read the control store.
- The second is the execute stage and consists of one or more cycles used by the functional execution units (FEUs) to calculate results and condition codes.
- The third is the retire stage and is a cycle used to write the results to the specified destination, and to update the condition codes.

All pipeline stages can be active simultaneously. In all cases, instructions are issued and retired in the order received.

3.4.5 EBox Physical Organization

The EBox logic is physically located in six MCUs (Figure 3–11). This section introduces each MCU and its related MCAs.

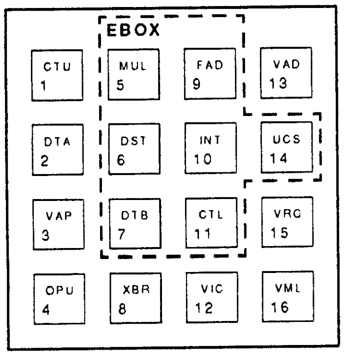


Figure 3–11 EBox MCUs

3.4.5.1 MUL MCU

The multiply (MUL) MCU contains seven MCAs that provide logic for the MUL and RET functional units. The MUL MCU is positioned at module location MCU 5 in the CPU planar module. The following list introduces the MCA functions:

- **MULX MCAs** — The MUL0, MUL1, and MUL2 MCAs comprise the logic that performs multiplication.
- **MACC MCA** — This MCA is the accumulator logic for the MUL unit. Partial products for multiplications are accumulated in this MCA and are added to the other partial products of the multiplication.
- **MPCK MCA** — This MCA contains the control logic for the MUL unit and the logic for floating-point rounding and packing.
- **MACC RETX** — The RET0 and RET1 MCAs contain the bit-slices of the RET unit.

3.4.5.2 FAD MCU

The floating add and divide (FAD) MCU contains seven MCAs and provides floating-point addition, data type conversion and divide arithmetic logic. The following list introduces the MCAs of the FAD MCU:

- **FRSX MCAs** — The FRSA and FRSB MCAs contain the logic that controls the floating-point unit (FLOAT) and sticky bit generation.
- **FADR MCA** — This MCA contains the logic for fraction addition and subtraction, priority encoding, normalization, and parity logic.
- **FPCK MCA** — This MCA contains the logic that processes exponents, performs rounding, generates condition codes, and performs packing.
- **DUNP MCA** — This MCA performs the unpack functions for the DIV unit.
- **DIV0 MCA** — This MCA performs the divide functions.
- **MACC DPCK** — This MCA contains quotient accumulators, performs rounding and packing, and control and error detection logic for the DIV unit.
- **PCHBX** — One 4K × 4 STRAM is used to implement the PC history buffer.

3.4.5.3 CTL MCU

The control (CTL) MCU contains the MCAs that support the issue and queue logic units. This MCU also contains STRAMs that provide storage for a portion of the control store.

- **ISSA** — This MCA contains logic for handling bad branch predictions, faults, temporary memory register bits (valid and fault), and the result queue and source list bits (valid and fault).
- **ISSB** — This MCA contains the VBox interface logic, bad branch prediction handling, destination pointer, result queue, retire queue, and scoreboard logic.
- **ISSC** — This MCA contains logic for the HIR class decoder bad branch prediction and fault handling, result queue, and vulnerable mode status bit.
- **ISSD** — This MCA contains the MBox interface logic, result queue, unit busy bits, memory bypass bad branch prediction handling logic, and data bypassing.
- **ISSE** — This MCA contains logic that handles bad branch prediction, errors, faults, and the result queue.
- **FRAMX** — FRAM1 through FRAM4 (1K × 4 STRAMs) are used for the queue logic.
- **QPTR** — This MCA contains control logic for the source queue, fork queue, destination queue, and the control store.
- **QPCS** — This MCA contains control, parity checking and generation, and match logic for the PC queue.
- **CSSX** — CSS31 through CSS37 and CSS39 (4K × 4 STRAMs) provide part of the control store logic for the microsequencer.

3.4.5.4 DIST MCU

The distribution (DIST) MCU contains the four MCAs that comprise the distribution logic, an MCA that controls the source pointer logic and source list, two STREGs that provide the EBox GPRs, and eight 1K × 4 STRAMs.

- **DSTX** — DST0 through DST3 provide the logic that comprises the distribution logic.
- **STGX** — STG0 and STG1 are STREGs that provide the EBox GPR logic.
- **EREGX** — EREG1 through EREG9 (1K × 4 STRAMs) are used for temporary microcode storage.
- **SRCS** — The SRCS MCA controls accesses to the source operands stored in the source list or used as GPR references.

3.4.5.5 INT MCU

The integer (INT) MCU contains 6 MCAs and 14 STRAMs. The MCAs provide the logic that comprises the INT unit and the microsequencer logic. The STRAMs provide a portion of the EBox control store.

- **IALU** — This MCA contains the logic that performs the integer add and subtract functions.
- **ISHF** — This MCA contains the integer logic shifter.
- **IALU** — This MCU contains the INT unit shifter.
- **USQX** — USQA, USQB, and USQC comprise the microsequencer logic.
- **CSS1X** — CSS11, CSS13, CSS14, CSS16, CSS17, and CSS19 are 4K × 4 STRAMs that store a portion of the control store.
- **CSF1X** — CSF11 through CSF18 are 1K × 4 STRAMs that store the fork RAM portion of the control store.
- **RLOG** — This MCA controls the GPR log.

3.4.5.6 UCS MCU

The control store (UCS) MCU is located at module location 14 and contains logic that supports both the EBox and VBox functions. The EBox logic in this MCU consists of twenty-seven 4K × 4 STRAMs that store a portion of the EBox control store.

3.4.6 EBox Basic Instruction Flow

This section describes the ADDL3 instruction execution flow, and is based on the following assumptions:

- GPR sources and destination
- All instruction data resident in the queues
- Execution activity divided into system clock cycles

The following list describes the instruction flow:

- **Clock 1, fork cycle** — The fork address (the first microword address), which is contained in the fork queue, is used by the microsequencer to start accessing the control store.
- **Clock 2, issue cycle** — During this cycle, the microword is read from the control store and distributed throughout the EBox. The source data is read out of the GPRs and distributed to the integer unit, while the issue unit verifies that the instruction data is valid and that the integer unit is available.
- **Clock 3, execute cycle** — The integer unit receives the appropriate microword fields and source data, and computes the result. Because this is an integer operation, the result write destination is determined during this cycle.

  NOTE
  For FEUs requiring multiple computation cycles, the result write destination is determined in the last compute cycle.
- **Clock 4, retire cycle** — The result is transferred through the retire unit to the destination GPR.
- **Clock 5, write cycle** — The result is written into the destination GPR.

All instructions follow this basic flow. At times, an instruction may stall at a certain point because of a missing or unavailable resource (for example: instruction data, an FEU, a pointer). In addition, because the multiply, FLOAT, and divide units require multiple computation cycles, there is the opportunity to issue other operations.

Regardless of the conditions encountered, instructions always flow through the pipe in the same order as received from the IBox. In addition, instructions are retired in the same order as received from the IBox.

3.5 VBox Introduction

The VAX 9000 VBox is an optional vector processor that connects to the VAX 9000 scalar CPU. The VBox performs arithmetic and logical operations on vector operands, as specified in the VAX vector instruction set.

3.5.1 VBox Hardware Implementation

The VBox is implemented with an MCA III and a custom bit-sliced vector register (VRG) file. The VBox contains the following five functional units (Figure 3–12):

VRG file
Adder
Multiplier
Mask logic
Control logic

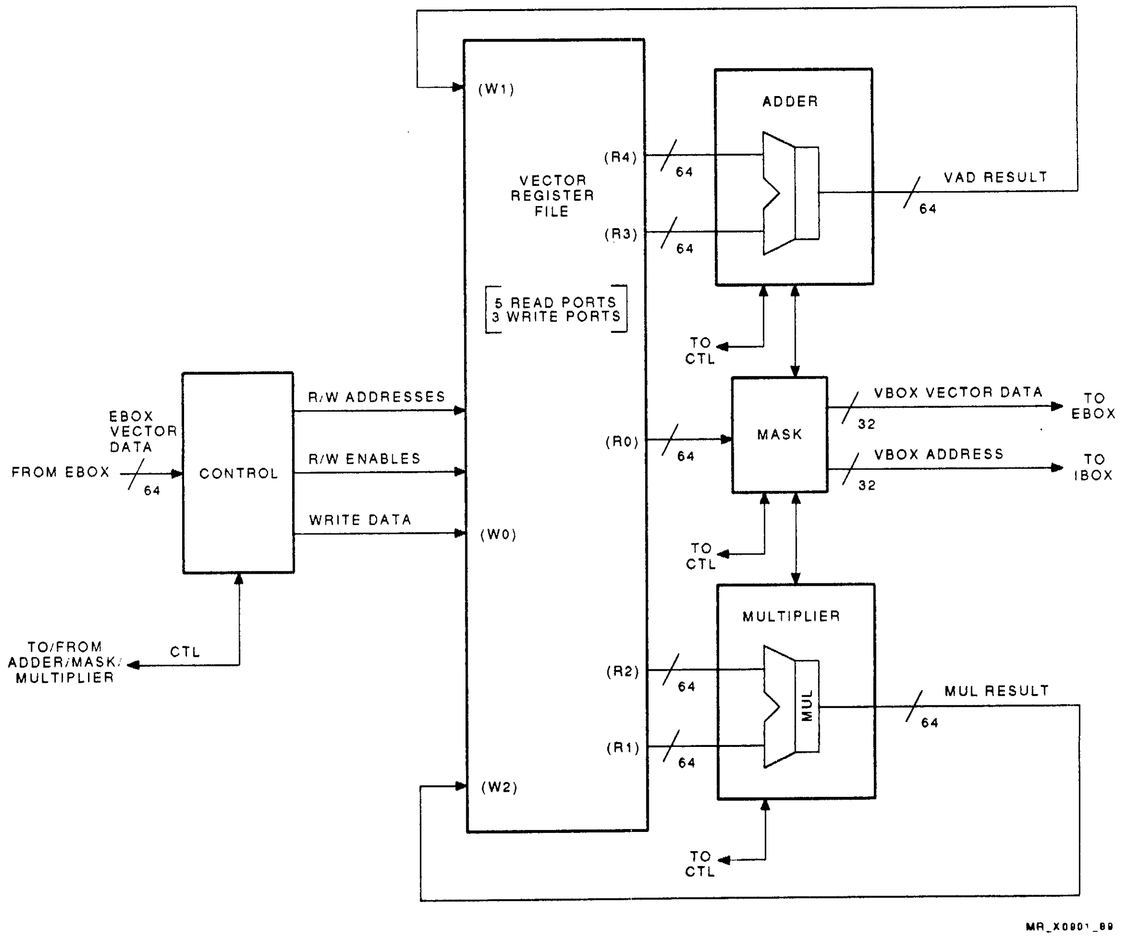


Figure 3–12 VBox Block Diagram

3.5.1.1 Vector Register File

The multiport VRG file stores vector operands that are operated on during vector operations. There are 16 VRGs that are each capable of storing one vector of data (64 × 64 bits). The VRG file also contains five scalar registers for storing scalar operands.

3.5.1.2 Vector Adder

The vector adder performs the VBox addition and subtraction operations. The adder also performs compare, shift, convert, logical, and merge operations. The adder performs pipelined operations and produces a result at every cycle. The pipelined design allows another operation to begin before the previous operation completes.

3.5.1.3 Vector Multiplier

The vector multiplier performs multiply and divide operations. Vector multiply operations can be pipelined, and they produce a result at every cycle. Vector divide operations cannot be pipelined.

3.5.1.4 Vector Mask Logic

The vector mask logic contains the mask register and the VBox interface to the scalar CPU.

The vector mask register (VMR) contains 64 mask bits that enable and disable the use of individual vector elements. For example, if only the odd numbered bits of an element are enabled in the VMR, the instruction processes only the odd numbered elements during execution. The enabling of masked operations and the enabling and disabling of bits in the VMR are controlled by a control word operand.

The VBox interface to the scalar CPU includes data lines to the EBox and address and control lines to the IBox. The addresses are passed through the IBox OPU port to the MBox. Data is passed through the EBox to and from the VBox.

3.5.2 VBox Physical Organization

The VBox logic is located in four MCUs (Figure 3–13). One of the MCUs contains both VBox and EBox MCAs. This section introduces each MCU and its related MCAs.

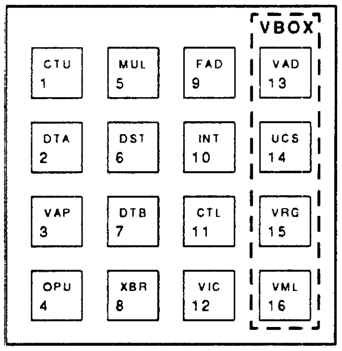


Figure 3–13 Planar Module Layout

3.5.2.1 VRG MCU

The vector register (VRG) MCU contains the eight MCAs that make up the VRG file. Each MCA contains a 9-bit slice of the VRG file (eight data bits and one parity bit). The following list introduces the MCAs and their functions in the VRG MCU:

- **VRG0** — Register file data bits [07:00]
- **VRG1** — Register file data bits [15:08]
- **VRG2** — Register file data bits [23:16]
- **VRG3** — Register file data bits [31:24]
- **VRG4** — Register file data bits [39:32]
- **VRG5** — Register file data bits [47:40]
- **VRG6** — Register file data bits [55:48]
- **VRG7** — Register file data bits [63:56]

3.5.2.2 VAD MCU

The vector adder (VAD) MCU contains six MCAs that provide the adder and mask logic of the VBox. The following list introduces the functions of the MCAs in the VAD MCU:

- **VFSA** — This MCA contains the unpacking, exponent processing, and fraction alignment logic for source 1.
- **VFSB** — This MCA contains the unpacking, exponent processing, and fraction alignment logic for source 2.
- **VFAD** — This MCA contains the addition, subtraction, logical operation, and a portion of the normalization logic.
- **VFPK** — This MCA contains normalization, rounding, packing, and exception logic.
- **VMKA** — This MCA contains the scalar CPU and memory interface logic.
- **VMKB** — This MCA contains the mask processing and exception logic.

3.5.2.3 VML MCU

The vector multiply and divide (VML) MCU contains the VBox multiply and divide logic. The following list introduces the MCAs of the VML MCU:

- **MULX** — MUL3, MUL4, MUL5, and MUL6 contain the unpacking, multiplication, and the exception generation logic of the vector multiplier logic.
- **VMLX** — The VMLA and VMLB MCAs contain the VML subproduct and quotient accumulator, rounding, and packing logic. Exception and opcode control logic is also contained in VMLB.
- **DIVU** — This MCA contains divide control, unpacking, and exponent and exception generation logic.
- **DIV1** — This MCA contains the vector divide logic.

3.5.2.4 UCS MCU

The control store (UCS) MCU contains the EBox and VBox logic. The EBox components of this MCU contain a portion of the control store, while the VBox portion provides control for VBox operations. The following list introduces the VBox MCAs in this MCU:

- **VCTA** — This MCA contains adder control and conflict logic.
- **VCTB** — This MCA controls multiplier control and conflict logic.
- **VCTC** — This MCA contains memory instruction control and conflict logic.

3.5.3 Vector Processing Overview

The VAX 9000 VBox is a coprocessor that attaches to the scalar CPU and performs simultaneous calculations on vectors. A vector is a quantity represented by an ordered set of numbers. The numbers in vector operands are called elements. There can be up to 64 vector elements in the VAX 9000 with each element containing up to 64 bits.

A vector operation consists of a number of identical operations on individual vector elements.

3.5.3.1 Overlapping and Chaining

Overlapping and chaining are software techniques that increase the efficiency of vector instructions. Overlapping reduces cycle time by better using the multiple functional units of the VBox. Chaining is a more efficient programming method, because it reduces the number of instructions executed by linking instructions together.

Overlapping combines two or more instructions so that they execute in parallel. The instructions that overlap are executed by the different functional units of the VBox. In the following example, the add and multiply instructions are overlapped and executed in parallel.

$$C_1 = B_1 + A_1$$

$$F_1 = E_1 * D_1$$

Chaining is also possible with multiple functional units. The following example shows an example of chaining an instruction. The first two instructions are linked together to create the more efficient third instruction. At the completion of the first instruction the result is stored in the register file and then accessed by the adder to execute the second instruction. By chaining the instructions, the result of the divide unit is passed directly to the adder and the add portion of the instruction is executed. Bypassing the storage of the multiply reduces the cycle time for the execution of this instruction.

$$F_1 = E_1 * D_1$$

$$H_1 = F_1 + C_1$$

By chaining the two instructions, the same result is produced from a single instruction.

$$H_1 = E_1 * D_1 + C_1$$

3.5.3.2 Vector Alignment

The vector instruction set requires that vector operands be naturally aligned in memory. Longwords must be aligned on longword boundaries and quadwords must be aligned on quadword boundaries.

3.5.3.3 Vector Stride

A vector stride is the number of memory locations (bytes) between the first location of one element and the first location of the next (consecutive) element. (See Figure 3–14.) The elements in the vector can be contiguous or noncontiguous in memory. Contiguous longwords have a vector stride equal to four, and contiguous quadwords have a vector stride equal to eight.

Vectors having contiguous or noncontiguous elements can be processed in the VAX 9000. The stride for noncontiguous elements can be a constant value or can vary from one element to the next.

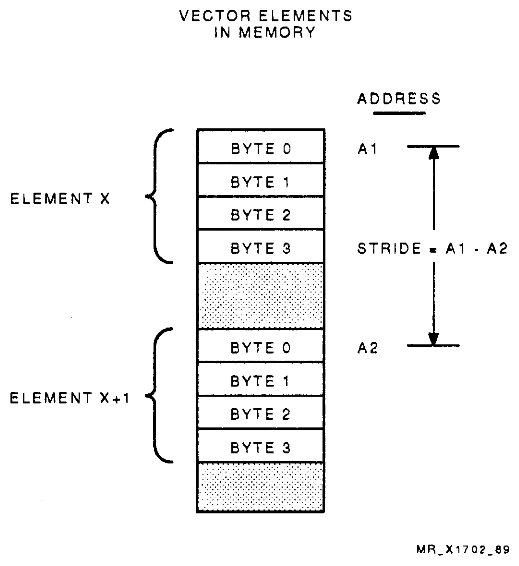


Figure 3–14 Vector Stride

3.5.4 VBox Operational Summary

Vector instructions can be grouped into six general categories:

Memory instructions
Integer instructions
Floating-point instructions
Logical and shift logical instructions
Edit instructions
Control instructions

3.5.4.1 Memory Instructions

Memory instructions consist of load, store, scatter, and gather instructions. Load and gather instructions load a vector operand from memory into a vector register. Store and scatter instructions store an operand from a vector register into memory.

The operands moved by the store and load instructions have a constant stride. Operands moved by the scatter and gather instructions can have a stride that varies from element to element.

There are two sets of memory instructions. One set transfers vectors having longword elements and the other transfers quadword elements.

3.5.4.2 Integer Instructions

Integer instructions perform fixed-point arithmetic operations and compare vector operands having longword elements. The arithmetic instructions are add, subtract, and multiply operands. One set of integer instructions performs vector-vector operations and another set performs vector-scalar operations.

3.5.4.3 Floating-Point Instructions

The floating-point instructions perform floating-point arithmetic on vector operands having longword elements (F format) or quadword elements (D and G formats). Floating-point vector instructions perform add, subtract, multiply, divide, and compare operations on F, D, and G formats in vector-vector and vector-scalar modes. There is a convert instruction that converts one format to another. The convert is limited to vector-vector operations.

3.5.4.4 Logical and Shift Instructions

The logical instructions perform XOR, bit set (OR), and bit clear (AND) on vector operands having longword elements. The shift logical instructions shift elements left or right in vector operands having longword elements. Logical and shift logical instructions exist for vector-vector and vector-scalar operations.

3.5.4.5 Edit Instructions

The edit instructions are the IOTA and merge instructions. The IOTA instruction constructs a vector of address offsets for the gather and scatter instructions. The merge instruction merges two source operands into a single vector operand. The vector mask register controls this instruction by selecting the source for the elements in the merged vector. The source operands can be two vector operands or a vector and a scalar operand.

3.5.4.6 Control Instructions

The control instructions, which are move to and from vector processor (MTVP and MFVP) instructions, are used to read and write the vector control registers in the VBox. Special implementations of the MFVP (SYNC and MSYNC instructions) allow user software to synchronize the CPU and VBox operations. SYNC is used to synchronize instruction execution and MSYNC is used to synchronize memory access. Synchronizing is achieved because the MFVP instruction does not complete until prior vector instructions (for SYNC) or prior memory accesses (for MSYNC) have completed.

4 System Control Subsystem

This chapter introduces the system control unit (SCU) and the main memory subsystem. It describes the major features and functions of each subsystem.

4.1 Overview

In the VAX 9000 family of systems, the SCU connects the CPU, SPU, I/O subsystem, and the memory subsystem. The SCU controls reads and writes to main memory, I/O reads and writes, and also provides cache consistency.

Cache consistency ensures that if copies of a cache block exist in different CPUs, the copies contain identical data. When the data in the cache blocks is not identical, a requester (CPU or I/O) receives the most recent copy of the data.

The SCU is partitioned into three functional blocks:

Junction box (JBox)
I/O control unit (ICU)
Array control unit (ACU)

Figure 4–1 shows a block diagram of the SCU. The following sections briefly describe the functions of the JBox, ICU, and the ACU, and also describe the data and address interconnects of the SCU.

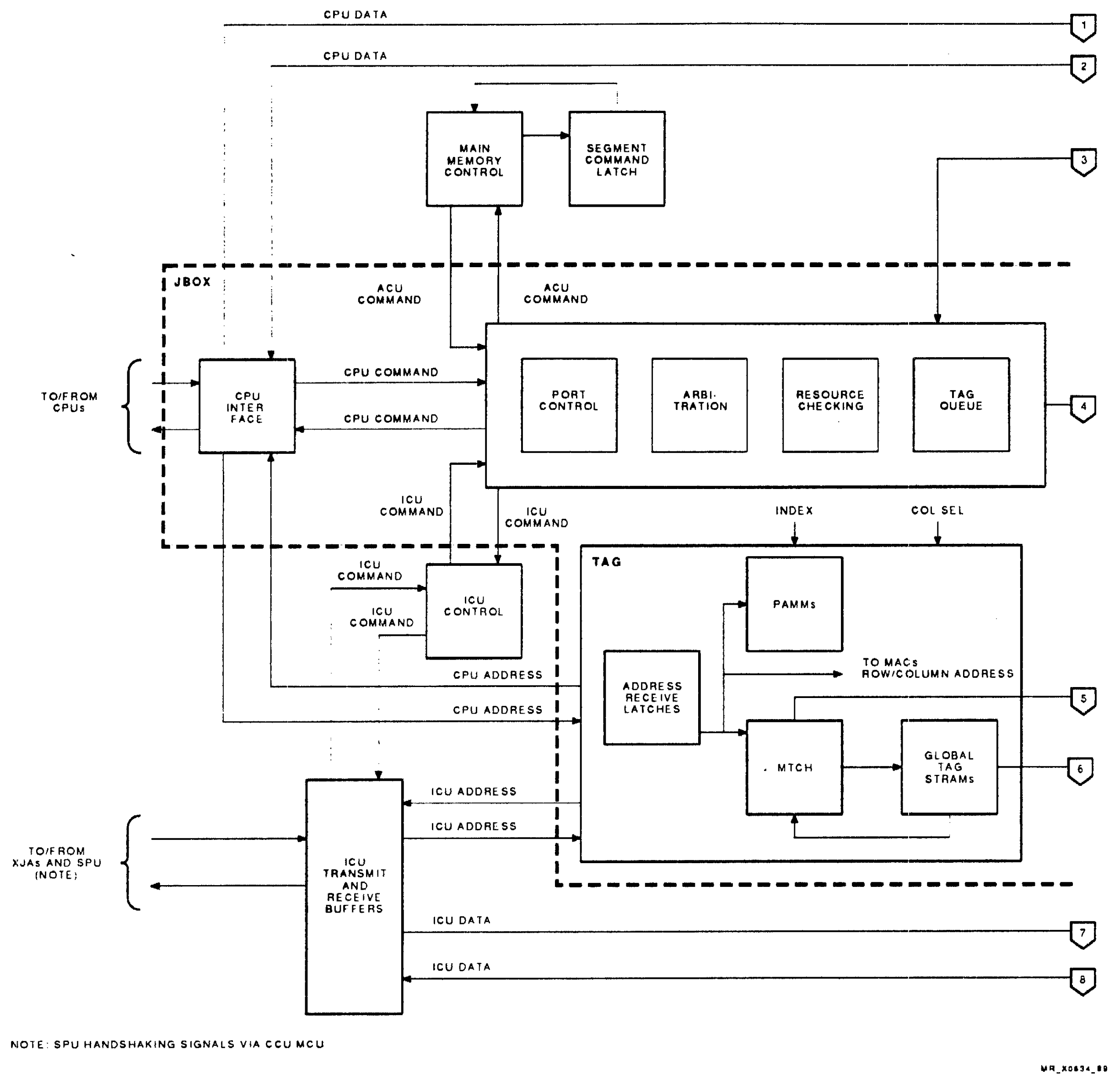


Figure 4–1 (Cont.) SCU Block Diagram

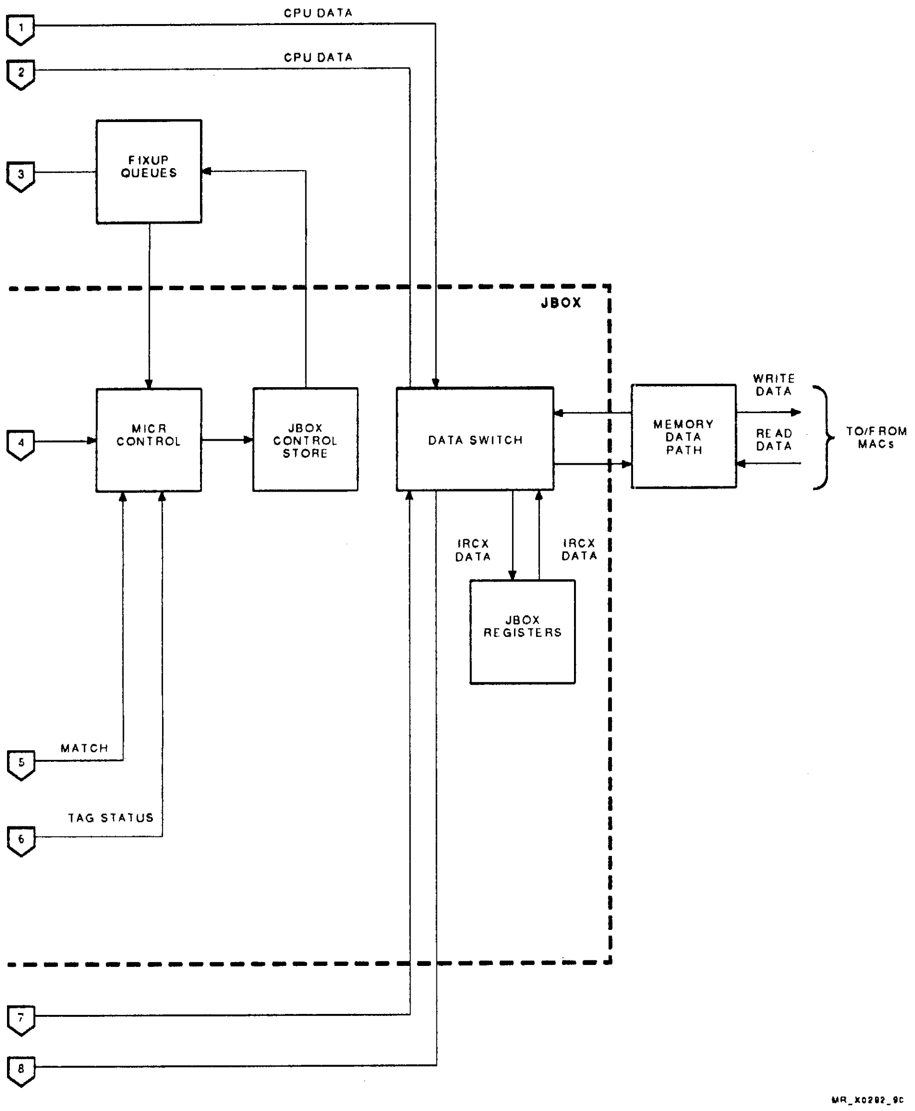


Figure 4–1 SCU Block Diagram

4.1.1 JBox

The JBox logic contains the SCU control, the data switch, and the address receive latches. JBox logic interfaces to the MBox, memory, ICU, and SPU. The JBox arbitrates requests from the memory subsystem, I/O subsystem, and up to four CPUs. It ensures cache consistency for the CPU write back cache, and checks for cache blocks that require invalidating as a result of I/O writes to memory. The cache consistency unit contains the global tag STRAMs.

The JBox contains the micromachine that consists of the control store STRAMs and the microcontrol logic. Most of the logic within SCU is controlled either directly or indirectly by bits within the microword. The JBox also contains three fixup queues that handle cache inconsistencies, nonexistent memory errors, and lock busy and lock deny requests.

The JBox connects nine ports, using address receive latches and a data switch, and allows simultaneous transactions. The JBox latches a request, sends it to arbitration, and determines which SCU resources are needed to complete the request. The JBox compares available resources with required resources, placing the request either in a tag queue for execution, or on a reserve list until resources become available. The SCU resources include source and destination data paths, command buffers, and address receive latches.

When the JBox receives a request, it transfers control to a port controller that monitors the status of the request from load, arbitration, and retire. The JBox also latches the physical address in address latches and holds onto this address until the request is retired. The address latches are available to the microcode, memory, and the JBox itself.

4.1.2 ACU

The ACU provides the command, data and address control, and status interface to the JBox for the main memory subsystem. The ACU sends DRAM control signals (RAS, CAS, and WE) to the memory modules, and supports built-in self-tests. The ACU uses main memory control and DRAM control to operate the write buffers, the read buffers, and the read bus on the memory modules. It provides the SCU memory data path to receive and send DRAM data to the JBox data switch.

4.1.3 ICU

The ICU supports the SPU and the XJAs by providing the command, data and address control, and status interface to the JBox. The ICU contains the receive buffers and transmit buffers to receive and send SPU and XJA commands to and from the JBox. The ICU uses I/O control to load and unload the receive and transmit buffers.

The ICU provides the interface between the JBox and the interrupt arbiter and JBox registers. The JBox can read the contents of three registers and send write data to the registers by interfacing to the ICU.

When the ICU receives a request, it loads a receive buffer with the command, address, and the data. The ICU sends the command to the JBox, the address to the address receive latches, and the data to the JBox data switch. When the ICU sends a request, it loads a transmit buffer with the command, address, and the data from the JBox.

4.1.4 Data and Address Interconnects

The SCU contains data and address interconnects between the MBox, the ICU, and ACU. All of the data interconnects are quadword-wide data paths and allow simultaneous transactions.

The address interface between the JBox and the MBox is two bytes wide and requires two cycles to transfer a complete address. All addresses transferred on this interface are physical addresses [31:02].

The address interface between the JBox and the ACU is divided into two parts:

- Row and column addresses are sent to the memory array cards (MACs) in either of the main memory units (MMUs). The address field is up to 13 bits wide depending on the DRAM chip size in use.
- Memory control logic receives memory port select, segment select, and bank select fields from the JBox.

The memory system does not return addresses to the MBox. Instead, it sends an index field to the JBox. The JBox uses the index field to associate return data with one of 16 addresses in the address receive latches.

The address interface between the JBox and the ICU is two bytes wide and requires two cycles to transfer a complete address.

4.2 JBox Functional Overview

The JBox provides the majority of the functionality in the SCU. It controls the interfaces to the ports by managing the address and data crossbars that route the data between ports. It also contains the logic that ensures cache consistency. The following sections describe the major functional units of the JBox.

4.2.1 JBox Control

The JBox controls the command, address, and data paths for the CPU, ACU, ICU, and SPU ports. All functions of the SCU are controlled by the JBox control store. The control store consists of six hundred and seventy 60-bit microwords.

The control store space allocation defines three general functions: DMA and SPU requests, CPU requests, and fixup requests (Figure 4–2).

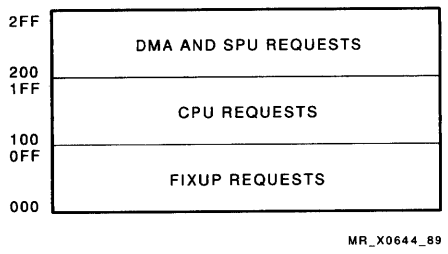


Figure 4–2 Control Store Space Allocation

4.2.2 CPU (MBox) Port Interface

To the CPU (Figure 4–3), SCU looks like a memory controller. CPUs implement write back caches, and SCU must ensure cache data consistency. SCU accomplishes this through the use of duplicate cache tag stores for each of the four CPUs.

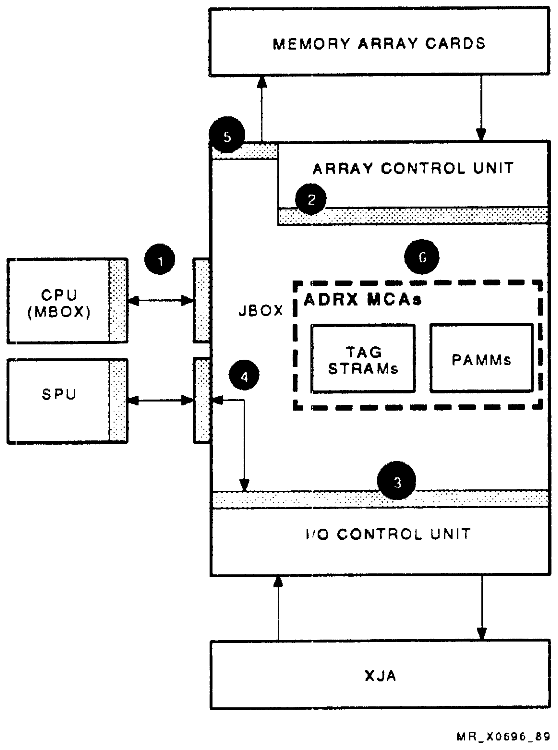


Figure 4–3 CPU Port

The JBox portion of SCU implements the cache consistency. Table 4–1 lists the command fields used on the CPU interface to the JBox, and Table 4–2 lists the command fields used on the JBox interface to the CPU.

Table 4–1 MBox-to-JBox Command Format Summary

Field	Description
Load command	Notifies the JBox that a command is going to be sent.
Buffer available	Indicates that a buffer is ready to receive a command from the JBox.
Data ready	Indicates that data is in the write back buffer, ready to be transferred to SCU.
Cache set	Indicates which cache set, 0 or 1, is involved.

Table 4–2 JBox-to-MBox Command Format

Field	Description
Fatal error	Indicates that JBox has detected a fatal error and has asserted the attention line to SPU.
Buffer available 0	Indicates the number of requests the JBox has retired. Works with buffer available 1 field. The output of the retire logic becomes buffer available.
Send data	Unloads the data in the write back buffer.
Load command	Notifies the MBox that a command has been sent.
Buffer available 1	Indicates the number of requests that the JBox has retired. Works with the buffer available 0 field.
Cache set	Indicates which cache set, 0 or 1, is involved.

The JBox latches the load command and sends it to the port state controller, and latches the command field and sends the command to the command arbitration logic.

All commands received by the JBox require a resource check. The availability of resources determines the microaddress sent to the microcode to initiate the routine that will handle the transaction.

The JBox responds to the MBox when a resource check is complete and the required resources are available. The response contains a send data field and identification of the buffers that will be loaded with the data for the MBox. The MBox also receives identification of the bank, segment, index, and cache set that receives the data.

When an error is detected in the JBox, an error bit is asserted and the CPU is notified.

4.2.3 ACU Port Interface

Each ACU is part of a separate memory subsystem and has its own JBox interface. A JBox-to-ACU interface permits the JBox to accept memory requests (CPU or I/O) and send the requests to the memory segment controllers. Figure 4–4 shows the JBox-to-ACU interface.

The ACU provides all communication between the JBox and main memory. The ACU performs the following functions:

- Accepts memory commands.
- Processes the commands to determine the availability of memory segments addressed by the commands.
- Sends commands to DRAM control.
- Determines the direction of data movement.

The ACU communicates with the JBox when any of the following conditions exist:

- A read request was made and the data is ready to send.
- An error was detected during the transfer of read or write data.
- A command buffer is available.

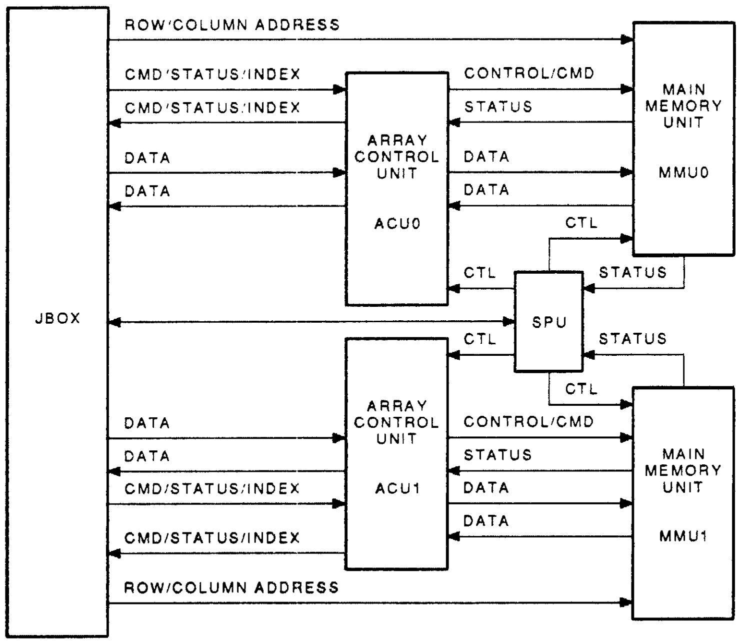


Figure 4–4 JBox-to-ACU Interface

4.2.4 ICU Port Interface

Each ICU is part of a separate I/O subsystem and has its own JBox interface. Figure 4–5 shows the JBox-to-ICU interface.

The ICU port is connected to the XJAs (through the JBox XMI data interface [JXDI]) and SPU, implementing the central system interrupt arbiter.

XJA and ICU provide an information path between the JBox and I/O devices. The SCU can have up to two ICUs, each capable of handling up to two XMI buses. To communicate with the XMI bus, a JXDI connects an XJA module to the ICUs. The JXDI has 16 data lines in each direction and cycles every 16 ns. It has a total bandwidth of 125 Mbytes/s in both directions. Address, command, and data are time multiplexed onto these wires.

The JBox latches the ID, length, and command from the ICU. The ID is saved until it is sent with the return data. The length field is decoded to determine the data switch control needed to transfer the data through the data switch. The command is sent to command arbitration, where the resources needed for the transaction are determined.

The JBox generates the command field for the response to the ICU. The command field is derived from command arbitration, arbitration index, and the JBox microcode field. The output load and send data command returns data to the ICU.

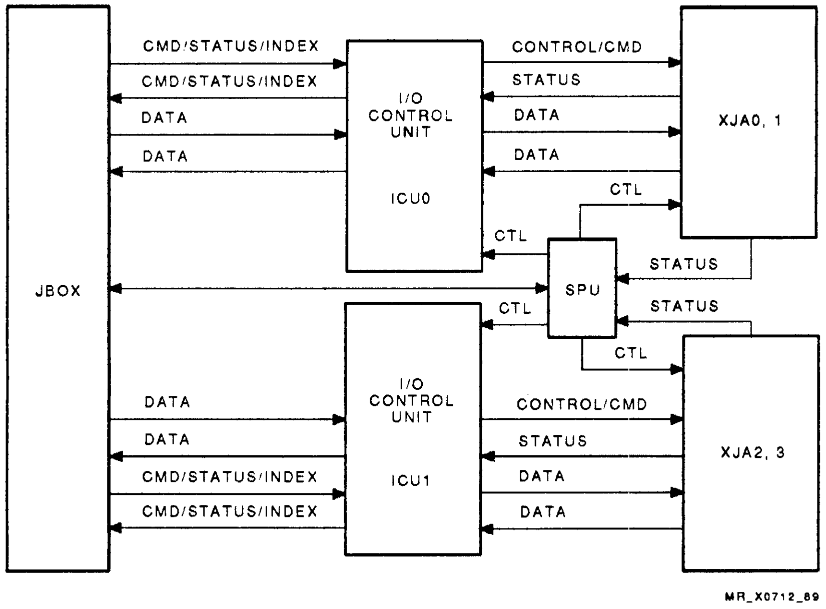


Figure 4–5 JBox-to-ICU Interface

4.2.5 SPU Port Interface

The SPU communicates with the CPU primarily through the logical interface that connects SCU to CPU. These paths are unidirectional, one byte wide, and capable of transferring one byte every 128 ns. This provides about 3.5 Mbytes of throughput for quadword transfers.

The SPU is MicroVAX system-driven, and BI interface-based, providing service and maintenance support for the computer system. The SPU serves as the operator console. It monitors the system, and tests and diagnoses hardware faults.

The ICU interfaces with the XJAs (over the JXDI cable) and SPU, implementing the central system interrupt arbiter.

The ICU connects the service processor (console) to the rest of the system. Communication uses SCU registers. These registers have I/O space addresses that configure memory and I/O. They also contain status about interrupts and exceptions.

CTLD receives inputs from SPU, JDC0, and JDBX and sends outputs to JDAX, JDC0, and SPU. CTLD receives SPU data and sends the data to the JDAX ICU input buffers. CTLD receives SPU handshaking signals and sends them to JDCX for control decode.

When the ICU sends command, address, and data with parity to the SPU, JDC0 sends handshaking signals to CTLD, JDBX sends data and address with parity to CTLD, and CTLD passes this on to SPU.

4.2.6 JBox Data Switch

The JBox uses a multipath data switch to connect CPUs, ACUs, and ICUs. The data switch provides paths for all inputs to be switched to any output.

The data switch allows multiple CPU, I/O, and SPU transactions to be executed simultaneously, as long as the transactions do not involve the same port. The CPUs, I/O units, and SPU can exchange data with main memory using the data switch.

The DSCT MCA receives, buffers, and decodes data transaction commands and determines the source ports (data switch inputs) and destination ports (data switch outputs). DSCT monitors the availability of data paths and sends the status to the JBox resource checking logic. The DSCT MCA also receives clock control signals from the clock module and sends these control signals to the MCUs.

The JBox keeps the ports active and resolves cache conflicts by handling communication between the ports and main memory. SCU handles a number of requests that may require the same data switch paths. These paths are considered resources. The JBox arbitration of port requests uses SCU resources in the most efficient manner and reserves previously unavailable resources for subsequent processing.

The JBox has 64-bit-wide data interfaces to the ACUs, CPUs, and ICUs. Each interface consists of two independent, 8-byte-wide data interfaces, one in each direction. Figure 4–6 shows the inputs and outputs of the data switch.

Data is transmitted over these interfaces every 16 ns (500 Mbytes/s). The data interfaces between the CPU and SCU originate and terminate at the MBox in the CPU. Figures 4–7 and 4–8 show the MBox-to-JBox and JBox-to-MBox data formats, respectively.

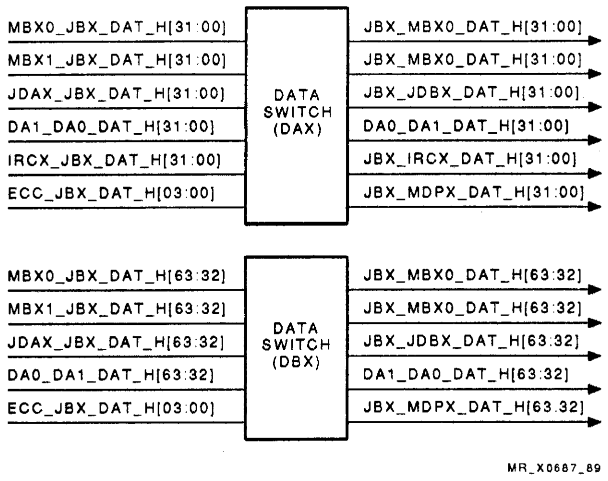


Figure 4–6 Data Switch

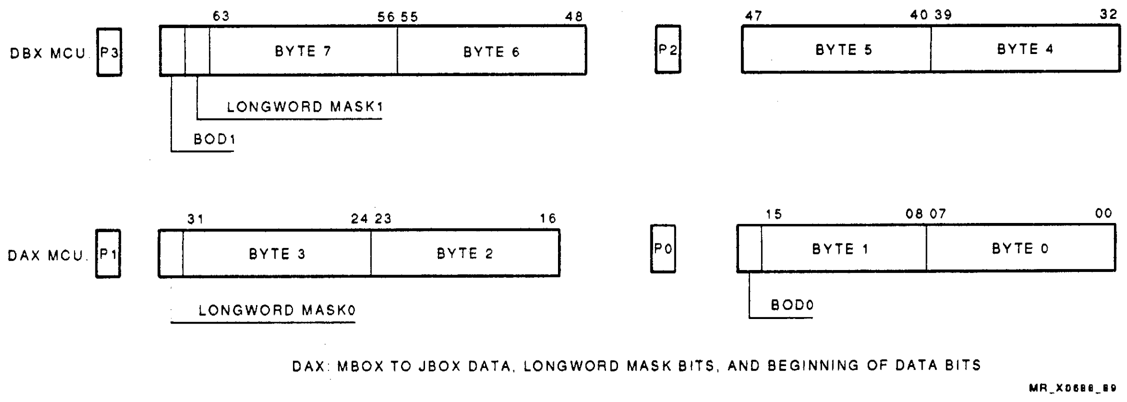


Figure 4–7 MBox-to-JBox Data Format

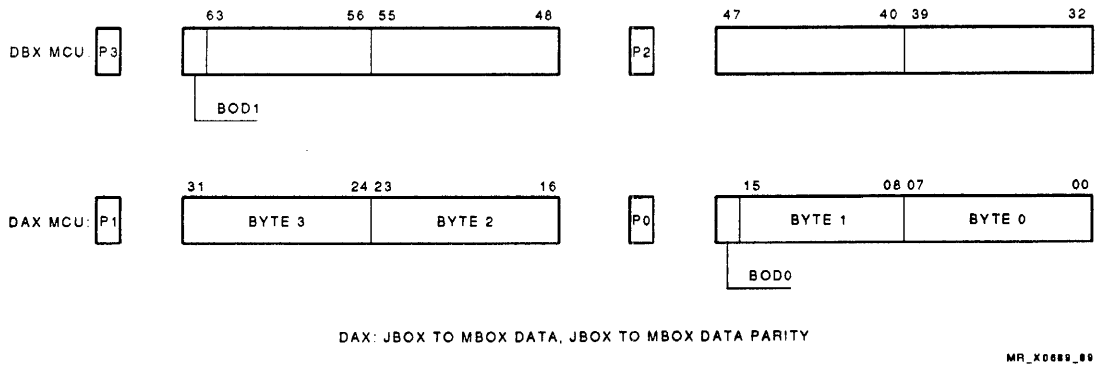


Figure 4–8 JBox-to-MBox Data Format

The data interfaces between the ACUs and JBox originate and terminate at the MDPX MCAs and DSXX MCAs. Write data flows from the DSXX MCAs to the MDPX MCAs. Read data flows from the MDPX MCAs to the DSXX MCAs.

The data interfaces between the ICUs and JBox terminate and originate at the JDBX and JDAX MCAs and the DSXX MCAs. Write data flows from the JDAX MCAs to the DSXX MCAs. Read data flows from the JDBX MCAs to the DSXX MCAs. Figures 4–9 and 4–10 show the JDAX-to-DSXX and DSXX-to-JDBX data formats, respectively.

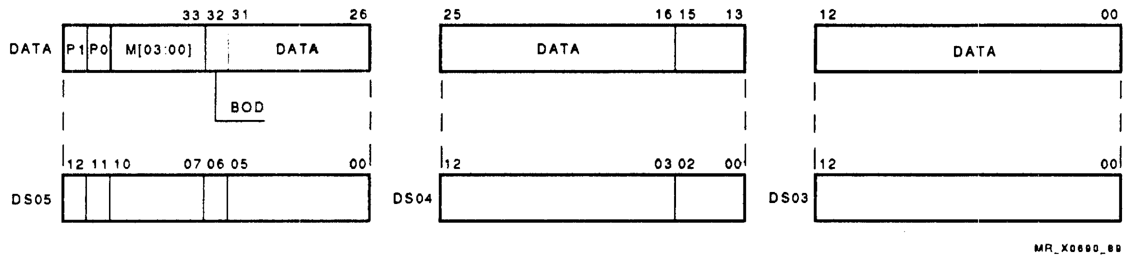


Figure 4–9 JDAX-to-DSXX Data Format

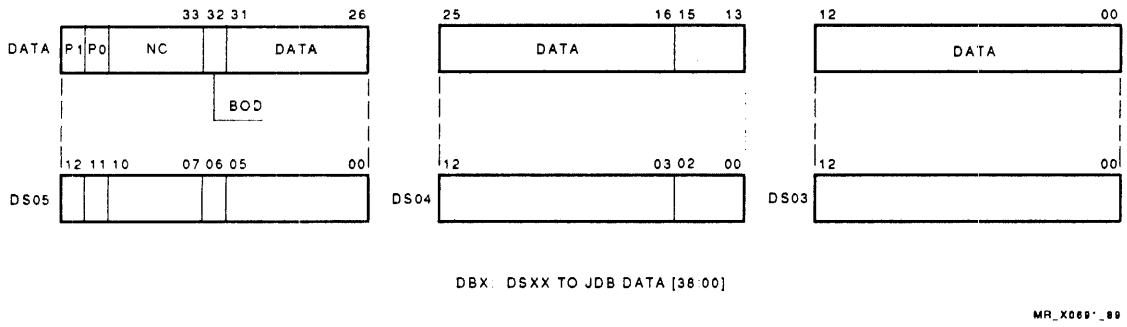


Figure 4–10 DSXX-to-JDBX Data Format

4.2.7 JBox Address Switch

The address switch receives and transmits addresses to and from the ports. The ADRX MCAs contain the address switch and send physical address bits to the following destinations:

- **Memory** — The address switch sends row and column address bits (scan determines row and column [32:26, 07, 06]). The row and column address bits address the following:

  Block and quadword within memory
  Unit, segment, and bank as defined by MPAMM
- **MTCH (match logic)** — The address switch sends PA[32:06]. MTCH drives the addresses and data of the tag STRAMs. MTCH also compares the physical address with the contents of the tag STRAMs and determines if there is match.
- **MBox** — The address switch sends eight 4-bit address slices to each MBox. This requires two cycles.
- **I/O controller** — The address switch sends two 4-bit address slices to each I/O controller. This requires two cycles.

The address switch sources row and column addresses to the main memory units. It receives addresses from MBox and I/O in two cycles. The row and column addresses sent to MMUs also require two cycles. Row address is sent first, then column.

The address switch contains three address receive latches for each CPU port and two address receive latches for each I/O port, a total of 16 address receive latches. The address latches receive 32-bit physical addresses in two cycles. The 16 latch addresses are wired in parallel to address memory, PAMM, CPU, and the STRAM match logic.

4.2.8 Cache Consistency Operations

Cache consistency ensures that if copies of a cache block exist in different CPUs, the data in the copies is consistent or identical. Whenever inconsistencies occur in the cache blocks, a requester (CPU or I/O) receives the most recent copy of the data.

In regard to cache consistency, the SCU performs either normal operations or exceptions. A normal operation is one in which the SCU receives a request for the most recent copy of the data. The data may reside in the requester's CPU cache or in another CPU cache.

An exception occurs when the SCU receives a request for data that: is not the most recent copy, and resides in the cache of another CPU. The tag status of the requester is inconsistent with the other CPUs, so the SCU must perform an exception operation. To fix the inconsistency, the SCU does the following:

1. Obtains a copy of the most recent data from the other CPU cache.
2. Sends a copy of the data to the requester.
3. Writes the tag status to the global tag.
4. Writes the data to main memory.

Inconsistency exists when a request, a requester's cache status, and another cache status are compared and different copies of the data exist for the same cache block.

Each CPU has a two-way, set-associative write back cache. The two sets are 0 and 1, with each containing 64 Kbytes of data. Each cache block is 64 bytes long and contains the following:

- One valid bit for each longword, for a total of 16 valid bits
- One written bit for the entire block

The valid and written bits are stored in a cache tag in the cache tag store. Figure 4–11 shows the format of the cache data found at each location in the data cache. Each cache set is 8K lines deep and 8 bytes wide. Each location consists of 8 bytes of cache data, 1 parity byte, and 1 ECC byte.

Figure 4–12 shows the relationship between the cache tags and the data cache in each CPU. Each cache reference results in the MBox addressing the cache tag to determine if the physical address being referenced is in cache set 0 or cache set 1. When a miss occurs in both cache sets, a cache refill is initiated.

The SCU maintains global tags. Whenever a CPU or I/O device makes a memory reference, the SCU examines the global tags and initiates the appropriate action to maintain cache consistency. Each tag STRAM location contains address, parity, and status bits. Figure 4–13 shows the global tag contents at one location in the global tag STRAMs.

BYTE 7	BYTE 6	BYTE 5	BYTE 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0	BYTE PARITY [07:00]	ECC [07:00]

MR_X0725_89

Figure 4–11 Cache Data (Quadword)

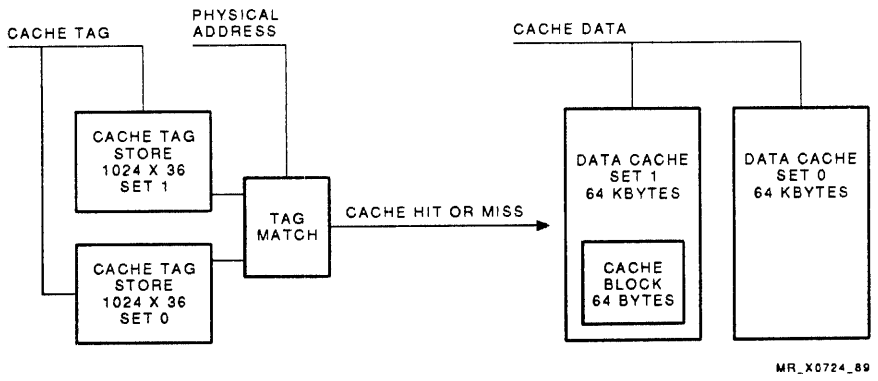

CACHE TAG
PHYSICAL ADDRESS
CACHE DATA
CACHE TAG STORE 1024 X 36 SET 1
CACHE TAG STORE 1024 X 36 SET 0
TAG MATCH
CACHE HIT OR MISS
DATA CACHE SET 1 64 KBYTES
CACHE BLOCK 64 BYTES
DATA CACHE SET 0 64 KBYTES
MR_X0724_89

Figure 4–12 Cache Tag Store and Cache Sets

ADDRESS BITS	AP	LP	SP	STATUS BITS

MR_X0728_89

Figure 4–13 Global Tag Contents

Global tag STRAMs contain copies of each of the CPU cache tag stores. The global tag STRAMs indicate read, written partial, written full, invalid, and lock status of cache blocks.

Figure 4–14 shows the address bits in the global tag STRAMs. The address match logic writes the address bits during a tag status write cycle and reads the address bits during a tag lookup read cycle.

Each tag location contains four status bits that define the state of that cache block in the CPU cache. Figure 4–15 shows the global tag status bits and parity. Table 4–3 lists the global tag status bits and corresponding descriptions.

For each memory reference, the SCU addresses the global tags and locates the data in one or more of the CPU cache sets. The SCU also determines the status of the CPU cache sets (invalid, written full, written partial, read, locked). An address match occurs when the requester's physical address matches address bits stored in the global tags.

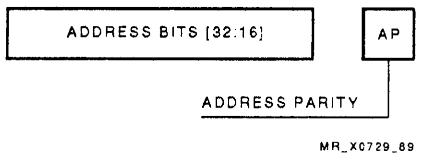


Figure 4–14 Global Tag Address Bits

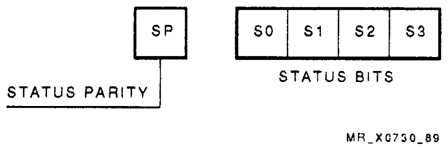


Figure 4–15 Global Tag Status Bits

Table 4–3 Global Tag Status Bits

Status Bit	Description
01:00	Indicates the status of the cache block for the address: 0 = Invalid 1 = Read 2 = Written partial 3 = Written full
02:00	Used for interlocks.

4.2.9.1 XJA-to-ICU Communication

All XJA-to-ICU transactions consist of one of the following transactions:

- **DMA (direct memory access)** — This transaction is a read or write of system main memory by an XMI device.
- **CPU** — This transaction is a read or write of an I/O register by the CPU. The I/O register may be in an XMI device or in the XJA.
- **Interrupts** — This transaction may be initiated by the XJA or by an XMI device. An interrupt initiated by the XJA is transferred over the JXDI to the system CPU for processing. An interrupt initiated by an XMI device is received by the XJA and passed on to the system CPU over the JXDI.

Transactions between the XJAs and the ICU are performed using packets. The packets contain command, length, IPL, address, and data information. The ICU contains control logic that decodes command packets and controls the loading and unloading of the ICU receive buffers.

When the ICU receives commands, it loads them into the JBox command buffers where they arbitrate for the resources required to complete the transactions. When the transactions win arbitration, the related data is loaded into the data switch and addresses are handled by the cache consistency logic. The JBox control logic initiates required cache consistency operations and initiates the reads or writes required to complete the I/O transactions.

For communication from the ICU to the XJAs, the ICU loads data and command packets into transmit buffers. The transmit buffers are controlled by the ICU and JBox control logic. Loading the transmit buffers and transmitting requires available JBox resources and controlling the data paths to the transmit buffers.

All ICU receive and transmit transactions consist of 32-bit data packets being sent in a single cycle.

4.2.9.2 SPU-to-ICU Communication

SPU communicates with ICU using the DMA, ECC, I/O, and interrupt transactions. The transactions use packets in which addresses are quadword-aligned, and parity must be correct for all valid and invalid bytes. Byte wrapping is not supported. The packets are transferred in 14 cycles, one byte at a time, and data context cannot be longer than a quadword.

The four transaction types are as follows:

- DMA transactions are reads, writes, read locks, or write unlocks, and can be up to a quadword in length.
- I/O transactions access the I/O portion of the physical address space.
- Interrupt transactions notify the operating system of console terminal receive and transmit, console storage device receive and transmit, powerfail, halt CPU, or keep-alive interrupts.
- ECC transactions notify SPU that the error checking and correction logic has detected an error and that SCU is sending the ECC address and the syndrome bits for error reporting and logging.

4.2.10 ACU Function

The SCU supports up to two ACUs, ACU0 and ACU1. ACU0 supports main memory unit (MMU) 0 and ACU1 supports MMU1. Each ACU contains built-in self-test (BIST) logic to support the SPU during self-tests. The ACUs send and receive commands and data to and from:

- **SPU** — The two ACUs receive SPU control signals for testing the memory modules.
- **MMUs** — The two ACUs send and receive commands and data to and from the two MMUs.
- **JBox** — The ACUs receive and send memory commands, and control and status information to and from the JBox. The JBox receives CPU and I/O memory requests and sends the requests to the ACU. The JBox contains a port controller that controls two command buffers for commands received from the ACU. The JBox also contains four memory segment controllers to which the ACU sends the status of each memory segment in each MMU.

The ACU controls the main memory unit, DRAMs, and DRAM data paths on the memory modules and provides the SCU memory data path from the memory modules to the JBox data switch. For addressing the DRAMs, the JBox holds the row and column addresses in the ADRX MCAs. The ACU uses an index value that the JBox sends with the command. The index value selects the appropriate value from the ADRX MCAs.

The ACU is comprised of main memory control MCAs, memory control MCAs, and memory data path MCAs. Each SCU can have up to two ACUs. ACU0 controls MMU0 and ACU1 controls MMU1.

The main memory control MCAs provide command, data and address control, and status interface to the JBox. They also provide DRAM control signals (RAS, CAS, and WE) for the memory data path and for the memory modules.

The memory data path MCAs provide a 4-byte wide data path (in each direction), check bit generation for write data, and a byte merge path. This MCA is also responsible for detecting and correcting single-bit errors for read data and can detect but not correct double-bit errors.

The following sections provide a brief overview of memory read and write operations.

4.2.10.1 Memory Read Operations

When a read data cycle executes, MMU reads 640 DRAMs and loads the data into read buffer 0 of the memory modules. The DRAMs load data into read buffer 0, and the memory module transfers the data to read buffer 1. This allows a second read operation from the other segment to continue. Read buffer 1 feeds into a 8:1 selector used to wrap data 80 bits per clock cycle. The first 80-bit word to be transferred is determined by the starting quadword field in the command buffer.

MMCX does not forward the length field to either MMU or the MDPX MCAs. The length is used in MMCX by the read data controller and write data controller state machines. The length field determines how long certain control signals from MMCX to MMU and MDPX should be asserted.

During writes, MDPX has no knowledge of the length of the data transfer. Whatever data enters MDPX from the data switch emerges from MDPX two clock ticks later with the appropriate check bits appended. It does not matter if the data is valid write data. If it is valid write data, MMCX sends the appropriate control signals to MMU to load the data into the data buffer. The length is conveyed to MMU by the number of write buffer strobe (WRTBSTROBE) pulses. A write buffer strobe accompanies each quadword and loads the quadword into the data input latch (write buffer 0). After the memory module loads the last quadword into write buffer 0, MMCX sends one write buffer strobe pulse and transfers the data from the data input latch (write buffer 0) into the write data buffer (write buffer 1). When the data is in the write data buffer, MMCX concurrently transmits the write flip-flop enable (WRTFFEN_L) with the write select lines.

4.2.10.3 MMUs

The memory subsystem consists of the ACUs, MMUs, and control for testing the memory modules provided by the SPU. The main memory subsystem is a nonbussed, block-oriented, high-bandwidth system. Cables connect the MMU to the SCU planar module. The ACUs provide the data path between the JBox and the MMUs. The tag MCU on the SCU planar module provides the address path between the SCU and the MMUs.

Parameters for a fully configured (MMU0 and MMU1) memory include the following:

- Four-way interleaving
- Maximum capacity of 512 Mbytes using 1-Mbit DRAMs
  (Each MMU provides 256 Mbytes using 1-Mbit DRAMs.)
- Read and write bandwidth of 500 Mbytes per second
- 280-ns read latency (from receipt of command to transmit of data to the JBox)
- Memory expansion support

The MMU has four extended hex memory modules that reside in a card cage. Each memory module consists of a main array card (MAC) and two daughter array cards (DACs).

Each MMU has two segments. Each segment contains two banks, 0 and 1. Control and address lines operate independently across segments. The write path and the read data path are common to both segments. The paths are separated into a read and write path of 20 bits per memory module (four memory modules = 80 bits). Each segment receives 11 control signals (from the MCDX MCA) and 12 (row and column [00:11]) address lines (from the ADRX MCAs).

4.2.10.4 Memory Modules

The memory module is a nonstandard hex-size module with a 480-pin, right-angle connector at the edge of the card. Each module stores 640 million data bits. The memory module contains the following components:

- Six hundred and forty double-sided surface-mounted DRAMs
- Four DRAM data path (DDP) gate arrays
- One DRAM control and address (DCA) gate array

Each memory module incorporates two-way interleaving (across segments) and each contains two segments (with two banks to a segment). Each module provides a maximum of 64 Mbytes of memory (using 1-Mbit DRAMs).

The DAC contains surface mounted DRAMs on both sides. The DAC provides 16 Mbytes of DRAM storage. Each MAC has two DACs, DAC0 and DAC1.

4.3 SCU Physical Description

The SCU logic resides on a single planar module located in the SCU cabinet (Figure 4–16). The SCU planar module can accommodate up to six MCUs. The minimum configuration for an SCU planar module consists of four MCUs and can support up to two CPUs, one ACU, and one ICU. Two additional MCUs are required to support a third and fourth CPU and an additional ACU and ICU.

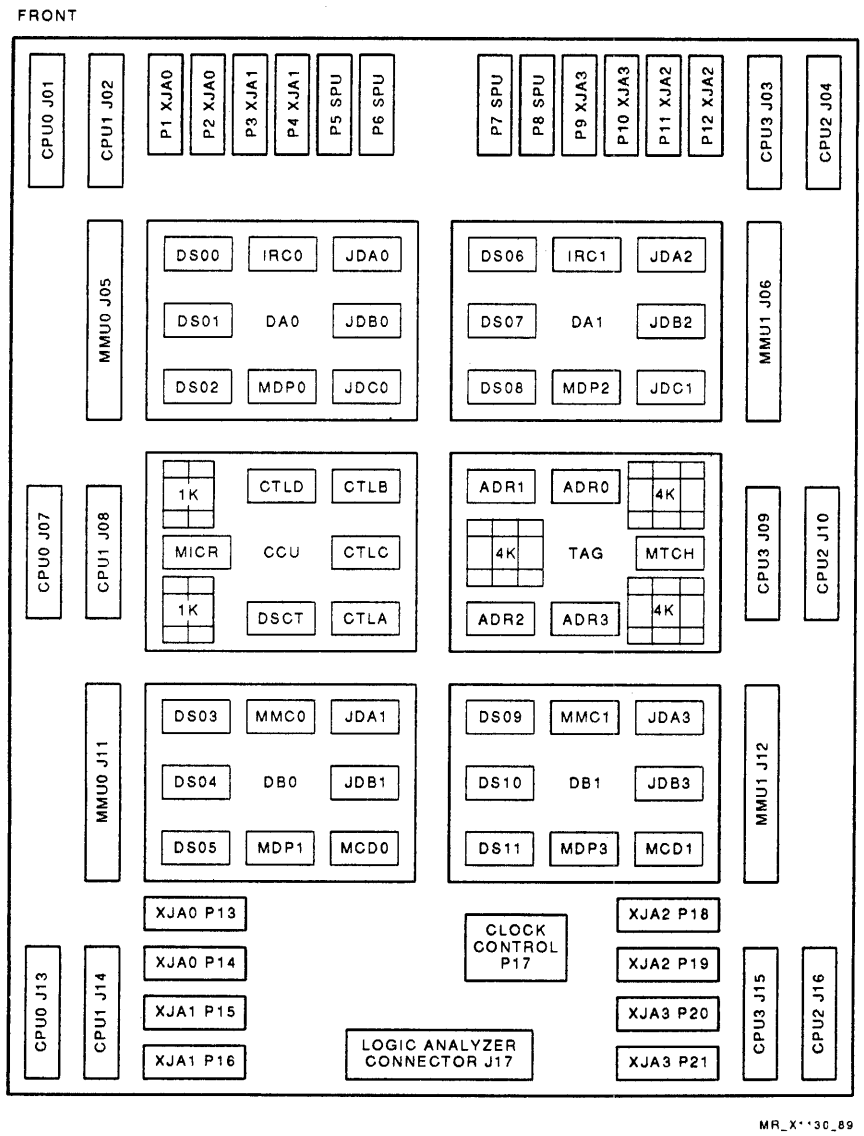


Figure 4–16 SCU Planar Module

The following sections introduce each of the SCU MCUs and describe their MCA content and function.

4.3.1 DA0 MCU

The DA0 MCU transfers I/O commands to the CCU (cache consistency unit) MCU. DA0 contains the SCU registers and provides a data path between the data crossbar and the memory array cards. DA0 contains eight MCAs and handles the following address and data slices:

Bytes 0 through 3 of the data to the data crossbar
Two of the four address bytes to the tag MCU
One of the two nibbles to the SPU interface
One of the two bytes of the JXDI interface

The following list introduces the DA0 MCAs and summarizes their functions:

- **JDA0** — This MCA is the XJA and SPU data buffer. It receives one byte of the interface from each of the two XJAs. It receives one nibble of the SPU interface. It sends one-half of the address bits to the tag MCU. It buffers and sends the data from the XJAs (through a 4-byte-wide path) to the group of DSXX MCAs. It sends commands from the ICU to CCU.
- **JDB0** — This MCA is similar to the JDA0 MCA but handles the signals in the reverse direction.
- **JDC0** — This MCA controls the operation of the JDA0 and JDB0 MCAs and monitors their errors. The control functions include coordination of handshaking signals to and from the JXDI and to and from the CCU. Clock signals that are transmitted to the XJAs are sourced in this MCA.
- **DSXX** — The data switch MCAs (DS00, DS01, and DS02) are 64-bit block multiplexers that provide crossbar capability for 4 bytes of data. They support CPUs 0 and 1, memory port 0, and XJAs 0 and 1. For register reads and writes, they have a 4-byte-wide path to and from the IRC.
- **IRC0** — This MCA contains the SCU registers and interfaces to the crossbar. It contains the interrupt logic for I/O-to-CPU interrupts and inter-CPU interrupts. It handles the handshaking signals for the SPU interface.
- **MDP0** — This MCA provides a 4-byte-wide path between the data crossbar and the memory array cards. It handles ECC and read-modify-write operations. It provides check bit generation for write data. It detects and corrects single-bit errors and detects double-bit errors. It generates data patterns during BIST. It contains the byte merge logic.

At maximum configuration, the SCU contains an identical copy of the DA0 MCU. The second MCU is DA1. It contains the following MCAs that support identical functions for the expansion ports:

JDA2
JDB2
JDC1
DS06, DS07, and DS08
IRC1
MDP2

4.3.2 CCU MCU

The CCU MCU contains the cache consistency unit and the JBox control unit. It tracks valid cache data locations and manages data to and from the ports. The MCU contains six MCAs and eighteen 1K × 4 STRAMs. The STRAMs contain the SCU microcode and a microPC history buffer. The following list introduces the CCU MCAs and STRAMs and describes their functions:

- **CTLA** — The control A MCA receives requests (load commands) for data movement. It contains the port arbitration logic. It generates an index that points to a command (in CTLB) and an address (in tag).
- **CTLB** — The control B MCA receives and stores 20 port commands and distributes the commands to other ports.
- **CTLC** — The control C MCA sends commands to the data switch controller MCA (DSCT). It sends cache consistency information to a queue (in CTLD) and sends consistency commands to the ports.
- **CTLD** — The control D MCA contains the consistency cache queue. The microcode accesses the queue that contains cache check information.
- **DSCT** — The data switch controller MCA controls the data switch MCAs (DSXX) located in the DBX MCU.
- **MICR** — This MCA controls the SCU microcode.

4.3.3 DB0 MCU

The DB0 MCU provides control for the memory array cards and monitors the memory array status. DB0 contains eight MCAs and handles the following address and data slices:

Bytes 4 through 7 of the data to the data crossbar
Two of the four address bytes to the tag MCU
One of the two nibbles of the SPU interface
One of the two bytes of the JXDI interface

The following list introduces the DB0 MCAs and summarizes their functions:

- **JDA1** — The XJA and SPU data buffer MCA receives one byte of the interface from each of the two XJAs. It receives one nibble of the SPU interface. It sends one-half of the address bits to the tag MCU. It buffers and sends the data from the XJAs (through a 4-byte-wide path) to the group of DSXX MCAs. It sends commands from the ICU to CCU.
- **JDB1** — This MCA is similar to JDA1 but handles the signals in the reverse direction.
- **DSXX** — The data switch MCAs (DS00, DS01, and DS02) are 64-bit block multiplexers that provide crossbar capability for 4 bytes of data. They support CPUs 0 and 1, memory port 0, and XJAs 0 and 1. For register reads and writes, they have a 4-byte wide path to and from the IRC.
- **MMC0** — The main memory control MCA provides the control signals to the memory array cards and receives status from them. It provides the command, control, and status interface to the JBox. It provides the data path control and DRAM control commands to the MCD. It provides the error detection on all MMC control lines and supports BIST operations.

- **MDP1** — The memory data path 1 MCA provides a 4-byte-wide path between the data crossbar and the memory array cards. It handles ECC and read-modify-write operations. It provides check bit generation for write data. It detects and corrects single-bit errors and detects double-bit errors. It generates data patterns during BIST. It contains the byte merge logic.
- **MCD0** — This MCA provides DRAM timing and control as well as commands to the MMC MCA during self-test.

When the SCU is fully configured the DB1 MCU is installed. It contains the following MCAs that support identical functions for the expansion ports:

JDA3
JDB3
DS09, DS10, and DS11
MMC1
MDP3
MCD1

4.3.4 Tag MCU

The tag MCU receives and transmits addresses to and from the ports. It controls the tag STRAMs and determines whether there is an address match. The tag MCU contains five MCAs, twenty-four 4K × 4, and three 1K × 4 STRAMs. The following list introduces the tag MCAs and describes their functions:

- **MTCH** — The MTCH MCA drives the addresses and data to the tag STRAMs. It receives addresses from the tag STRAMs and matches these addresses with the addresses received from the ports. It sends address match signals to the CCU.
- **ADRX** — Each address MCA (ADR0, ADR1, ADR2, and ADR3) handles one-fourth of the address bits. Each receives one-fourth of the address field from the four CPU ports and two I/O ports and transmits one-fourth of the address field to the same ports. They source row and column addresses to the MMUs. The address signals from each port are double buffered to accommodate the frequent occurrence of a write back accompanying a refill.

5 SPU and Scan Subsystem

This chapter describes the service processor unit (SPU) hardware and software and the scan system. The scan system description includes a general description of scan and its implementation.

5.1 Service Processor Functional Overview

The SPU is a subsystem driven by a MicroVAX chip and based on the BI adapter. It provides service and maintenance support for the computer system. The SPU serves two major functions:

- It is the operator console and initialization controller used to start up, shut down, and monitor the operations of the system.
- It is the maintenance processor used by Customer Services to test, diagnose, and isolate hardware faults in the system either locally at the customer's site or remotely through a dial-up port.

In model 400 systems, the SPU is located at the front right of the CPU cabinet that contains CPU0 (Figure 5–1). In model 200 systems, the SPU is located in the XMI cabinet (Figure 5–2).

The following list identifies the major components of the SPU:

- BI backplane assembly for mounting and connecting the SPU modules
- Service processor module (SPM)
- Scan control module (SCM)
- Power and environmental monitor (PEM)
- KFBTA disk controller module
- RD54 disk drive
- DEBNK network and tape controller module
- TK50 tape drive
- H7214 +5 V power supply
- H7215 -5.2 V power supply

5.1.2.1 SJI (SCU Interface)

The SPU-to-JBox interface (SJI) provides a logical interface that connects the SPU to the SCU. This interface is used to:

- Load the primary bootstrap (VMB) into main memory.
- Transfer error information from the SPU to the operating system.
- Transfer SPU updates from the operating system to the SPU.
- Transfer machine check information to the operating system.
- Access main memory and I/O registers.

The interface supports two 8-bit data buses, one for transferring data from the SPU to the SCU and one for transferring data from the SCU to the SPU. The data transfer rate is about 3.5 Mbytes per second. Five registers (mask, command, address (2), and length) in the SPU-to-JBox adapter (SJA) chip support DMA transfers over the SJI.

5.1.2.2 SCI (Scan System Interface)

The scan interconnect (SCI) provides direct read/write access to over 20,000 latches in the CPUs, SCU, and master clock module (MCM). It consists of six separate cables, four for the CPUs and one each for the SCU and MCM. Information is transferred serially between the SCM and the CPUs, SCU, and MCM. This interface is used to:

- Initialize all the latches in the CPU and SCU during system startup.
- Control the MCM (start/stop clocks, change frequency, and monitor status).
- Load control store self-timed RAMs (STRAMs).
- Access STRAMs and self-timed register (STREG) data in the CPUs and SCU.
- Read the state of individual address, data, control, and status latches in the CPUs, SCU, and MCM.
- Set the individual address, data, control, and status latches in the CPUs, SCU, and MCM.
- Signal errors from the CPU, SCU, and MCM to the SPU.
- Execute scan pattern diagnostics.

5.1.2.3 SPI (PCS Interface)

The SPU-to-PEM interface (SPI) supports transferring control and status information between the SPU and the PCS. This interface is used to:

- Control turning power supplies on and off.
- Monitor the voltage output of all power supplies.
- Monitor switch settings and control indicators on the operator control panel (OCP).
- Monitor the state of various temperature sensors and air flow indicators in the CPU and I/O cabinets.

The SPI includes an 8-bit bidirectional data path to transfer message packets between the SPM and the PEM.

5.1.3 Module Descriptions

This section provides a brief overview of the function of each of the five modules in the SPU.

5.1.3.1 T2051 Service Processor Module (SPM)

The SPM is the main processing element of the SPU. It contains all the hardware facilities to support storing and executing the VAXELN service processor software application. The SPM acts as the host computer connected to the VAXBI bus and controls the operation of the other four modules: SCM, PEM, KFBTA, and DEBNK.

The SPM is driven by a MicroVAX 78032 chip. A 16-Mbyte, dynamic ECC RAM provides the SPU main memory. The SPM firmware and self-tests are contained in a 128-Mbyte, on-board RAM. Additional storage is provided by a 32-Kbyte EEPROM for system parameters, and a 128-Kbyte PROM for self-tests and resident firmware.

A standard 78332 system support chip (SSC) is incorporated into the SPM. The SSC provides a time-of-year (TOY) clock, two asynchronous terminal ports, 1 Kbyte of battery backed-up memory, PROM unpacking logic, and a general-purpose, 4-bit output port. It also includes a 100-Hz timer for the MicroVAX chip.

The following list summarizes the functions of the SPM:

- **MicroVAX processor** — The SPM is driven by a MicroVAX 78032 chip that provides the MicroVAX subset instruction set with full VAX memory management features at about 90 percent the performance of a VAX-11/780 system.
- **RAM** — A 16-Mbyte, dynamic ECC RAM, contained on a daughterboard, provides the main memory for the SPU.
- **On-board RAM** — The SPM firmware and self-tests use this 128-Mbyte RAM.
- **EEPROM** — This 32-Kbyte memory is used to store system parameters (bad page bit map, default load device).
- **SPU memory controller (SMC)** — This custom designed chip provides the interface between the II32 bus and the ECC RAM.
- **PROM** — A 128-Kbyte PROM provides storage for the SPM self-tests and the resident firmware required to load the SPU software from the RD54 disk during system initialization.
- **System support chip (SSC)** — Digital's standard 78332 chip provides a time of year (TOY) clock, two asynchronous terminal ports (CTY and PTY), 1 Kbyte of battery backed-up memory, PROM unpacking logic, and a general-purpose 4-bit output port. It also includes a 100-Hz timer for the MicroVAX chip.
- **RTY interface** — Full modem support provides remote dial-in and dial-out capability to allow the Customer Support Center (CSC) to perform system testing.
- **SPU-to-SCU interface (SJI)** — A custom designed SPU-to-JBox adapter (SJA) chip provides the logic facilities to transfer information between the SPU and the SCU.
- **SPU-to-PEM interface (SPI)** — The SPI is a communications path between the SPM and the PEM module.
- **BIIC/BCI3** — This standard VAXBI chip set provides the hardware interface to the BI bus.

- **Integrated circuit interconnect (II32)** — This standard internal bus connects the MicroVAX chip to the SMC, SSC, SJA, SPI, PROM, and BCI3. All information transfers between these elements occur over the II32 bus.

5.1.3.2 T2050 Scan Control Module (SCM)

The SCM is driven by a MicroVAX 78032 chip. A 128-Kbyte PROM provides storage for the fixed segment of the SCM firmware and self-tests. A 512-Kbyte RAM provides local storage for the data structures used to access the scan system, and to buffer data transfers between the SCM and SCI. The RAM also provides storage for additional firmware routines loaded by the SPM.

The scan control chip (SCC) provides the control logic to transfer information between the SCM and scan system through the SCI. Two scan distribution chips (SDCs) interface the SCC to six individual SCI ports. Each SDC drives three ports: one drives ports to CPU0, CPU1, and the SCU; the other drives ports to CPU2, CPU3, and the MCM.

The following list summarizes the functions of the SCM:

- **MicroVAX processor** — The SCM is driven by a MicroVAX 78032 chip that provides the MicroVAX subset instruction set with full VAX memory management features at about 90 percent the performance of a VAX-11/780 system.
- **PROM** — A 128-Kbyte PROM provides the storage for the fixed segment of the SCM firmware and the module self-tests.
- **RAM** — A 512-Kbyte RAM provides local storage for all the data structures used by the firmware to access the scan system and also to buffer all data transfers between the SCM and the SCI. It also provides storage for additional firmware routines loaded by the SPM via the BI bus.
- **Dynamic RAM controller chip (DYRC)** — Digital's standard 78584 chip provides the interface to the RAM.
- **Scan control chip (SCC)** — This custom designed chip contains all the logic to control the transfer of information between the SCM and the VAX 9000 scan system via the SCI.
- **Scan distribution chip (SDC)** — Two custom designed gate array chips interface the SCC to six physically separate SCI ports. Each SDC chip drives three SCI ports; one connects to CPU0, CPU1, and the SCU, while the other connects to CPU2, CPU3, and the MCM.
- **BIIC/BCI3** — This standard VAXBI chip set provides the hardware interface to the BI bus.
- **Integrated circuit interconnect (II32)** — This standard internal bus connects the MicroVAX chip to the DYRC, RAM, PROM, SCC, and BCI3. All information transfers between these elements occur over the II32 bus.

5.1.3.3 T1060 Power and Environmental Monitor (PEM)

The PEM module contains all the hardware facilities and firmware that allow the SPU software to control and monitor the status of the VAX 9000 power and environment:

- **Power control system (PCS)** — The PEM module turns power supplies on and off, and it responds to any abnormal changes in the state of the power system.
- **Operator control panel (OCP)** — The PEM module reads the state of the OCP switches and controls the state of the OCP indicators.

- **Environment** — The PEM module monitors temperature and air flow throughout the system and responds to any abnormal changes.

The PEM is driven by an 8031 microprocessor. The 16-bit SPI provides the PEM interface to the SPM. The 32-Kbyte PROM contains the PEM firmware and module self-tests. A 32-Kbyte EEPROM provides storage for firmware updates loaded by the SPM. In addition, a 32-Kbyte RAM provides storage used by the firmware to process and buffer information packets.

The following list summarizes the functions of the PEM:

- **Intel 8031 microprocessor** — The PEM module is driven by an 8031 microprocessor that executes the PEM firmware.
- **PROM** — A 32-Kbyte, read-only memory provides local storage for the PEM firmware and module self-tests.
- **EEPROM** — A 32-Kbyte, electronically erasable PROM provides local storage for firmware updates that are down-line loaded from the SPM.
- **RAM** — A 32-Kbyte, read/write memory provides local storage used by the firmware to process and buffer information packets.
- **SPU-to-PEM interface (SPI)** — A 16-bit communications bus provides the data transfer path between the SPM and the PEM modules.

5.1.3.4 T1031 KFBTA Disk Controller Module

The KFBTA disk controller provides access to the RD54 disk, which contains the SPU file system. It is a standard BI adapter implemented as a single-host BI storage systems port that supports standard communication architecture (SCA). All file transfers to and from the disk are controlled by the SPM and occur over the BI bus.

The KFBTA module is driven by a MicroVAX 78032 chip. A 128-Kbyte PROM contains the module firmware and self-tests. A 32-Kbyte RAM provides storage for the firmware buffers. A disk controller chip (DP844) supports the standard MPSC disk interface protocol.

The following list summarizes the KFBTA functions and logic implementation:

- **MicroVAX processor** — The KFBTA module is driven by a MicroVAX 78032 chip that provides the MicroVAX subset instruction set with full VAX memory management features at about 90 percent the performance of a VAX-11/780 system.
- **DP844** — This disk controller chip supports the standard MPSC disk interface protocol.
- **PROM** — A 128-Kbyte, read-only memory provides local storage for the KFBTA firmware, which includes diagnostic self-tests.
- **RAM** — A 32-Kbyte, read/write memory provides local storage for buffers used by the firmware.
- **BIIC/BCI3** — This standard VAXBI chip set provides the interface to the BI bus.
- **Integrated circuit interconnect (II32)** — This standard internal bus connects the MicroVAX chip to the RAM, ROM, and disk control logic.

5.1.3.5 T1034 DEBNK Network/Tape Controller Module

The DEBNK module provides access to the TK50 tape drive and the Ethernet (NI). It is a standard BI adapter that contains both network and tape interfaces.

The DEBNK module is driven by a MicroVAX chip. A 128-Kbyte PROM contains the MicroVAX firmware and self-tests, while a 128-Kbyte RAM provides buffer storage for the firmware.

An 80186 microprocessor chip and a multiprotocol serial controller chip (MPSC) provide the interface to the TK50 tape drive. The multiprocessor is supported by a 32-Kbyte ROM, which contains the firmware and self-tests, and a 16-Kbyte RAM, which provides firmware buffer storage.

The following list summarizes the functions and physical characteristics of the DEBNK module:

- **MicroVAX processor** — The DEBNK module is driven by a MicroVAX 78032 chip that provides the MicroVAX subset instruction set with full VAX memory management features at about 90 percent the performance of a VAX-11/780 system.
- **PROM** — A 128-Kbyte, read-only memory provides local storage for the MicroVAX firmware and self-tests.
- **RAM** — A 128-Kbyte, read/write memory provides buffer storage for the firmware.
- **LANCE chip** — This chip is a local area network controller for Ethernet.

  NOTE
  The network interface was used only for prototype debugging of the VAX 9000 and for remote service delivery during field test.

- **Intel 80186/MPSC** — An Intel 80186 microprocessor chip and a multiprotocol serial controller chip provide the interface to the TK50 tape drive supported by a:
  - **PROM** — A 32-Kbyte, read-only memory provides local storage for the 80186 firmware, which includes diagnostic self-tests.
  - **RAM** — A 16-Kbyte, read/write memory provides local storage for buffers.
- **BIIC/BCI3** — This standard VAXBI chip set provides the interface to the BI bus.
- **Integrated circuit interconnect (II32)** — This standard internal bus connects the MicroVAX chip to the internal memory bus.

5.2 SPU Software

The SPU software is a dedicated VAXELN application that resides in BI common memory on the SPM. It is executed under control of the VAXELN kernel and deals solely with controlling and monitoring the operation of the VAX 9000. Figure 5–4 shows the functional relationship between the major elements in the SPU software image that is loaded into BI common memory during system initialization.

The VAXELN operating system is a real-time, real-memory system that provides a memory-resident kernel that can be used by a dedicated application. It provides the following functionality:

- VMS compatible file service
- Phase IV DECnet end node network service
- System services required for real-time multitasking applications
- Support for VAX C, VAX FORTRAN (VAXELN V2.3), VAXELN Pascal, and VAX MACRO programs

The kernel provides memory management (VAX P0/P1 mapping) without paging or swapping. The absence of page files and swap files increases system reliability and performance by removing the dependency on the RD54 disk drive.

The majority of the SPU software is written in the VAX C language with the remainder written in the VAX MACRO and VAXELN Pascal languages. Using the VAXELN kernel makes it possible to include service processor functions in the operating system itself rather than executing them as application programs, which provides more consistent functionality across the various console modes of operation.

The VAXELN operating system, as used in the SPU, is not intended to be a general-purpose operating system for developing customer applications.

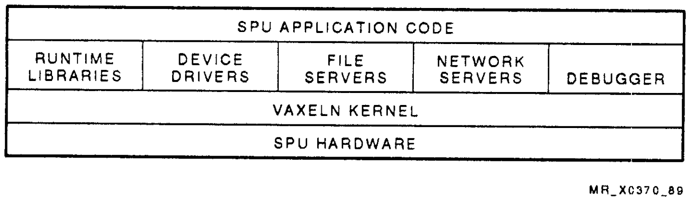


Figure 5–4 VAXELN System Elements

5.3 Scan Concepts

The VAX 9000 is the first computer system designed by Digital Equipment Corporation that uses scan. This section briefly describes what scan is and how it works.

Scan is a technique for designing testable logic. It considers the problems of logic testing and fault isolation during the design stage. Incorporating a scan system into a logic design provides a more accurate test method, because the state of latches can be accessed independently of the combinational logic. Scan tests the operation of latches before it tests the combinational logic of latches. Access to these latches allows:

Initialization of the CPUs, SCU, and MCM at system startup
Control of the system clocks (start, stop, etc.)
Loading control store STRAMs
Reading or setting the state of latches in the CPUs and SCU
The CPUs and SCU to signal errors to the SPU
Execution of scan-based diagnostics

Scan-based diagnostics set latches to known states and then read the states of the latches to verify that they are functioning correctly. When testing multiple CPUs, it is possible to set the latches in all the CPUs to the same state, to read out the state of the latches, and to compare the results from each CPU.

5.3.1 Non-Scan Model

Most digital systems are similar to the simple model shown in Figure 5–5.

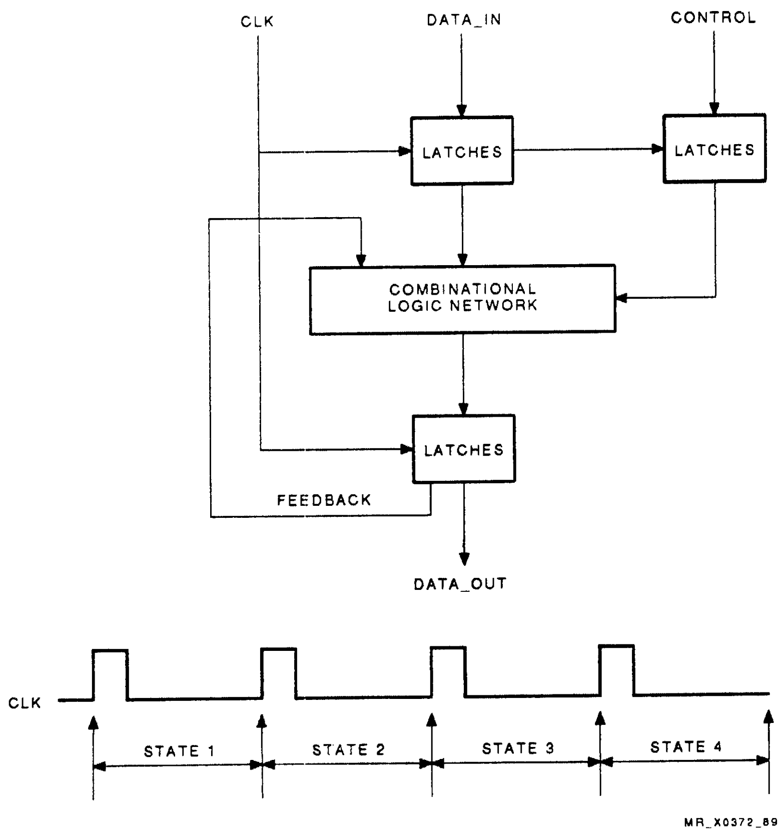


Figure 5–5 Model of a Simple Digital System

Regardless of size, most digital systems consist of a combinational logic network and bi-stable latch elements. The combinational logic network contains hundreds or even thousands of logic gates (AND, OR, and NOT gates) that perform the required decision-making functions. The latches surround the combinational logic and serve as memory elements to temporarily store input data, output data, or control information.

The state of the system is defined by the state of all of its latches. Usually, the state of the system changes at the occurrence of each CLK pulse. At each CLK pulse, the next state of the system is determined by the state of the input latches, control latches, output latches, and the structure of the combinational logic network.

The four CLK pulses shown in Figure 5–5 would sequence the machine through four unique states, state 1 through state 4. The feedback path in Figure 5–5 implies that the next state of the machine is influenced by the current state of the machine. This feedback is characteristic of any sequential machine and complicates the testing problem.

5.3.2 Scan Model

Adding a scan system modifies the design of the system to add mechanisms that permit direct access and control of individual latch elements by the test program. Figure 5–6 shows the model from Figure 5–5, modified to incorporate scan.

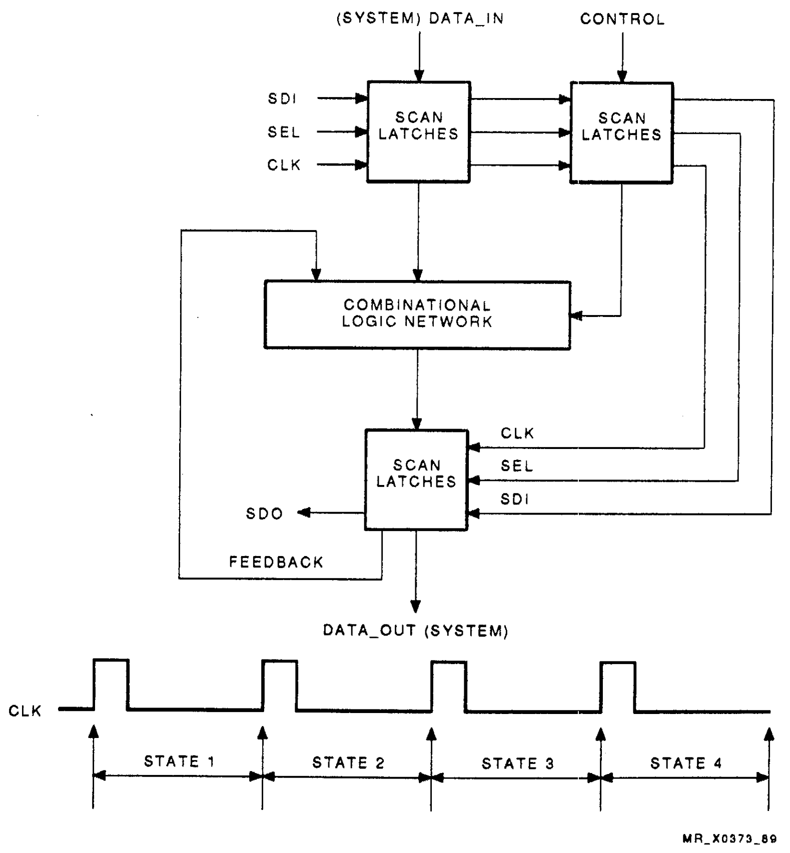


Figure 5–6 Model of a Simple Digital System Using Scan

All the latches are modified to function like parallel-load, serial-shift registers connected end to end. This allows reconfiguring of the latches into one giant serial shift register for test purposes. Three additional signals are added to control the latches:

- **SEL** — The select input signal modifies operation of the latches to disconnect the normal system data and control inputs, and enables the latch to function as a shifter.
- **SDI** — The serial data in signal provides a diagnostic path to shift binary test patterns into the system to establish a known state.
- **SDO** — The serial data out signal provides a path to shift patterns out of the system to test the results.

5.3.3 Testing with Scan

Assume that the CLK, SEL, SDI, and SDO signals are connected to a special diagnostic test machine. The basic test strategy consists of the following steps:

1. Use scan mode (SEL = 1) to test the connectivity of the scan path and the operation of all the scan latches in the path by shifting test patterns in at SDI and out at SDO.
2. Determine the optimum set of test patterns required to test the combinational logic network.
3. Apply each of the test patterns as follows:
   a. Use the scan mode (SEL = 1) to scan in the test pattern.
   b. Use the non-scan mode (SEL = 0) along with a system clock to load the output from the combinational logic network into the latches.
   c. Use the scan mode (SEL = 1) to scan out the result pattern and compare it to an expected result pattern.

Scan latches can also be used to access RAM structures. Reading or writing a RAM requires latches surrounding the RAM that hold the:

Address to be read or written
Input data
Output data
Control information that enables the RAM and specifies read or write

By connecting the latches that support the control, addressing, and data functions of the RAM in a serial scan path, it is possible to test the RAM. Testing begins when the scan system loads address, function, and input data into the RAM and its supporting latches. The scan system is used to enable a read of the RAM and the data from the RAM is then read and compared to a known state.

5.4 Scan System Overview

The VAX 9000 scan system shown in Figure 5–7 consists of a combination of software, firmware, and hardware that controls access to the scan latches in each of the four CPUs, the SCU, and the MCM. The following sections identify and describe the major components.

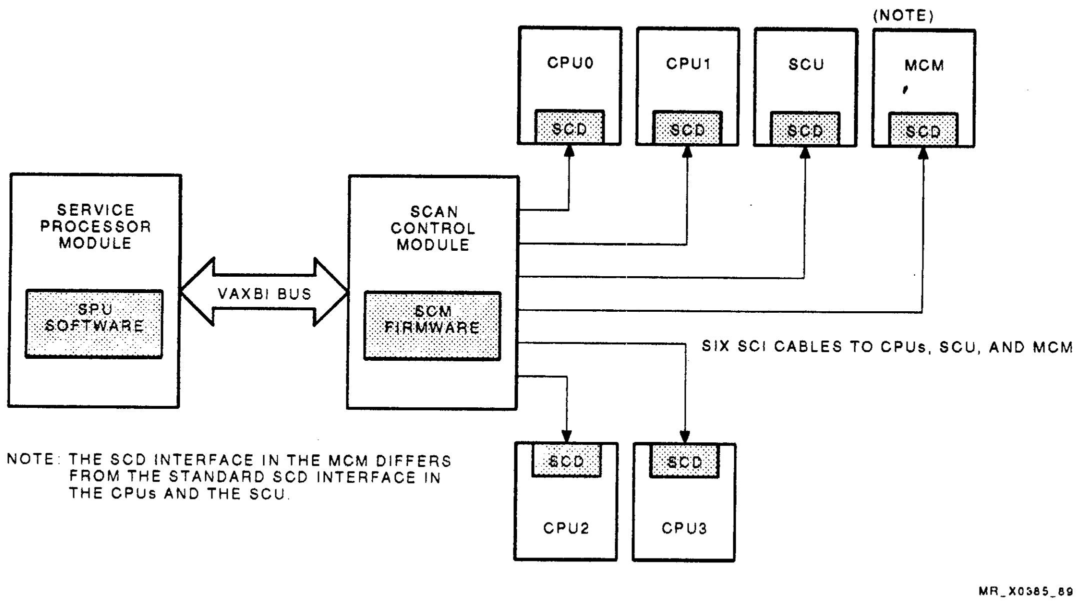

(NOTE)
CPU0
CPU1
SCU
MCM
SCD
SERVICE PROCESSOR MODULE
SPU SOFTWARE
VAXBI BUS
SCAN CONTROL MODULE
SCM FIRMWARE
SIX SCI CABLES TO CPUs, SCU, AND MCM
NOTE: THE SCD INTERFACE IN THE MCM DIFFERS FROM THE STANDARD SCD INTERFACE IN THE CPUs AND THE SCU.
CPU2
CPU3
MR_X0385_89

Figure 5–7 Scan System Overview

5.4.1 SPU Software

The SPU software resides on the service processor module and contains all the programs and data files required to access the scan system during:

System startup and initialization
Error handling and recovery
System testing and diagnosis

The SPU software is also the primary user interface to the scan system. It communicates with the SCM firmware on the VAXBI bus using command and response queues stored in the SPM main memory.

5.4.2 SCM Firmware

The SCM firmware resides in a ROM on the scan control module (SCM). It provides independent control of the scan system latches in the CPUs, SCU, and the MCM through the scan interconnect (SCI). It uses signal and STRAM descriptor tables stored in SCM local RAM to access the scan latches. A local SCM RAM buffers scan ring information to and from the scan latches.

5.4.3 Scan Control Module

The scan control module is a BI adapter specific to the VAX 9000. It contains hardware to provide access to the scan system latches in the CPUs, SCU, and MCM. It is driven by a MicroVAX chip under the immediate control of the resident SCM firmware. The heart of the SCM is the scan control chip (SCC), a custom gate array designed to perform the following functions:

- Read scan latch rings into SCM local RAM from the CPU/SCU/MCM.
- Write scan latch rings from SCM local RAM to the CPU/SCU/MCM.
- Compare ring patterns read from the scan latches with expected patterns stored in SCM local RAM.
- Compare ring patterns read from two or more CPUs (XOR testing) and store results in SCM local RAM.
- Load CPU/SCU STRAMs during system initialization.
- Control operation of the master clock module (MCM) (start, stop, change frequency, burst, and so on).
- Respond to attention signal interrupts (CPU/SCU errors) generated by the scan system.

Scan system operations in the SCM are overlapped with operations occurring in the SPM. For example, the SCM can be retrieving and formatting scan ring information from the CPU while the SPM is processing information retrieved by a previous SCM operation.

5.4.4 SCI

The SCM connects to the scan latch rings in the CPUs, SCU, and MCM through the scan interconnect (SCI). Six SCI cables are connected to the SCM module: one each for the four CPUs, one to the SCU, and one to the MCM. Five of the six cables (CPUs and SCU) are identical and consist of 30 lines (13 differential signal pairs and 4 grounds). The MCM SCI cable differs in that it carries only six differential signal pairs.

During normal operation, each SCI port is independently selected by the SCM and only one port is selected at any one time. The four CPU ports are an exception to this rule. More than one CPU may be enabled at the same time for scanning. In this case, the selected CPU is called the primary CPU. During system initialization, the SCM can scan out the same ring information to all CPUs simultaneously if there are no hardware revision conflicts between CPUs. During CPU XOR testing, two or more CPUs can be scanned at the same time. This permits the comparison of a known "good" CPU with one that is suspected to be "bad."

5.4.5 SCD MCA

The scan distribution (SCD) MCA provides a standard interface in each of the four CPUs and the SCU. It distributes the SCI signals to the MCUs on the CPU and SCU planar modules. In the CPU, the SCD is located in the EBox. It is a logical MCA contained within the RLOG MCA in the INT MCU. A similar interface is located within the DSCT MCA on the CCU MCU in the SCU. The MCM contains an equivalent but simpler interface.

5.4.6 SCI Signal Descriptions

The SCI consists of 13 differential ECL signals, 2 data and 11 control, that initiate and control scan operations between the SCM and the scan latches located in each CPU, the SCU, and the MCM. A hardware register, the SCI control register (SCICR), contained in the SCC chip on the SCM module, provides the primary interface between the firmware and the SCI.

5.5 Scan System Functions

All scan latches in the CPU and SCU are accessible for reading and writing by the SPU through the SCI. Within each MCA, including the clock distribution chips (CDCs), the individual scan latches are serially connected to form uniquely addressable rings.

The number of latches in any ring is variable and depends on MCA type. Each MCA contains a single ring and each CDC contains three rings. To read and write any scan latch, the SCM does the following:

1. Selects a unit (CPU or SCU).
2. Selects 1 of 22 MCUs (maximum).
3. Selects a ring (MCA or CDC).
4. Shifts data into (scan in) or out of (scan out) the selected ring one bit at a time.

Random access to a single scan latch is not possible. To access a single scan latch, the SCM must read and write the entire ring to prevent leaving the CPU or SCU in an undefined state. During a scan read operation, the data scanned in is looped back out into the scan ring to restore the machine state. During a scan write operation, new data is scanned out of the SCM.

Overlapped read and write operations are possible. This allows the SCM to read the current state of the ring while scanning out new data to change the next machine state. In the case of multiple CPUs, the SCM can select the same rings simultaneously in up to four CPUs. During system initialization, this allows the SPU to scan out the same reset pattern to all CPUs at the same time.

The following sections briefly describe some of the basic system-level scan system functions.

5.5.1 Primitive Functions

The primitive scan system functions are ring read and ring write.

5.5.1.1 Ring Read

Ring read scans in the contents of all the scan latches in one or more rings and transfers the data to a specified ring buffer located in the SCM local RAM. It involves the following sequence:

1. Stop the system clocks.
2. Select the MCU(s) and MCA(s) that contain the ring.
3. Shift in the contents of the selected ring and transfer the data to a ring buffer in SCM local RAM. The data is also looped back into the scan latches to restore machine state.

5.5.1.2 Ring Write

Ring write scans out the contents of a specified ring buffer located in the SCM local RAM and transfers the data into the scan latches in one or more MCA rings. It involves the following sequence:

1. Stop the system clocks.
2. Select the MCU(s) and MCA(s) that contain the ring.
3. Shift out the contents of the specified ring buffer and transfer the data into the scan latches in the selected ring(s).

5.5.2 Diagnostic Functions

The primitive ring read and write functions are combined with data comparison functions by the SCM hardware and firmware to form four higher-level system functions:

Scan pattern execute (SPE)
Scan pattern verify (SPV)
CPU XOR testing
Attention handling

5.5.2.1 Scan Pattern Execute

The SPE function is the primary operation used by the SPU-based scan pattern diagnostics to test and isolate hardware faults in the CPU and SCU. It combines simultaneous ring writes and ring reads with a bit-by-bit comparison of the result data scanned back in. The data comparison may be masked. SPE uses four equal-length ring buffers in SCM local RAM that are set up by the SCM firmware and SPU software:

- **Test pattern buffer** — Contains the test data to be shifted out into the scan latches.
- **Expected pattern buffer** — Contains the normal result data that should be shifted out of the scan latches if no hardware faults exist.
- **Mask pattern buffer** — Contains the mask pattern that enables selective comparison of the data scanned in against the expected data.
- **Result pattern buffer** — Contains either the data actually scanned in or the result of the comparison between the expected and result data.

The basic sequence of events during an SPE operation is as follows. Refer to Figure 5–8.

1. Turn off the CPU system clocks.
2. Scan out test pattern n from the test pattern buffer to set up all the scan latches in the CPU.
3. Generate a single pair of system clocks (A_CLK followed by B_CLK).
4. Scan in the result of test pattern n and compare it, bit by bit, with the contents of the expected pattern buffer using the contents of the mask pattern buffer to enable each comparison. At the same time, scan out test pattern n + 1 from the test pattern buffer.
5. Store the result of the comparison in the result pattern buffer. Any bits that failed to compare are stored as a 1 in the corresponding bit position. Note that in Figure 5–8 bits A and B failed to compare, however bit A was masked out and did not show up as a 1 in the result pattern.
6. Set the pattern compare error (PCE) flag if any bit fails to compare.

7. Set the done flag to interrupt the SCM firmware that processes the result.

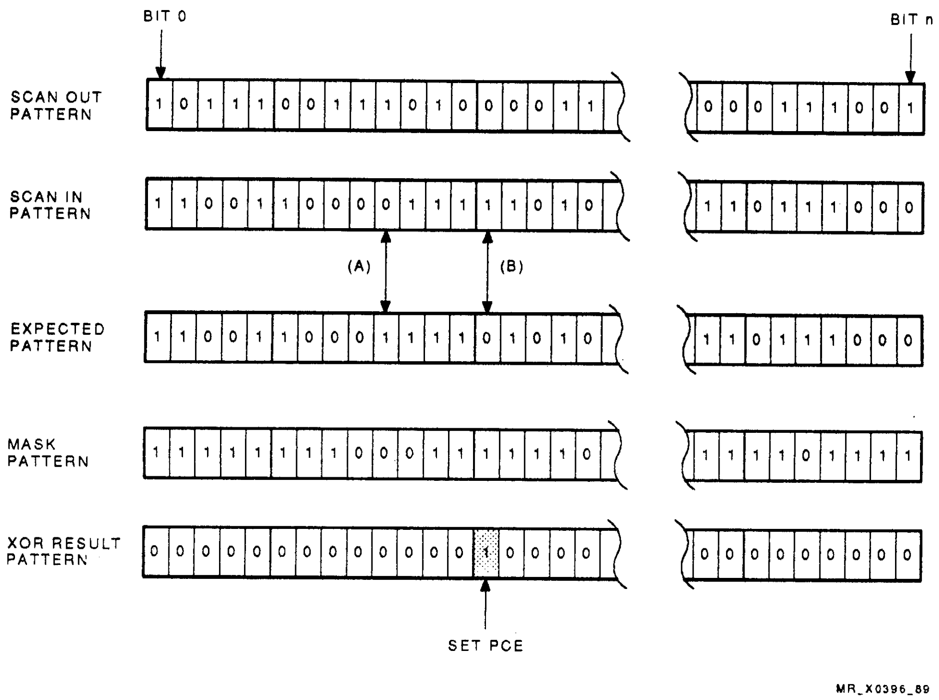


Figure 5–8 Scan Pattern Execute Example

At a normal scan clock rate of 100 ns, a 20,000-bit SPE can be executed in approximately 2 ms using broadcast mode. Since ring writes may be overlapped with ring read and compare, each test pattern requires 2 ms to execute. This requirement would permit 1,000 test patterns in 2 seconds.

5.5.2.2 Scan Pattern Verify

This operation simply verifies that a pattern scanned out can be scanned back in with no errors. The SPV is similar, but faster and simpler than the SPE. The major differences are as follows:

- The test pattern buffer and the expected pattern buffer are the same.
- No system clocks are issued after scanning out the data.
- The result is not stored unless an error is detected.

Basically, an SPV operation consists of the following steps:

1. Turn off the system clock.
2. Scan out the contents of the test pattern buffer to load the scan latches.
3. Scan in the contents of the scan latches and compare it, bit by bit, with the data just scanned out. At the same time, scan out the next test pattern (if any).
4. Set the PCE flag if any bit fails to compare.
5. Set the done flag to interrupt the SCM firmware that processes the result. If an error is detected, the firmware repeats the failing pattern and stores the result of the comparison.

Like the SPE, the SPV can overlap ring read with ring write. So, pattern n + 1 can be scanned out while pattern n is being scanned in. SPV can be used in conjunction with SPE to verify the scan latches themselves and that the connectivity of the scan system is intact.

The SPV function is also used for tracing changes in signal states when microstepping the machine. To achieve signal tracing using SPV, the firmware stores the initial state in the expected pattern buffer and sets up the mask pattern buffer to enable the signals that are to be traced. After each microstep (SYS_CLK), the current state of the machine is scanned in and compared to the expected pattern buffer. Any signals that change state (enabled by the mask pattern buffer bits) set the PCE flag to indicate the change. The results of the SPV can be analyzed to determine which signals changed state.

5.5.2.3 CPU XOR Testing

Since the scan system allows the SCM to access two or more CPUs simultaneously, it is possible to compare the responses from a known "good" CPU with one that is suspected to be "bad" and analyze the differences. When this feature is used, one of the CPUs is designated as the primary CPU and the others are compared to it. For example, assume that CPU1 appears to be faulty and CPU0 is known to be operating normally. XOR testing involves the following sequence:

1. Turn off the CPU0 and CPU1 system clocks.
2. Enable the SCI to CPU0 and CPU1.
3. Scan out a test pattern to initialize the CPU0 and CPU1 scan latches to the same state.
4. Generate a single pair of system clocks to both CPUs.
5. Scan in the contents of the scan latches in both CPUs. Compare the patterns, bit by bit, from both CPUs.
6. Set a CPU compare error (CCE) flag to indicate any errors and store the result pattern.
7. Set the done flag to interrupt the SCM firmware that processes the result.

5.5.2.4 Attention Handling

During operation, the scan system is used to signal SPU intervention in the event of any CPU/SCU-detected errors. Typically, when the system is running, the SCI to all units (CPUs and SCU) is deselected. All CDCs on every MCU are deselected and executing a NOP function. These conditions enable each CDC to assert the SCI_DATA_IN line to signal the SCM if an error is detected in one of the CPUs.

Assertion of the SCI_DATA_IN line sets a flag that interrupts the SCM firmware to request service. Each SCI is assigned a unique flag. Once interrupted, the SCM firmware determines which CPU generated the error and signals the SPU software. The SPU software fault-handling routines then intervene to process and log the error.

5.5.3 Scan Distribution

In the CPUs and the SCU, scan loops are used to connect the scan system to the scan latches. A scan loop consists of clock signals, a scan data line, and scan control signals. The scan loop passes through a certain number of MCUs. Scan control signals permit all or specific MCUs and MCAs in the loop to be selected for testing.

Each CPU contains three scan loops and the SCU contains two. Table 5–1 lists the MCUs in each scan loop.

Table 5–1 Scan Distribution

Loop Number	MCUs
CPU	
Loop 0	FAD, MUL, CTU, DTA, VAP, OPU, XBR, VIC
Loop 1	INT, DST, DTB, CTL, UCS
Loop 2[1]	VML, CDC, VAD
SCU	
Loop 0	CCU, DA0, TAG, DB0
Loop 1[1]	DA1, DB1

[1]The MCUs in these scan loops are optional and configuration dependent.

Figure 5–9 shows the distribution of CPU scan loop 0.

The scan loop in Figure 5–9 contains eight MCUs (00 through 07). When an MCU and MCA are selected, the scan system can perform a read or a write of the scan latches in the MCA. A read operation is performed to determine the state of the scan latches when an error has been detected. A write operation is performed to initialize the MCA to a predetermined state.

The scan data and scan clock lines in Figure 5–9 are routed through the MCUs in the opposite direction of the scan function and the bus select lines. This routing configuration allows connectivity verification between MCUs.

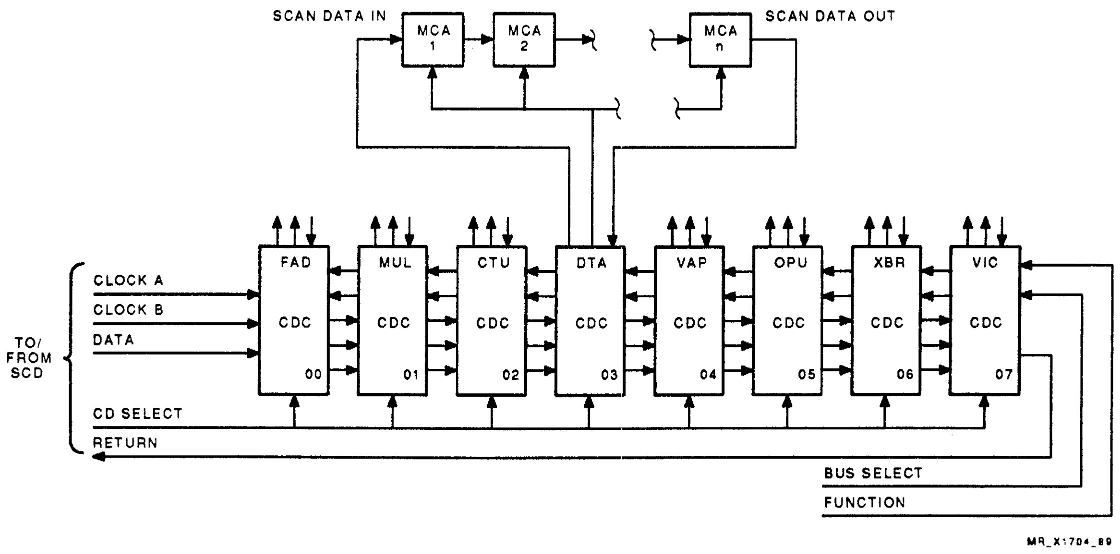


Figure 5–9 CPU Scan Distribution for Loop 0

6 Power and Control Subsystem

This chapter provides an introduction, basic specifications, and an overview of the power system, including power distribution and control.

The various system models have significant differences in their power configurations. Consequently, this chapter uses two block diagrams to demonstrate the differences and to describe the configuration of the VAX 9000 model 200 and 400 systems.

6.1 Power System Introduction

The dc output of the power front end (either an H7390 PFE or H7392 UPC) is distributed to the IOA cabinet. The dc power is then stepped down to a low-voltage dc and distributed to the system logic units.

The power system contains components that monitor environmental and electrical conditions of the system. The conditions include cabinet temperature, air flow, and voltage and current level outputs of the power components. The components that measure and monitor the power and environmental states interface to the SPU. When errors occur, the SPU is notified and corrective action is initiated.

6.1.1 Model 200 Power System

Figure 6–1 shows a block diagram of the model 200 power system. The block diagram includes ac power distribution. The ac distribution and generation of the 280 Vdc shown in Figure 6–1 is provided by the H7390 power front end (PFE).

6.1.1.1 AC Distribution

The H7390 PFE receives input ac from the utility power and provides an ac output to the following components:

Battery backup units (BBUs)
Two service outlets
Bias supply PSA
Power line monitor connector

The H7390 PFE provides a basic set of power monitoring signals to the IOA cabinet. These signals allow the power control system (PCS) to monitor status from the PFE. Status information includes: BUS LO, AC LO, thermal fault, and so on. The PCS also monitors the ground current in the PFE.

The PFE passes and receives power control signals through the signal interface panel (SIP). The SIP distributes these signals to other units of the PCS. Control signals include: power request, power inhibit, total off, and so on.

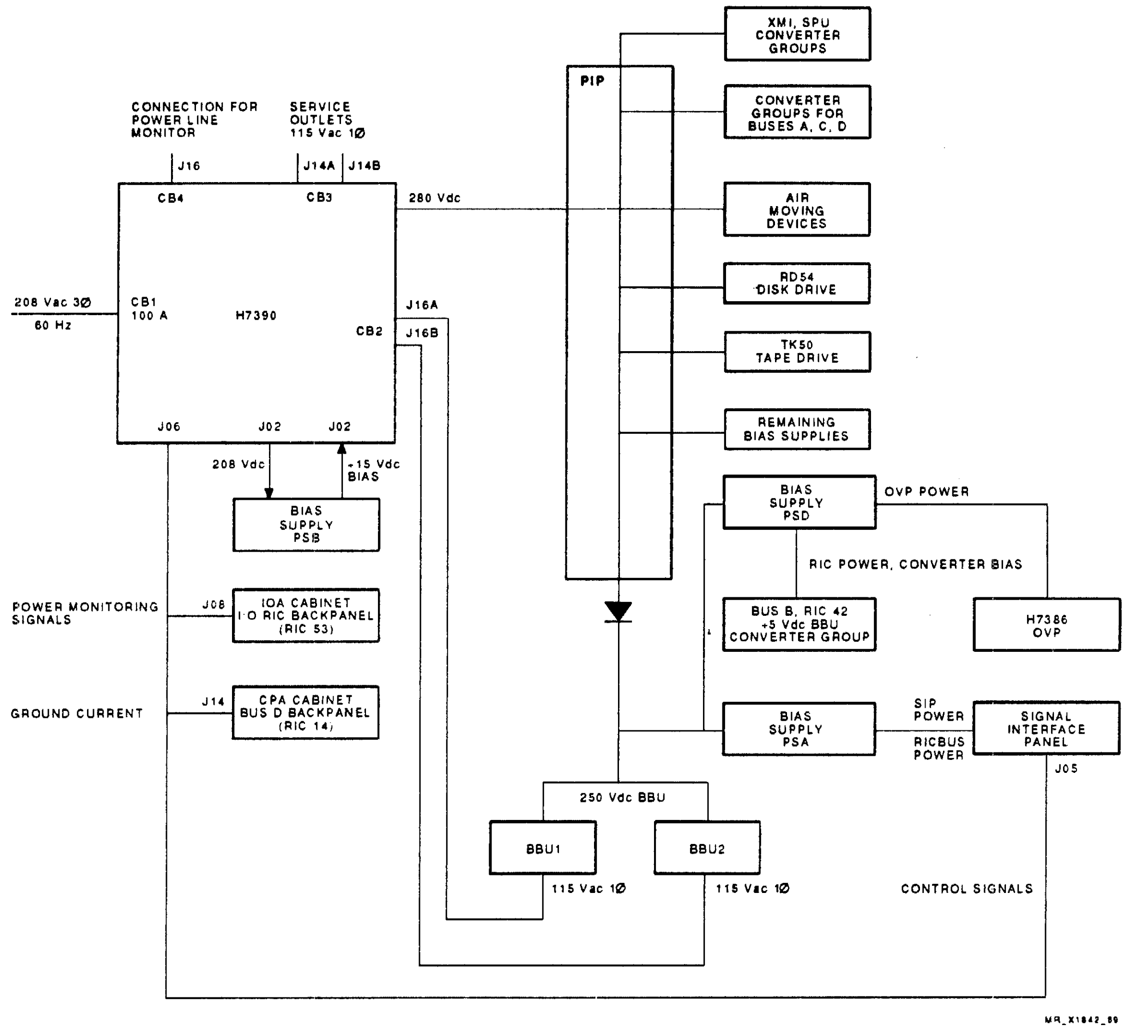


Figure 6–1 Model 200 Power System Block Diagram

6.1.1.2 DC Distribution

The H7390 PFE converts 3-phase ac input to 280 Vdc output. The output of the PFE is then distributed to the power input panel (PIP) in the IOA cabinet. Power is distributed from the PIP to the converter group components (bus A through bus D, SPU, and XMI), to the RD54 disk drive and the TK50 tape drive of the SPU and to the air moving devices.

Bus B receives dc power from the H7390 and the BBUs. The 250 Vdc from the BBUs is sufficient to operate the H7380 converters.

6.1.2 Model 440 Power System

Figure 6–2 shows a block diagram of the power system that supports a model 440 configuration. The ac distribution and generation of the 280 Vdc (Figure 6–2) is provided by two H7392 utility port conditioners (UPCs). However, only one UPC is required for single and dual processor configurations.

The power system for the model 440 differs from the model 210 because it supplies power to more system components, and it provides the ability to shut down one CPU pair, while system operations continue in the other CPU pair.

Two UPCs and associated logic are required to provide the shutdown function. The logic provides the ability to remove power from one UPC, while maintaining power to the common elements of the system.

6.1.2.1 UPC Power Distribution

The UPC receives its input from the utility power and distributes the dc power to similar components as the PFE.

The H7392-0 and H7392-1 output control and monitoring signals to the PCS. H7392-0 provides power monitoring signals to the PCS through the IOA cabinet, and H7392-1 provides the same set of signals to the PCS through the IOB cabinet. Each CPU cabinet (CPA and CPB) receives ground current monitoring signals from an associated UPC.

The power control bus signals that interface between the UPC and SIP are identical to those of the PFE. However, the UPC provides a more extensive set of status and fault conditions to the PCS, for example: thermal fault, input overload, input overvoltage, phase loss, and so on.

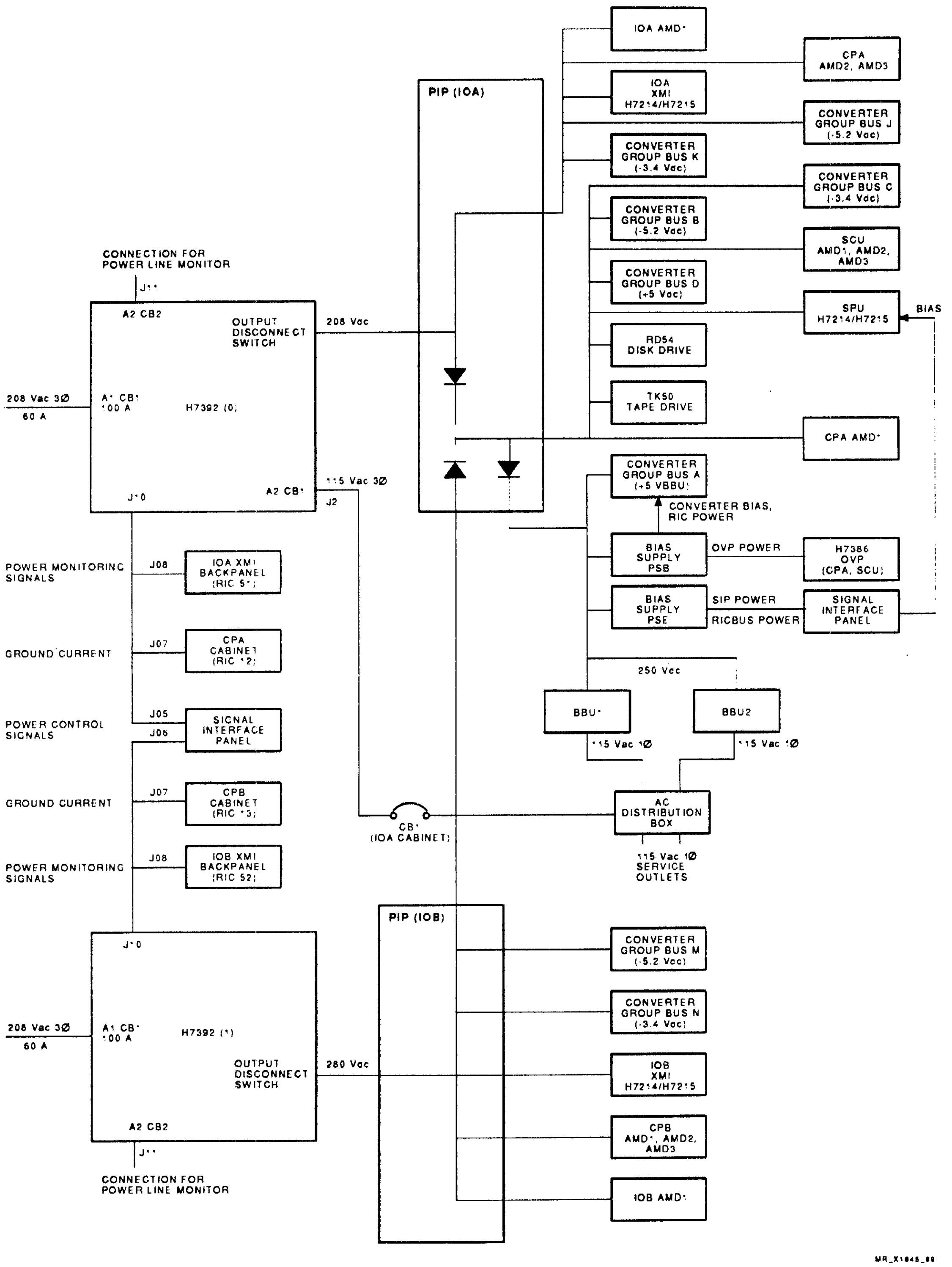


Figure 6–2 Model 440 Power System Block Diagram

6.1.3 Power System Components

Table 6–1 lists the power system components discussed in this chapter and describes their functions.

Table 6–1 Power System Components

Component	Number	Description
Power front end (PFE)	H7390	Located in the model 200 IOA cabinet and connects to the utility power. The PFE generates 280 Vdc for the power system components.
Utility port conditioner (UPC)	H7392	A standalone unit that receives power and generates 280 Vdc for the power system components.
Power input panel (PIP)	70-27242-01	Receives 280 Vdc from H7390 or H7392 and distributes it to the power system components.
Battery backup unit (BBU)	H7231	Provides 250 Vdc to system main memory for 10 minutes during a power failure. The BBU also provides +5 Vdc to the operator control panel (OCP) and the SPU time of year (TOY) clock.
Power and environmental monitor (PEM)	T1060	Interfaces between the PCS and the SPU. The PEM is in the SPU and is responsible for polling and reporting power system conditions to the SPU.
CPU regulator intelligence card (RIC)	H7388	Turns power converters on and off and monitors and controls power converter voltage output. The CPU RIC also interfaces to the PEM and reports converter overcurrent and overvoltage, system air flow, and temperature conditions.
I/O RIC	H7389	Provides the same functions as the CPU RIC, except that it does not control the power converter voltage and current outputs.
Operator control panel (OCP)	54-19030-01	Located on the front of the IOA cabinet. The OCP controls and/or displays power on/off, system restart, SPU access, system fault codes, and CPU and SPU status.
Signal interface panel (SIP)	5419028-0-1	Provides a signal interface to the following: PEM RIC OCP BBU H7392 H7390 Power converters SPU The SIP also contains logic that enables the BBUs at power failure and generates the clocks for the power converters.
Bias supply	H7382	Operates on 280 Vdc and generates bias voltages for the H7390 and the power converters. Bias supply provides a reference voltage for temperature measurement and power for the RICs, RIC buses, OCP, overvoltage protection (OVP) modules, and Ethernet transceivers.

Table 6–1 (Cont.) Power System Components

Component	Number	Description
Power converters	H7380	Steps down the 280 Vdc to produce voltages between 3.3 Vdc and 5.5 Vdc. These outputs are used to produce the +5 Vdc, -5.2 Vdc, and -3.4 Vdc used by the CPU, system control unit (SCU), and memory. Power converter voltage levels are determined by control voltages supplied by the RICs.
	H7214/H7215	Provides power to the SPU backpanel, and each XMI or BI expansion cabinet.
Overvoltage protection (OVP) module	H7386	Monitors the +5 Vdc, -3.4 Vdc, and -5.2 Vdc buses in the CPU and SCU cabinets for overvoltage conditions. The OVP module notifies a RIC of an overvoltage condition. When overvoltage is detected on the -3.2 Vdc bus, the OVP module fires an SCR that shorts the -3.4 Vdc bus to ground, which protects the ECL logic.

6.1.4 H7392 UPC Overview

The UPC provides harmonic-free rectification, high EMI suppression, and an improved power factor. It can be configured in two variations: variation 1 operates from a nominal 416 Vrms, 50 Hz source, and variation 2 operates from a nominal 208 Vrms, 60 Hz source. The 280 Vdc regulated output is maintained by using a pulse width modulation technique.

The UPC is housed in a custom designed cabinet (similar to a VAX-11/780 cabinet). The UPC components (that is, assemblies and modules) are assembled into two bays. Printed circuit boards (PCs) are also mounted on the assemblies as well as on panels in the bays.

6.1.4.1 Power Input Control and Distribution

The 3-phase ac input power is applied to the power input control and distribution block (Figure 6–3). This functional unit provides the UPC on/off control through a high capacity circuit breaker. The unit distributes ac voltage to the main power transformer, convenience outlets, BBUs, and low-voltage power supply used for the control and interface circuits.

The A1 module provides input EMI filtering. It also contains a current transformer to monitor the UPC and kernel ground current.

The main power transformer consists of a delta-wound transformer primary. The secondary windings convert the 3-phase ac input to three, single-phase outputs (phases A, B, and C). Auxiliary secondary windings on T1 provide 3-phase output power through A2 to the kernel BBUs and UPC fan assemblies.

6.1.4.2 AC Filter and Rectification

The ac filter and rectification functional units represent the ac input filter module A4, and the three diode rectifier modules: A5 (ØA), A6 (ØB), and A7 (ØC).

The A4 module provides the ac filter networks for each output phase of the power transformer. The filter networks prevent input power disturbances from affecting UPC operation by attenuating input line EMI and transients. The networks also prevent UPC power converter switching (PWM ripple) from being coupled back into the utility.

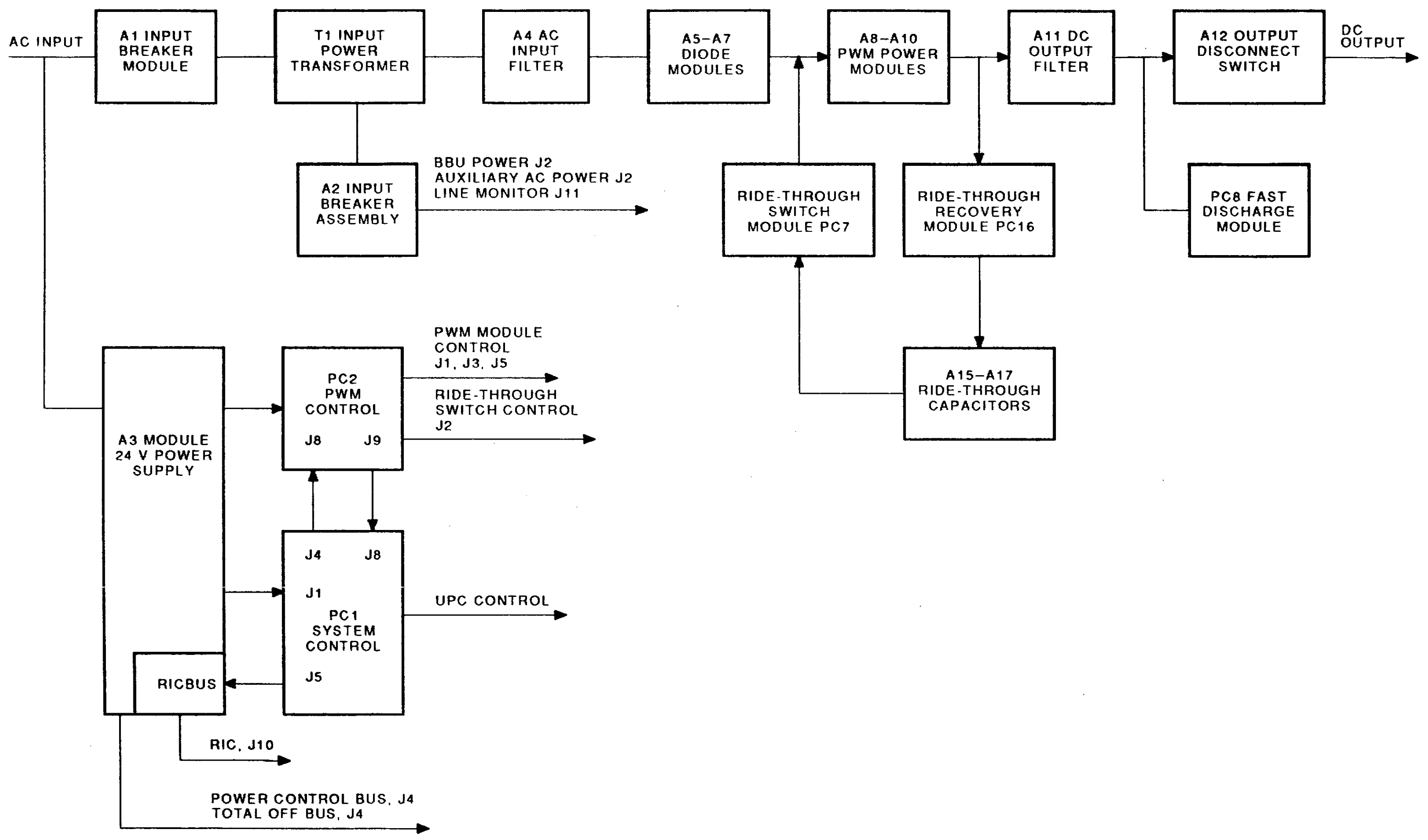


Figure 6–3 Basic Functional Block Diagram

Each diode rectifier module (A5 through A7) contains a full wave rectifier and an additional filter. The rectified dc output current of each phase is passed through an associated current sensor. The sensor measures the rectifier output current to control the UPC input current, and to provide input power overload protection.

6.1.4.3 Pulse Width Modulator

Each phase of a diode rectifier module output is applied to a pulse width modulator (PWM) module. The functional unit block represents PWM modules A8 (ØA), A9 (ØB), and A10 (ØC). The PWM module output provides a regulated output over a wide range of ac input voltage and dc output voltage.

During normal UPC operation, the PWM converters are programmed to draw ac input current. However, in ride-through mode, the converters draw dc current from the ride-through capacitors.

The PWM modules perform two major functions: wave shaping and amplitude control.

- Wave shaping controls the input current to conform to the wave shape of the incoming voltage. This eliminates rectification harmonics.
- Amplitude control controls the amplitude of the incoming current based on the output dc voltage. This provides output regulation.

6.1.4.4 Ride-Through Functional Unit

The ride-through unit and its control logic provides the capability to maintain the dc output during a momentary interruption of input power up to 100 ms. The ride-through time is long enough to overcome approximately 95% of the power interrupt problems the UPC will encounter. The ride-through block represents several banks of high-capacitance electrolytic capacitors contained in modules A15, A16, and A17, which are charged to the full dc output.

With an input power interruption, the ride-through control logic — contained in A13 on PC7, PC8, and PC16 — initiates the discharge of the capacitors to maintain the dc output. The discharge path is through the PWM module (which provides regulation during the ride-through period) through the output filter, and into the load.

The ride-through circuit modules control discharge and recharge of the ride-through capacitors. When the UPC is initially set to on or power is restored after a power interrupt, the ride-through recovery circuit provides a fast capacitor charging rate. With the UPC set to off, the fast discharge circuit provides a discharge path for the ride-through and load capacitance.

6.1.4.5 Output Filter and Disconnect Switch

The functional unit block represents the final dc output filter (A11 module). In addition, an optional output disconnect switch may be installed after the filter on a second UPC of a quad processor configuration.

The output filter provides the common phase connection point, and the final dc filter network that removes residual ripple. The filter module also contains a current sensor to monitor the output current. The sensor output is coupled into the output overload monitor circuit on the system control PC.

6.1.4.6 Low Voltage Power Supply

AC power from the A1 module is distributed to the low voltage dc power supply (A3 module). The supply provides a nominal, unregulated 24-V output, which provides the required dc voltages for the UPC control and logic circuitry. The A3 module also provides the regulator intelligence card (RIC) bus interface. The RIC interface is isolated from the PCS and routes status and fault functions from the UPC to the PCS.

6.1.4.7 System Control

The majority of UPC control, status, and fault monitoring is provided by the system control PC (PC1). PC1 provides the UPC on/off functions, and the ride-through initiation. It also provides the following general functions:

- Current monitoring, input current control, and output regulation
- Operational status monitoring and related LED indicators
- Fault and error detection and related LED fault indicators
- Power-up, power-down, and sequencing functions

PC1 also contains an isolated power supply that provides the regulated +12 Vdc and -12 Vdc for the control and monitor circuits.

6.1.5 H7390 Power Front End Overview

The PFE components (assemblies and modules) are integrated into a single cabinet component and located in the bottom of the IOA cabinet. The PFE is highly modularized through extensive use of connectors on major components and modules.

The majority of the PFE components are contained in four assemblies: power entry, ac line filter, controller module, and capacitor bank assemblies. Each assembly is a major functional unit (Figure 6–4).

6.1.5.1 Power Entry Assembly

The power entry assembly is the ac power input entry point. Power is distributed from the assembly through the main circuit (CB1), which serves as the main power switch. CB1 has the capability to be tripped off in the event of a severe internal failure, or from the system through the total off function.

Input ground current is monitored from the assembly. The ground current is routed through the controller module assembly to the RICBUS. The assembly also supports a pair of duplex convenience outlets, as well as a power line monitor connector to monitor the ac line input.

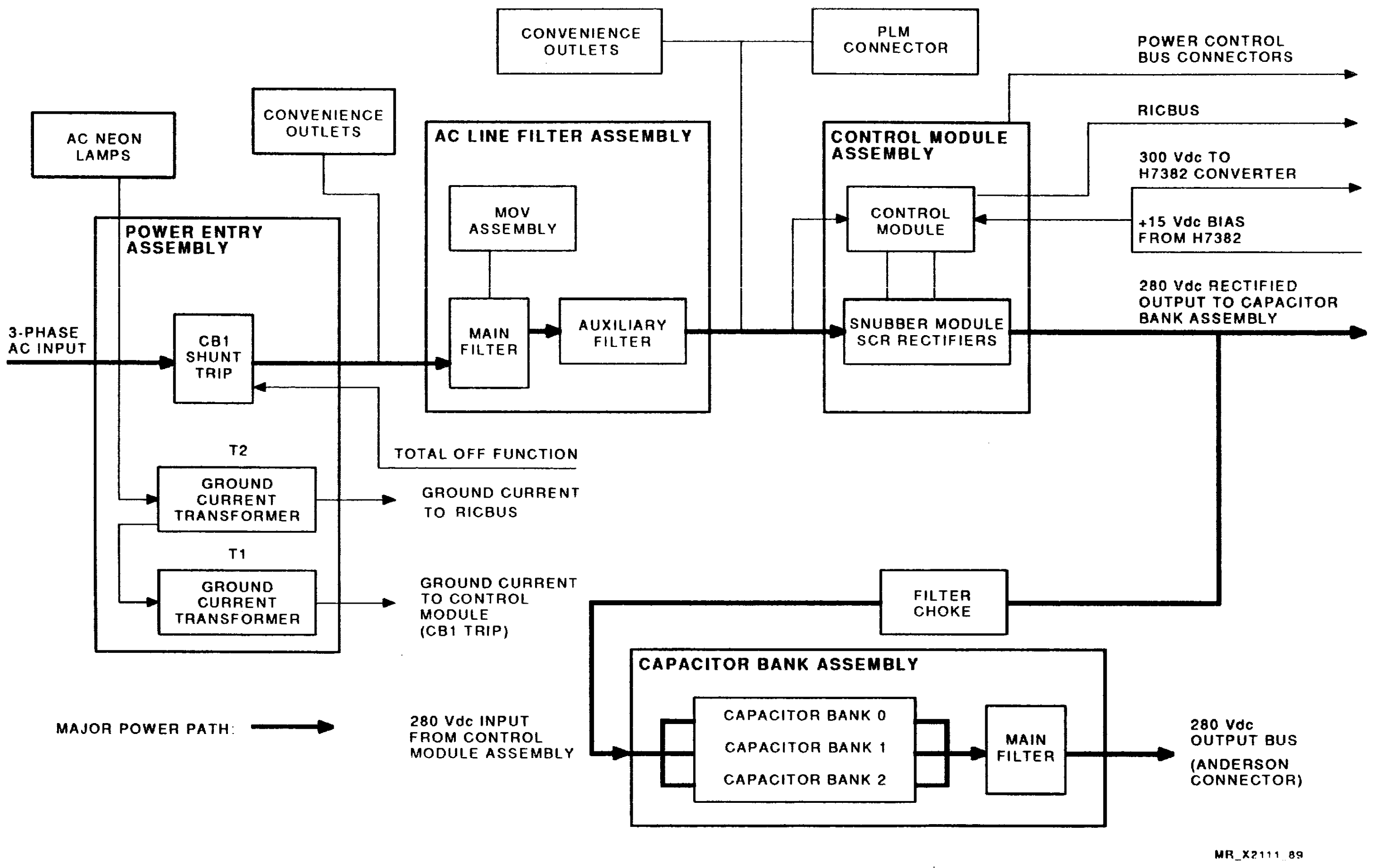
6.1.5.2 AC Line Filter Assembly

The ac line filter is *not* an FRU assembly. The filter is used to suppress input power EMI as well as to suppress PFE noise feedback into the utility. The assembly includes a set of metal oxide varistors (MOVs) that provides overvoltage and transient protection.

6.1.5.3 Controller Module Assembly

The controller module assembly contains the SCRs, which provide the full wave rectifier for the main dc output, and phase-up (power-up). The PFE control module is also contained in the assembly. The module provides the required control and interfacing functions, for example: SCR control logic for rectification and phase-up, dc output bus monitoring, and so on. The assembly also contains the cooling fan that provides cooling air for the entire PFE.

Figure 6–4 Basic PFE Block Diagram



6.1.5.4 Capacitor Bank Assembly

The capacitor bank assembly provides the primary PFE energy storage required for the ride-through and hold-up times. These times allow the PFE to maintain its output in the event of a power interrupt or sag, and to provide an orderly system shutdown. The assembly consists of three banks of five capacitors each, and it is monitored by a set of fault indicators. In addition, the assembly also contains the final dc filter network.

6.1.6 DC Power Components

The dc power components of the VAX 9000 receive 280 Vdc from the PFE or UPC and convert it to the regulated voltages used by the individual logic elements of the system. The dc power system is comprised of power converters, bias supplies, OVP modules, dc/dc converters, and control logic.

The dc power components are configured into a number of converter groups or buses. (See Figure 6–1.) The bus A through bus D converter groups (H7380 converter groups) supply power to the CPU and SCU cabinets. The SPU and XMI converter groups supply power to the SCU and XMI backpanels.

The number of converter groups in a system is configuration dependent. That is, a model 440 requires more H7380 converter groups than a model 210 system because of the power requirements for the multiple CPUs. Each XMI converter group can supply power to one XMI backpanel. Therefore, additional XMI backpanels require additional XMI converter groups.

6.1.6.1 H7380 Converter Group

Figure 6–5 shows a block diagram of the H7380 converter group.

The H7380 converter group consists of the following:

Bias supply
OVP module
CPU RIC
H7380 power converters

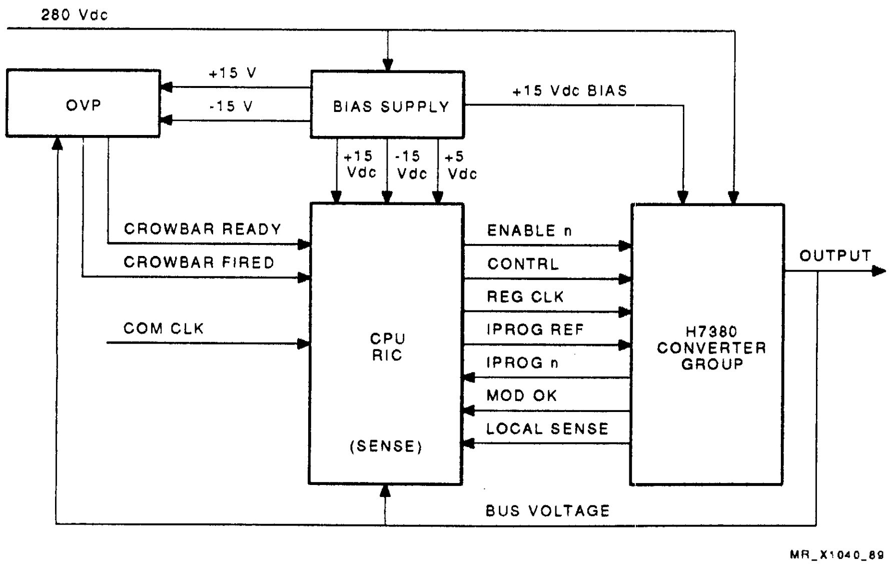


Figure 6–5 H7380 Converter Group

The bias supply of the H7380 converter group operates at 280 Vdc and provides the following outputs:

+15 Vdc to the H7380 converter control logic
±15 and +5 Vdc to the CPU RIC
+15 and +5 Vdc to the I/O RIC

The +15 Vdc provided to the H7380 converter powers the control circuitry of the converter and is ORed with the +15 Vdc output of the converter. This allows the converter to power the control logic after it becomes operational.

The H7380 converters receive 280 Vdc and output a dc voltage specified by the CPU RIC of the converter group (+5, -5.2, or -3.4 Vdc). The CPU RIC also enables and disables the operation of the converter, and it provides the control for voltage margining. The PEM directs the RIC to enable the converter. The RIC can also disable the operation of the converter if errors occur.

The OVP module monitors the voltage outputs of the converters and informs the CPU RIC if an overvoltage condition exists. The RIC shuts down the converters when overvoltage conditions exist.

6.1.6.2 XMI Converter Group

The XMI converter group consists of a bias supply, I/O RIC, and H7214 and H7215 power converters. This configuration operates in a similar manner to the H7380 converter group. The bias supply and the converters receive 280 Vdc, and then the bias supply powers the I/O RIC and provides an initial voltage to the converters. The I/O RIC controls the startup and output voltages of the converters. The differences in this configuration are the voltage outputs and the absence of an OVP module.

6.1.6.3 SPU Converter Group

The SPU converter group consists of a bias supply and the H7214 and H7215 power converters. The SIP and power switch for the BI backpanel interact with the SPU converter group.

The bias supply and power converters operate on 280 Vdc. The bias supply provides the initial voltage to the converters through the SIP. Control for the regulators is supplied directly from the PEM through the SIP.

Power-up on the SPU converters is disabled when the power switch to the BI backpanel is open.

6.2 Power Control System

The PCS provides a control and monitor interface between the SPU and the power system. Its primary functions are as follows:

- Monitoring the status of the power system, cooling system, and local environment
- Informing the SPU of power and environmental fault conditions
- Controlling power system power-up, power-down, and voltage margining
- Powerfail handling (AC LO and DC LO)
- Battery backup operation
- Controlling the power control bus and total off bus
- Controlling the OCP

6.2.1 PCS Hardware

The PCS continuously monitors the state of the cabinet environment and the power system, including the UPC and PFE. Its major hardware components are shown in Figure 6–6.

Under normal operating conditions, the PCS executes commands it receives from the SPU and informs the SPU of faults that occur in the system. When the PCS detects a fault condition, it informs the SPU by issuing an exception report. In some cases, a normal or emergency power-down sequence is initiated.

In a quad CPU system, the PCS supports the power-down of one CPU pair, while the other CPU pair continues to operate. This is performed through a PCS software command that trips the correct circuit breaker. Manual resetting of the circuit breaker is required to restore power.

The PCS contains a number of H7388 and H7389 microprocessor-based RICs distributed throughout the power system under the control of the PEM. Information is exchanged between the RICs and the PEM using the serial data link of the RICBUS.

The status of a power module, thermistor, or sensor can be determined locally by the RIC and relayed to the PEM over the RICBUS as a response to a command or through an interrupt. The PEM can command a RIC to power up or power down its associated modules, and also margin the output voltage.

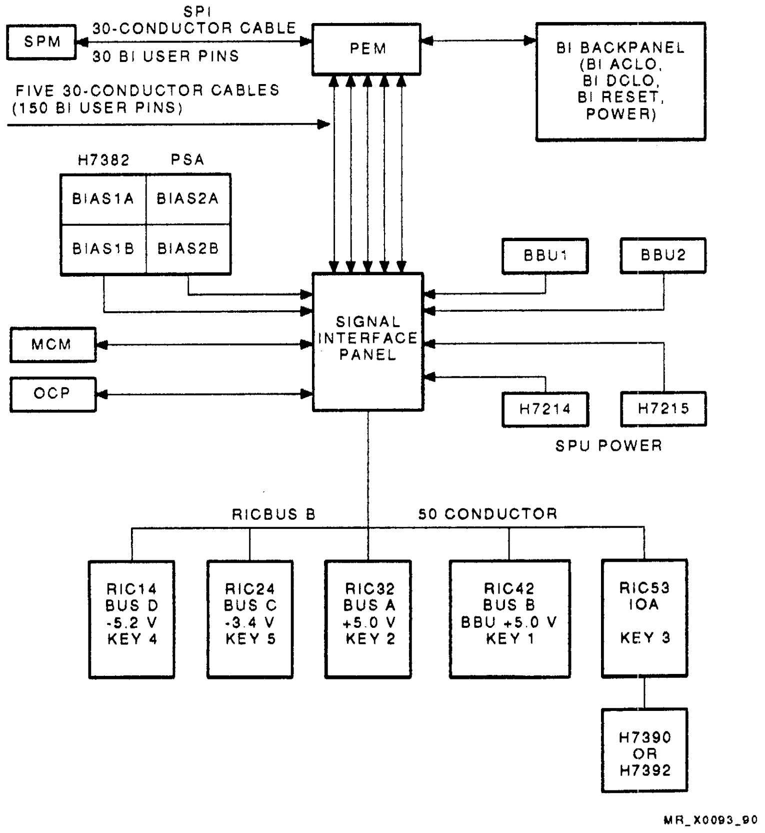


Figure 6–6 PCS Hardware Block Diagram

The following list contains the general PCS measurements. In all cases, parameter measurements that fall outside the acceptable limits cause an exception report to be issued. Cases that may initiate a power-down are identified.

- H7380 regulator bus voltages
- Cabinet air temperature (Fault may initiate a power-down.)
- Ground current

The following list contains the monitored parameters. In all cases, faults cause an exception report to be issued. Cases that may initiate a power-down are identified.

- H7380 regulator GROUP LO status signals
- I/O regulator MOD (module) OK status signals
- Cabinet air flow (Fault may initiate a power-down.)
- Overvoltage conditions
- BBU status signals

The PCS monitors the status and fault signals for the UPC and PFE. Depending on the fault severity, the PCS may initiate a normal or emergency power-down.

The PCS performs self-tests during power-up to verify PCS integrity and then indicates which modules failed the self-tests. The PCS runs and maintains the power and environmental systems without SPU intervention. However, the PCS does not initiate a system power-up unless commanded by the SPU.

The PCS controls the diagnostic display (DD) on the operator control panel (OCP), and the DD indicates the cause of an emergency shutdown (ESD). On startup, the last error written to the status and fault display is read by the PEM and is available to the SPU for entry into the error log before the PEM overwrites the display contents. The PCS also supports an emergency off button on the rear of the IOA cabinet. Pressing the button trips the main system circuit breaker and disconnects the utility input power.

6.2.1.1 Power and Environmental Monitor

The main functions of the PEM are to monitor and control the power system and system environment and to provide a communications path between the PCS and the SPU. Power system control is performed through commands from the SPU and internal PEM software.

The PEM reports out-of-limit conditions or status changes to the SPU on an exception basis. The SPU can read the state of the system at any time through commands to the PEM. The PEM also controls the RICBUS power system communications bus.

The PEM resides in the SPU BI backpanel, and it contains a custom 8051-driven module for control of the RICBUS, system interface logic, and interfacing logic for the II32 bus. The PEM does not communicate over the BI bus but connects to the II32 bus of the SPM through 30 of the 180 BI user pins. Interface signals to the power system are accommodated by the other 150 BI user pins.

6.2.1.2 Signal Interface Panel

The SIP acts as a focal point for signal cables coming from various parts of the power system. All signal lines are concentrated into one cable assembly that attaches to the user pins on the BI backpanel for interface to the PEM.

The SIP provides the logic required to support BBU operation, total off, SPU power supply control, and SPU power-down. The SIP connectors concentrate on the following system components:

PEM module
OCP
SPU power supplies
Bias power supplies
CPU and I/O RICs
Master clock module (MCM)
BBUs
Power interface panel
System total off shunt switch

6.2.1.3 RICBUS Configuration

The RICBUS provides communication between the PEM and RICs. However, there is little communication between RICs. The RICBUS includes a single-wire serial data bus that uses a CSMA/CD scheme (carrier sense multiple access with collision detection) referred to as XXNET. Each node (PEM or RICs) on the network uses a loopback scheme to receive simultaneously its own transmissions to ensure that the data was not corrupted by a collision. The XXNET protocol has provisions for recovering from a data collision.

Although logically a single bus, the RICBUS contains three cables originating at the SIP and servicing separate cabinets or groups of cabinets. Some signals are common to all cables, while other signals appear only on particular cables. However, no single cable is required to carry all signals. This partitioning of signals is required to allow one dual CPU in a quad CPU configuration to be powered down, while the other dual CPU continues to operate.

Communication over the RICBUS is in one of the following two forms:

- Commands and command responses between the PEM and a RIC
- Exception messages that are sent asynchronously by a RIC to the PEM

Commands to a RIC can be initiated by the PEM firmware or the SPU commands to the PEM.

6.2.2 PCS Diagnostic Features

The PCS diagnostics consist of startup self-tests (SSTs) and ROM-based diagnostics (RBDs) that are run under the SPU RBDs supervisor. The SSTs verify the integrity of the PCS and the RBDs aid in diagnosing and locating faulty PCS modules.

Each RIC contains a diagnostic summary register. The register is available to the PEM through the RICBUS. This register contains the pass or fail results of the SSTs. In some cases, it is possible for a RIC that fails an SST to communicate the SST to the PEM.

The PEM has two yellow LEDs, one is on the top of the module and the other is on the front of the module. These LEDs are turned on by the PEM when it completes its self-tests. If these LEDs are not on within 10 seconds after power-up, the PEM is faulty.

The PEM has a diagnostic summary register that can be read directly by the SPU.

6.2.3 Operator Control Panel

In the VAX 9000, the OCP is located on the front of the IOA cabinet. The OCP displays and controls the following functions:

Power on/off
System restart
SPU access
System fault codes
CPU and SPU status

Figure 6–7 shows the layout of the OCP.

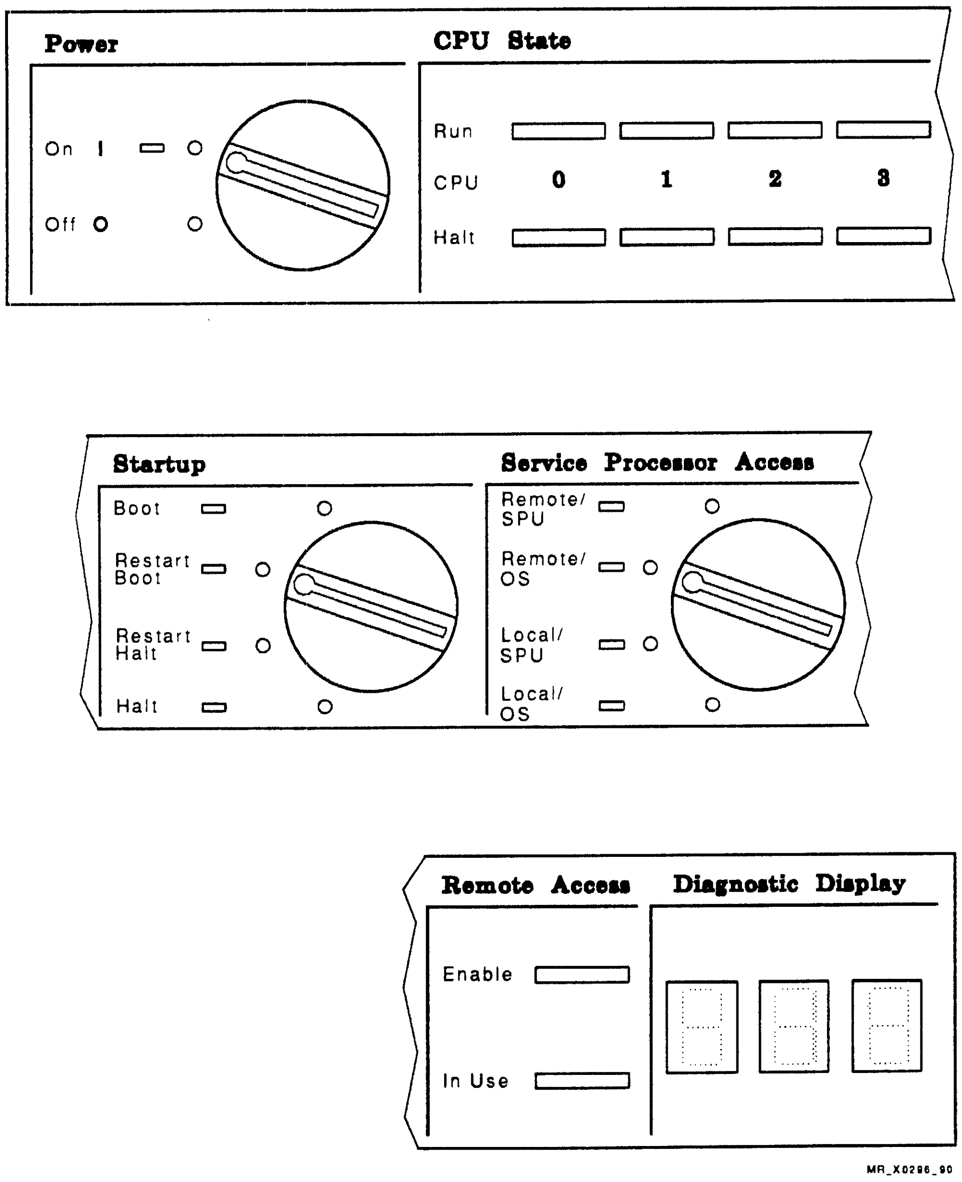


Figure 6–7 OCP Layout

6.2.3.1 OCP Keyswitches

The three OCP keyswitches (Power, Startup, and Service Processor Access) control the following functions:

Power on/off
System startup action
Service processor access

The keyswitch functions are described in the following sections. Included in the descriptions are the BBU test switch and the system total off shunt switch. Although these switches are not physically part of the OCP, they are either logically part of, or are monitored by, the OCP.

6.2.3.1.1 Power Keyswitch

The Power keyswitch is a 3-pole, 2-position rotary switch that controls system power. The first pole controls the power control bus to apply power to the system. The second pole controls the BBU interlock. The third pole is used on the OCP to allow the PEM module to monitor the keyswitch position.

6.2.3.1.2 Startup Keyswitch

The Startup keyswitch controls the startup action taken by the system when power is applied to the system. The keyswitch is a single-pole, 4-position rotary switch that is monitored by the PEM. The keyswitch positions are described in Table 6–2.

Table 6–2 Startup Keyswitch Positions

Position	Action
Boot	During power-up, the system attempts a boot operation. If the operation fails, the system enters console I/O mode.
Restart Boot	During power-up, the system attempts a restart operation. If the restart fails, the system attempts a boot operation. If the boot operation fails, the system enters console I/O mode.
Restart Halt	During power-up, the system attempts a restart operation. If the restart fails, the system enters console I/O mode.
Halt	During power-up, the system initializes the processor set and enters console I/O mode.

7 I/O Subsystem

This chapter provides a functional overview of the I/O system hardware components. Also included is a summary of typical I/O configurations, configuration rules, and supported devices, adapters, and controllers.

7.1 Introduction

The I/O system is based on the XMI bus. The I/O system can be configured with up to four I/O channels that interface to dedicated ports in the ICU (of the SCU). Each of the channels includes the adapters installed on the XMI bus and the following three interface components:

- **XMI bus** — This bus is a pended, synchronous bus with centralized arbitration. The XMI bus is contained in a card cage with a 14-slot backplane. Each XMI bus contains a clock/arbiter module (CCARD) that contains arbitration logic and clock generation logic.
- **XJA adapter** — The XJA is the interfacing adapter between the SCU and the XMI bus. The XJA adapter is implemented in a single logic module that resides in the XMI card cage.
- **JXDI interface** — The JXDI is a bus that interfaces the ICU with the XJA adapter. The JXDI bus consists of four 30-line cables that connect the ICU to the XJA.

Figure 7–1 shows a simplified block diagram of the I/O system.

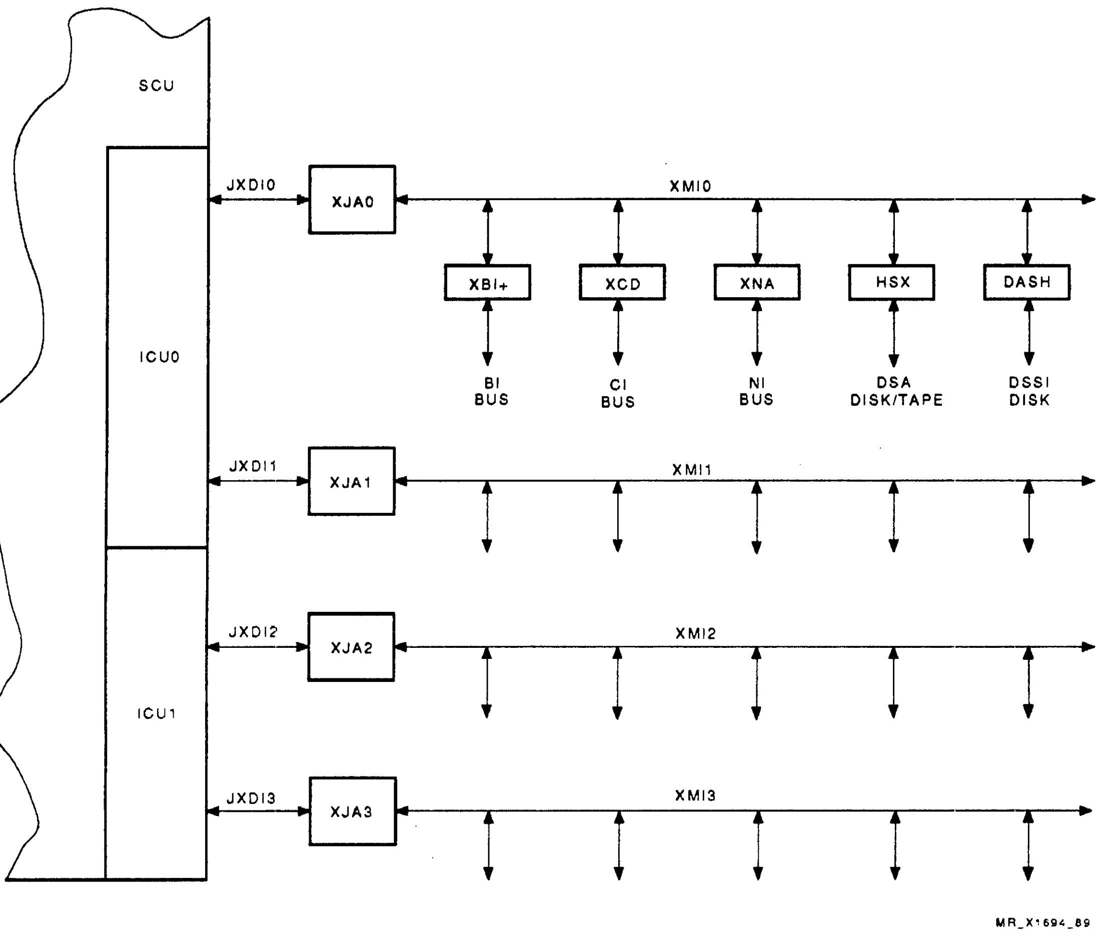


Figure 7–1 I/O System Block Diagram

The XMI bus receives and transmits data packets that have a data length of 64 bits per cycle and are clocked at a 64-ns rate. The JXDI bus receives and transmits data packets that have a data length of 16 bits per cycle and are clocked at a 16-ns rate. The XJA adapter contains the logic that performs the format, timing, and data length changes for data transactions between the two buses. Four types of I/O transactions occur within the XJA:

- **DMA (direct memory access)** — This transaction is a read or write of main memory by an XMI device.
- **CPU** — This transaction is a read or write of an I/O register by the CPU. The I/O register may be in an XMI device or in the XJA.
- **Interrupt** — This transaction may be initiated by the XJA or by an XMI device. An interrupt initiated by the XJA is transferred over the JXDI to the system CPU for processing. An interrupt initiated by an XMI device is received by the XJA and passed on to the system CPU over the JXDI.
- **AOST (add-on self-test)** — This transaction consists of a test packet injected into the XJA by the AOST logic. The test packet is processed as a CPU transaction and the results are checked in the AOST logic.

7.1.1 XMI Card Cage

The XMI card cage contains a 14-slot backplane. Two of the backplane slots are dedicated to housing the clock/arbiter module (CCARD) and the XJA adapter modules. Figure 7–2 shows the XMI card cage.

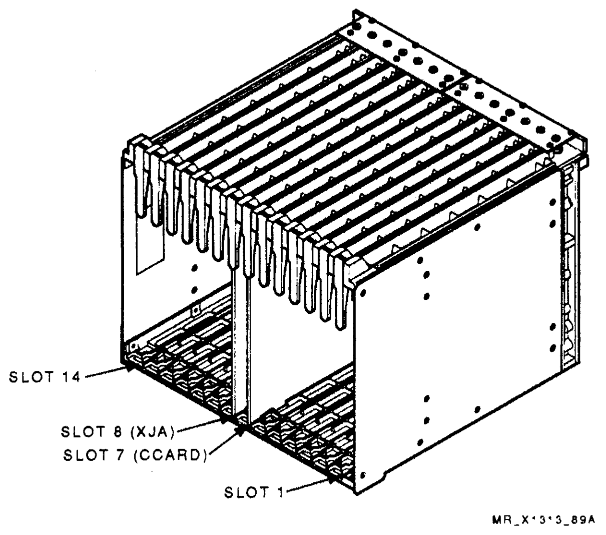


Figure 7–2 XMI Card Cage

The CCARD module contains the XMI arbitration and clock generation logic and is mounted in slot 7 of the backplane. The central position provides an even radial distribution of the XMI signals, which minimizes clock skew between the adapters and improves signal integrity. The XJA adapter is mounted in slot 8.

XMI configuration rules require that the first XMI adapter installed be placed in either slot 1 or 14. Other adapters should be installed in alternate slots, for example: 2, 13, 3, 12, and so on. Adapters in the higher numbered slots have a higher arbitration priority than those in the lower numbered slots.

Each slot of the XMI card cage has a 4-bit, hardwired, node ID number that identifies the slot and the adapter in the slot. XMI adapters match the node ID against selected address bits to determine if an XMI transaction is directed to their node.

7.1.2 XMI Node Adapters

The adapters supported on the XMI bus are listed in Table 7–1 along with the device to which they interface.

Table 7–1 XMI Node Adapters and Interfaced Devices

Interfaced Device	Adapter	Mnemonic	Adapter Module(s)
KFMSA	DSSI[1] disk interface	DASH	T2036
DEMNA	XMI-to-NI adapter	XNA	T2020
CIXCD	XMI-to-CI adapter	XCD	T2080
KDM70	Local DSA disk/tape interface	HSX	T2022 and T2023
DWMJA	XMI-to-JBox adapter	XJA	T1061
DWMBB	XMI-to-BI adapter	XBI+	T2018 and T1043

[1]Digital storage system interconnect

A summary of the physical and address space limitations of the XMI node adapters is given in Table 7–2.

Table 7–2 XMI Node Adapter Limitations

Limitation	Reason
Maximum of 12 adapters per card cage	Only 12 physical slots are available.
Maximum of 8 XBI+ adapters per XMI card cage	A maximum of 8 BI units is accessible per XMI cabinet. Also, XMI protocol limits BI adapters to 8 per XMI bus.
Maximum of 14 XBI+ adapters per system	System I/O space has room for only 14 BI windows.

7.1.3 XMI Configurations

The model 200 systems configurations are limited to one XMI channel. The XMI card cage is located in the IOA cabinet and is connected to the SCU with the JXDI cables.

The model 400 systems configurations may have up to four XMI channels. The model 410 and model 420 systems are configured with two XMI channels. The model 430 and model 440 systems are configured with four XMI channels.

The model 410 and model 420 systems contain only the IOA, CPA, and SCU cabinets (and possibly BI expander cabinets). In these two configurations, the two XMI card cages are resident in the IOA cabinet.

The model 430 and model 440 systems include additional CPU and I/O cabinets. In these configurations, the additional XMI card cages are resident in the IOB cabinet.

7.2 XJA Adapter Functional Overview

The XJA is logically divided into two functional units, an ECL data path interfaced to the JXDI and a CMOS data path that interfaces to the XMI (XDC in Figure 7–3). These two data paths are connected and communicate over the CBI bus.

The logic that comprises the ECL data path consists of two identical data path gate arrays (XDE0 and XDE1) and control logic (XCE). The XDE0 and XDE1 data paths transfer data packets between the JXDI and the XDC under control of the XCE. The XDC array also implements the main XJA control functions and interfaces to the XCLOCK chip and the XCE.

Data to be transmitted to the JXDI is delivered by the CBI in longword format. The data is split and delivered in word format to the two XDE gate arrays. The XDEs split the data words into byte format and deliver to the JXDI.

The XJA data path CMOS gate array (XDC) implements all of the required data path interface functions to the XLATCH chips. The XDC implements a 3-hexword DMA write buffer, four XMI receive command/address (C/A) buffers, two hexword DMA return read data buffers, a single XMI transmit C/A buffer, and the XMI required and XJA-specific registers.

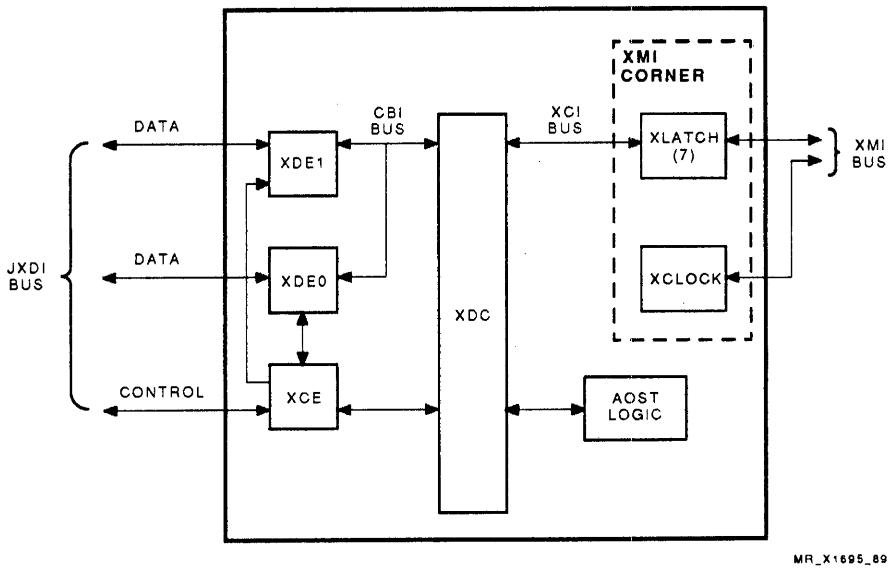


Figure 7–3 XJA Block Diagram

The following are located in the XDC array. The add-on self-test logic, however, interfaces with the XDC array:

- **XMI receive logic (XRC)** — The XRC receives XMI transaction cycle, checks XMI parity, and performs XMI protocol checking to ensure bus integrity. The XRC generates the XMI CNF code for XMI commands that select the XJA. If the XJA is selected, the XRC initiates the receive control machine (RCM).
- **Receive register file (RRF)** — The RRF contains four XMI command/address buffers and three hexword XMI write data buffers. The XRC loads the RRF when it receives valid XMI commands. Return read data XMI cycles are also stored in the RRF. The RRF contains the JXDI command generator for CPU, interrupt, and XJA-to-ICU status transactions. The RCM controls the status of the buffers in the RRF and sends this status to the XRC.

- **Register (REG)** — The REG implements the XMI required and XJA-specific registers. The XMI required registers are implemented as visible from the XMI and the CPUs. In general, the XJA-specific registers are only visible from the CPUs. The REG is also responsible for most of the XJA error handling in conjunction with the receive and transmit control machines.
- **Transmit register file (TRF)** — The TRF receives JXDI transfers from the ICU and buffers them for transmit on the XMI. It contains two buffers, each capable of handling a hexword return read data transaction. The TRF is loaded under control of the JXDI control array (XCE). The TRF is unloaded to either the REG or the XMI under control of the transmit control machine (TCM). Status of the TRF buffer is controlled by the TCM, which sends the status to the XCE array.
- **Receive control machine (RCM)** — The RCM controls the receipt of XMI transactions, and notifies the XCE array that a given transaction is to be sent to the ICU. The RCM controls the status of the buffers in the RRF and sends the status to the XRC. If an XMI transaction references the XMI registers, the RCM passes control to the TCM.

  The RCM handles the XJA DMA flow control. If no DMA command/address (C/A) or write buffers are available in the XJA for servicing XMI DMA requests, the XRC NOACKs any DMA XMI transactions that select the XJA. The XJA always accepts return read data XMI transactions (provided the XJA has a CPU read outstanding) and XMI transactions that select the XJA's XMI visible registers.
- **Transmit control machine (TCM)** — The TCM controls the generation of XMI command and response sequences for receipt of a valid JXDI transfer, or response to an XMI read request that referenced the XJA's XMI registers. The TCM also maintains control of all accesses to the REG. The XDC logic of the XJA provides the interface to the XMI, and controls most XJA functions. The XCD contains the XJA registers (REG), transmit and receive control machines (TCM and RCM), receive and transmit register files (RRF and TRF), and XMI receive logic (XRC).
- **Add-on self-test (AOST)** — The XJA contains add-on self-test logic that interfaces with the XDC array. The logic provides test inputs to check XJA operation and report any test failures.

7.2.1 XJA Adapter Transactions

The XJA communicates with the ICU and the XMI through three transaction types: DMA, CPU, and interrupt.

7.2.1.1 DMA Transactions

XMI transactions that select the XJA as the responder node are forwarded to the ICU as DMA transactions. DMA transactions can be reads, writes, read locks, or write unlocks and can be quadword, octaword, or hexword in length. The XJA can accept up to four DMA read transactions from the XMI at any given time. Read data returned from the ICU is forwarded on the XMI as read data response transactions.

The JXDI command prefix code for DMA transactions is DMA.

7.2.1.2 CPU Transactions

VAX 9000 CPUs can access the I/O portion of VAX physical address space through CPU transactions. These transactions are received by the XJA from the JXDI and are forwarded on to the XMI with the XJA as the commander. CPU transactions are longword in length and the XJA can accept only a single CPU transaction at a time.

The JXDI command prefix code for CPU transactions is CPU.

7.2.1.3 Interrupt Transactions

The XJA fields interrupt transactions from the XMI and forwards them to the ICU using interrupt transactions. The resulting SCB offset vector fetch initiated by a VAX 9000 CPU is a CPU transaction.

The JXDI command prefix code for interrupt transactions is INTR.

7.2.2 XJA Clock Systems

The XJA uses three separate, asynchronous, clock systems:

The XCI_C clock system sends and receives data from the XMI.
The CLKJ clock system receives data from the ICU.
The CLKX clock system sends data to the ICU.

Each clock system drives a specific portion of synchronous logic. When a given set of logic needs to communicate with another set of logic running on a different clock system, synchronizing logic must be used.

The XJA communicates across asynchronous boundaries as shown in Figure 7–4. Logic A and logic B run on different clocks. The data buffer is unidirectional and is loaded by CLK A (logic A clock) and unloaded by CLK B (logic B clock). Control lines from logic A inform logic B that the data has been completely loaded into the data buffer and can now be unloaded. Data is never loaded and unloaded at the same time.

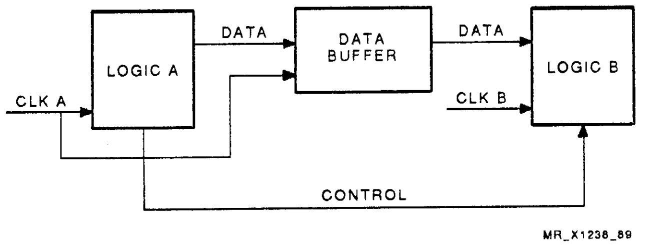


Figure 7–4 Data Communication Across Asynchronous Boundaries

7.2.2.1 XCI_C Clock System

The logic that interfaces to the XMI runs off the XCI_C clock system. The six XCI_ C clock signals each have a 64-ns duration. The XCI_C clocks are derived from the XCLOCK chip in the XMI corner. The XCLOCK generates the XCI_C clocks using the XMI_TIME and XMI_PHASE clock signals from the XMI. Figure 7–5 shows a block diagram of the XCI_C clock system.

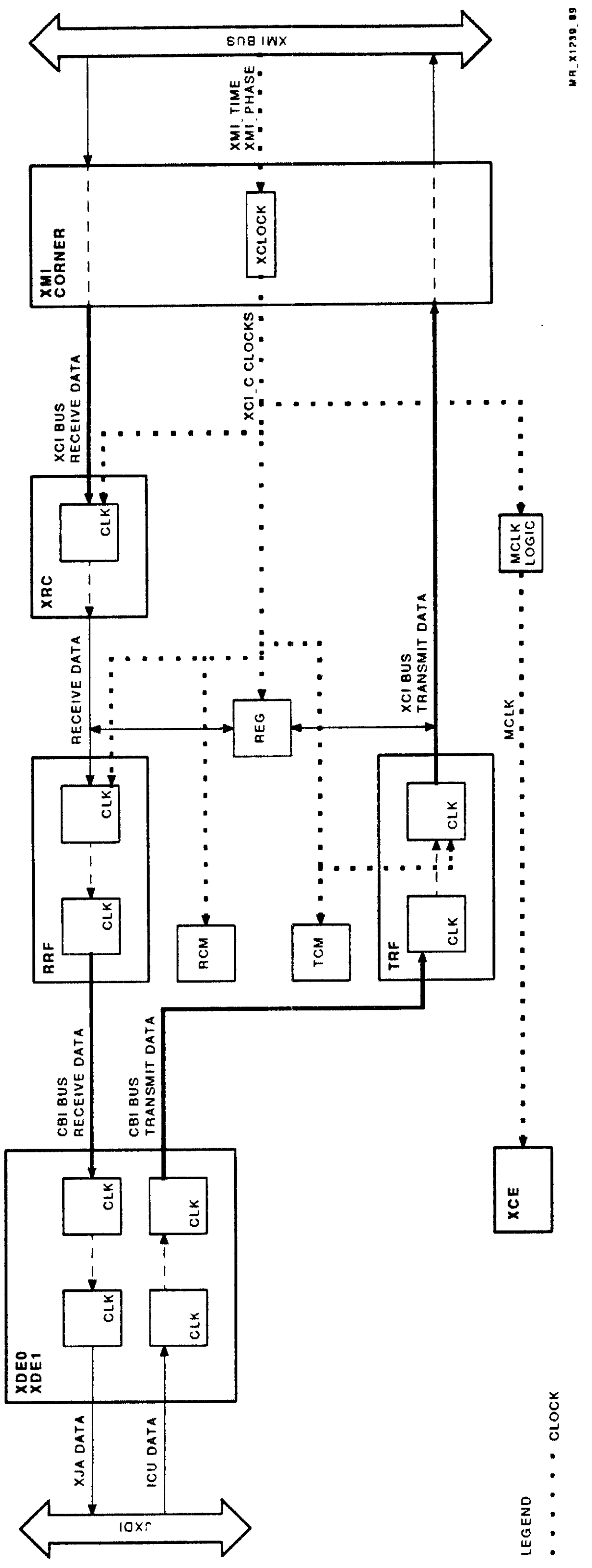


Figure 7–5 XCI_C Clock System

The XCI_C clock system does the following:

- Clocks the receive data from the XMI bus, through the XRC, and into the RRF.
- Supplies 64-ns clocks to the RCM to synchronize that portion of the RCM that controls the reception of the XMI data in the XRC and RRF.
- Clocks the transmit data out of the TRF to the XMI bus.
- Supplies 64-ns clocks to the TCM to synchronize that portion of the TCM that controls the transmission of the XMI data from the TRF.
- Clocks data into and out of the XJA registers.

All the receive and transmit data synchronized by the 64-ns XCI_C clocks is in 64-bit format except register data, which is longword in length.

Two XCI_C clocks (XCI_C34 and XCI_C61) are ORed to generate an asymmetrical 32-ns clock called MCLK. MCLK is supplied to the XCE to clock in control signals from the RCM and the TCM that are synchronized to the XCI_C clock system.

7.2.2.2 CLKJ Clock System

The logic that receives data from the ICU, through the JXDI, is clocked by the CLKJ clock system. ICU_CLKJ is received from the ICU as a 16-ns (nominal system cycle time) square wave. ICU_CLKJ is derived in the ICU from system CLOCK_B. The XJA uses the 16-ns ICU_CLKJ to clock data off the JXDI into the XDE0 and XDE1 chips. ICU_CLKJ also synchronizes the portion of the XCE logic that controls the data flow into the XDE chips and the portion that processes the ICU control signals associated with the data flow from the ICU. The JXDI data that is transferred into the XDE chips is in 16-bit word format. Figure 7–6 shows a block diagram of the CLKJ clock system.

ICU_CLKJ is divided by two in the XCE to form a 32-ns clock called SCLKJ. SCLKJ (slow CLKJ) is used to do the following:

- Clock the transmit data out of the XDE0 and XDE1 chips onto the CBI bus.
- Clock the transmit data off the CBI bus into the TRF.
- Synchronize that portion of the TCM that controls the transfer of the transmit data from the CBI bus to the TRF.

7.2.2.3 CLKX Clock System

The logic that transmits data to the ICU, through the JXDI, is clocked by the CLKX clock system. CLKX is derived from a B-phase clock (CLKB) in the master clock module (MCM). It is used to clock data from the XDE0 and XDE1 chips to the ICU across the JXDI. CLKX also synchronizes the portion of the XCE logic that controls the data flow out of the XDE chips and the portion that processes the ICU control signals associated with data flow to the ICU. The data that is transferred from the XDE chips to the ICU is in 16-bit word format.

The 16-ns CLKB is divided by two to produce two 32-ns waveforms (SCLKX and UNLOAD_CLK). UNLOAD_CLK is used to clock the 32-bit receive data from the RRF to the XDE chips over the CBI bus, and to synchronize the XCE logic that controls unloading of the receive data onto the CBI. SCLKX is used to load the 32-bit data into the XDE chips, and to synchronize the XCE logic that controls the loading of the data into the XDE chips. Figure 7–7 shows a block diagram of the CLKX clock system.

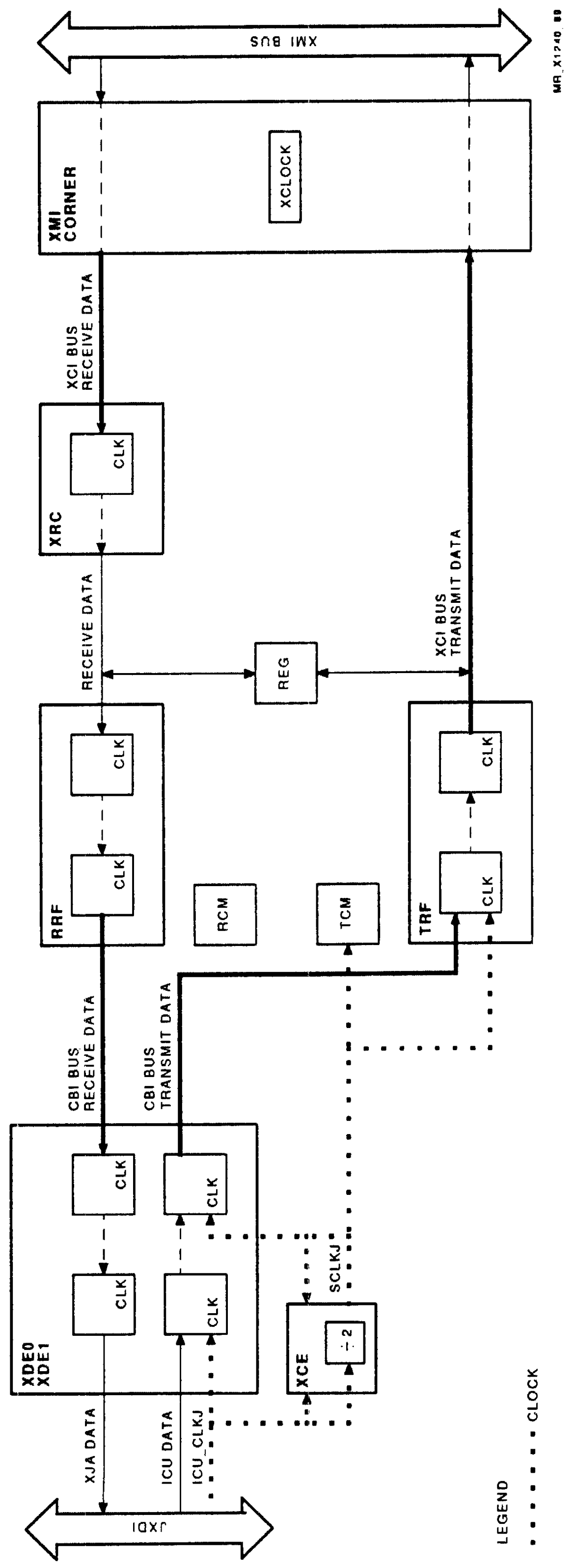


Figure 7–6 CLKJ Clock System

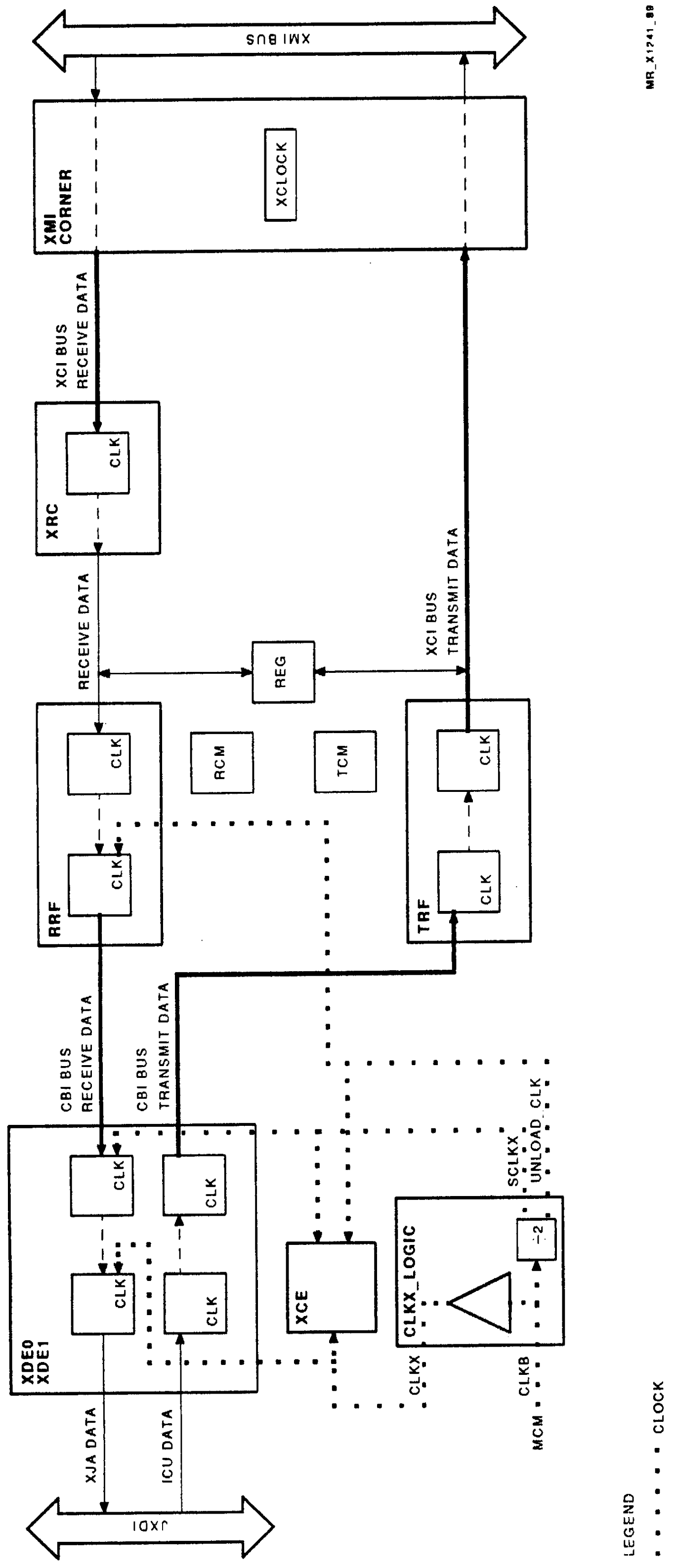


Figure 7–7 CLKX Clock System

7.3 JXDI Interface Functional Overview

The JXDI interfaces 19 signals between the XJA adapter and the ICU portion of the SCU (Figure 7-8). All JXDI signals are unidirectional. Signals going from the XJA to the ICU are prefixed with XJA. Those going from the ICU to the XJA are prefixed with ICU. Two signals not prefixed (SPU_RESET and SPU_XJA_CLK_STOP_OUT) are generated by the SPU and transmitted to the XJA across the ICU. Table 7-3 describes the functions of the JXDI signals.

Twelve of the 19 JXDI signals are symmetrical. That is, six lines from the ICU correspond to identical lines from the XJA. The symmetrical lines are the first six lines of each signal line group shown in Figure 7-8.

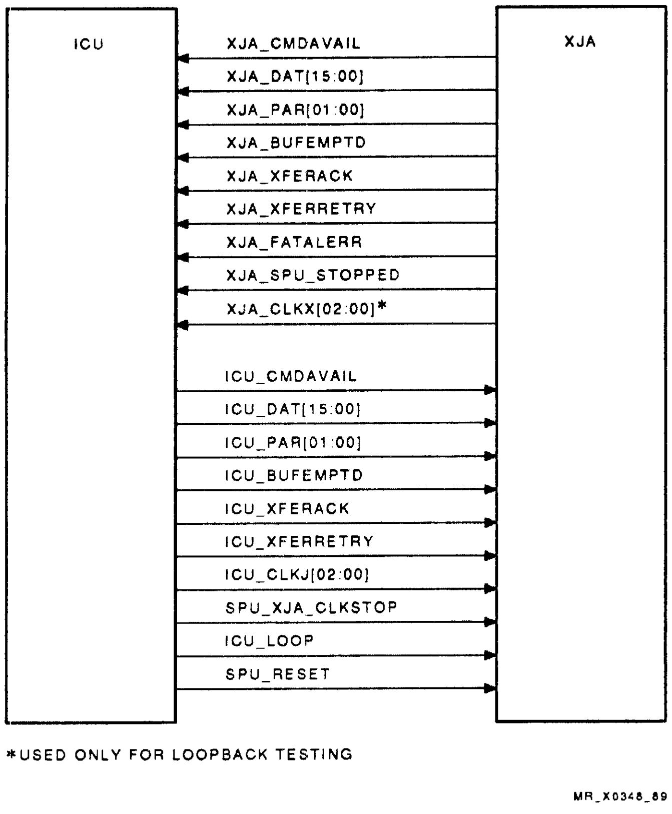


Figure 7-8 JXDI Signals

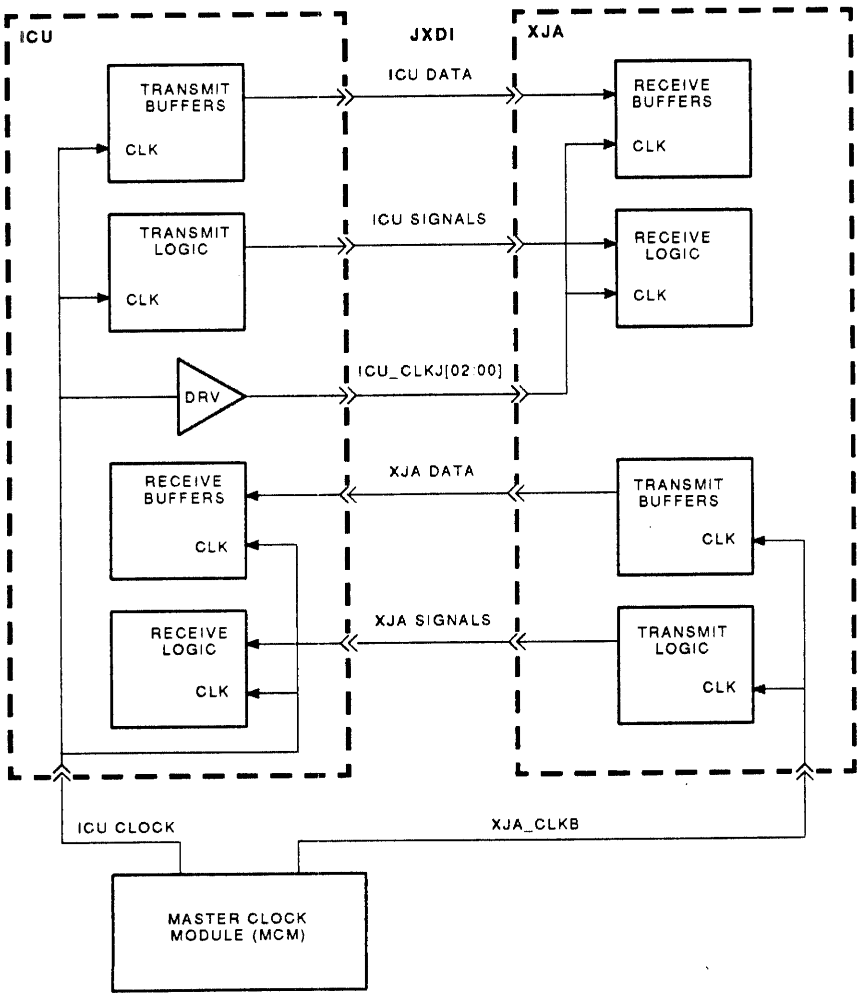
Table 7–3 JXDI Signal Summary

Signal	Function
XJA_CMDAVAIL	XJA command available. Informs the ICU that a data packet follows in the next cycle.
XJA_DAT[15:00]	XJA read/write data lines. Also includes command, address, mask, ID, length, and IPL.
XJA_PAR[01:00]	Odd data line parity.
XJA_XFERACK	XJA transfer acknowledge. Informs the ICU that a data packet has been accepted.
XJA_XFERRETRY	XJA transfer retry. Informs the ICU that a data packet was rejected and that the ICU should retransmit the packet.
XJA_BUFEMPTD	XJA buffer empty. Informs the ICU that a receive buffer is empty, a data packet parity error was detected, or the XJA was busy. If the packet was not accepted, this function indicates that the receive buffer for the unaccepted packet is still available.
XJA_SPU_STOPPED_OUT	A response to XJA_SPU_STOP_OUT from the SPU. The function informs the SPU that the XJA will not transfer any packets over the JXDI until the SPU clock stop function is negated.
XJA_FATALERR	Indicates that the XJA has detected a fatal error.
XJA_CLK[02:00]	Used only for loopback testing.
ICU_CMDAVAIL	ICU command available. Informs the XJA that a data packet follows in the next cycle.
ICU_DAT[15:00]	ICU read.
ICU_PAR[01:00]	Odd data line parity.
ICU_XFERACK	ICU transfer acknowledge. Informs the XJA that a data packet has been accepted.
ICU_XFERRETRY	ICU transfer retry. Informs the XJA that a data packet was rejected and that the ICU should retransmit the packet.
ICU_BUFEMPTD	ICU buffer empty. Informs the XJA of similar conditions as specified for XJA_BUFEMPTD.
ICU_CLKJ[02:00]	A 3-line fanout of the 16-ns ICU clock transmitted to the XJA for clocking data and control signals.
SPU_RESET	SPU-generated reset function transferred to the XJA during system initialization.
SPU_XJA_CLK_STOP_OUT	An SPU-generated, stop clock function that informs the XJA of an impending clock stoppage. The XJA completes the current transaction but will initiate any new transactions.
ICU_LOOP	A test command that loops back all symmetrical ICU-sourced JXDI signals onto the corresponding XJA lines.

The XJA and the ICU have transmit buffers from which to transmit data to the JXDI, and receive buffers to receive data from the JXDI (Figure 7–9). The MCM (master clock module) supplies an ICU clock that controls the transmit buffers and transmit logic in the ICU. This clock is transferred to the XJA as ICU_CLKJ[02:00], where it controls the receive buffers and receive logic in the XJA. The ICU data, signals, and clock (ICU_CLK[02:00]) experience the same delay over the JXDI. Therefore, ICU_CLK[02:00] arrives at the XJA at the correct time to clock in the ICU data and signals. However, due to the JXDI delay, the data clocked into the XJA is asynchronous with respect to the system clocks and must be synchronized later on within the XJA.

Another clock from the MCM (XJA_CLKB) is supplied to the XJA where it controls the transmit buffers and transmit logic. XJA_CLKB is delayed with respect to the ICU clock that is used to clock the ICU receive buffers and receive logic. The delay compensates for the XJA-to-ICU skew and time delay, resulting in the XJA data and signals being clocked into the ICU asynchronously with respect to the system clocks.

The JXDI bus cycle time is equal to the nominal CPU cycle time of 16 ns. The transfers executing over the JXDI are part of DMA, CPU, or interrupt transactions.



MR_X0349_89

Figure 7–9 ICU and XJA Transmit and Receive Buffers

7.3.1 DMA Transactions

DMA transactions are reads and writes of main memory from the XMI. A DMA transaction may be quadword, octaword, or hexword in length. Up to four DMA transactions may be outstanding at a time.

7.3.2 CPU Transactions

CPU transactions are system CPU reads and writes of registers in the XJA or on the XMI. CPU transactions are longwords. Only one CPU transaction can be outstanding at a time.

7.3.3 Interrupt Transactions

Interrupt transactions notify the CPU of errors detected by the XJA, faults on the XMI bus, or interrupts generated by the XMI nodes.

7.3.4 JXDI Transfer Commands

Ten transfer commands (listed in Table 7–4) execute on the JXDI. The table indicates command direction and whether the command is executing in a DMA or a CPU transaction. Note that an interrupt request from the XJA to the ICU may occur during a DMA transaction or a CPU transaction.

Table 7–4 JXDI Transfer Commands

	Direction	
Command	**XJA to ICU**	**ICU to XJA**
READ_REQUEST	DMA	CPU
READ_LOCK_REQUEST	DMA	–
READ_DATA_RETURN	CPU	DMA
READ_LOCK_DATA_RETURN	–	DMA
WRITE_REQUEST	DMA	CPU
WRITE_UNLOCK_REQUEST	DMA	–
Reserved	–	–
Reserved	–	–
INTERRUPT_REQUEST[1]	DMA, CPU	–
READ_LOCKED_STATUS	–	DMA
READ_ERROR_STATUS	CPU	DMA
WRITE_COMPLETE	CPU	–
Reserved	–	–
Reserved	–	–
Reserved	–	–
Reserved	–	–

[1]An interrupt may occur during a DMA or a CPU transaction.

7.4 XMI Bus Functional Overview

The XMI is a limited length, pended, synchronous bus with centralized arbitration and a 64-ns bus cycle. A pended bus operates such that a node does not hold the bus while waiting for a response. Several transactions can be in progress at a given time.

Arbitration and data transfers occur simultaneously, with multiplexed data and address lines. The bus supports quadword, octaword, and hexword reads and writes to memory. In addition, the bus supports longword read and write operations to I/O space. These longword operations implement byte and word modes required by certain I/O devices.

The CCARD arbitrates among the nodes requesting use of the bus, and supplies a standard XMI clock to all the nodes. When a node requires the XMI, it issues a request. The arbiter honors the request with a grant signal (XMI_GRANT). There are two types of XMI requests, a command request and a response request. A command request asks for the bus to initiate a new transaction. A response request asks for the bus to respond to a previous request, for example, a response request to place read data on the XMI that was requested by a previous read request.

The arbiter has two queues, one for command requests and one for response requests. The queues use an algorithm that simulates a round-robin scheme. The arbiter gives the response queue priority over the command queue. Hence an XMI response request has priority over an XMI command request.

The arbiter also responds to a hold signal (XMI_HOLD) and a suppress signal (XMI_SUP). XMI_GRANT gives the XMI bus to the requesting node for only one cycle. If the node needs the bus for another cycle, it asserts the hold signal and the arbiter allows the node to use the XMI for another cycle. By continuously asserting XMI_HOLD, the node can hold the bus for a maximum of four consecutive cycles.

If a node is momentarily unable to keep up with bus traffic, it asserts the suppress signal (XMI_SUP). When XMI_SUP is asserted, the arbiter does not respond to any new command requests until the node "catches up" and removes the suppress signal.

7.4.1 XMI Corner

Each module on the XMI uses a standard interface called the XMI corner. The corner interfaces all the control and data signals going to and from the XMI. The corner uses a predefined etch and consists of seven latches and a clock chip.

The clock chip receives two clock signals (XMI_TIME and XMI_PHASE) from the XMI CCARD. The clock chip generates a set of subclocks that are used to control the corner latches. The subclocks are also made available to the node-specific logic on the module.

Table 7–5 summarizes the function of each XMI signal.

Table 7–5 XMI Signal Line Summary

Class	Signal	Function	XJA Input/Output
Arbitration	XMI_CMD_REQ	Command request	Output
	XMI_RES_REQ	Response request	Output
	XMI_GRANT	Request granted	Input
	XMI_HOLD	Hold the bus	Output
	XMI_SUP	Suppress new requests	Input/output
Information	XMI_ FUNCTION[03:00]	Command function	Input/output
	XMI_DATA[63:00]	Data, addr-mask-cmd	Input/output
	XMI_ID[05:00]	Commander ID	Input/output
	XMI_PARITY[02:00]	Parity over info lines	Input/output
Response	XMI_CNF[02:00]	Confirmation	Input/output
Control	XMI_BAD	Node failure	Input/output
	XMI_FAULT	Uncorrectable error	Input
	XMI_DEFAULT	Idle XMI cycles	Input
	XMI_RESET	Initialize system	Input/output
	XMI_TIME	Clock reference	Input
	XMI_PHASE	Phase reference	Input
	XMI_AC_LO	Low ac line voltage	Input/output
	XMI_DC_LO	Impending loss of dc	Input
Miscellaneous	XMI_NODE_ID[03:00]	Node ID	Input

7.5 System Address Space

Figure 7–10 shows the system address space. Thirty-four address bits are used to specify 15.5 Gbytes of main memory (0 0000 0000 to 3 DFFF FFFF) and 512 Mbytes of I/O space (3 E000 0000 to 3 FFFF FFFF). DMA transactions to main memory are checked in the XRC for a valid memory address. DMA addresses outside the correct range are not acknowledged. CPU transactions to I/O space are addressed to the 512-Mbyte I/O space region.

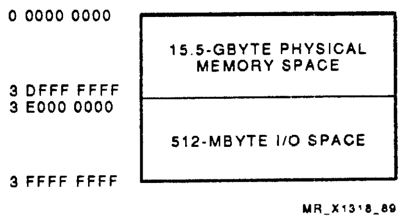


Figure 7–10 System Address Space

7.5.1 System I/O Space Allocation

System I/O space is divided into three regions:

XMI node space
BI window space
XJA private register space

Figure 7–11 is a breakdown of the VAX 9000 I/O space. The first four 8-Mbyte segments are designated as XMI node space for the four XMI I/O channels. Each segment has sixteen 512-Kbyte address slots for the XMI adapters (including XBI+ adapters). Address slots 0 and 15 are not used. The remaining 14 address slots correspond to the 14 physical slots in the XMI card cage (Figure 7–2).

Address slot 7 is not used as physical slot 7 is occupied by the XMI CCARD. (The CCARD is not a node on the XMI bus.) Address slot 8 (node 8) is used for the XJA. This leaves 12 address slots (1 through 6 and 9 through 14) available for XMI adapters.

Located within each of the adapter address slots are the adapters' XMI space registers. Figure 7–12 shows the addresses of the eight XMI space registers. The base block (bb) address is the address of the first XMI node space segment (3 E000 0000) plus 80 0000 for each XMI node segment, plus 8 0000 for each XMI node adapter. The register addresses are the bb address plus the address specified by bits [04:02].

The XMI space registers are described in the *VAX 9000 Family XJA Technical Description* (EK-KA90A-TD).

Following the XMI node space is the XBI+ adapter space (BI windows). Space is allocated for 14 XBI+ adapters. The BI window segments are each 32 Mbytes in length.

Following the BI window space are four 512-Kbyte address segments for the XJA private registers.

The final 30 Mbytes of I/O space is allocated to SCU and SPU registers. These registers are located in the SCU, therefore, access to these registers is not through the I/O channels.

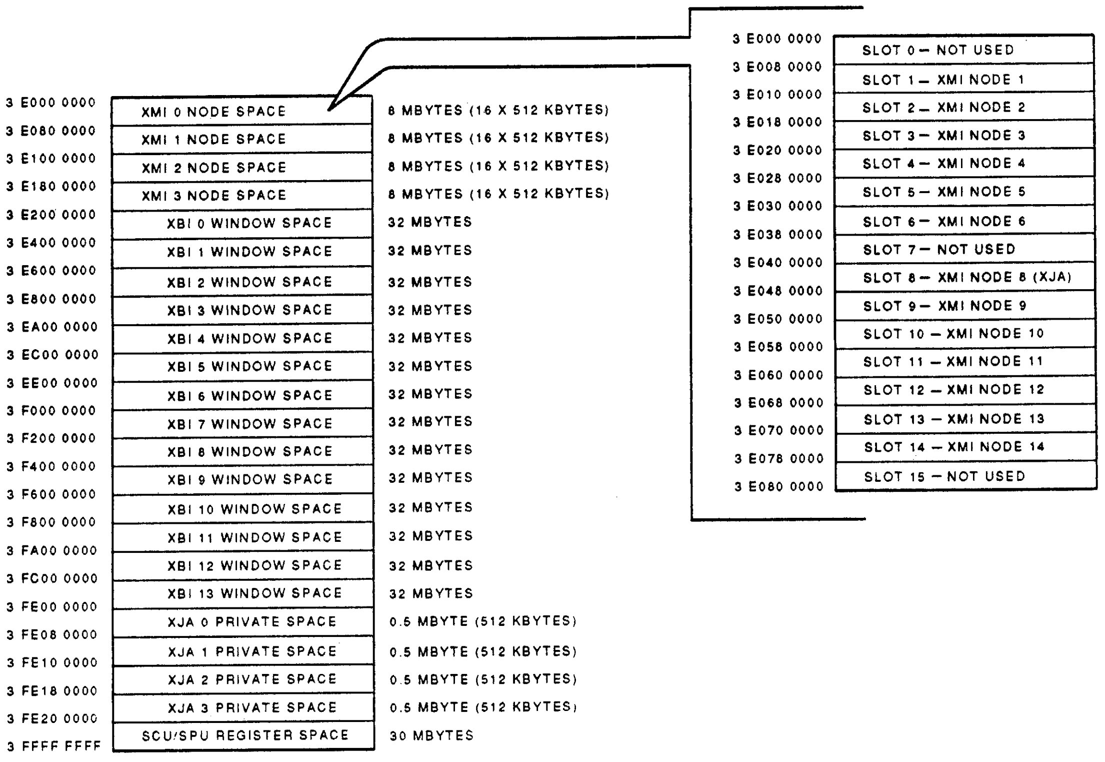
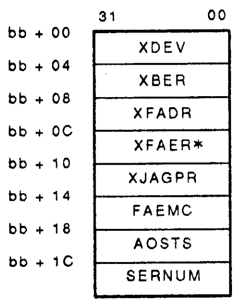


Figure 7–11 System I/O Space Allocation



bb = 3 E000 0000 + (XJA NUMBER X 80 0000) + (XMI NODE NUMBER X 8 0000)
*XFAER IS ALSO ACCESSED AT ADDRESS bb + 2C.

MR_X1320_89

Figure 7–12 Addresses of XMI Space Registers

7.5.1.1 XJA Private Register Space

CPU access to the XJA private registers does not involve an XMI transaction. The XJA recognizes an XJA private space address and accesses the specified register directly. The address specifies the XJA channel as well as the register itself. Figure 7–13 shows the addresses of the 15 XJA private registers. The bb address is the address of the first XJA private register space segment (3 FE00 0000) plus 8 0000 for each XJA private register segment. The register addresses are the bb address plus the address specified by bits [04:02].

The XMI space registers are described in the *VAX 9000 Family XJA Technical Description* (EK-KA90A-TD).

The SCU determines which XJA is being requested by the CPU. Each XJA channel receives only the CPU requests for its own private registers.

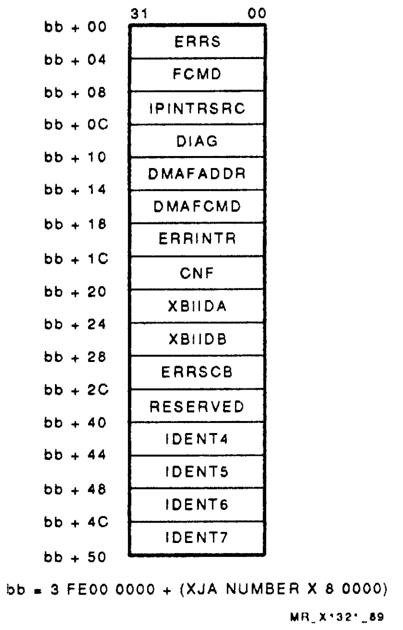


Figure 7–13 Addresses of XJA Private Registers

7.5.1.2 XMI Node Space

The XMI bus conforms to XMI standard protocol. One of the XMI standards is a 40-bit addressing scheme with the MSB (bit [39]) dividing the XMI node space into memory space (bit [39] = 0) and I/O space (bit [39] = 1). Figure 7–14 is a breakdown of XMI I/O space starting at address 80 0000 0000 (bit [39] = 1).

XMI protocol designates the first 24 Mbytes of XMI I/O space as XMI private space. This space is not used by the VAX 9000. The next eight Mbytes are the node space for the XMI adapters. The node space has sixteen 512-Kbyte address slots comparable to the 16 address slots of Figure 7–11.

The node space is followed by fifteen 32-Mbyte segments of XBI+ window space. The VAX 9000 system has only 14 BI windows in its I/O space; therefore, the last BI window in XMI I/O space is not used.

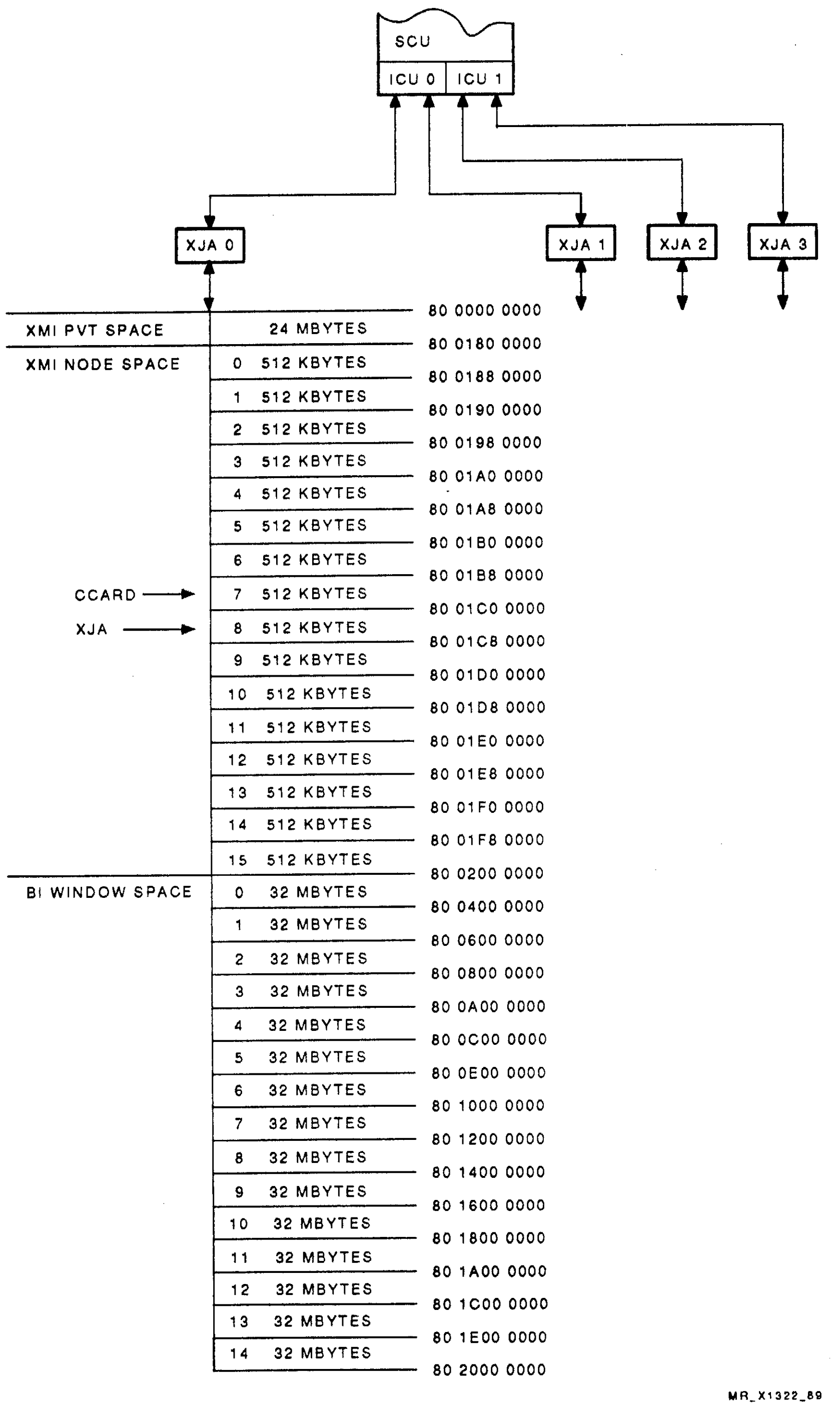


Figure 7–14 XMI I/O Space Allocation

7.5.2 XMI Address to VAX 9000 Address

In executing a DMA transaction, a data packet goes from an XMI address to a VAX 9000 address. As previously mentioned, the XRC checks the XMI address for a valid reference to VAX 9000 main memory. To do this, the XRC makes sure that:

- XMI address bit [39] (the MSB) is 0. This signifies that the XMI reference is to memory space.
- XMI address bits [38:34] are all 0s. Otherwise, the reference is above VAX 9000 space.
- XMI address bits [33:29] are not all 1s. If address bits [33:29] are all 1s, the reference is to VAX 9000 I/O space (Figure 7–10).

If the preceding is true, the DMA reference is to VAX 9000 memory space and XMI address bits [33:00] specify the target location in main memory.

7.5.3 VAX 9000 Address to XMI Address

Figure 7–15 shows the VAX 9000 I/O space allocation given in Figure 7–11, but with the addition of binary address bits [33:16]. Refer to Figure 7–15 during the following discussion of VAX 9000 to XMI addressing.

MBYTES			BINARY	HEX
			33 16	
0			11 1110 0000 0000 0000	3 E000 0000
	XMI 0 NODE SPACE	8 MBYTES		
8			11 1110 0000 1000 0000	3 E080 0000
	XMI 1 NODE SPACE	8 MBYTES		
16			11 1110 0001 0000 0000	3 E100 0000
	XMI 2 NODE SPACE	8 MBYTES		
24			11 1110 0001 1000 0000	3 E180 0000
	XMI 3 NODE SPACE	8 MBYTES		
32			11 1110 0010 0000 0000	3 E200 0000
	XBI 0 WINDOW SPACE	32 MBYTES		
64			11 1110 0100 0000 0000	3 E400 0000
	XBI 1 WINDOW SPACE	32 MBYTES		
96			11 1110 0110 0000 0000	3 E600 0000
	XBI 2 WINDOW SPACE	32 MBYTES		
128			11 1110 1000 0000 0000	3 E800 0000
	XBI 3 WINDOW SPACE	32 MBYTES		
160			11 1110 1010 0000 0000	3 EA00 0000
	XBI 4 WINDOW SPACE	32 MBYTES		
192			11 1110 1100 0000 0000	3 EC00 0000
	XBI 5 WINDOW SPACE	32 MBYTES		
224			11 1110 1110 0000 0000	3 EE00 0000
	XBI 6 WINDOW SPACE	32 MBYTES		
256			11 1111 0000 0000 0000	3 F000 0000
	XBI 7 WINDOW SPACE	32 MBYTES		
288			11 1111 0010 0000 0000	3 F200 0000
	XBI 8 WINDOW SPACE	32 MBYTES		
320			11 1111 0100 0000 0000	3 F400 0000
	XBI 9 WINDOW SPACE	32 MBYTES		
352			11 1111 0110 0000 0000	3 F600 0000
	XBI 10 WINDOW SPACE	32 MBYTES		
384			11 1111 1000 0000 0000	3 F800 0000
	XBI 11 WINDOW SPACE	32 MBYTES		
416			11 1111 1010 0000 0000	3 FA00 0000
	XBI 12 WINDOW SPACE	32 MBYTES		
448			11 1111 1100 0000 0000	3 FC00 0000
	XBI 13 WINDOW SPACE	32 MBYTES		
480			11 1111 1110 0000 0000	3 FE00 0000
	XJA 0 PRIVATE SPACE	512 KBYTES		
480.5			11 1111 1110 0000 1000	3 FE08 0000
	XJA 1 PRIVATE SPACE	512 KBYTES		
481.0			11 1111 1110 0001 0000	3 FE10 0000
	XJA 2 PRIVATE SPACE	512 KBYTES		
481.5			11 1111 1110 0001 1000	3 FE18 0000
	XJA 3 PRIVATE SPACE	512 KBYTES		
482.0			11 1111 1110 0010 0000	3 FE20 0000
	SCU/SPU REGISTER SPACE	30 MBYTES		
512.0			11 1111 1111 1111 1111	3 FFFF FFFF

MR_X1323_89

Figure 7–15 VAX 9000 I/O Address Bits

All CPU transactions are to one of the three VAX 9000 I/O space regions previously mentioned:

XMI node space — Containing all I/O adapters except BI adapters
BI window space — Containing all the BI adapters
XJA private register space

When a CPU transaction is initiated, the SCU determines which XJA channel is being addressed and transmits the CPU request packet to the correct channel. To do this, the SCU first makes sure that address bits [33:29] are all 1s to verify that the CPU reference is to I/O space. The SCU then strips off address bits [33:30] (because these bits are not required) and transfers an address field of [29:00] to the XJA.[1]

Now the SCU checks address bits [28:25] to determine which I/O area the CPU is requesting. If address bits [28:25] are all 0s, the CPU request is for XMI node space. In this case, address bits [24:23] specify the XJA channel being referenced and the SCU directs the request to that channel.

If address bits [28:25] are all 1s, the CPU request is for XJA private register space. In this case, address bits [20:19] specify the XJA channel being referenced and the SCU directs the CPU request to that channel.

When address bits [28:25] are neither all 1s nor all 0s, the CPU request is to BI window space. In this case, address bits [28:25] specify which BI window is being referenced and the SCU directs the CPU request to that channel.

Address processing of CPU requests to XMI node space or BI window space involve going from a VAX 9000 address to an XMI address. Because the XMI I/O address space shown in Figure 7–14 applies to all four XMIs, address translation is required to convert the VAX 9000 address from that shown in Figure 7–11 to the XMI I/O space address of Figure 7–14.

7.5.3.1 VAX 9000 Address to XMI Node Space

When the CPU reference is to XMI node space (CPU address bits [28:25] = 0s), the XJA must translate the CPU address to the base block address for XMI node space. In Figure 7–14, you can see that this address is 80 0180 0000. The XJA locates this address on the XMI by:

- Setting XMI address bit [39] to 1. This is done by placing CPU address bit [29] (which is always a 1) onto XMI address bit [39].
- Setting XMI address bits [38:25] to 0s.
- Setting XMI address bits [24:23] to 1s.

The preceding establishes an XMI address of 80 0180 0000. CPU address bits [22:00] provide XMI address bits [22:00] to reference the specific XMI adapter and the XMI space register within the adapter. Address bits [22:19] locate the adapter address slot, while bits [18:00] select the target location within the adapter.

[1] Although bit [29] is always a 1, it is transferred to the XJA to be used as an XMI address bit [39] to address the XMI's I/O space.

7.5.3.2 VAX 9000 Address to BI Window Space

When the CPU reference is to BI window space (CPU address bits [28:25] are both 1s and 0s), the XJA must translate the CPU address into the XMI address for the referenced BI adapter. The operating system loads the XBI ID A and XBI ID B registers in each XJA with XMI address bits [28:25] for all the BI adapters in the system.

Referring to the XMI I/O space shown in Figure 7–14, you see that BI window space starts at XMI address 80 0200 0000 and increases by 200 0000 for each BI window. The XJA locates this address on the XMI by:

- Setting XMI address bit [39] to 1.
- Setting XMI address bits [38:29] to 0s.
- Using address bits [28:25] to reference the XBI ID A or XBI ID B register and unload the 4-bit nibble that specifies the location of the referenced window on the XMI bus. (The operating system has loaded the XBI ID A and XBI ID B registers in each XJA with XMI address bits [28:25] for all the BI adapters in the system.)

CPU address bits [24:00] are then used to reference the desired location within the BI window.

8 System Exception and Interrupts

This chapter provides an overview of system interrupt and exception handling. It defines the types of interrupts and exceptions, and provides a sequence of events that occurs when handling them.

Descriptions of the registers involved in exception and interrupt handling are also provided.

8.1 Overview

In the VAX 9000, interrupts and exceptions are handled by the EBox, IBox, MBox, and JBox. Depending on the type of interrupt or exception, one or more functional units is involved in the handling routines.

Because the EBox is pipelined, the exception and interrupt handling routines are designed to function in correlation with the EBox pipeline. The EBox microcode handles interrupts at the beginning or end of the EBox pipeline.

The IBox handles memory faults, reserved opcodes, and reserved addressing mode faults. The memory management faults in the IBox can be I-stream and operand references.

The MBox contains registers for memory management handling and does not get involved with other exceptions or interrupts. The MBox handles a translation buffer (TB) miss without EBox intervention. This allows the EBox to handle only exceptions that change the instruction flow and cause an IBox flush.

The system control unit (SCU) contains registers for hardware interrupts. The system control block (SCB) offset for pending interrupts is also kept in the SCU or the interrupting adapter. The adapters for the I/O are reached by reading physical addresses in the SCU.

8.2 Interrupt Handling

Interrupts are defined in two basic groups: hardware-generated interrupts and software-generated interrupts.

Hardware interrupts are generated in the SCU. The SCU dispatches a particular EBox to respond to the interrupt. The EBox responds when its interrupt priority level (IPL) is lowered to the correct level. The EBox microcode reads an SCU register to determine the correct SCB vector. The SCB vector then provides the correct address of the interrupt service routine. The MBox and IBox do not get involved in interrupt servicing. The SCU assigns interrupts to a single CPU or assigns them in a rotating manner.

The software-generated interrupts are handled entirely in the EBox by microcode and hardware and are serviced by the same CPU where they are generated.

8.3 Hardware Interrupts

Hardware interrupts are processed at the end of an instruction. Interrupts between instructions are generated by the hardware and divert the instruction flow to the interrupt microtrap address.

The following list summarizes the events for servicing hardware-generated interrupts:

1. SCU receives an interrupt request from an I/O device.
2. SCU selects the EBox to service the interrupt and sends the interrupt to the EBox.
3. EBox responds to the interrupt by causing a microtrap. The microtrap does not occur until the EBox IPL is lower than the interrupt IPL.
4. EBox microcode branches to the interrupt sequence. This requires two branch cycles that are used to decode the exact interrupt being serviced.
5. EBox microcode sends a read request to the SCU.
6. SCU responds with the SCB offset value.
7. EBox adds the SCB offset to the value of the system control block base (SCBB) register and performs a physical read of that memory location.
8. Response back contains the dispatch address to the interrupt handler.
9. IBox is dispatched.

The I/O control unit (ICU) logic portion of the SCU is responsible for interrupt handling. Interrupts that the ICU receives must arbitrate for a CPU to be serviced. The ICU contains an interrupt arbiter mapping mechanism that determines which interrupt has the highest IPL. The interrupt arbiter map contains the IPL levels for the system. Interrupt requests are mapped with their IPLs, and the device with the highest IPL wins arbitration.

When arbitration determines the highest IPL and the interrupt to be serviced, the ICU must select the EBox to service the interrupt. The selection of the EBox (or CPU) is based on the contents of the CPU configuration register. This register contains installation-specific configuration information and a bit determining whether round-robin interrupts or broadcast interrupts are enabled. Figure 8–1 shows the CPU configuration register, and Table 8–1 lists the fields and descriptions.

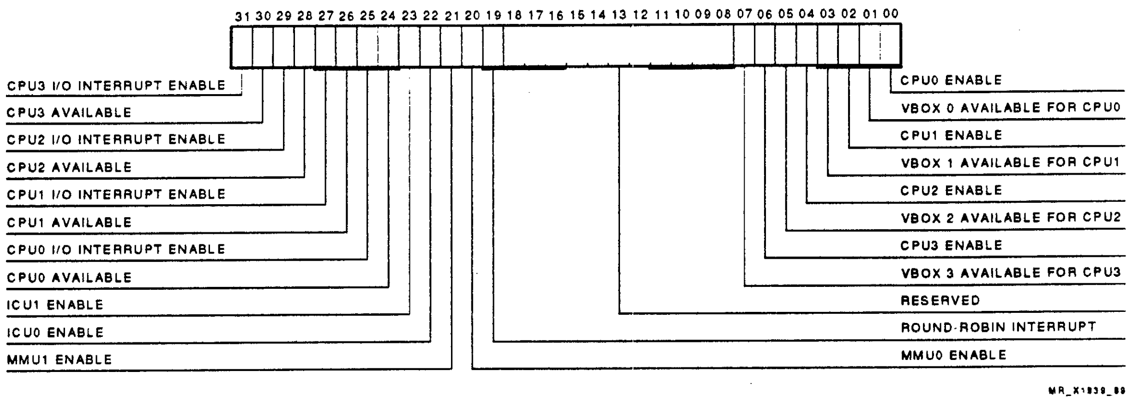


Figure 8–1 CPU Configuration Register Format

Table 8–1 CPU Configuration Register Descriptions

Field	Description
00	CPU0 enable
01	VBox 0 available for CPU0
02	CPU1 enable
03	VBox 1 available for CPU1
04	CPU2 enable
05	VBox 2 available for CPU2
06	CPU3 enable
07	VBox 3 available for CPU3
18:08	Reserved
19	Round-robin interrupt
20	MMU0 enable
21	MMU1 enable
22	ICU0 enable
23	ICU1 enable
24	CPU0 available
25	CPU0 I/O interrupt enable
26	CPU1 available
27	CPU1 I/O interrupt enable
28	CPU2 available
29	CPU2 I/O interrupt enable
30	CPU3 available
31	CPU3 I/O interrupt enable

If bit 19 (round-robin interrupt) is set for XJA interrupts, the ICU evenly distributes interrupts to CPUs that are available (defined by bits 30, 28, 26, and 24) and can be interrupted (defined by bits 31, 29, 27, and 25). If bit 19 is cleared, the ICU broadcasts interrupts.

If bit 19 (round-robin interrupt) is set for service processor unit (SPU) interrupts and the interrupt packet has CPU_ID_H[03:00] cleared, interrupts are not sent to the CPUs.

CPU_ID_H[03:00] indicates which CPUs (for SPU interrupts) the ICU interrupts. The ICU delivers the following interrupts to the selected EBox:

- Console TRX at IPL 14(hex)
- Console TTX at IPL 14(hex)
- Console storage receive at IPL 17(hex)
- Console storage transmit at IPL 17(hex)
- Power failure at IPL 1E(hex)
- JBox memory errors at IPL 1D(hex)
- XJA fatal errors at IPL 1D(hex)
- CPU interprocessor interrupts at IPL 14(hex)
- XJA interrupts at IPLs 14(hex) through 17(hex)

8.3.1 Interprocessor Interrupts

Using the interprocessor interrupt register, a CPU can interrupt one or more CPUs, including itself, to support applications involving more than one processor. Using interprocessor interrupts, processors can share data or send information from one processor to another. The CPU initiates an I/O register write to the interprocessor interrupt register, and sets the CPU's corresponding bits. The ICU encodes the CPU interrupt (IPL 16) and sends the interrupt to the CPUs in which the bits have been set.

A write request to this register initiates an interprocessor interrupt to the CPUs coded in bits [03:00]. If these bits are cleared, a CPU does not receive an interrupt.

If multiple CPUs are writing to the interprocessor interrupt register before arbitration, the ICU delivers interprocessor interrupts to CPUs in which bits have been set. For example, CPU0 writes bit 3 and CPU1 writes bit 2. ICU sends an interprocessor interrupt to CPU2 and CPU3 when the interprocessor interrupt register wins arbitration.

SPU and the CPUs can read the interprocessor interrupt register. If the content of this register is nonzero, the arbiter has a pending interrupt. The ICU delivers the interprocessor interrupt to the destination CPUs and clears the register. Figure 8–2 shows the interprocessor interrupt register, and Table 8–2 lists the fields and descriptions.

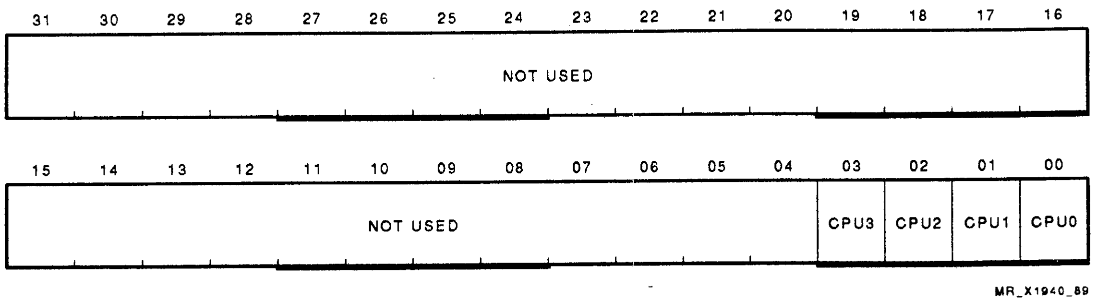


Figure 8–2 Interprocessor Interrupt Register Format

Table 8–2 Interprocessor Interrupt Register Field Descriptions

Field	Description
00	CPU0
01	CPU1
02	CPU2
03	CPU3
31:04	Reserved

8.3.2 XJA Interrupts

XJA-generated interrupts result from a data or protocol error on the JXDI, on the XMI bus, or in the XJA. Some XJA errors are classified as fatal, that is, operation of the XJA is unpredictable and cannot respond to CPU commands. If the error is nonfatal, operation of the XJA is predictable and the XJA can respond to CPU commands; however, some CPU transactions may fail. XJA-detected nonfatal interrupts are at IPL 17(hex).

If the error detected by the XJA is fatal, the XJA asserts the XJA_FATALERR line on the JXDI. The SCU responds by interrupting the system CPU at IPL 1D(hex). The CPU then proceeds to service the interrupt by executing the appropriate service routine.

If the error detected is nonfatal, the XJA sends an interrupt packet to the ICU. The interrupt packet is a one-cycle packet specifying IPL 17(hex). The CPU responds by reading the SCB offset IPL register in the XJA, corresponding to the IPL of the interrupt (IDENT7) register. The IDENT7 register is read by a CPU using a read transaction. Reading the IDENT7 register causes the XJA to return the contents of the error SCB (ERRSCB) offset register to the ICU. The ERRSCB register contains the vector required by the CPU to locate the appropriate service routine. When the system CPU receives the SCB offset, it proceeds to service the interrupt.

8.3.2.1 XJA Vectored Interrupts

The XJA delivers vectored interrupts to the ICU using the XJA interrupt packet format on the JXDI. The interrupt packet contains the IPL in bits [05:04]. The JDAX MCA encodes bits [05:04] on JDA_IRC_INTR_H[01:00]. IRCX decodes JDA_IRC_INTR_H[01:00] and sends the interrupt to an EBox. IRCX uses the CPU configuration register (not the XJA) to determine which EBox to interrupt.

As soon as the EBox IPL is lower than the requested interrupt IPL, the EBox issues a read request to one of the four XJA SCB offset registers corresponding to the four IPL levels. This happens when the EBox receives the interrupt request from the central system interrupt arbiter (CSIA). The data returned to this read request is the offset to the SCB where the EBox can find the vector that points to the interrupt service routine.

8.3.2.2 Fatal XJA Interrupts

The XJA asserts an error signal in the SCU to initiate an interrupt to service the error. The XJA error signal is asserted when the following XJA or XMI errors occur:

- Fatal XMI errors

  Node reset
  Node halt
  XMI fault
  Write error interrupt
  XMI power failure
  XMI arbitration timeout

- Fatal XJA errors

  XCE macrocell array (MCA) internal state machine error
  ICU buffer count error
  CPU request overrun
  JXDI receive buffer overrun
  JXDI receive command error
  XJA CMOS to bipolar interconnect (CBI)

The SCU delivers the requested interrupt IPL of 1D(hex) to the selected EBox. The EBox calculates the SCB offset and redirects the system to an XJA fatal error routine. XJA fatal errors do not necessarily indicate the XJA is incapable of responding to further CPU requests.

8.3.3 XMI Interrupts

XMI interrupts are generated by an XMI device requiring service by the system CPU. XMI interrupts fall into two classes: normal interrupts and implied vector interrupts. Normal XMI interrupts require communication between the XJA and the interrupting device to obtain the SCB offset vector required to service the interrupt. In an implied vector interrupt, the SCB offset vector is implied by the type of interrupt; therefore, no communication is required with the interrupting device.

If the XMI interrupt is a normal interrupt, the XJA sends an interrupt packet to the ICU, specifying the IPL of the interrupt. The CPU responds by reading the contents of the SCB offset IPL register (identify [IDENT] register) in the XJA and by corresponding to the IPL of the interrupt. The XJA responds to the CPU read request by initiating an IDENT transaction to the interrupting node on the XMI bus. The interrupting node returns an IDENT response to the XJA containing the SCB offset vector. The XJA passes the offset vector to the CPU as read data return for the CPU read transaction. The CPU then uses the vector to locate the required service routine.

If the XMI interrupt is an implied vector interrupt, the XJA determines whether the interrupt is fatal or nonfatal. If the interrupt is fatal, the XJA_FATALERR line on the JXDI is asserted, causing the SCU to interrupt the CPU at IPL 1D(hex). The CPU then proceeds to service the interrupt.

If the interrupt is nonfatal, the XJA transfers the interrupt to the CPU by sending an interrupt packet at IPL 16(hex) to the ICU. The CPU responds by reading the IDENT register corresponding to the interrupting IPL (IDENT6 register). The XJA responds by returning an offset vector of 80(hex), which the CPU uses to locate the appropriate service routing.

NOTE
An XMI IDENT transaction was not required for the implied vector interrupt.

The XJA error handling implementation centers on its error status registers stored in the XJA module. The XJA can detect and recover from most transient bus transmission errors on the JXDI and the XMI bus.

The XJA can send two types of interrupts: vectored interrupts at IPL 14(hex) through 17(hex) and fatal interrupts at IPL 1D(hex).

The XJA reports errors to the system where it recovered, using IPL level 17(hex). Nonrecoverable errors are separated into two categories: nonfatal (vectored interrupts) and fatal.

8.3.3.1 XMI Device Vectors

The vectors returned by native XMI devices (including XBI vectors returned for XBI error interrupts) in response to XMI IDENT transactions (generated in response to a read from an XJA SCB offset register), for which an XMI node has generated the interrupt request, are described in Figure 8–3 and Table 8–3.

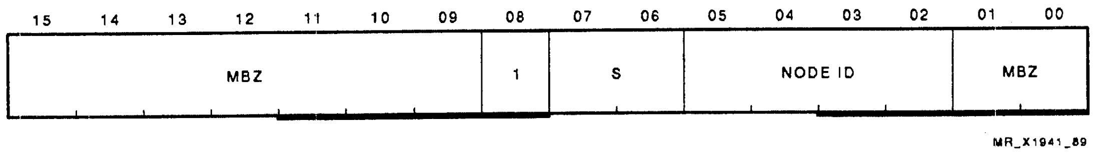


Figure 8–3 XMI Device Vector Format

Table 8–3 XMI Device Vector Field Descriptions

Field	Description
Node ID	Identifies the XMI node ID of the interrupting node (0–15).
S	Contains one of four possible interrupt vectors per node.
1	Initialized to 1 to ensure that the device vector resides above the processor area of the SCB (the first 256 bytes of the SCB).
MBZ	Must be 0 to specify the first page of vector offsets in the SCB.

The system assigns vector values to each vector register (four possibilities) that exists on XMI devices capable of generating interrupt requests during initialization.

The EBox servicing an XMI interrupt only appends the XJA number (0 through 3) to bits [10:09] of the returned XMI vector if bits [15:09] of the returned vector are 0.

8.3.3.2 BI Device Vectors

The second type of interrupt vector is the format returned by BI nodes. The major difference between this vector format and the XMI format is that bits [15:09] are nonzero and are assigned a value by the system software during initialization. This offset value, contained in the vector offset register (BVOR) on the XBI is concatenated with the vector value returned by a BI node, bits [08:02], providing that bits [13:09] of the BI vector are equal to 0 (nonoffsettable bus adapter).

This new value is returned to the XJA during an XMI IDENT cycle generated in response to a read from an XJA SCB offset register (for which a BI node has generated the interrupt request). If bits [13:00] of the BI vector are nonzero, the vector is not concatenated with the VOR register and is passed unchanged to the XJA. The assumption is that a vector with bits [13:09] set to nonzero is generated by a BI node with an offsettable bus.

BI device-initiated interrupts return vectors (Figure 8–4 and Table 8–4) in response to an XMI IDENT transaction.

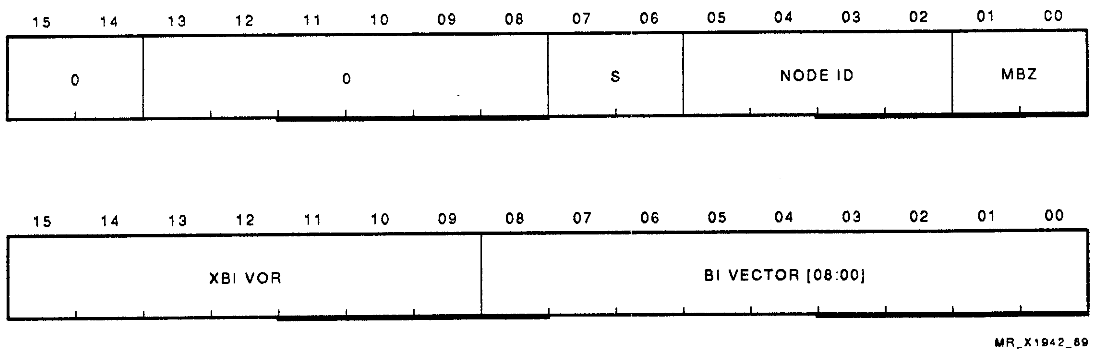


Figure 8–4 BI Device Interrupt Vector Format

Table 8–4 BI Device Interrupt Vector Field Descriptions

Field	Description
Node ID	Identifies the BI node ID of the interrupting node (0–15).
S	Contains one of four possible interrupt vectors per node.
VOR	Must be a nonzero software-assignable offset value to index to the SCB with a unique vector for multiple BI devices. The VOR bits [15:09] may be supplied by the XBI (as noted above).

The VOR register is necessary, since an XMI bus can support multiple XBI nodes, where the same device may exist on multiple BIs. Since some BI nodes may have fixed vectors that are unchangeable by software, the VOR is used as a means of ensuring that multiple BI devices with fixed vectors have a unique entry point to the SCB.

The system assigns unique XBI VOR values that do not conflict with the native XMI device SCB pages. That is, the value of XBI VORs must be different and must possess a greater value than the number of XJAs in a given system minus 1.

The EBox servicing an XMI interrupt uses an unmodified, returned XMI vector only if bits [15:09] of the returned vector are nonzero.

8.3.4 Console Interrupts

The console subsystem contains a BI MicroVAX chip, an RD54 disk drive, a TK50 tape drive, a console terminal, a remote diagnosis port, and block storage devices.

The ICU distinguishes between terminal operations and block storage device operations by decoding the command field in the interrupt packet that the SPU sends to the ICU. (The SPU sends an interrupt packet to the ICU through the CTLD MCA.) When the JDCX MCA decodes the SPU command and finds that the command is an interrupt, it generates an IPL and sends the command and IPL to the ICU for interrupt arbitration.

Transfers to console devices are made through internal processor registers and device drivers in I/O space. Direct memory transfer is not made between a CPU and a console device; instead, the SPU sends an interrupt to the CPU, which executes an interrupt service routine and reads the appropriate registers in the SPU internal register area.

The console subsystem contains four internal SPU processor registers for communication with a console terminal:

Console receive control and status register (RXCS)
Console receive data buffer register (RXDB)
Console transmit control and status register (TXCS)
Console transmit data buffer register (TXDB)

The SPU supports three console terminal devices: a local terminal (CTY), a remote terminal (RTY), and the logical console port. The SPU uses the RXCS and RXDB registers to enable the terminals and to hold the data entered at the terminal. The SPU loads RXCS and RXDB registers and sets the interrupt enable bit in RXCS. The SPU builds the interrupt packet with the interrupt command (console receive, IPL 14[hex]) and CPU ID, and sends the packet to the ICU.

The SPU uses the TXCS and TXDB registers to determine which of the three terminals (CTY, RTY, and console port) are enabled and to hold the data to be transferred to the terminal. The SPU loads TXCS and TXDB registers and sets the interrupt enable bit in TXCS. The SPU builds the interrupt packet with the interrupt command (console transmit, IPL 14[hex]) and CPU ID, and sends the packet to the ICU.

The console subsystem contains four internal SPU processor registers for communication with a console storage device:

Receive function request register (RXFCT)
Receive parameters register (data or address) (RXPRM)
Transmit function request register (TXFCT)
Transmit parameters register (data or address) (TXPRM)

The SPU uses the RXFCT and RXPRM registers for system functions and diagnostic functions. The SPU loads the RXFCT and RXPRM registers with a function and parameter and sets the interrupt bit in the RXFCT register. The SPU builds the interrupt packet with an interrupt command (console storage receive, IPL 17[hex]) and CPU ID. RXFCT functions include receive block of data, halt CPU, send error log entry, get error log entry, and set keep-alive state.

The SPU uses the TXFCT and TXPRM registers for system functions and diagnostic functions. The SPU loads the TXFCT and TXPRM registers with a function and parameter and sets the interrupt bit in the TXFCT register. The SPU builds the interrupt packet with an interrupt command (console storage transmit, IPL 17[hex]) and CPU ID. Functions include transmit block of data, keep alive, reboot system, boot secondary system, reset the XJA, halt CPU, set interrupt mode, margin clock, margin power, and get IPAMM and PAMM configuration.

The console subsystem also sends interrupts for power failure, CPU halt, and EBox keep alive.

8.4 Software Interrupts

Software-requested interrupts are handled entirely in the EBox and are assigned IPLs 01(hex) to 0F(hex). This section describes the registers associated with the software interrupts.

8.4.1 Interval Counter Interrupt

Each CPU contains an interval counter implemented in the EBox hardware. The interval clock is used for accounting, for time dependent events, and to maintain the software time and date. The interval counter is incremented at 1-microsecond intervals. The following three registers control the interval count, the interval count register (ICR), the next interval count register (NICR), and the interval count control and status register (ICCS):

- **ICR** — A read-only register that gets incremented once every microsecond. When an overflow occurs or a carry-out of bit 31 ICR automatically reloads from NICR (and if interrupts are enabled), an interrupt is generated.
- **NICR** — A write-only register that contains the value to be loaded into ICR when it overflows.
- **ICCS** — Contains control and status information for the interval clock. Figure 8–5 and Table 8–5 show the format for the ICCS.

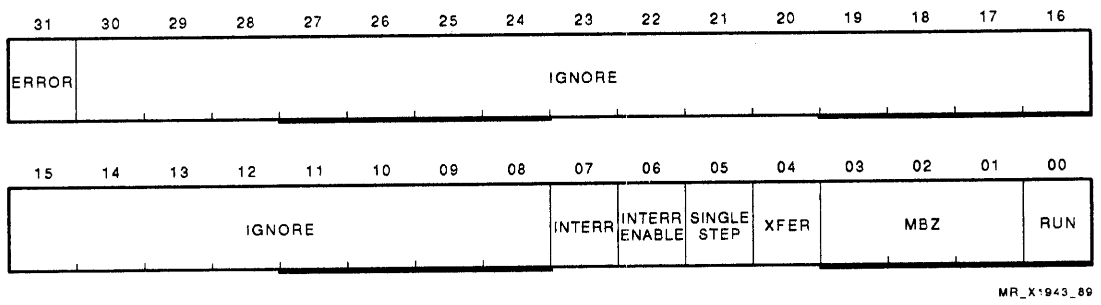


Figure 8–5 ICCS Register Format

Table 8–5 ICCS Register Field Descriptions

Bit	Name	Description
00	Run	When bit 0 is set, ICR increments each microsecond. When bit 0 is cleared, ICR does not automatically increment.
04	Transfer	When a 1 is written to this bit, NICR is transferred to ICR.
05	Single step	If run is clear, ICR increments by 1 each time this bit is set.
06	Interrupt enable	When bit 6 is set, an interrupt request is generated each time ICR overflows. When bit 6 is clear, an interrupt is not requested. Processor initialization clears this bit.
07	Interrupt	Set by hardware every time ICR overflows. If interrupt enable (bit 6) is set, then an interrupt is generated. Writing a 1 to this bit (with MTPR) clears the bit and reenables the clock tick interrupt.
31	Error	If interrupt is set when ICR overflows, then this bit is set. An error indicates a missing clock tick.

To use the interval clock, the negative value of the desired interval is loaded in NICR. The run, transfer, and interrupt fields are enabled in ICCS. This loads the interval value in ICR and starts the interval count. When the value loaded in ICR is incremented beyond 32, an interrupt occurs. The interrupt routine must clear the interrupt bit in ICCS so that if the interrupt is not serviced when the next ICR overflow occurs, then the error bit in ICCS is set.

8.5 Processor Interrupt Registers

The EBox registers are in the STRAMs or the EBox hardware. Some registers store a copy in both places. The STRAM data is sent to the data path through the usual source methods. The hardware registers enter the data path through the EBox INT unit shifter. The microcode can select the STRAM data on source 1 or source 2. The hardware registers must be selected through the source 2 inputs. Some hardware registers are write-only.

8.5.1 Asynchronous System Trap Level Register

The asynchronous system trap level (ASTLVL) register stores the most privileged access mode for which an asynchronous system trap (AST) is pending. An REI checks the ASTLVL register and if the value of the access mode is greater than or equal to the access mode where REI is returning, then a software interrupt is generated to service the AST. If REI is returning to the interrupt stack, an interrupt is not generated to service the AST.

The ASTLVL register is implemented in the EBox STRAM and is an architecturally defined register. Processor initialization sets ASTLVL to access mode 4 (no pending AST). Figure 8–6 and Table 8–6 show the register format for ASTLVL.

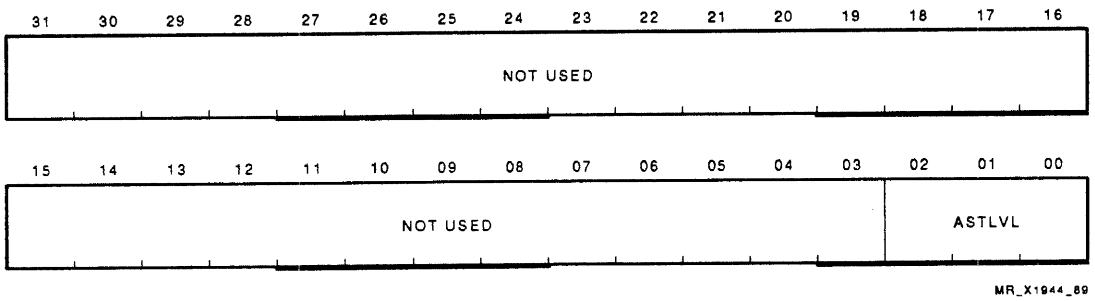


Figure 8–6 ASTLVL Register Format

Table 8–6 ASTLVL Register Field Descriptions

Value	Description
0	AST pending for access mode 0 (kernel).
1	AST pending for access mode 1 (executive).
2	AST pending for access mode 2 (supervisor).
3	AST pending for access mode 3 (user).
4	No pending AST.

8.5.2 Software Interrupt Summary Register

The software interrupt summary register (SISR) records pending software interrupts and contains 1s in the bit positions corresponding to levels where software interrupts are pending.

The SISR register is implemented in the EBox STRAM and contains a copy of the register in the EBox hardware. The EBox checks this register against the current IPL to determine whether an interrupt should be generated. The microcode uses the EBox INT unit encoder circuit to set and to clear bits in the SISR. The microcode must allow adequate cycle time after writing the register for the hardware to check the new value.

SISR is an architecturally defined register and is cleared on processor initialization. Figure 8–7 shows the register format for SISR.

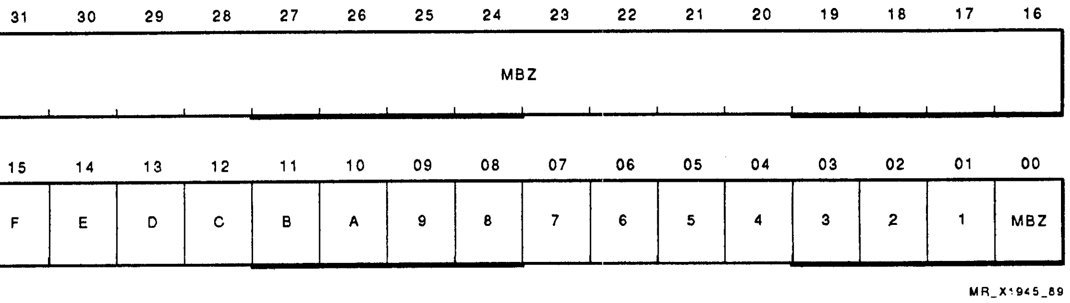


Figure 8–7 SISR Register Format

8.5.3 Software Interrupt Register

The software interrupt request register (SIRR) is a write-only, 4-bit register used for requesting software interrupts. Each bit position in the SIRR corresponds to the associated IPL of the software interrupt.

There is no physical register for the SIRR; instead, the microcode makes an entry in the SISR. This is done by the encoder logic in the EBox INT unit. The SIRR value is loaded in the encoder and then passed to the SISR. The correct bit is set and the SISR is reloaded with the new value. The EBox hardware compares the IPL against the value of the highest software interrupt and issues an interrupt when the IPL is lower than the value in the SISR.

SIRR is an architecturally defined register. Figure 8–8 shows the format for the SIRR.

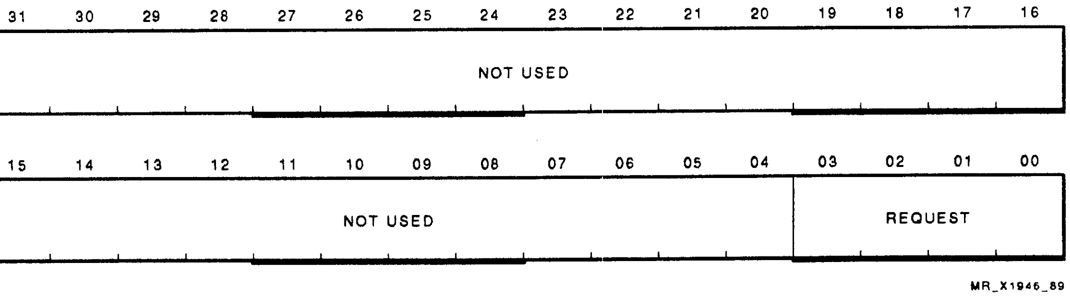


Figure 8–8 SIRR Register Format

8.5.4 Interrupt Priority Level Register

The interrupt priority level (IPL) register is five bits wide and contains the same information in the IPL field of the processor status longword (PSL [20:16]). This register allows setting the IPL without writing the entire PSL. When the microcode writes this register, it must allow enough cycle time for the IPL to be set before setting instruction done; otherwise, an interrupt might be realized an instruction late.

The IPL register is an architecturally defined register and is set to 1F(hex) at processor initialization. Figure 8–9 shows the format of the IPL register.

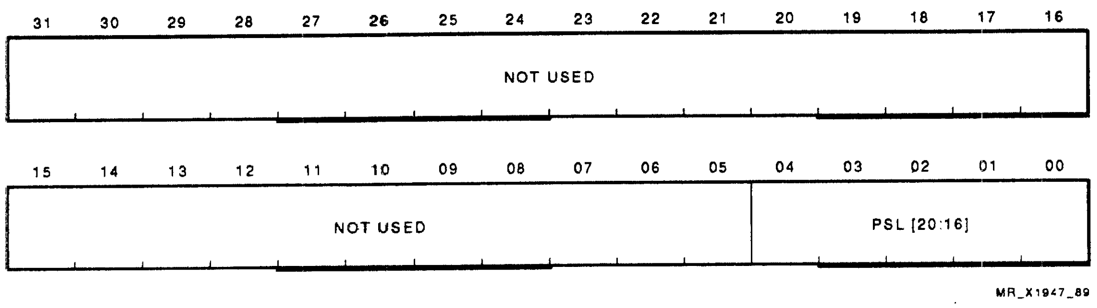


Figure 8–9 IPL Register Format

8.5.5 System Control Block Base Register

The system control block base (SCBB) register contains the physical address of the SCB. This register resides in the EBox STRAM. It is accessed by the microcode to provide the base address to the SCB. The SCB provides the information on handling exceptions and interrupts. It contains the address of the handling routine and the stack pointer to use while servicing an interrupt exception.

The SCBB is an architecturally defined register. On processor initialization, the contents of the SCBB are unpredictable. Figure 8–10 shows the format of the SCBB.

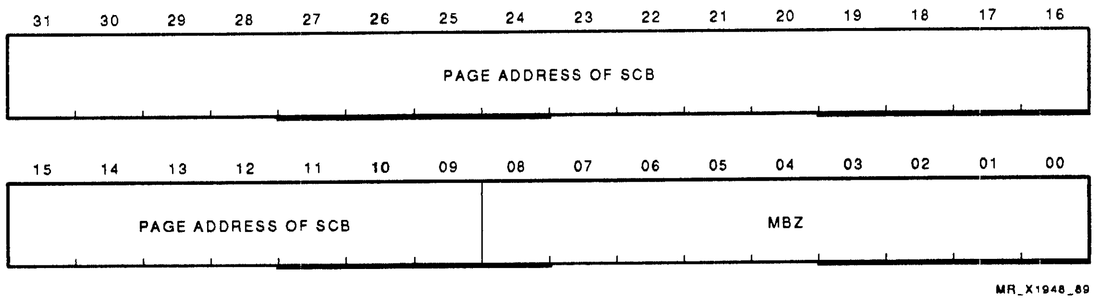


Figure 8–10 SCBB Register Format

8.5.6 Interrupt Control Register

The interrupt control (INTRCTRL) register is in I/O space at location 3E200000(hex) and is initialized to 0. Figure 8–11 and Table 8–7 provide the register format and field definitions.

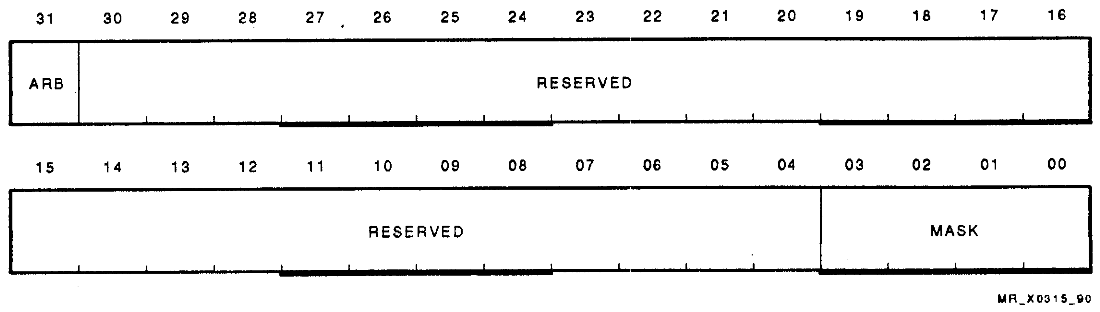


Figure 8–11 INTRCTRL Register Format

Table 8–7 INTRCTRL Register Field Descriptions

Field	Description
Arbitration	If this bit is set, the SCU JBox arbitration mode is set to round-robin. If this bit is clear, the SCU broadcasts interrupts to the CPUs enabled by the mask bits.

Mask	Enables the broadcast of interrupts to a specific CPU. If a bit is set, it allows the interrupt broadcast. The following specifies the field encoding:
	
	0000 = No interrupts.
	xxx1 = CPU0 receives interrupts.
	xx1x = CPU1 receives interrupts.
	x1xx = CPU2 receives interrupts.
	1xxx = CPU3 receives interrupts.

8.5.7 Interprocessor Interrupt Control Register

The content of the interprocessor interrupt control (IPINTRCTRL) register is CPU-specific and directs an interrupt from one processor to another. The register address is 3E200004(hex) in I/O space. Figure 8–12 and Table 8–8 provide the register format and field definitions.

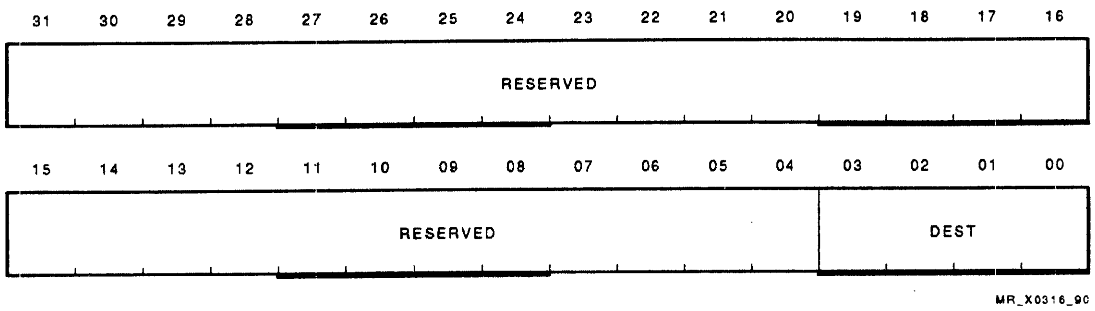


Figure 8–12 IPINTRCTRL Register Format

Table 8–8 IPINTRCTRL Register Field Descriptions

Field	Description

Destination	Indicates which CPU receives the interrupt. When the interrupt is delivered, the bits are cleared. The following specifies the field encoding:
	
	0000 = No interrupts.
	xxx1 = CPU0 receives interrupts.
	xx1x = CPU1 receives interrupts.
	x1xx = CPU2 receives interrupts.
	1xxx = CPU3 receives interrupts.

8.6 Exceptions

An exception is an event detected by hardware that causes a change in the usual flow of instruction execution. An exception is caused by the execution of an instruction as opposed to an interrupt, which is caused by an activity in the system independent of the current process. There are three types of hardware exceptions: traps, faults, and aborts. Table 8–9 lists the exceptions.

NOTE
Machine check exceptions are handled differently from the other exceptions and are described in Section 8.6.1.1.

Table 8–9 Exceptions

Name	Source	Type	SCB Offset
Integer overflow	EBox	Trap	34
Integer (divide by 0)	EBox	Trap	34
Floating overflow	EBox	Fault	34
Floating (divide by 0)	EBox	Fault	34
Floating underflow	EBox	Fault	34
Decimal (divide by 0)	EBox	Trap	34
Decimal overflow	EBox	Trap	34
Subscript range	EBox	Trap	34
Reserved operand	EBox	Fault	18
Reserved address mode	IBox	Fault	1C
Access violation	EBox, IBox	Fault	20
Translation not valid	EBox, IBox	Fault	24
Trace trap	EBox	Trap	28
Is not valid	EBox	Halt	None
Kernel stack pointer not valid	EBox	Abort	08
Machine check	EBox	Abort	04
Subset emulation	EBox	Trap	C8

The arithmetic exceptions use the same SCB vector. A code is pushed on the stack to identify the particular exception. Table 8–10 lists the arithmetic exceptions and their appropriate codes.

Table 8–10 Arithmetic Exception Codes

Code	Exception Type
Traps	
1	Integer overflow
2	Integer (divide by 0)
4	Decimal (divide by 0)
6	Decimal overflow
7	Subscript range
Faults	
8	Floating overflow
9	Floating (divide by 0)
A	Floating underflow

8.6.1 EBox Exceptions

When an exception occurs, the PSL and the PC are pushed onto the stack at the time of the exception. The EBox dispatches the microcode to execute an exception service routine and then exits the service routine by executing the REI instruction. The PC that is pushed onto the stack depends on the type of exception that occurs. The following describes which PC is saved with each exception type:

- **Fault exceptions** — Cause the PC of the faulting instruction to be pushed onto the stack. When the fault is corrected, the faulting instruction is executed again.
- **Trap exceptions** — Cause the PC of the next instruction to be pushed onto the stack. Instructions that cause traps are not reexecuted when the service routine is completed.
- **Abort exceptions** — Cause a PC in the middle of the instruction to be pushed onto the stack. Aborts are not restartable.

8.6.1.1 Machine Check

A machine check is an exception that is reported when the CPU, or an external adapter, detects an internal error. The basic philosophy of the machine check handler is to keep as much of the system running as possible.

In previous VAX systems, CPU errors were reported to the EBox, and the EBox invoked the machine check handlers in an attempt to recover from the error. In the VAX 9000, errors are reported to the SPU, and the SPU attempts to perform the recovery.

The SPU recovery process includes stopping the system clocks, scanning in the state of the CPU, and selecting a recovery method based on the characteristics of the error. If the recovery method is successful, the SPU restarts the system clocks and the process running when the error occurred. The system records the error in the error log using error information provided by the SPU.

A machine check occurs when the SPU error recovery is unsuccessful. The SPU places the machine check stack frame in the system in memory and restarts the EBox microcode at the system machine check handler. The EBox machine check handler then makes the final decision on error recovery. Figure 8–13 shows the machine check stack that is built by the SPU.

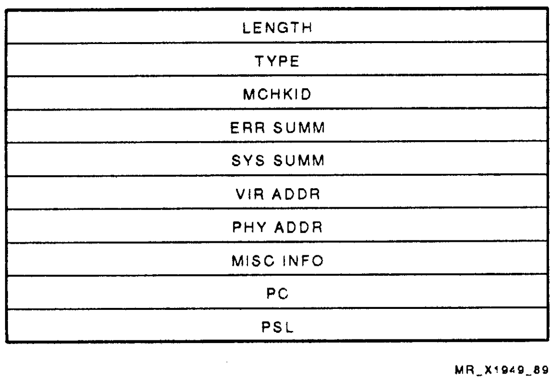


Figure 8–13 Machine Check Stack Frame

I/O read nonexistent memory (NXM) machine checks are handled differently than machine checks handled by the SPU. The EBox microcode independently generates the I/O read NXM machine check and performs the recovery process without intervention by the SPU. This allows these errors to be handled without stopping the system clocks. Figure 8–14 shows the machine check stack that is built by the EBox for I/O read NXM machine checks.

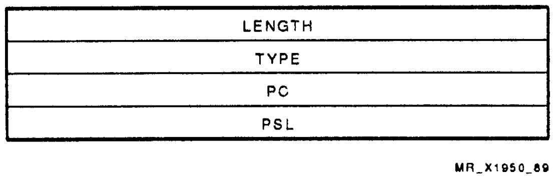


Figure 8–14 NXM Machine Check Stack Frame

8.6.2 MBox Exceptions

Memory management traps may be generated in the MBox or IBox and are limited to two principal exception types: translation-not-valid faults and access-control-violation faults.

Access-control-violation faults occur when the protection field of the page table entry (PTE) indicates that the intended page reference in the specified access mode is illegal.

A translation-not-valid fault occurs when a read or write reference is attempted through an invalid PTE.

Both of these faults are reported to the EBox. The EBox reads the fault parameter register that contains the faulting virtual address (VA). Figure 8–15 shows the layout of the fault parameter register and Table 8–11 describes the bit fields.

Both of these faults are reported to the microcode and supply a microtrap vector.

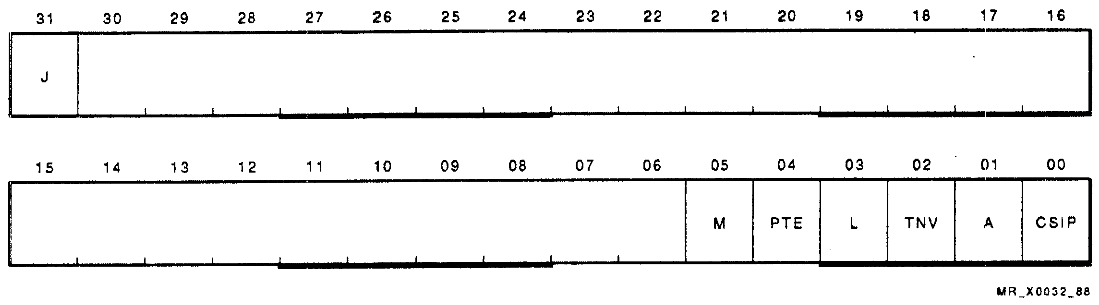


Figure 8–15 MBox Fault Parameter Register

Table 8–11 MBox Fault Parameter Register Field Descriptions

Bit	Name	Definition
31	J	Indicates that the OPU was jammed.
30:06	–	Reserved.
05	M	Indicates a modify intent reference.
04	PTE	Indicates a PTE reference violation.
03	L	Indicates a length violation.
02	TNV	Indicates translation not valid.
01	A	Indicates an access violation.
00	CSIP	Indicates a cache sweep in progress.

8.6.3 IBox Exceptions

The IBox decodes reserved opcode faults, reserved addressing mode faults, and memory management faults that occur in conjunction with transactions on the instruction buffer interface and the OPU interface.

The reserved opcode fault is passed to the EBox in the same manner as other fork addresses. The EBox forks to the exception location in a typical manner and the EBox microcode processes the fault. The SCB offset for a reserved opcode fault is 10(hex).

The reserved addressing mode fault is detected in the IBox, and the specifier causing the fault is loaded in the EBox queue along with the pointer to the specifier. The fault is detected by the EBox when the microcode selects the operand of the instruction that contains the fault.

When the EBox microcode selects the operand, the microsequencer directs the code to the handler location where it is handled as a typical microinstruction sequence. The functional units propagate their previous data to retire. The result queue is entirely empty before the handler routine retires results. The SCB offset for reserved addressing mode faults is 1C(hex).

The instruction buffer memory faults are passed to the EBox if the data that faults is accessed by the decode units of the IBox. The EBox provides the microaddress for the fault.

The EBox is notified of read operand memory faults by the MBox. The EBox does not take action on the fault until the operand is selected. The issue unit directs the microsequencer to signal the fault.

Write operand faults are kept by the MBox until the EBox writes the data. The EBox must then flush the pipeline of further instructions.

The SCB offsets for the memory faults are 20(hex) for access violations and 24(hex) for translation not valid.

8.6.4 System Control Block

The SCB contains the vectors used to dispatch software and hardware interrupts and exceptions. The starting physical address of the SCB is in the SCBB.

Table 8–12 shows the SCB for VAX 9000 systems. The table specifies the vectors, vector names, type of interrupt or exception, and related notes for each vector.

Table 8–12 System Control Block Table

Vector	Name	Type	Notes
00	Passive release	Interrupt	–
04	Machine check	All	Error information.
08	Kernel stack not valid	Abort	IPL is raised to 1F and interrupt stack is used.
0C	Power failure	Interrupt	IPL is raised to 1E.
10	Reserved instruction	Fault	Reserved or privileged opcodes.
14	Customer	Fault	XFC instruction.
18	Reserved operand	Fault	Reserved operand.
1C	Reserved address	Fault	–
20	Access violation	Fault	–
24	Translation not valid	Fault	Translation not valid.
28	Trace pending	Fault	Trace pending fault.
2C	Breakpoint	Fault	Breakpoint instruction.
34	Arithmetic	Trap, fault	A type code is pushed onto a stack.
40	CHMK	Trap	–
44	CHME	Trap	–
48	CHMS	Trap	–
4C	CHMU	Trap	–
50	XJA error	Fault	XJA fatal error (IPL is 1E).
84	Software level 1	Interrupt	IPL is 1.
88	Software level 2	Interrupt	IPL is 2 (used for AST).
8C–BC	Software levels 3–F	Interrupt	Vector corresponds to IPL.
C0	Interval timer	Interrupt	IPL is 18.
C8	Subset emulation	Trap	FPD clear.
CC	Suspend emulation	Fault	FPD set.
F0	Console storage receive	Interrupt	IPL is 17.
F4	Console storage transmit	Interrupt	IPL is 17.
F8	Console terminal receive	Interrupt	IPL is 14.
FC	Console terminal transmit	Interrupt	IPL is 14.
100–1FC	Adapters	Interrupt	IPL is 14 to 17.
200–5FC	Devices	Interrupt	IPL is 14 to 17.

A
Related Documents

Table A–1 lists the documentation related to the VAX 9000 family of systems with the full title and order number.

Table A–1 VAX 9000 Family Documents

Title	Order Number
VAX 9000 Family Technical Description Kit	EK-KA900-TD
VAX 9000 Family System Description	EK-KA90S-TD
VAX 9000 Family VBox Technical Description	EK-KA90V-TD
VAX 9000 Family EBox Technical Description	EK-KA90E-TD
VAX 9000 Family IBox Technical Description	EK-KA90I-TD
VAX 9000 Family MBox Technical Description	EK-KA90M-TD
VAX 9000 Family SCU Technical Description	EK-KA90J-TD
VAX 9000 Family Clock Subsystem Technical Description	EK-KA90K-TD
VAX 9000 Family XJA Technical Description	EK-KA90A-TD
VAX 9000 Family Power System Technical Description	EK-KA90P-TD
VAX 9000 Family SPU Technical Description	EK-KA90C-TD
VAX 9000 Family Site Preparation Guide	EK-9000S-SP
VAX 9000 Model 200 Installation Guide	EK-9200I-IN
VAX 9000 Model 400 Installation Guide	EK-9400I-IN
VAX 9000 Family Maintenance Document Kit: Volumes 1, 2, and 3	EK-KA900-MG
VAX 9000 Model 200 User Document Kit	EK-9200U-UP
VAX 9000 Family Operator Reference Card	EK-9000O-RC
VAX 9000 Model 200 Hardware User Guide	EK-9201U-UG
VAX 9000 Family Console Command Description	EK-9000C-CD
VAX 9000 Family System Introduction	EK-9000S-SI

Table A–1 (Cont.) VAX 9000 Family Documents

Title	Order Number
VAX 9000 Model 400 User Document Kit	EK-9400U-UP
VAX 9000 Family Operator Reference Card	EK-9000O-RC
VAX 9000 Model 400 Hardware User Guide	EK-9401U-UG
VAX 9000 Family Console Command Description	EK-9000C-CD
VAX 9000 Family System Introduction	EK-9000S-SI

Glossary

ACU — array control unit
The MCA III logic of the memory system. Consists of two memory data paths (MDPs), one memory control DRAM (MCD), and one main memory control (MMC), all of which are located in the system control unit (SCU).

APG — address pattern generator
A linear shift feedback register in the DRAM control and address gate array that generates pseudorandom address patterns for the built-in self-test (BIST).

ASD — automatic shutdown
The normal system power-down initiated through the power control subsystem (PCS), utility port conditioner (UPC), or power front end (PFE).

BCAI — BCI adapter interface
A 133-pin ZMOS chip that functions as a buffer between user-designed processors, memories, and adapter modules and the VAXBI bus.

BCI3 — BCI-to-MicroVAX II bus interface
A 132-pin chip that connects the integrated circuit interconnect bus of the MicroVAX processor to the VAXBI bus through the VAXBI BIIC.

BIIC — backplane interconnect interface chip
A 133-pin ZMOS chip that serves as the primary interface between the VAXBI bus and a master or slave port interface.

BIST — built-in self-test
A series of diagnostic functions that are designed into the hardware to provide go-no-go testing. Typically, the initial power-up of a device invokes this type of test.

BPC — branch prediction cache
A 1K virtual cache of target addresses (PCs), ready to use if the branch prediction unit (BPU) decides to branch. For performance reasons, the BPC is not flushed.

BPU — branch prediction unit
Detects branch instructions, predicts branch direction, and redirects the instruction buffer to fetch instructions from the new I-stream.

BVP — BI VAX port
A standard software architecture that defines the data structures and protocols required to move information between a VAX host processor and an adapter across the VAXBI bus.

CCU — cache consistency unit
Part of the system control unit (SCU). Responsible for content consistency between multiple CPU caches. Maintains cache consistency with duplicate cache tag stores — one for each CPU.

CDB — configuration database
Scan access and configuration database containing scan ring size, scan ring addresses, and scan ring locations, STRAM and STREG structure definition, and signal names.

CDC — clock distribution chip
A custom VLSI chip in the center of a multichip unit (MCU). Distributes clock signals to macrocell arrays (MCAs) and self-timed RAMs (STRAMs).

CIO mode — console I/O mode
During this operating mode, the service processor unit (SPU) interprets characters entered at the console terminal as SPU (console) commands. *See also* PIO mode.

CIXCD adapter
A high-performance, intelligent, I/O interface that connects to the Computer Interconnect (CI) VAXcluster systems. Allows communication over both CI paths simultaneously and in any order. Provides up to a four-fold increase in I/O performance over previous CI interfaces.

commander
An XMI term. The node that initiated the transaction in progress. In any write transaction, the commander is the node that requested the write. For read transactions, the commander is the node that requested the data.

The commander node is maintained for the duration of the transaction, although it might appear that the commander changes in some cases. For example, a commander initiates a read transaction. The responder (data source) initiates the return data transfer, but the node that requested the data is still the commander.

CPU — central processing unit
Refers to the EBox, IBox, and MBox.

CSIA — central system interrupt arbiter
Located in the system control unit (SCU). Arbitrates interrupt requests from I/O devices and other CPUs.

CSSE connector
CSSE analyzer match (CAM) module.

DAC — daughter array card
Contains 160 dynamic RAMs, having a capacity of 16 Mbytes of MOS memory.

DCA — DRAM control and address
An AMCC Q3500 gate array, located on the memory array card (MAC), that buffers the DRAM control signals from the main memory control (MMC) and latches the row and column address.

DEMNA controller
A high-performance, XMI-compatible, Ethernet controller. Contains a CVAX processor for port command and data flow control and 512 Kbytes of on-board memory for command and data packet buffering.

DDP — DRAM data path
An AMCC Q3500 gate array. Each memory array card (MAC) has four DDPs that buffer data to and from the DRAMs.

DIV — EBox divide unit
EBox functional unit that performs integer and floating-point divide operations.

DPG — data pattern generator
A linear shift feedback register that produces pseudorandom data patterns for the built-in self-test (BIST).

DWMBB interface
VAXBI-to-XMI bus interconnect for VAX 9000 systems. Allows connection of up to 14 VAXBI buses to the VAX 9000.

EBox — execution box
Serves as the CPU execution unit. Accepts instruction source and destination data from the IBox and MBox. Provides integer, floating-point, multiply, and divide operations.

ECC — error correction code
Logic in the memory data path MCA to detect double-bit errors and to correct single-bit errors in a 39-bit data field (32 bits data, 7 bits ECC).

ECL — emitter-coupled logic
An CI logic design using transistors that are connected at the emitter. Results in high speed, low circuit density, and high power dissipation.

ESD — emergency shutdown
A system power-down initiated through the power control subsystem (PCS), utility port conditioner (UPC), or power front end (PFE) due to a severe fault condition.

fault isolation matrix
A probability matrix that correlates each detected hardware error with the components and FRUs that could have caused the error. The matrix is implemented in the service processor unit (SPU) and provides component and FRU isolation for each scan latch.

FLOAT — EBox floating-point unit
EBox functional unit that performs floating-point addition, subtraction, and data format conversions.

FPL — free pointer logic
Prefetches I-stream from the MBox for the EBox. Decodes opcodes and operands, updates the PC, and provides destination address queuing.

HDSC — high density signal carrier
A multilayer substrate on which macrocell arrays (MCAs) and other integrated circuits are mounted. Contains the intercomponent connections. Also connects to the planar module.

IBEX — instruction buffer extension register
Contains 8 bytes for additional I-stream prefetching.

IBEX2 — Instruction buffer extension register 2
Another 8 bytes added on the IBEX buffer, which is added to the IBUF. This makes a total of 25 bytes of I-stream that can be prefetched from the virtual instruction cache (VIC).

IBox — Instruction decode unit
An independent functional unit that fetches and decodes instructions and their specifiers from the MBox and passes them to the EBox for execution.

IBUF — Instruction buffer
A 9-byte instruction buffer that satisfies all VAX instructions and their addressing modes.

IBUFFER — Instruction buffer
The 25-byte instruction buffer that decodes the I-stream. The IBUFFER is partitioned into the:

- 9-byte instruction buffer (IBUF)
- 8-byte extended instruction buffer (IBEX)
- 8-byte extended instruction buffer (IBEX2)

The IBUF decodes all VAX instruction and addressing modes. The remaining 16 bytes of the extended buffers contain prefetched I-stream from the virtual instruction cache (VIC).

ICU — I/O control unit
Part of the system control unit (SCU). The logical interface between the SCU and two XJAs across the JXDI. Also interfaces to the service processor unit (SPU) and implements the central system interrupt arbiter. Maximum of two ICUs per system.

INT — EBox integer unit
EBox functional unit that performs integer addition, subtraction, and logical functions.

ISSUE — Issue functional unit
Performs overall control of the EBox pipeline. Controls the issuing and completion of an instruction. Control is maintained in conjunction with the microsequencer and microcode, which direct specific execution functions.

JBox — junction box
One of three partitions of the system control unit (SCU). Provides up to four ports that interface up to four CPU subsystems. Includes address and data crossbars, and provides cache consistency logic.

JXDI — ICU-to-XJA interface
A 12-foot cable located in the system control unit (SCU) that connects the I/O control unit (ICU) to the individual XJA adapters.

KDM70 controller
An XMI controller for RA disks, TA tapes, and ESE20 disks. Provides eight ports, any two of which can be used for tape interconnect. Achieves sustained data rates of 4.0 Mbytes/s with two ESE20s and 3.4 Mbytes/s with two RA90s, at speeds of up to 700 I/O requests per second.

kernel
A configuration that includes a service processor, a power system, a populated CPU planar module (which includes the EBox, IBox, and MBox), an SCU planar module, memory array modules, and a clock system.

MAC — memory array card
Each MAC contains 32 Mbytes of on-board MOS memory (with 1 Mbit of DRAM) and two daughter array cards (DACs) that provide 16 Mbytes each, for a total of 64 Mbytes per MAC. Four MACs in a main memory unit (MMU) provide a minimum of 256 Mbytes of MOS memory.

MBox
An independent functional unit that provides the CPU interface to main memory, I/O, and other processor subsystems. It provides data cache and address translation functions, performs memory management, and fetches I-stream data on behalf of the IBox.

MCA — macrocell array
The generic term for a VLSI device consisting of an array of cells that may be configured by a hardware designer to perform a specific function.

MCA III — macrocell array III
Third generation ECL gate array with an equivalent gate count of 10,000 gates.

MCD — memory control DRAM
A macrocell array (MCA), located on the system control unit (SCU), that contains the DRAM controller and self-test controller.

MCM — master clock module
The system clock module, located in the system control unit (SCU) cabinet.

MCU — multichip unit
Contains macrocell arrays (MCAs), self-timed RAMs (STRAMs), or a combination of both and is assembled on the CPU and system control unit (SCU) planar modules.

MDP — memory data path
A macrocell array (MCA), located in the array control unit (ACU) of the system control unit (SCU), that transfers one longword between the ACU and the MMU. Also contains the ECC logic for that longword.

MMC — main memory control
A macrocell array (MCA), located in the array control unit (ACU) of the system control unit (SCU), that provides control signals for the data path, address path, and DRAMs.

MMU — main memory unit
The VAX 9000 MOS memory. Consists of four memory array cards (MACs) with each MAC carrying two daughter array cards (DACs). Each MMU (two maximum) contains 256 Mbytes of MOS memory (1 Mbit of DRAM).

MUL — EBox multiply unit
EBox functional unit that performs integer and floating-point multiplication.

naturally aligned
A data quantity whose address could be specified as an offset, from the beginning of memory, of an integral number of data elements of the same size. The lower order address bits of naturally aligned data items are characteristically 0s. All XMI reads and writes transfer a naturally aligned block of data.

node
A hardware device that connects to the XMI backplane. The largest XMI subsystem supports 14 nodes.

OCP — operator control panel
The main operator panel, it controls and/or displays the following functions: system power on/off, system restart, SPU access, and system fault codes.

PCS — power control subsystem
A subsystem that monitors, measures, and controls the state of the power subsystem. Consists of regulator intelligence cards (*see* RIC), the power and environmental monitor (*see* PEM), and power converters.

PEM — power and environmental monitor
The module that controls the regulator intelligence cards (RICs) in the power control subsystem (PCS) by communicating over the RICBUS.

PFE — H7390 power front end
Converts ac line voltage to a high-voltage dc that powers the dc/dc converters of the power system. Low-cost alternative to the utility port conditioner (UPC) and offered only in the model 210 system.

pin-fin — heat sink
Heat sink attached to multichip units (MCUs). The heat sink consists of a number of small pins that extend from the MCU and provides heat dissipation when cooling air is drawn through the pins.

PIO mode — program I/O mode
During this operating mode, the service processor unit (SPU) passes all user input to the VMS operating system. *See also* CIO mode.

PIP — power interconnect panel
Interconnect panel that receives 280 Vdc from the power front end (PFE) or utility port conditioner (UPC) and distributes it to the power system components.

planar module
A multilayer module on which multichip units (MCUs) are mounted and interconnected. The module mounts vertically in the system cabinet and can be air or water cooled.

RBDs — ROM-based diagnostics
Diagnostics resident in ROM that may be initiated on power up, or in some cases, executed manually.

receiver
An XMI term. The complement of the transmitter in a transfer. The receiver is the sink of data being moved during a transfer.

responder
An XMI term. The complement to the commander in a transaction.

RETIRE — EBox instruction retire unit
Operates with the issue functional unit to maintain execution control between instruction issue and retire. Provides two data paths to the VBox (64-bit VBox input and 32-bit VBox output).

RIC — regulator intelligence card
The module that provides the interface between the slave processor (on the RIC) and the power module it controls, which may be a power converter or utility port conditioner (UPC).

RICBUS — regulator intelligence card bus
The single-wire, multidrop, serial-communication bus that links the master processor (PEM) to the slave processors.

SBE — single-bit error
A 1-bit error detected by the ECC logic on the memory data path MCA.

SBus
The interface between the scan and clock distribution (SCD) logic, the eight CD chips on the CPU planar module, and the three CD chips on the SCU planar module.

scan — scan system
A design technique that partitions logic into small networks (rings) of combinational logic. A loading mechanism serially shifts patterns into the ring input. The ring content is serially shifted out of the ring(s) and compared to a known-good pattern related to each ring.

scan latch
Specially designed two-stage latch used to implement all the state elements in the CPU and system control unit (SCU). During normal operation, the scan latches are parallel loaded with system data and, during initialization or testing, they are serially loaded with initialization/diagnostic data from the service processor unit (SPU).

SCC — scan control chip
A CMOS gate array, located on the scan control module (SCM), that controls all scan operations. Responsible for shifting the scan rings, DMA access to local memory, pattern comparison, and control of the master clock module (MCM).

SCD — scan and clock distribution
A part of the RLOG and DSCT MCA logic that distributes scan signals throughout the planar module.

SCI — scan interconnect
Provides the scan interface to the CPU(s), system control unit (SCU), and multichip unit (MCU).

SCM — scan control module
A module in the VAXBI backplane of the service processor unit (SPU) that contains the scan logic.

SCU — system control unit
Interconnects the CPU, I/O subsystem, main memory arrays, and SPU. Partitioned into three functional units:

- JBox, CPU interface
- I/O control unit (ICU), I/O subsystem interface
- Array control unit (ACU), main memory interface

SDC — scan distribution chip
A custom gate array on the scan control module (SCM) that distributes and receives scan control and data lines. The scan interconnect (SCI) starts at the SDC.

SDD — symptom-directed diagnosis
A diagnostic technique that isolates intermittent and hard failures to a failing device by analyzing machine state information retrieved at the time the error occurred. Supported with hardware error detection circuits, scan latches, and operation history buffers.

SIP — signal interconnect panel
Provides signal interface between power control components including the power and environmental monitor (PEM), regulator intelligence cards (RICs), operator control panel (OCP), and power converters. Contains logic associated with battery backup units and total off circuitry.

SJA — SPU-to-JBox adapter
Interfaces the service processor unit (SPU) to the JBox through the RXCS, RXDB, TXCS, and TXDB registers.

SL, SLU — short literal unit
An IBox functional unit dedicated to short literal operand expansion.

SPM — service processor module
The host processor of the service processor unit (SPU). Contains the SJA, terminal ports, and processor.

SPU — service processor unit
A VAXBI-based MicroVAX subsystem that provides the traditional console processor functions, including system initialization. Also acts as the diagnostic processor, controls the scan system rings, and retrieves symptom data used by the SDD analysis tools.

SSTs — startup self-tests
Tests that verify the integrity of all components in the power control subsystem (PCS).

STECL — series-terminated ECL
ECL logic that is terminated in a manner that reduces overshoot and ringing on transmission lines. Series termination uses a resistor on the output gate of a circuit, where the resistor plus the circuit output impedance is equal to the impedance of the transmission line.

STRAM — self-timed RAM
Dynamic RAM array with internal latches that allow synchronous operation. Used in place of standard RAMs.

STREG — self-timed register
A 64 $\times$ 18-bit register file containing three write ports and two read ports. Its 64 locations are separated into four 16-location storage array banks. The read and write clocks control address, write enable, byte select, and data input latching. However, array timing is internally generated and controlled.

TDD — test-directed diagnosis
A diagnostic technique that attempts to detect fault conditions by applying a controlled stimulus to a circuit and then observing the output to verify that the circuit operates as expected.

transaction
An XMI term. Composed of one or more transfer. *Transaction* is the name given to the logical task being performed (for example, read, write). In the case of a read operation, the transaction consists of a command transfer followed some time later by a return data transfer.

transfer
The smallest quantum of work that occurs on the XMI bus. Typical examples are the command cycle of a read operation and the command and following data cycles of a write operation.

transmitter
An XMI term. The node that is sourcing the information on the bus. For example, during a read transaction, the commander is the transmitter during the command transfer, and the receiver during the return data transfer.

UPC — utility port conditioner
A system that converts the ac line voltage to a regulated high-voltage dc. Reduces harmonic line distortion and causes the power system to appear as a unity power-factor load. The high-voltage dc powers the dc/dc converters.

VBox — vector accelerator
An optional accelerator that operates with a scalar processor to provide additional performance for vector operations.

VIC — virtual instruction cache
An 8-Kbyte virtually addressed, direct-mapped, one-way associative cache that reduces the number of I-stream requests issued to the MBox. The VIC is refilled from the MBox data cache.

VREG — vector register
Bit-sliced custom VLSI storage device used to implement the vector register file of the VBox. The vector register file consists of sixteen 64 × 9-bit VREGs and provides 1024 vector storage locations.

wraparound read
An octaword or hexword read operation where read data is returned in a specific pattern in which the specifically addressed quadword is returned first independent of alignment. The remaining data in the naturally aligned data block containing the addressed quadword is returned in subsequent transfers in descending quadword address order.

XBAR — crossbar
Multiple specifier decode unit. Can simultaneously decode up to three operand specifiers. Consumes the I-stream data from the IBUF. The I-stream is presented to the XBAR 9 bytes at a time.

XBI — XMI-to-BI adapter
Provides the control and data interface between the XMI bus and the VAXBI bus.

XCA — XMI-to-CI adapter
Provides the control and data interface between the XMI bus and the CI bus.

XJA — XMI-to-SCU adapter
A module that resides in the XMI backplane and interfaces the XMI I/O bus to the system control unit (SCU).

XMI bus
VAX 9000 I/O bus. The bus is a limited length, pended, synchronous bus with centralized arbitration and a 64-ns bus cycle.

XMI card cage
An enclosure containing a backplane that holds devices interconnected with an XMI bus.

XXNET
The CSMA/CD (carrier sense multiple access with collision detection) protocol that is used by the power and environmental monitor (PEM) and the power control subsystem (PCS).

Index