iAPX 432 INTERCONNECT ARCHITECTURE REFERENCE MANUAL

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Order Number: 172487-001

iAPX 432
INTERCONNECT ARCHITECTURE
REFERENCE MANUAL

Order Number: 172487-001

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

dml/1282

REV.	REVISION HISTORY	DATE
-001	Original Issue	12/82

This manual describes the iAPX 432 interconnect architecture. It is assumed that the reader is already familiar with other Intel literature related to the iAPX 432 system. A list of references is provided below.

Related Publications

For general information about the iAPX 432 system:

iAPX 432 General Data Processor Architecture Reference Manual, Order Number 171860.

iAPX 432 Interface Processor Architecture Reference Manual, Order Nunber 171863.

For detailed information about iAPX 432 processor components that may be used with the interconnect architecture, refer to:

iAPX 43201/43202 General Data Processor Data Sheet, Order Number 171873.

iAPX 43203 Interface Processor Data Sheet, Order Number 171874.

For more information about iAPX 432 interconnect components, refer to:

iAPX 43204/43205 Data Sheet, Order Number 172866, which describes the Bus Interface Unit (BIU) and Memory Control Unit (MCU) interconnect components.

For more information on the general topic of fault tolerance, refer to:

The Theory and Practice of Reliable System Design, Sieworik and Swartz, 1982, Digital Press. This book contains a variety of background information. Chapter 18 is devoted to the iAPX 432: "A Design Method for High-Reliability Systems: The Intel 432", Daniel P. Sieworik and David P. Johnson.

CHAPTER 1 PAGE
KEY CONCEPTS

CHAPTER 2
FUNDAMENTAL FAULT HANDLING CONCEPTS

CHAPTER 3
FUNDAMENTAL SYSTEM STRUCTURES

CHAPTER 4
OVERVIEW OF THE FAULT HANDLING MECHANISMS

CHAPTER 4 - Continued PAGE
OVERVIEW OF THE FAULT HANDLING MECHANISMS

CHAPTER 5
CONFINEMENT AREAS/DETECTION MECHANISMS

CHAPTER 6
REDUNDANCY

CHAPTER 6 - Continued PAGE
REDUNDANCY

CHAPTER 7
ERROR REPORTING

CHAPTER 8
RECOVERY

CHAPTER 9 PAGE
SPECIAL CASES OF INTEREST

CHAPTER 10
SOFTWARE INTERFACE SUMMARY

TABLES

INTRODUCTION

The first phase of the iAPX 432 program introduced two processor types: the General Data Processor (GDP) and the Interface Processor (IP). The GDP was implemented with two VLSI components: iAPX 43201 and iAPX 43202. The IP was implemented as a single VLSI component: the iAPX 43203. These three VLSI components implement the processor architecture for the iAPX 432. System builders have constructed multiple processor systems by surrounding the VLSI processors with discrete logic, which provided the interface to shared memory and the interprocessor communication paths. The method for interconnecting iAPX 432 processors and memories was unique for each for each system, since no standard had been defined.

This manual describes a unifying interconnect architecture for building iAPX 432 systems. The interconnect architecture has been implemented in a pair of VLSI components: the iAPX 43204 Bus Interface Unit (BIU) and the iAPX 43205 Memory Control Unit (MCU). Together, these components form the basis for constructing multiple-processor iAPX 432 systems. The BIU performs as an intelligent switch. The MCU acts as an intelligent memory controller.

The iAPX 432 interconnect architecture provides:

- Integrated fault tolerance. The VLSI interconnect components (BIU/MCU) integrate all the detection and recovery logic required to build a system that can tolerate any single component failure.

- Software-transparent fault tolerance. Hardware performs all fault detection and recovery functions transparent to application software. The architecture never dies.

- Configurability. The BIU and MCU support a range of fault tolerance and performance options to meet a diverse set of cost, performance, and reliability needs.

- Standard VLSI solution. Very little external logic is required.

- Reliable software. The iAPX 432 system's "need to know" (capability) addressing confines errors, protecting the system from errant software.

The object-based architecture of the iAPX 432 provides a robust and flexible environment for cooperating, concurrent software systems. The iAPX 432 processors use a cooperating, self-dispatching mechanism to automatically share the workload between the available processors. The number of processors available in the system is transparent to software.

The BIU and the MCU extend the logical robustness and flexibility of the iAPX 432 processors into the physical implementation of iAPX 432 systems. The BIU and MCU allow the iAPX 432 hardware to modularly and transparently extend the processing power (from 1 to 63 modules of processors or memories), bus bandwidth (1 to 8 backplane buses), and fault-tolerant capabilities of the system.

As Figure 1-1 shows, an iAPX 432 system based on the interconnect architecture may be expanded gracefully. A system with one processor and one memory may be built with a single memory bus. Transparent multiprocessing may be achieved by simply adding processor modules. When additional memory is required, memory modules may be added onto the single memory bus. When more memory bandwidth is required, additional memory (bus or buses) can be added. None of these alternative systems require any change to application software.

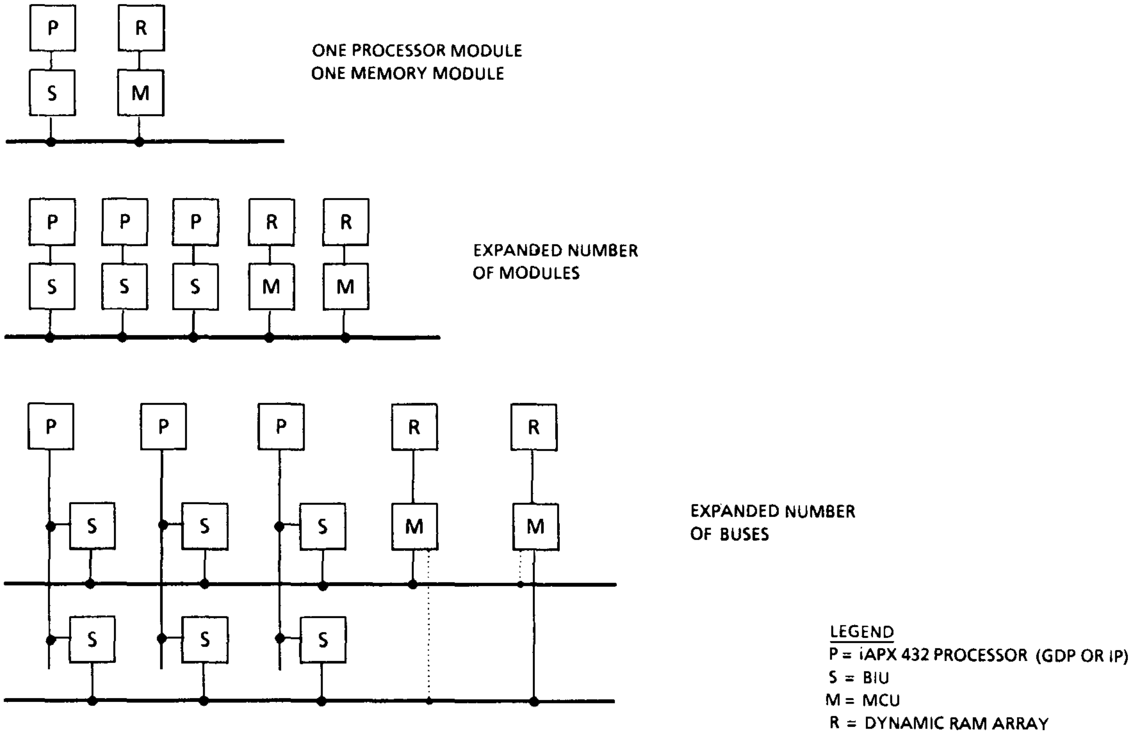


Figure 1-1. Modular Expansion F-0396

In an iAPX 432 system, each processor is unaware of the manner in which the memory address space is actually implemented. Hardware located in the BIUs determines how processor addresses are mapped to buses and memory systems.

Figure 1-2 illustrates how iAPX 432 processors, BIUs, and MCUs are organized into a multiprocessor interconnect system. Each processor and its associated BIUs form a module. Each MCU and its associated storage array form a module. These modules form the basis for detection and confinement of errors. Every BIU and MCU constitutes a node in the interconnect system. A unique BIU or MCU may thus be named in a consistent way by referring to its node address, which consists of the number of the bus to which it is attached and the number of the module in which it is contained. The benefit of the unified, modular design of the iAPX 432 interconnect system is that all modules are alike -- no special backplane slots are required to change the number or mix of processors (GDPs and IPs) and memories in an iAPX 432 system.

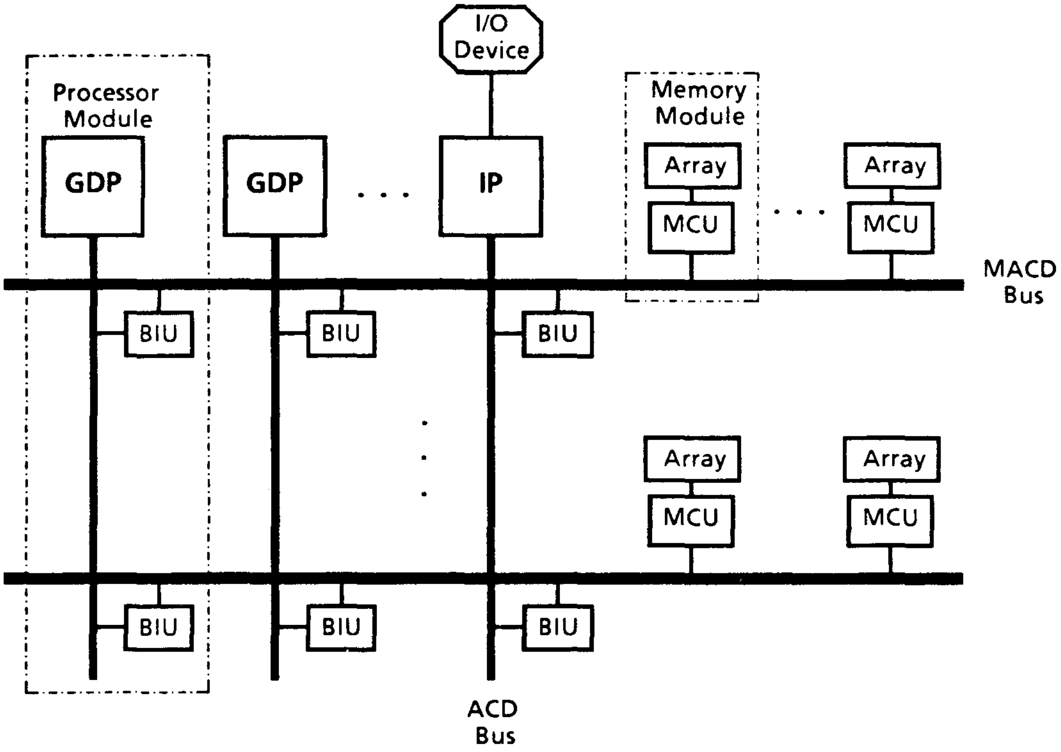


Figure 1-2. iAPX 432 Interconnect Topology F-0398

BUS INTERFACE UNIT

The Bus Interface Unit or BIU provides the switching function of the iAPX 432 interconnect system. That is, it accepts the access requests from an iAPX 432 processor and, based on the physical address, it decides which memory bus(es) will be used to perform the access. The BIU is also responsible for arbitrating the usage of the memory bus.

MEMORY CONTROL UNIT

The Memory Control Unit, or MCU, interfaces memory storage arrays to the memory bus. The storage arrays will typically be constructed with high-density dynamic RAM (DRAM) components. All types of DRAMs are supported: 16K, 64K, 256K, even partially good components. The MCU manages the storage array as a logical collection of 32 data bits, 7 bits of error correcting code (ECC), and an optional spare bit. The MCU can automatically refresh the dynamic storage array. In addition, the MCU can scrub single-bit errors from the storage array as a background task. Scrubbing is accomplished by periodically reading the storage array, correcting all single-bit errors, and detecting and reporting all double-bit errors. The MCU accepts variable length data requests from the memory bus and performs the necessary access sequencing to read or write the data into the storage array. A modest amount of external logic is required to interface the MCU to the storage array RAMs -- for simple configurations, as few as 12 external TTL packages are required.

MEMORY BUS

The memory bus (sometimes referred to as the MACD bus) provides the principal communication path, carrying all memory access requests and interprocessor communication. The memory bus connects BIUs to MCUs. Each node in the interconnect system tracks each operation on the memory bus to which it is attached. Thus, unlike most bus protocols, each BIU and MCU keeps track all of outstanding requests on the bus -- not just the ones it made. Control for the bus is fully distributed; there is no centralized bus controller.

INTERLEAVING

When several processors in a shared memory multiprocessor system demand information from a common area of memory (e.g., a shared instruction or data segment), they tend to impede the efforts of one another. All the processors are competing for the same memory bus. The iAPX 432 interconnect system answers this problem by interleaving memory buses. Interleaving spreads the accesses to a common segment of memory across several memory buses. Different memory buses support the alternation of consecutive regions of a processor's address space. On the basis of the mid-order address bits from the processor's physical address, the BIUs control interleaving. A system designer may opt for no interleaving, two-way interleaving, or four-way interleaving under software control. Address bits 6, 7, and 8 may be used to select interleaving on 64-, 128-, or 256-byte boundaries. For example, when two-way interleaving is performed at 64-byte boundaries, a processor's requests for data between addresses 0 and 63 are served by one memory bus and those for data between addresses 64 and 127 are served by a different memory bus.

REQUEST/REPLY PAIRS AND PIPELINING

The interconnect system also supports pipelining on each of its memory buses. This allows several processors to issue memory requests before a memory module has performed all its previous requests. Each memory bus in an iAPX 432 interconnect system can queue a maximum of three memory requests. (That is, the maximum memory bus pipeline depth is three.) Pipelining can occur because each request from a processor is separated in time from the corresponding reply from the memory module that supports the request. Each memory access consists of a request/reply pair. Every request deepens the memory bus pipeline by one entry. Each reply removes one entry from the pipeline. All BIUs and MCUs track the state of the memory bus pipeline to which they are attached in order to determine when it is permissible to issue new requests or to return awaited replies. The memory requests are issued to a memory bus as packets of information. Each request also has a data length field. On the memory bus, data requests may be variable length. Access lengths of from 1 to 16 bytes are supported.

INTERPROCESSOR COMMUNICATION

In contrast to standard systems, the iAPX 432 interconnect system also provides a facility for rapid interprocessor communication (IPC). In the iAPX 432, a processor may signal another processor or all other processors of a condition that requires attention (e.g., "stop execution", "flush caches", etc.). The IPC facility is supported in the interconnect system by a special form of memory bus message called a bus notification. Unlike standard memory request/reply pairs, the bus notification is a stand-alone memory bus packet that notifies the processor(s) of an IPC. Each BIU decides if its associated processor should act on the IPC on the basis of whether the processor ID contained in the bus notification matches the processor's unique ID or the global ID. Each processor in the system has a unique processor ID, held in its associated BIU(s), which is established during system initialization.

INTERCONNECT REGISTERS

The programmable attributes of BIUs and MCUs in an iAPX 432 interconnect system are contained in registers of the interconnect address space, which is separate from the memory address space. These interconnect registers are accessed with the Move To Interconnect and Move From Interconnect instructions of iAPX 432 processors. Interconnect registers are logically organized by interconnect objects in the iAPX 432 software system. If a processor is given permission to access an interconnect object, it may change the attributes of the interconnect system.

Refinements of interconnect objects may be used to allow a processor a limited view of the total set of interconnect registers, restricted to those the processor has a "need to know": for example, to obtain its processor ID. A programmer can access the interconnect registers of any of the nodes in the interconnect system by node-local addressing (N-local). A node's address is composed of the unique module number and unique bus number that locate the node in the interconnect system. Each BIU and MCU in an interconnect system contains 32 unique interconnect registers.

FAULT-TOLERANT SYSTEMS

Three basic principles form the basis for implementation of the iAPX 432 fault handling mechanisms. First, the fault-tolerant functionality is achieved by replication of VLSI components. Second, the machine is partitioned into a set of confinement areas. These areas form the basis for error detection and recovery. Third, only bus-oriented communication paths are used to provide system communication.

VLSI replication is fundamental to achieve effective use of VLSI technology. The iAPX 432 allows the system designer to build a wide range of systems from a small set of VLSI components. These same components provide modular expansion of performance, memory storage, detection, and recovery capabilities. There are no special purpose components aimed solely at fault-tolerant applications.

The purpose of a confinement area is to limit damage from error propagation and to localize the faulty area for recovery and repair. A confinement area is defined as a unit (module or memory bus) of the system which has a limited number of tightly controlled interfaces. Detection mechanisms are placed at every interface to ensure that no inconsistent data can leave the area and corrupt other confinements areas. When an error occurs in the system, it is immediately isolated to a confinement area. The error is known to be in that confinement area, and all other confinement areas are known to be error-free.

By defining confinement areas, we provide a conceptual framework for the systematic and coherent placement and definition of the detection mechanisms. The confinement areas also provide a conceptual view of the system under fault conditions. This clarifies the external (software) view of the hardware, and eliminates the need for diagnostic probing as a method of fault isolation.

All communication in the iAPX 432 system is done over buses. There are no point-to-point signals or daisy-chained signals. This makes modular growth possible since no signal definition is dependent on the number of resources in the system. This approach also makes on-line repair possible. The presence or absence of any module cannot prevent communication between any other modules. The memory bus defined by the BIU and MCU provides a uniform and regularly structured communications path that supports the modular expansion of both fault-tolerant and standard system capabilities.

In the iAPX 432 there are three distinct steps in responding to an error. First, the error is detected and localized to a confinement area. Next, the error is reported to all of the modules in the system. This prevents the incorrect data from propagating into another confinement area and provides all the modules with the information required to perform recovery. Finally, the faulty confinement area is isolated from the system. Recovery occurs through the application of the redundant resources available in the system.

ERROR CONFINEMENT

Figure 1-3 shows the four types of confinement areas in a iAPX 432 system. There is a confinement area for each module and each memory bus in a system. These confinement areas were chosen because modules and memory buses are the natural building blocks for iAPX 432 systems. Thus, when an error is detected, it is confined to one of the system building blocks. This allows the recovery and repair strategies to be built around the replacement of system building blocks. When a module or bus has its confinement mechanisms activated, it can be viewed as a self-checking unit. The operation of a self-checking unit is designed so that no inconsistent data will be allowed to leave the unit and corrupt another confinement area. Detection mechanisms reside at every interface, and all data is checked as it flows across the interface between confinement areas.

The GDP confinement area includes the GDP and its associated BIUs plus the processor bus and support logic in the module. The only interfaces to a GDP confinement area are the memory buses. The BIUs are responsible for checking all of the information that leaves the GDP module. No information (control, address, or data) can leave a GDP confinement area without first being checked for correctness by one of the BIUs in the module. Error detection is performed by duplicating the GDP module. The duplicate module is built from identical components that are in checker mode. Any disagreement between a master and its checker generates an error signal. A duplicated module thus forms a self-checking module.

The IP confinement area includes the IP and its associated BIUs plus the processor bus and support logic in the module. An IP module has interfaces to the memory buses in the system, plus an interface to an external I/O subsystem. The interfaces to the memory buses are checked by the BIUs in the same manner that was described for the GDP confinement area. The IP component is responsible for checking any data that leaves the confinement area via the peripheral subsystem (PS) bus. No information can leave an IP confinement area without first being checked for correctness by one of the BIUs or the IP. (The peripheral subsystem is not a confinement area.)

At this time the application hardware or software must apply its own detection mechanisms to this subsystem. The PS bus represents a firewall between the central system and the I/O subsystem. The IP confinement area checks data as it leaves the IP: the application hardware and software must check data that leaves the I/O subsystem and enters the IP module. Error detection is performed by a duplicate checker module, as it was with the GDP module.

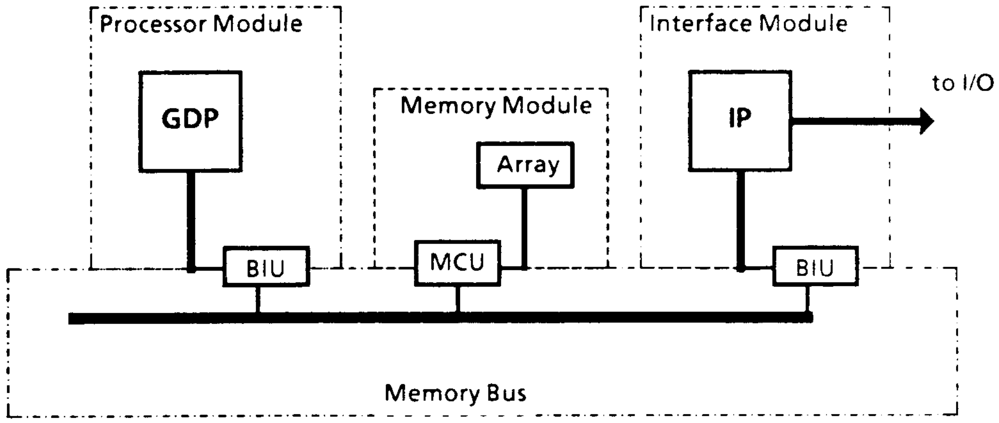


Figure 1-3. iAPX 432 Confinement Areas F-0400

The memory confinement area includes the MCU, the RAM array, and the buses and support logic inside the module. A memory module has interfaces to two of the memory buses in the system. The MCU is responsible for checking all information that leaves the memory confinement area. No information can leave the confinement area without first being checked for correctness by the MCU. Error detection is performed by duplicating the MCU and applying an ECC code to the memory array. Thus, a self-checking memory module has two MCUs but only one memory array.

Each memory bus confinement area includes a memory bus and the interface logic residing in the BIUs and MCUs attached to the memory bus. Each memory bus has interfaces to all of the GDP and IP modules plus some of the memory modules. Every node (BIU or MCU) attached to this bus is responsible for checking all of the information that flows off the memory bus and into its module. No information can leave the memory bus and enter a module without first being checked for correctness by either a BIU or an MCU. Error detection is performed by two interlaced parity bits, which cover the control and address/data lines. Error detection for the arbitration lines is achieved by duplication.

Figure 1-4 illustrates how interconnect components are logically paired to form self-checking units. Functional Redundancy Checking (FRC) provides the logical error detection mechanism. The master processor and its associated BIUs generate memory bus requests. The checker processor and its associated BIUs are functionally redundant elements used for error detection. Logic in the master and checker BIUs compares all information sent to a memory bus by the master. If any disagreement is detected, an error report is generated.

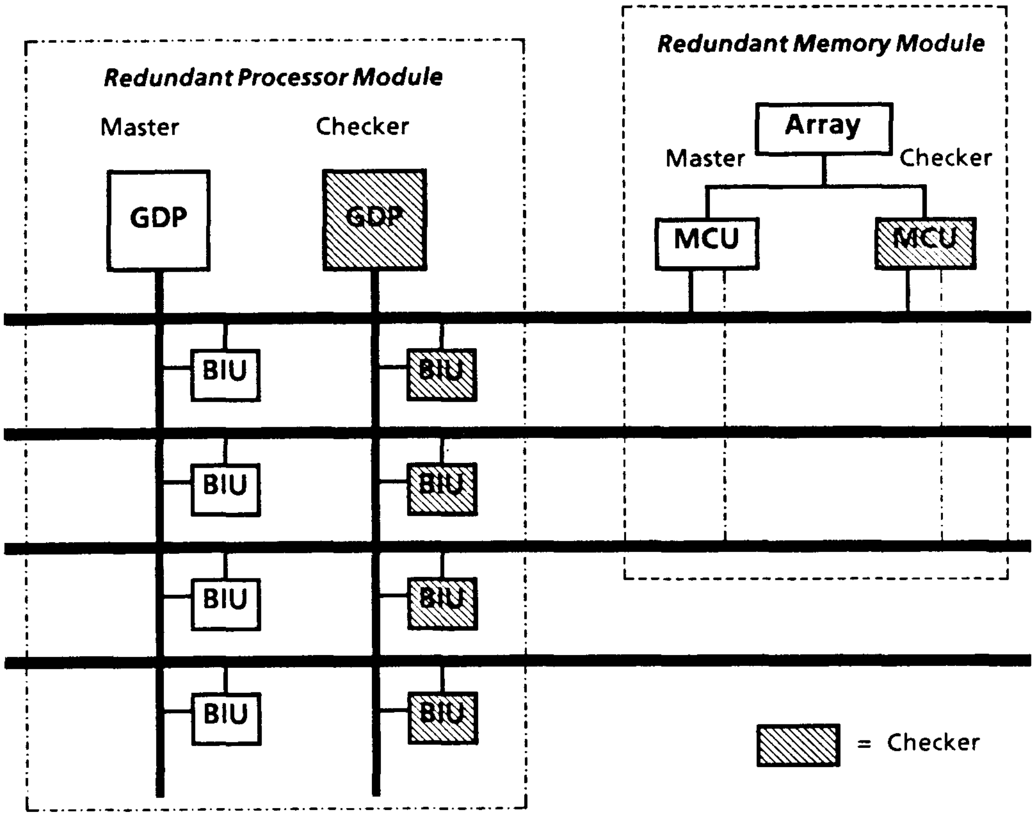


Figure 1-4. FRC Configuration Pairing F-0399

An example of processor-memory operation may help to clarify the operation of the confinement areas. This example is shown graphically in Figure 1-5. Assume that a GDP makes a read request to a memory location. That request will be mapped to a BIU on the addressed memory bus. As the information flows onto the memory bus it will be checked by the BIU. If any failure has occurred in the GDP confinement area (GDP, processor bus, BIUs, etc.) it is detected at this time. The information flows across the memory bus and into the addressed memory module. Before the information is accepted by the module, the MCU checks it for correctness. If a failure is detected, it is confined to the memory bus because the information was valid when it left the GDP confinement area. The MCU performs the memory operation and returns data onto the memory bus. As data flows onto the bus it is checked for correctness by the MCU. As the data flows into the GDP module from the memory bus it is checked for correctness by the BIU before being used by the GDP module.

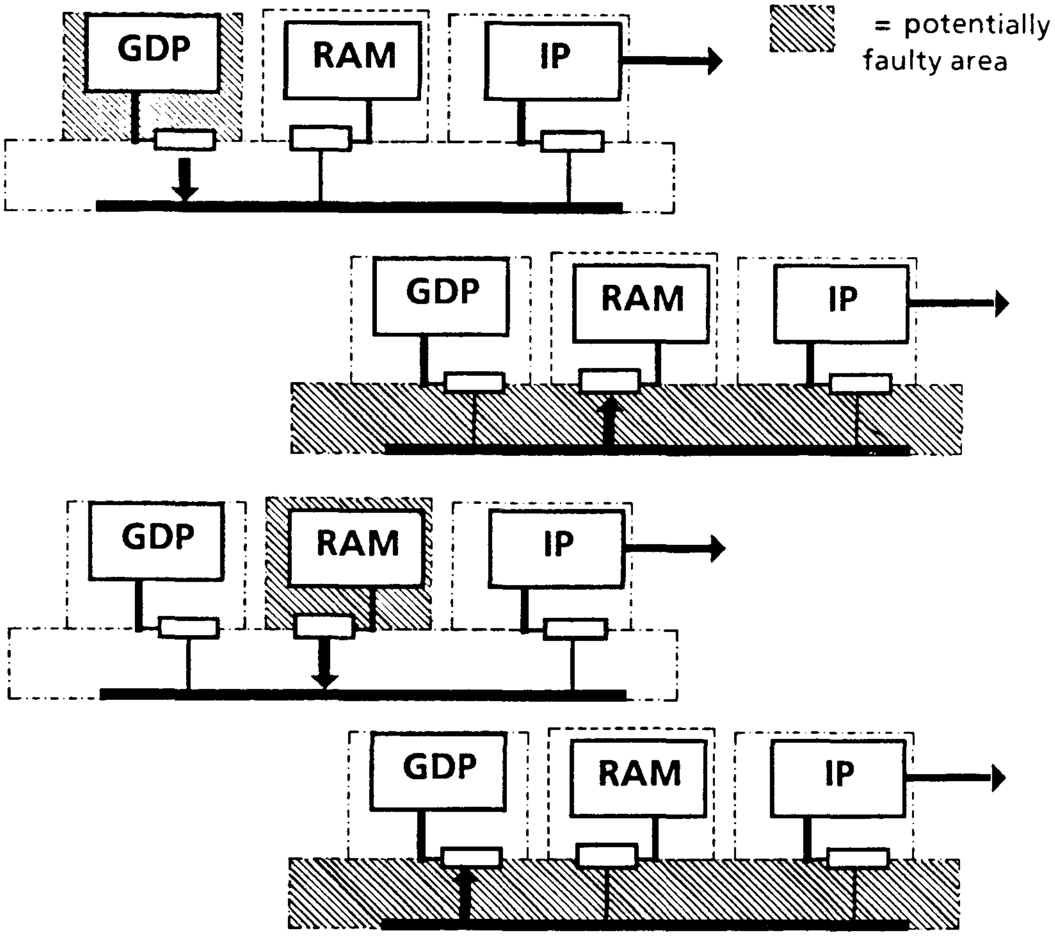


Figure 1-5. Confinement Area Operation F-0401

The confinement area interfaces provide very tight error control and isolate the failure to one of the building blocks present in the system. The only remaining question concerns checking the detection mechanisms. Most of the detection mechanisms are self-checking in that the detection circuits are checked as part of normal operation. Those circuits that are not self-checking can be exercised during normal system operation to flush out any latent faults.

Four hardware error detection mechanisms are employed in the interconnect architecture: parity, Functional Redundancy Checking (FRC), error correcting codes (ECC), and buffer checking. All data, address, and control information that is moved across a memory bus is checked by a pair of parity bits that are interlaced among the signals that they check. FRC is a mechanism that allows identical VLSI components to be connected in parallel so that they may check each other's operation in real time. FRC error detection is an optional mechanism. MCUs perform ECC checks across all the address and data bits that are written to or read from the storage arrays they control. To cover the miscellaneous logic external to BIUs and MCUs, the VLSI components support a method of buffer checking that performs a sanity check to ensure that buffers are operating.

REPORTING

Immediately upon detecting an error, a message is broadcast to all the nodes in the system. This error report message identifies the fault confinement area, the type of error that occurred, and whether the error is permanant or transient. There are two reasons for sending this error report. First, it informs the rest of the system that an error has occurred. This prevents other confinement areas from using the inconsistent data. Second, it provides the necessary information for system recovery. After recovery, the error message is recorded in a log register in every node in the system. This log is available to software and is useful in monitoring the health of the system.

The error messages are broadcast over a set of serial buses, which are totally independent from the buses used during normal operation. However, this network of serial buses follows exactly the same topology as the buses used for normal operation. A failure on one of these buses is limited to one of the confinement areas discussed earlier. The error reporting circuitry may be tested during normal operation to uncover any latent faults.

Figure 1-6 illustrates the three-phase flow of error information in the interconnect system. In phase 1, an error is detected at a node in the interconnect system. The example illustrates an an error detected at BIU(2,1); i.e., the BIU on memory bus 2 in processor module 1. The detecting component reports the error to all components attached to the same bus. (A bold line indicates an active error reporting path.) At this point, if all error reporting nodes are intact, all nodes have received the error message. In phase 2, all components that received the phase 1 error message rebroadcast the message along their module paths. Finally, in phase 3, each component that has received an error message rebroadcasts the message along its bus path. This second rebroadcast ensures that all nodes receive the error message even if one module or one bus error report line has failed. At the end of phase 3, all interconnect components in the system have been informed of the error.

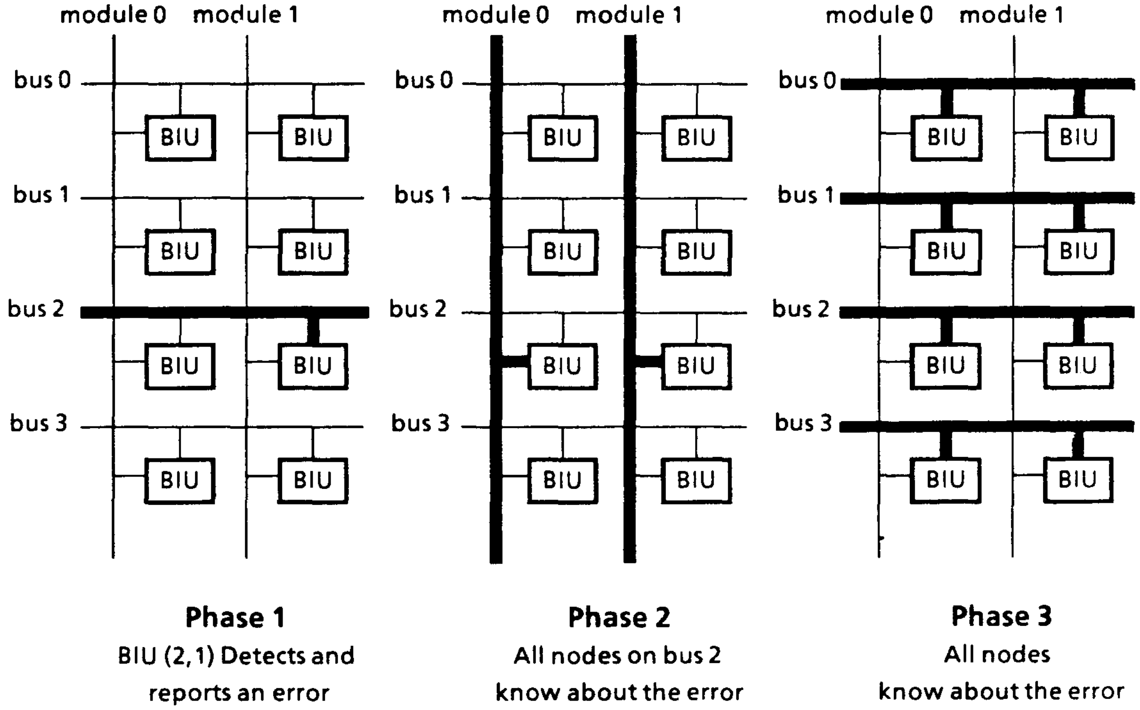

module 0
module 1
bus 0
bus 1
bus 2
bus 3
BIU
Phase 1
BIU (2,1) Detects and reports an error
Phase 2
All nodes on bus 2 know about the error
Phase 3
All nodes know about the error

Figure 1-6. Three-Phase Error Reporting Mechanism F-0402

The actual error reporting paths are separate from, but run parallel to, the MACD and ACD (Address, Control, and Data) buses so that error reports may propagate even if a bus is inoperative. In addition, the reporting paths may be duplicated to remove any single-point dependency in delivering an error report.

RECOVERY

The recovery process begins after an error report message has been broadcast around the system. Recovery is a distributed operation on the iAPX 432. Each node in ths system reads the error report message and decides what recovery needs to be taken.

For recovery to be successful, there must be redundant resources available in the system. There are five redundancy mechanisms in the iAPX 432: retry buffers, ECC, module shadowing, bus switching, and the spare memory bit. The first two of these mechanisms provide information useful in recovering from transient errors, while the other mechanisms allow recovery from permanent errors in the system. These redundant resources cover the entire system and allow recovery from any detected error. The presence of redundant resources has no impact on system performance.

For transient errors, each BIU maintains an internal buffer that allows outstanding processor requests to be retried if a transient error occurs. A single-bit correcting ECC code is applied to each word in the memory arrays. Although this provides redundancy for both permanent and transient errors, its primary purpose is to correct soft errors, which occur in DRAMs.

For permanent errors, every module in the system may be paired with another self-checking module of the same type. This pair of self-checking modules operates in lock step and provides a complete and current backup for all state information in the module. This mechanism is known as module shadowing because if one module (the primary) fails, a duplicate module (the shadow) is ready to fill in. A fault-tolerant module is also called a QMR (Quad Modular Redundant) module because most components are replicated four times. There are two self-checking modules and each of these has a master and a checker. See Figure 1-7 for an example of the pairing of QMR modules.

Each memory bus in the system may be paired with another memory bus. During normal operation the buses run independently. Both contribute to the total memory bandwith available in the system. However, if one bus fails, the other bus is capable of handling the bus requests (switching buses and rerouting accesses) that normally would have been handled by the failed bus.

Inside of each memory module, a spare bit may be added to each word in the memory array. If one bit in the array fails, the spare bit can be switched in to replace the failed bit.

For transient errors, all of the outstanding access will be retried, and the MCUs will return corrected data if there are any single-bit errors in the memory arrays.

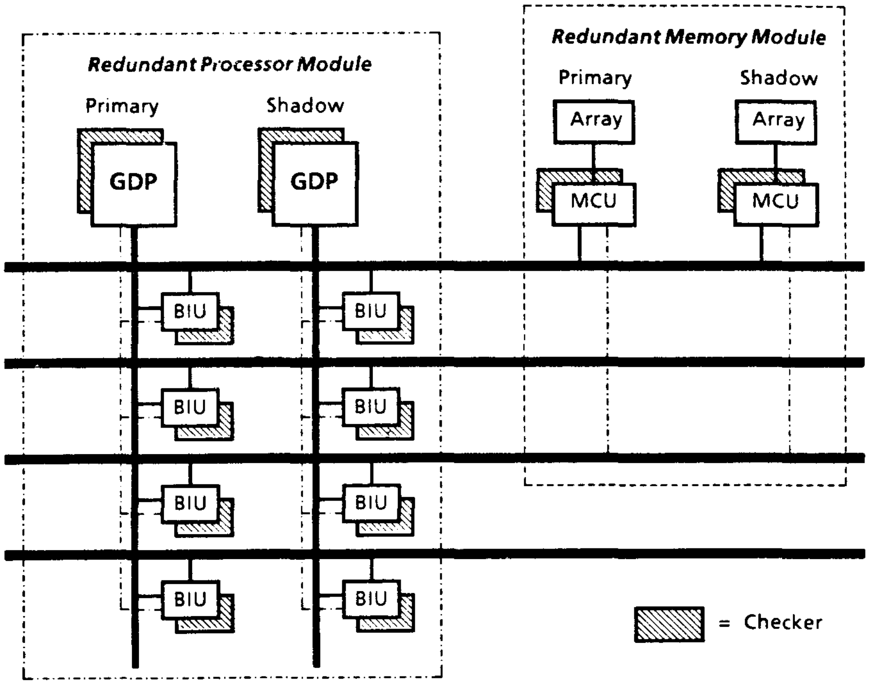


Figure 1-7. QMR Configuration Pairing F-0403

For permanent errors, the redundant resource is switched in to replace the failed unit. This switch is done on a node-by-node basis; there is no centralized element that controls the switch. Each node knows which module or memory bus it is backing up (shadowing). If the error report identifies its partner as the faulty unit, it then becomes active and takes over operation for the faulty unit. After the resource switch is complete, all of the outstanding accesses are retried. This allows operation to resume at a point before the failure corrupted data.

These reconfiguration and recovery actions are performed by the hardware without any software intervention. After recovery is complete, the hardware informs the system software of the error and subsequent recovery actions. System software now makes policy decisions regarding optimum system configuration, given the resources remaining in the system. Software can elect to continue running with same number of modules but with reduced fault tolerance (because of the failure), or it can remove the module from service, thus reducing performance but retaining the same level of fault tolerance. The "marrying" of primary and shadow modules is under software control, giving the system a great deal of configuration flexibility. These policy decisions are carried out while normal system operation continues.

Figures 1-8 and 1-9 show two examples of recovery operation. Table 1-1 lists the recovery mechanisms provided by the interconnect system and the areas of the system which each covers.

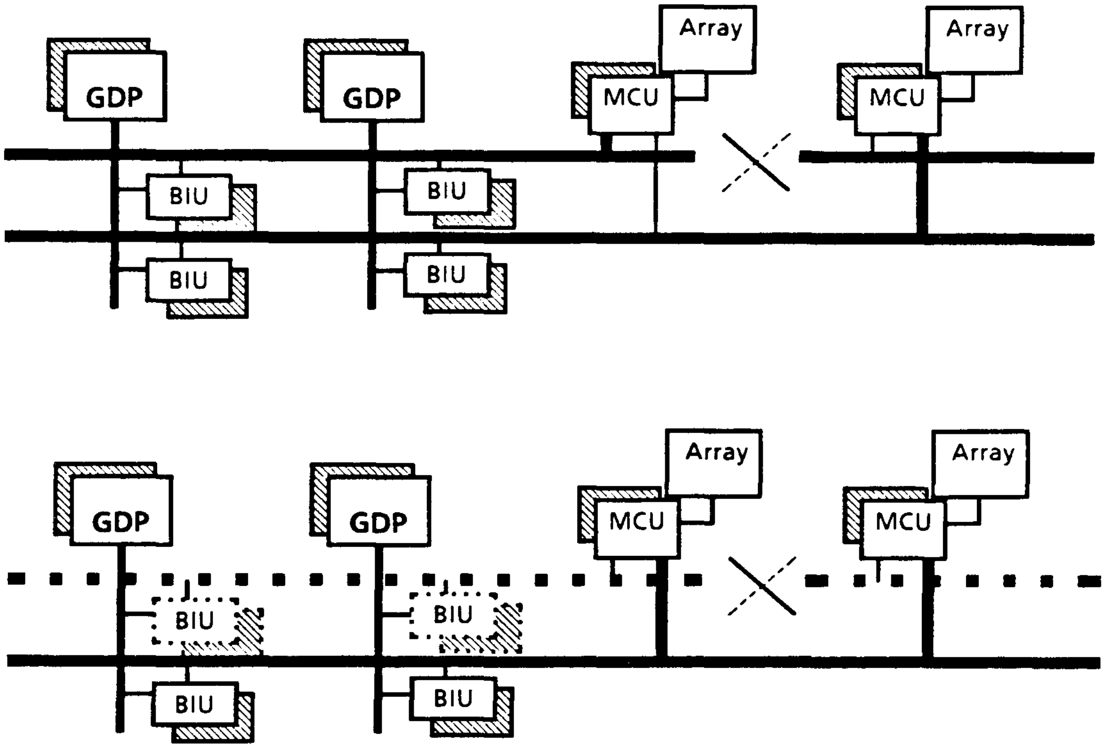


Figure 1-8. Bus Reconfiguration

F-0404

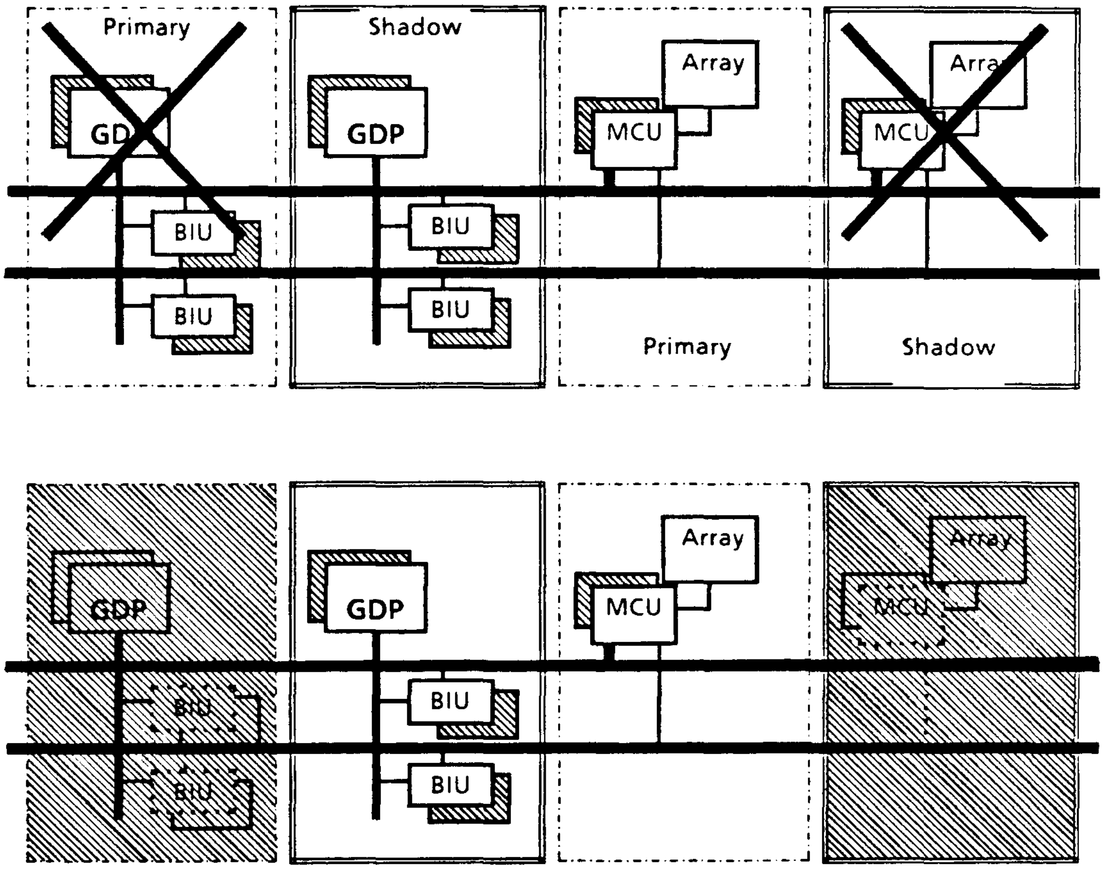


Figure 1-9. Module Reconfiguration F-0405

Table 1-1. Recovery Mechanisms and Coverage

MECHANISM	COVERAGE
Retry	Transient Errors
ECC	Storage Array Address and Data
Spare Bit	DRAM Replacement
Memory Bus Pairs	Memory Bus Failure
Module Shadowing	Module Failure, GDP, IP, or Memory

Figure 1-10 illustrates the range of alternatives available to system designers when they build iAPX 432 systems. The most fault-tolerant systems are built from a QMR configuration of processors that can tolerate any single component failure without crashing the system.

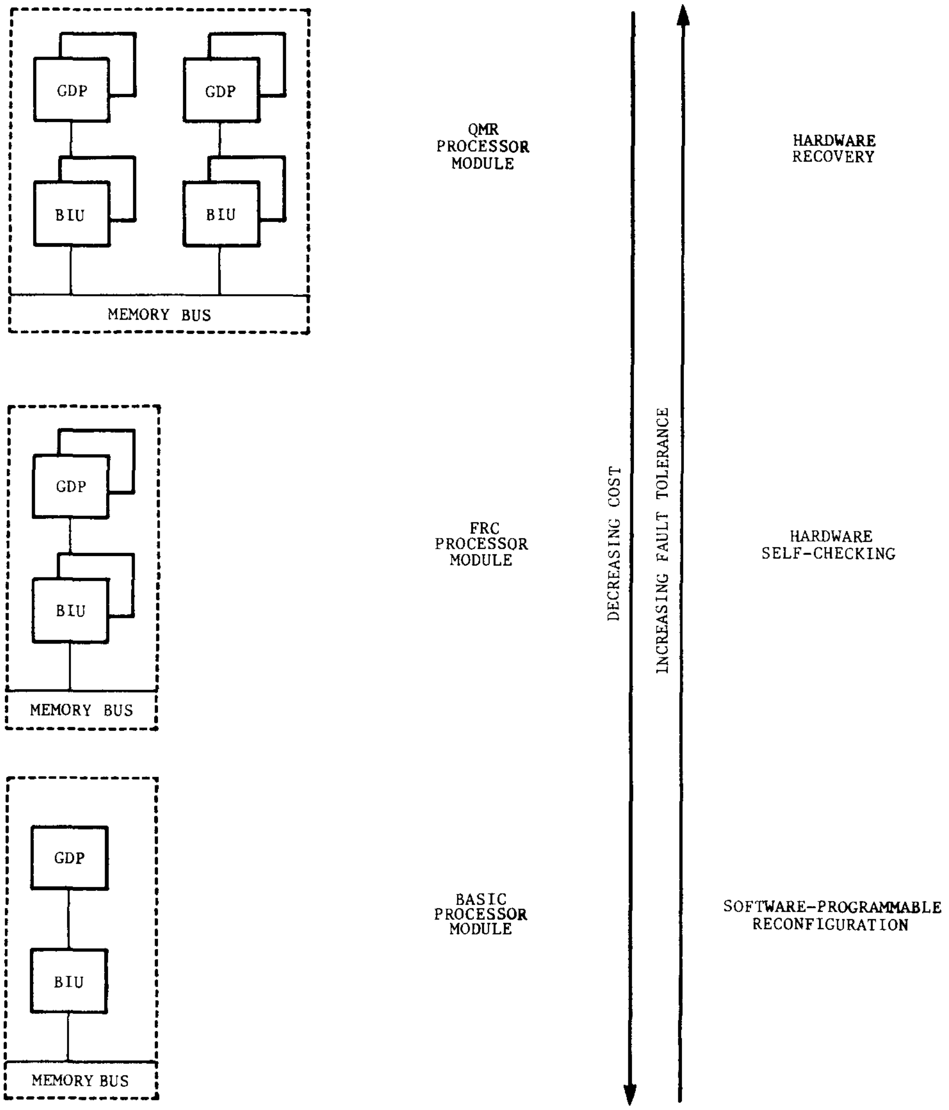


Figure 1-10. Fault-Tolerant Alternatives F-0397

The lowest cost configurations can be built using basic processor modules without FRC or QMR. This type of configuration will crash if a component fails, but can be made "self-healing" by adding intelligent software to the I/O subsystem. Unlike QMR, self-healing does not protect against the system crashing, but it does allow the system to recover from a failure in a short period of time. The "healing" takes place in three steps. First, a watchdog timer on an I/O subsystem alerts I/O subsystem software that the central system has failed. Second, the I/O subsystem checks BIU/MCU error logging registers and runs diagnostics to identify which resource (e.g., processor, bus, or memory) has failed. Third, the I/O subsystem reinitializes the system using the configuration control within the BIU and MCU to configure out the failed resource. The system is up and running without human intervention after only a short period of down time.

The basic configuration is the lowest cost alternative, but for some applications it is desirable to be able to have a very high degree of confidence that calculations are performed correctly. A QMR system will do this since all components have a checker that alerts the system if a mistake is made. But a QMR configuration may be overkill for some applications that can tolerate an occasional system failure, as long as they are confident that the computations are correct when they do complete. FRC configurations offer an alternative between the basic and QMR approaches. Adding a second set of checker components to each module improves the error detection capabilities of the system, providing "high-confidence" computing. No single hardware failure will go undetected and corrupt the results of a critical computation. FRC insures that any error is caught before it can propagate to another module in the system. FRC alone does not provide automatic hardware recovery like a QMR system, but it does detect errors as soon as they occur so that the system does not become corrupted. It is then the responsibility of system software to implement a "self-healing" strategy where the faulty resource is disabled and the system reinitialized.

The software configurability of a BIU/MCU system allows a system to use a combination of the above strategies. Software can configure a system as a full QMR system in the morning for critical applications, and then switch to an FRC-only system in the afternoon. This doubles the system throughput (twice as many processors are working in parallel) without making any hardware changes.

LATENT FAULTS

Earlier sections of this chapter have discussed how hardware faults are detected and handled by the interconnect architecture as it operates. The interconnect system can be used to detect another class of faults known as latent faults. Latent faults can exist in a part of the system which is normally dormant. Under normal operation of the system, it would take a long time before the fault reached an interface where it would be detected by the other mechanisms. The iAPX 432 interconnect system can be used to exercise the normally dormant parts of the system in a strategy to uncover the latent faults. Table 1-2 summarizes some of the areas in which latent faults may exist and the methods used to uncover the faults.

Table 1-2. Exercising Latent Faults

DORMANT AREA	EXERCISE
Memory Locations	MCU periodically reads every array location (scrubbing).
Detection Mechanisms	Software* periodically forces error conditions into the detection mechanisms.
Reporting Mechanisms	Software* periodically initiates and observes error reports.
Recovery Mechanisms	Software* periodically invokes recovery operations.
	*Special commands to support exercising dormant areas are provided in BIUs and MCUs.

FAULT-TOLERANT SYSTEM DESIGN RESPONSIBILITIES

The interconnect architecture and the VLSI components provide a stable base for developing fault-tolerant iAPX 432 systems. The iAPX 432 interconnect components address the issues concerning fault tolerance which are encountered when constructing the iAPX 432 central system.

A number of system-wide issues remain the responsibility of the iAPX 432 system designer. These issues include:

- A fault-tolerant I/O system
- Fault-tolerant power supplies and distribution method
- A fault-tolerant method for clock generation and distribution
- The electrical and physical provisions for on-line repair
- Environmental issues, such as system cooling

SUMMARY

The iAPX 432 interconnect architecture provides a standard VLSI method for constructing multiple processor VLSI computer systems. The iAPX 432 interconnect architecture is implemented by a pair of VLSI components, the Bus Interface Unit (BIU) and the Memory Control Unit (MCU). Together with iAPX 432 processors, these components permit the construction of modular, extensible, multiprocessor computer systems. The components are designed to support the construction of fully fault-tolerant iAPX 432 systems. However, there is no penalty in performance or in cost for those applications that do not require fault tolerance.

The 432 fault-tolerant mechanisms are designed to provide a flexible and complete solution to the problems of fault-tolerant hardware. For basic systems (those without checkers for error detection or QMR for recovery), a user may decide to use only a few detection mechanisms and provide recovery only for transient errors. This functionality comes at no additional charge in the VLSI interconnect system. To reduce maintenance costs and increase system availability, a system may use all of the detection mechanisms (i.e., may add checker components) but may not add any extra recovery capability (i.e., may not marry self-checking modules into a fault-tolerant QMR module). Continuous operation is available to the user who adds the extra recovery capabilities.

None of the fault-tolerant mechanisms reduce system performance. Systems that do not require the highest level of fault tolerance are not penalized in any way (cost, size, or performance) for the unused fault-tolerant capabilities. Increased levels of fault tolerance are achieved by replicating the iAPX 432 VLSI components. The hardware fault tolerance in the iAPX 432 is transparent to application software. The system's fault-tolerant capabilities may be changed without any changes to the application software system.

A MODEL FOR FAULT HANDLING

To provide a conceptual framework for our thinking, this chapter presents a model for systems under fault conditions. Our first observation is that a fault handling cycle has three phases: a fault is generated, detection of the fault occurs, and then a recovery procedure is activated. Fault generation ranges from hardware errors to software bugs. We define the detection phase to include detection, diagnosis, logging, and reporting of the fault. During this phase, system resources detect the presence and extent of the failure and pass this information on to the recovery mechanisms. After fault detection, recovery mechanisms are employed to mask the effects of the fault from the rest of the system and possibly isolate and repair the faulty subsystem.

Next, we assume that a system is made up of a hierarchy of these fault handling cycles. Although the definition of the levels varies depending on system design and the technology of implementation, we believe that these levels exist in every system. For clarity, five levels that exist in most current systems are described below:

1. Component Level. This is the lowest level of hardware that will be considered in the system. Failures are generated by faults within the component itself. An example of such a failure is the loss of a memory bit within a RAM chip. This could be detected by employing checksums across the internal rows of storage cells in the chip. Recovery might be implemented by an on-board associative memory, which provides back-up storage for bad cells in the normal array. By employing fault handling strategies such as the ones mentioned, higher levels in the system may be totally unaware that a fault occurred.

2. Module Level. This level is composed of a group of interconnected components (such as a GDP, IP, or memory module). The failures at this level are generated by the components or the signal paths connecting the components. An example of such a failure at this level would be the loss of a memory bit in a 32-bit word of a memory array (e.g., a bad solder joint at a RAM data-out pin). This could be detected by employing parity to cover single failures within a memory word. Recovery could be accomplished by replacing the parity bit with a single error-correcting Hamming code (ECC).

3. Hardware System Level. This level consists of a network of interconnected modules. Failures may occur when either a module or the interconnect network fails. An example of such a failure might be an incorrect response from the memory module. This could be detected by duplicating the memory module and comparing the outputs. Recovery is possible if a third memory module is added and majority voting is performed at the outputs.

4. System Software Level. The operating system is responsible for providing expanded service to the application software. Failures may occur when defects exist in the algorithms. An example might be allowing a deadlock situation to occur. This can be detected by utilizing some secondary routines which monitor the operation of the system resources. Recovery might be accomplished by changing some of the previous resource allocation decisions.

5. Application Level. This level completes the system description. This software may have defects in its algorithms or specifications. The human interface of the system may even be used to detect and recover from application failures.

We also assume that these system levels are connected. Failures that occur at one level may flow up to higher levels if either the detection or recovery at that level is inadequate to handle the fault. Each time a failure is reflected up, the detection mechanisms at the next level will treat the failure as if it were originally generated at the higher level. This treatment causes a loss of information about the fault by potentially masking the true cause of the failure, as well as by increasing the amount of the system which must be considered suspect. An example of fault detection being reflected up is an undetected memory error that appears as an invalid data structure at one of the software levels. An example of recovery being reflected to higher levels is an uncorrectable memory error in an application program segment. It may be possible for the operating system or the user to handle this fault. Recovery can also be reflected up because of improper detection or incorrect operation of the recovery mechanisms. An example of this is a 3-bit memory error that is detected as a single-bit error by the Hamming code and subsequently "corrected."

The model we have constructed is pictured in Figure 2-1.

We deduce from the model several key points about system fault handling. Faults can occur at many different levels in the system. Each level has its own characteristics: different detection and recovery strategies will be appropriate for different levels. Faults not handled at one level will propagate up to the higher levels in the system. Higher levels have more complex environments, which make recovery a more complex and slow task: failure modes increase in complexity, the interaction between subsystems grows, and the original source of the failure becomes more ambiguous. Thus, faults should be handled at the lowest possible level.

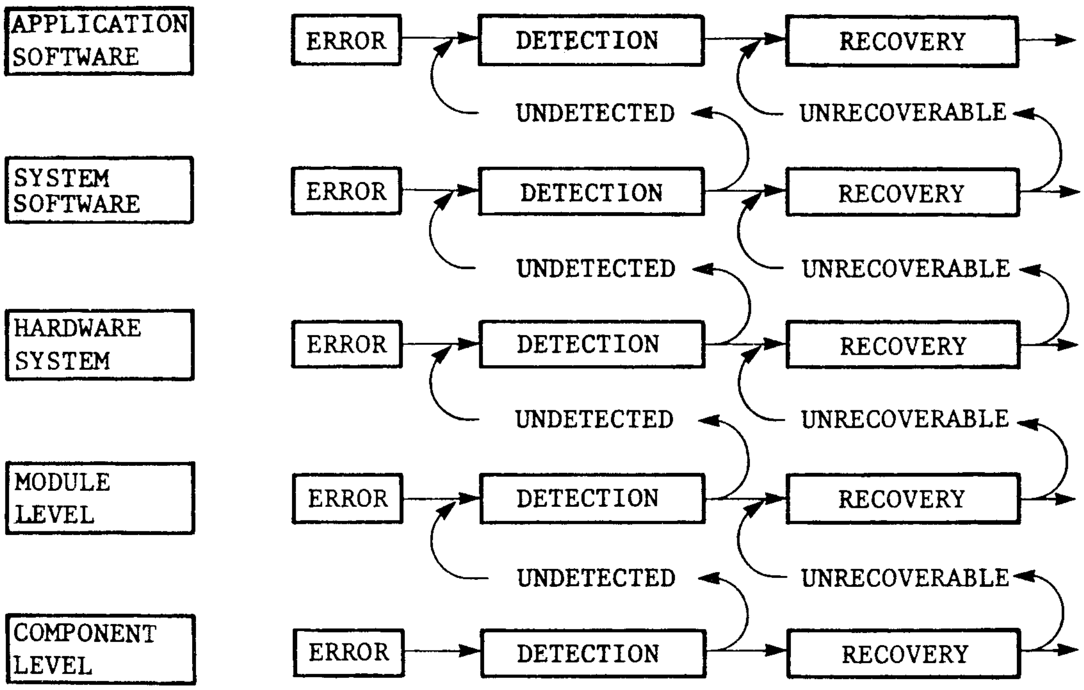


Figure 2-1. Fault Handling Model F-0418

PHILOSOPHY

The goal of the iAPX 432 fault handling approach is to provide general-purpose, adaptable, and software-transparent fault-tolerant capabilities. Our design philosophy has two fundamental principles. First, the number of hardware errors which will be reflected up to the software levels of the system must be minimal. Second, all of the fault handling mechanisms must be designed as independent and orthogonal capabilities.

Figure 2-2 shows the location of the barrier that prevents hardware failures from being reflected into higher layers of the system. It is important to note that it is impossible for any system to detect and recover from all failures that might occur. However, the iAPX 432 hardware reduces the rate of failures reflected into the software to such a low level that in virtually every fault-tolerant application the software system can ignore errors reflected up by the hardware. (At this time we do not have enough data to publish an absolute value for the rate of failures reflected into the software. However, we have done enough modeling to confirm that the rate is extremely low.)

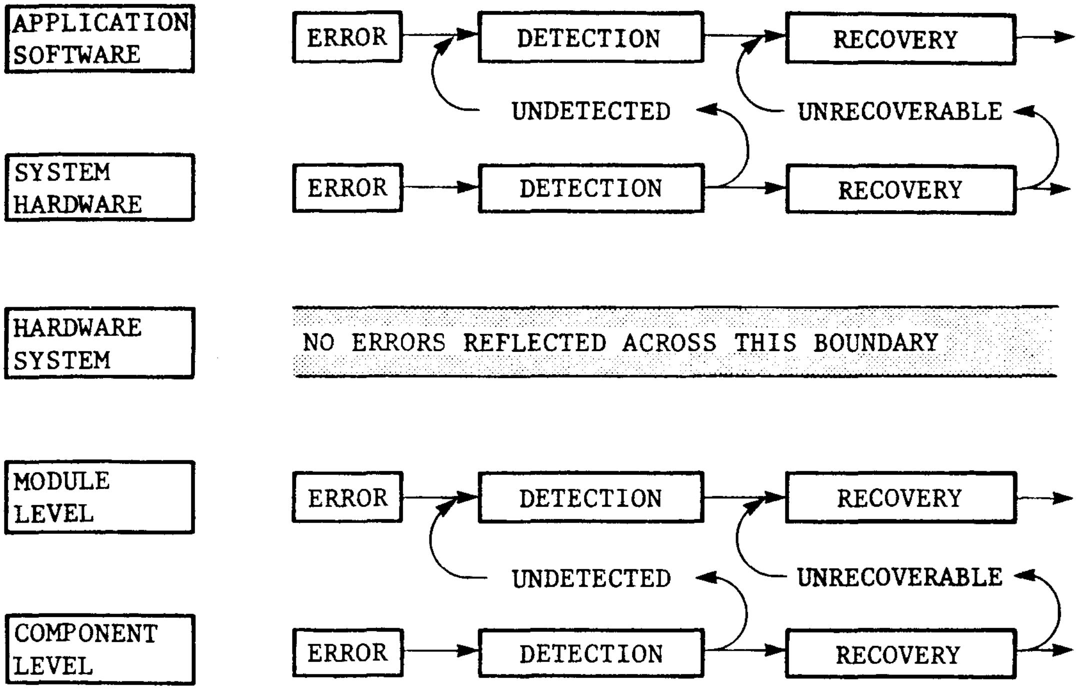


Figure 2-2. Separation of HW and SW Layers F-0419

The reflection of errors needs to be minimized to prevent information overload at higher levels in the system structure. If all failures are allowed to propagate to the top, the system becomes overloaded and loses its ability to react to the fault conditions. The complexity of the failure modes may make implementation impossible or force a reduction in the completeness of fault coverage or generality of operation.

By performing detection and recovery from hardware failures at a low level in the system, a more general and complete solution is possible. This approach divides the responsibilities for fault tolerance, allowing faster, simpler, and more general solutions to fault detection and fault recovery. The mechanisms for detection and recovery from software errors need only address the set of faults that can be generated at those levels. (For example, the capability-based addressing mechanism in the iAPX 432 processors addresses only errors in the software system. If it were possible for memory errors to propagate up to this level, then the mechanism would need to be much more elaborate. It would need to support the updating and accessing of two sets of object and access descriptors in case there were a failure in a memory location containing a needed descriptor.) By controlling and reducing the amount of errors reflected up to the next level, parallel and independent development may proceed on different levels (hardware, system software, applications). The designers at one level can assume that lower levels will always provide consistent and correct operation.

All of the fault handling mechanisms are designed to be orthogonal. Expansion of bus bandwidth, logical resources, detection capabilities, or redundancy may be done without any side effects on the rest of the system. Minimizing the interaction between these variables provides the system with a very flexible and modular basis for growth and adaptation to the application environment. System capabilities may be added or removed without any impact on the application software. There is no penalty in performance, cost, or system size for those fault-tolerant mechanisms not used in a system.

Although all of the hardware fault handling occurs without software assistance, software remains responsible for managing the overall fault-tolerant policy. This division of labor allows the software to tailor the resources of a system to the needs of an application without giving up any of the benefits derived from placing the fault handling mechanisms in the hardware. Software may also periodically test the detection and recovery mechanisms while the system is on-line. This allows any latent errors in the system to be uncovered before another error forces the system to face a double-failure condition.

IMPLEMENTATION

Three basic principles form the foundation for the implementation of the iAPX 432 fault handling mechanisms. First, the fault-tolerant functionality is achieved by replication of VLSI components. Second, the machine is partitioned into a set of confinement areas. These areas form the basis for error detection and recovery. Third, only bus-oriented communication paths are used to provide system interconnection.

VLSI replication is fundamental to achieving effective use of VLSI technology. To be successful, each VLSI component must reach high-volume production. In the iAPX 432, this high-volume production is achieved by building a wide range of systems from a small set of VLSI components. The same components provide modular expansion of performance, memory storage, detection, and recovery capabilities. There are no special-purpose components aimed solely at fault-tolerant functions.

The purpose of a confinement area is to limit damage from error propagation and to localize the faulty area for recovery and repair. A confinement area is defined as a unit (module or memory bus) of the system which has a limited number of tightly controlled interfaces.

Detection mechanisms are placed at every interface to ensure that no inconsistent data can leave the area and corrupt other confinement areas. When an error occurs in the system, it is immediately isolated to a confinement area. The error is known to be in that confinement area, and all other confinement areas are known to be error-free.

By defining confinement areas, we provide a conceptual framework for the systematic and coherent placement and definition of the detection mechanisms. The confinement areas also provide a conceptual view of the system under fault conditions. This clarifies the external (software) view of the hardware, and eliminates the need for diagnostic probing as a method of fault isolation.

All communication in the iAPX 432 system is done over buses. There are no point-to-point signals or daisy-chained signals. This makes modular growth possible, since no signal definition is dependent on the number of resources in the system. This approach also makes on-line repair possible. The presence or absence of any module cannot prevent communication between any other modules. The memory bus defined by the BIU and MCU provides a uniform and regularly structured communications path that supports the modular expansion of both fault-tolerant and standard system capabilities.

CHAPTER 3
FUNDAMENTAL SYSTEM STRUCTURES

INTRODUCTION

Three basic mechanisms provide hardware fault tolerance for the iAPX 432:

1. Error confinement

2. Error reporting

3. Error recovery

This chapter will describe each of these mechanisms at a high level of abstraction, providing a full system view of the fault handling mechanisms before describing each mechanism in detail.

In the iAPX 432 there are three distinct steps in responding to an error. First, the error is detected and localized to a confinement area in the system. Next, the error is reported to all of the modules in the system. (This prevents the incorrect data from propagating into another confinement area and provides all of the modules with the information required to perform recovery.) Finally, the faulty confinement area is isolated from the system, and recovery occurs using redundant resources available in the system.

ERROR CONFINEMENT

Figure 3-1 shows the four types of confinement areas in a iAPX 432 system. There is a confinement area for each module and memory bus in a system. These confinement areas were chosen because modules and memory buses are the natural building blocks for iAPX 432 systems. Thus, when an error is detected, it is confined to one of the system building blocks. This allows the recovery and repair strategies to be built around the replacement of system building blocks. When a module or bus has its confinement mechanisms activated, it can be viewed as a self-checking unit. The operation of a self-checking unit is designed so that no inconsistent data will be allowed to leave the unit and corrupt another confinement area. Detection mechanisms reside at every interface, and all data is checked as it flows across the interface between confinement areas.

The GDP confinement area is exactly the same as the GDP module. The only interfaces to a GDP confinement area are the memory buses. The BIUs are responsible for checking all of the information that leaves the GDP module. No information (control, address, or data) can leave a GDP confinement area without first being checked for correctness by one of the BIUs in the module. Error detection is performed by duplicating the GDP module.

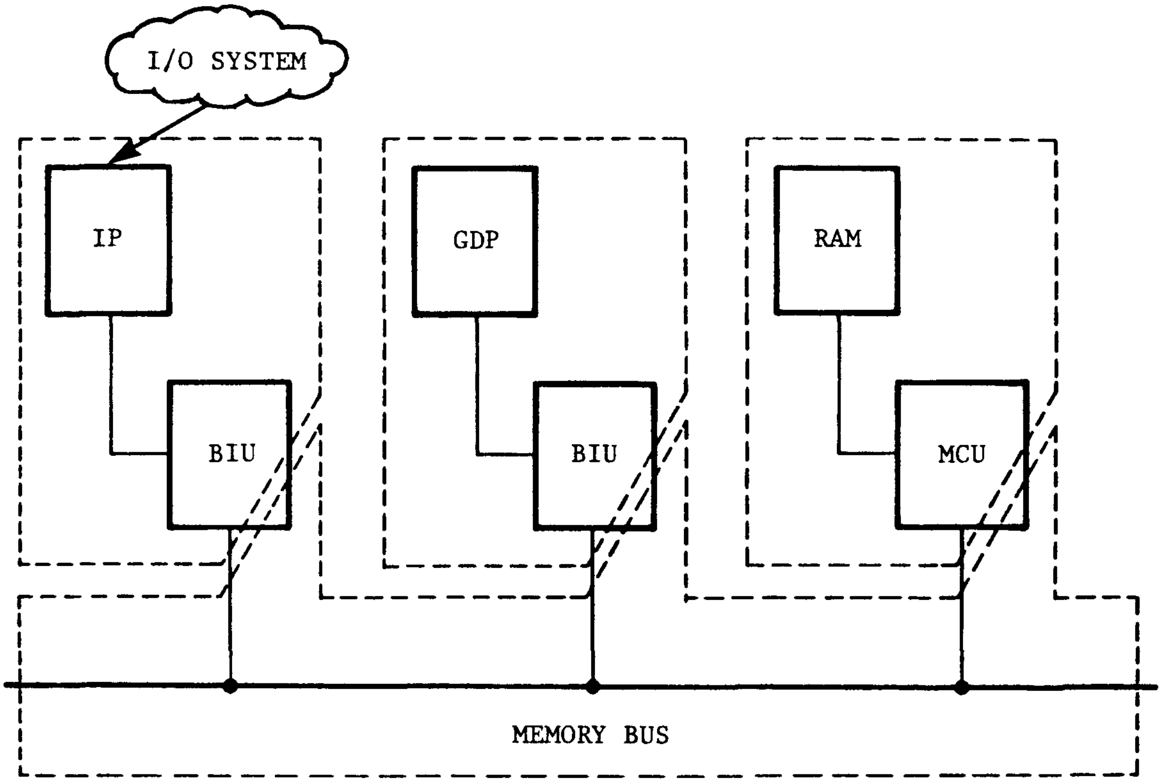


Figure 3-1. iAPX 432 Confinement Areas F-0420

The IP confinement area is exactly the same as the IP module. An IP module has interfaces to the memory buses in the system, plus an interface to an external I/O subsystem. The interfaces to the memory buses are checked by the BIUs in the same manner as those in the GDP confinement area. The IP component is responsible for checking any data that leaves the confinement area via the peripheral subsystem (PS) bus. No information can leave an IP confinement area without first being checked for correctness by one of the BIUs or by the IP. The peripheral subsystem is not a confinement area. At this time, the application hardware or software must apply its own detection mechanisms to this subsystem. The PS bus represents a firewall between the central system and the I/O subsystem. The IP confinement area checks data as it leaves the IP; the application HW and SW must check data that leaves the I/O subsystem and enters the IP module. Error detection is performed by duplicating the IP module.

The memory confinement area is exactly the same as the memory module. A memory module has interfaces to two of the memory buses in the system. The MCU is responsible for checking all information that leaves the memory confinement area. No information can leave the confinement area without first being checked for correctness by the MCU. Error detection is performed by duplicating the MCU and applying an ECC code to the memory array.

Each memory bus confinement area includes a memory bus and the interface logic residing in the BIUs and MCUs attached to the memory bus. Each memory bus has interfaces to all of the GDP and IP modules and to some of the memory modules. Every node (BIU or MCU) that is attached to this bus is responsible for checking all of the information that flows off the memory bus and into its module. No information can leave the memory bus and enter a module without first being checked for correctness by either a BIU or an MCU. Error detection is performed primarily by parity bits.

An example processor memory operation may help to clarify the operation of the confinement areas. (This example is shown graphically in Figure 3-2.) Assume a GDP makes a read request to a memory location. That request will be mapped to a BIU on the addressed memory bus. As the information flows onto the memory bus it will be checked by the BIU. If there has been any failure in the GDP confinement area (GDP, processor bus, BIUs, etc.), it will be detected at this time. The information flows across the memory bus and into the addressed memory module. Before the information is accepted by the module, the MCU checks it for correctness. If a failure is detected, it is confined to the memory bus because the information was valid when it left the GDP confinement area. The MCU performs the memory operation and returns data onto the memory bus. As data flows onto the bus it is checked for correctness by the MCU. As the data flows into the GDP module from the memory bus it is checked for correctness by the BIU before being used by the GDP module.

The confinement area interfaces provide very tight error control and isolate the failure to one of the building blocks present in the system. The only remaining question concerns checking the detection mechanisms. Most of the detection mechanisms are self-checking. (The detection circuits are checked as part of normal operation.) Those circuits that are not self-checking can be exercised during normal system operation to flush out any latent faults.

REPORTING

Immediately upon detecting an error, a message is broadcast to all the nodes in the system. This error report message identifies the faulty confinement area, the type of error that occurred, and whether the error is permanent or transient. There are two reasons for sending this error report. First, it informs the rest of the system that an error has occurred. (This prevents other confinement areas from using the inconsistent data.) Second, it provides the necessary information for system recovery. After recovery, the error message is recorded in a log register in every node in the system. This log is available to software and is useful in monitoring the health of the system.

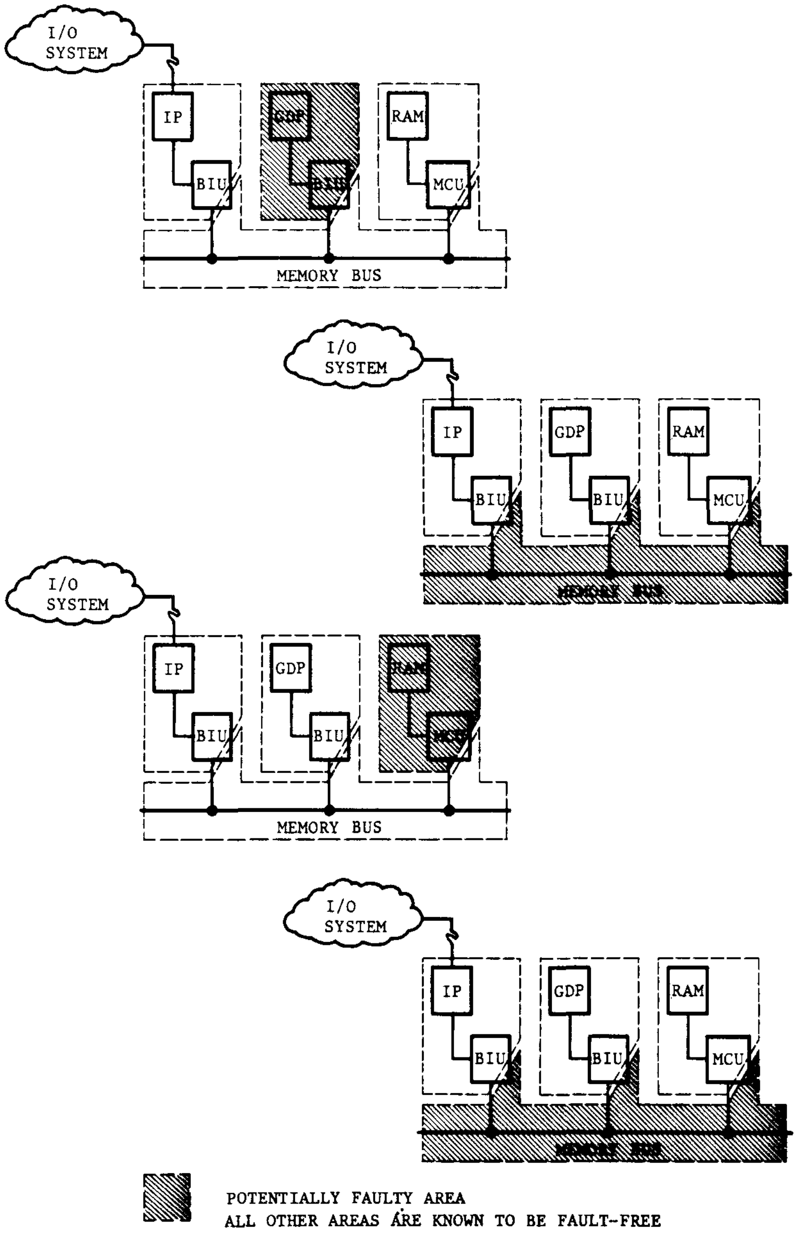


Figure 3-2. Confinement Area Operation - An Example F-0421

The error messages are broadcast over a set of serial buses, which are totally independent from the buses used during normal operation. However, this network of serial buses follows exactly the same topology as the buses used for normal operation. A failure on one of these buses is limited to one of the confinement areas discussed earlier. The error reporting circuitry may be tested during normal operation to uncover any latent faults.

RECOVERY

The recovery process begins after an error report message has been broadcast around the system. Recovery is a distributed operation on the iAPX 432. Each node in the system reads the error report message and decides what recovery action needs to be taken.

For recovery to be successful, there must be redundant resources available in the system. There are five redundancy mechanisms in the iAPX 432. Two of these mechanisms provide redundant information useful in recovering from transient errors, while the other three mechanisms allow recovery from permanent errors in the system. These redundant resources cover the entire system and allow recovery from any detected error. The presence of redundant resources has no impact on system performance.

For transient errors:

- Each BIU maintains an internal buffer, which allows outstanding processor requests to be retried if a transient error occurs.

- A single-bit correcting ECC code is applied to each word in the memory arrays. Although this provides redundancy for both permanent and transient errors, its primary purpose is to correct soft errors that occur in DRAMs.

For permanent errors:

- Every module in the system may be paired with another module of the same type. This module pair operates in lock step and provides a complete and current backup for all of the state information in the module. This mechanism is known as module shadowing.

- Each memory bus in the system may be paired with another memory bus. During normal operation the buses run independently. Both contribute to the total bandwidth available in the system. However, if one bus fails, the other bus is capable of handling the bus requests that normally would have been handled by the failed bus.

- Inside of each memory module a spare bit may be added to each word in the memory array. If one bit in the array fails, the spare bit can be switched in to replace the failed bit.

For transient errors, all of the outstanding accesses will be retried, and the MCUs will return corrected data if there are any single-bit errors in the memory arrays.

For permanent errors, the redundant resource is switched in to replace the failed unit. This switch is done on a node-by-node basis; there is no centralized element that controls the switch. Each node knows which module or memory bus it is backing up (shadowing). If the error report identifies its partner as the faulty unit, then it becomes active and takes over operation for the faulty unit. After the resource switch is complete, all of the outstanding accesses are retried. This allows operation to resume at a point before the failure corrupted data.

These reconfiguration and recovery actions are performed by the hardware without any software intervention. After recovery is complete, the hardware informs the system software of the error and subsequent recovery actions. System software now makes policy decisions regarding the optimum system configuration, given the resources remaining in the system. These policy decisions are carried out while normal system operation continues.

Figures 3-3 and 3-4 show two examples of recovery operations.

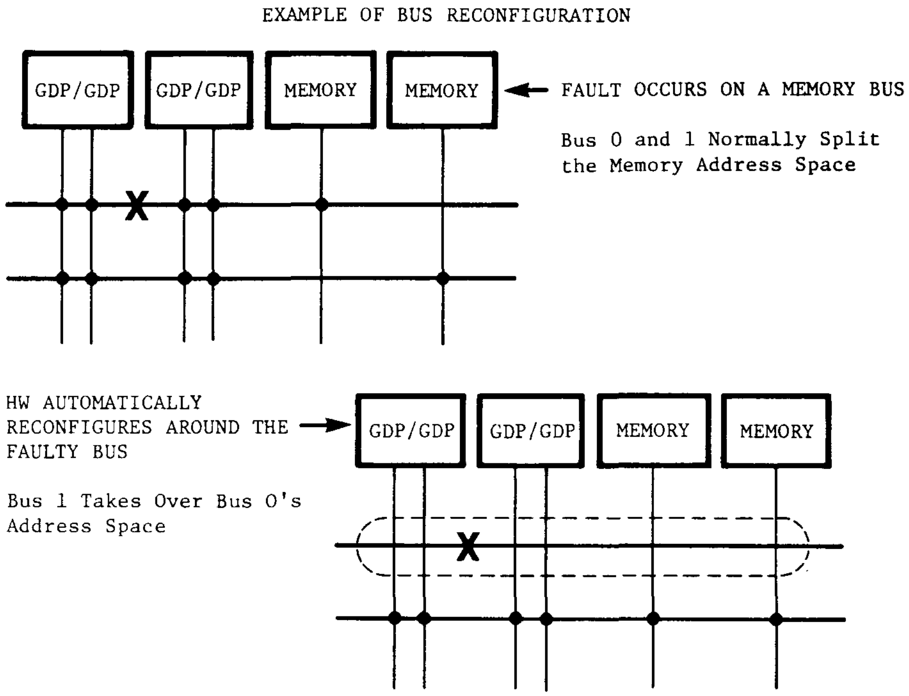

EXAMPLE OF BUS RECONFIGURATION
GDP/GDP
GDP/GDP
MEMORY
MEMORY
FAULT OCCURS ON A MEMORY BUS
Bus 0 and 1 Normally Split the Memory Address Space
X
HW AUTOMATICALLY RECONFIGURES AROUND THE FAULTY BUS
Bus 1 Takes Over Bus 0's Address Space
GDP/GDP
GDP/GDP
MEMORY
MEMORY
X

Figure 3-3. Permanent Bus Error Recovery F-0422

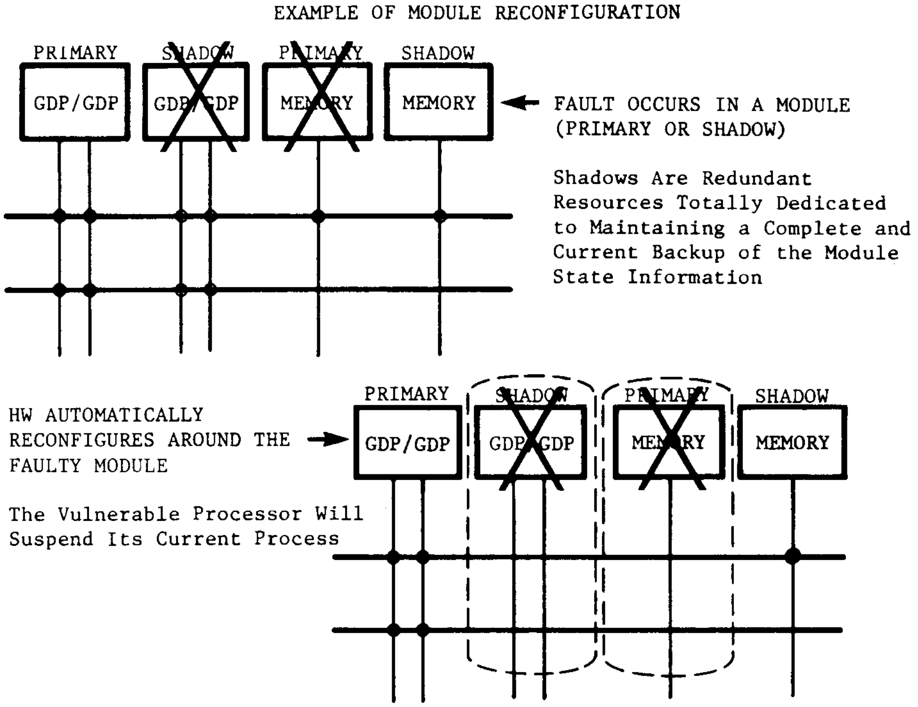

EXAMPLE OF MODULE RECONFIGURATION
PRIMARY
SHADOW
PRIMARY
SHADOW
GDP/GDP
GDP/GDP
MEMORY
MEMORY
FAULT OCCURS IN A MODULE (PRIMARY OR SHADOW)
Shadows Are Redundant Resources Totally Dedicated to Maintaining a Complete and Current Backup of the Module State Information
HW AUTOMATICALLY RECONFIGURES AROUND THE FAULTY MODULE
The Vulnerable Processor Will Suspend Its Current Process
PRIMARY
SHADOW
PRIMARY
SHADOW
GDP/GDP
GDP/GDP
MEMORY
MEMORY

Figure 3-4. Permanent Module Error Recovery F-0423

SUMMARY

The iAPX 432 fault-tolerant mechanisms are designed to provide a flexible and complete solution to the problems of fault-tolerant hardware. For basic systems, a user may decide to use only a few detection mechanisms and provide for transient errors only. This functionality is included at no extra cost in the VLSI interconnect system. To reduce maintenance costs and increase system availability, a system may use all of the detection mechanisms, without adding any extra recovery capability. Nonstop operation is available to the user by adding the extra recovery capabilities.

None of the fault-tolerant mechanisms reduce system performance. Systems that do not require the highest level of fault tolerance are not penalized in any way -- in cost, size, or performance -- for the unused fault-tolerant capabilities. Increased levels of fault tolerance are achieved by replicating the iAPX 432 VLSI components. The hardware fault tolerance in the iAPX 432 is transparent to application software. The system's fault-tolerant capabilities may be changed without any changes to the application software system.

CHAPTER 4
OVERVIEW OF THE FAULT HANDLING MECHANISMS

INTRODUCTION

The previous chapter described the general structure of the fault handling mechanisms. This chapter moves one level deeper into the implementation of the mechanisms.

CONFINEMENT AREAS/DETECTION MECHANISMS

This section describes the actual construction of the confinement areas described in the previous chapter. With the confinement areas defined, detection mechanisms that provide a high level of fault coverage and are cost-effective for the given confinement area were selected. Because the confinement areas are independent units, there is no requirement for using the same detection mechanisms in all of the confinement areas. The iAPX 432 system uses five types of detection mechanisms.

1. Parity

2. Loopback check

3. Hamming code (ECC)

4. Functional Redundancy Check (FRC)

5. Timeout

The loopback check and FRC are explained in more detail in the following paragraphs. The reader is assumed to be familiar with the other three detection mechanisms.

The loopback check is used to supplement the parity detection mechanism on the memory buses. The parity detection mechanism assumes that failures on the signal lines are independent events. Because TTL drivers are used to interface to the memory bus, there are failures that violate this assumption. A single failure (power, enable, etc.) could cause inconsistent data to be generated by all of the drivers in a package. The loopback check provides a detection mechanism that can detect these common mode failures, which are undetectable by parity.

Figure 4-1 shows the basic loopback concept. One spare buffer is reserved out of each package housing the buffers that drive the memory bus. Each interface will normally consist of three TTL packages. The BIU or MCU sends an oscillating signal through all of the spare drivers. The oscillating signal is then fed back to the BIU or MCU. This received signal will oscillate correctly only if all of the spare drivers in the memory bus interface are operating correctly. Any single failure that corrupts all of the drivers in a package will be detected by the BIU or MCU by way of the loopback check.

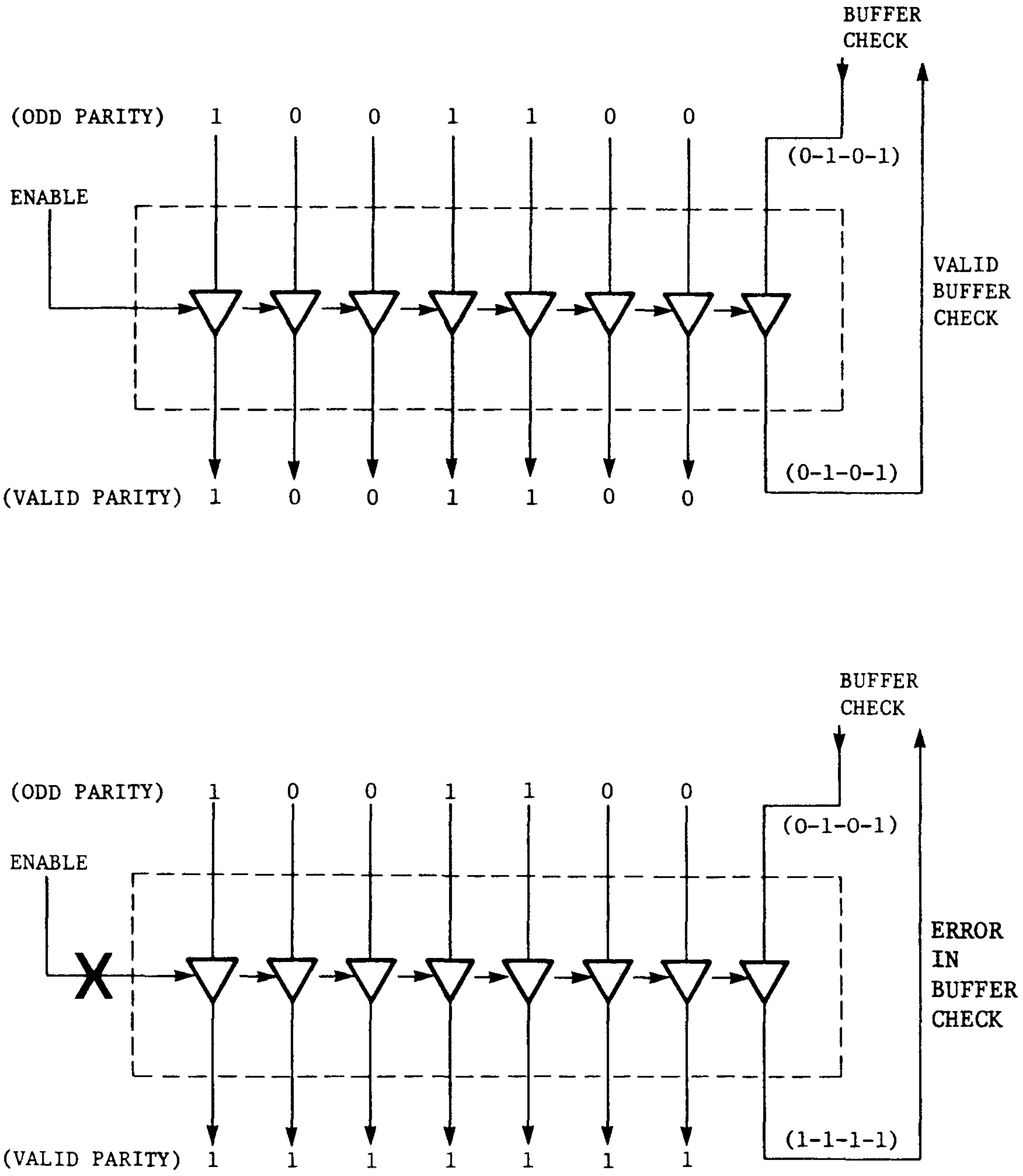

BUFFER
CHECK
(ODD PARITY) 1 0 0 1 1 0 0
(0-1-0-1)
ENABLE
VALID
BUFFER
CHECK
(0-1-0-1)
(VALID PARITY) 1 0 0 1 1 0 0
BUFFER
CHECK
(ODD PARITY) 1 0 0 1 1 0 0
(0-1-0-1)
ENABLE
ERROR
IN
BUFFER
CHECK
(1-1-1-1)
(VALID PARITY) 1 1 1 1 1 1 1

Figure 4-1. Buffer Check Detection Mechanism F-0424

The use of encoding techniques for reduction of information redundancy is not a practical proposition when the information undergoes transformation. To provide error detection for data transformation, the iAPX 432 system hardware provides complete duplication of all circuitry that transforms data. Additional circuitry compares the results generated by the two data transformations and detects any errors. This detection mechanism is called Functional Redundancy Checking (FRC).

In order to provide FRC, the hardware is divided up into blocks of functionality which may include any number of components and interconnections. Each block is then duplicated and equipped with comparison logic. One of the pair is selected to be the master; the other becomes the checker. Selection is done at initialization time. The master and checker run in clock cycle lock step. All of the comparison and error detection logic is located inside the iAPX 432 system VLSI components.

The master block is responsible for carrying out the normal operation of the system. (For example, it behaves just like conventional circuitry.) The checker block has its outputs disabled, and instead of presenting its output it checks the output of its master. It is responsible for duplicating the operation of the master and using its comparison circuitry to detect any inconsistency between the master and its checker. Figure 4-2 is a block diagram of the FRC detection mechanism.

This method of checking will detect any operational error occurring in either the master or the checker block of functionality. It will not, however, cope with errors occurring in both blocks simultaneously (e.g., a design fault).

The only circuitry that must be relied on in the event of a failure is the comparison and fault reporting logic of the checker. Periodic testing of this circuitry during the normal operation of the system will uncover latent failures in this critical block of circuitry.

GDP CONFINEMENT AREA

The GDP confinement area is formed by applying the FRC and loopback check detection mechanisms at every memory bus interface. The GDP confinement area is shown in Figure 4-3. It is important to note that FRC is applied at a module level, rather than at the component level (as was done in earlier releases of the IP and GDP components). All logic in the module operates normally except for the memory bus interface drivers in the checker BIUs.

Each checker BIU constantly monitors all of the signals at the memory bus interface. If there is ever a disagreement between the master and checker BIUs, then an error has occurred in the confinement area. The FRC mechanism will detect errors in either the master or checker blocks of logic. This includes the GDP, the processor bus, the support logic, and the BIUs (except for memory bus arbitration logic, which is included in the memory bus confinement area).

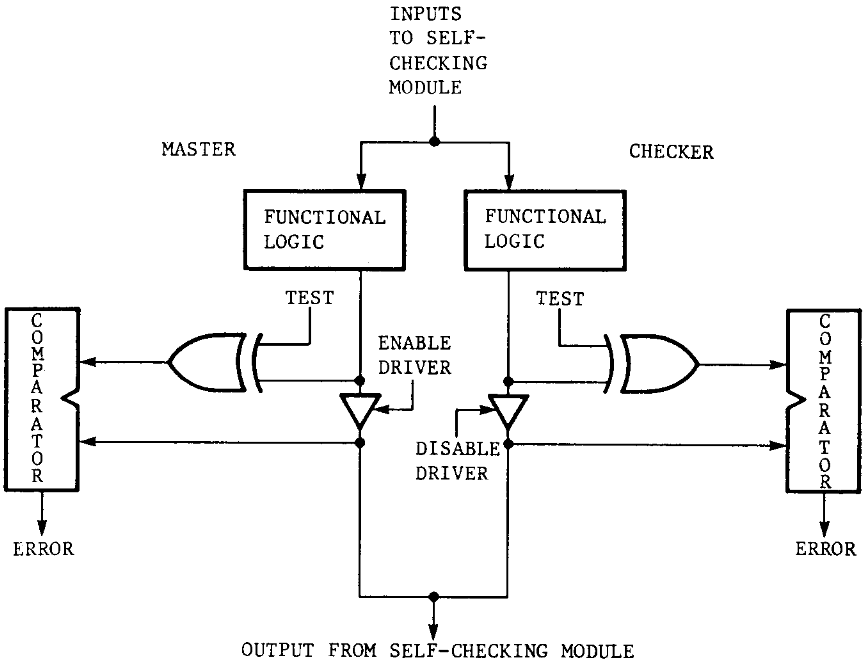


Figure 4-2. Functional Redundancy Checking F-0425

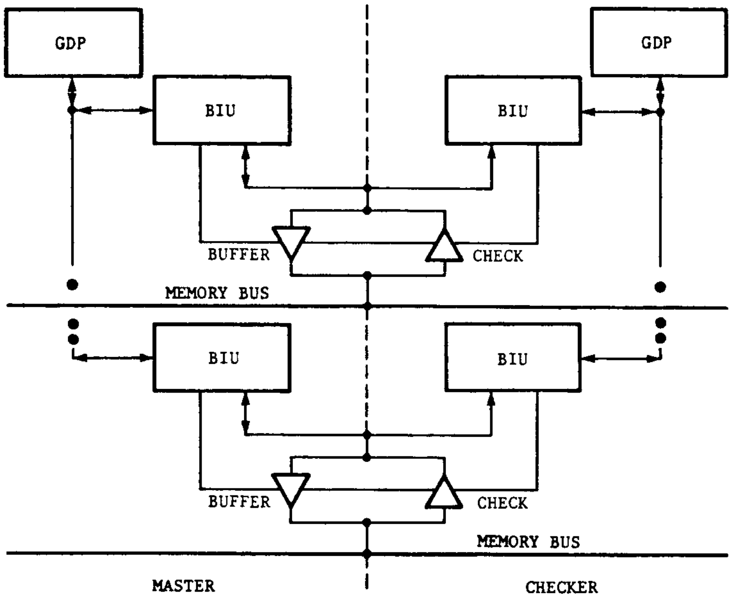


Figure 4-3. GDP Confinement Area F-0426

The loopback check, which is used at each memory bus interface, will detect failures in the TTL buffers interfacing this module to the memory bus.

By applying these two detection mechanisms a GDP module can be made into a self-checking unit. No information can leave the module without being checked for correctness.

IP CONFINEMENT AREA

The IP confinement area is shown in Figure 4-4. This confinement area is identical to the GDP confinement area except for the addition of an interface to the peripheral subsystem. The IP confinement area is established by using the FRC and loopback detection mechanisms.

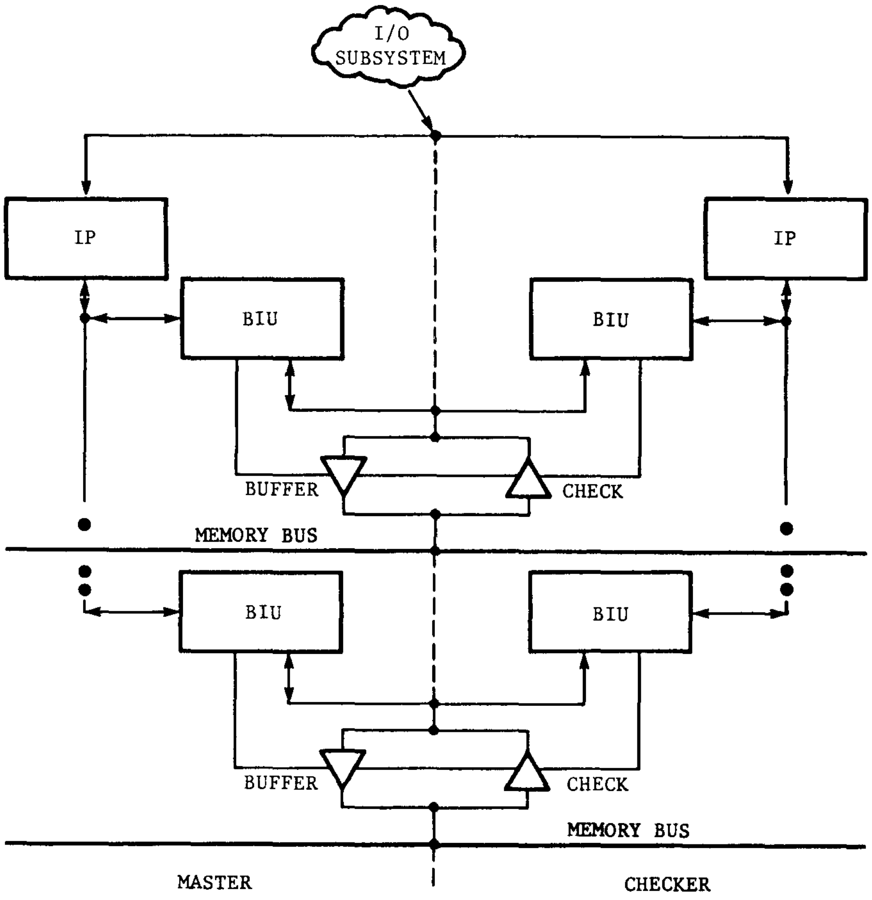


Figure 4-4. The IP Confinement Area F-0427

The interfaces between the IP module and the memory buses are handled in exactly the same way the GDP module handles the interfaces. A firewall is established between the IP module and the I/O subsystem by applying the FRC detection mechanism to this interface. The checker IP has its PS bus drivers disabled. The checker IP constantly monitors the PS bus for errors. If there is ever a disagreement between the master and checker IPs, then an error has occurred in this confinement area. This FRC mechanism will detect errors in either the master or checker blocks of logic. This includes the IP, processor bus, support logic, and the BIUs. It is important to realize that only the PS interface of the checker IP is in checker mode. The processor bus interface of the IP operates normally. There is no check on the operation of the I/O subsystem. This check must be performed by application hardware and software.

By applying these two detection mechanisms, an IP module can be made into a self-checking unit. No information can leave the module without being checked for correctness.

MEMORY CONFINEMENT AREA

Figure 4-5 shows the memory confinement area. FRC, loopback check, and ECC detection mechanisms are used to form the confinement area. FRC covers the operation of the MCU because it does data transformation. A single-bit correcting, double-bit detecting Hamming code is used to cover the memory array.

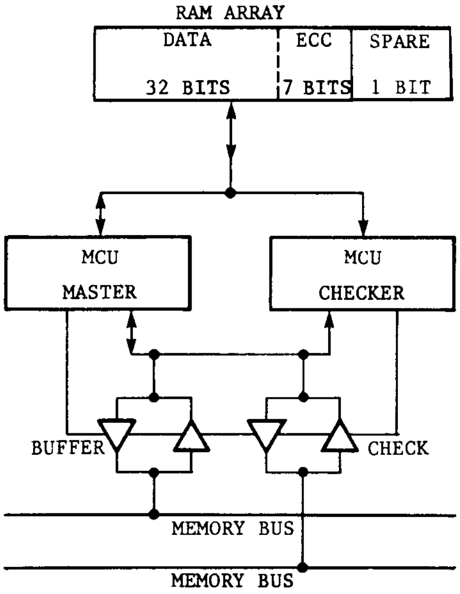


Figure 4-5. Memory Confinement Area

F-0433

Because the MCU interfaces directly to two memory buses, the detection mechanisms are applied in a slightly different manner. FRC is used to check the outputs from the master MCU in exactly the same way as was done with the BIUs in processor modules. The loopback check on memory modules must check two sets of buffers. Those drivers on the active bus must all be operating correctly, and those drivers on the back-up bus must all be turned off. The MCU loopback check tests both sets of buffers. The final step in sealing this interface of the confinement area is to prevent the memory module from ever corrupting data on both memory buses to which it is attached. The TTL buffers, which isolate the MCUs from the each memory bus, can be activated only if both MCUs agree on which bus to activate. Because of this reliability buttressing, no single failure can cause both sets of memory bus buffers to be active at the same time. This prevents a failure in the memory bus interface from corrupting both the primary and the back-up memory buses.

FRC is also applied at the interface between the MCU and the memory array. The checker constantly monitors all of the array signals to make sure that the master MCU is correctly managing and accessing the array. If the master MCU ever attempts to perform an incorrect operation with the array, it will be caught by the checker MCU.

A 7-bit ECC code is used for detecting errors which occur within the RAM array. The ECC code is computed from the data to be stored and the address of the storage location. The ECC coding allows the MCU to take the following actions:

- All single-bit errors in the data can be corrected.

- All single-bit errors in the address can be detected.

- All double-bit errors in data or the address can be detected.

- Multiple-bit (odd numbers of) errors are detected.

This detection mechanism is implemented totally within the MCU. ECC coverage is applied to refresh accesses as well as all normal accesses. On each refresh request, one location in the row is actually read and the ECC is checked. If a correctable error is found, the MCU corrects the data and writes the corrected value back to the array. This refresh operation is called scrubbing. Scrubbing is used to detect any latent faults that may exist in memory locations infrequently accessed by software. Because scrubbing is a part of the refresh operation, this function is available without any additional performance impact and without any software intervention. This virtually eliminates the possibility of double-bit errors in the memory array.

Error coverage is further amplified because the MCU performs all write operations as read-modify-write (RMW) operations. This allows the MCU to check that it has reached the correct memory address (by checking the ECC code) before it changes the data value in the array. This prevents valid data in one location from being overwritten by data intended for another location because of an address line failure.

These detection mechanisms provide a complete check on the operation of the memory module. They check that the memory operation is correctly carried out and that the correct data is accessed from the correct location. Data cannot leave the confinement area without passing through these checks.

MEMORY BUS CONFINEMENT AREA

Figure 4-6 shows the memory bus confinement area. The confinement area includes the backplane signals, the TTL buffers, and the arbitration sections of the BIUs and MCUs attached to this bus. Parity and duplication (FRC) are the detection mechanisms used in this confinement area.

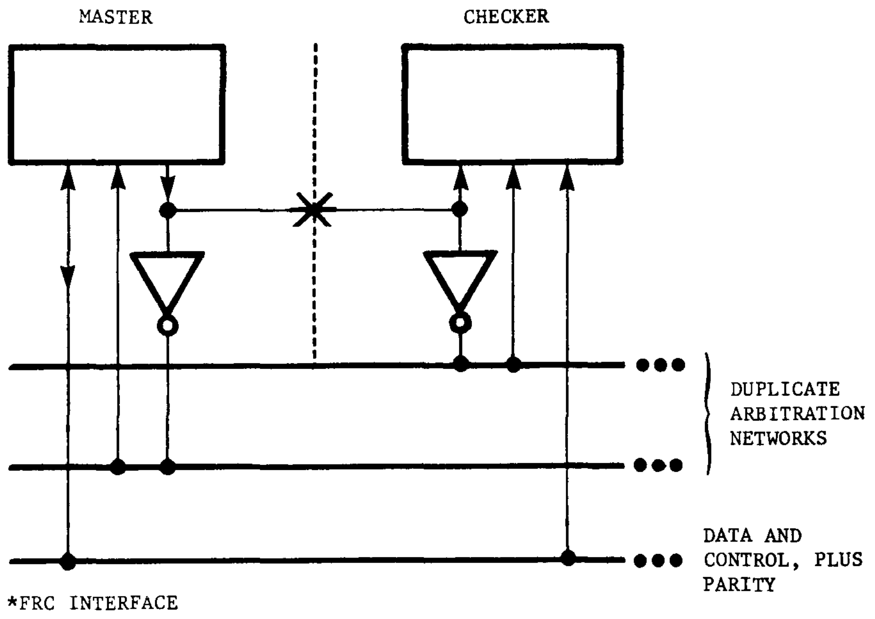


Figure 4-6. Memory Bus Confinement Area F-0434

Two interlaced parity bits are appended to the address/data and control lines on the memory bus. The parity bits are checked for validity by all nodes on the bus on every clock cycle. The parity bits will detect the following failures:

- All single-bit errors
- Multiple-bit (odd numbers of) errors
- Double-bit errors, when each parity bit covers only one of the errors (e.g., adjacent signal lines)
- Stuck-at-zeroes

The arbitration lines and circuitry are duplicated. One arbitration network is used by masters, while the other network is used by checkers. The masters are responsible for driving both arbitration networks. Any single error in the arbitration signals or in the arbitration state machines in the components will be detected. The error will be detected as an FRC error on the arbitration line outputs from the master.

These detection mechanisms check that the memory bus correctly transmits information between modules. Information can leave the memory bus confinement area only after being checked by the nodes attached to the bus.

OTHER POINTS OF INTEREST

At the interfaces between confinement areas it is possible for a single failure to be detected as multiple errors. To solve this problem, errors are given priorities based on the error type. This is the basic priority (from highest to lowest):

1. Module error (highest)

2. Arbitration error

3. Parity error

This ordering reflects the fact that a module failure may also appear as an arbitration or parity error. However, a bus failure can never appear as a module error. Thus, whenever a failure appears as multiple errors, only the highest priority error is reported.

Both the processor buses and the memory buses have timeouts. These timeouts are used to check for configuration errors. They are not used in the establishment of the confinement areas. The only type of error which uses a timeout is when an access is made to a nonexistent resource. This is a software configuration error, not a hardware error. Any error in the hardware will be detected by the mechanisms used to establish the confinement areas.

ERROR REPORTING

Error reporting is the backbone of fault isolation and recovery. When an error is detected, the node detecting the error reports the type and location of the error to all the other nodes in the system. (Note that errors detected at an IP:PS interface are not reported over the error reporting network. These errors are reported to the AP by means of an interrupt.) The error reporting system is designed so that, independent of the error in the system, each node not only receives an error report, but is guaranteed to receive the same error report. Error information is uniformly logged in all of the nodes in the system. With this information, each node then independently proceeds with the appropriate recovery procedure.

TOPOLOGY OF THE REPORTING NETWORK

The reporting network follows the same matrix topology as the normal data paths. There is an error reporting line associated with each memory bus and processor bus in the system. Figure 4-7 provides an overview of the topology. Each BIU is connected to one Bus Error Reporting Line (BERL) and one Module Error Reporting Line (MERL). Each MCU has connections to two BERLs, but at any instant only one of the connections is active. Because the error reporting system is fundamental to the correct operation of a fault-tolerant system, error detection and masking mechanisms are built into each error report line. These mechanisms guarantee that the error report network will function correctly in the presence of single failures.

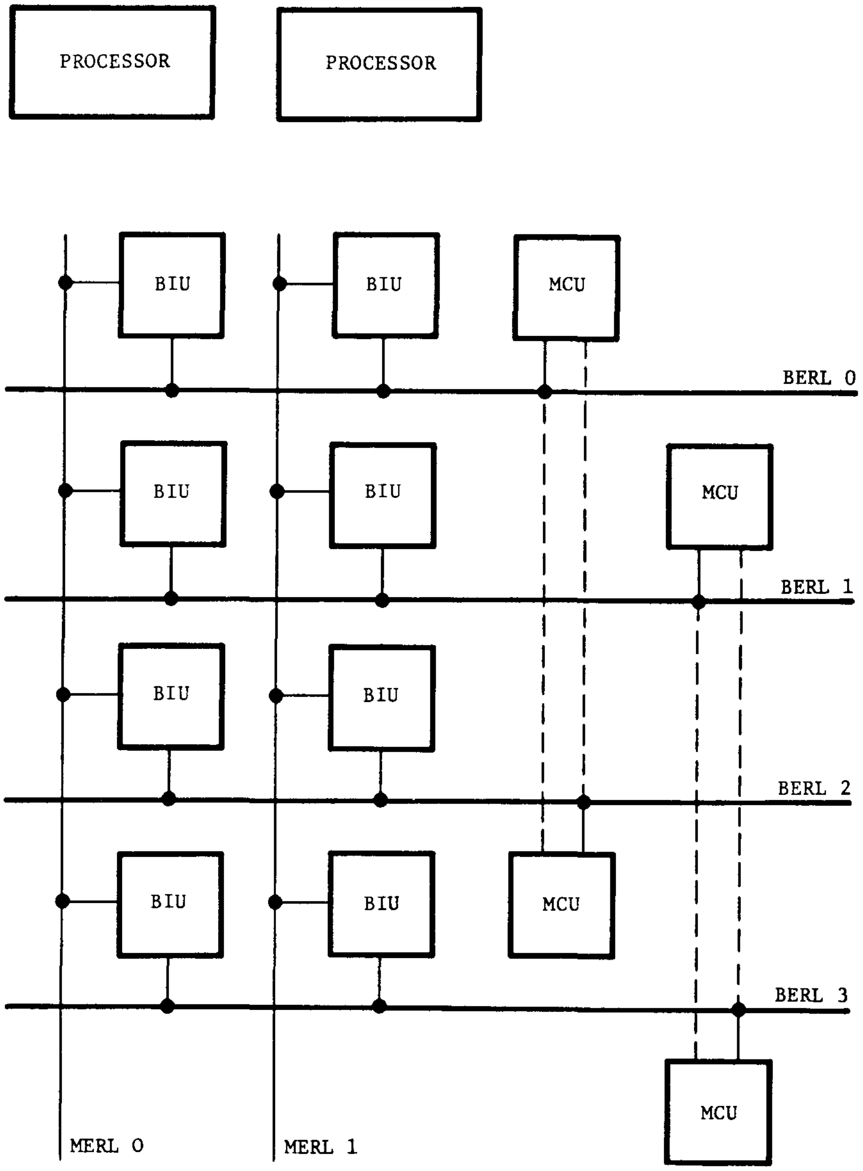


Figure 4-7. Reporting Network Topology

In summary, the error reporting network has the following important characteristics:

1. No single-point dependencies. No single failure can prevent the correct reporting of the error. This is achieved by the topology and by the redundancy provided for each error report line. The worst-case failure will destroy one MERL and one BERL. The system will correctly report and isolate the failure.

2. The error reporting topology allows graceful degradation. Each error reporting line resides in one of the confinement areas established for the system. Thus, as a system grows or degrades, the error reporting network changes with the system. The system's fault-tolerant capability will never be artificially limited by the error reporting network.

ERROR REPORTING PROTOCOL

The error reporting protocol is designed to ensure that all of the nodes in the system receive error messages in a timely manner and that if multiple error messages are broadcast, one and only one message is received by the nodes in the system. The propagation of error messages is shown in Figure 4-8.

- Phase 1. The node detecting the error sends an error report message along its BERL. The delay on this transmission is critical to guarantee that a confinement area does not operate on incorrect data. All nodes on this bus stop operation immediately upon the start of the error report message. The nodes attached to other buses need not be informed of the error immediately. The corrupt information can enter a module only by means of one of the nodes on the bus where the error was detected. Thus, fast message propagation is important only over that BERL line.

- Phase 2. After the complete error report message has been received over BERL, all the BIUs on this bus rebroadcast the message on their MERLs. The MERL message does not stop the operation of the BIUs on other buses. The MCUs do not receive any MERL messages.

- Phase 3. All BIUs in the system now know about the error. After receiving the complete error report message over MERL, all BIUs rebroadcast the message received over MERL on their respective BERL lines. All nodes on the bus terminate operation upon receiving the start of the error message. All nodes in the system now know about the error, and all the nodes on each bus have stopped operation in unison. This keeps the integrity of the normal bus protocol intact. There is no risk of a BIU retrying an operation that an MCU thought had been completed.

After receiving the complete error report message over BERL, all nodes will load this last error message into their error report logs. This completes the error reporting cycle.

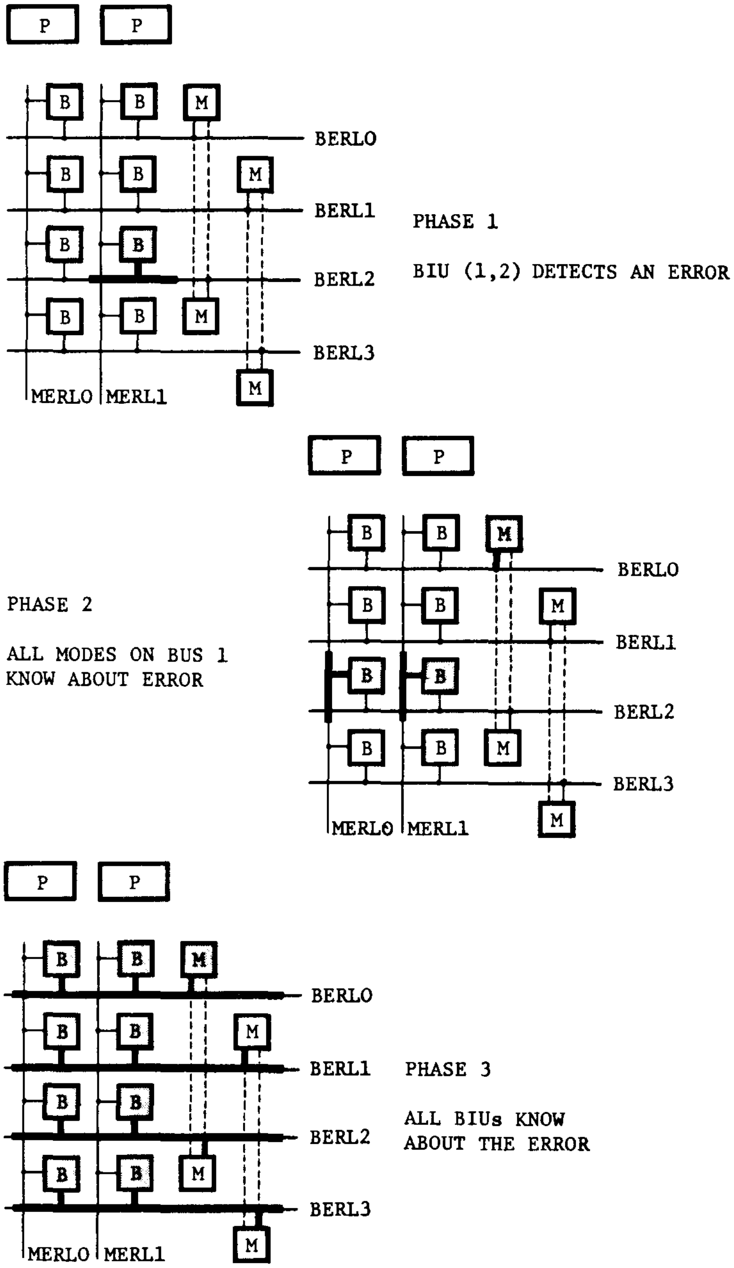


Figure 4-8. Error Report Propagation

On top of this basic transmission protocol there exists an arbitration mechanism that is used to resolve conflicts in cases where multiple nodes detect errors. There are two arbitration mechanisms. The primary one is First In, First Out (FIFO) ordering. With FIFO ordering, the first error report message has highest priority. The second mechanism handles simultaneous messages. If two error report messages are initiated at exactly the same time, then the report with the higher priority error type wins the arbitration. (If both error reports have the same error type, then the report with the larger location identifier wins the arbitration.)

These two arbitration mechanisms weed out the lower priority error reports during the reporting sequence. During phase 1, multiple error reports may exist in the system. However, on each bus, arbitration will allow only one report to complete successfully. During phase 2 any conflicts between the error reports on different buses will be resolved. At the end of phase 2, arbitration will have eliminated all but one error report throughout the system. During phase 2 an MCU may begin to generate a new error report in phase 1 (since it is unaware of the activity on MERL). The FIFO ordering arbitration will kill this report when the BIUs on this bus enter phase 3. After phase 3, every node in the system will have logged the same error message. All information about errors lost in the arbitration is also lost. If the error is permanent, it is reported again during retry. If the error is a transient, then recovery occurs as part of the retry operation for the error that has been reported.

ERROR LOGS

Each node in the system has an error report log, which holds the most recent error report message. This log is accessible by software, and it identifies the error type and location of the error. In addition to the error report logs, each MCU maintains an array error log. This log holds information specific to this memory array. The array error log records failures detected by the ECC detection mechanism. The log identifies the memory location that failed and the bit within the memory word which failed provided that the error was a single-bit error.

These logs provide the means of communication from the hardware fault handling mechanisms to the system software. They allow the software to monitor the health of the system and to properly respond to critical error conditions. The information gathered by the detection mechanisms and placed in the logs eliminates the need for diagnostic probing as a means of fault isolation.

RECOVERY

REDUNDANT RESOURCES

This section covers three topics. The optimization of redundancy to handle frequent errors efficiently, the lock step operation of a pair of shadowed modules, and graceful degradation policies.

Optimization

Published data plus design experience indicates that systems in the field will experience transient failures approximately 10 times as frequently as they will permanent failures. Because of the dominance of transient errors, special forms of redundancy (retry and ECC) are available in the system to address transient errors. This redundancy is less expensive than the redundancy required to recover from permanent failures.

In many systems, memory components account for a large portion of central system cost -- and central system failures. To provide effective redundancy for this critical section of the system, a spare bit is available in each memory array. This mechanism provides a great deal of redundancy at far less cost than shadowing the entire memory module.

Since each confinement area has its own independent set of recovery mechanisms, these may be mixed and matched to provide the optimal tradeoff between cost and fault-tolerant performance for a specific application. There is no requirement that all of the confinement areas have the same level of redundancy, e.g., full redundancy for the processor modules, retry for the buses, and ECC and a spare bit for each memory array. Such a system offers a high level of fault tolerance with reduced costs. Clearly, there are failures from which this configuration could not recover automatically (permanent MCU failure). One limitation of the mix-and-match strategy is that all processor modules should have the same level of redundancy. Because the processors operate as a cooperative pool, the processors manipulate data structures that are shared among all processors. Thus, a failure in an unprotected processor could stop processing by all processors, even if the other processors were protected by full redundancy. If the processors were split into separate pools, it might be possible to have processor modules with different levels of redundancy. However, communication between the two sets would need to be very limited and very carefully controlled to avoid any cooperative action between the two pools.

Lock Step Operation

Module redundancy is achieved by linking together two modules of the same type to form a primary/shadow module pair. This primary/shadow marriage is performed completely by software. There are no special signals to connect the two modules together. During normal operation the two modules operate in lock step as a single logical module. This provides a complete and current backup for the module and allows recovery from any single failure in either module without any interruption to the logical software environment. One module in the pair will be the active module, while the other one is passive. Initially, the primary module is the active module. The active module is responsible for driving the memory bus when this module issues a request or reply. The passive module monitors the bus and arbitration lines to track the operation of the active module. By tracking the active module, the passive module is able to maintain exactly the same state information as the active module. Data leaves only by way of the active module, but data always enters both modules.

The roles of active and passive module are switched after each bus request (reply for memory modules) issued by this module. This Ping-Pong action exercises all of the logic in both the primary and shadow modules. Any latent failure that exists in either module will be detected immediately. All of the logic to perform this lock step operation is contained in the BIU (MCU). Neither the processors nor the external TTL is aware that the module is participating as one half of a primary/shadow pair.

It is important to understand that each physical module (primary and shadow) remains a self-checking module. Whether active or passive, all detection mechanisms remain enabled and are continuously checking the operation of the module. Note that the passive module is checking itself. It is not performing any type of double check on the operation of the active module. It should also be clear that the mechanisms for providing redundancy are totally independent from the mechanisms which are used to form the confinement areas.

In the processor modules lock step operation is maintained on a clock cycle basis. On every clock edge the primary and shadow modules contain exactly the same state information. The memory modules do not maintain lock step operation on a clock cycle basis. Refresh operations and ECC correction cycles mean that the access time to a memory location is variable depending on the state of the memory module. Thus, it is not possible to keep the primary and shadow memory modules in clock cycle lock step. Instead, the memory modules operate in lock step based on bus messages. After each bus reply by the module pair, the two memory arrays are guaranteed to have the same state information before the next bus request is handled by the module pair.

The Ping-Pong action of the module pair prevents one of the modules from getting behind and dropping data. In the worst case, the passive module can be one access behind the active module. After the active module has replied to a memory request, the passive module will always complete that access, even if another error is reported in the system before the access is complete. This is required so that the memory modules maintain the same state information. Likewise, if the passive module encounters an uncorrectable ECC error after the the active module has replied to a memory request, the passive module will immediately report a permanent module error. This is required because the two arrays now hold different state information. Since the request has already been acknowledged, it will not be retried. Thus, there is no way to bring the two arrays back into lock step. This added complexity is needed to free the memory modules from the clock cycle lock step requirement.

Graceful Degradation

Because module shadowing is established solely through software operations, it allows the user a great deal of freedom in selecting graceful degradation policies. Spares may be a part of the system from the beginning, or the survivor module in a primary/shadow pair may become a spare after a failure. If performance is the absolute goal, then performance can be maintained in the presence of a failure by not deallocating the surviving module. Because of the fine-grain system partitioning, there is only a small probability that a second error will strike the surviving module. These tradeoffs are made by the system software and can be used to optimize the remaining system resources for the application needs of the system.

Recovery

This section will describe the recovery sequence at a high level. Figure 4-9 shows the recovery sequence.

The error recovery sequence begins when the nodes in the system receive an error report. This error message is logged, and the system becomes quiescent, waiting for any transient noise to subside. This delay period may be tailored to individual applications by system software. The delay, which may range from 20 microseconds to 2 seconds, is long enough to handle transients induced by mechanical stress as well as by electrical interference.

At the end of the transient waiting period, all of the accesses outstanding in the system are retried. If the same error recurs during the retry period, then the error is labeled a permanent error. Based on the location and error type information in the error message, the faulty resource is isolated from the system, and redundant resources are activated to replace the failed unit. This reconfiguration is performed in an independent and distributed manner by all the nodes in the system.

After a second transient waiting period, the accesses are retried again. When the recovery operation has been completed, the system software will be informed of the error and subsequent recovery actions. In the case of a permanent error, the BIUs send an IPC message to all of the processors in the system. This informs the software of the critical system condition immediately. For transient errors, system software simply polls the error report logs in the BIUs and MCUs. This completes the recovery sequence, and the system resumes normal operation.

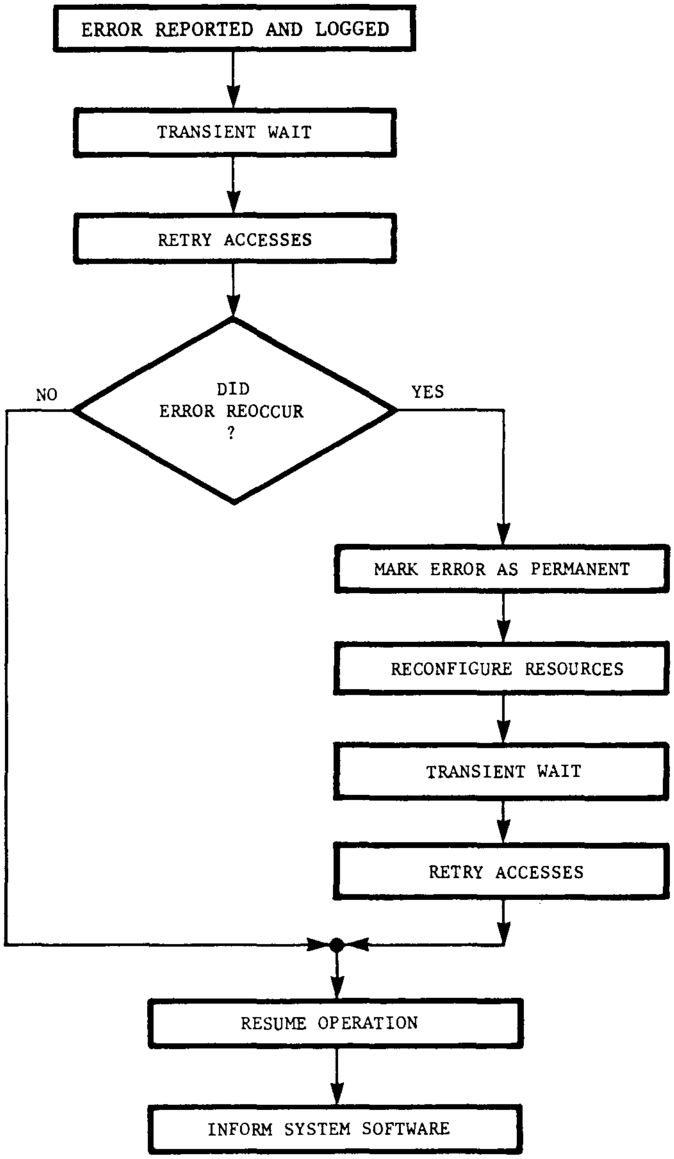
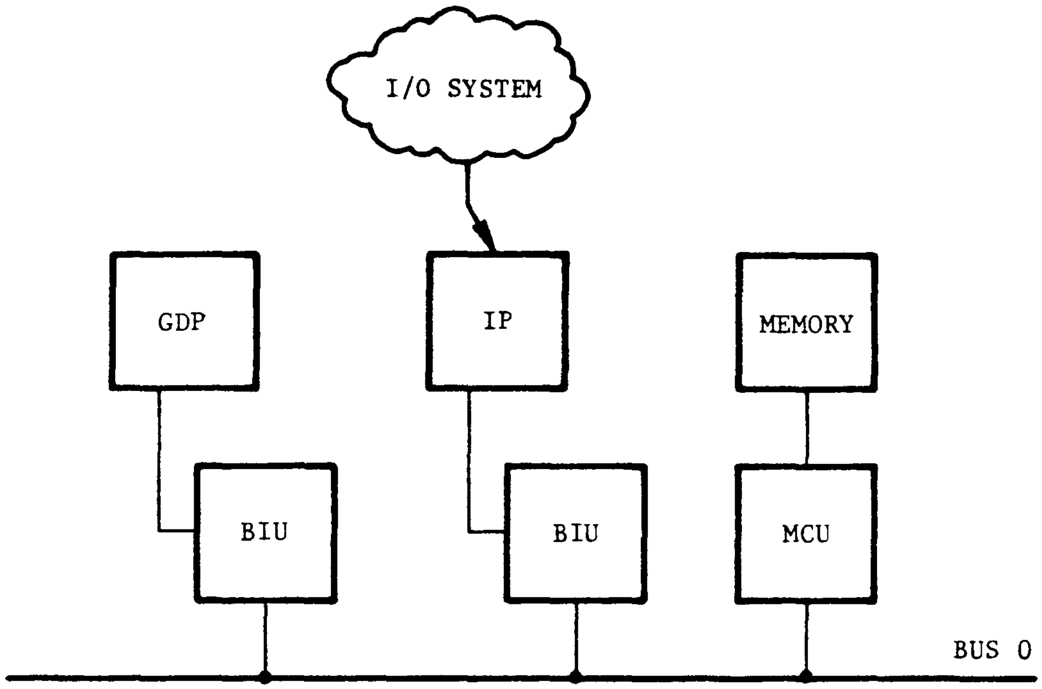


Figure 4-9. High-Level Recovery Sequence F-0437

CONFIGURATION EXAMPLES

This chapter contains a number of diagrams that provide examples of configurations over a wide range of capabilities and physical resources.

Figure 4-10 shows the most basic system that can be constructed in an interconnect-based iAPX 432 system. The only detection mechanisms are parity and retry. If any of the resources should fail with a permanent error, the system will be out of service until the module is repaired.

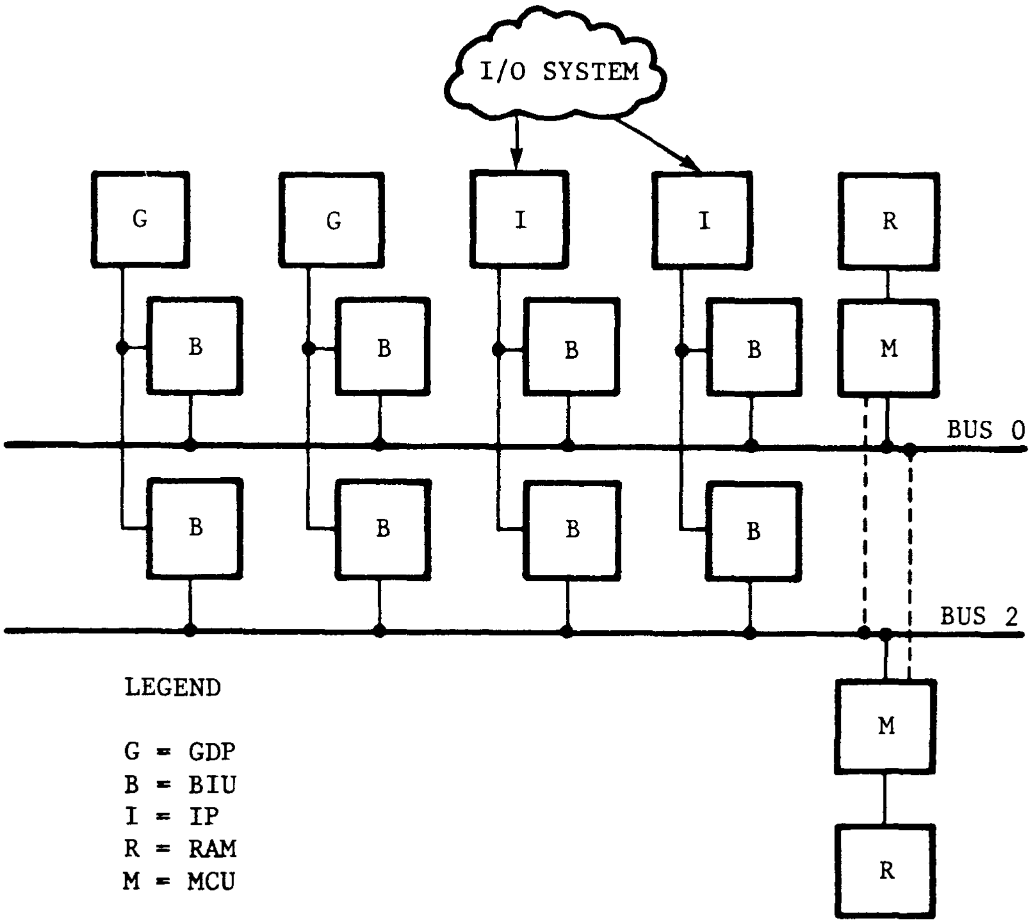


BASIC SYSTEM

LOGICAL CONFIGURATION	PHYSICAL CONFIGURATION
1 GDP	1 GDP
1 IP	1 IP
1 MEMORY ARRAY	2 BIUs
1 MEMORY BUS	1 MCU
	1 MEMORY BUS
	1 39-BIT-WIDE RAM ARRAY

Figure 4-10. Basic System F-0438

Figure 4-11 shows a more flexible iAPX 432 configuration. This system has only parity and retry, but it has two of each type of resource. If any one of these resources should fail, software could reconfigure the system. The system would be on-line as soon as diagnostics had isolated the faulty resource.

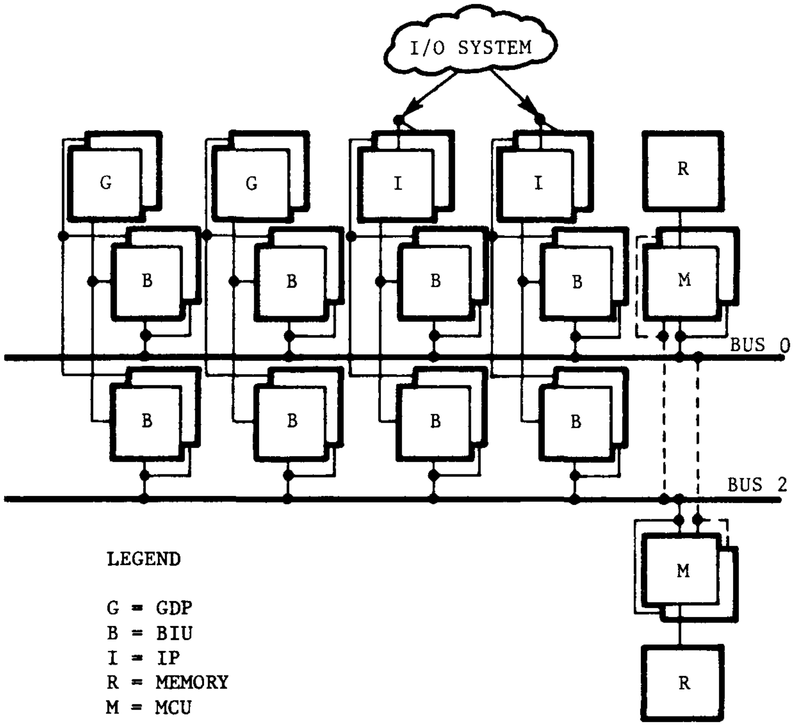


FLEXIBLE SYSTEM

LOGICAL CONFIGURATION	PHYSICAL CONFIGURATION
2 GDPs	2 GDPs
2 IPs	2 IPs
2 MEMORY ARRAYS	8 BIUs
2 MEMORY BUSES	2 39-BIT-WIDE RAM ARRAYS
	2 MCUs
	2 MEMORY BUSES

Figure 4-11. Flexible System F-0439

Figure 4-12 shows a system capable of a comprehensive deferred maintenance strategy. The system is using all of the detection mechanisms available in the BIU and MCU, and it has more than one of every resource in the system. When an error occurs, it will be detected and isolated by the hardware. Software can then rapidly reconfigure the system around the faulty module and be on-line again.

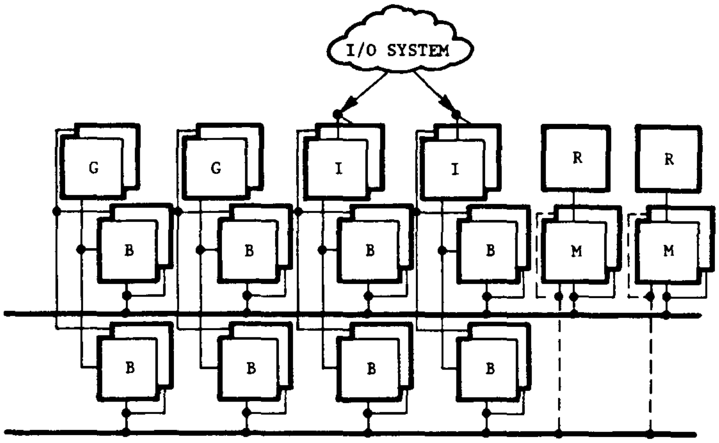


SELF-HEALING SYSTEM

LOGICAL CONFIGURATION	PHYSICAL CONFIGURATION
2 GDPs	4 GDPs
2 IPs	4 IPs
2 MEMORY ARRAYS	16 BIUs
2 MEMORY BUSES	2 40-BIT-WIDE RAM ARRAYS
	4 MCUs
	2 MEMORY BUSES

Figure 4-12. Self-Healing System F-0440

Figure 4-13 shows a small nonstop system. This system has been configured so that every resource in the system has a current and complete backup. The system can continue operation in the presence of any single failure. All of the detection and recovery mechanisms are enabled.

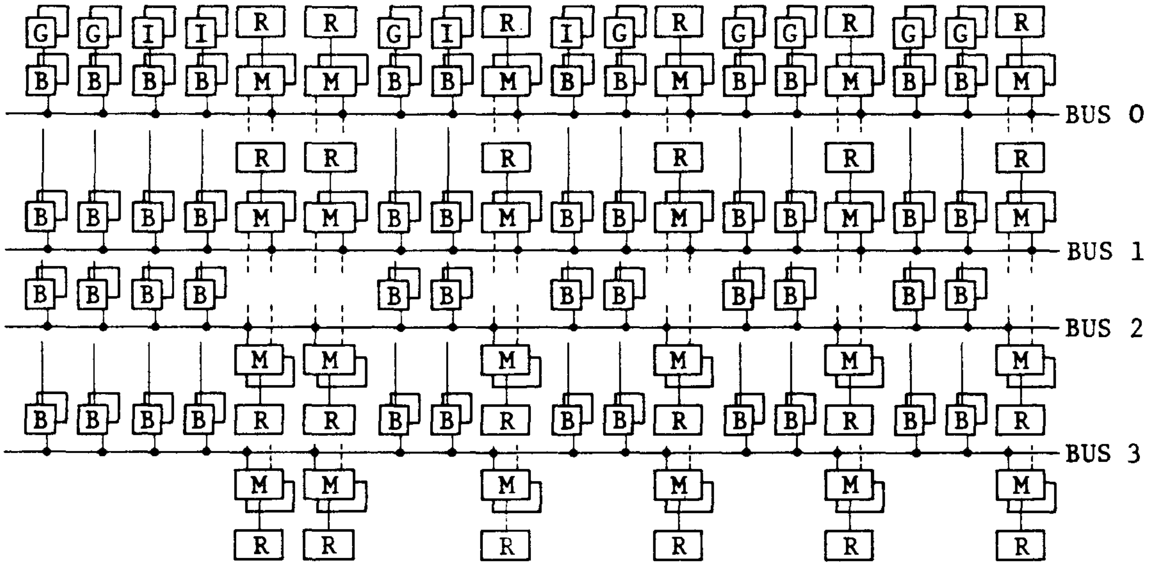


SMALL NONSTOP SYSTEM

LEGEND	LOGICAL CONFIGURATION	PHYSICAL CONFIGURATION
G = GDP	1 GDP	4 GDPs
B = BIU	1 IP	4 IPs
I = IP	1 MEMORY ARRAY	16 BIUs
R = MEMORY	1 MEMORY BUS	2 40-BIT-WIDE RAM ARRAYS
M = MCU		4 MCUs
		2 MEMORY BUSES

Figure 4-13. Small Nonstop System F-0441

Figure 4-14 shows a large nonstop system.



LARGE NONSTOP SYSTEM

LEGEND	LOGICAL CONFIGURATION	PHYSICAL CONFIGURATION
G = GDP	4 GDPs	16 GDPs
B = BIU	2 IPs	8 IPs
I = IP	12 MEMORY ARRAYS	96 BIUs
R = MEMORY	4 MEMORY BUSES	24 40-BIT-WIDE RAM ARRAYS
M = MCU		48 MCUs
		4 MEMORY BUSES

Figure 4-14. Large Nonstop System F-0442

EXAMPLE SYSTEM OPERATION

This summary provides an example of the operation sequence of a fault-tolerant machine. The example is intended to help tie together all of the mechanisms presented in the earlier portions of the document.

A DAY IN THE LIFE OF AN FT SYSTEM

1. The system is powered up, and the components receive the INIT signal. This places the components in a consistent state with each BIU and MCU that has a unique local address.

2. The initialization software sizes the system, runs confidence tests on all of the resources in the system, and reviews the past error history in the system health log. The software decides which resources are available for use in the system configuration.

3. Based on the needs of the application and the available resources in the system, software decides on the optimal system configuration. Address ranges are assigned, modules are married, and buses are enabled with the correct redundant information. (See Chapter 6.) Timeout values are loaded, and any of the optional fault-tolerant capabilities are set to the desired state. (See Chapter 5.) The memory is loaded with the information required to run the basic operating system. The system is then ready for logical operation and is in a fully fault-tolerant state. The software that performs this step is probably split between the AP and the GDPs.

4. The system is passed to the operating system, and normal operation begins.

5. During normal operation a background task that is constantly checking for latent errors in the system is run. This software uses the various commands to the BIU and MCU to verify the correct operation of the detection and recovery mechanisms. (See the "Management and Testing" sections in Chapters 5 and 6.) This software also polls the error report logs to check on any transient error conditions that may have been reported. (See Chapters 7 and 8.) Infrequently used operations in the processors may also be tested during this time.

6. An error occurs and is detected by the hardware. (See Chapter 5.)

7. An error report message is broadcast throughout the system, identifying the type of error and the location at which the error was detected. (See Chapter 7.)

8. The system becomes quiescent and waits for any transient to subside. (See Chapter 8.)

9. All accesses outstanding in the system are retried. If the error does not recur, then normal operation will resume. If the error recurs, then the recovery operation proceeds to the next stage. (See Chapter 8.)

10. The error recurs, and an error report message is broadcast throughout the system.

11. The system becomes quiescent and waits for any transient to subside. During this time each node will take whatever recovery action is appropriate, based on the type and location of the error. The faulty resource is isolated from the system, and redundant resources are activated automatically by the hardware. (See Chapter 8.)

12. All accesses outstanding in the system are retried. Each processor in the system receives a reconfiguration IPC from its BIU. This sends an interrupt to each AP and will cause each GDP to suspend its current process at the earliest possible time. The GDPs will then go to their reconfiguration dispatching port. (See Chapter 8.)

13. A process is waiting at the reconfiguration dispatching port(s). This process sends a message to higher software authorities, notifying them of the system reconfiguration. This process may do some basic housekeeping, but it will almost immediately send the processors back to their normal dispatching ports. (See Chapter 8.)

14. The system is running normally, but management software must make some decisions about the optimal configuration of the system (should resources be deallocated, should a spare be activated, etc.). Once these decisions are made, the software may alter the active configuration to put the system in its optimal state for continued operation. Normal operation may need to be suspended for a brief moment if a spare memory module is brought on-line. (See Chapter 10.)

15. The system returns to normal operation.

INTRODUCTION

This chapter describes details about the detection mechanisms used to implement the confinement areas described earlier. There are two general topics that need to be discussed before each confinement area is described. The first is the handling of the delay between detecting an error and receiving the error report message. The second topic is layout constraints imposed by the fault handling mechanisms.

There is a three-cycle delay (delay for detection mechanisms and error report propagation) from the time data is received until that data can be guaranteed correct. Thus, for three cycles, the receiving confinement area may not use this data to permanently modify state information. This only introduces additional delay when data flows back to the processor from the BIU. In all other cases this holding period is overlapped with other activity, which will not permanently modify state information.

The correct operation of the FRC detection mechanism places some constraints on the physical implementation. All signals that are FRCed must follow the layout convention shown in Figure 5-1. These layout rules are imposed so that a single break in the line will either be caught by FRC or appear as a single input failure in the external component. Any time that duplication is used as a detection mechanism, the duplicate circuitry cannot have any common components. This would introduce common mode failures, which would reduce fault coverage of the detection mechanism.

GDP CONFINEMENT AREA

Figure 5-2 shows the actual signal connections required to implement FRC for the GDP confinement area. The following pins on the master are directly connected to the matching pins on the checker: MACD0-15, CTL0-2, CHK0-1, and NREQOUT. The FRC check is performed continuously on the NREQOUT pin. FRC checking is done on the other signals only when this node is driving the memory bus (i.e., when MBOUT is asserted). An even parity bit is appended to the 16-bit messages broadcast over the MERL line. This parity bit provides error detection for single-bit and multiple-bit (odd number of) errors, plus stuck-low failures on the MERL line. Figure 5-3 shows the actual signal connections required to implement the loopback buffer check for the GDP confinement area.

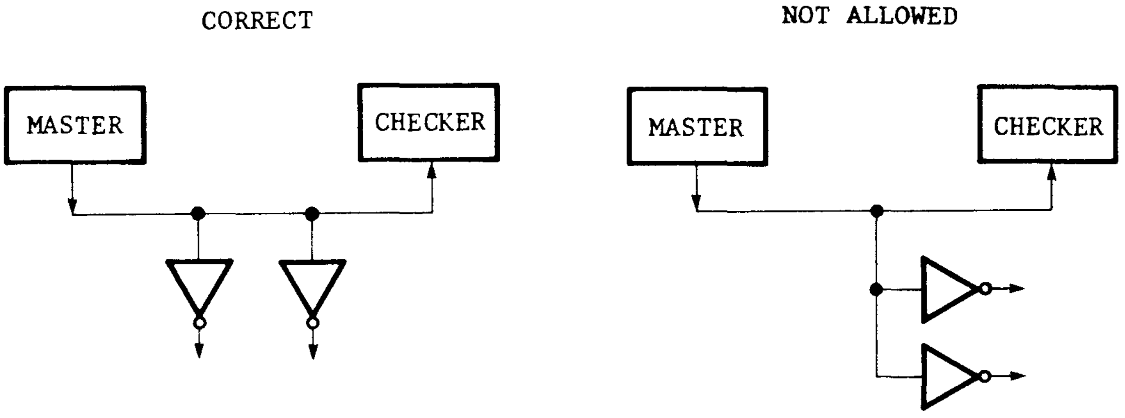


Figure 5-1. Layout Rules for FRCed Signals F-0443

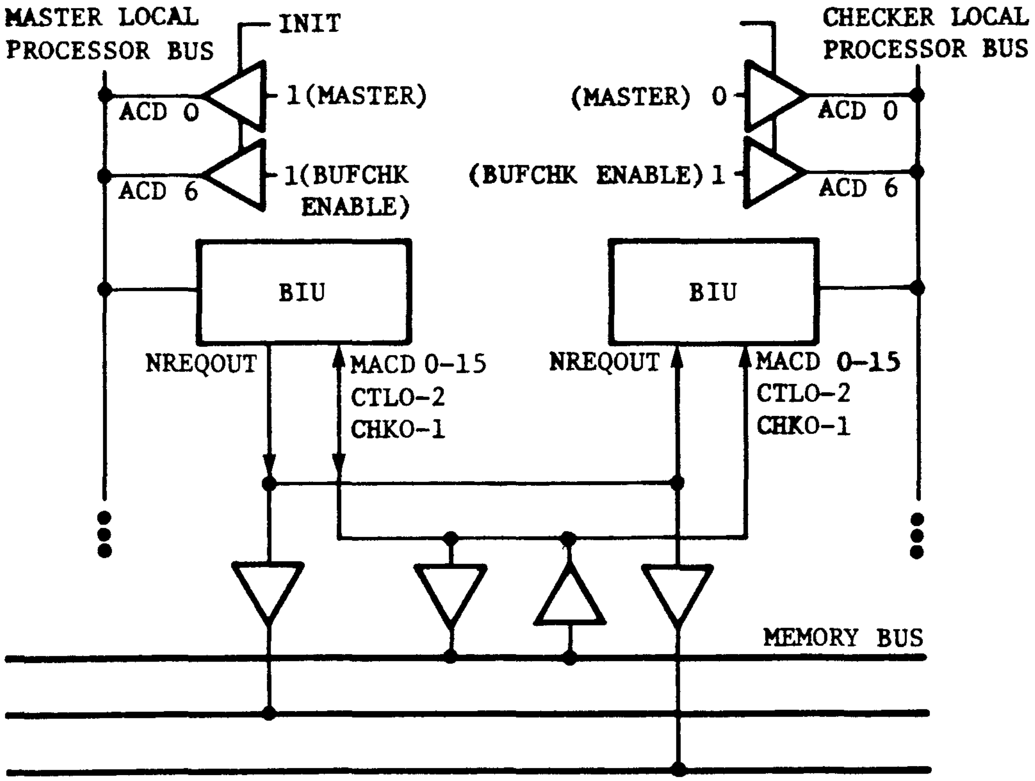


Figure 5-2. FRC Details for the GDP Confinement Area F-0444

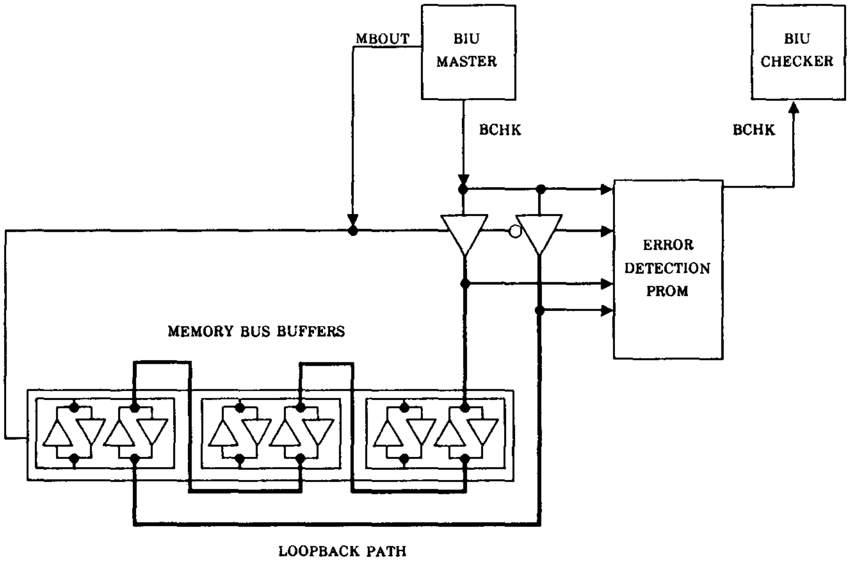


Figure 5-3. Buffer Check Detection Mechanism F-0445

MANAGEMENT AND TESTING

The FRC checking may be selectively enabled under software control using the Disable MACD Bus FRC Detection bit. At initialization, FRC checking is automatically enabled. At initialization time, bit 0 on the local processor bus specifies whether this processor module (i.e., all BIUs on this processor bus) should be a master or a checker. Software cannot convert two master GDP modules into a single self-checking module. This requires reinitializing the components.

Buffer checking may be selectively enabled at initialization time by setting the BUFCHK Enable bit in the Interconnect Device Type register. This bit is loaded in from bit 6 on the local processor bus at initialization; it cannot be modified by software.

MERL parity checking is always enabled.

Two interconnect register operations are provided to test the detection mechanisms in the GDP confinement area. First, software may toggle the master/checker function of the nodes in the system. This operation checks that the checker has not inadvertently become a master. An inadvertent master will be detected by an FRC error after the toggle operation, because now this module will have two checkers and no masters. The toggle operation is controlled by the master/checker toggle bit in the Diagnostics register. The correct assignment of master and checker functionality is also tested by the Test Error Report command, which is directed explicitly to either the master or checker. (See Chapter 7.) Lack of response from the checker indicates that the checker is not operating correctly.

The second interconnect register operation is the Test Detection command. This request may not be performed using the "my-BIU" addressing mode. The BIU responds to this request by returning data from its Test Detection Data register. When this data is driven onto the memory bus, one side of each FRC detector is inverted, forcing an error at every FRC bit comparator. (Note that correct data is driven out onto the memory bus. This test is confined to the node addressed by the Test Detection command.) Errors are also forced into the buffer check error circuitry, the MERL parity detection circuits, and the NREQOUT FRC circuit. All of these detection mechanisms must report an error. If any one of them does not report an error, then this is detected and reported as a module error. If all of the detection mechanisms are working correctly, then a NO ERROR error report will be broadcast. Software may write different data patterns into the Test Detection Data register to provide a more complete test sequence over the FRC detection circuitry. Even if a detection mechanism is disabled during normal operation, it will be checked by the Test Detection command. This approach means that the same test and exercise software may be used in systems with different detection capabilities. The test plan does not need to consider what detection mechanisms are enabled.

These tests provide very high, but not 100%, fault coverage for the detection mechanisms used in this confinement area. There are a few PLA minterms that are not tested; these represent the only hardcore circuitry in the confinement area. Table 5-1 is a summary of the management and testing capabilities available in the GDP confinement areas.

IP CONFINEMENT AREA

The IP confinement area has been shown in Figure 4-4. This confinement area is identical to the GDP confinement area except for the addition of an interface to the peripheral subsystem. Thus, the same detection mechanisms used in the GDP module provide detection coverage for the interfaces to the central system.

Table 5-1. GDP Confinement Area Management and Testing Summary

MANAGEMENT

Function	Initialization	SW Writeable
Memory Bus FRC	Enabled	Yes
Buffer Check	ACD6	No
MERL Parity	Enabled	No
Master/Checker Function (BIU)	ACD0	Toggleable

TESTING

Function	Test Strategy
Memory Bus FRC	Test Detection Command
Buffer Check	Test Detection Command
MERL Parity	Test Detection Command
Master/Checker Function (BIU)	Toggle M/C, Test Report Command

Errors are detected at the PS interface by applying FRC on the PS side of the IP. One PS interface operates as the checker while the other IP performs as the master. Any failure in the IP components, the local processor bus, the BIUs, or the TTL support and buffering components will be detected as data flows across this interface. The following pins are directly connected together on the PS side of the IP components: AD0-15, INT, INH1, BIU#, DEN#, HLD, and NAK#. The checker samples these outputs at one time during the window in which they are valid. The AP may be sampling these signals at different times, since this is an asynchronous interface. The FRC comparators on pins AD0-15 are activated only when the IP is driving data on the bus. The FRC check on the control signals is done on every clock cycle. If an error is detected, it is signaled by asserting the HERROUT signal, which causes an interrupt in the AP. The rest of the system is unaware of the failure because the error is not reported over the central system error reporting network.

MANAGEMENT AND TESTING

The detection mechanisms are managed and tested in the manner described earlier for the GDP confinement area. The FRC detection mechanisms on the PS interface may be selectively enabled at initialization. During initialization, a one on the HERROUT pin will force the PS interface into checker mode.

The detection mechanisms in the IP cannot be directly tested by software. The corruption of data to test the detection circuits would have to be done by external logic under the control of the AP. The master/checker function of an IP cannot be toggled. Table 5-2 is a summary of the management and testing capabilities available in the IP confinement areas.

MEMORY CONFINEMENT AREA

This confinement area is actually split into two parts, the RAM array and the MCU/memory bus.

Table 5-2. IP Confinement Area Mangement and Testing Summary

MANAGEMENT

Function	Initialization	SW Writeable
Memory Bus FRC	Enabled	Yes
Buffer Check	Enabled	No
MERL Parity	ACD6	No
Master/Checker Function (BIU)	ACDO	Toggleable
Master/Checker Function (IP)	HERROUT	No

TESTING

Function	Test Strategy
Memory Bus FRC	Test Detection Command
PS Bus FRC	External logic/AP SW
Buffer Check	Test Detection Command
MERL Parity	Test Detection Command
Master/Checker Function (BIU)	Toggle M/C, Test Report Command
Master/Checker Function (IP)	PS Bus FRC Test

FRC is applied to both the memory bus and the RAM array interface. On the memory bus side, the MACDO-15, CTLO-2, and CHKO-1 pins are checked by FRC. These pins are directly tied together between the master and checker. FRC checking is active whenever the MCU is driving the memory bus. On the array side, the SLADO-19, RAS#, DEIN#, WE#, and REFRESH pins are FRCed. The SLAD bus pins are checked whenever the MCU is driving them; the control pins are checked continuously. All of these pins are tied together between the master and checker. The ABCHK pin comes from the TTL logic, which derives the CAS# signal from RAS# driven out of the MCU. The ABCHK signal can be used to check that the external TTL and the lines between the TTL and the RAMs are functioning correctly. ABCHK is a feedback signal from the array control. It is the logical OR of the CAS# and WE# control signals to the array. Figure 5-4 shows the actual connections required for FRC on the array interface.

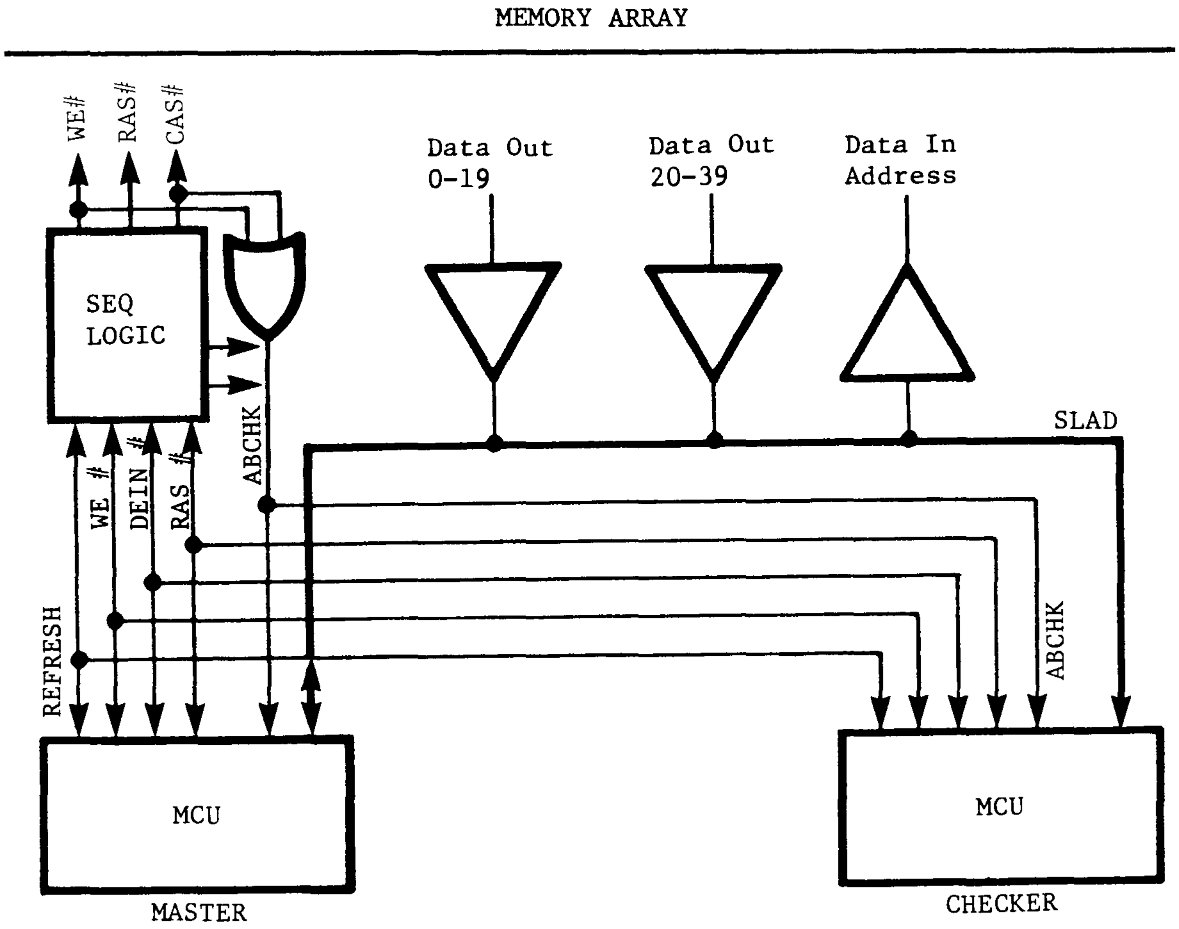


Figure 5-4. Details of SLAD FRC Detection F-0446

The MCU buffer check loopback test provides error detection for the external buffers on both buses attached to the MCU. This checks that the buffers on the currently active bus are enabled and driving in the correct direction, and that the buffers on the inactive bus are disabled. This check acts as an indirect FRC on the BUSSEL signals from the MCUs. Unlike the BIU, both master and checker MCUs are always receiving the results of the buffer check. This guarantees that an error in the bus interface will always be reported to any receiving confinement areas within the three clock window. This problem can occur if the two MCUs disagree on which bus is the active bus. The error must be reported over both buses immediately to guarantee the error report response time since it is not known which bus is actually the active bus. Figure 5-5 shows the details of the MCU buffer check connection. The MCU buffer check requires external logic because of the interface to two buses. On the active bus, the buffer check needs to ensure that all of the buffers are active. On the passive bus, the

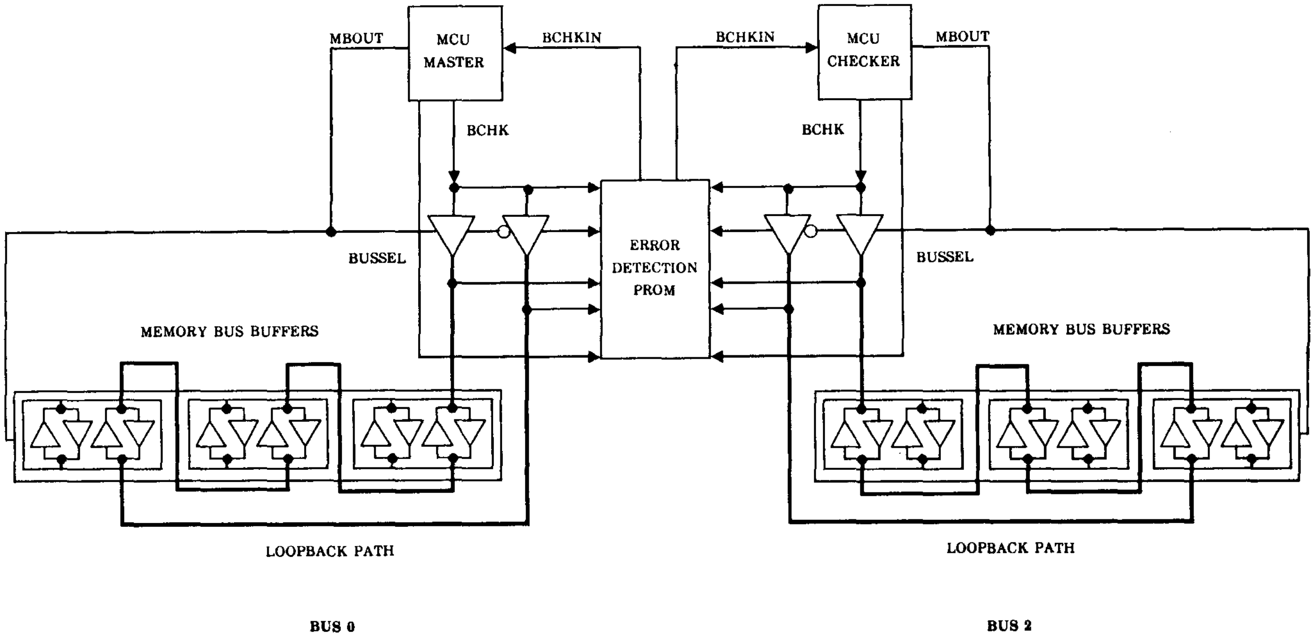


Figure 5-5. MCU Buffer Check Connection F-0447

buffer check needs to ensure that none of the buffers are active. This information can easily be combined by some logic (shown here as an error detection PROM), which also knows the current state of the buses and the buffer check signal. The delay through the external logic may force the use of a latch to meet MCU timing requirements.

Figure 5-6 shows the connection of BUSSEL to the memory bus buffers. BUSSEL is driven by both the master and the checker. These two BUSSEL signals are ANDed together externally before being used as an enable signal to the bus buffers. This reliability buttressing is done to guarantee that no single failure can corrupt both of the buses attached to the MCU. Any failure in the BUSSEL signals or the external logic will be detected by the buffer check mechanism.

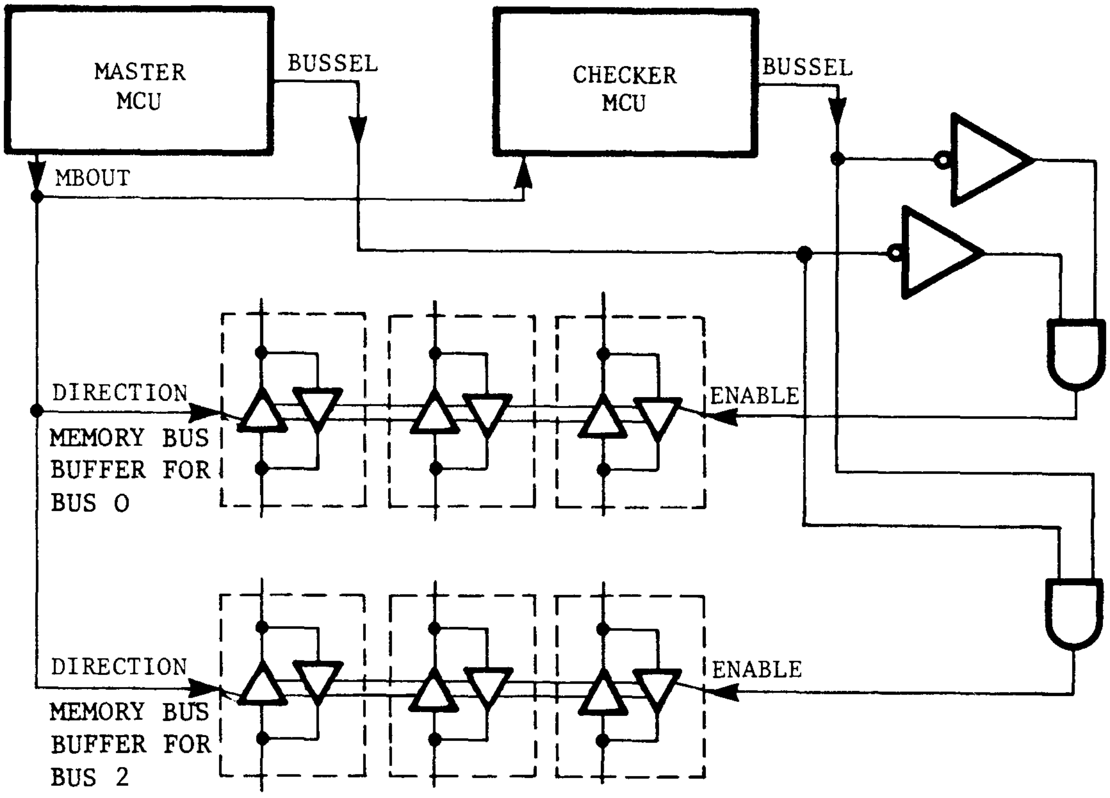


Figure 5-6. MCU Memory Bus Buffer Logic F-0448

These detection mechanisms plus the testing functions specified below provide 100% fault coverage for single failures in the MCUs, memory bus buffers, and BUSSEL logic, or in the TTL logic, which generates CAS. The TTL latches and buffers on the other SLAD signals are checked as part of the RAM array. An error detected by the buffer check mechanism or FRC on the memory bus side is reported as a module error. An error detected by FRC on the RAM array interface is reported as an unsafe module error. This error report type is required because unknown damage may have been done to the array contents. The control signals have an immediate impact on the RAM chips; thus, we cannot guarantee that an MCU failure has not corrupted data in the memory array. This report is critical because, even if the failure were transient, the corrupt data might not be detected on a later access. (For example, the master MCU converts a read operation into a write operation. The write is performed correctly by the MCU, using garbage data from its input queue. Once the checker detects the error by means of the FRC on the SLAD interface, it is too late to prevent the writing of data into the array. If this data were accessed at a later time, it would appear valid because the location would have a valid ECC code.)

The locks necessary for the correct operation of RMW sequences are held in the MCUs. There is no timeout associated with the locks. Any hardware failure will be detected by one of the other detection mechanisms. These are short-term locks used solely by the processor microcode. Software does not use these locks; it is impossible for a software error to cause the incorrect usage of the RMW locks.

MEMORY ARRAY

A Hamming code has been selected for error detection and correction within the memory storage array. Seven check bits are appended to the 4-byte storage array and are computed from the data to be stored and the address of the storage location. This encoding will ensure that:

- All single-bit errors in the data can be corrected.
- All single-bit errors in the address can be detected.
- All double-bit errors in data or the address can be detected.
- Multiple-bit (odd numbers of) errors are detected.

This detection mechanism is implemented totally within the MCU. ECC coverage is applied to refresh accesses as well as all normal accesses. On each refresh request, one location in the row is actually read and the ECC is checked. If a correctable error is found, the MCU corrects the data and writes the corrected value back to the array. This refresh operation is called scrubbing. Scrubbing is used to detect any latent faults that may exist in memory locations infrequently accessed by software. Scrubbing guarantees an access to every memory location every few seconds. This virtually eliminates the possibility of double-bit errors in the memory array.

This coverage is further amplified by performing all write operations as read-modify-write (RMW) operations. This action allows the MCU to check that it reached the correct memory address (by checking the ECC code) before it changes the data value in the array. The steps in an array write operation are listed below:

- Send address.
- Read in data and ECC checksum.
- Check ECC verifying address.
- Send write data to the array.

This detection mechanism provides coverage for all single and double failures except incorrect operation of a complete TTL buffer package (which may be detected depending on data pattern and failure mode).

Failures detected by the ECC code will be reported as either a correctable ECC error or an uncorrectable ECC error, depending on the type of error. If an uncorrectable ECC error is detected by the passive module in a primary/shadow pair, and the active module has already replied, then the error will be reported as an unsafe module error. This must be done because the original access will not be retried. If the access were a scrub access, then the error would be reported only if the ECC error log were empty. This mechanism prevents a single permanent failure from putting the system into an infinite error reporting loop.

During normal operation, memory data is returned to the BIU before the ECC check has been completed. If an error is detected, it will be reported before the data is passed from the BIU to the processor. Having the BIU hold the data decreases the memory access time by overlapping data transmission and error checking. During retry operation, the memory data is not returned until after the ECC check has been completed. This is called staged operation. When memory accesses are being staged, correctable ECC errors are corrected in the MCU, and the corrected result is passed on to the requesting BIU. No error report is generated for correctable errors detected during staging. (Reporting of scrub errors is the same as during normal operation.) This approach allows permanent single-bit RAM errors to be handled by retry.

MANAGEMENT AND TESTING

Setting up a master/checker pair of MCUs is done at initialization. During initialization, the BCHKIN/M pin becomes an input. External logic pulls the line low (to zero) if this component is to be a checker MCU, or high if the node is to be a master. Buffer checking is also selectively enabled during initialization time. The BCHK Enable bit is loaded from SLAD15 during the D1 time slot. This bit can be read by software from the Interconnect Device Type Register (bit 10). The ECC and FRC detection are enabled at initialization. Software may selectively disable these detection mechanisms by setting the correct bits in the diagnostic register (Disable MACD Bus FRC Detection, Disable SLAD Bus FRC Detection, Disable ECC Error Reporting). When ECC error reporting is disabled, the ECC log will still be updated if an ECC error occurs. Additional bits allow software to selectively enable scrubbing and to force the MCU to always run in the staged mode of operation.

Six commands are available to test the detection mechanisms in the memory confinement area. The Toggle Master/Checker bit in the diagnostic register and the Test Reporting command are used to check the operation of the checker in the way described under the GDP confinement area. The MCU also has a Test Detection command, which performs a function similar to its counter part in the BIU. Commands are always sent using the BIU/MCU register address space. This is true even when the command causes a memory operation as part of the response to the command. The Test Detection command to an MCU will cause a memory read to be done to the address held in array address-high register and the array address-low register. The FRC comparators on the SLAD bus are checked as the address is sent to the array by inverting the internal side of the comparators. (See Figure 4-2 shown previously.)

The array control line FRC circuitry is sampled twice during this time to check the comparators in both signal polarities. While the array access is being performed, the BUSSEL signal from the checker is inverted for two clocks. This checks that the buffer check external logic and MCU detection mechanisms are working correctly by forcing a disagreement between master and checker BUSSEL values. (Any information lost while these buffers are disabled will be recovered during the retry period, which always follows a Test Detection command.) The MCU will then return the low-order double byte of data from the array to the BIU. While this data is being driven onto the memory bus, the memory bus FRC comparators are checked by inverting the internal side of the comparators. If all of the FRC comparators detect errors and the buffer check detects an error, then a NO ERROR error report will be sent. If some detection mechanism has failed, then a MODULE error report will be sent. Even if a detection mechanism is disabled during normal operation, it will be checked by the Test Detection command. This approach means that the same test and exercise software may be used in systems with different detection capabilities. The test plan does not need to consider which detection mechanisms are enabled.

The ECC circuits are checked by writing bad data into the memory array. This is done by any one of three commands (Array High Access, Array Low Access, ECC Access). These commands use the high and low Array Address registers as an address and provide direct access to all the bits in the memory array (high double byte, low double byte, ECC bits and the spare bit). Software sets up the address registers and then writes any arbitrary data pattern to either the data or ECC portions of the selected memory word. The correct operation of the ECC mechanism both during scrubbing and during normal access is checked by the FRC check on the operation of the MCU. Software can also monitor the ECC log in the MCU to check that the expected error was reported by the MCUs.

These commands provide very high coverage for latent faults that may exist in the error detection mechanisms. Table 5-3 provides a summary of the management and testing capabilities in the MCU confinement area.

Table 5-3. Memory Confinement Area Management and Testing Summary

MANAGEMENT

Function	Initialization	SW Writeable
Memory Bus FRC	Enabled	Yes
Buffer Check	SLAD15/D1	No
SLAD Bus FRC	Enabled	Yes
ECC Detection	Enabled	Yes
Scrubbing	Disabled	Yes
Master/Checker Function	BCHKIN/M	Toggleable

TESTING

Function	Test Strategy
Memory Bus FRC	Test Detection Command
Buffer Check	Test Detection Command
SLAD Bus FRC	Test Detection Command
ECC Detection	Array Access/FRC/ECC log
Scrubbing	Array Access/FRC/ECC log
Master/Checker Function	Toggle M/C, Test Report Command

MEMORY BUS CONFINEMENT AREA

Figure 4-6, shown earlier, displays the memory bus confinement area. The confinement area includes the TTL buffers and the arbitration sections of the BIUs and MCUs attached to this bus. Parity is used for the detection of errors on the memory bus. Two check bits are appended to the 19 bits (16 data, 3 control) of information. One parity bit covers the even-numbered signals (10 signals and odd parity), while the other parity bit covers the odd-numbered signals (9 signals and even parity). This interlaced parity scheme is used to provide more complete coverage of bridging faults between adjacent signal lines. This detection mechanism provides the following coverage:

- All single-bit errors are detected.
- Multiple-bit (odd numbers of) errors are detected.
- Double-bit errors are detected when each parity bit covers only one of the errors (i.e., adjacent signal lines).
- Stuck-at-zeroes are detected.

It is important to realize that a failure in the parity generation or detection logic may be flagged as a parity error.

The three arbitration lines (NREQ#, RQ#, and CONT#) are duplicated. Figure 5-7 shows the detailed connections for the arbitration network. MBOUT is included in the arbitration section because it is effectively a "grant" signal from the arbitration circuits.

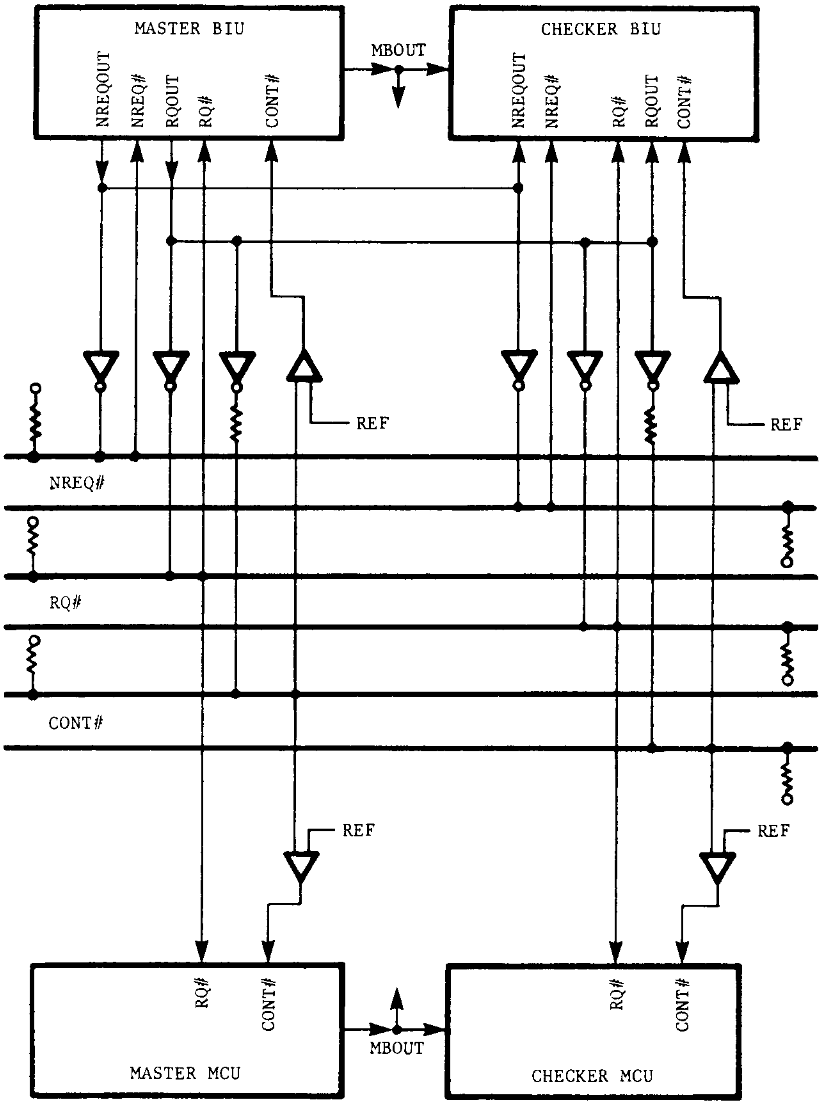


Figure 5-7. Details of Arbitration Detection Mechanism F-0449

The value of the MBOUT and RQOUT signals are dependent on the previous values of the three arbitration lines plus the state of the memory bus. The arbitration state machine in the master is using inputs that are isolated from the inputs used by the checker's state machine. Any single error in the arbitration lines or any of the arbitration state machines will be detected as an FRC error on the MBOUT or RQOUT signals. It is impossible to distinguish the difference between a failure on the arbitration lines and a failure in a state machine.

There is also an error reporting line (BERL or Bus Error Report Line) associated with the memory bus. An even parity bit is appended to the 16-bit serial message broadcast on BERL. This provides fault coverage for single-bit and multiple-bit (odd numbers of) errors, plus stuck-low failures.

These mechanisms combine to provide 100% fault coverage for single failures and very high fault coverage for common multiple failures. It is important to remember that failures that cause a whole buffer failure receive 100% coverage by the buffer check detection circuits.

Because this confinement area involves components that also reside in other confinement areas, the reporting of errors must be very carefully controlled to guarantee that the fault is correctly isolated from the rest of the system. At the interfaces between confinement areas it is possible for a single failure to be detected as multiple errors. To solve this problem, errors are given priorities based on the error type. This is the basic priority, from highest to lowest:

- Module error
- Arbitration error
- Parity error

This ordering reflects the fact that a module failure may also appear as an arbitration or parity error. However, a bus failure can never appear as a module error. Thus, whenever a failure appears as multiple errors, only the highest priority error is reported. The following paragraphs enumerate the failures that may cause conflicting errors to be reported.

If a node detects a parity error, the actual failure must reside in one of the following places:

- Memory bus lines on the backplane
- TTL buffers on another module
- Signal lines between VLSI and TTL on another module
- TTL buffers and lines on this module
- Parity detection circuits in this node
- An arbitration error, resulting in two nodes driving the bus
- Parity generation circuits at the node driving the bus

If the failure occurs in the parity generation logic, this failure is detected by FRC in that module. If the failure occurs in the signal lines between the VLSI component and the TTL buffers on another module, this will be detected by FRC or buffer checking in that module. Both errors are reported as module errors and have priority over parity errors. If the failure is an arbitration error, it will be reported as either a bus arbitration error or a bus/module-high error. These error reports have priority over parity error reports. Thus, the errors that remain have occurred either on the memory bus and its TTL buffers or in the node which detected the error. It is important to note that even if the error has occurred in this node, it does not cause any failure inside that module's confinement area. The failure is localized to the memory bus interface.

If the error is detected by a BIU, then a BUS PARITY error is reported. This report correctly isolates the error to this bus confinement area. If the error is detected by an MCU, it will be reported as a bus/module-low error. This error report has lower priority than the BUS PARITY error report. If both a BIU and an MCU detect a parity error, the BUS PARITY error report will win reporting arbitration. This correctly isolates the error to this bus confinement area because the error must be in the backplane or in the TTL buffers on another node. If only the MCU reports the error, the failure cannot be localized to a single confinement area. Both the memory module and the bus are identified as having failures. The module confinement area is also identified in this case because MCUs have contact with two memory buses. Since the failure may in fact be in the MCU's parity detection circuitry, the MCU must not be allowed to use this second bus. This is not a problem for BIUs, since they are connected only to a single bus.

If a node detects an arbitration error, the actual failure must reside in one of the following places:

- The arbitration lines on the backplane
- The external TTL logic on another module
- The signal lines between the TTL and the VLSI on another module
- The TTL or signal lines in this module
- The arbitration state machine in the master in this module

If it existed in the signal lines on another module, the failure would have been detected as an arbitration error by that module during a previous cycle. Thus, the failure is localized to this node or the signal lines and their associated support TTL logic. If the error is detected by a BIU, then the failure is localized to the bus confinement area and a BUS ARBITRATION error report is generated. If the error is detected by an MCU, then error cannot be isolated to a single confinement area. A BUS/MODULE-HIGH error report is broadcast because the failure may be in either the bus or memory confinement areas. If both a BIU and MCU detect an error, the BUS ARBITRATION error report has higher priority. The error must be in the bus confinement area, because the MCU has no outputs into arbitration network, thus a failure in the MCU cannot cause another node to see an arbitration error. The error is in the arbitration lines or the associated TTL support logic in a processor module.

If a parity error is detected on the BERL line then the actual failure must reside in one of the following places:

- The parity generator on the node generating the BERL message
- The parity detector at this node
- The signal lines between the VLSI and the TTL on another node
- The TTL buffers on another node
- The TTL or signal lines on this node

Once again the mechanisms must differentiate between failures at the MCU and other types of failures. This case is slightly different from the previous cases because error messages are not a part of normal system operation. BIUs will report this type of error as a BERL parity error, while MCUs report it as bus/module-high. The BERL parity error has higher priority. Also the MCU's have some special detection circuitry that allows them to detect if their BERL (between the VLSI and the TTL) is stuck asserted (low). The combination of these two mechanisms allows the failure to be isolated to the correct area.

A bad parity generator will be detected during testing of the error reporting network. (See Chapter 7 for details.) A bad parity detector is isolated because the MCU will label this a bus/module error. Bad signal lines will be detected by the special "stuck asserted" circuitry in the MCU. This will force the MCU to stop operation, thus resulting in an FRC error or a buffer check error. If the failure is in the TTL buffers or the signal lines, the BIU report will have higher priority and the error will be correctly isolated to the bus confinement area. If the failure is in the MCU's TTL input buffers, then the error will be detected only by the MCU and thus labeled bus/module.

The memory bus protocol includes a timeout (called the pipeline timeout). The length of the timeout is controlled by the Timeout Duration register in each node. Software controls the length of the timeout (which may range from 16 microseconds to 2 seconds).

The timeout is used during initialization. The function is to catch errors in the configuration established by the system software. The timeout is not used to detect hardware failures; any failure in the hardware will be detected by another detection mechanism. Timeouts do not cause error report messages to be generated. They simply act as pseudoreplies in each node; then normal bus operation can continue.

The node that has been waiting for the reply will send a bus error sequence to its processor. In physical mode, the IP will pass this report on to the AP as an interrupt. In all other cases, the processor (IP or GDP) will not be able to recover from this error condition. Once again, in systems with hardware error detection, this timeout will occur only because of a software failure.

All hardware failures will be detected by one of the detection mechanisms used to form the confinement areas. In non-fault-tolerant 432 systems, the timeout may also catch hardware failures and prevent the bus from hanging up.

MANAGEMENT AND TESTING

The memory bus parity detection is under software control. It is enabled at initialization and software may change the Disable MACD Bus Parity Detection bit in the diagnostic register at any time. The FRC detection in the arbitration lines is also under software control. It is enabled at initialization, and software may change the Disable MACD Bus FRC Detection bit at any time. The BERL parity detection is always enabled.

All of these detection mechanisms are checked by the Test Detection command. (This command was explained in detail in the module confinement area descriptions.) When the node being tested returns the read data onto the memory bus, the 2 parity check bits sent to the parity checker are inverted. All of the arbitration FRC comparators are fed inverted signals on their internal inputs, and a BERL parity error is forced internally. Different data patterns may be used to exercise the parity tree thoroughly. As a normal part of system operation, the memory bus parity generator is constantly checked by all of the parity checkers on the bus. Even if a detection mechanism is disabled during normal operation, it will be checked by the Test Detection command. This approach means that the same test and exercise software may be used in systems with different detection capabilities. The test plan does not need to consider which detection mechanisms are enabled.

These test commands provide very high coverage for any latent failures that may exist in the detection circuitry. Table 5-4 gives a summary of the management and testing capabilities of the memory bus confinement area.

The bus confinement area is the only confinement area that does not naturally present itself as a repair area. Because this confinement area may include multiple boards and a backplane, some diagnostic probing will be required before it is possible to identify a board that needs to be repaired.

Table 5-4. Bus Confinement Area Management and Testing Summary

MANAGEMENT

Function	Initialization	SW Writeable
Memory Bus FRC (Arbitration)	Enabled	Yes
Memory Bus Parity	Enabled	Yes
MERL Parity	Enabled	No
Master/Checker Function (BIU)	ACDO	Toggleable

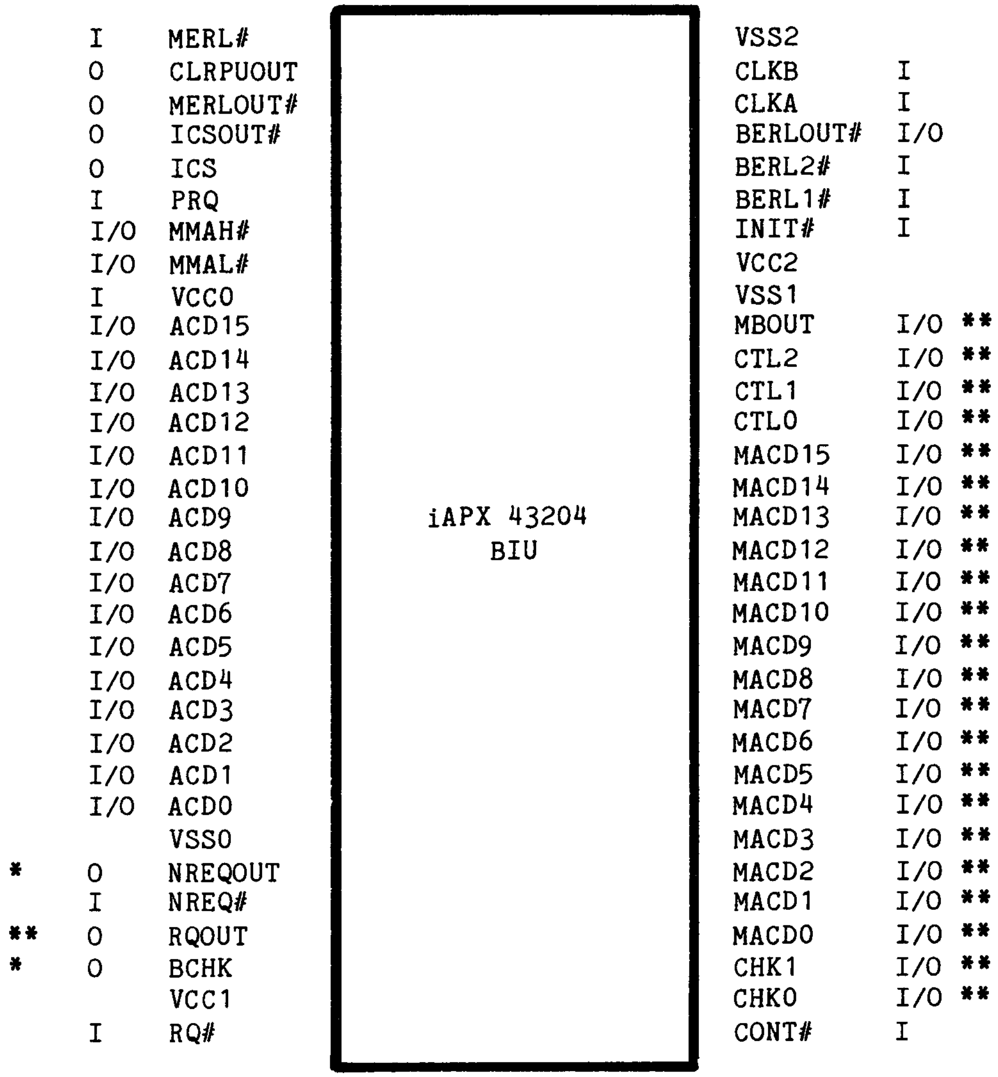
TESTING

Function	Test Strategy
Memory Bus FRC (Arbitration)	Test Detection Command
Memory Bus Parity	Test Detection Command
MERL Parity	Test Detection Command
Master/Checker Function (BIU)	Toggle M/C, Test Report Command

COMPONENT SUMMARY

Tables 5-5, 5-6, and 5-7 give a summary by component of the FRCed pins and the type of error report generated by errors detected at each pin.

Table 5-5. Summary of BIU FRC Detection



	I	MERL#	VSS2		
	O	CLRPUOUT	CLKB	I	
	O	MERLOUT#	CLKA	I	
	O	ICSOUT#	BERLOUT#	I/O	
	O	ICS	BERL2#	I	
	I	PRQ	BERL1#	I	
	I/O	MMAH#	INIT#	I	
	I/O	MMAL#	VCC2		
	I	VCCO	VSS1		
	I/O	ACD15	MBOUT	I/O	**
	I/O	ACD14	CTL2	I/O	**
	I/O	ACD13	CTL1	I/O	**
	I/O	ACD12	CTLO	I/O	**
	I/O	ACD11	MACD15	I/O	**
	I/O	ACD10	MACD14	I/O	**
	I/O	ACD9	MACD13	I/O	**
	I/O	ACD8	MACD12	I/O	**
	I/O	ACD7	MACD11	I/O	**
	I/O	ACD6	MACD10	I/O	**
	I/O	ACD5	MACD9	I/O	**
	I/O	ACD4	MACD8	I/O	**
	I/O	ACD3	MACD7	I/O	**
	I/O	ACD2	MACD6	I/O	**
	I/O	ACD1	MACD5	I/O	**
	I/O	ACDO	MACD4	I/O	**
		VSSO	MACD3	I/O	**
*	O	NREQOUT	MACD2	I/O	**
	I	NREQ#	MACD1	I/O	**
**	O	RQOUT	MACDO	I/O	**
*	O	BCHK	CHK1	I/O	**
		VCC1	CHKO	I/O	**
	I	RQ#	CONT#	I	

LEGEND

I = Input
O = Output
I/O = Input/Output
* = FRC Error causes a module error
** = FRC Error causes a bus error

Table 5-6. Summary of MCU FRC Detection

*	O	REFRESH		VSS2		
*	I	ABCHK		CLKB	I	
*	O	WE#		CLKA	I	
*	O	DEIN#		BERLOUT#	O	
*	O	RAS#		BERL2#	I	
		VCC0		BERL1#	I	
*	I/O	SLAD19		INIT#	I	
*	I/O	SLAD18		VCC2		
*	I/O	SLAD17		VSS1		
*	I/O	SLAD16		MBOUT	O	***
*	I/O	SLAD15		CTL2	I/O	**
*	I/O	SLAD14		CTL1	I/O	**
*	I/O	SLAD13		CTL0	I/O	**
*	I/O	SLAD12		MACD15	I/O	**
*	I/O	SLAD11		MACD14	I/O	**
*	I/O	SLAD10	iAPX 43205	MACD13	I/O	**
*	I/O	SLAD9	MCU	MACD12	I/O	**
*	I/O	SLAD8		MACD11	I/O	**
*	I/O	SLAD7		MACD10	I/O	**
*	I/O	SLAD6		MACD9	I/O	**
*	I/O	SLAD5		MACD8	I/O	**
*	I/O	SLAD4		MACD7	I/O	**
*	I/O	SLAD3		MACD6	I/O	**
*	I/O	SLAD2		MACD5	I/O	**
*	I/O	SLAD1		MACD4	I/O	**
*	I/O	SLAD0		MACD3	I/O	**
		VSS0		MACD2	I/O	**
	I	BCHKIN/M		MACD1	I/O	**
	O	BUSSEL		MACD0	I/O	**
	O	BCHK		CHK1	I/O	**
		VCC1		CHK0	I/O	**
	I	RQ#		CONT#	I	

LEGEND

I = Input
O = Output
I/O = Input/Output
* = FRC Error causes an unsafe module error
** = FRC Error causes a module error
*** = FRC Error causes bus/module error

Table 5-7. Summary of IP FRC Detection

*	O	INT		AD15	I/O	*
	O	ALE		AD14	I/O	*
	O	OE		AD13	I/O	*
*	O	INH1		AD12	I/O	*
		VSS		AD11	I/O	*
*	O	XACK#		AD10	I/O	*
*	O	DEN#		AD9	I/O	*
*	O	HLD		AD8	I/O	*
	I	HDA		VCC		
	I	SYNC		AD7	I/O	*
*	O	NAK#		AD6	I/O	*
	O	BOUT		AD5	I/O	*
	I	ICS		AD4	I/O	*
	O	PRQ		AD3	I/O	*
		VCC		AD2	I/O	*
	I/O	ACD15	iAPX 43203	AD1	I/O	*
	I/O	ACD14	IP	AD0	I/O	**
	I/O	ACD13		VSS		
	I/O	ACD12		PSR	O	*
	I/O	ACD11		BHEN#	I	
	I/O	ACD10		WR#	I	
	I/O	ACD9		CS#	I	
	I/O	ACD8		ALARM#	I	
	I/O	VSS		CLR#	I	
	I/O	ACD7		HERR/M#(PS)		
	I/O	ACD6		FATAL#/M#(ACD)		
	I/O	ACD5		PCLK#	I	
	I/O	ACD4		INIT#	I	
	I/O	ACD3		VCC		
	I/O	ACD2		CLKA	I	
	I/O	ACD1		CLKB	I	
	I/O	ACD0		VSS		

LEGEND

I = Input
O = Output
I/O = Input/Output
* = FRC Error causes a HERR signal

OVERVIEW

The previous chapter described the confinement areas established in the iAPX 432 central system. Each module and bus in the system can now be viewed as a self-checking module. When an error occurs, it will be detected and isolated to a single confinement area (sometimes two confinement areas, one module and one bus). Once the error has been isolated to a confinement area, recovery may occur if there are redundant resources in the system. This section deals with establishing redundant resources that can be used to recover from hardware failures without any impact on logical software operation.

There is an important distinction between redundant resources and alternate resources. Redundant resources provide a complete and current backup for the resources in the system. When a failure occurs, the backup can mask the fault from the rest of the system. This recovery occurs automatically under the control of the hardware. Alternate resources provide multiple modules capable of performing the same tasks. However, there is no complete and current backup for the resources in the system. Alternate resources allow reconfiguration around failures, but do not mask the failure from the rest of the system. Alternate resources can be used to achieve deferred maintenance capabilities. Some examples are given in the "Configuration Examples" section of Chapter 4.

The mechanisms and resources used for redundancy are totally orthogonal to the mechanisms and resources used for detection. When configuring a system, the designer may make totally independent decisions about which detection mechanisms and which redundant resources to include in the system. It is important to understand that adding redundancy for recovery does not degrade or improve any of the detection capabilities or performance characteristics of the machine.

Each of the self-checking modules and buses can be paired with an identical back-up module or bus. This feature is known as shadowing, allows continued operation in the presence of any single failure. A physical constraint is that there cannot be any common components shared between a module or bus and its back-up resource. Any common components would introduce common mode failures from which the system could not recover. Modules can be paired with any other identical module. No predetermined electrical connections are required; shadowing is done by means of logical configuration information. The two modules in the pair then operate in lock step, providing a complete and current backup of all state information in the case of a failure. Module shadowing for all three module types will be described in the following sections.

Buses can also be paired together to provide redundant bus channels. However, since buses hold no state information, during normal operation, both buses in a pair may be used to carry traffic. Thus, the back-up bus capability carries no penalty of decreased bus bandwidth. The mechanisms for establishing and maintaining redundant operation reside totally in the BIU and MCU. The GDP and IP are unaware that they are operating in a redundant configuration.

REDUNDANCY FOR TRANSIENT ERRORS

RETRY BUFFERS

When a processor makes a request, all of the information associated with the request (address, control, and data) is stored in retry buffers in the BIUs attached to the processor. This information is held until the access is finished. If a transient error occurs during this access, the BIUs will retry the entire access. This allows recovery from transient errors on the memory bus and in the memory module. The chapter on isolation and recovery explains the retry sequence in detail.

This buffering is always enabled. It adds no delay to the access latency for a processor request. The correct operation of the buffer is tested by software issuing a Test Report command (interconnect register write -- data may have any value) to any node in the system. The command will cause a retry of all accesses in the system. If any buffer has failed, it will be detected by FRC in the module.

MEMORY ARRAY -- CORRECTION BY WAY OF ECC CODE

The 7-bit Hamming code described in the section on memory module confinement areas also provides correction for single-bit failures in the memory array. This redundant information can be used to correct permanent as well as transient array failures. Because of the high soft error rates of dynamic RAM components, this redundancy is intended to handle only transient errors. Other forms of redundancy are available to handle permanent errors in the memory array. The management and testing of the ECC mechanism is described in the previous chapter on detection mechanisms.

GDP MODULE REDUNDANCY

This section will describe the three phases of shadow operation (startup, lock step operation, and deactivation) for the GDP modules, plus the management and testing of the mechanism.

STARTUP

Any two GDP modules may be paired together (married) to form a shadowed module. This is possible because all GDP modules have identical interfaces. Startup is entirely a software operation, which is performed by a processor other than the two about to be married. This start-up procedure is the same whether done at initialization or on-line during normal system operation. The basic function of the start-up sequence is to:

1. Have the primary/shadow processors use the same processor object so that every reference is identical.

2. Synchronize and coordinate the primary/shadow processors at the microinstruction level to guarantee that memory references occur on the same clock cycle.

3. Make each physical processor module aware of its partners, so that, in the event of a failure, the correct recovery operations will take place. (Only the BIUs, not the GDPs and IPs, are aware of their partners.)

The start-up sequence is listed below.

1. The two processors involved in the marriage are stopped in a stable state. If both processors are running, then stoppage can be accomplished by sending the local IPC stop to both processors. If a module has failed or is in an unknown state, then a different procedure must be used. First the IPC register in the BIUs must be cleared. Then a Clear PU command is issued to each processor module. These two steps put the processors in a known and stable state.

2. IPCs to the two processors are disabled either by locking up both global and local IPCs or by software convention (such as assuring that nothing will generate an IPC).

3. The Processor ID in the primary-elect BIUs is changed to the selected value for the primary/shadow pair. This change is made by writing to the Processor ID register in all of the BIUs attached to this processor.

4. The Processor ID in the shadow-elect BIUs is changed to the selected value for the primary/shadow pair. Note that these two modules now have the same processor ID, giving both modules the same processor object and allowing them to respond in unison to IPCs. IPCs are addressed by using the processor ID as an address.

5. The logical module ID of the primary-elect BIUs is set to the selected value for the primary/shadow pair. This is done by writing to the Logical ID register in all of the BIUs attached to this processor.

6. The logical module ID in the shadow-elect BIUs is set to the selected value for the primary/shadow pair. (Note that these two modules will now be using the same ID -- the logical module ID -- when they arbitrate for the bus. This will allow the two modules to track each other's operation over the

memory bus. All logical IDs in the system must be unique. Normally the logical ID will also be unique from all physical IDs, but this is not required. The logical ID may be the same as either of the physical IDs of the two modules being married. If the logical ID is the same as one of the physical IDs, then that module must be the first one of the pair to have its married bit set.)

7. The BIUs in the shadow-elect module are marked as shadows by setting the shadow bit in the State register in all the BIUs in this module. This operation is accomplished by reading the State register, setting the shadow bit in this memory image of the State register, then writing back the new value into the State register.

8. The physical module ID (from the Interconnect Device ID register) of the shadow-elect module is loaded into the Spouse ID register of all the BIUs in the primary-elect module.

9. The physical module ID (from the Interconnect Device ID register) of the primary-elect module is loaded into the Spouse ID register of all the BIUs in the shadow-elect module.

10. The BIUs in the primary-elect module are marked as married by setting the married bit in the State register in all the BIUs in this module. This operation is accomplished by reading the State register, setting the married bit in this memory image of the State register, then writing back the new value into the State register. (Note that this module will now act like the primary module in a married pair. It will use the logical ID to recognize interconnect register accesses, and it will be the active module until after the first bus request by this module.)

11. The BIUs in the shadow-elect module are marked as married by setting the married bit in the State register in all the BIUs in this module. (Note that this module will now act like the shadow module in a married pair. It will use the logical ID to recognize interconnect register accesses, and it will remain passive until after the first bus request by the primary.) The two modules are now locked together and operating as a primary/shadow pair, blended into a single logical unit. There is <u>no</u> independent access to the physical modules.

12. A Clear PU command (an interconnect register write operation with any value in the data field) is sent to the married pair. This command is sent to only one BIU, not every BIU in the module. (Note that this means one BIU in the primary and one BIU in the shadow.) It may be addressed to any one of the BIUs in the module. This command causes the BIU to assert the CLRPUOUT signal, which is connected to the processor's CLEAR pin. This forces the processor to a known state and synchronizes the primary and shadow processor modules. (Note

that BIUs are synchronized by letting them both reach the idle state. However, the processors are in a two-instruction microcode loop when idle. Thus, they must be explicitly synchronized.) The number of processors in the system (used for broadcast IPC) is reduced by one, and IPCs are enabled.

13. A local IPC is sent to this processor ID. This resets the processor pair.

14. A local start-up IPC is sent to this processor ID. This causes the processor pair to read in their processor ID from the BIUs and then requalify all processor information. (Note that this requalification is done by a single logical processor, not two independent processors.) The processor will then respond to whatever command has been placed in the local communication segment (Stop, Dispatch, etc.).

LOCK STEP OPERATION

During normal operation the two modules operate in lock step as a single logical module. This provides a complete and current backup for the module and allows recovery from any single failure in either module without interruption to the logical software environment. One module in the pair will be the active module, while the other one is passive. Initially, the primary module is the active module. The active module is responsible for driving the memory bus when this module issues a request or reply. The passive module monitors the bus and arbitration lines to track the operation of the active module. By tracking the active module, the passive module is able to maintain exactly the same state information as the active module. Data leaves only by way of the active module, but data always enters both modules.

The roles of active and passive modules are switched after each bus request issued by this module. This Ping-Pong action exercises all of the logic in both the primary and shadow modules. Any latent failure that exists in either module will be detected immediately. All of the logic to perform this lock step operation is contained in the BIU. Neither the processors nor the external TTL is aware that the module is participating as half of a primary/shadow pair. It is important to understand that each physical module (primary and shadow) remains a self-checking module. Whether active or passive, all detection mechanisms remain enabled and are continuously checking the operation of the module. Note that the passive module is checking itself. It is not performing any type of double check on the operation of the active module.

DEACTIVATION

Deactivation is not recovery. The recovery mechanisms are described in a later chapter. Deactivation allows a primary/shadow pair of modules to be stopped and logically split apart during normal operation. The steps of deactivation follow.

1. The processor is stopped in a stable state by the "stop" local IPC.

2. IPCs to the processor are disabled either by locking up both global and local IPCs or by software convention.

3. The married bit (State register) in the BIUs is reset. Note that this bit will be reset simultaneously in both the primary and shadow modules. After it is reset, both modules will become active, each capable of responding to requests on its own. The interconnect registers in the two modules may now be addressed individually by software using the physical module ID. However, the two modules are still using the same logical ID for arbitration.

4. Reset the shadow bit (State register) in the BIUs in the ex-shadow module.

5. Assign the BIUs in the ex-shadow module a new logical module ID for use in arbitration. (Note that the primary could also have its logical ID changed.)

6. Assign a new Processor ID in the ex-shadow module BIUs. (Note that the primary could also have its processor ID changed.)

7. The number of processors in the system (used for global IPC) is increased by one, and IPCs are enabled.

8. A local start up IPC is sent to both processor IDs. The start-up IPC causes the processor ID to be read in from the BIUs. Afterwards, all processor information is requalified.

MANAGEMENT AND TESTING

The redundancy mechanism is managed by way of the start-up and deactivation procedures described in this section. These procedures allow this redundancy to be invoked or terminated at any time during the life of the system.

The mechanisms are self-testing. The correct operation is verified simply by using the mechanism. Any failure will be caught by the detection mechanisms in the module. Remember that both the primary and shadow modules are self-checking modules using master/checker GDP pairs.

IP MODULE REDUNDANCY

On the central system side, IP shadowing is done identically to GDP module shadowing. On the PS side, there is no VLSI support for shadowing. The external support logic must provide the functionality to achieve shadowing on the PS interface. Even if the PS interface does not offer total redundancy into the I/O system, IP module shadowing is important.

IPs, like GDPs, perform complex operations over system objects in the central system. Module shadowing guarantees that no object will be left in an inconsistent state because of a failure in the IP module. Without module shadowing, it is possible for an IP failure to cause a system crash when an object is left in an inconsistent state.

STARTUP AND DEACTIVATION

The external logic can use signals from the AP plus the CLEAR PU signal from the BIUs to trigger functions for startup and deactivation. The IP is unaware of primary/shadow or active/passive designation, and thus cannot be used to help control the external logic. The external logic could be as simple as connecting the primary and shadow IPs to a single bus that attaches to a single AP or as complex as extending shadowing on into the AP subsystem itself. Startup and deactivation require a coordinated effort between software in the peripheral subsystem and software in the central system.

LOCK STEP OPERATION

The IPs must be kept in lock step for the correct operation of the mechanisms on the BIUs. The two IPs cannot have independent asynchronous interfaces to the PS because the two IPs could react to asynchronous signals over different time intervals, thus losing lock step operation. To achieve lock step operation, the synchronization must be done in a cooperative manner that guarantees that both the primary and the shadow module will respond just as they do to the signals from the PS.

MANAGEMENT AND TESTING

On the central system side, the management and testing are carried out in the same manner as for GDP modules. On the PS side, these issues are dependent on the implementation of the external logic and software. The PS interface will probably have substantially less management and testing flexibility than the central system interface.

MEMORY MODULE REDUNDANCY

Within memory modules there are two forms of redundancy. A spare bit in the memory array is used to provide redundancy for any single failure in the RAM components. Module shadowing is available to provide redundancy for any single failure anywhere in the memory module.

SPARE BIT

The memory array that interfaces to the MCU logically contains 39 bits of information (32 data bits plus 7 ECC bits). The physical interface to the MCU can actually handle 40 bits of information. This last bit is the spare bit. (The array is ordered with the spare bit as the low-order bit, followed by the data word, then the ECC bits as the high-order bits.) The Spare Bit Select register in the MCU controls the use of the spare bit. This register is managed entirely by software; there is no automatic handling of this register by the hardware. The Spare Bit Select register identifies the bit to be replaced by the spare bit. If the register has a value other than 0 to 38, then the spare bit is not in use.

The operation of the spare bit is checked by switching it into the array. When the switching is done, the MCU should report a burst of single-bit errors in the bit location just covered by the spare bit. These errors occur as scrubbing moves through the array and updates the spare bit to match the value that should exist in that bit location. If a double-bit error is encountered, the word can be fixed by switching out the spare bit and correcting the single-bit error that remains. The first burst of errors indicates that the switch was done correctly. Software then clears the ECC error log register (Array Error Log-Low and Array Error Log-High). After a delay, the log is checked. If the spare RAM component is working correctly, there will be no errors in the log. If there is a failure in the MCU which causes incorrect operation, this will be detected by FRC. If the spare RAM chip or signal lines fail to work properly, this will be detected by ECC. If the memory module is shadowed, this test may be performed on-line, because the shadow module can take over if a double-bit error is encountered. If the module is not shadowed, then the system can be either stopped for a few seconds or exposed to the very small probability of encountering a double-bit error.

MEMORY MODULE SHADOWING

Establishing a back-up module for a memory module is done in a way very similar to one by which the processor modules are married. The following sections will specify the steps taken during startup, lock step operation, and deactivation.

The start-up sequence is listed below.

1. All accesses to the two memory modules involved in the marriage are stopped. This must be done by the system software: it has nothing to do with the physical memory modules. This quiescent state can be reached in two basic ways. The system could be stopped while the two modules are linked together. (The procedure will take approximately one second to complete.) Or the system software could use its memory management routines to make this portion of memory unaccessible. There are two problems with the second approach. First, certain objects (known as frozen memory) must always be accessible. Second, memory interleaving means that a very large block of memory (1 or 2 Megabytes) would have to be deallocated. These two problems make stopping the system the preferred approach.

2. The two memory arrays must be made to hold identical contents. This can be done in two ways. If the memories do not hold any data, then both arrays can be cleared using the Clear Memory command. This will take approximately 40 milliseconds if the two arrays contain 256K bytes of 64K RAMs. The second approach is to copy the data from one array into the other. This will take approximately 800 milliseconds for 256K byte arrays.

3. The MCUs in the shadow-elect modules are marked as shadows by setting the shadow bit in the State register in each MCU. This operation is accomplished by reading the State register, setting the shadow bit in this memory image of the State register, and then writing back the new value into the State register.

4. The memory address range for the primary-elect module is set to the selected value for the primary/shadow pair by writing to the Memory Start and End register.

5. The memory address range for the shadow-elect module is set to the selected value for the primary/shadow pair by writing to the Memory Start and End register.

6. The MCUs in the primary-elect module are marked as married by setting the married bit in the State register. This operation is accomplished by reading the State register, setting the married bit in this memory image of the State register, then writing back the new value into the State register. (Note that this module will now act like the primary module in a married pair. It will be the active module until after the first memory reply by this module.)

7. The MCUs in the shadow-elect module are marked as married by setting the married bit in the State register. (Note that this module will now act like the shadow module in a married pair. It will remain passive until after the first memory reply by the primary.)

The two modules are now locked together and operating as a primary/shadow pair. These two physical modules operate as a shadow pair only for memory array accesses. They retain distinct interconnect register address ranges and may be accessed independently.

Unlike the processor modules, the primary and shadow memory modules do not operate in cycle-by-cycle lock step. Instead, the modules are synchronized at the level of messages on the memory bus. Both modules recognize bus requests addressed to them and carry out the required action on the memory array. The active module is responsible for generating the reply message. (Remember, each MCU will always reply to interconnect register accesses and commands independent of its active/passive status.) The active/passive roles are switched after each reply to a memory access request.

The lock step constraint is loosened for memory modules so that the two arrays may be managed independently. This is needed to handle situations like ECC data correction or refresh, which may not occur simultaneously in both modules. Loosening of the lock step operation is possible because memory modules never generate bus request messages. Cycle-by-cycle lock step operation is required to track the operation of the arbitration network for request messages because the arbitration lines carry only a partial decoding of the arbitration ID. The arbitration for replies is fully decoded (the address in the request message is combined with FIFO ordering of replies) and thus does not require cycle-by-cycle lock step operation.

Because the two modules are not operating in lock step, it is possible for the passive module to fall behind the active module. Because the active/passive role is reversed after every reply, a module can never be more than one access behind the operation of its partner. (The reply to the second access is made by the slow module, thus the system waits until it has caught up before continuing.) To accommodate this variance, the MCU's input buffer can hold four requests even though the bus pipeline is limited to three outstanding requests. If the passive module is behind (i.e., the active module has sent its reply message, but the passive module has not completed its array access) when an error is reported, it will complete the final access required to catch up to the active module before stopping. The passive module is responsible for monitoring the bus and tracking the state of the active module. The passive module will always remain in step with the active module at a memory bus level. The passive module takes special steps to guarantee that the RMW lock and the array contents are always consistent with the state of the active module. These steps ensure that the two arrays always have identical information content.

It is important to understand that each physical module (primary and shadow) remains a self-checking module. Whether active or passive, all detection mechanisms remain enabled and are continuously checking the operation of the module. Note that the passive module is checking itself. It is not performing any type of double check on the operation of the active module. Any latent failure in either module will be detected immediately.

Memory module shadowing is deactivated in the following way:

1. All accesses to the memory modules are stopped. This must be done by the system software; it has nothing to do with the physical memory modules.

2. The married bit (State register) in the MCUs is reset. Note that this bit must be reset separately for both the primary and shadow modules. After the bit is reset, both modules will become active.

3. Reset the shadow bit (State register) in the MCUs in the ex-shadow module.

4. The two modules can now be assigned independent address ranges and begin operation as independent units.

MEMORY BUS REDUNDANCY

All the memory buses in the system are physically identical. However, when a system is operational, each bus handles a unique address range. The memory buses have been designed so that it is possible to pair together two memory buses and have them act as redundant or alternate resources for each other.

Because the bus does not hold state information, both buses may be used to transmit data during normal operation. There is no degradation of system throughput to achieve memory bus redundancy. The following paragraphs describe the memory bus, BIU, and MCU characteristics that allow this redundancy.

BUS CHARACTERISTICS

- The address field sent on a memory bus is unique on a system-wide basis. When the BIU does address manipulation for interleaving, it does not strip out any of the bits; it only reorders them. This action allows address spaces from multiple buses to be combined onto one bus without redefining any of the address recognition logic.

BIU CHARACTERISTICS

- The BIU is capable of recognizing processor requests for two address ranges. These two address ranges are called the primary address range and the back-up address range. Either or both sets of these address ranges may be enabled at any time. The constraints and limitations on them is described in later paragraphs in this section.

- Buses are reconfigured as part of the retry sequence (which is described in the Chapter 8). During this sequence, the BIUs are able to recompute address recognition for any processor request that will be retried.

- BIUs in a processor module record the status of partially completed MMA accesses (Multiple Module Access - accesses which span two memory buses). This allows partially completed MMAs to be retried correctly. Two signals are used to transmit this information. MMAH# tracks the high-order half of the MMA access; MMAL# tracks the low-order half of the MMA access. These signals are asserted (low) at the beginning of the access and released when that portion of the MMA has been completed. If an error report that invalidates the access is received then the MMA line will be asserted for one cycle. This action informs the other BIUs in the module that the access has been cancelled. The MMAH# and MMAL# signals are monitored by all the BIUs in the module. Internal state information on the completion status of the operation is kept so that, in the case of a bus reconfiguration, the access can be retried in the proper way.

MCU CHARACTERISTICS

- Each MCU has a physical connection to two buses. At any instant only one of these connections may be active. The MCU signal BUSSEL is used to select the currently active bus. (Chapter 5 explained attaching the MCU to two buses.)

- Bus reconfiguration may occur during the retry sequence. MCUs are capable of switching their physical bus connections as part of the retry sequence.

- The locks associated with RMW operations are held in the MCUs. Thus, when the MCUs switch buses as part of bus reconfiguration, all of the state information regarding RMW operations automatically moves to the new bus. (The lock is held with the data that has been locked.)

The hardware imposes two constraints that limit how buses may be combined to form redundant bus pairs. Hardware requires that a bus request have a single destination. Thus, a single memory request may not go to two MCUs. This imposes restrictions on the redundancy because the recovery actions will place MCUs onto the same bus that used to be on separate buses. Moreover, BIUs have only a single address recognition register (Memory Start and End register). The address space for a bus can be expanded only during recovery or only by changing the interleaving type from four-way to two-way. Buses are always paired in the same way (0 and 2, 1 and 3, 4 and 6, 5 and 7). No other pairings are possible. The options available for bus redundancy are listed in Table 6-1.

Table 6-1. Memory Bus Redundancy Options

Number of Buses	Interleaving	Redundant Configuration
1	One-way	Not Possible
2	One-way	One bus plus a spare
2	Two-way	Deferred maintenance only
3	---	Not possible
4	One-way	Two buses plus two spares
4	Two-way (64 bytes)	Two buses plus two spares
4	Four-way	Four buses each with a backup
6 or 8	---	Combinations of two- and four-bus configurations

Interleaving, which must be present, prevents the hardware recovery actions from violating these constraints. However, in some deferred maintenance environments it is desirable to enforce the single request-single destination rule by software rather than hardware. Such a case arises when a two-bus system is converted into a one-bus system. For the hardware to guarantee correct operation, a single bus must have only one MCU. In a deferred maintenance environment, all the MCUs on both buses could be moved to one bus. In this situation, software must guarantee that a single access does not address multiple memory arrays. The recommended approach is to turn off all interleaving, and guarantee correct operation by careful allocation of memory. The memory management routines must guarantee that no free block of memory contains locations from more than one memory array. Since interleaving is turned off, this breaks the memory space into physical chunks that will most likely be 256K to 512K bytes long.

With two buses in the system, the redundant configuration requires that one of the buses be a spare. Only one of the two buses can be active at any time. With four buses it is now possible to have bus redundancy without sacrificing any bus bandwidth. Buses 0 and 2 are paired, and buses 1 and 3 are paired. Thus, the paired buses do not have adjacent address spaces. (Any MMA access will address one module in each pair of buses.) A BIU can now cover the expanded address range by switching from four-way interleaving to two-way interleaving.

Two-way interleaving needs some further explanation. Two-way interleaving may be done on 64-byte boundaries or 128-byte boundaries. If the interleaving is done on 64-byte boundaries, buses 0 and 1 are interleaved. This allows construction of nonstop systems based on two-way interleaving (two active buses with two spare buses). The more useful configuration utilizes interleaving on 128-byte boundaries.

This interleaving is done across buses 0 and 2, allowing the construction of two bus systems capable of deferred maintenance. This is possible because the MCUs may be moved between the two buses by software and because either of the buses may act as the IPC bus. Nonstop operation is not possible in this configuration with two-way interleaving.

MANAGEMENT AND TESTING

There are three sets of bits that control memory bus redundancy: the redundant bus enable bit, bus state bits, and interleaving control bits.

- Redundant Bus Enable bit. This bit resides in the Interconnect Device Type register. Its value is determined at initialization time. This bit must be set if the system is to be able to recover from failures that may occur before the software initialization sequence has been completed. This bit must also be set if the system wishes to have bus redundancy without four-way interleaving during normal operation.

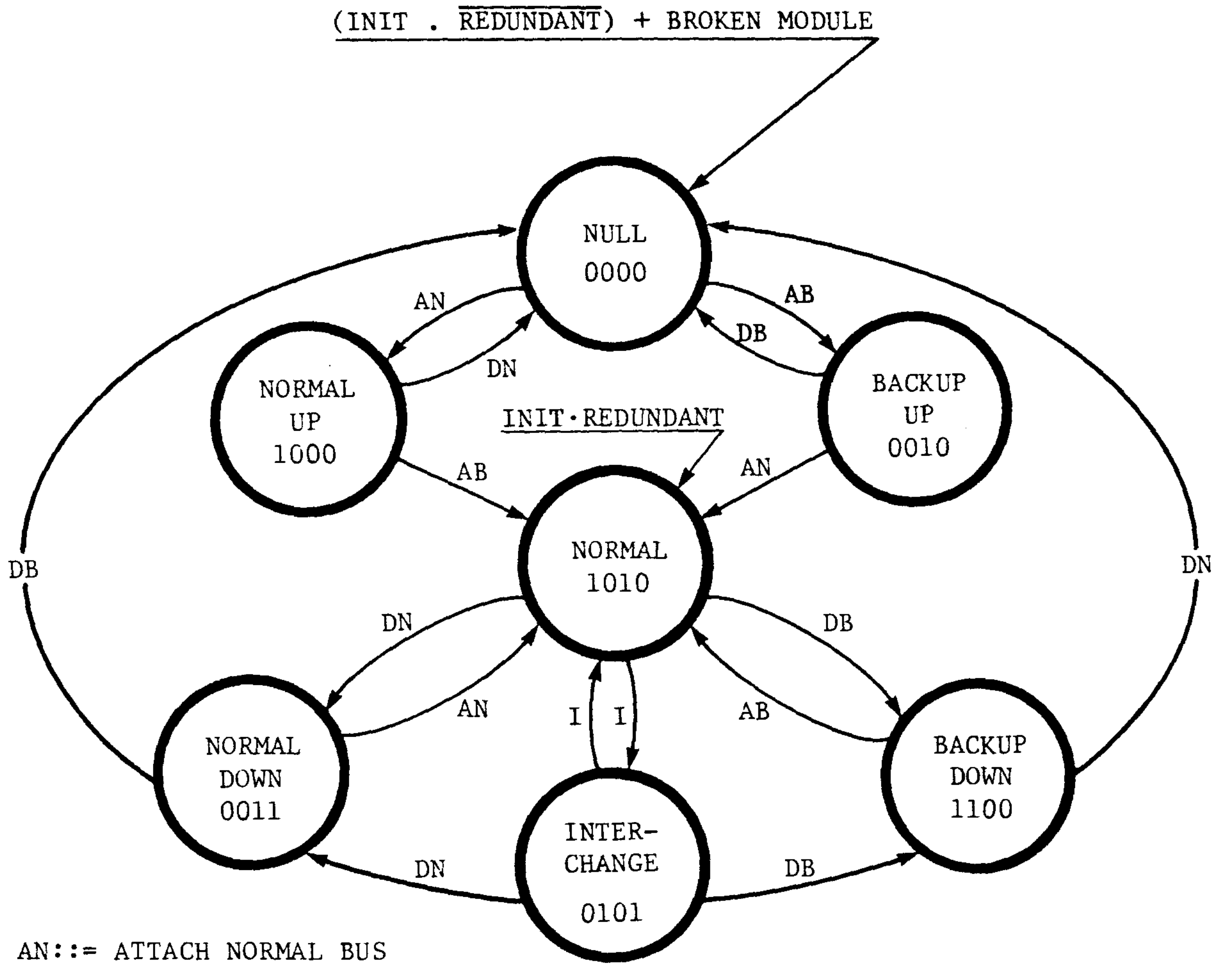
- Bus State bits. These 4 bits represent the state variables for the state machine that controls bus recovery. These bits describe the state for a pair of buses. Buses are always paired in the same way (0 and 2, 1 and 3, 4 and 6, 5 and 7). No other pairings are possible. Table 6-2 describes each of the states. Bus State bits are directly writeable, but only for diagnostic or initialization purposes. These bits should never be changed by direct register access during normal system operation. Special commands (described below) direct the hardware to change these bits in a manner that is guaranteed to work during normal operation. Figures 6-1 and 6-2 show the state diagrams for the bus recovery state machine.

- Interleaving Control bits. In the MCU there is one control bit, Four-Way Bus Interleave Mode, which is located in the State register. This bit is set if the BIUs are doing four-way interleaving. It is used only for bus recovery and has no impact on address recognition. In the BIU the Interleave register holds the Interleaving Control bits. The 3 low-order bits determine the type of bus interleaving being performed by the BIU (00: one-way, 01: two-way on 64-byte boundaries, 10: two-way on 128-byte boundaries, 11: four-way on 64-byte boundaries). These bits are used for address recognition as well as for bus recovery. The third bit in this set is used to contol interleaving in the memory modules and has no impact on bus switching.

Table 6-2. Description of the Bus States

Enable my primary address range.
Enable my back-up address range.
My back-up bus's primary address range is enabled.
My primary bus's backup address range is enabled.

A	B	C	D	State	State Description
0	0	0	0	NULL	Neither of the buses in this pair is active. This is the initialization state if the Redundant Bus Enable is off. In this state no memory accesses can be performed.
1	0	0	0	PRIMARY UP	The bus to which this node is attached is active. The second bus in the pair is not active. This is the normal state for half of the buses in a system that has bus redundancy, but is not four-way interleaved.
0	0	1	0	BACKUP UP	The other bus in this bus pair is not active, and the bus to which this node is attached is not active. This is the normal state for half the buses in a system that has bus redundancy but is not four-way interleaved.
1	0	1	0	NORMAL	Both buses in this pair are active. This is the normal mode of operation for a pair of buses in a four-way interleaved system. This is the initialization state if the Redundant Bus Enable is on. In this state both buses can perform all types of bus accesses.
0	1	0	1	INTERCHANGE	Both buses in this pair are active. However, they are both using their back-up address ranges. This is used to test that the bus recovery mechanisms work correctly. There is no impact on the system operation. Thus, this is identical to the normal state except that the address ranges covered by the two buses have been interchanged.
0	0	1	1	PRIMARY DOWN	This bus is not working, but the back-up bus has taken over the address range previously handled by the bus. This is the state of all the nodes on a bus after the bus has had a permanent failure.
1	1	0	0	BACKUP DOWN	The other bus in this bus pair has failed. This bus is handling the address ranges normally covered by both buses. This is the state of all the nodes on a bus after its partner bus has failed.



AN::= ATTACH NORMAL BUS
AB::= ATTACH BACKUP BUS
DN::= DETACH NORMAL BUS
DB::= DETACH BACKUP BUS
I:: = INTERCHANGE COMMAND

Figure 6-1. State Diagram for Four-Way Interleaved Bus Recovery F-0450

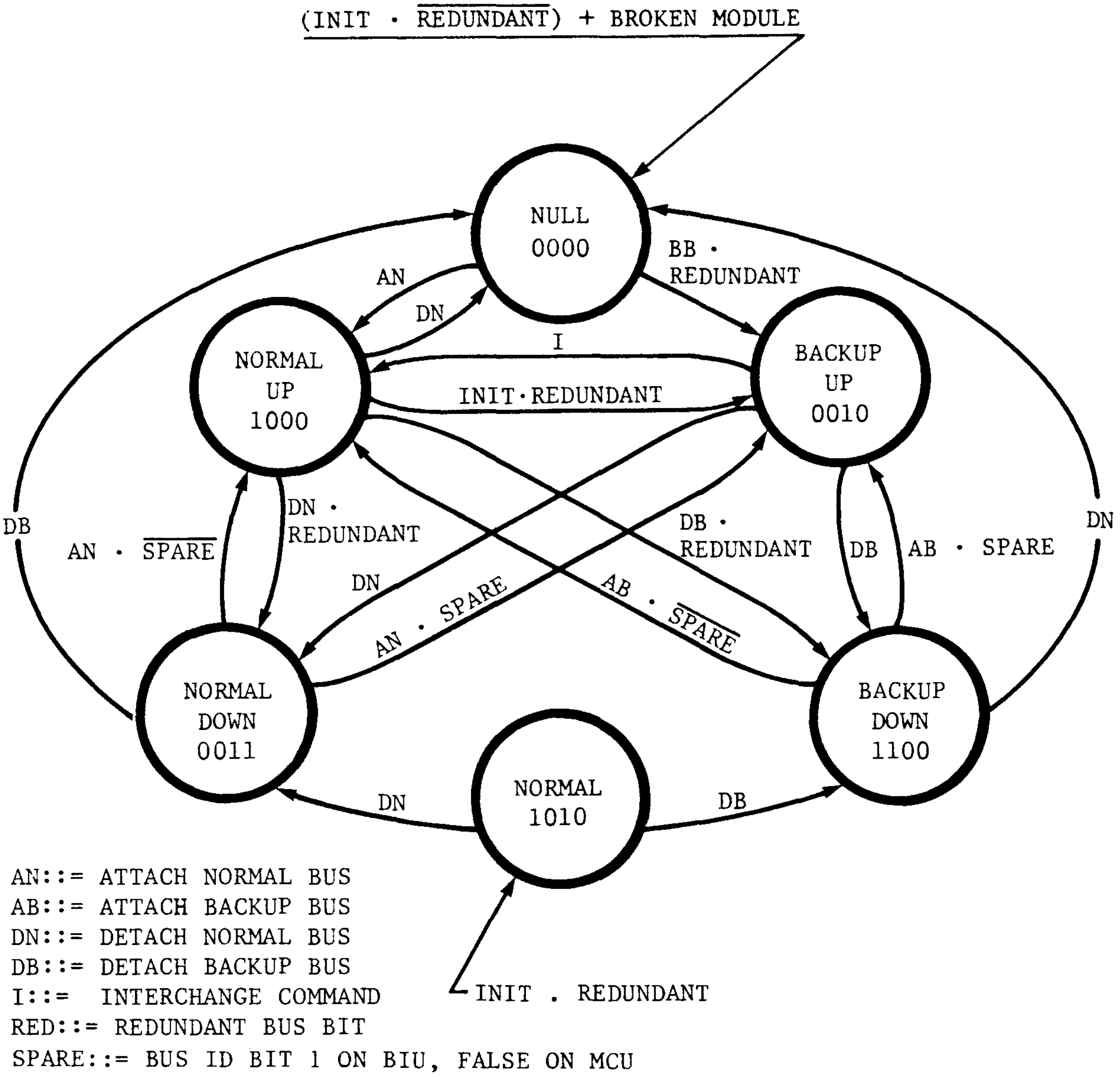


Figure 6-2. State Diagram for Two-Way Interleaved Bus RecoveryF-0451

Four special commands -- Toggle, Attach, Detach, and Interchange -- are provided to set up and test the memory bus redundancy mechanisms. These commands may be sent to any node in the system for the purpose of changing the bus configuration. As part of these commands, an error report message will be generated, and the retry procedure will be performed. By going through this sequence, the bus configuration may be changed without any danger of dropping an access that may have been in progress.

The operation of these commands is described below.

1. Toggle. This command is used as part of the software initialization sequence to assign memory modules to buses for normal operation. This command toggles the value of the middle bit in the physical bus ID (Interconnect Device ID register) of the MCU that received the command. (This command may be sent only to MCUs.) At hardware initialization, the MCU is assigned a bus ID. This bus may not be appropriate once software understands the total system resources available. Toggle is then used to distribute memory modules between the two buses of a bus pair. The command does not have any effect on the bus state machine.

2. Attach. This command is used to activate a bus and may be sent to any node. The bus that will be activated is the back-up bus for the bus connected to the node receiving the command. (If the command is sent to a node on bus 2, then bus 0 will be attached.) The command must be sent to the back-up bus because the deactivated bus has been isolated from the system and thus cannot propagate any error report messages (including ATTACH). See Figures 6-1 and 6-2 for the exact impact of this command.

3. Detach. This command is used to deactivate a bus and may be sent to any node. The bus that will be deactivated is the the bus connected to the node receiving the command. Detach allows software to perform exactly the same recovery sequence that the hardware automatically takes if it detects a permanent bus error.

4. Interchange. This command is used to check whether the redundant bus and recovery mechanisms are working correctly. It causes all buses in the system to reverse roles with their back-up buses. If a bus does not have a back-up bus, then Interchange has no effect on that bus. This command allows testing of all of the logic and signal paths associated with both buses. Invoking Interchange in no way reduces the system's ability to respond to error conditions.

In a four-way interleaved system, only the toggle command would be used during software initialization. The bus state machine is initialized to the correct state by the hardware. If four-way interleaving is not used, then software must perform the following sequence (in addition to the toggle sequence) for each bus pair in the system.

1. Detach the spare bus in the pair.

2. Attach the spare bus in the pair.

These two steps cycle the state machine to the correct state for normal operation. At this time the software may assign memory address ranges to the BIUs and MCUs.

OVERVIEW

Error reporting is the backbone of fault isolation and recovery. When an error is detected, the node detecting the error reports the type and location of the error to all the other nodes in the system. (Note that errors detected at an IP:PS interface are not reported over the error reporting network. These errors are reported to the AP by way of an interrupt.) The error reporting system is designed so that, independent of the error in the system, each node not only receives an error report but is guaranteed to receive the same error report. This error information is uniformly logged in all of the nodes in the system. With this information, each node then independently proceeds with the appropriate recovery procedure.

TOPOLOGY OF THE REPORTING NETWORK

The reporting network follows the same matrix topology as the normal data paths. There is an error reporting line associated with each memory bus and processor bus in the system. (Figure 4-7 provides an overview of the topology.) Each BIU is connected to one BERL and one MERL. Each MCU has connections to two BERLs, but at any instant only one of the connections is active. The selection between the two BERLs uses BUSSEL, the same signal that provides memory bus selection for the MCU. Because the error reporting system is fundamental to the correct operation of a fault-tolerant system, error detection and masking mechanisms are built in to each error report line. The detection mechanisms were described in Chapter 5; the redundancy and recovery mechanisms are described in the following paragraphs.

Redundancy is provided to guarantee that, independent of a failure, the error report network will always function correctly. Each BERL line has a back-up line, and every logical BERL (one per memory bus) is actually implemented as a pair of physical signals. Thus, any single failure in the lines will be masked by the operation of the redundant line. Figure 7-1 shows the connection of the two BERLs, which form a single logical BERL line. Because of pin limitations, each component has only a single BERL output. If this pin fails "stuck-off," that node will be unable to report any of the errors it detects. This latent fault can be uncovered for using the Test Report command. Any other single failure, including "stuck-on" failures in the BERL outputs, will not prevent the reporting of any error in the system (including an error in this BERL). The BERL input circuitry is responsible for correctly interpreting the error message in the presence of failures in the BERL line. If one line has an error (detected by parity or incorrect start bit sequence), it will be ignored and the message received from the other line will be used. If

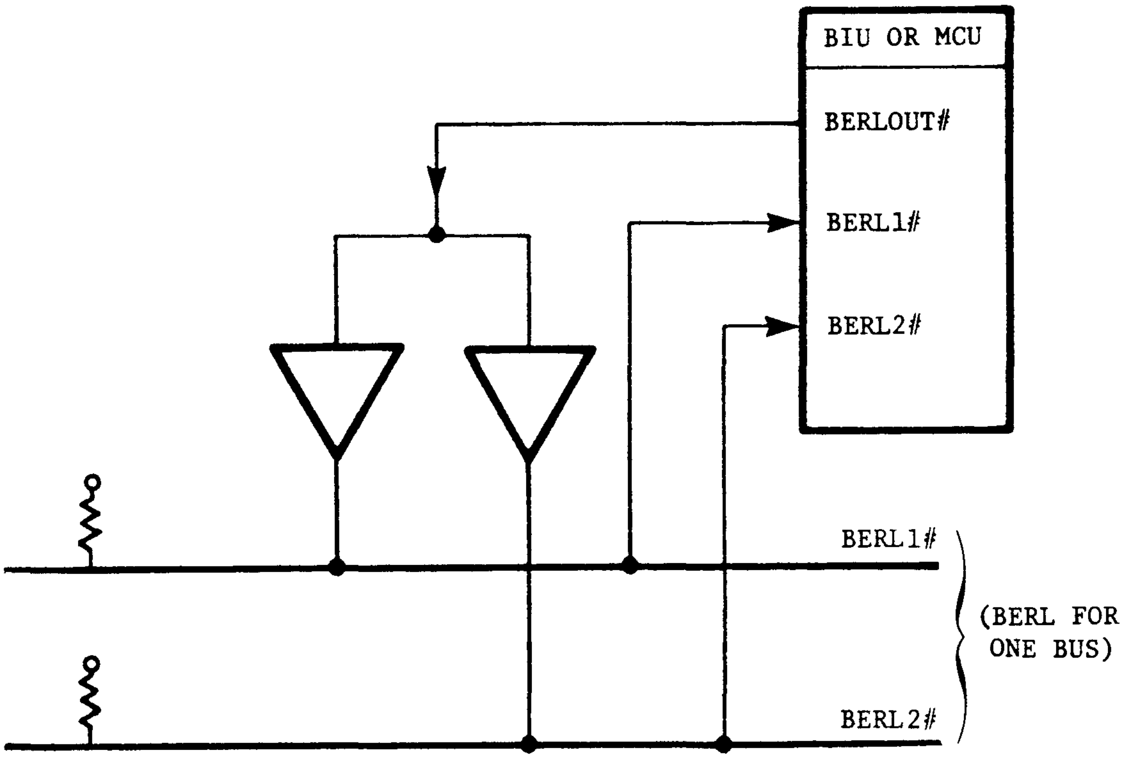


Figure 7-1. BERL Redundancy F-0452

both lines have errors, this is automatically converted into an error message related to BERL failures. By having two inputs and two signal lines, it can be guaranteed that a node will never be isolated from the error reporting system (i.e., not see an error report broadcast by another node).

The redundancy for MERL lines is achieved implicitly, rather than by adding explicit redundancy. All that is required for correct error reporting in the system is one correctly operating MERL. We are guaranteed to have multiple MERL lines in any functional system because each processor module must have one MERL for the master half and one MERL for its checker half. Thus, a single failure cannot corrupt both MERL lines. At least one of the MERL lines provides correct propagation of the error report message. In systems that do not have self-checking modules, the system is still guaranteed to have a minimum of two MERL lines because every functional system must have two processors (one GDP and one IP).

Figures 7-2 and 7-3 show the exact connection of each node to the appropriate error reporting lines. The MCU connection to the two BERL lines is critical. Both the master and the checker must be able to report an error even if they have a disagreement on BUSSEL. (Otherwise BUSSEL errors could not be reported.) Yet, some type of reliability buttressing is required to prevent a failure in one of the MCUs from corrupting the operation of both of the memory buses to which it is connected. The circuit shown in Figure 7-3 provides this capability. As long as there are no BUSSEL failures, the flip-flops are transparent. If a BUSSEL error occurs, then the value of BUSSEL used by the external buffers is latched and held at the last known consistent value. If the flip-flops or XOR logic fails, this will be detected as a stuck-off BERLOUT pin by the Test Report command. Any other failure will be detected by causing a single line failure during an error report.

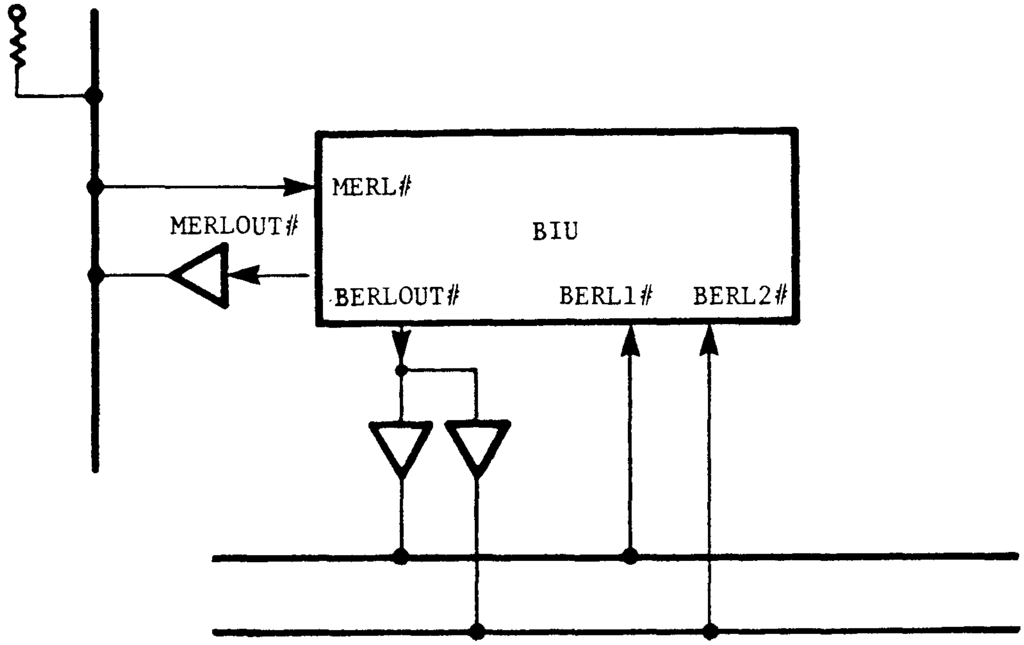
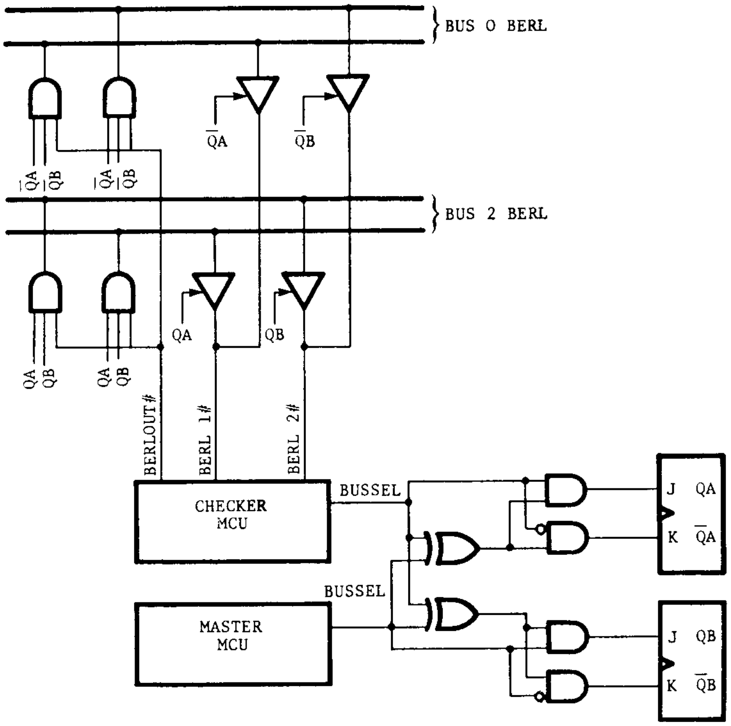


Figure 7-2. BIU Interfaces to Error Reporting Network F-0453



AN IDENTIAL NETWORK IS ATTACHED TO THE MASTER.

Figure 7-3. MCU Interfaces to Error Reporting Network F-0454

In summary, the error reporting network has the following important characteristics.

1. No single-point dependencies. No single failure can prevent the correct reporting of the error. The topology and the redundancy provided for each error report line are responsible. The worst-case failure will destroy one MERL and one BERL. However, the system will correctly report and isolate the failure.

2. Error reporting topology that allows graceful degradation. Each error reporting line resides in one of the confinement areas established for the system. Thus, the error reporting network changes with the system as it grows or degrades. The system's fault-tolerant capability will never be artificially limited by the error reporting network.

ERROR MESSAGE FORMAT

The format of the error report message is given below. It is followed by a description of each of the fields in the message.

Message format	START	2 bits
	SEQUENCE	1 bit
	ERROR TYPE	4 bits
	NODE ID	9 bits
	PARITY (EVEN)	1 bit

Start Field — These bits indicate the beginning of an error report message. While the line is idle, it has the value zero (high). The first bit of a message is a one (low), the second bit is a zero (high). This 2-bit sequence is used to prevent stuck-low failures from generating erroneous error reports. A stuck fault will generate only a single transition, while an error message requires a double transition to start (1-0-1).

Sequence Field — This bit indicates if this message is the original error report (0), or if it is being rebroadcast to propagate the error report message (1). The sequence field is used by the MCUs to track the propagation of error report messages.

Error Type Field — These 4 bits specify the type of error detected. The field is fully encoded; thus, each error message specifies one of sixteen possible error types. Each of the possible error types is described in a following section.

Node ID Field — This field will hold the node ID of the node that detected the error. The node ID is made up from two subfields, a bus ID and a module ID. The node ID specifies the physical interconnect location of this node. The node ID cannot be modified by software.

Node ID	Bus ID	3 bits (values 0-7)
	Module ID	6 bits (values 0-62; 63 is not valid)

Parity Field — This 1-bit field terminates the report message. The bit provides even parity over the message. (Even parity was chosen so that a stuck line -- (all ones) -- fault could be detected by the parity mechanism.)

ERROR REPORTING PROTOCOL

The error reporting protocol is designed to ensure that all of the nodes in the system receive the error message in a timely manner and, if multiple error messages are broadcast, that one and only one message is received by the nodes in the system. The propagation of error messages is shown in Figure 7-4.

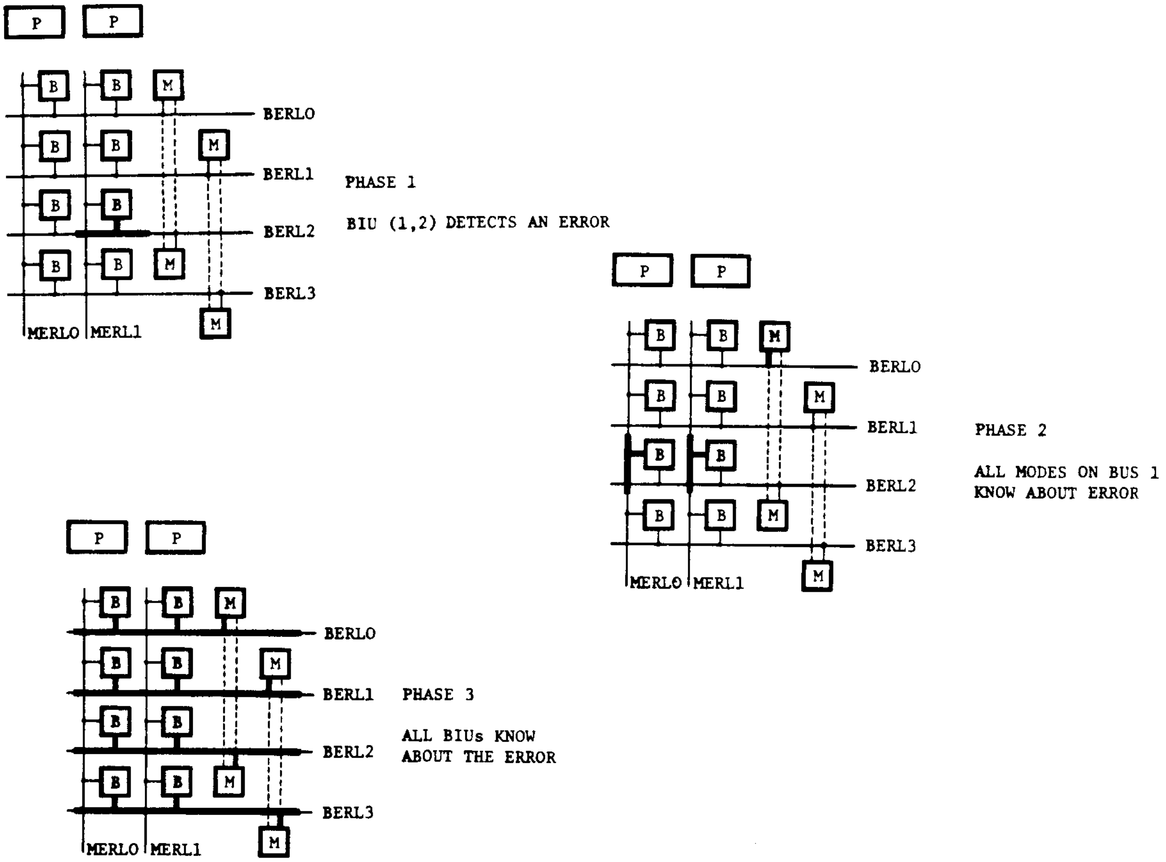


Figure 7-4. Error Report Propagation F-0436

PHASE 1

The node detecting the error sends an error report message along its BERL. The sequence field is a zero, indicating that this is the first BERL message in the error reporting sequence. All nodes on the bus terminate operation upon receiving a correct 2-bit start sequence at the beginning of an error message. The delay on this transmission is critical to guarantee that a confinement area does not operate on incorrect data. The node detecting an error will send the start bit of an error report message no later than one clock after the last cycle of the MACD bus message. The receiver of a MACD bus message waits for two clock cycles after the end of the bus message before using the message it received. This protocol guarantees that only valid data will be allowed to enter a confinement area.

The nodes attached to other buses need not be informed of the error immediately. The corrupt information can enter a module only by way of one of the nodes on the bus where the error was detected. Thus, fast message propagation is important only over that BERL line.

PHASE 2

After the complete error report message has been received over BERL, all the BIUs on this bus rebroadcast the message on their MERLs. This time, the sequence field is a one, indicating that this is not the first step in the error reporting sequence. The error report over MERL informs the BIUs on the other buses that an error has occurred. This information does not affect the normal operation of the BIUs.

PHASE 3

All BIUs in the system now know about the error. After receiving the complete error report message over MERL, all BIUs rebroadcast the message received over MERL on their respective BERL lines. The sequence field is a one, indicating that this is the second BERL message in the error reporting sequence. All nodes on the bus terminate operation upon receiving a correct 2-bit start sequence at the beginning of an error message. All nodes in the system now know about the error and all the nodes on each bus have stopped operation in unison. This action keeps the integrity of the normal bus protocol intact. There is no risk of a BIU retrying an operation that an MCU thought had been completed.

After receiving the complete error report message over BERL, all nodes load the last error message into their error report logs. This error message is known to be the last step in the sequence because it is a BERL message and because the sequence bit is set to one. The error reporting cycle is now complete.

On top of this basic transmission protocol there exists an arbitration mechanism that is used to resolve conflicts in cases where multiple nodes detect errors. There are two arbitration mechanisms. The primary one is FIFO ordering, in which the first error report message in a group has highest priority. The other mechanism is priority ordering, which is used when two error report messages are initiated at exactly the same time. In this scheme, the report with the higher priority message will win the arbitration. The implementation of these two arbitration mechanisms is described below.

FIFO ORDERING

The sequence field and the error report line used for a message (BERL or MERL) are also used to track the error reporting phases described earlier. The error report that has progressed the most, has highest priority For example, a MERL message has priority over a phase 1 BERL message). Once an error reporting phase has begun over a given line (e.g., phase 1 on BERL for bus 2), no other reports in that same phase are allowed on the line. Special circuits at the BERL and MERL receivers are responsible for blocking this node's BERL and MERL outputs if another error report message started prior to the one this node wanted to send. These special circuits prevent two messages from being combined to form an incorrect message. The message that starts first is received by all nodes on this bus.

PRIORITY ORDERING

During the transmission of an error report, a node may continue to broadcast only if the bit received in the last time interval was the bit it sent. Because the error report line is a wired-OR signal, a one (low) will override a zero (high) on the line. Thus, messages are ordered first by error type (because it is broadcast first), then by the ID of the node that detected the error.

These two arbitration mechanisms weed out the lower priority error reports during the reporting sequence. During phase 1, multiple error reports may exist in the system. However, on each bus, arbitration will allow only one report to successfully complete. During phase 2, any conflicts between the error reports on different buses will be resolved. At the end of phase 2, arbitration will have eliminated all but one error report throughout the system. During phase 2, an MCU may begin to generate a new error report in phase 1 (since it is unaware of the activity on MERL). The FIFO ordering arbitration will kill this report when the BIUs on the bus enter phase 3. After phase 3, every node in the system will have logged the same error message. All error information lost in the arbitration remains lost. If the error is permanent, it will be reported again during retry. If the error is a transient, then recovery occurs as part of the retry operation for the other error.

All nodes in the system will leave phase 3 of reporting and enter the recovery procedure at the same time. There are two cases when nodes in the system may not finish reporting at the same time. If a BERLOUT driver has stuck-on, then the bus on which the failure occurred will finish one cycle earlier than the other buses. If an MCU generates an error report while the BIUs are in phase 2 of another report, the bus with the late report may be up to 19 cycles behind the other buses. These skew problems are allowed to exist (i.e., they are treated as double-error conditions) because of the extremely low probability of a failure occurring as a result of the skew. Chapter 8 describes this skew problem in detail.

The only item left for the protocol to handle is error conditions. Clearly understanding the correct handling of errors in the error reporting lines requires an understanding of the recovery procedures. Thus, this section may be easier to grasp once Chapter 8, which discusses recovery, is read.

An error detected while a report is in progress may not be reported immediately. If the failure were immediately reported, all nodes might not log the same information. It is critically important to keep all the logs consistent because recovery is performed independently by each node, and different log values could cause inconsistencies in the system state. (For example, one BIU detects a MERL parity error while receiving a report over MERL. If it broadcasts a MERL parity error error report on BERL, then only that bus will have logged a MERL error report. If the error is reported again during the permanent error window, then only this bus will mark the MERL error as permanent. This will cause different reconfiguration action by those nodes seeing a permanent MERL error.) To prevent this inconsistency, errors that occur on the error reporting lines during error report transmission are handled in two steps.

When the error is detected, an internal flip-flop that will cause the error to be reported during the next error reporting cycle (one flip-flop for BERL errors and one for MERL errors) is set. If this node does not have a valid message to propagate, it will send a NO ERROR error message. This informs all the nodes that an error occurred but has no impact on the recovery procedures.

At the end of a reporting cycle, each node checks its MERL and BERL error flip-flops. If either one is set, then the node sets an override flip-flop. (There is only one flip-flop per node.) This will allow the node to broadcast a MERL or BERL error if it detects an error during the next error reporting cycle. The override flag guarantees that if the error condition is permanent, the error will be correctly reported. There is no danger of an inconsistent system state because this node is reporting the error at the very first opportunity during the next reporting cycle. All nodes will see this error report. (Assume, for example, that one node has a bad parity checker on its MERL input. On a routine error report, this node detects a MERL parity error. It will propagate a NO ERROR error report. During the next error reporting cycle, this node will generate a MERL ERROR error report. Once again it will detect a MERL error, but this time it will propagate a MERL error because the override is set. All of the nodes will see a MERL ERROR error report. On the next reporting cycle, a MERL error is again reported and logged by every node. This time it is labeled permanent, and the module is deallocated.)

DEFINITION OF ERROR TYPES

The error type field in the error message has 4 bits. The 16 error types are defined in Table 7-1. For cross-referencing, the detection mechanisms that can cause that error report are also listed and listed in priority order. (BERL error, code 1111, is the highest priority error type.)

Table 7-1. Error Type Field Definition

Error Type	Code	Detection Mechanisms
BERL	1111 (15)	BIU: BERL parity or sequence error MCU: None
MERL	1110 (14)	BIU: MERL parity or sequence error MCU: None
Unsafe Module	1101 (13)	BIU: None MCU: FRC on array side
Bus Arbitration	1100 (12)	BIU: FRC on RQOUT or MBOUT MCU: None
Bus/Module-High	1011 (11)	BIU: None MCU: BERL parity or sequence error FRC on MBOUT
No Error	1010 (10)	BIU/MCU: Sucessful Test Detection command or error report error and override Not Set
Module	1001 (9)	BIU: FRC on MACD or on NREQOUT MCU: FRC on MACD
Bus Parity	1000 (8)	BIU: MACD bus parity error MCU: None
Bus/Module-Low	0111 (7)	BIU: Any error when this module is broken MCU: MACD Bus Parity Error or any error when this module is broken
Uncorrectable ECC	0110 (6)	BIU: None MCU: Uncorrectable ECC Error
Correctable ECC	0101 (5)	BIU: None MCU: Correctable ECC Error
Test Report	0100 (4)	BIU/MCU: Command
Interchange	0011 (3)	BIU/MCU: Command
Attach	0010 (2)	BIU/MCU: Command
Detach	0001 (1)	BIU/MCU: Command
Toggle	0000 (0)	BIU/MCU: Command

The error type serves two purposes. First, together with the node ID, it identifies the confinement area in which the failure occurred. Second, it provides additional information to help field service staff repair the failure. The major considerations involved in establishing the error types are listed below.

- A single failure may trigger multiple detection mechanisms in several nodes. The error types must ordered such that the true failure is correctly identified.

- MCUs have connections to two buses and may switch buses in response to a failure. If a failure is in the MCU, it must not be allowed to switch buses and corrupt the back-up bus. If there is a failure in the MCU's MACD bus interface, both the bus and the module confinement areas will be identified as faulty. (See Chapter 5 for details about MCU error isolation.)

- Buses are a more critical resource than modules. When an error could be isolated by deallocating either a bus or a module, the module will be deallocated. Module shadowing is more flexible than MACD bus redundancy, and module repair is easier than bus repair.

Table 7-2 explains the motivation for all the error types and their placement in the priority list.

Table 7-2. Motivation for Error Types

Error Type	Confinement Area	Motivation
BERL	Bus	Critical resource damaged.
MERL	Module	Critical resource damaged.
Unsafe Module	Module (Unsafe)	Memory array that may hold corrupt data. Module must be killed immediately.
Bus Arbitration	Bus	Must be higher priority than MACD FRC and MACD parity because an arbitration failure's side effects could also cause these mechanisms to report errors.
Bus/Module-High	Bus and Module	An MCU error report. Must be lower than BERL and bus arbitration but higher than MACD FRC and MACD parity.

Table 7-2 - Continued

No Error	No Error	Special testing procedure. This code is used to allow testing of all of the bits in the error type field.
Module	Module	Identifies a confinement area. Must be higher priority than MACD parity error because a module failure at the MACD interface could also cause MACD parity mechansims to detect errors.
Bus Parity	Bus	Identifies a confinement area.
Bus/Module-Low	Bus and Module	MCU parity error must be lower priority than BIU parity error. Allows a node that cannot shut down to be isolated from the system.
Uncorrectable ECC	Module	Extra information.
Correctable ECC	Module (Cor ECC)	Extra information.
Test Report	No Error	Extra information.
Interchange	Bus (All)	Special testing procedure.
Attach	Bus (Return)	Special management procedure.
Detach	Bus	A management facility.
Toggle	No Error	Extra information.

ERROR REPORT LOG

There is an error report log in every node in the system. These logs are accessible to software, and they represent the primary form of communication between the hardware fault handling mechanisms and the software routines responsible for system management. When software reads an error log, it may read any log in the system. All logs will hold exactly the same information, except when a node is broken. A broken node will continue to detect errors and update its own log but cannot report the errors since it is isolated from the the rest of the system. The organization of the error log is described on the following pages.

Log Format

PERMANENT/TRANSIENT	1 bit (MSB)
ERROR COUNT	2 bits
ERROR TYPE	4 bits
NODE ID	9 bits

Permanent/Transient Field

This bit indicates whether a permanent or a transient error was reported. The bit is a one if a permanent error was reported; it is set by the recovery procedures operating in each node. (See Chapter 8 for details about the control of this bit.) If multiple error reports occur and any of the errors was labeled a permanent error this field will be a one.

Error Count Field

This 2-bit field is incremented each time an error is reported. The counter does not wrap around; if it reaches the value three, it stops until explicitly cleared. The counter is cleared by the INTERCHANGE error report. The error count field is used to identify an overflow condition to the software system. None of the other fields may be modified by software. When the count field has a value of zero, the other fields hold the information about the Interchange command. There is no error information in the log.

Value	Implication
0	No error information
1	A single error report
2	A single error if permanent bit is set and error type is not unsafe module; otherwise two errors have occurred
3	Multiple error reports

The error report log holds information relevant to the most recent error report. Except for the count field, all information associated with previous error reports is lost.

Error Type Field	This field is identical to the error type field in the error report message. If multiple error reports occur, this field will hold the error type of the most recent error reported.
Node ID Field	This field is identical to the node ID field in the error report message.

The error report logs are used independently by the hardware and software. The hardware recovery procedures use the error type and node ID fields in determining which recovery actions should be performed. This information always reflects the latest error report information. The hardware reacts immediately to each error report; thus, it never encounters an overflow condition.

The software system uses all of the log contents to monitor the health of the hardware system. Because the software system is only loosely bound to the fault handling mechanisms in the hardware, the count field is used to provide some synchronization information to the software. The rest of the log contents informs the software about the failure that was detected and the recovery actions that were invoked. The hardware updates the log as a single quantity; thus, if software reads the log as a single 16-bit quantity it will always receive correct and consistent information.

Software may clear the error count field by issuing an Interchange command. This command causes the interchange error report to be broadcast. The error report will cause all of the nodes in the system to clear their error count fields. This is the only way in which software may modify the contents of the error report log. The Interchange command also sets the permanent bit.

At initialization time the count field is zero and the permanent/transient bit is set to transient. The error type field is not modified, so that error information will be available after a system crash.

This capability allows the diagnostic software in non-fault-tolerant configurations to utilize the information collected by the hardware. It also means that after cold start in fault-tolerant configurations the error log should not be read until an Interchange command has been issued. Otherwise, the master and checker nodes may have different values in their logs, which will result in an FRC error.

ARRAY ERROR LOG

Each MCU has an array error log, which is used to log any failures detected by the ECC detection mechanism. Unlike the error report logs, the array error logs hold information that is local to this memory module. These logs never hold any information about memory arrays in other modules. The format of the array error log is described in the following paragraphs. The 32-bit register is divided into two 16-bit registers in the MCU.

Log Format

SCRUB ERROR FLAG	1 bit (MSB)
ECC SYNDROME	7 bits
FBE FLAG	1 bit
LOF OVERFLOW FLAG	1 bit
UNCORRRETABLE ERROR	1 bit
CORRECTABLE ERROR	1 bit
ERROR LOCATION	20 bits

Scrub Error Flag

This bit is set if the logged error resulted from a scrub operation. It is valid only if either the uncorrectable error flag or the correctable error flag is set. Scrub errors will be logged and reported over the error reporting network only if the array error log is empty (both correctable and uncorrectable error flags are reset). This prevents a hard failure in the array from needlessly saturating the system with error reports.

ECC Syndrome

This field contains the syndrome that was calculated from the failed location. If a single-bit failure has occurred, the syndrome identifies which bit failed. If a double-bit failure has occurred, the syndrome does not hold any useful information. If the syndrome has an odd number of ones, then it is a syndrome from a single-bit error. Note that the validity of the syndrome can be known only by counting the ones in the syndrome. None of the flags in the log provide this information. (Some types of uncorrectable errors are single-bit errors.) Table 7-3 gives the mapping between syndrome values and failed bit locations. All values not listed in the table represent multiple-bit errors. This field is valid only if the log is not empty.

FBE Flag

This bit is set whenever a Force Bad ECC (FBE) operation is performed. The FBE flag is always valid independent of the state of the other bits in the log, and it may be cleared by software. See Chapter 8 for a complete description of FBE operation.

Table 7-3. Syndrome Bit Mapping

SYNDROME VALUE	FAILED BIT	SYNDROME VALUE	FAILED BIT	SYNDROME VALUE	FAILED BIT
01	E0	31	A1	61	D17
02	E1	32	D26	62	D10
04	E2	34	D12	64	D24
07	D7	38	A19	67	A7
08	E3	3B	D27	68	A17
0B	A11	3D	D13	6B	D11
0D	D29	3E	D6	6D	A0
0E	D22				
10	E4	40	E6	70	A18
15	A5	43	A3	73	D19
16	D30	45	D21	75	D5
19	D9	46	D14	76	A8
1A	D2	49	D0	79	A9
1C	D28	4A	A10	7A	A2
1F	D31	4C	A4	7C	D20
		4F	D15	7F	FBE
20	E5	51	D1		
23	D3	52	D8		
26	A14	54	A12		
29	D25	57	D23		
2A	D18	58	A16		
2C	D4	5B	D16		
2F	A15	5D	A13		
		5E	A6		

Note: Syndrome value is in hexadecimal.

Failed bit codes:

A0..A19	Address bits
D0..D31	Data bits
E0..E6	ECC bits
FBE	Force Bad ECC

Log Overflow Flag

The log can hold the information for only one error. If an ECC error is detected and the array error log is not empty, then this bit is set. The log is empty if both the correctable error flag and the uncorrectable error flag are reset. All information about these later failures is lost. The log holds only the information relevant to the first error that was detected. The log overflow flag bit may be cleared by software.

Uncorrectable Error — This bit is set if the logged error was uncorrectable. Uncorrectable errors include: any failure in the address lines, double-bit failures in the array, and unused ECC codes. If this bit is set, the log is not empty. The uncorrectable error bit may be cleared by software.

Correctable Error — This bit is set if the logged error was correctable. Correctable errors include single-bit failures in the array. If this bit is set, the log is not empty. The correctable error bit may be cleared by software.

Error Location — This 20-bit field holds the array address of the word that failed. This is not the address sent over the memory bus, or the address sent out from the processor, it is the local array address. It is a word, not a byte address (a word is 4 bytes), and the address is array-relative (that is, the first location in each array is location zero.)

The array error log is updated by hardware but used only by software. The correctable and uncorrectable error flags act as synchronization locks to guarantee that software always receives consistent data. These locks are required because the 32 bits of information can be read by software only in two 16-bit accesses. The hardware will modify the contents of the syndrome and error location fields only if the log is empty. The FBE and overflow flags may be modified even if the log is not empty.

When software wants to read the array error log, it must read the Array Error Log - High register first. If the status bits indicate that the log is empty, then the syndrome and address fields are invalid. If the log is not empty, then the address field may be read. When the software is done with its analysis, it clears the error log by resetting the overflow, correctable, and uncorrectable flags. This sequence guarantees that the address and syndrome the software reads will always provide correct and consistent data. However, it is possible that additional ECC errors could set the overflow flag and that this action would not be noticed by the software.

MANAGEMENT AND TESTING

Error reporting is enabled at initialization, but may be disabled by software by way of the Disable Error Report bit in the diagnostic register in every node. Additionally, software may disable the reporting of ECC errors completely or disable only those ECC errors detected during scrub accesses. Disable scrub report and log and disable ECC error reporting disables are also located in the diagnostic register in MCUs. Under normal operating conditions, none of these disables will be asserted.

The major managerial chore for the software is monitoring the error logs in the system. The logs have been designed so that this function can be done in a simple and efficient manner. The basic software flow of control is given below:

1. Read any one of the error report logs in the system. The recommended approach is to read the log using a "my-BIU" access to a BIU on a working bus.

2. If the log is empty (error count is zero), then no errors have occurred in the system. There is no need to read any of the other logs in the system (including array error logs). If the log is not empty, then proceed to the next step in the algorithm.

3. If the error type is correctable ECC or uncorrectable ECC, then read the array error log in the MCU at the location specified in the node ID field of the error report.

4. If overflow occurred in the error reporting log and the permanent error bit is set, then the software must determine what type of reconfiguration has occurred. This check is performed by reading the state register in at least one node in every module and by reading one node (in a working module) on every bus in the system. (The Married bit tells if the module still has a shadow, the Enable Local bit tells if the module is working, and the Enable Primary and Enable backup bits tell if this bus is still working.)

5. If overflow occurred in the error reporting log, then the array logs in all of the MCUs must be checked for error activity.

6. Software now holds all of the available information about the failure(s) that occurred. This information can be used to update a system health file, invoke other layers of system management, etc.

7. Software's final action is to clear the logs. All of the error report logs are cleared by issuing a single Interchange command. Each of the array error logs that were not empty must be explicitly cleared by writing to the Array Error Log - High register. While software is clearing the logs, it is possible for a transient error to be logged and not noticed. It is impossible for software to miss the occurrence of a permanent error. Part of the recovery procedure from a permanent error causes a reconfiguration IPC message to be sent to every processor in the system. (See Chapter 8.)

The error reporting mechanisms are tested by the Test Report command. This command tests that the addressed BIU or MCU can correctly generate an error report message. It tests virtually all of the error report generation logic, receiving and sequencing logic, and error report log register and logic, and it provides a 100% test on the BERLOUT and MERLOUT drivers (internal and external). The command also tests that every MERL and BERL line in the system is working correctly. Unlike the other commands, the data field in the Test Report command is used to specify which component should generate the error report. Table 7-4 defines the allowable data values.

Table 7-4. Data Field for Test Report Command

Data Value	Component
0	Primary, Checker
1	Primary, Master
2	Shadow, Checker (BIU Only)
3	Shadow, Master (BIU Only)

Because the primary and shadow MCU modules are individually addressable, values 2 and 3 are not used by the MCU. Once the command has been issued, the software reads the error report log in the node addressed by the command. No type of delay is needed in the software because the read is guaranteed to be performed after the error report has been completed. The read must not use a "my-BIU" access. If the error log holds a test report error type and a node ID for the node that was addressed in the command, then the node's BERLOUT buffers and its error report log and logic are working correctly. Also, this confirms that all of the BERL lines are working correctly. If no error has been recorded in the log, this indicates a failure in the node's BERLOUT buffer or in its error report generation logic. Any other failure in the error reporting network will be correctly isolated and reported with the BERL or MERL error types.

A failure in the error report log will result in a module error when the attempt is made to read the log contents. Because the BIUs in a primary/shadow pair cannot be independently addressed, it is impossible to directly read each log. However, by writing a small loop that reads the same log many times, you will have a very high probability of reading the logs in both the primary and the shadow. This procedure checks all of the logs in the module pair. A complete test can be performed by testing the modules while they are not married.

The log is also checked indirectly by the recovery actions. The recovery procedures get their information from the error log, so a failure in the log will be reflected by incorrect recovery actions, which will be detected by FRC in the module. If the log holds some other error type, it is possible that the test report lost in the error report arbitration sequence. In this case, the test should be repeated.

The correct operation of a MERL line is checked by reading the Working/MERL register in a BIU. The MERL bit in this register is toggled each time an error report message is received over the MERL line. The operation of MERL is then checked by reading the value of the MERL bit before and after the Test Report command. If the MERL line is working correctly, then the two values will be different.

Because the BIUs in a primary/shadow pair cannot be independently addressed, it is impossible to directly check the MERL lines in married modules. This is not a major problem because of the massive redundancy in the MERL lines. (A system needs only one MERL for correct operation, and each married processor has four MERL lines.) The MERL lines can be checked in two ways. The first way is to do a probabilistic test while the module remains married. This is done simply by a tight loop that reads the register a number of times (about 10). With very high probability, the primary and the shadow each respond to at least one of the read requests. This gives a complete check of the MERL line. A second approach is to split apart the module pair for the MERL test. This will give the user a guaranteed check on the MERL line.

OVERVIEW

Previous chapters have explained the mechanisms for detecting and isolating failures to a confinement area, and the redundancy that can be carried in the system. This chapter explains how these capabilities can be used to mask the effect of any single failure in the system. The recovery procedures are located totally within the BIU and MCU. The same recovery algorithm is used independent of the amount of detection or redundancy available in the system.

Figure 8-1 provides a simple view of the recovery procedure. This diagram identifies the major steps in the recovery sequence. Each of the steps will be described in detail in this chapter. There will be a section on:

- Unsafe module decision
- Retry sequence
- Permanent error decision
- Resource reconfiguration
- Communication with the system software

Error detection, reporting, and logging have been discussed in earlier chapters.

The recovery algorithm is executed in parallel by all the BIUs and MCUs in the system. There is no global agent responsible for correct recovery actions. Each node performs its recovery sequence independently from all of the other nodes in the system. The thread tying all the recovery actions together is error reporting. The error reporting cycle ends with all nodes in unison, entering recovery in lock step.

UNSAFE MODULE DECISION

The recovery sequence is the same for all error types except one. Unsafe module errors are handled as permanent errors immediately. Unsafe module errors can only be reported by MCUs. This error type corresponds to a class of errors that corrupt data in the memory array, but may not be detected if the access is retried. (Chapter 5 has a more complete description of this class of errors.) Because the error may not be detected if the access is retried, the unsafe module must be immediately isolated from the system. For all other error types, retry is the first step in the recovery sequence.

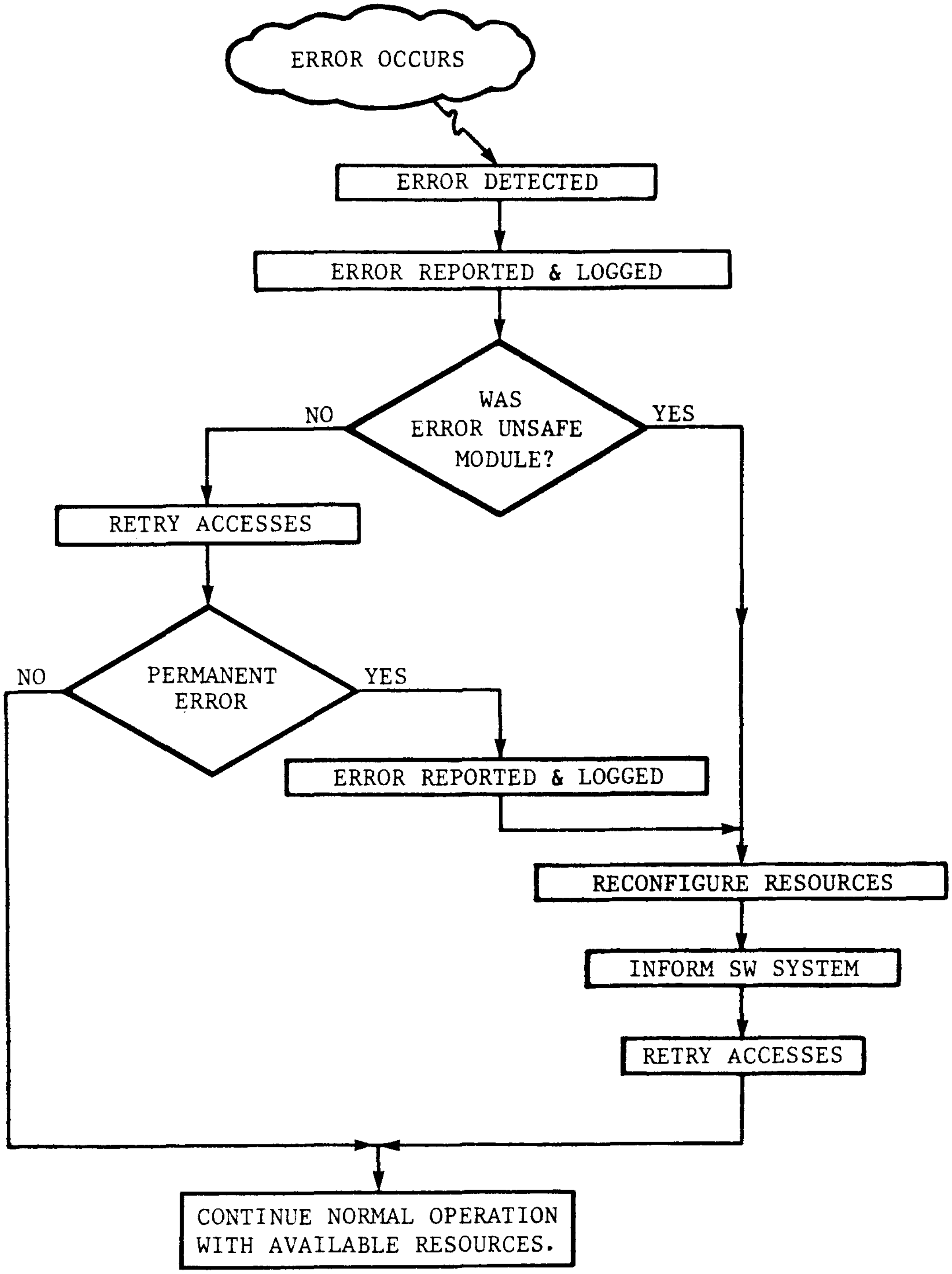


Figure 8-1. Recovery Procedure F-0456

The unsafe module decision is made simply by using the error type in the error report log. If the error type is unsafe module (code = 13), then the BIUs and MCUs will directly enter the reconfiguration sequence. In all other cases the nodes will begin the retry sequence immediately.

RETRY SEQUENCE

The retry sequence is broken into two parts. First, there is a waiting period during which the machine is quiescent. This is followed by a time window in which the nodes check for the same error to recur and all pending accesses are retried. It is important to realize that all nodes in the system enter this phase of retry on the same clock cycle. (Certain conditions can cause the nodes to enter retry out of step. These conditions were described in Chapter 7. Their effect on retry will be described later in this section.)

The quiescent waiting period is defined by the Timeout Duration register in all of the nodes in the system. This register is writeable by software. At initialization, the register is set to the value zero. All nodes in the system must have the same value in their Timeout Duration registers. The length of the time delay ranges from approximately 16 microseconds to over 2 seconds. The exact formula for the time delay is: (Timeout Duration * 2^16) + 128) * system clock period. The Timeout Duration register is 8 bits long (0 to 255).

The quiescent period allows transients in the system to subside before the accesses are retried. During this time there is no activity over the memory buses. Processors may generate new requests, but the BIUs will stretch the processor until the end of the waiting period. During the waiting period, refresh continues normally for the memory arrays. Errors are not reported during this time interval. The only type of error that will be latched and reported later is an unsafe module error at a MCU. All other error conditions will be ignored. (If they have affected the state of the module, they will be detected during the second phase of retry.) The time interval is adjustable over a wide range to allow the waiting period to be tailored to the operating environment of the system. The delay is long enough to handle transients induced by mechanical stress as well as by electrical interference.

At the end of the quiescent waiting period, all the nodes enter the second phase of retry -- the permanent error window. During this time interval, all pending memory accesses will be retried, and the system will be monitored for a permanent error condition. The access may not be retried by the same node. At the start of retry, the primary module always becomes the active module. This approach is required to keep the primary and shadow modules in step in the presence of bus errors. Otherwise, both modules might think they are passive. The Timeout Duration register also controls the length of the permanent error window. The quiescent waiting period and the permanent error window will always use the same time interval.

Every access pending at the start of the permanent error window will be retried. This includes those accesses that may have originated while the system was in the quiescent waiting period. During the quiescent period, all of the MACD bus arbitration state information was flushed out. There is no bus state left over from the period before the error occurred. All of the pending accesses are placed in one of two groups, depending on access type.

The first group holds writes, RMW-writes, FBE commands, and RMW-read enqueue (if it is an MMA and the other half has been completed). All BIUs with an access in this group will place their requests in the first slot of the time-ordering request queue. The second group contains all of the read and RMW-read enqueue requests. All BIUs with an access in this group will place their requests in the second slot of the time ordering request queue. Priority arbitration will then provide a second level of arbitration just as it does during normal operation. This sequencing means that all accesses in the first group will be completed before any access in the second group is completed. This action provides logical ordering for the whole system; but physically, this ordering is on a bus-by-bus basis, not system-wide (just like normal arbitration ordering). Any request that arrives after the start of the permanent error window will be handled normally. None of these requests will be completed until all of the retried requests have been completed.

Retry has the following characteristics:

- The ordering of requests during retry is likely to be different from the ordering of requests before the error.

- All writes will occur before reads. This guarantees that all of the read requests will return with consistent information. When the error occurred, there may have been some half-finished MMA accesses. This leaves the system in an inconsistent state. By issuing all writes before reads, the system is returned to a consistent state before any read access occurs.

- The uncompleted half of a partially completed RMW-read operation will be enqueued before any other RMW-read enqueue operations. This is done to guarantee that a deadlock will not occur between two RMW requests. (A deadlock could occur if two RMW MMA requests spanned the same location, with one request locking the memory on one bus, and the other request locking the memory on the second bus.) This does not mean there will be only one enqueue during the first time period, or that the RMW-read is guaranteed to be successful. It simply allows all the requests that have locked another resource to have priority over those requests that have not been granted a lock, thereby preventing deadlock.

- There will not be any correctable ECC error reports generated during the permanent error window. During the permanent error window the MCUs check the ECC code before data is returned to the

BIU. If a correctable ECC error is detected, the MCU will correct the data before it is sent to the BIU. The error will be logged in that particular MCU's array error log, but no error report message will be generated.

- The completed half of a partially completed RMW-read operation will be converted into a normal read access. Because the lock has already been set for the RMW operation, if an RMW-read is retried, it will never be granted; the location is locked and will never be unlocked.

- The completed half of a partially completed RMW-write operation will be converted into a normal write access. This must be done to prevent clearing a lock that may have been set by another access.

Except for the special arbitration sequence for the retried accesses, operation during the permanent error window is identical to normal operation. The difference is in the response to errors. The permanent error window defines the only time period in which an error can be labeled a permanent error (except for the unsafe module, which is always a permanent error). Because there is no guarantee that the same node will retry the access, it is important to have a reasonably long permanent error window. This prevents a permanent error from appearing as a string of transients.

Chapter 7 presented two error conditions (stuck-on BERLOUT driver and MCU error reports that begin during phase 2 of another error reporting cycle -- see page 7-8) that will prevent the system from entering retry in lock step. The skew may be from 1 to 19 cycles in length. This skew is a problem because of the special arbitration sequence used for retried accesses. It is possible for an MMA access to be treated differently by the two buses. One bus may place the access with accesses that will be retried, while the other bus may handle the access as a normal access. (Only one of the buses has entered the permanent error window.) This action violates one of the rules that guarantee consistent data is returned from memory: accesses that span two buses must be handled in the same order on both buses. The exact conditions necessary for this error to occur are these:

1. One of the two conditions that can cause a skew must be present. (See page 7-8.)

2. A processor must make an access in the failure window caused by the skew. This condition can occur only if the processor does not have an access pending for retry, and only if it does not make an access for the duration of the quiescent waiting period. This condition cannot be met by a GDP. The probability of an IP not making an access during the quiescent period (which is probably hundreds of milliseconds long) is very small. The failure window is defined as the time interval after the first bus has entered retry but before the second bus ends its quiescent period.

3. The access that hits this failure window (100 nanoseconds to 2 microseconds long) must be an MMA write or an MMA RMW-read access.

4. This location must be a location that is being shared with other processes.

5. Another process must have an access pending to this same location which will be retried.

If all of these conditions are present, then it is possible for inconsistent data to be read from the memory. The probability of all of these occurrences happening is extremely small. Thus, this error condition is similar to the rare occasions when multiple errors occur simultaneously, and the system may not be able to recover properly.

PERMANENT ERROR DECISION

Permanent errors are defined as errors that occur twice in a row within a specified time interval. The time interval is defined by the permanent error window. The second occurrence of an error report must have an error type field identical to that of the first report, and the node ID field must be a node in the same confinement area as the first error report. Thus, bus type errors must have the same bus ID; module type errors must have the same module ID; and bus/module type errors must have the same node ID (bus and module). Note that it is not possible for a correctable ECC error report to be labeled as a permanent error because correctable ECC errors are not reported during the permanent error window. (The error is simply corrected by the MCU.)

If an error is identified as a permanent error, then the permanent error bit in the error report log will be set, and the BIUs and MCUs will enter the reconfiguration sequence of the recovery algorithm.

ISOLATION OF FAULTY CONFINEMENT AREA

Table 7-2, shown previously, lists the error reports and the confinement areas each one identifies. Each error report identifies either a module, or a bus, or one module and one bus as the areas where the fault occurred. Each node reads the node ID field in the error report and decides if it is in the faulty confinement area. The following paragraphs describe the deactivation of each of the four types of confinement areas that exist in an iAPX 432 system.

GDP OR IP MODULE

A processor module is deactivated when all of its BIUs:

- Disable primary and back-up memory address ranges. This is done using the 4 Bus State bits in the State register. These bits are all reset to zero.

- Disable interconnect register accesses. This is done by clearing the Enable Local bit in the State register.

- Ignore the MERL line. The MERL receiver is disabled in each BIU.

In this state the module is totally deactivated. No requests of any type may be generated by this module. The BIUs will still reply to interconnect register and IPC requests issued from other processors. The error detection circuits at the MACD interface remain enabled, and if an error is detected it will be reported. All errors will have the bus/module error type.

MEMORY MODULE

A memory module is disabled when the MCU disables its memory address range. This is done by clearing the Memory Enabled bit in the State register. The MCU will still respond to interconnect register requests. All of the error detection circuits, except buffer check, remain enabled. All errors will have the bus/module error type. The MCU will remain attached to the same MACD bus.

MEMORY BUS

A memory bus is disabled when all of the BIUs attached to the bus disable their memory address range. This is done by clearing the Enable Primary and Enable Backup bits in the State register. Interconnect register accesses may still use the bus. The action of MCUs depends on the availability of a back-up bus. If possible, the MCU will switch to the redundant bus; otherwise, the MCU will clear all four of the bus state bits. Error report generation and MERL propagation are turned off in all of the BIUs on this bus. This action permits error reports on the faulty bus from propagating into the rest of the system. MCU error report generation is turned off only when both its bus and module are isolated.

These actions isolate the faulty confinement area from the rest of the system. If any of the BIUs or MCUs does not correctly deactivate, it will be physically isolated from the system by deactivating the other confinement area (bus or module) it touches. This occurs because the ERROR REPORT will be of type BUS/MODULE or another BIU in this module may generate a module error after a bus has been deactivated. Now the failed component is totally isolated from the system by a set of nodes known to be good. At this point the bus and the module attached to this node have been deactivated.

RESOURCE RECONFIGURATION

After the faulty confinement area has been isolated from the system, then either the back-up resources are activated or, if no redundant resources are available, the damaged resources are locked. Reconfiguration, like all the other aspects of recovery, is done independently by each BIU and MCU. There is no central unit responsible for reconfiguring the system. All reconfiguration actions take place during the first few cycles of the quiescent waiting period.

ACTIVATE REDUNDANT RESOURCES

Each BIU and MCU reads its error report log and decides if it is part of the redundant confinement area that should be activated to recover from the error.

GDP, IP, OR MEMORY MODULE CONFINEMENT AREA

If the module ID in the error report matches the spouse ID in this node, and if the married bit in the State register is true, then this module is the redundant resource and should be activated. The only action required to activate a module is to clear its married bit. This will cause the module to be active on every access, rather than alternating accesses with its spouse. Thus, the redundant module will mask the permanent failure in the spouse module.

MEMORY BUS CONFINEMENT AREA

If the bus ID in the error report matches the bus ID of this node's back-up bus, and if either Enable Primary or Enable Backup is set, and if either the memory buses are four-way interleaved or there is a redundant bus, then this bus is the redundant resource, and it should activate its back-up address range. The bus is activated to cover the full address range by setting both the Enable Primary and Enable Backup bits in the State register. MCUs that were attached to the faulty bus must switch to the back-up bus. If the MCU's bus ID matches the bus ID in the error report, and if the Bus State bits indicate that the back-up bus is available, then the MCU will toggle BUSSEL, which will cause the MCU to attach to its back-up bus. All MCUs have moved to the working bus in the bus pair, and the BIUs are recognizing the address range that used to be covered by two buses. Thus, the back-up bus will mask the permanent failure on its partner bus. Figures 6-1 and 6-2 describe the state machines that control the Bus State bits during memory bus reconfiguration.

LOCK-DAMAGED RESOURCES

The actions described previously will activate a redundant resource if one is available. If none is available, then the system must lock any resources that may have been damaged as a result of the error condition. Since the faulty confinement areas have already been removed from the system, the only resource that may be damaged is a memory module corrupted by a partially completed MMA access. If an error occurs during an MMA write operation, one of the memory modules may hold new data while the other memory module holds old data. Thus, this logical memory structure holds inconsistent, and therefore corrupt, data. This corrupt memory structure is locked to prevent access by other modules. (See Chapter 9 for a complete description of this locking operation.) It is important to understand that resources can be damaged only if a system is not carrying full redundancy. This situation can never occur in a fully redundant system.

At this point, reconfiguration is complete. The faulty confinement areas have been isolated from the system. If redundant resources were available, they will have been activated to mask the effects of the failure. If redundant resources were not available, any memory location that might have been damaged by a partially completed MMA write access will have been locked.

COMMUNICATION WITH SYSTEM SOFTWARE

If the failure is a transient error, then communication with the system software is done only by way of the error report logs. These logs are periodically polled as part of the general housekeeping services provided by the software. If the failure is a permanent one, it requires immediate attention to minimize the system's exposure to errors while it is vulnerable to failure (i.e., while at least one resource in the system no longer has a redundant back-up resource).

This action is achieved by an IPC message automatically generated by the BIUs. Any time an error condition is labeled a permanent error, all of the processors in the system receive a reconfiguration IPC message. This is done independently by each BIU in the system; no IPC message is actually sent down the memory bus. This IPC causes the GDPs to suspend their current processes as soon as possible and then dispatch themselves from the reconfiguration dispatching port. At this port the software system will have placed a process designed to activate other processes that can make management decisions about the best mode of operation for the system. When an IP receives a reconfiguration IPC, it will send an interrupt to its AP. In this way the system software is immediately informed of the vulnerable state of the system and can react quickly to degrade the system in the manner optimal for the application.

RECOVERY IN THE MEMORY ARRAY

The previous sections have dealt with recovery at the level of a confinement area. The iAPX 432 also has recovery capability within its memory module. There are three basic mechanisms for recovery of errors in the memory array: ECC, refresh scrubbing, and a spare memory bit.

The ECC code, described in the chapter on detection mechanisms (Chapter 5), is capable of correcting single-bit errors in the memory array and of detecting double-bit errors. Normal operation is optimized for performance under error-free conditions. One aspect of this optimization is that memory data is sent over the memory bus before the ECC code has been checked. During the permanent error window, the MCU operates in staged mode. Staging means that the MCU will return memory data only after correcting any correctable errors detected by the ECC code. (Correctable ECC errors will not be reported.) This mode of operation is needed to correctly handle permanent single-bit failures in the RAM array. During retry, the access to the faulty location will return valid data to the processor without any error report, thereby allowing computation to move forward.

Scrubbing is a recovery mechanism that is an integral part of refresh. During each refresh access, one location in the array is actually read. If an error is detected, it is reported, and corrected data is written back into the array (if possible). This provides an ECC check on every location in the memory system approximately once a second. These frequent ECC checks and corrections virtually eliminate the possibility of an access encountering a word with a double-bit error. By tying scrubbing to refresh, this function is achieved without any additional performance degradation or software intervention.

Each memory array may carry a spare RAM bit. This bit may be swapped-in to replace the data in any of the normal 39-bit locations. The spare bit is totally under the control of the software. The operation and use of the spare bit are described in Chapter 6.

SUMMARY

Figure 8-2 gives a detailed flow diagram of the recovery process. If possible, error recovery is done by way of retry. Retry will provide recovery for the most frequent failure modes (RAM soft errors, noise on the backplane, etc.). If retry fails, then it is possible to use a redundant resource to mask the fault from the rest of the system. Resource reconfiguration can successfully mask any single failure in the system. The entire recovery algorithm is implemented in the BIU and MCU, and is orthogonal to both detection and redundancy mechanisms.

There are three significant metrics for judging the effectiveness of the recovery strategy: the amount of overhead during normal operation, the amount of computation time lost during recovery, and the extent of system vulnerability after recovery.

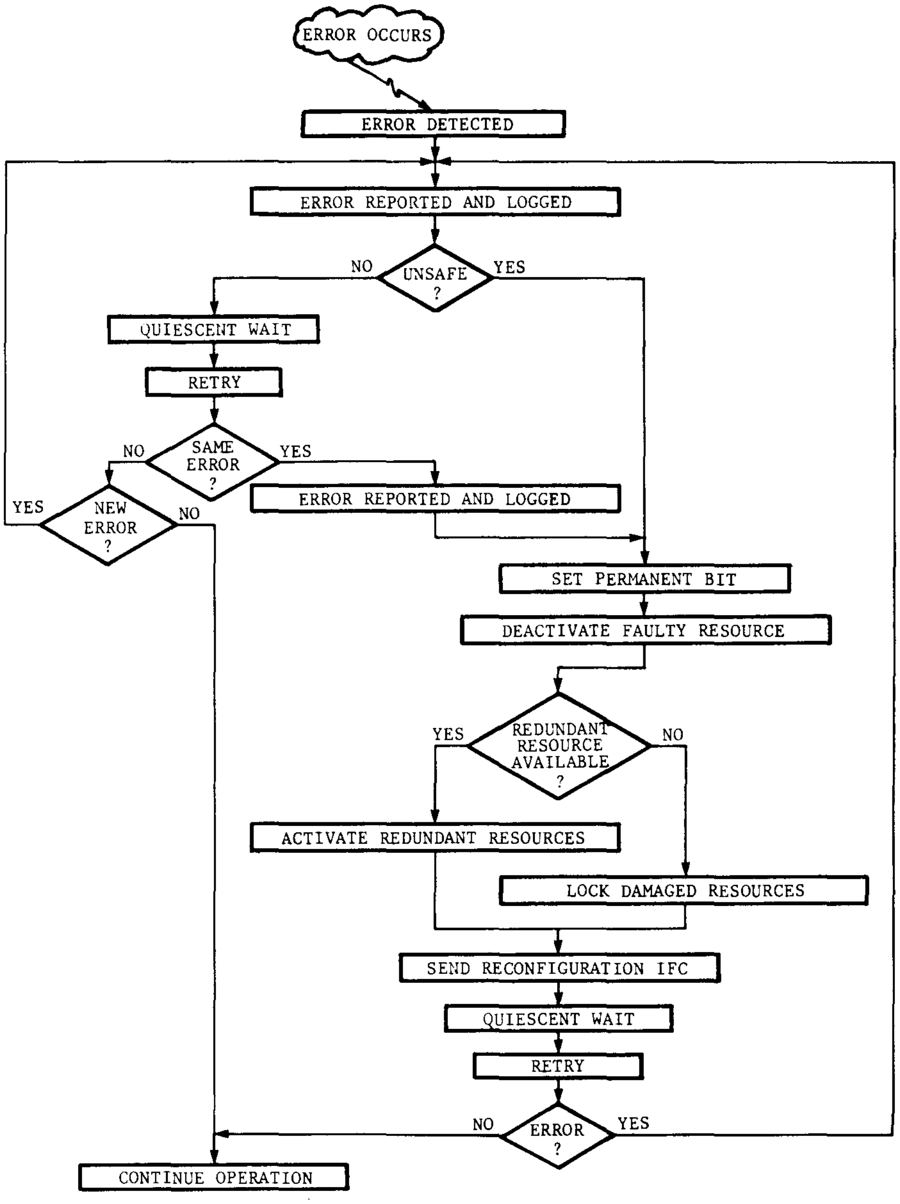


Figure 8-2. Detailed Flow Diagram of the Recovery Process F-0457

With the iAPX 432, the only impact on normal operation is an increase of about 10% in memory access latency to support ECC. None of the other recovery mechanisms impact normal operation. The recovery time is adjusted by the Timeout Duration register. By adjusting the length of the quiescent waiting period, the recovery time can be traded off against the ability of the system to mask transients. When the system is optimized for fast recovery (short timeout duration), the system is quiescent for approximately 32 microseconds before operation is restored. The system is vulnerable to a second failure until the exposed resource is either deactivated or married to a spare module in the system. For processors, this exposure is less than a millisecond. For memories, the exposure is less than a second. For buses, the system may be exposed until the bus can be repaired. It is important to remember that the modular nature of the iAPX 432 reduces this exposure even further. During the vulnerable period, only one part of the machine -- the confinement area which has failed -- is vulnerable. Other processors, or buses, or memories are not vulnerable. The system can survive a very large number of multiple faults. However, part of the system is vulnerable to a second error, and it needs to be deactivated or protected as rapidly as possible.

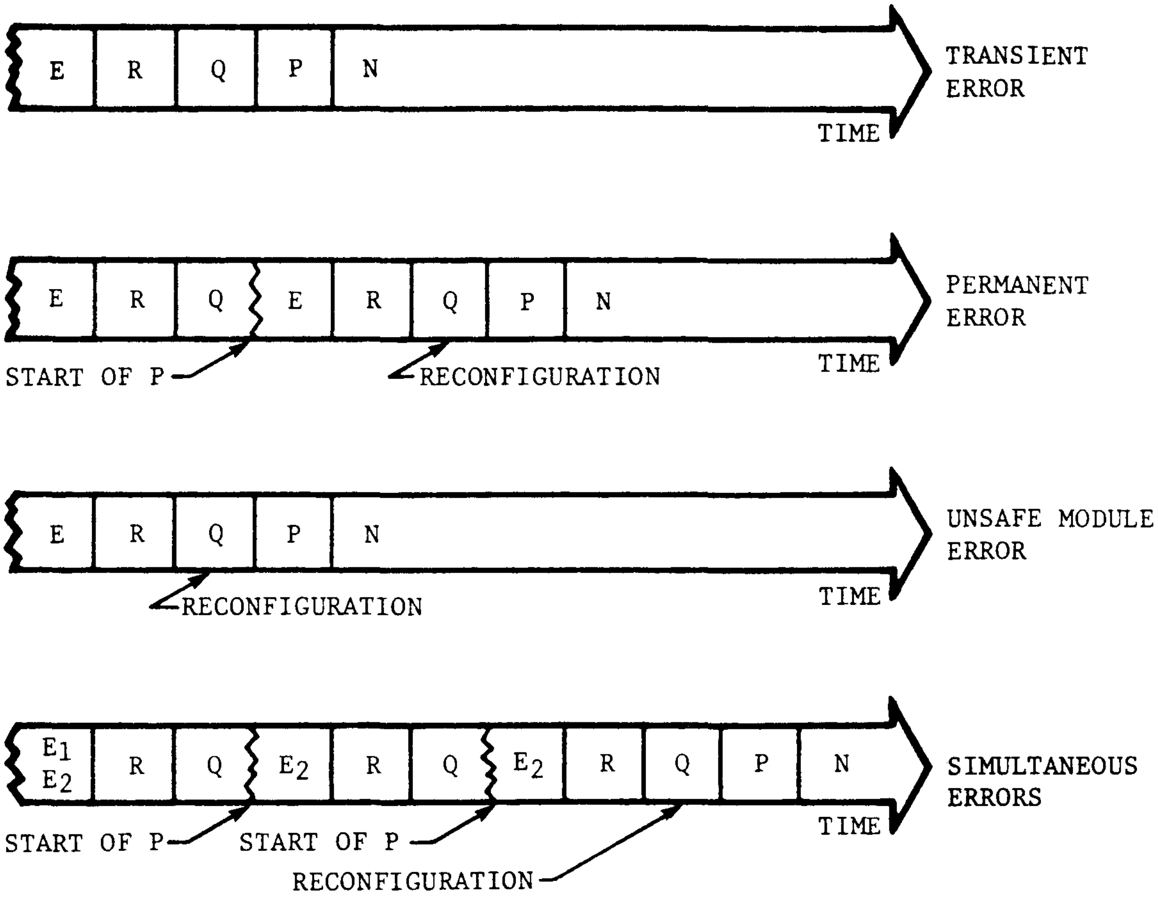
Figure 8-3 shows the recovery sequence for four different error cases. These diagrams trace the behavior of the system as it responds to an error.

MANAGEMENT AND TESTING

There are two key management functions that software is required to perform: reconfiguration policy, and reviewing the recovery decisions made by the hardware.

Chapter 6, on redundant resources, described all of the procedures used to attach or detach redundant resources. There also needs to be a process that is always waiting at the reconfiguration dispatching port. There is no reason that all of the processors must use the same reconfiguration dispatching port. It may be desirable to associate each processor with a unique dispatching port for recovery. The process waiting at the reconfiguration dispatching port is used to notify management software that hardware reconfiguration has occurred. After doing this, it will most likely redispatch the processors from their normal dispatching ports. This software needs special consideration because, while it is running, none of the normal operating system functions are available. (Those processes are available only at the normal dispatching port.)

System software must also determine the value for the Timeout Duration register. This value controls the length of the quiescent waiting period and of the permanent error window. It allows the user to trade off error response time against transient error masking. At the minimum value, permanent error recovery will occur in approximately 32 microseconds. However, any transient with a duration of more than 16 microseconds will be classified as a permanent error. As the value of the Timeout Duration register grows, the time for permanent error recovery grows longer, while longer transient errors will be successfully masked.



E - ERROR DETECTED
R - ERROR REPORT
Q - QUIESCENT WAIT
P - PERMANENT WINDOW
N - NORMAL OPERATIONS
E1 - HIGH PRIORITY TRANSIENT
E2 - LOW PRIORITY PERMANENT

Figure 8-3. Example Recovery Sequences

F-0458

On top of these basic functions must reside a management strategy that describes the action that should be taken when a particular resource in the system is deactivated. The hardware has handled the actual reconfiguration and maintained the correct operation of the machine, but the system software must now decide which configuration is optimal, given the reduced resources available to the system.

The second software function is the review of the hardware reconfiguration decisions. The hardware algorithm for classifying errors as permanent or transient may not be optimal for a given system environment. By looking at the history of errors in the machine, the software may be able to spot intermittent failures or transient failures which were labeled permanent failures. This is important, because there are many transient failures in the processors which cannot be recovered from by retry. These failures will be labeled as permanent errors. In some environments, software's first response to a permanent error in a processor may be to put the deactivated confinement area back in service. Only if it fails again will software actually consider the error to be permanent. This management software is also responsible for activating the spare bit in a memory module that has a bad bit.

Basically, this review process involves using an error history of the machine and a special knowledge about the machine environment to optimize the recovery procedures in the system. The hardware will never allow a faulty confinement area to continue running. However, it may deactivate a confinement area that is error-free. (That is, the hardware may have labeled a transient error a permanent error.) Some errors can be isolated by deactivating either a bus or a module confinement area. A particular system application may choose to deactivate a different confinement area than the one originally deactivated by the hardware.

INITIALIZATION

The fault handling mechanisms address two aspects of initialization: checking that the critical initialization data was correctly transferred into the components, and providing an external indication that the component has received the initialization signal.

The initialization data is loaded in from the memory bus and the ACD or SLAD bus while the INIT signal is asserted. Appendix D, "External Initialization," describes all of the initialization parameters. Only two of these parameters are critical to the basic operation of the fault handling mechanisms: bus ID and module ID. This information must be correctly loaded, or the error reporting mechanisms will not operate correctly.

The bus ID is loaded in from the memory bus pins. The 2 parity bits on the memory bus are valid during initialization. If there is an error in the initialization data, it will be detected by parity. Once the bus ID has been correctly loaded in during initialization, any failure in the Interconnect Device ID register which corrupts the bus ID will result in a module failure (or be uncovered during testing of the error reporting circuits). The bus ID is used only to recognize requests on the processor bus and to send the node ID in error report messages. Thus, the faulty bus ID will lead to an FRC error because of incorrect address recognition, or the software checking the error reporting network will notice that the node responded to a Test Report command with the wrong bus ID in the error report message.

The module ID not only needs to be loaded correctly, but also needs to be checked continuously during normal operation. The module ID is loaded from the ACD or SLAD bus during initialization. An even parity bit is appended to the 6-bit ID. This parity bit is stored in the Interconnect Device ID register along with the module ID. Any single error in the ID during initialization or during normal operation will be detected by the parity bit. This more comprehensive coverage is required because a faulty module ID occurring during system operation would almost certainly lead to a system crash. It would most likely lead to a permanent module error being reported with the wrong module ID and thus the wrong module (possibly this module's spouse) would be killed.

These conditions call for total shutdown of the node. If a memory bus parity error is detected during initialization or if the module ID has a parity failure at any time, this node will be immediately locked into the initialization state. This disables outputs and turns off all address recognition. The node simply disappears from the system. The only way to recover the node is to assert the external INIT pin. If correct data is reloaded into the internal registers, then the component will return to normal operation.

These drastic measures are required because this information represents the fundamental base on which the rest of the system operation is built. If a node disappears, this error will be detected by an FRC error in the checker (if the master failed) or by software checking the operation of the checker.

While an MCU or BIU has its INIT pin asserted, the component will assert is BCHK signal (low). This signal gives the external world an indication that the component is receiving the INIT signal. Normally the BCHK signal will either be oscillating or low, depending on the state of the BCHK enable. If the component did not receive the initialization signal, then external logic can be used to isolate the component from the system. This may be useful for on-line repair of modules.

INTERCONNECT REGISTER ACCESSES

The interconnect register address space is fixed to the physical resources in the system. This address space does not change. It is unaffected by any of the recovery mechanisms. This allows diagnostics and configuration software direct access to a specific physical node rather than through a logical mapping, which might change because of recovery. A side effect of this approach is that the software may address a node that is not accessible (bus switching of MCUs) or use a bus that is not reliable. These problems are handled by having a bad access flag in the BIUs (in the diagnostic register).

The bad access flag has two purposes. First it prevents any access from hanging up indefinitely. Second, it provides a flag that software can check to make sure that all of its interconnect accesses were correctly performed. The bad access flag in a BIU is set if a timeout occurs on a request from the BIU or if the bus to which the BIU is attached is deactivated. If an interconnect register access is performed and the bad access flag is set, then the BIU will treat the access as a NO-OP, but will acknowledge the processor's request normally (returning garbage data if it was a read). A bus error will not be signaled to the processor. This mode of operation prevents any interconnect access from hanging up under any error conditions. The processor will continue as if the access had been carried out correctly.

Upon completing a series of interconnect accesses, the software must check that the accesses were carried out correctly. If the bad access flag is cleared, then the accesses were completed correctly. If the flag is set, then one or more of the interconnect accesses were not carried out. The flag is always accessible to the software by way of a "my-BIU" type of register access. This type of access will always return valid data. (It is done totally within this processor module.) A problem with using the bad access flag is that the process may be moved between physical processors during execution. Thus, additional information is required to correctly interpret the flag. During normal operation the only way the flag can be set is if a permanent error occurs. (Timeouts will only occur during system configuration.)

This condition can be detected by the software residing in the reconfiguration dispatching port and be communicated to the software that may be performing interconnect accesses. The software could expand a single interconnect access to include a check on the physical processor being used. A single interconnect register operation would become: Read processor ID ("my-BIU"), perform interconnect access, read bad access ("my-BIU"), read processor ID ("my-BIU"). If the processor IDs were identical, then the bad access flag would be valid.

Because of the physical implementation of the BIU and MCU, only one process may access a node at any given time. If multiple nodes are accessing the same node, the results of the accesses are indeterminate. Software alone is responsible for guaranteeing that this mutual exclusion requirement is met.

INTERPROCESSOR COMMUNICATION

Interprocessor Communication (IPC) messages are handled in a special way by the BIUs. Logically, all IPCs use bus 0 and the BIUs on bus 0 are responsible for informing the processor about any IPC messages pending for that processor. This information must be duplicated to allow for recovery from bus 0 failures.

Bus 2 is the back-up bus for bus 0; thus, all IPC messages are sent over both bus 0 and bus 2. Because two buses are used for the message transmission, IPCs are treated like MMA accesses. This guarantees that the information in the IPC registers on both buses remains consistent. While both BIUs participate on the memory bus side of the IPC message, only one of the BIUs actually responds to the request on the processor bus. When the processor reads its IPC register, the BIU on bus 0 responds if its primary address range is enabled, while the BIU on bus 2 responds if its back-up address range is enabled. During normal operation, the BIU on bus 0 will return data on IPC requests. The operation of the BIU on bus 2 can be checked by doing an Interchange command, which will then cause the BIU on bus 2 to return data on IPC requests. The BIU that does not respond on the processor bus updates its IPC register to maintain an accurate copy of the state of IPC messages.

Both IPC read and write use the memory bus and are handled as MMA accesses over bus 0 and bus 2. This approach utilizes the bus arbitration protocol to guarantee that the IPC information is always consistent. For example, an IPC read following behind an IPC write will return the same data from both IPC registers because the read cannot complete until the write has completed. The order of access will be the same on both buses.

MULTIPLE MODULE ACCESS

Multiple module accesses (MMA) are those accesses that span two memory buses. Because the two buses operate independently, an MMA may be in different states on each bus when an error occurs. This requires special consideration during recovery. There are two cases of interest: Read-Modify-Write (read) requests and write requests. The RMW requests were described in Chapter 8.

Write requests are a problem because they may leave the memory data in an inconsistent state (part old data, part new data). This failure can occur only if a BIU fails before completing its write request, but after the other half of the MMA has been completed. If the error is permanent and there is no shadow module, then there is no way to correct the inconsistent data structure in the memory. A failure in the MCU or the memory array can never cause this problem. If the failure is in the memory module, no other processor will be allowed to access the memory array.

By monitoring the MMAL and MMAH signals on the processor bus, the BIUs can track the progress of the MMA operation on the other bus. If this error situation occurs (with the conditions: permanent module error, my module, not married, MMA write access, my half complete), the BIU that completed its part of the MMA write access must lock the memory location on its bus. This is done by issuing a Force Bad ECC (FBE) request on the memory bus. This request will cause the MCU to write a special ECC code (the complement of the correct ECC code) into the addressed location. Any requests to this location will be rejected because the special FBE code (an error syndrome of all ones) is interpreted as an uncorrectable error. The FBE code will prevent any further accesses to the corrupt location.

It is important to realize that only one of the two locations involved in the original MMA access receives an FBE command. The other location may be accessed without any problems. This does not cause any logical inconsistencies. The only inconsistency occurs when a processor tries to access both locations as a single unit. To prevent that, one of the locations is forced to have a bad ECC code.

ACTIVATION OF SPARE RESOURCES

The iAPX 432's concept of shadowed, self-checking modules allows a number of spare strategies to be used in a system. The first tradeoff involves a decision between carrying the extra resources as spares or as extra horsepower available to the system. If it is possible to do some load shedding in the event of a failure, then spare resources probably are a bad idea. Spare resources are activated for three reasons: the system management has decided to maintain the old performance level, a totally fault-tolerant environment, or both. Depending on the resources that are available, these decisions may require activating spare resources. Spare resources are activated on-line by using the procedures outlined for module shadowing startup in Chapter 8.

REGISTERS

This section provides a general review of all the registers that play a role in the communication between software and the hardware fault handling mechanisms. For each register there is a general summary of its function, its state after initialization, and any access restrictions. For a detailed description of the interconnect registers see Appendices A, B, and C of this manual.

DIAGNOSTIC

This register holds a set of bits that enable many of the optional capabilities in the MCU and BIU components. The register also contains a few bits (M/C Toggle, Bad Access) that are used during normal operation. This register is initialized to zero, which enables all of the detection and reporting mechanisms. There are no restrictions on access to this register.

The specific bit definitions follow.

- Disable MACD Bus Parity Detection - Self-explanatory.

- Disable Recovery - The purpose of this bit is to allow a diagnostic to run without dying due to some type of fault but still allow error reporting and logging. This may be used with a software loop to permit observation of the fault by oscilloscope or logic analyzer. The Disable Recovery bit disables any type of bus switching due to error report or interconnect command except for Bus Interchange. Additionaly, the Alarm bit in the BIU's IPC register cannot be set if recovery is disabled.

- Disable Error Report - This bit prevents a node from reporting errors on the error report lines. It does not, however, prevent the node from logging its own errors. This feature is useful for isolating a bus from the rest of the system so that diagnostic software can evaluate the bus without subsequent errors affecting the rest of the system. It should be noted that errors reported on the error report lines will be responded to normally.

MCU-Specific Bits

- Disable Scrubbing ECC Error Report - This bit disables the loading of the array error log or error report log or reporting with scrub ECC errors. It allows other ECC errors to be logged by preventing the log to be filled because of a hard bit failure continually caught by scrub. Note, however, that the normal array log status bits will prevent scrub errors from being reported if the array log is already full.

- Disable ECC Error Reporting - This bit disables the reporting of either correctable or uncorrectable ECC errors. Even when reporting is disabled, errors will continue to be logged in the Array Error Log register.

- Disable MACD Bus FRC Detection - Self-explanatory. It is worth noting that MACD FRC errors are reported as module errors while SLAD FRC errors are reported as unsafe module errors.

- Disable SLAD Bus FRC Detection - Self-explanatory.

- Disable ECC Access Correction - In a read access, this bit disables the data correction when an access is staged. It does not affect error detection or reporting. This will allow the actual data to be read by diagnostic software. It must be noted that ECC error detection may also need to be disabled so that the BIU will not respond to the error report. Since, in a write access, writes are of a read-modify-write format to the array, this bit will prevent the read data from being corrected before the data is written back to the array. Note that this could allow undetected data corruption to the array if the read data had a correctable error: a new ECC code, which matches the word written back to the array, will be generated. If the read data had an uncorrectable error, the MCU would not write to the addressed location (unless the scrub enable bit were off) and therefore it could not corrupt the data in the storage array.

- Enable Scrubbing - This bit enables data correction during refresh cycles. It also ensures that if an uncorrectable error is detected during the read portion of an RMW operation, the write portion will be aborted -- avoiding corruption of data in the storage array. If Enable Scrubbing is off, an uncorrectable error is ignored and the new data is written to the storage array.

- Disable Refresh - This bit totally inhibits refresh cycles from occurring. It is useful for certain types of array testing.

- Continuous Refresh - This bit will continuously cause refresh requests to be made to the internal array state machine. This interrupt request, unlike the refresh interval timeout, has lower request priority than an access request or an interconnect access request.

- Enable Bus State Writeability - This bit enables the bus state bits in the State register to be written. This special enable is provided to safeguard against inadvertantly modifying the bus state bits when writing to the state register, since these bits control the BUSSEL pin on the MCU.

BIU-Specific Bits

- Diagnostics Mode - This bit disables the returning of bus error back to the processor due to a bus timeout for non-N-Local accesses. This allows the diagnostic evaluation of a bus and associated address ranges.

- Bad Access - During interconnect accesses, a permanent module error or a permanent bus error will set this bit. If the Bad Access flag is set, all N-local accesses that are not "my-BIU" accesses will be ignored, and garbage will be returned to the processor. An Interchange command sets the permanent error bit and thus sets the Bad Access bit as well.

ERROR REPORT LOG

This register holds the information from the most recent error report. Only the count and the permanent bit fields are cleared at initialization. Thus, after a crash it may hold information relevant to understanding the cause of the crash. If the node is FRCed, this register should not be read until at least one Interchange command has been sent. The Interchange command is required to set all of the count fields to a known value.

INTERCONNECT DEVICE ID

This register is of interest only in MCUs: the bus ID resides here. If software wants to move an MCU, it toggles the middle bit of the bus ID by sending a Toggle command to the MCU it wishes to move. This register is read only.

INTERCONNECT DEVICE TYPE

Two bits in this register are of interest to the fault handling mechanisms: Redundant Bus Enable and BCHK Enable. These 2 bits are loaded during hardware initialization. This register is read only.

SPOUSE ID

This register holds the physical ID of the module that is married to this module. It is used in setting up a primary/shadow pair of modules. In processor modules, this register must not be written to while the module is married. Reading the register will not always yield the same value since the primary and shadow have different values in the register. The register is initialized to the value zero.

STATE

This register holds all of the relevant state information about the module and bus to which this node is attached. This information may be altered by the hardware as a result of recovery operations. In processor modules, the primary and shadow modules will have different values for the shadow bit. Software must not write to this register during normal operation. This register must be treated as read-only as long as the module is active in the system. Before activating a module, the software must double-check that the bits in the state register are all consistent with the system state (i.e., that software did not overwrite a hardware update). The specific bit definitions follow.

- Married - This bit is used to unify a primary and shadow pair.

- Shadow - This bit specifies whether the module will become a primary or a shadow when the married bit is set. The designation of primary or shadow serves only to determine which of the married pair will participate in the first access when Ping-Ponging is initiated by the setting of the married bit.

- Master/Checker Toggle - This bit is used typically by maintainance software to swap master and checker to evaluate error report line connections and other register information in order to verify the correct operation of the checker. If this bit is zero, then the master/checker state is the same as it was at INIT time. If it is a one, then the master/checker state is opposite the INIT time setting.

- Bus State Bits - These bits specify the state of the primary bus and the back-up bus (if one exists). Both MCUs and BIUs have a set of these bits, as they must both track their primary and back-up buses, when bus switching functions operate.

MCU-Specific Bits

- Four-Way Bus Interleave Mode - This bit is used by the MCU to track BIUs in a system during bus switching. It specifies whether address mapping in the system is functioning with four-way interleaving on bits 6 and 7.

- Force Staging Always - A one in this bit causes all read accesses to be staged and read data to be corrected if there is a correctable error. A zero in this bit will cause read data to be returned uncorrected and unstaged, but will cause errors to be reported. If an error occurs and retry is enabled in the system, then during the retry period read data will be staged and corrected independent of this bit. If this bit is set, no correctable ECC errors will be reported.

- Warm Start INIT Occurred - This bit denotes if the last INIT pulse was a warm INIT, as specified by the state of the MACD11 pin during INIT. A warm INIT will not change the state of the Refresh Address register, Refresh Timer register, the Spare Bit Select register, or the array error log as would a normal, cold INIT.

BIU-Specific Bits

- Disable Retry - This bit disables the retry mechanism after an error report. All accesses that are outstanding after the start of an error report will always be retried if retry is enabled. If this bit is set, accesses that would normally require a retry will possibly return corrupted data. Disable Retry could be used by a diagnostic program to analyze the actual corrupted data or garbage that was received. Disabling retry could also allow the processor to run faster, since reply data will not have to be staged in the BIU.

TIMEOUT DURATION

This register controls the length of three timeouts: the memory bus protocol timeout, the quiescent waiting period, and the permanent error window. All three are always the same length. The length of the timeout can range from approximately 16 microseconds to 2 seconds. The exact formula for the time delay is: (Timeout Duration * 2^16) + 128) * system clock period. The Timeout Duration register is 8 bits long (0 to 255). The register is initialized to zero, which provides the shortest timeout (16 microseconds).

ARRAY LOW/HIGH ORDER ADDRESS - MCU

These registers provide the address used by Interconnect commands that access the memory array. These commands are used in testing the operation of the MCU. At initialization the low register gets the contents of the MACD bus. There are 12 bits that are not defined by the hardware and may be used by the system designer to convey information to initialization software. The high register is initialized to zero.

ARRAY ERROR LOG LOW/HIGH - MCU

These registers hold information about ECC errors that this MCU has detected. After a cold INIT, the error status bits (FBE, Correctable Error, Uncorrectable Error, Additional Error, and Scrub Error) are cleared. In an FRCed system, an ECC error must be generated (to update the fields this register) before it may be read from the master/checker.

SPARE BIT SELECT - MCU

This register controls the spare bit multiplexing in the MCU. Values between 0 and 38 provide for the replacement of an array bit by the spare bit. All other values have no affect on the array operation. The register is initialized to all ones.

LOGICAL ID - BIU

This register holds the ID that will always be used during arbitration. The logical ID replaces the module ID for interconnect register accesses when a processor module is married. The value in this register must be unique in the system. All of the BIUs in one module must have the same logical ID. At initialization, this register is loaded with a reverse image of the module ID. (MSB of the module ID is the LSB of logical ID, LSB of module ID is the MSB of logical ID, etc.) This provides a good arbitration distribution in sequentially numbered modules.

TEST DETECTION DATA REGISTER - BIU

This register is used to provide the read data during Test Detection commands. This register may be used as a general-purpose scratch register by software. At initialization this register gets the contents of the MACD bus. There are 12 bits that are not defined by the hardware and may be used by the system designer to convey information to initialization software.

WORKING/MERL - BIU

This register is used to monitor the operation of the processor and the module's MERL line. This register is initialized to zero.

COMMANDS

Commands use the interconnect register address space. Instead of accessing a register, they cause a function to occur within the MCU or BIU. Commands still have a read or write specification. All commands that generate error reports cause the full recovery sequence, just like a true error. The commands that generate error reports are acknowledged before the error report and, thus, they will not be retried.

BUS INTERCHANGE [WRITE]

This command interchanges the address ranges of all the bus pairs in the system. It also resets the count fields in the error logs and sets the permanent bit. This command is used to test the operation of the back-up address recognition and bus switching logic. The command is sent to any node in the system. If a bus is not paired, then no bus switching will occur. This command generates an error report message with error type interchange.

DETACH [WRITE]

This command allows a bus to be deallocated from the system while the system is on-line. The command is sent to any node on the bus that is to be deallocated. This command generates an error report message with error type detach.

TEST DETECTION [READ]

This command tests all of the detection mechanisms in the addressed node. All detection mechanisms are tested independent of the state of the detection enables. This command generates an error report message. If there is a failure in the detection mechanisms, then the report will have a module error type. If no failures occur, then the error type will be no error.

TEST REPORT [WRITE]

This command is used to test a specific component's ability to generate error report messages. The data field in the command specifies which component will respond to the command (0 - primary/checker, 1 - primary/master, 2 - shadow/checker, 3 - shadow/master). Only values zero and one are valid on the MCU. This command generates an error report message with error type TEST REPORT.

ACCESS DATA LOW/HIGH [READ/WRITE] - MCU

These two commands are provided to allow software direct access to the 32 data bits in the array. The location accessed is determined by the low/high order array address registers.

ACCESS ECC BITS [READ/WRITE] - MCU

This command is provided to allow software direct access to the ECC bits in the array. The location accessed is determined by the low/high order array address registers. This command allows direct access to the ECC bits in the memory array. It will be used for memory testing and diagnostics.

CLEAR LOCATION [WRITE] - MCU

This command allows a location with bad ECC to be initialized to a zero data value with good ECC. The location accessed is determined by the low/high order array address registers.

CLEAR MEMORY [WRITE] - MCU

This command is identical to Clear Location, except it clears the entire array. A reply is sent before the MCU has completed the command. The MCU will remain busy until it has completed this command. The programmer must beware that bus timeouts might occur if memory accesses are issued before the MCU has completed the Clear Memory command.

TOGGLE PRIMARY AND BACK-UP BUS ID [WRITE] - MCU

This command is used to assign a memory module to its alternate bus. This is different from a bus switch. The MCU will identify the new bus as its primary bus. This is useful during initialization or during software recovery from a permanent error. This command generates an error report message with an error type of no error. An error report is issued in order to allow the MCU to become synchronized with the arbitration on its new bus.

CLEAR PU [WRITE] - BIU

This command is used to synchronize the processor (GDP or IP) components when two modules are being married. This operation is required because the processors have a two-cycle idle loop. Thus, idle processors are not guaranteed to be in lock step.

SOFTWARE SUPPORT

There are two basic levels of software support required by the hardware mechanisms. At one level the software is providing management and policy decisions. On the second level the software is completing fault-tolerant functionality by providing capabilities that could not be placed in the hardware. This section does not describe any of the general system support software that may also be present (e.g., system confidence, off-line diagnostics). This view of software is from the bottom, looking up. The software may be implemented in modules that are organized differently, based on a top-down, system capability view.

Management software includes:

- Configuration and graceful degradation policy. This software is responsible for establishing the system configurations that most closely match the needs of the application. This includes reconfiguration decisions made after a permanent error as well as the normal initialization decisions. This is the software that marries modules and establishes back-up buses.

- System health monitor. This software provides a long-term record of failures in the system. This information may be used by other software modules or by the field maintenance team.

Low-level software includes:

- On-line testing for latent errors in the system. This software completes the functionality of the hardware detection mechanisms. This software exercises the detection and recovery mechanisms as well as the infrequently used operations in the processors. In this way, latent errors are exposed and configured out of the system before the system is placed in a potentially dangerous double-error situation.

- The process that waits at the reconfiguration dispatching port. This module completes the process of sending a message about system reconfiguration to the appropriate policy software. This software may also make some preprogrammed configuration decisions. This is a very small module, but it must run in an environment that does not have normal operating system support.

- Review the permanent/transient decisions made by the hardware. Using information from the system health log, this software may act to reverse decisions made by the hardware. This software can use information not available to the hardware. For instance, information about the environment of the system, application needs, and long-term error history may all be used by this s ftware to optimize system operation.

- On-line diagnostics which isolate errors in a bus confinement area to a replaceable module. The bus confinement area is the only confinement area that does not map nicely with a repair area. Diagnostic probing may be required to isolate the failure to a replaceable module. In the worst case, isolating the repairable unit may require physical extraction of modules from the bus.

Figure 10-1 provides a general diagram of the relationship between the hardware mechanisms and the support software.

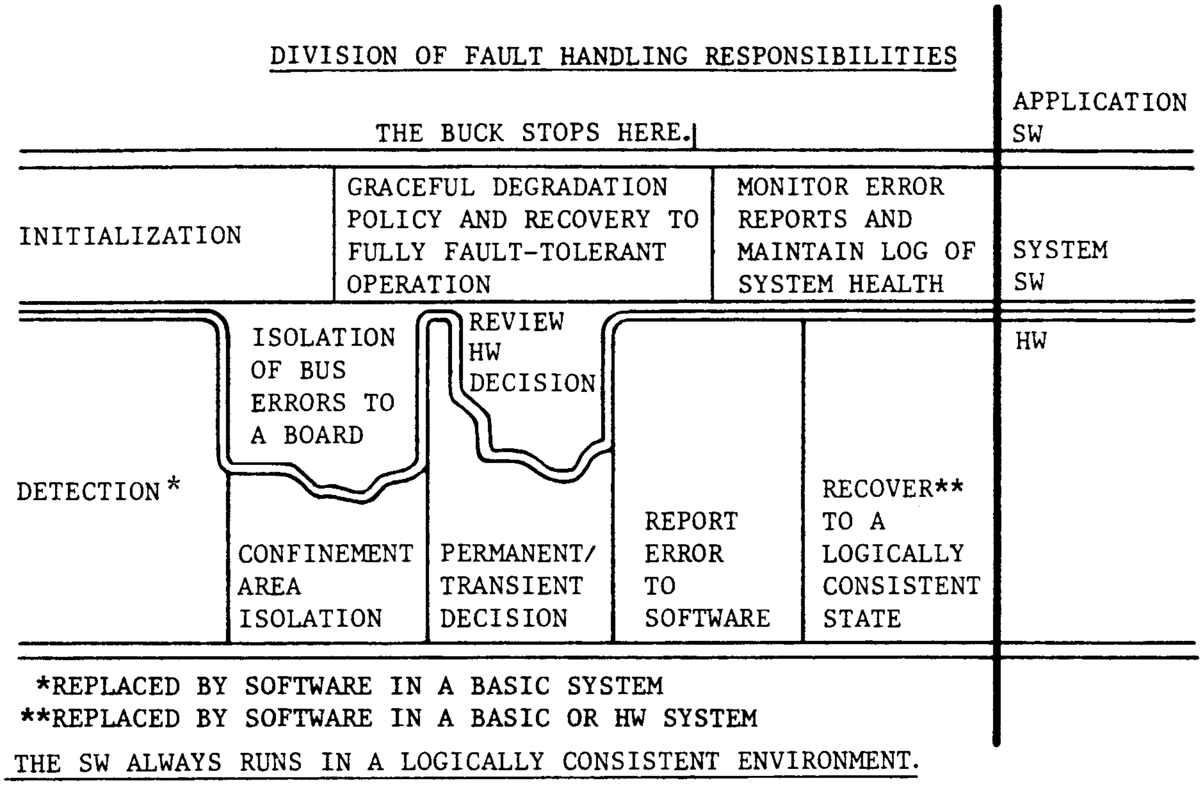

DIVISION OF FAULT HANDLING RESPONSIBILITIES
THE BUCK STOPS HERE.
APPLICATION SW
INITIALIZATION
GRACEFUL DEGRADATION POLICY AND RECOVERY TO FULLY FAULT-TOLERANT OPERATION
MONITOR ERROR REPORTS AND MAINTAIN LOG OF SYSTEM HEALTH
SYSTEM SW
ISOLATION OF BUS ERRORS TO A BOARD
REVIEW HW DECISION
HW
DETECTION*
CONFINEMENT AREA ISOLATION
PERMANENT/ TRANSIENT DECISION
REPORT ERROR TO SOFTWARE
RECOVER** TO A LOGICALLY CONSISTENT STATE
*REPLACED BY SOFTWARE IN A BASIC SYSTEM
**REPLACED BY SOFTWARE IN A BASIC OR HW SYSTEM
THE SW ALWAYS RUNS IN A LOGICALLY CONSISTENT ENVIRONMENT.

Figure 10-1. Software Support

F-0459

APPENDIX A
iAPX 43204/43205 REGISTER SUMMARY

INTRODUCTION

Each iAPX 43204 Bus Interface Unit (BIU) and each iAPX 43205 Memory Control Unit (MCU) occupies a 32 double-byte register location of iAPX 432 interconnect address space. This address space is accessed by way of the iAPX 432 instructions Move to Interconnect and Move from Interconnect. Each register is a double-byte quantity aligned to a double-byte boundary. Each register has a name and a hexadecimal register number; e.g., the Interconnect Device Type register is MCU Register 01. See Appendix E, "Address Formats," for more information on how interconnect registers are addressed.

The BIU supports two registers dedicated to the specific processor to which it is attached. These two registers are normally accessed only by the iAPX 432 processor through two double-byte registers located in the processor-local, or P-local, address space. One register contains the processor ID (P-local byte address 0). The other contains the interprocessor communication interface (P-local byte address 2).

The similarities between the register sets of the BIU and MCU may be observed from the side-by-side register summaries show in Tables A-1 and A-2. Appendixes B and C provide more detail about the individual registers.

Table A-1. iAPX 43204 Bus Interface Unit Register Summary

N-local Address	P-local Address	BIU Register	Page
1F		Attach Command	B-29
1E		Detach Command	B-28
1D		Bus Interchange Command	B-27
1C		Test Report Command	B-26
1B		Test Detection Command	B-25
1A		Activity	B-24
19		***RESERVED***	
18		Clear Processor Unit Command	B-23
17		***RESERVED***	
16		***RESERVED***	
15		***RESERVED***	
14		***RESERVED***	
13		***RESERVED***	
12		***RESERVED***	
11		***RESERVED***	
10		***RESERVED***	
0F		***RESERVED***	
0E		***RESERVED***	
0D		Interleave	B-22
0C	01	Interprocessor Communication	B-20
0B		Test Detection Data	B-19
0A	00	Processor ID	B-18
09		Memory Start and End	B-17
08		Timeout Duration	B-16
07		Diagnostic	B-14
06		State	B-11
05		Spouse ID	B-10
04		Logical ID	B-9
03		***RESERVED***	
02		Interconnect Device ID	B-8
01		Interconnect Device Type	B-7
00		Error Report Log	B-3

Table A-2. iAPX 43205 Memory Control Unit Register Summary

N-local Address	MCU Register	Page
1F	Attach Command	C-39
1E	Detach Command	C-38
1D	Bus Interchange Command	C-37
1C	Test Report Command	C-36
1B	Test Detection Command	C-35
1A	***RESERVED***	
19	Clear Location Command	C-34
18	Clear Memory Command	C-33
17	Toggle Normal and Back-up Bus Command	C-32
16	Access ECC Bits Command	C-31
15	Access Data High Command	C-30
14	Access Data Low Command	C-29
13	***RESERVED***	
12	Refresh Address Decrementer	C-28
11	Array Address Size	C-27
10	Refresh Interval Decrementer	C-26
0F	Refresh Interval	C-25
0E	Array Error Log High	C-23
0D	Array Error Log Low	C-22
0C	Array High-Order Address	C-21
0B	Array Low-Order Address	C-20
0A	Spare Bit Select	C-19
09	Memory Start and End	C-18
08	Timeout Duration	C-17
07	Diagnostic	C-15
06	State	C-12
05	Spouse ID	C-11
04	***RESERVED***	
03	***RESERVED***	
02	Interconnect Device ID	C-10
01	Interconnect Device Type	C-7
00	Error Report Log	C-3

APPENDIX B
iAPX 43204 BIU REGISTERS

INTRODUCTION

Appendix B describes the set of Interconnect registers supported by each iAPX 43204 Bus Interface Unit (BIU) component. Some of the BIU registers provide access to the data that the BIU provides or requires. BIU commands are intiated by interconnect access to other registers.

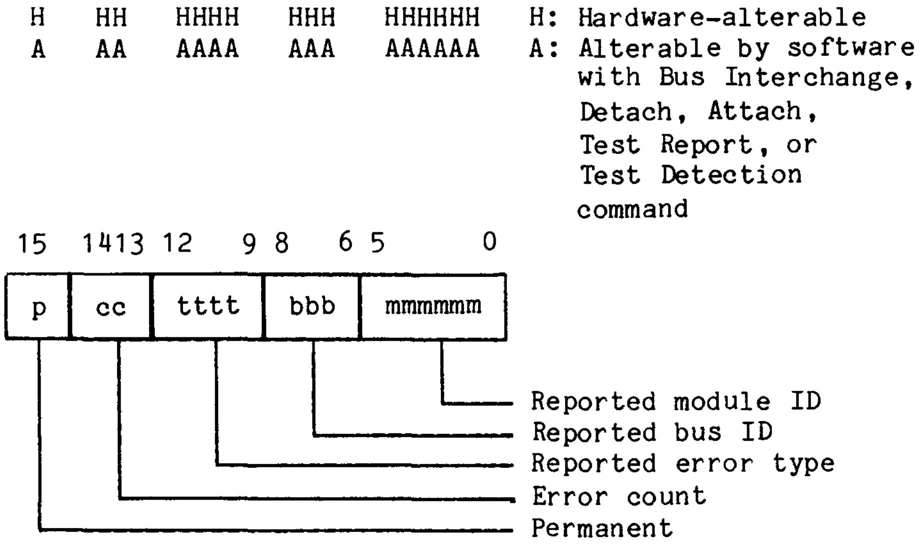
Many of the registers have fields (a bit or a group of bits) that are labeled "reserved." Reserved fields may not be modified. The value of the information read from reserved fields is undefined and should be masked off before any comparisons are performed. BIUs always return consistent values in reserved fields (zeroes in the current implementation) to prevent errors in systems employing Functional Redundancy Checking (FRC).

A summary of the BIU interconnect registers is provided on page B-2. Following the summary, shown in Table B-1, each register is described individually.

Table B-1. BIU Register Summary

N-local Address	P-local Address	BIU Register	Page
1F		Attach Command	B-29
1E		Detach Command	B-28
1D		Bus Interchange Command	B-27
1C		Test Report Command	B-26
1B		Test Detection Command	B-25
1A		Activity	B-24
19		***RESERVED***	
18		Clear Processor Unit Command	B-23
17		***RESERVED***	
16		***RESERVED***	
15		***RESERVED***	
14		***RESERVED***	
13		***RESERVED***	
12		***RESERVED***	
11		***RESERVED***	
10		***RESERVED***	
0F		***RESERVED***	
0E		***RESERVED***	
0D		Interleave	B-22
0C	02	Interprocessor Communication	B-20
0B		Test Detection Data	B-19
0A	00	Processor ID	B-18
09		Memory Start and End	B-17
08		Timeout Duration	B-16
07		Diagnostic	B-14
06		State	B-11
05		Spouse ID	B-10
04		Logical ID	B-9
03		***RESERVED***	
02		Interconnect Device ID	B-8
01		Interconnect Device Type	B-7
00		Error Report Log	B-3

BIU Register 00
Error Report Log



The mmmmmm field contains the module ID of the reporting source.

The bbb field contains the bus ID of the reporting source.

The tttt field contains the type code for the reported error. The error types and encodings are summarized in Table B-2.

Table B-2. BIU Error Report Types

tttt Code	Message Type	BIU Error
1111	BERL Parity	BERL Error
1110	MERL Parity	MERL Error
1101	Unsafe Module	n/a
1100	Bus Arbitration	Bus Arbitration
1011	Bus/Module-High	n/a
1010	No Error	Command Report
1001	Module	Module
1000	Bus Parity	Bus Parity Error
0111	Bus/Module-Low	Broken Module
0110	Uncorrectable ECC	n/a
0101	Correctable ECC	n/a
0100	Test Report	Command Report
0011	Interchange	Command Report
0010	Attach	Command Report
0001	Detach	Command Report
0000	Toggle	n/a

The cc field contains the value of a counter which increments each time an error report is logged by the BIU. If the cc counter reaches the maximum count of three, it will hold the value three until it is explicitly cleared. These bits may be used by software to track accesses to the error report logs. The cc counter is cleared during initialization and by the Bus Interchange command.

The p field indicates whether an error is permanent (p=1) or transient (p=0). This information is used to determine if the BIU must switch to a new bus or module and whether the BIU must enter broken bus or broken module mode. The first report of any error is labeled transient (p=0). If a second report of the same error (e.g., the second error is of the same type and from the same location -- reported module ID -- as the first error occurs during the transient error window), the error is labeled permanent (p=1). The p field is set to transient (p=0) during initialization.

Table B-3 indicates how a BIU will respond to permanent and transient errors for each of the types of errors.

Table B-3. BIU Error Recovery Mechanisms

Code	Message Type	Recovery Action	
		Transient (p=0)	Permanent (p=1)
1111	BERL Parity	Retry	Bus Switch
1110	MERL Parity	Retry	Module Switch
1101	Unsafe Module	(Note 5)	Module Switch
1100	Bus Arbitration	Retry	Bus Switch
1011	Bus/Module-High	Retry	Bus/Module Switch
1010	No Error	None	n/a
1001	Module	Retry	Module Switch
1000	Bus Parity	Retry	Bus Switch
0111	Bus/Module-Low	Retry	Bus/Module Switch
0110	Uncorrectable ECC	Retry/Stage	Module Switch
0101	Correctable ECC	Retry/Stage	n/a
0100	Test Report	None	n/a
0011	Interchange	n/a	Bus Exchange
0010	Attach	Attach Bus	n/a
0001	Detach	Detach Bus	n/a
0000	Toggle	None	n/a

Note that:

- Any error report will cause all outstanding requests to retry, and all memories will stage data for a reply. "Retry," "None," or "Stage" indicate the mechanism required, since all three will function after an error report.

- All bus switches are followed by a retry and transient error window.

- Any error reports during a transient error window will be classified as permanent or transient based on the last reported error.

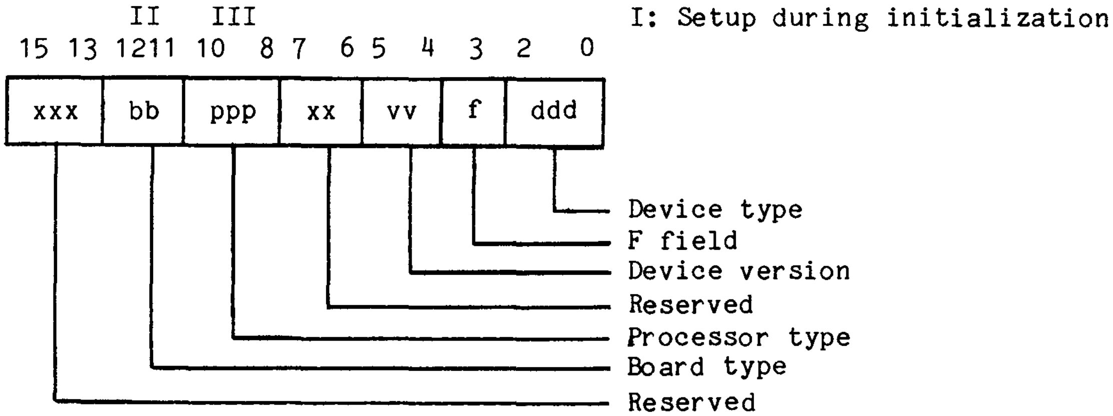
- The bus switch or module switch will occur only if the appropriate interconnect register fields are enabled.

- An UNSAFE MODULE report is always considered a permanent error.

SPECIAL CONSIDERATIONS

At initialization time, the cc field (error count) and the p field (permanent) are set to zero. The tttt field (reported error type), the bbb field (reported bus ID), and the mmmmmm field (reported module ID) are not modified by the initalization so that error information will be available after a system crash. After a cold-start initialization in a system using FRC (Functional Redundancy Checking), the error log should not be read until a Bus Interchange command has been issued. Otherwise, the master and checker nodes may have different values in their error report logs and thereby cause an FRC error.

BIU Register 01
Interconnect Device Type



The ddd, f, and vv fields of the Interconnect Device Type register are fixed within the BIU at the time of manufacture.

The ddd field contains the interconnect device type. For a BIU, the value of the ddd field is 001.

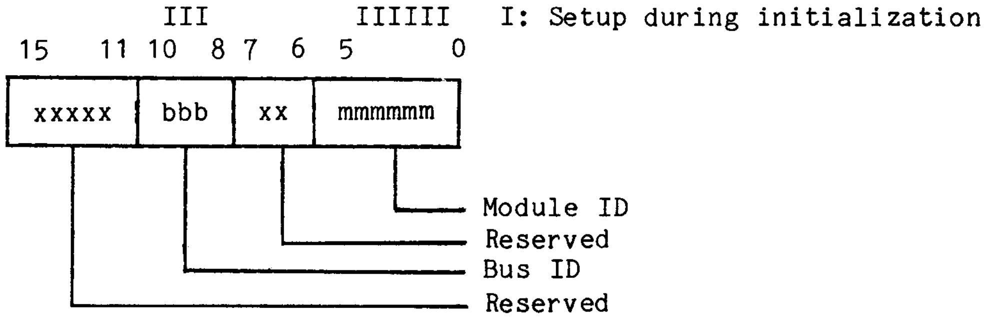
The f field always contains the value one (1).

The vv field contains the device version number for the BIU.

The ppp and bb fields of the Interconnect Device Type register are loaded from the ACD (processor Address, Control, and Data) bus when the BIU is initialized. The ppp field is loaded with the value of ACD5...ACD3 when the BIU is initialized. The bb field is loaded with the value of ACD7..ACD6 when the BIU is initialized. The ppp field indicates the type of the processor to which the BIU is connected. General Data Processors (GDPs) formed from iAPX 43201/43202 components are indicated with type ppp=001. Interface Processors (IPs) formed from iAPX 43203 components are indicated with type ppp=010. The ppp field is not interpreted by the BIU but may be read by initialization software to determine the type of processor resources in the system.

The bb field indicates the type of board on which the BIU resides. The least significant bit of the bb field (b0) indicates if the BIU is to enable (b0=1) or disable (b0=0) buffer checking. The most significant bit of the bb field (b1) indicates if a redundant bus is available (b1=1). The bb field is loaded with the value of ACD7...ACD6 when the BIU is initialized.

BIU Register 02
Interconnect Device ID

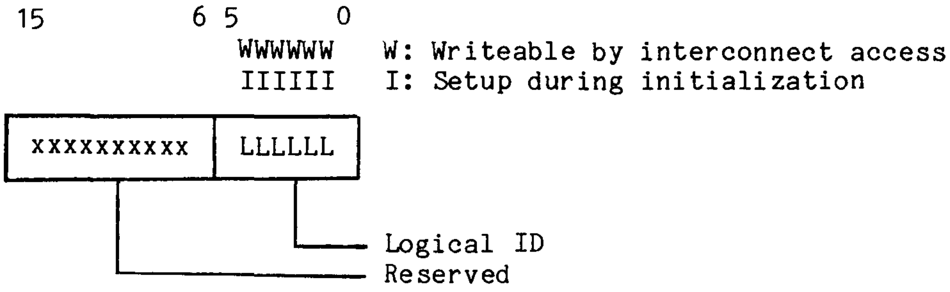


The mmmmmm field contains the module ID. These bits are loaded from the processor ACD bus, bits ACD13...ACD8, during initialization. The initialized module ID is identical to the value of the least significant 6 bits of the initialized processor ID (BIU Register 0A). The mmmmmm field defines a unique module position across all of the memory buses. The module ID field is read-only. The BIU uses its module ID when it decides whether to participate in interconnect accesses.

During initialization, the BIU loads an internal flip-flop with a bit (from the processor bus ACD1 bit position) that corresponds to even parity across the mmmmmm field. This flip-flop is not visible by interconnect access. If the BIU detects that the parity across the module ID (mmmmmm field) and flip-flop is not even, then the BIU will permanently assert INIT within the chip. By reporting errors with an incorrect module ID this parity mechanism prevents a module with a corrupt module ID from causing collapse of the system. Externally, a BIU ceases to toggle the BUFCHK output when the internal INIT signal is asserted.

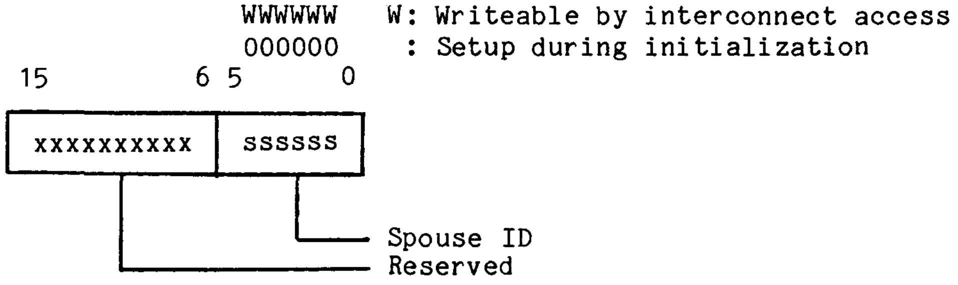
The bbb field contains the bus ID code, which indicates the bus to which the BIU is attached. In configurations that support bus switching, the back-up bus ID is formed by complementing the middle bit of the bbb field.

BIU Register 04
Logical ID


15
6 5
0
WWWWWW
IIIIII
W: Writeable by interconnect access
I: Setup during initialization
xxxxxxxxxx
LLLLLL
Logical ID
Reserved

The LLLLLL field contains the logical ID for the BIU. When the BIU is initialized, the logical ID is loaded with the value of the module ID from the Interconnect Device ID register (BIU Register 02). However, the significance of the module ID bits is reversed during the initializing load of the logical ID field. This reassignment provides one type of load balancing since the BIU uses the value of the logical ID field when arbitrating for use of the MACD bus. When it is performing shadowed operation (with its married bit set in the State register) the BIU uses the value of the logical ID field (rather than its module ID) to decide when to participate in interconnect accesses.

BIU Register 05
Spouse ID

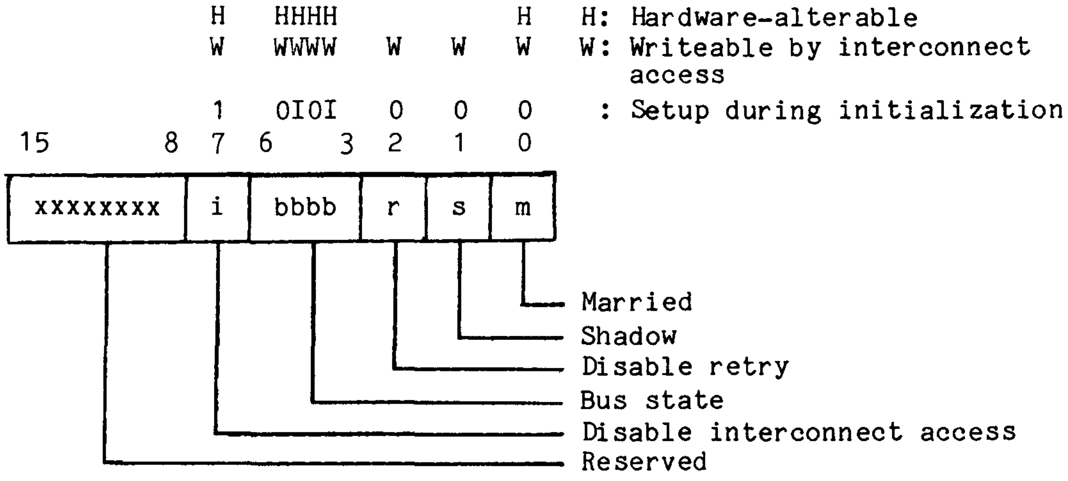


W: Writeable by interconnect access
: Setup during initialization

The ssssss field contains the module ID for the spouse (other partner) in the primary/shadow pair of modules for systems employing shadowing. The ssssss field is initialized to 0000000, and should be modified to the module ID of the actual spouse before the married bit is set in the State register (BIU Register 06). Once a BIU is married to its spouse, this register must be treated as read-only. Furthermore, interconnect reads of this register will return different values, since the married pair is performing Ping-Pong operation and since the spouse IDs of the primary and shadow are different.

The value of the ssssss field is matched against the reported module ID of an error report to determine if its spouse has an error. If the spouse reports a permanent module error, this BIU divorces itself (clears the married bit in the State register, BIU Register 06) and assumes responsibility for all future activities of the module.

BIU Register 06
State



The m field contains the married bit, which indicates whether this BIU is married to another BIU (m=1) to form a primary/shadow pair. Before the m bit is set, the spouse ID (BIU Register 04) must be configured with the module ID of the BIU which is the spouse. When a BIU is married, it performs in the following manner.

- The married BIU checks error reports of permanent module errors to determine if the module ID of the reporter is the same as that of its spouse. When the BIU detects that a permanent module error has occurred in its spouse, the BIU will reset its married bit.

- The BIU and its married spouse begin Ping-Pong operation in which the partners alternately service the interconnect and memory accesses requested by the processor. The designated primary BIU will service the first access request.

- When married, BIUs respond to interconnect accesses based on the value of their logical ID rather than their module ID. Since the logical IDs of the married BIUs must be identical, the primary and shadow BIUs alternately service interconnect access requests.

- The activity between a married pair of BIUs can be synchronized with the Clear Processor Unit command.

The s field indicates whether the BIU will become a shadow (s=1) or a primary (s=0) unit when the married bit is set.

The r field indicates whether the BIU will disable (r=1) or enable (r=0) the retrying of accesses after an error report. All accesses that are outstanding after the start of an error report will be retried if retry is enabled (r=0). When retry is disabled (r=1) the BIU will not stage data. The data received by means of the memory bus, possibly incorrect, is immediately presented to the processor without awaiting a possible error report.

The bbbb field contains the bus state code. The individual bits of the bbbb field have the following names:

- Bit 0 - Enable Normal (EN)

- Bit 1 - Enable Backup (EB)

- Bit 2 - Backup's Enable Normal (BEN)

- Bit 3 - Backup's Enable Backup (BEB)

EN and BEN are loaded with the value of ACD7 (enable redundant bus) when the BIU is initialized. Thus, a BIU without a redundant bus is initialized to the value bbbb=0000, and a BIU with a redundant bus is initialized to the value bbbb=0101. The seven valid combinations of the bbbb field are indicated in Table B-4.

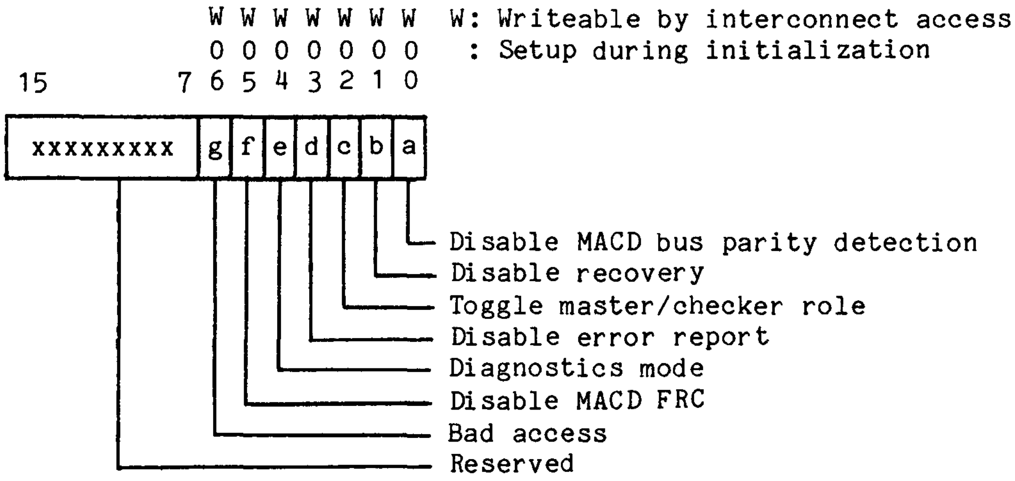
Table B-4. Bus State Field Encoding

bbbb Field				State	Interpretation
BEB	BEN	EB	EN		
0	0	0	0	0	Null State
0	0	0	1	1	This node's normal bus is up.
0	1	0	0	2	This node's back-up bus is up.
0	1	0	1	3	Normal four-way Address Interleaving mode.
1	0	1	0	4	Interchanged four-way Address Interleaving.
1	1	0	0	5	Normal bus is down.
0	0	1	1	6	Back-up bus is down.

In states 2, 4 and 5, a BIU will operate on its back-up bus. In all other states, a BIU will operate on its normal bus. In state 0, the null state, the BIU will not use the memory bus for memory accesses, although interconnect accesses are allowed. The response of a BIU to the commands Attach, Detach, and Bus Interchange depends on the bus state and the system configuration (four-way, redundant bus). The allowed state transitions of the bus state field are described in Appendix F.

The i field determines whether the BIU is disabled (i=1) or enabled (i=0) from generating access requests on the memory bus to either the memory or the interconnect address spaces. When accesses are disabled (i=1) the BIU also disables the MERL (Module ERror Line) receiver, thereby ignoring any MERL-type stuck faults. The i field is reset on permanent module failures.

BIU Register 07
Diagnostic



The a field determines if detection of MACD bus parity errors is to be disabled (a=1) or enabled (a=0).

The b field determines if the BIU is to disable (b=1) or enable (b=0) its fault recovery mechanisms. If recovery is disabled, the BIU will continue to detect, report, and log errors, but automatic bus switching is not allowed (except for the explicit Interchange command).

The value of the c field allows software to modify the master/checker relationship from the one set on initialization. The c field determines whether the BIU should operate with a reversed (c=1) or assigned (c=0) FRC master/checker role. For example, a BIU which was assigned on initialization to act as a master FRC device can be commanded to reverse roles (c=1) and operate as a checker. This capability allows software to swap master and checker BIUs to verify correct operation of the original checker.

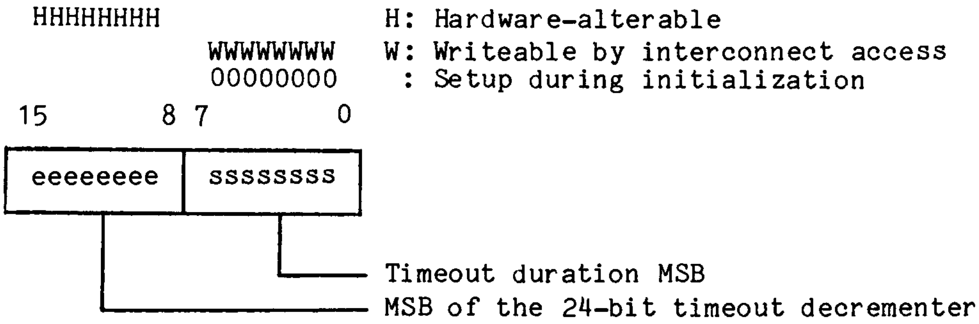
The d field determines whether the BIU disables (d=1) or enables (d=0) error reporting. When error reporting is disabled, the BIU will continue to log its own errors and receive error reports from other agents.

The e field determines whether the BIU enables (e=1) or disables (e=0) diagnostic mode. In diagnostic mode, the BIU does not report bus errors caused by the timeout of non-N-local accesses. Thus, the diagnostic mode permits the evaluation of a memory bus and its associated address ranges.

The f field determines whether the BIU disables (f=1) or enables (f=0) the detection of FRC errors on the MACD bus.

The g field indicates if a bad access (g=1) has occurred. A bad access is detected when a permanent module error or a permanent bus error occurs during an interconnect access. When the bad access flag is set (g=1), all general interconnect access to the BIU is denied, interconnect access reads will return indeterminate data, but the BIU will continue to respond to "my-BIU" interconnect accesses.

BIU Register 08
Timeout Duration


HHHHHHHH
WWWWWWWW
00000000
H: Hardware-alterable
W: Writeable by interconnect access
: Setup during initialization
15
8
7
0
eeeeeeee
ssssssss
Timeout duration MSB
MSB of the 24-bit timeout decrementer

A timeout decrementer in the BIU is used to regulate the bus timeouts and the transient error window. When a timing function begins, the BIU loads the 24-bit timeout decremeter. The value from the timeout duration field is loaded into the most significant 8 bits of the decrementer, and the value 128 is loaded into the least significant 16 bits. As the decrementer operates (once each component clock cycle), the value of the 8 most significant bits may be read from the eeeeeeee field of the timeout duration register.

During initialization, the timeout duration MSB is set to the value zero. Thus, the shortest timeout (128 clock cycles) will be employed during the period when the system configuration is being determined. Once configuration is complete, the timeout duration must be lengthened so that normal accesses may complete before the decremeter has expired.

BIU Register 09
Memory Start and End

```
 WWWWWWWW  WWWWWWWW   W: Writeable by interconnect access
 00000000  00000000    : Setup by initialization
15       8 7       0
+--------+--------+
|eeeeeeee|ssssssss|
+--------+--------+
     |        |
     |        +------ Memory start address
     +--------------- Memory end address
```

The memory start address and the memory end address define the range on the most significant 8 bits of the 24-bit physical memory address, which the BIU will recognize as a memory access request. An iAPX 432 processor issues the physical memory addresses to the BIU by means of the ACD bus.

The first physical memory address supported by the BIU is defined by the 24-bit address:

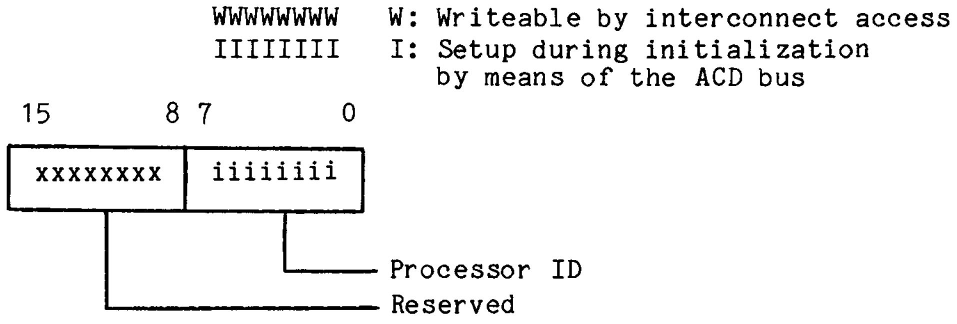
```
23      16 15               0
+--------+----------------+
|ssssssss|0000000000000000|   BIU starting physical address
+--------+----------------+
```

The last physical memory address supported by the BIU is defined by the 24-bit address:

```
23      16 15               0
+--------+----------------+
|eeeeeeee|1111111111111111|   BIU ending physical address
+--------+----------------+
```

Thus, if the memory start and end addresses are identical, the BIU supports 64K bytes of memory.

BIU Register 0A Processor ID


WWWWWWWW
IIIIIIII
W: Writeable by interconnect access
I: Setup during initialization by means of the ACD bus
15
8
7
0
xxxxxxxx
iiiiiiii
Processor ID
Reserved

The processor ID register contains the unique 8-bit number that identifies the processor connected to the BIU. The BIU uses the processor ID to acquire the interprocessor communication (IPC) messages with a matching ID which are carried on the memory bus. This ID also is accessed by the iAPX 432 processor to locate objects unique to the particular processor (e.g., processor objects).

The processor ID field is loaded with the value of ACD15...ACD8 when the BIU is initialized. The processor ID of zero (00000000) is reserved and indicates the broadcast address when an interprocessor communication is sent to all processors.

BIU Register 0B
Test Detection Data

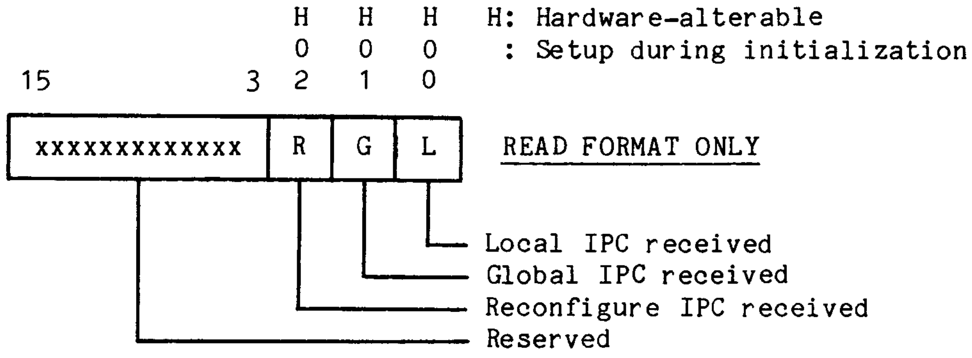
```
 WWWWWWWWWWWWWWWW   W: Writeable by interconnect access
   MACD15..MACD0     : Setup during initialization
15                0
+-----------------+
|xxxxxxxxxxxxxxxx |
+-----------------+
          |
          +---------- Test detection data
```

The Test Detection Data register supplies the data returned in response to the Test Detection command. Diagnostic software can use this register to generate the data required to test the MACD data paths, one at a time. The register may also be used as a general-purpose 16-bit work register. The Test Detection Data register is loaded with the value on the MACD bus, MACD15...MACD0, when the BIU is initialized, allowing customer-defined information to be captured.

BIU Register 0C
Interprocessor Communication

Note that:

The format of the Interprocessor Communication register is different for read and write accesses. In normal system operation this register must be accessed only by an iAPX 432 processor by means of its P-local address space. Interconnect access to this register is provided for testing purposes and off-line diagnostics.

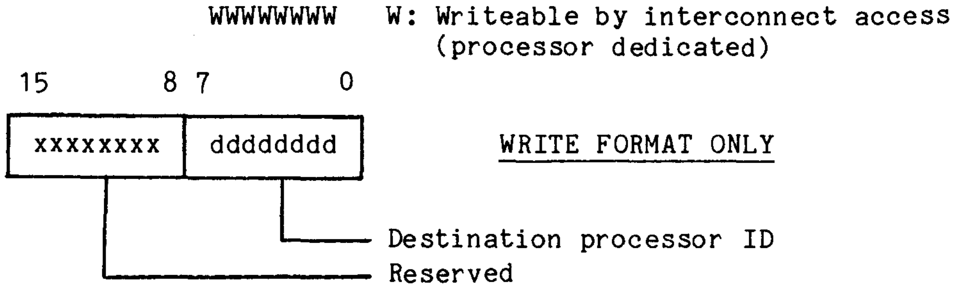


The R field indicates that the BIU has received a reconfigure IPC message (R=1) for the iAPX 432 processor.

The G field indicates that the BIU has received a global IPC message (G=1).

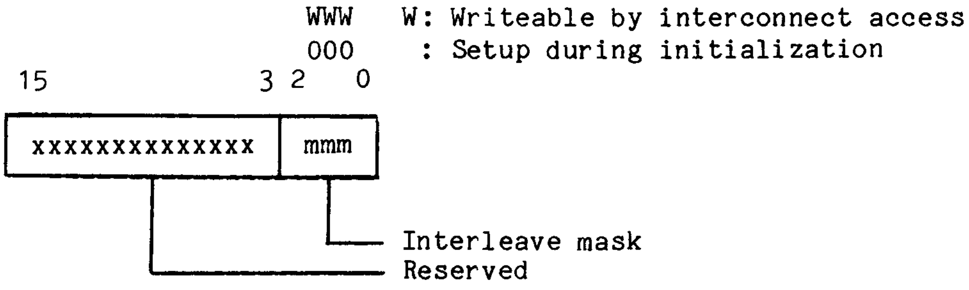
The L field indicates that the BIU has received a local IPC message (L=1) in which the destination ID matched the processor ID in BIU Register 0A.

The iAPX 432 processor will be signaled if any of the 3 IPC bits has been set. As the processor responds, by reading the Interprocessor Communication register, the most significant active bit will be cleared. This implies that reconfigure IPCs have highest priority, and local IPCs have the lowest priority. The IPC register must be read once for each active IPC indicator. The BIU will notify the iAPX 432 processor of an IPC arrival once for each active IPC indicator.



An iAPX 432 processor writes to the Interprocessor Communication register to send the notification of an IPC to the destination processor(s) indicated by the dddddddd field. The destination ID of 00000000 is reserved to indicate that all processors are to receive the IPC. Thus, no processor may be given a processor ID of zero. This information is provided by an iAPX 432 processor when the Send to Processor instruction or the Broadcast to Processors instruction is executed, and is not normally written otherwise.

BIU Register OD
Interleave


WWW
000
W: Writeable by interconnect access
: Setup during initialization
15
3 2
0
xxxxxxxxxxxxxx
mmm
Interleave mask
Reserved

The mmm field determines the type of address interleaving that the BIU is to perform. The values of the mmm field and the corresponding interleaving action is summarized in Table B-5. The reordered address column indicates how the BIU will reorder bit positions in the 24-bit physical memory address from the processor to prepare the address for presentation on the memory bus.

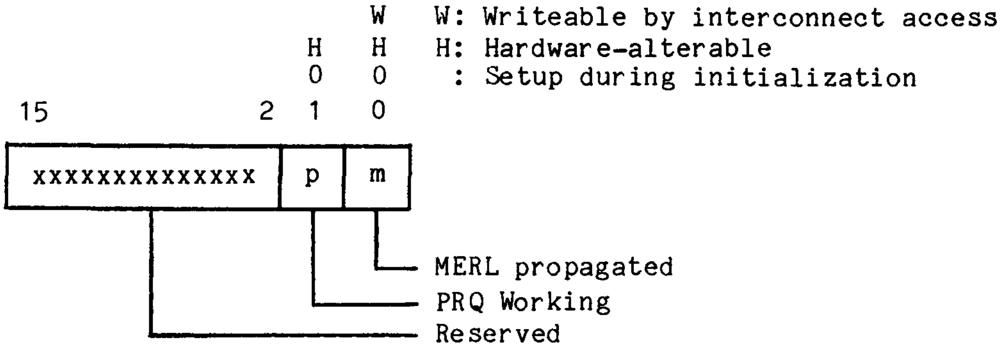
Table B-5. Reordered Addresses with Interleaving

mmm	Interleaving Action	Reordered Address
000	No Interleaving	23...0
001	Interleaving on Address Bit 6	6,23..7,5..0
010	Interleaving on Address Bit 7	7,23..8,6..0
011	Interleaving on Address Bits 6 and 7	7..6,23..8,5..0
100	RESERVED	
101	Interleaving on Address Bits 6 and 8	8,6,23..9,7,5..0
110	Interleaving on Address Bits 7 and 8	8..7,23..9,6..0
111	Interleaving on Address Bits 6, 7, and 8	8..6,23..9,5..0

BIU Register 18
Clear Processor Unit

An interconnect access write to BIU Register 18 will cause the BIU to execute the Clear Processor Unit command. This command is used to synchronize the operation of the primary and shadow BIUs in systems that employ shadowing. When this command is issued, the primary and shadow BIUs must have the same logical ID (BIU Register 04), and the married bit must be set in the State register (BIU Register 06).

BIU Register 1A
Activity



An interconnect access read of the Activity register (BIU Register 1A) provides 2 bits of information about activity of the BIU. The p and m fields are cleared during initialization.

The m field toggles each time the BIU has successfully propagated a MERL error report.

The p field is set each time the processor activates the PRQ signal (Processor Request) for the BIU. The p field can be cleared by an interconnect access write to this register with the p field set to zero.

BIU Register 1B
Test Detection Command

An interconnect access read of BIU Register 1B commands the BIU to test all its error detection circuits: parity, buffer checking, and FRC. The Test Detection command will execute whether error detection is enabled. The BIU performs the command by executing the following sequence of steps.

When the command is received, the BIU requests an interconnect access read of its own Test Detection Data register (BIU Register 0B) by means of the memory bus. The BIU responds to the memory bus request by placing the Test Detection Data register data on the memory bus. Within the BIU, every FRC circuit, MACD bus parity detector, and buffer check circuit is provided the complement of the data sent to the memory bus. The data in the Test Detection register is also issued to the processor to complete the original interconnect access read request. The BIU checks that each detection comparator on the memory bus detects an error. If any comparator fails to generate an error, the BIU will issue a MODULE ERROR report. If each comparator operates correctly, the BIU will issue a NO ERROR report.

Note that the data path of the memory bus may be tested one bit at a time by issuing multiple Test Detection commands and appropriately varying the data in the Test Detection Data register.

The Test Detection command cannot be issued to a "my-BIU" interconnect address (module ID=111111) since "my-BIU" accesses are serviced without the use of the memory bus.

BIU Register 1C
Test Report Command

An interconnect access write to BIU Register 1C commands the BIU to test the MERL (Module ERror Line) and BERL (Bus ERror Line) error reporting paths. The BIU returns a memory bus reply message before the command is performed. The BIU responds to the command by sending the test report error report message over its MERL and BERL reporting circuits.

The data associated with this command selects which component should respond in FRCed and QMR configurations. The least significant 2 bits of the data control the selection: 00 selects the primary/checker, 01 selects the primary/master, 10 selects the shadow/checker, 11 selects the shadow/master.

BIU Register 1D
Bus Interchange Command

An Interconnect register write to BIU Register 1D causes all nodes in the system to interchange buses, i.e., to switch to their redundant buses. This command allows software to test that each bus and its error recovery mechanisms is working correctly. If a bus does not have a back-up bus, the command has no effect.

The BIU will acknowledge receipt of the command, by means of a memory bus reply, before the command is executed. The BIU will transmit a "bus interchange" error report after the command is executed. Error report logs (the cc field of the Error Report Log, BIU Register 00) are cleared when a bus interchange error report is received. The data associated with the Interconnect register write is ignored.

BIU Register 1E
Detach Command

An Interconnect register write to BIU Register 1E detaches (deactivates) the bus to which the BIU is attached. The BIU will acknowledge the receipt of the command, by means of a memory bus reply, before it is performed. The BIU will issue a detach error report after the command is performed. When a bus is detached it can no longer carry memory requests but will continue to support interconnect access requests. The data associated with the write to this interconnect register is ignored.

The Detach command allows software to perform exactly the same recovery sequence that the hardware automatically takes if it detects a permanent bus error.

BIU Register 1F
Attach Command

When an interconnect access write is performed on BIU Register 1F, the BIU will attach (activate) a bus that had previously been out of service. The Attach command can only be issued to a BIU that is operating on a working back-up bus.

The BIU acknowledges receipt of the command, by means of a memory bus reply, before it is performed. The BIU transmits an attach error report after the command is completed. The data associated with the write to this Interconnect register is ignored.

APPENDIX C
iAPX 43205 MCU REGISTERS

INTRODUCTION

Appendix C describes the set of interconnect registers supported by each iAPX 43205 Memory Control Unit (MCU) component. Some of the MCU registers provide access to data that the MCU provides or requires. MCU commands are initiated by interconnect access to other registers.

Many of the registers have fields (a bit or a group of bits), which are labeled "reserved." Reserved fields may not be modified. The value of the information read from reserved fields is undefined and should be masked off before any comparisons are performed. MCUs always return consistent values in reserved fields (zeroes in the current implementation) to prevent errors in systems employing Functional Redundancy Checking (FRC).

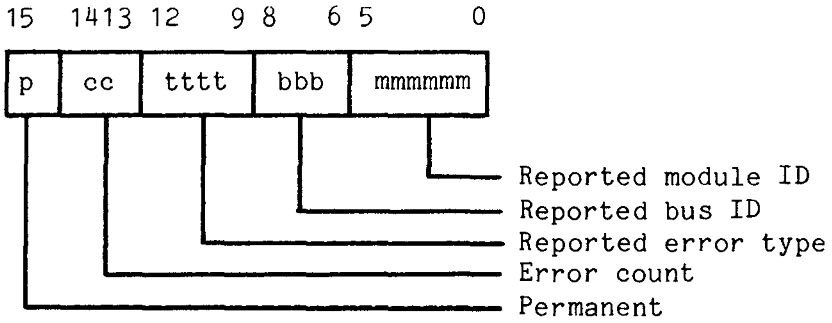
A summary of the MCU interconnect registers is provided on page C-2. Following the summary, each register is described individually.

Table C-1. MCU Register Summary

N-Local Address	MCU Register	Page
1F	Attach Command	C-39
1E	Detach Command	C-38
1D	Bus Interchange Command	C-37
1C	Test Report Command	C-36
1B	Test Detection Command	C-35
1A	***RESERVED***	
19	Clear Location Command	C-34
18	Clear Memory Command	C-33
17	Toggle Normal and Back-up Bus Command	C-32
16	Access ECC Bits Command	C-31
15	Access Data High Command	C-30
14	Access Data Low Command	C-29
13	***RESERVED***	
12	Refresh Address Decrementer	C-28
11	Array Address Size	C-27
10	Refresh Interval Decrementer	C-26
0F	Refresh Interval	C-25
0E	Array Error Log High	C-23
0D	Array Error Log Low	C-22
0C	Array High-Order Address	C-21
0B	Array Low-Order Address	C-20
0A	Spare Bit Select	C-19
09	Memory Start and End	C-18
08	Timeout Duration	C-17
07	Diagnostic	C-15
06	State	C-12
05	Spouse ID	C-11
04	***RESERVED***	
03	***RESERVED***	
02	Interconnect Device ID	C-10
01	Interconnect Device Type	C-7
00	Error Report Log	C-3

MCU Register 00
Error Report Log

```
H   HH   HHHH   HHH   HHHHHH   H: Hardware-alterable
0   00                          : Setup during initialization
A   AA   AAAA   AAA   AAAAAA   A: Alterable by software
                                  by means of Bus Interchange,
                                      Detach, Attach,
                                      Test Report, or
                                      Test Detection
                                      command
```



The mmmmmm field contains the module ID of the reporting source.

The bbb field contains the bus ID of the reporting source.

The tttt field contains the type code for the reported error. The error types and encodings are summarized in Table C-2.

Table C-2. MCU Error Report Types

tttt Code	Message Type	MCU Error
1111	BERL Parity	n/a
1110	MERL Parity	n/a
1101	Unsafe Module	Array FRC Error
1100	Bus Arbitration	n/a
1011	Bus/Module-High	BERL or Bus Arbitration
1010	No Error	Command Report
1001	Module	Module
1000	Bus Parity	n/a
0111	Bus/Module-Low	Broken Module or Bus Parity
0110	Uncorrectable ECC	ECC
0101	Correctable ECC	ECC
0100	Test Report	Command Report
0011	Interchange	Command Report
0010	Attach	Command Report
0001	Detach	Command Report
0000	Toggle	Command Report

The cc field contains the value of a counter that increments each time an error report is logged by the MCU. If the cc counter reaches the maximum count of three, it will hold the value three until it is explicitly cleared. Software may use these bits to track its accesses to the Error Report Logs. The cc counter is cleared during initialization and whenever the Interchange command is executed.

The p field indicates whether an error is permanent (p=1) or transient (p=0). This information is used to determine whether the MCU should switch to a new bus or module and then whether the MCU should enter broken bus mode or broken module mode. The first report of any error is labelled transient (p=0). If a second report of the same error (e.g., the second error is the same type and from the same location, or reported module ID, as the first error) occurs during the transient error window, the error is labeled permanent (p=1). The p field is cleared during initialization and whenever the Bus Interchange command is executed.

Table C-3 indicates how an MCU will respond to permanent and transient errors for each of the types of errors.

Table C-3. MCU Error Recovery Mechanisms

Code	Message type	Recovery Action	
		Transient (p=0)	Permanent (p=1)
1111	BERL Parity	Retry	Bus Switch
1110	MERL Parity	Retry	Module Switch
1101	Unsafe Module	(Note 5)	Module Switch
1100	Bus Arbitration	Retry	Bus Switch
1011	Bus/Module High	Retry	Bus/Module Switch
1010	No Error	None	n/a
1001	Module	Retry	Module Switch
1000	Bus Parity	Retry	Bus Switch
0111	Bus/Module Low	Retry	Bus/Module Switch
0110	Uncorrectable ECC	Retry/Stage	Module Switch
0101	Correctable ECC	Retry/Stage	n/a
0100	Test Report	None	n/a
0011	Interchange	n/a	Bus Exchange
0010	Attach	Attach Bus	n/a
0001	Detach	Detach Bus	n/a
0000	Toggle	None	n/a

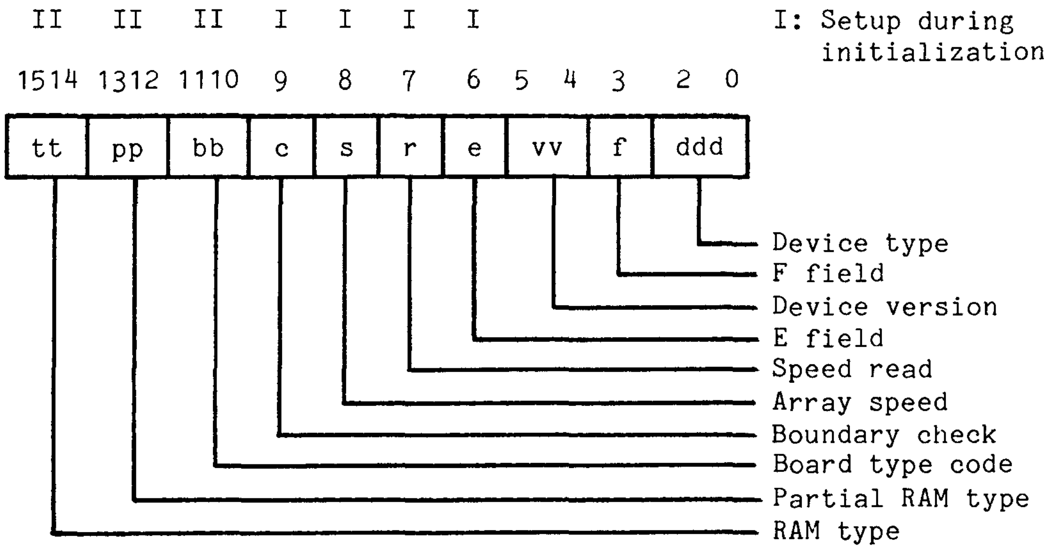
Note that:

- Any error report will cause all outstanding requests to retry, and all memories will stage data for a reply. "Retry," "None," or "Stage" above only indicates which mechanism is required, since all three will occur after an error report.
- All bus switches are followed by a retry and transient error window.
- Any error reports during a transient error window will be classified as permanent or transient based on the last reported error.
- The bus switch or module switch will occur only if the appropriate interconnect register fields are enabled.
- An unsafe module report is always considered a permanent error.

Special Considerations

At initialization time, the cc field (error count) and the p field (permanent bit) are set to zero. The tttt field (reported error type), the bbb field (reported bus ID), and the mmmmmm field (reported module ID) are not modified by the initialization so that error information will be available after a system crash. After a cold-start initialization in a system using FRC, the error log should not be read until a Bus Interchange command has been issued. Otherwise, the error report logs of the master and checker nodes may have different values, which will result in an FRC error.

MCU Register 01
Interconnect Device Type



The ddd, f, and vv fields of the Interconnect Device Type register are fixed within the MCU at the time of manufacture.

The ddd field contains the interconnect device type. For an MCU, the ddd field is 000.

The f field always contains the value one (1).

The vv field contains the device version number for the MCU.

The e, r, s, c, bb, pp, and tt fields of the Interconnect Device Type register are loaded from the SLAD bus when the MCU is initialized. The e field is loaded with the value from SLAD4 during the D2 phase of initialization. The r field is loaded with the value from SLAD5 during the D2 phase of initialization. The s field is loaded with the value of SLAD2 during the D2 phase of initialization. The c field is loaded with the value from SLAD3 during the D2 phase of initialization. The bb field is loaded with the value of SLAD16..15 during the D1 phase of initialization. The pp field is loaded with the value of SLAD18..17 during the D1 phase of initialization. The tt field is loaded with the value of SLAD1..0 during the D2 phase of initialization.

The e field must be set to zero when the MCU is initialized.

The r field indicates if the MCU is to access the storage array in Speed Read mode. When r=1, the MCU performs accesses to the storage array assuming that Ripplemode™ dynamic RAMs (DRAMs) are present. When r=0, the MCU performs accesses with standard multiplexed DRAM function.

The s field indicates the characteristics that the MCU is to use when accessing the external memory array. The s field indicates whether the MCU will access the storage array with nominal (c=1) or extended (c=0) access time. Extended access time lengthens the nominal access time by one component clock cycle.

The c field indicates whether the MCU should perform boundary checking on the row addresses for the storage array. Boundary checking detects when a row boundary is crossed so that the MCU may strobe a new address into the row address latches in the DRAM components. When c=1, boundary checking is performed. When c=0, it is disabled. When the MCU is not in Speed Read mode (that is, r=0) the c field must be set to one.

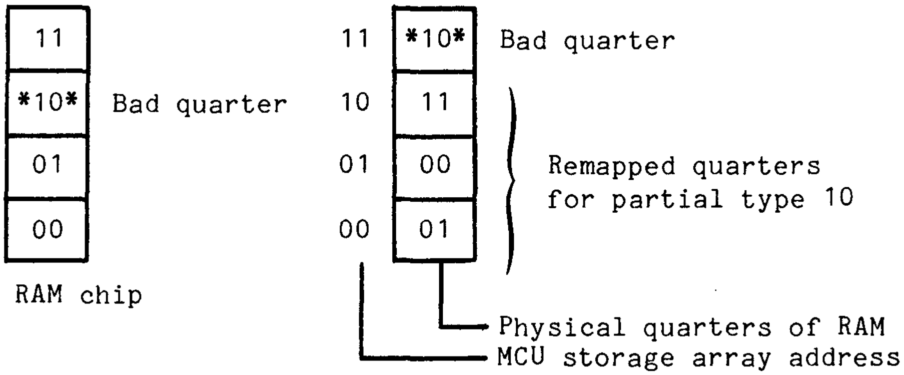
The bb field contains the board type code. The least significant bit of the bb field determines if the MCU is to exercise the optional external buffer checking logic. If enabled (b0=1), the MCU will perform the buffer checking, and if disabled (b0=0) the MCU will perform no checks. The most significant bit of the bb field indicates if there is a spare, unused bus to which the MCU can switch for error recovery. If a spare bus is available (b1=1), the MCU must be connected to two memory buses, a normal bus and a back-up bus.

The pp field contains the partial RAM type. The value of pp specifies how the MCU maps 2 address bits from the memory bus into 2 bits of the storage array address. When 64K RAMs are used in the storage array, memory bus physical address bits A15 and A14 are mapped. When 256K RAMs are used in the storage array, memory bus physical address bits A17 and A16 are mapped.

Memory Bus	Storage Array Mapping			
11	00	01	10	11
10	01	00	11	10
01	10	11	00	01
00	11	10	01	00

Partial type 11, good RAM or bad quarter (11)
Partial type 10, bad quarter (10)
Partial type 01, bad quarter (01)
Partial type 00, bad quarter (00)

The following diagram illustrates one such storage array remapping. This example shows how a RAM array constructed of chips that have a defective quarter (partial type 10) can have storage array addresses reassigned by the MCU in order to allow quarters 00, 01, and 11 to be addressed linearly. The Memory Start and End register (MCU Register 09) must be programmed with corresponding start and end addresses for the storage array.

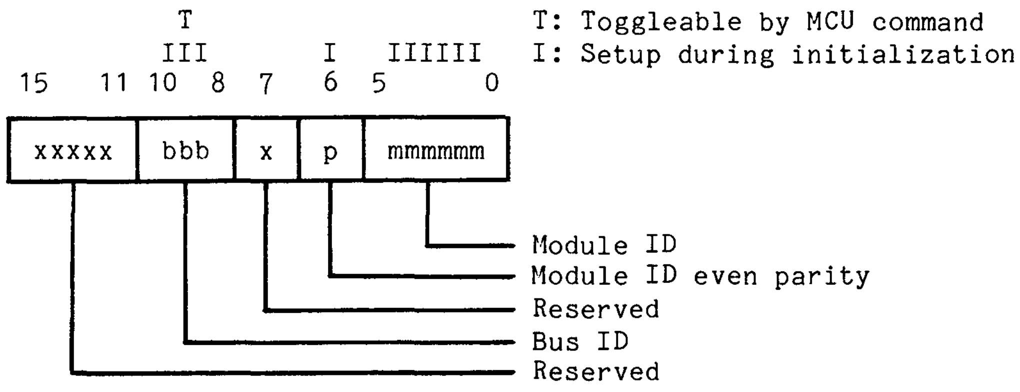


The tt field contains the RAM type code and indicates the type of RAM chips that comprise the storage array. The tt field indicates how the memory bus address bits are to be positioned in the address that is presented to the storage array. The following diagram indicates mappings for the three valid tt field values: tt=00 for 256K dynamic RAMs and static RAMs, tt=01 for 64K dynamic RAMs, and tt=10 for 16K dynamic RAMs. The tt code of 11 is undefined. The numbers in the body of the table correspond to the bit positions of the memory bus address which are mapped onto the corresponding SLAD bus bit position. The memory bus address bit positions denoted here reflect any modification that might occur as a result of partial type mapping.

SLAD Bus Bit Position

	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
tt	Bank				Column Address							Bank		Row Address						
10	15	14	19	17	13	12	11	10	9	8	7	18	16	6	5	4	3	2	1	0
01	17	16	19	15	14	13	12	11	10	9	8	18	7	6	5	4	3	2	1	0
00	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

MCU Register 02
Interconnect Device ID



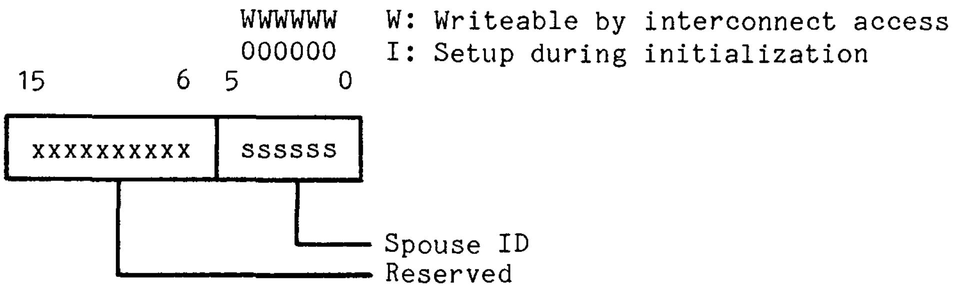
T: Toggleable by MCU command
I: Setup during initialization

The mmmmmm field contains the module ID. This field is loaded from SLAD6..SLAD1 during the D1 phase of MCU initialization. The mmmmmm field defines a unique module position across all of the memory buses. The module ID field is read-only. The MCU uses the module ID when it decides whether to participate in interconnect accesses.

The p field contains the module ID even parity bit. This bit is loaded from SLAD7 during the D1 phase of MCU initialization. If the MCU detects that the parity across the module ID (mmmmmm field) and the p field is not even, then the MCU will permanently assert INIT within the chip. This parity mechanism prevents a module with a corrupt module ID from causing collapse of the system due to reporting errors with an incorrect module ID. Externally, an MCU ceases to toggle the BUFCHK output when the internal INIT signal is asserted.

The bbb field contains the bus ID code, which indicates the MCU's normal bus. In configurations that support bus switching, the back-up bus ID is formed by complementing the middle bit of the bbb field. During software-controlled initialization or reconfiguration, the middle bit of the bbb field may be toggled by MCU interconnect register access (Toggle Normal and Back-up Bus command) to redefine the normal bus ID. When an MCU switches to its back-up bus, the bbb field continues to indicate the MCU's normal bus ID. The MCU output signal BUSSEL (BUS SELect) is a function of the middle bit of the bbb field and the bus state bits in the State register (MCU Register 06).

MCU Register 05
Spouse ID

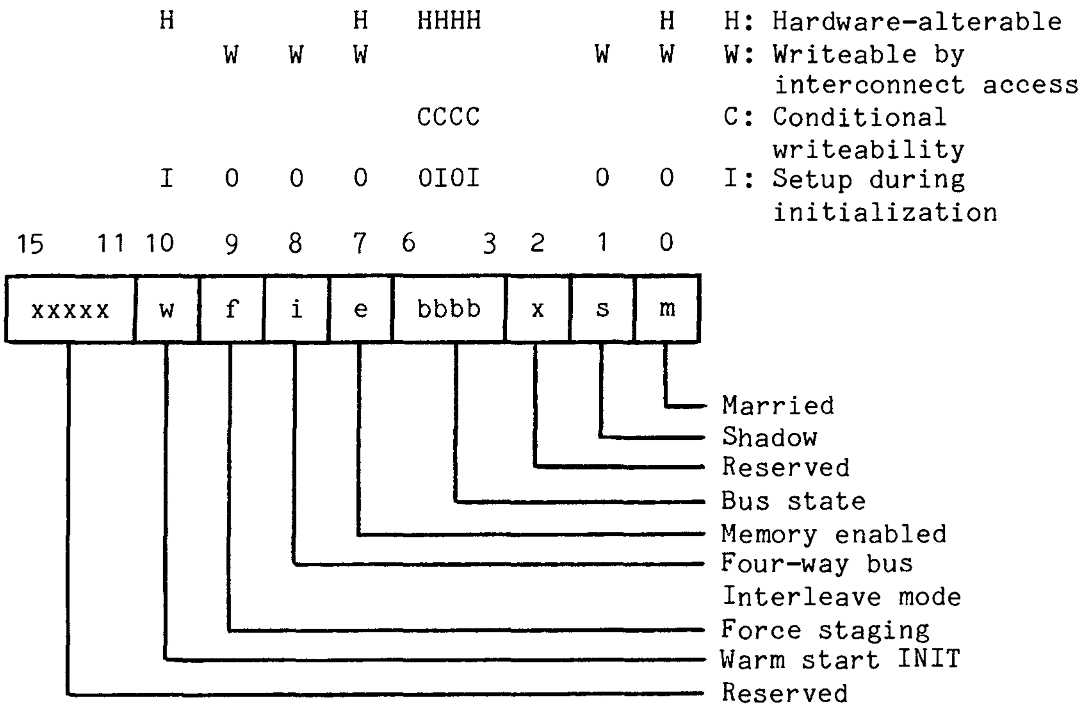


W: Writeable by interconnect access
I: Setup during initialization

The ssssss field contains the module ID for the spouse (other partner) in the primary/shadow pair of modules for systems employing shadowing. The ssssss field is initialized to 000000. The ssssss field should be modified to the module ID of the actual spouse before the married bit is set in the State register (MCU Register 06). Once a MCU is married to its spouse, this register must be treated as read-only. Furthermore, interconnect reads of this register will return different values since the married pair is performing Ping-Pong operation and since the spouse IDs of the primary and shadow are different.

The value of the ssssss field is matched against the reported module ID of an error report to determine if its spouse has an error. If the spouse has reported a permanent module error, this MCU will divorce itself (clear the married bit in the State register, MCU Register 06) and assume responsibility for all future activities of the module.

MCU Register 06
State



The m field contains the married bit, which indicates whether this MCU is married (m=1) to another MCU to form a Primary/Shadow pair. Before the m bit is set, the spouse ID (MCU Register 04) must be configured with the module ID of the MCU which is the spouse. When a MCU is married, it performs in the following manner.

- The married MCU checks error reports of permanent module errors to determine if the module ID of the reporter is the same as that of its spouse. If the MCU detects that a permanent module error has occurred in its spouse, the MCU will reset the married bit.

- The MCU and its married spouse begin a Ping-Pong operation in which the partners alternately handle memory replies. The designated primary MCU will handle the first reference. Even though they are married, the primary and shadow MCUs respond individually to interconnect accesses, since each MCU contains the unique ECC error log information for the storage array it controls.

- Married MCUs do not operate in lock step because the activity in their separate storage arrays (ECC errors, refreshing, and scrubbing) is not identical. Ping-Pong operation keeps the married MCU pair in loose synchronization, since their alternating replies are ordered by the sequence of ordered memory bus requests.

The s field indicates whether the MCU module will become the shadow (s=1) or the primary (s=0) when the married bit is set (m=1).

The bbbb field contains the bus state code. The individual bits of the bbbb field have the following names:

- Bit 0 -- Enable Normal (EN)

- Bit 1 -- Enable Backup (EB)

- Bit 2 -- Backup's Enable Normal (BEN) -- a copy of the enable normal bit for this bus's back-up bus

- Bit 3 -- Backup's Enable Backup (BEB) -- a copy of the enable back-up bit for this bus's back-up bus

EN and BEN are loaded with the value of SLAD12 (REDUNDANT BUS enable) during the D1 phase of MCU initialization. Thus, an MCU without a redundant bus is initialized to the value bbbb=0000, and an MCU with a redundant bus is initialized to the value bbbb=0101. The seven valid combinations of the bbbb field are indicated in Table C-4.

Table C-4. Bus State Codes

bbbb Field				State	Interpretation
BEB	BEN	EB	EN		
0	0	0	0	0	Null State.
0	0	0	1	1	This Node's Normal Bus Is Up.
0	1	0	0	2	This Node's Back-up Bus Is Up.
0	1	0	1	3	Normal Four-Way Address Interleaving Mode.
1	0	1	0	4	Interchanged Four-Way Address Interleaving.
1	1	0	0	5	Normal Bus Is Down.
0	0	1	1	6	Back-up Bus Is Down.

In states 2, 4, and 5 above, an MCU will operate on its back-up bus. In all other states, an MCU will operate on its normal bus. In state 0, the null state, the MCU does not participate in any memory accesses but continues to respond to interconnect accesses. The response of an MCU to the commands Attach, Detach, and Interchange depends on the bus state, the device type (fault-tolerant or standard), and the system configuration (four-way, redundant bus). The state transitions of the bus state field are described in Appendix E.

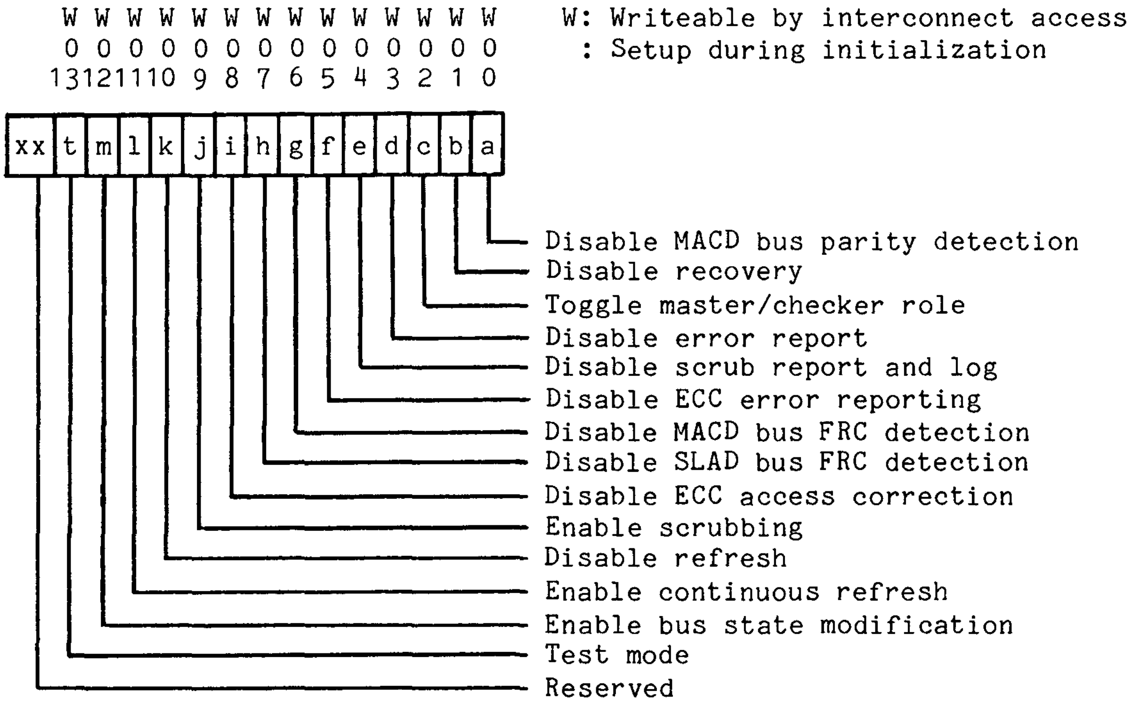
The i field indicates whether four-way bus interleaving (i=1) is being employed in the system. The MCU bases some bus switching decisions on the value of the i field.

The f field indicates whether the MCU is to stage data (f=1) for all memory read accesses or return data immediately (f=0). When data staging is forced (f=1) the MCU performs a storage array read and ECC correction before any data is returned by means of the memory bus.

When no data staging is forced (f=0) the MCU performs a storage array read and immediately returns data before ECC correction is performed. In this case, the BIU that requested the data must buffer it and await a possible error report from the MCU before returning the data to the processor that requested it. Should the MCU detect an ECC error, the BIU would be informed by means of an error report and might retry the access. On the retry, the MCU will stage data automatically.

The w field indicates whether the last INIT pulse was a warm start (w=1) or a cold start (w=0). The w field is loaded from the value on the MACD11 pin during INIT. A warm start will not change the state of the Refresh Address Decrementer register, the Refresh Interval Decrementer register, the Spare Bit Select register, or the Array Error Log register.

MCU Register 07
Diagnostic


W W W W W W W W W W W W W W
0 0 0 0 0 0 0 0 0 0 0 0 0 0
13 12 11 10 9 8 7 6 5 4 3 2 1 0
xx t m l k j i h g f e d c b a
W: Writeable by interconnect access
: Setup during initialization
Disable MACD bus parity detection
Disable recovery
Toggle master/checker role
Disable error report
Disable scrub report and log
Disable ECC error reporting
Disable MACD bus FRC detection
Disable SLAD bus FRC detection
Disable ECC access correction
Enable scrubbing
Disable refresh
Enable continuous refresh
Enable bus state modification
Test mode
Reserved

The a field determines if detection of MACD bus parity errors is to be disabled (a=1) or enabled (a=0).

The b field determines whether the MCU is to disable (b=1) or enable (b=0) its fault recovery mechanisms. If fault recovery is disabled, the MCU will continue to report and log errors; but automatic bus switching is not allowed (except for the explicit Bus Interchange command). The value of the c field allows software to modify the master/checker FRC relationship from the one that was set on initialization.

The c field determines whether the MCU should operate with a reversed (c=1) or an assigned (c=0) FRC master/checker role. For example, an MCU that was assigned on initialization to act as a master FRC device can be commanded to reverse roles (c=1) and operate as a checker. This capability allows software to swap master and checker MCUs to verify correct operation of the original checker.

The d field determines whether the MCU disables (d=1) or enables (d=0) error reporting. When error reporting is disabled, the MCU will continue to log its own errors and receive error reports from other agents.

The e field determines whether the MCU disables (e=1) or enables (e=0) the logging and reporting of ECC errors that occur while scrubbing memory. The reporting and logging of scrub errors might be disabled (e=1) to stop the reports of a known hard-bit failure in the storage array, which will be detected on each scrubbing pass.

The f field determines whether the MCU disables (f=1) or enables (f=0) the reporting of correctable and uncorrectable ECC errors.

The g field determines whether the MCU disables (g=1) or enables (g=0) the detection of FRC errors on the memory address, control, and data (MACD) bus.

The h field determines whether the MCU disables (h=1) or enables (h=0) the detection of FRC errors on the SLAD bus.

The i field determines whether the MCU disables (i=1) or enables (i=0) ECC access correction. If ECC access correction is disabled (i=1), the MCU will not correct data read from the storage array, although detected ECC errors will continue to be reported and logged. For write accesses (a read-modify-write cycle in the storage array), an MCU with its ECC correction disabled (i=1) operates in the following manner: read the storage array with no ECC correction on data, form the composite data (modify), write the composite data to the storage array with the new ECC code (if an uncorrectable error did not occur -- see the j field for more details).

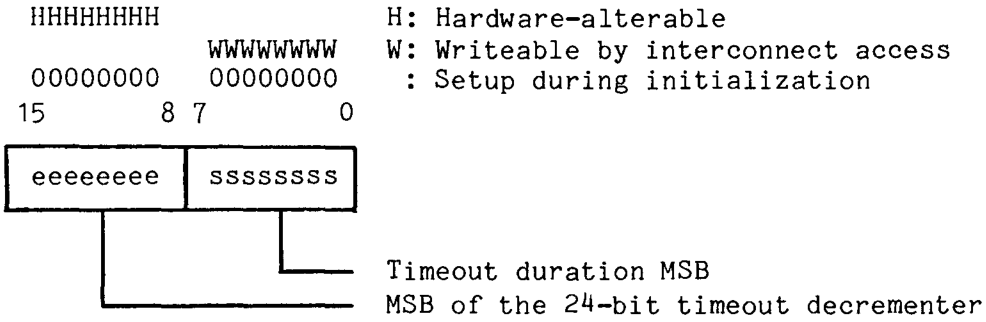
The j field determines whether the MCU enables (j=1) or disables (j=0) scrubbing of the storage array. The j field also controls whether the write portion of a storage array RMW cycle should be aborted when an uncorrectable error is detected in the read portion of the access. When j=1, an uncorrectable error in the read cycle of the request will be logged and reported and the write cycle will be aborted. When j=0, an uncorrectable error will be logged and reported but the write cycle will not be aborted.

The k field determines whether the MCU disables (k=1) or enables (k=0) refreshing of the storage array.

The l field determines whether the MCU enables (l=1) or disables (l=0) continuous refreshing of the storage array. Continuous refresh forces a refresh cycle whenever the storage array is not being accessed. Access to the storage array is permitted on a priority basis. Continuous refresh cycles (lowest priority) have less priority than memory bus accesses. Memory bus access cycles have less priority than refresh cycles that are requested by expiration of the refresh interval decrementer.

The m field determines whether the MCU enables (m=1) or disables (m=0) modification of the bus state field in the state register (MCU Register 06). The t field is used by Intel for testing purposes. A user must never set the t field to a one.

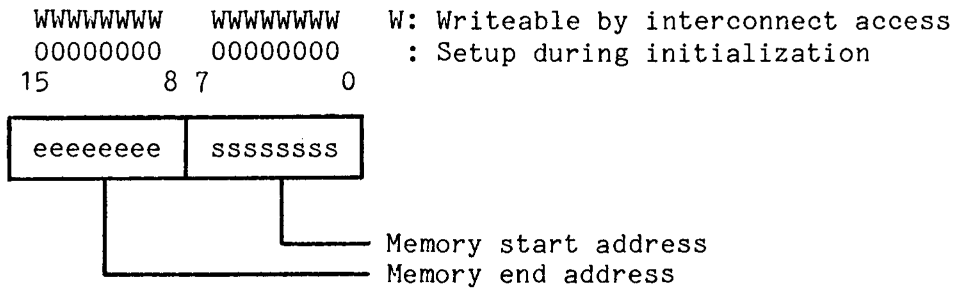
MCU Register 08
Timeout Duration


HHHHHHHH
WWWWWWWW
00000000 00000000
15 8 7 0
eeeeeeee
ssssssss
H: Hardware-alterable
W: Writeable by interconnect access
: Setup during initialization
Timeout duration MSB
MSB of the 24-bit timeout decrementer

A timeout decrementer is used to regulate bus timeouts and the transient error window. When a timing function begins, the MCU loads the 24-bit timeout decrementer. The value from the timeout duration MSB field is loaded into the most significant 8 bits of the decrementer, and the value 128 is loaded into the least significant 16 bits. As the decrementer operates, the value of the 8 most significant bits may be read from bits 8 through 15 of the Timeout Duration register.

During initialization, the timeout duration MSB is set to the value zero. Thus, the shortest timeout (128 clock cycles) will be employed during the period when the system configuration is being determined. Once configuration is complete, the timeout duration must be lengthened so that normal accesses may complete before the decrementer has expired.

MCU Register 09
Memory Start and End



The memory start address and the memory end address define the range on the most significant 8 bits of the 24-bit memory bus address, which the MCU will recognize as an access request. Note that the 24-bit memory bus address is not necessarily the same as the physical address from the processor since a BIU may have modified the address for interleaving.

The first memory bus address supported by the MCU is defined by the 24 bit address:

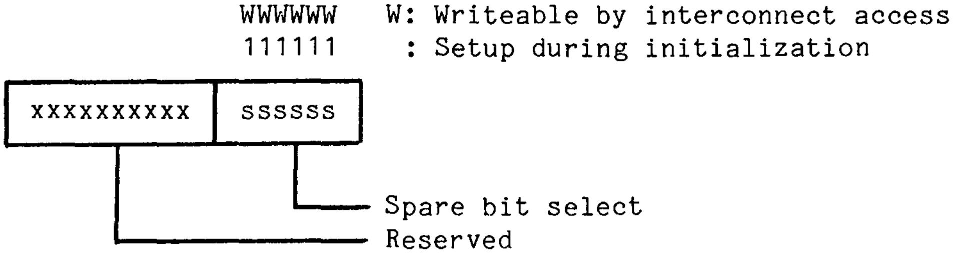
```
 23       16 15                  0
+----------+--------------------+
| ssssssss |  0000000000000000  |   MCU memory bus starting address
+----------+--------------------+
```

The last memory bus address supported by the MCU is defined by the 24 bit address:

```
 23       16 15                  0
+----------+--------------------+
| eeeeeeee |  1111111111111111  |   MCU memory bus ending address
+----------+--------------------+
```

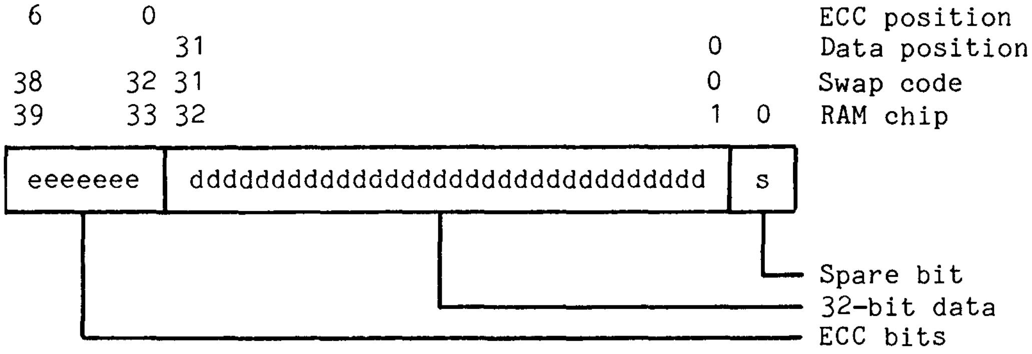
Thus, if the memory start and end addresses are identical, the memory size is 64K bytes.

MCU Register 0A
Spare Bit Select


WWWWWW W: Writeable by interconnect access
111111 : Setup during initialization
xxxxxxxxxx
ssssss
Spare bit select
Reserved

The spare bit select field specifies the bit position in the storage array which is to be replaced with the spare RAM chip. With the spare RAM chip, the MCU can be commanded to repair any single hard failure in the storage array. When a cold INIT occurs, the MCU sets the spare bit select field to 63 (all ones) to signify that no spare RAM chip is selected.

Physically, the storage array must be organized in the following manner.



The sssssss field must be programmed with the swap code for the RAM chip that is to be replaced. For example, sssssss=0000000 (swap code 0) will cause the MCU to replace data bit 0 (RAM chip 1) with the spare RAM chip. With sssssss=0100000 (swap code 32) the MCU will replace ECC bit 0 (RAM chip 33) with the spare RAM chip.

MCU Register 0B
Array Low-Order Address

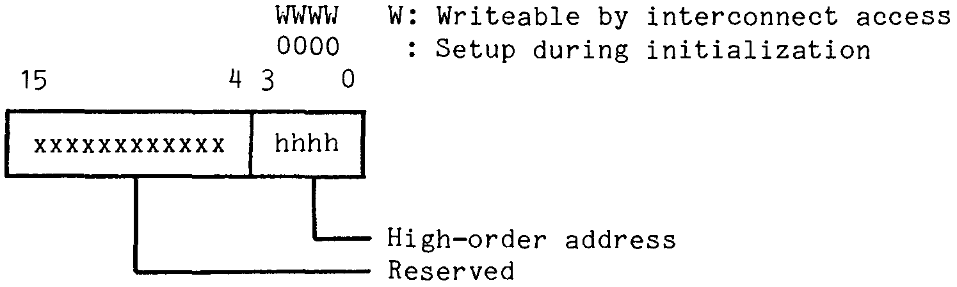
```
 WWWWWWWWWWWWWWWW   W: Writeable by interconnect access
    MACD15..0        : Setup during initialization
15               0
+----------------+
|aaaaaaaaaaaaaaaa|
+----------------+
         |
         +-----------Low-order address
```

The 16-bit low-order address is the least significant 16 bits of a 20-bit address used by the MCU to perform a Clear Location, Access ECC Bits, or Access Data command. The most significant 4 bits of the 20-bit address are contained in the Array High-Order Address register (MCU Register 0C).

When INIT occurs, the Array Low-Order Address register is loaded with the 16 bits on the MACD bus. MACD11 (warm-start bit) and MACD10..MACD8 (bus number) have assigned roles but the remainder may be defined by the user.

The Array Low-Order Address register may also be used as a general-purpose 16-bit register.

MCU Register 0C
Array High-Order Address


WWWW
0000
W: Writeable by interconnect access
: Setup during initialization
15
4
3
0
xxxxxxxxxxxx
hhhh
High-order address
Reserved

The high-order address field contains the most significant 4 bits of a 20-bit storage array address used by the MCU to perform the Clear Location, Access ECC Bits, and Access Data commands. The low-order 16 bits of the 20-bit storage array address are provided by the Array Low-Order Address register (MCU Register 0B).

The high-order address field is set to zero when INIT occurs.

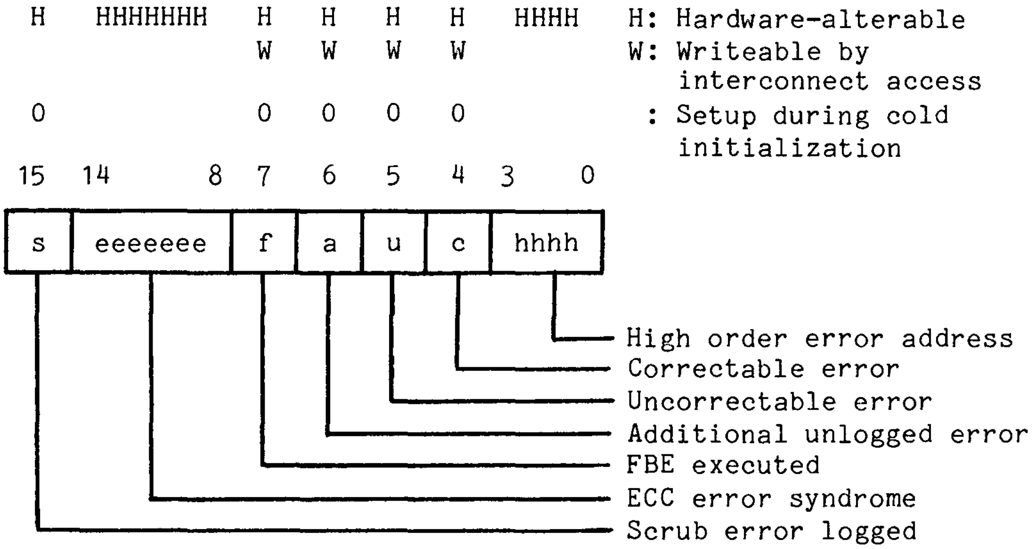
Recall that the address presented to the MCU on the MACD bus may be different from the address that a BIU receives (from its associated processor) if the BIU is performing interleaving.

MCU Register 0D
Array Error Log-Low

```
 HHHHHHHHHHHHHHHH   H: Hardware-alterable
15              0
 aaaaaaaaaaaaaaaa
        |
        +-------- Low-order error address
```

The 16-bit low-order error address is the least significant 16 bits of the 20-bit storage array address at which an ECC error occurred. The Array Error Log-High register (MCU Register 0E) contains additional information about the ECC error.

MCU Register 0E
Array Error Log-High



The high-order error address contains the most significant 4 bits of the 20-bit storage array address at which the error occurred. The high order error address is set to zero when INIT (cold initialization) occurs. The least significant 16 bits of the storage array address are contained in the Array Error Log-Low register (MCU Register 0D).

The c field indicates if a correctable ECC error (c=1) has occurred. Correctable errors may arise from single-bit errors in either the data or the ECC fields of the storage array. Single-bit errors in the address field are considered uncorrectable errors.

The u field indicates if an uncorrectable ECC error (u=1) has occurred. Uncorrectable errors arise either from double-bit errors in the data or ECC fields of the storage array or from single-bit errors in the address field of an access. The single-bit address errors are fixed by a retry of the access by the BIU that requested it.

When the MCU detects an ECC error (either correctable or uncorrectable) the storage array address of the error is latched into the Error Log register along with the ECC syndrome bits and an indication if the error was caused by a scrubbing operation (s=1) or a normal memory access (s=0).

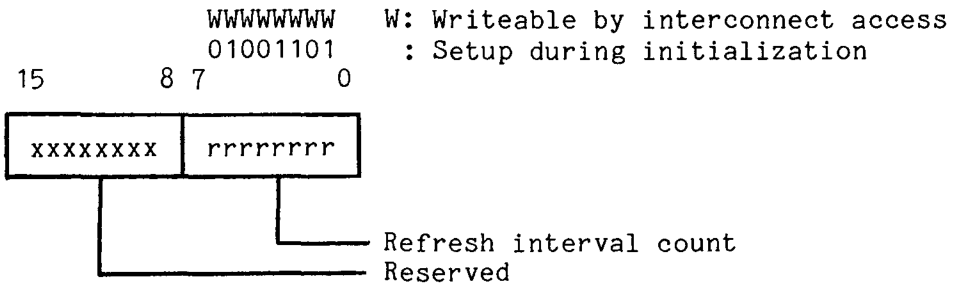
The a field (additional unlogged error) indicates if two or more ECC errors have occurred (a=1) before software has serviced the error log. The MCU sets the a field to one when an ECC error occurs, and either the c field or the u field is already set to one. The MCU inhibits new error address, syndrome information, and scrub indicator from being loaded into the Error Log register so that the information about the first ECC error is retained.

The f field indicates when a Force Bad ECC (FBE) command has been executed (f=1). The ECC syndrome field will contain all ones for an error logged as a result of the FBE command.

The eeeeeee field contains the ECC error syndrome bits which were calculated for the storage array location where the error was encountered.

The s field indicates if the ECC error occurred during the scrubbing of a storage array location (s=1). If an ECC error occurred, but was associated with a normal memory access, the scrub error logged field will be cleared (s=0).

MCU Register OF
Refresh Interval



The refresh interval count determines the nominal time between refresh cycles for the storage array. When the refresh interval decrementer (MCU Register 10) counts to zero, the decrementer is reloaded with the refresh interval count (rrrrrrrr field). Each MCU component clock cycle decrements the count in MCU Register 10.

Upon initialization, the refresh interval count is loaded with the binary value 01001101 (77 decimal). This value will provide the storage array RAM chips with 128 refresh cycles about every 2 milliseconds (exactly 1.9712 milliseconds) for an MCU that is operated with a 5-MHZ clock. Any changes in the MCU component clock or the type of RAM chip refresh cycles requires that software modify the refresh interval count from the default value.

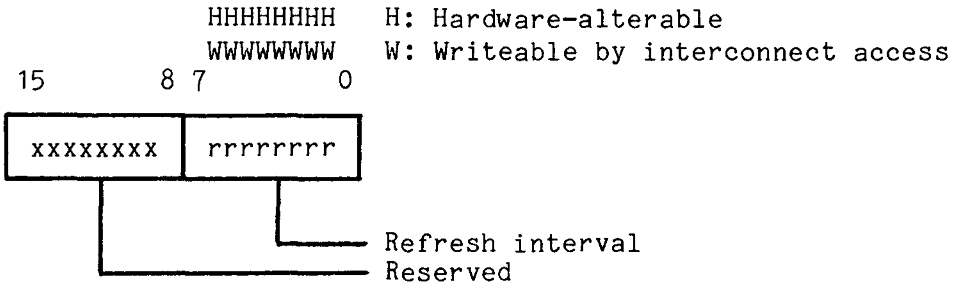
If it is necessary that memory data be preserved during initialization, the length of INIT must be included in the calculation for the revised refresh interval. The MCU will not refresh the storage array during INIT. Furthermore, the MCU will not begin the INIT sequence until it completes any storage cycle that may be in progress. For example, when using a 5-MHz (200-nanosecond) MCU with 64K dynamic RAMs with 128 cycle/2 millisecond refreshing, the MCU must perform 128 refresh cycles in each refresh period. The time available for refresh in each refresh period is:

Time For Refresh = (2 milliseconds) - time (INIT) - time (write cycle).

The refresh interval value for this example would be calculated:

Refresh Interval = Integer ((Time For Refresh/128) / 200 nanoseconds).

MCU Register 10
Refresh Interval Decrementer

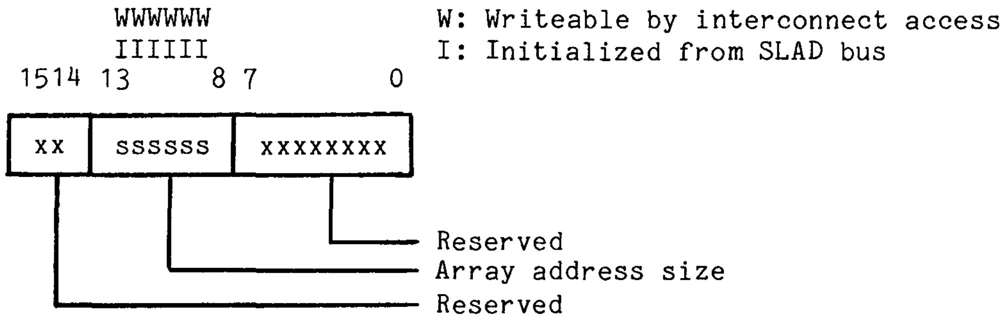


The rrrrrrrr field indicates the number of MCU component clocks remaining until the next refresh request.

This register holds its current value (does not decrement) during initialization. After initialization, the Clear Memory command must be issued to load the Refresh Interval register into the refresh interval decrementer. This preconditioning of the refresh interval decrementer must be performed in a FRC system before it can be read error-free.

Note that the disable refresh field in the Diagnostic register (MCU Register 07) does not disable the decrementation of the rrrrrrrr field. Only refresh requests from the decrementer are disabled.

MCU Register 11
Array Address Size



The ssssss field specifies the amount of memory which is scrubbed by the MCU. The Array Address Size register works in conjunction with the refresh address decrementer (MCU Register 12).

The ssssss field is loaded into the top 6 bits of the 20-bit refresh address decrementer when the decrementer count reaches zero or when the Clear Memory command is issued. The low-order 14 bits of the refresh address decrementer are all set to one when the decrementer reloads. The ssssss field value plus one specifies the number of regions (each containing 16K storage array locations) that will be scrubbed.

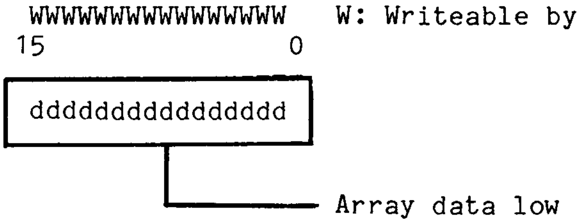
The ssssss field is loaded from the SLAD15..SLAD10 field of the SLAD bus during the D1 phase of MCU initialization

MCU Register 12
Refresh Address Decrementer

```
       HHHHHHHHHHHHHH   H: Hardware-alterable
       11111111111111   A: Alterable by Clear Memory command
       WWWWWWWWWWWWWW   W: Writeable by interconnect access
  1514 13            0
 ______________________
|  xx | dddddddddddddd |
|_____|________________|
   |         |
   |         |________ Refresh address decrementer
   |                   least significant 14 bits
   |__________________ Reserved
```

The least significant 14 bits of a 20-bit refresh address decrementer are accessible through this register. The upper 6 bits of the refresh address decrementer may be cleared by an interconnect access double-byte write to MCU Register 12. (A byte write to this register will modify only a portion of the register.) The upper 6 bits of the refresh address decrementer are loaded from the Array Address Size register (MCU Register 11) during cold-start initialization, each time the decrementer reaches the count of zero, and whenever the Clear Memory command is issued. The Clear Memory command also loads the least significant 14 bits of the refresh address decremeter with all ones.

MCU Register 14
Access Data-Low Command



W: Writeable by interconnect access

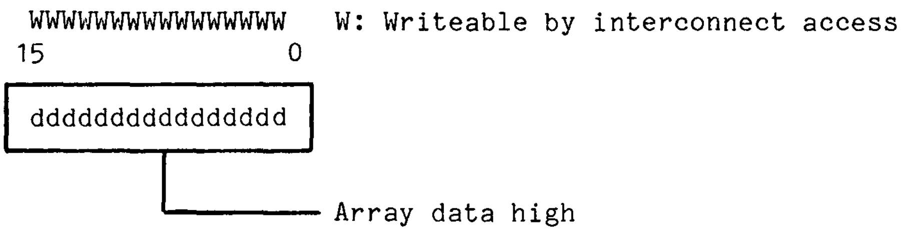
Interconnect access of MCU Register 14 commands the MCU to read or write the least significant 16 bits of one of the 32-bit storage array locations. The Array Low-Order Address (MCU Register OB) and the Array High-Order Address (MCU Register OC) specify the location of the storage array which is to be accessed.

For an interconnect read of MCU Register 14, the storage array is read and the least significant 16 bits of data are returned with no ECC correction taking place. This allows direct access to the least significant 16 bits of the storage array location. Any errors detected during the operation will be logged and reported.

For an interconnect write to MCU Register 14, the storage array is read and any errors in the most significant 16 bits of the location are corrected. Then the composite 32-bit value is written to the array with correct ECC bits. Any errors detected during the operation will be logged and reported.

Recall that ECC correction and the reporting of ECC errors can be disabled via the Diagnostic register (MCU Register 07).

MCU Register 15
Access Data-High Command



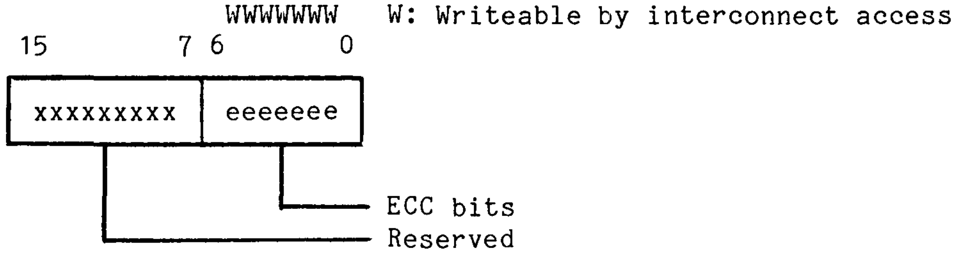
Interconnect accesses to MCU Register 15 command the MCU to read or write the most significant 16 bits of one of the 32-bit storage array locations. The Array Low-Order Address (MCU Register 0B) and the Array High-Order Address (MCU Register 0C) specify the location of the storage array to be accessed.

For an interconnect access read of MCU Register 15, the storage array is read, and the most significant 16 bits of data are returned with no ECC correction taking place. This allows direct access to the most significant 16 bits of the storage array location. Any errors detected during the operation will be logged and reported.

For an interconnect access write to MCU Register 15, the storage array is read and any errors in the least significant 16 bits of the location are corrected. Then the composite 32-bit value is written to the array with correct ECC bits. Any errors detected during the operation will be logged and reported.

Recall that ECC correction and the reporting of ECC errors can be disabled via the Diagnostic register (MCU Register 07).

MCU Register 16
Access ECC Bits Command



The Access ECC Bits command may be used to directly access the 7 ECC bits associated with each location in the storage array. The Array Low-Order Address (MCU Register 0B) and the Array High-Order Address (MCU Register 0C) specify the storage location to access.

An interconnect access read of MCU Register 16 causes the MCU to read the storage array with no ECC checking or correction. The value of the ECC bits for the addressed storage location is returned by means of a memory bus reply.

An interconnect access write to MCU Register 16 causes the MCU to read the storage array, check and correct the 32 bits of data, and then write the ECC bits provided by the interconnect access to the storage array with the data.

MCU Register 17
Toggle Normal and Back-up Bus Command

An interconnect access write to MCU Register 17 will cause the MCU to toggle the middle bit of the bus ID field of the Interconnect Device ID register (MCU Register 02). The value of the data written to MCU Register 17 is ignored.

The MCU will perform the command by the following sequence of operations:

- Immediately acknowledge acceptance of the command by means of the memory bus

- Internally modify the middle bit of the bus ID field, thereby switching to the alternate bus

- Report a "toggle error" on the bus to flush the bus pipeline and cause each outstanding access to be retried

MCU Register 18
Clear Memory Command

An interconnect access write to MCU Register 18 causes the MCU to clear the storage array by writing data zeroes to all locations. The MCU utilizes the refresh address decrementer to generate the required sweeping address sequence.

The MCU performs the Clear Memory command in the following sequence:

- Immediately acknowledge the acceptance of the command via a memory bus reply.

- Load the array address size (MCU Register 11) into the most significant 6 bits of the 20-bit refresh address decrementer. Load ones into the least significant bits of the 20-bit refresh address decrementer. Load the Refresh Interval register (MCU Register OF) into the refresh interval decrementer (partly accessible by means of MCU Register 10).

- Clear all storage locations.

The value of the data that accompanies the interconnect access write to MCU Register 18 is ignored. Though it will continue to accept memory requests until the MACD pipeline is filled, the MCU will not reply to the requests until the Clear Memory command has been completed.

The user must guarantee that the Timeout Duration register (MCU Register 08) contains a sufficient time length for this operation to complete.

MCU Register 19
Clear Location Command

An interconnect access write to MCU Register 19 causes the MCU to clear one location of the storage array. The storage location to be cleared is specified by a 20-bit address formed from fields of the Array High-Order Address register (MCU Register 0B) and the Array Low-Order Address register (MCU Register 0A). The addressed location is cleared by writing a zero to each of the 32 data positions and by computing and storing 7 ECC bits (based on the 32 bits of data and 20 bits of storage array address). The data that accompanies the inte፣connect access write is ignored by the MCU. The interconnect access write is acknowledged by means of the memory bus after the Clear Location command has been performed.

MCU Register 1B
Test Detection Command

An interconnect access read of MCU Register 1B causes the MCU to execute the Test Detection command. This command may be issued to check that all the detection circuits (FRC circuits, MACD bus parity circuits, and buffer checking circuits) are operating correctly.

The FRC circuits contain comparators (exclusive-OR gates) that check that the external signals bear the same information that internal circuits computed. The MCU tests the FRC circuits by issuing information on its external signal paths, providing the complement of the information to internal FRC circuits, and checking that all FRC comparators indicate an error. The MCU checks the MACD bus parity bits and the buffer checking logic in a similar fashion.

The MCU exercises the storage array side signals by issuing a storage array read of the location addressed by the contents of the Array High-Order Address (MCU Register 0C) and the Array Low-Order Address (MCU Register 0B). The MCU exercises the memory bus signals, including parity, by returning the 2 least significant bytes of the data returned from the storage array.

If any FRC comparator fails to indicate an error or the buffer checking logic fails, the MCU will issue a MODULE ERROR error report. Otherwise the MCU will issue a NO ERROR error report.

MCU Register 1C
Test Report Command

An interconnect access write to MCU Register 1C causes the MCU to execute the Test Report command. The MCU acknowledges the command by means of a memory bus reply before the command is executed. The MCU performs this command by sending the TEST REPORT error report by means of the Module Error Report Line (MERL) and Bus Error Report Line (BERL) circuits. The least significant bit of the data associated with the interconnect access write controls whether the master (MACD0 = 0) or the checker (MACD0 = 1) generates the report.

MCU Register 1D
Bus Interchange Command

An interconnect register write to MCU Register 1D causes all nodes in the system to interchange buses, i.e., to switch to their redundant buses. This command allows software to test that each bus, and its error recovery mechanisms, is working correctly. If a bus does not have a back-up bus, the command has no effect.

The MCU will acknowledge receipt of the command, by means of a memory bus reply, before the command is executed. The MCU will transmit a BUS INTERCHANGE error report after the command is executed. The error report counters (cc field of the error report log, MCU Register 00) are cleared when a BUS INTERCHANGE error report is received. The data associated with the interconnect register write is ignored.

MCU Register 1E
Detach Command

An Interconnect register write to MCU Register 1E detaches (deactivates) the bus to which the MCU is attached. The MCU will acknowledge receipt of the commmand before executing it. The MCU will issue a DETACH error report after the command is performed. When a bus is detached, it can no longer support memory requests, but it will continue to support interconnect access requests. The data associated with the write to this interconnect register is ignored.

The Detach command allows software to perform exactly the same recovery sequence that the hardware automatically follows if it detects a permanent bus error.

MCU Register 1F
Attach Command

When an Interconnect register write is performed on MCU Register 1F, the MCU will attach (activate) a bus which had previously been out of service. The Attach command can be issued only to an MCU that is operating on a working back-up bus.

The MCU acknowledges receipt of the command, by means of a memory bus reply, before it is performed. The MCU transmits an ATTACH error report after the command is completed. The data associated with the write to this interconnect register is ignored.

INTRODUCTION

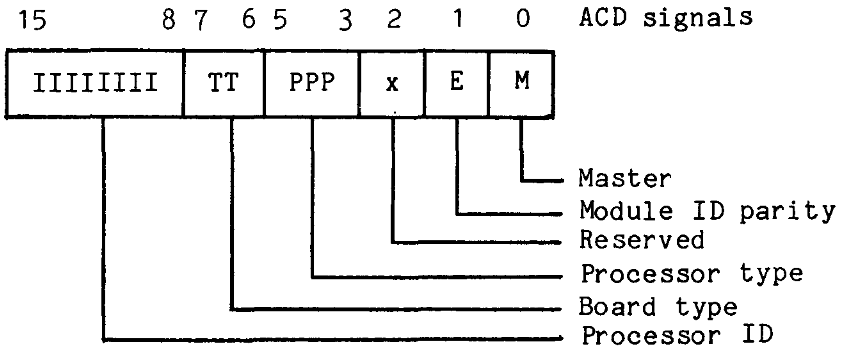
Appendix D describes the information that each BIU and MCU acquires from external logic when the device is initialized. This information transfer allows the required configuration data and optional customer-defined parameters to be loaded into fields of Interconnect registers in order to specify the actual hardware environment. Once the devices are initialized, the information may be accessed by the usual Interconnect register operations.

For each component type, the physical location of each field of the initialization parameters is presented. Then a cross-reference is provided to locate the destination field in the interconnect registers that receive the parameters. Further information about initialization is provided in the applicable register descriptions for the BIU (Appendix B) and MCU (Appendix C).

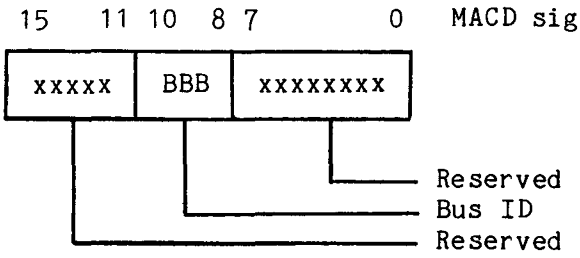
BIU EXTERNAL INITIALIZATION PARAMETERS

During initialization, the BIU acquires initialization parameters that are present on its processor side (ACD15..ACDO signals) and its memory bus side (MACD15..MACDO signals).

On the processor-side, the BIU acquires the following information.



On the memory bus side, the BIU acquires the following information.



Two valid memory bus parity bits must also be presented to the BIU that receives this data.

BIU INITIALIZATION PARAMETER DESTINATIONS

The BIU initializes various fields of its interconnect registers with fields of the initialization parameters.

The 16-bit value on MACD15..MACD0 is loaded into the Test Detection Data register (BIU Register 0B).

The M field is loaded into the internal master flip-flop, which determines if the BIU is to be a master (M=1) or checker (M=0) device for FRC purposes. The state of the master flip-flop may not be read by interconnect register access. The master/checker role of the BIU may be reversed from this assigned role by setting the Toggle Master/Checker bit in the BIU Diagnostic register (BIU Register 07).

The E field is loaded into an internal flip-flop in the BIU. This flip-flop contains the even parity bit for the module ID.

The PPP field is loaded into the Processor Type field of the Interconnect Device Type register (BIU Register 01).

The TT field is loaded into the Board Type field of the Interconnect Device Type register (BIU Register 01).

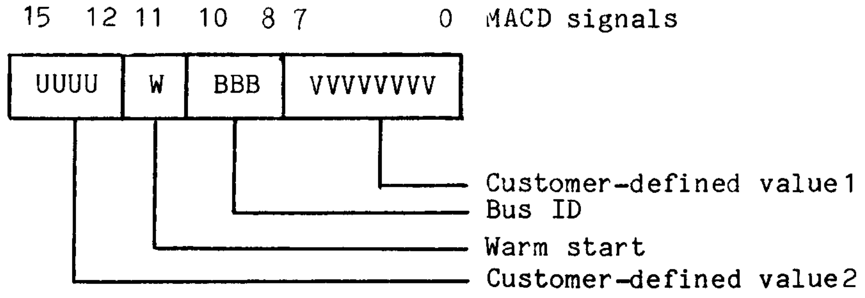
The IIIIIII field is loaded into the processor ID field of the Processor ID register (BIU Register 0A). The least significant 6 bits of the IIIIIII field (ACD13..ACD8) are loaded into the module ID field of the Interconnect Device ID register (BIU Register 02). The least significant 6 bits of the IIIIIII field (ACD13..ACD8) are also loaded into the logical ID field of the Logical ID register (BIU Register 04) in reversed bit order. The BBB field is loaded into the Bus ID field of the Interconnect Device ID register (BIU Register 02).

Any information present in reserved fields of the external parameters is ignored by the BIU.

MCU EXTERNAL INITIALIZATION PARAMETERS

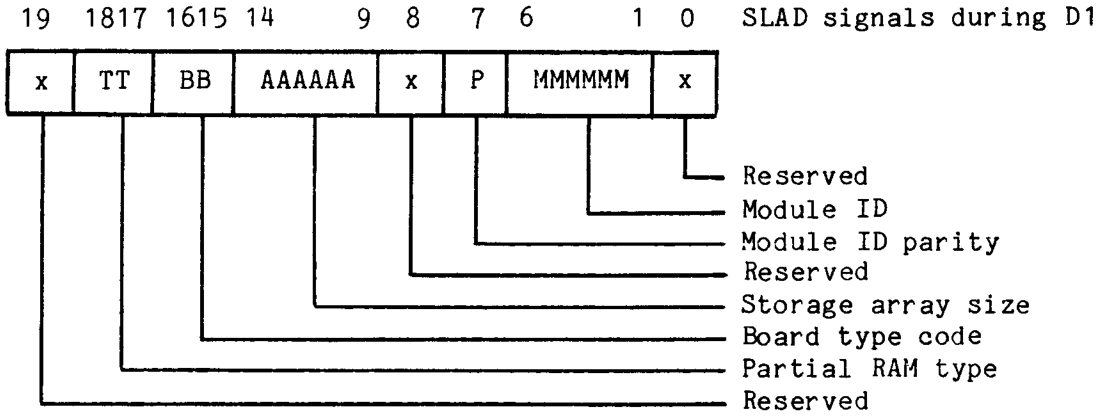
The MCU acquires external initialization parameters from both its memory bus side (MACD15..MACD0 signals) and its storage array side (SLAD19..SLAD0 signals) during initialization. The storage array side information is presented in two phases (D1 and D2).

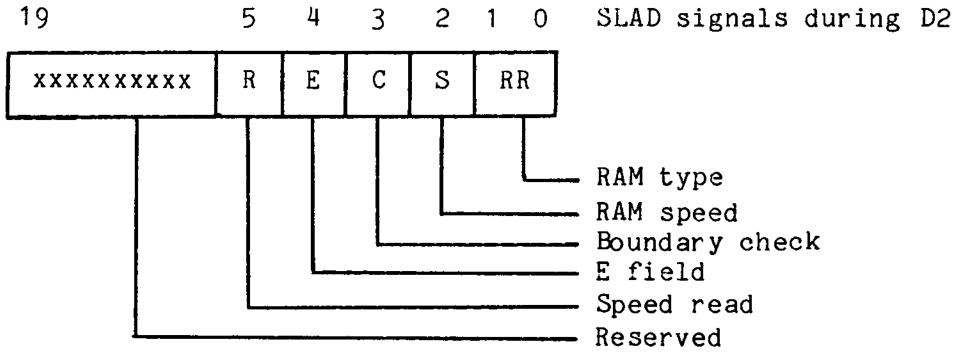
On the memory bus side, the MCU acquires the following information.



The 2 valid MACD bus parity bits must also be presented to the MCU when the above data is generated.

On the storage array side, the MCU acquires the following information.





MCU INITIALIZATION PARAMETER DESTINATIONS

The MCU loads fields of the initialization parameters into specific fields of its interconnect registers.

The value on MACD15..MACDO is loaded into the Array Low-Order Address register (MCU Register OB).

The BBB field is loaded into the Bus ID field of the Interconnect Device ID register (MCU Register 02).

The W field is loaded into the Warm-Start INIT field of the State register (MCU Register 06).

The MMMMMM field is loaded into the module ID field of the Interconnect Device ID register (MCU Register 02).

The P field is loaded into the module ID even parity field of the Interconnect Device ID register (MCU Register 02).

The AAAAAA field is loaded into the array address size field of the Array Address Size register (MCU Register 11).

The BB field is loaded into the board type code field of the Interconnect Device Type register (MCU Register 01).

The TT field is loaded into the partial RAM type field of the Interconnect Device Type register (MCU Register 01).

The RR field is loaded into the RAM type field of the Interconnect Device Type register (MCU Register 01).

The S field is loaded into the array speed field of the Interconnect Device Type register (MCU Register 01).

The C field is loaded into the boundary check field of the Interconnect Device Type register (MCU Register 01).

The E field is loaded into the E field of the Interconnect Device Type register (MCU Register 01).

The R field is loaded into the speed read field of the Interconnect Device Type register (MCU Register 01).

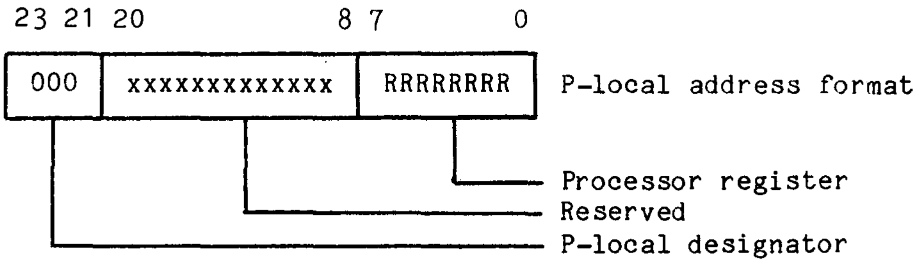
Any information presented in reserved fields of the external parameters is ignored by the MCU.

INTRODUCTION

Appendix E describes the format of addresses in the interconnect architecture. The interconnect architecture supports three distinct address spaces: the P-local (processor-local), N-local (general interconnect register access), and physical memory address space.

P-LOCAL ADDRESSES

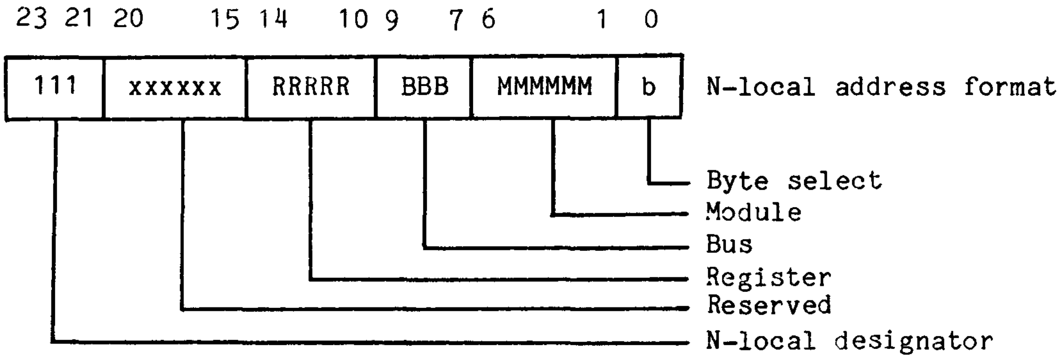
The P-local (short for Processor-local) address space is independent of the interconnect system and allows a processor to obtain its processor ID and to use the interprocessor communication (IPC) facility. The P-local address space is normally accessed only by an iAPX 432 processor. The format of a P-local address, appearing on the ACD bus between a processor and its associated BIU follows:



The RRRRRRRR field indicates which processor register is to be accessed. Currently, two of the RRRRRRRR codes are assigned and the remainder are reserved. RRRRRRRR=00000000 provides access to the processor ID register, which is also accessible as BIU Register 0A of the N-local address space. RRRRRRRR=00000010 provides access to IPC register, which is also accessible as BIU Register 0C of the N-local address space. P-local accesses are generated by specifying the P-local address format with the iAPX 432 Move to Interconnect and Move from Interconnect instructions.

N-LOCAL ADDRESSES

The N-local (short for Node-local) address space provides access to interconnect registers in any interconnect component located at a node in the system. The format of an N-local address, as it appears on the ACD bus and on the MACD bus, follows:



The byte select field indicates if the access is to begin on the even (b=0) or odd (b=1) interconnect address boundary.

The module field indicates which module is addressed. The module address of all ones (111111 binary or 63 decimal) is reserved as the "my-BIU" address and allows direct access to registers in the same module as the processor. Access of a "my-BIU" address does not generate a memory bus interconnect register access.

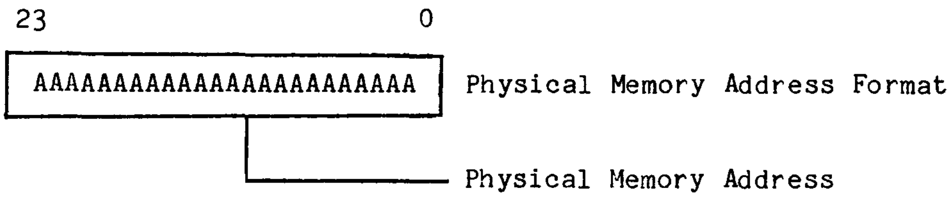
The bus field indicates which bus is to be used for the access.

The register field corresponds to the number of the interconnect register to be accessed. For example, the Interconnect Device ID register (BIU Register 02 or MCU Register 02) of an interconnect component is accessed with RRRRR=00010.

N-local accesses are generated by specifying the N-local address format with the iAPX 432 Move to Interconnect and Move from Interconnect instructions.

PHYSICAL MEMORY ADDRESSES

The physical memory address format is a 24-bit binary byte address. Note that a memory address from a processor may be modified by a BIU before it is presented to the memory bus if interleaving is used.



APPENDIX F
MEMORY BUS

INTRODUCTION

Appendix F describes the iAPX 432 interconnect system memory bus and the protocol that governs communication between iAPX 432 BIUs and MCUs. First, the memory bus message formats are described. Then, the algorithms that regulate the use of the memory bus are provided. Finally, the state diagrams that describe automatic bus switching are presented.

MEMORY BUS MESSAGE TYPES

There are three categories of memory bus messages:

1. Requests
2. Replies
3. Bus Notifications

Each BIU and MCU on a memory bus maintains a three-level pipeline of unsatisfied requests. Each memory bus request message deposits one entry into the pipeline. Each memory bus reply message removes one entry from the pipeline. The entries in the pipeline are serviced on a first come-first served (FCFS) basis. The order of requests and replies is always rigorously preserved. Bus notification messages do not affect the state of the pipeline.

The format for each of the three memory bus message types is presented in the following pages. The message format tables indicate the values on the high- and low-order byte positions of the memory bus (MACD signals), and the corresponding bus control code for each step in a message transfer. The progression of time (0..C-1 where C is the number of bus cycles required to complete the message) is indicated in the T field of the tables.

The following abbreviations are used in the message format tables:

MADRB(n)	- Memory Address Byte "n".
DATAB(n)	- Data Byte "n".
IADRB(n)	- Interconnect Address Byte "n".
T	- Bus Time.
IDLE	- Data bus is high-impedance.
DDDDDDDD	- Destination processor ID for interprocessor communication.

The CCC field indicates the value of the three BUSCTL signals that control the activity on the memory bus. The activity of the CCC field is indicated in the message format tables. The interpretation for the CCC field follows.

CCC	Interpretation
111	Idle : MACD data bus is high-impedance.
001	Request : Normal.
000	Request : Last cycle warning.
011	Reply : Normal.
010	Reply : Last cycle warning.
110	Reply : First cycle of write acknowledge. Non-data reply (last cycle significance).
101	Reply : Last cycle of a RMW Read Reply.
101	Other : Other replies and acknowledges.
100	Bus Notification : Last cycle warning.

The LLLL field indicates the number of bytes of memory data to be transferred by a memory access. The encoding of the LLLL field follows.

LLLL	Memory Access Length
0000	1 byte
0001	2 bytes
0010	3 bytes
0011	4 bytes
0100	5 bytes
0101	6 bytes
0110	7 bytes
0111	8 bytes
1000	9 bytes
1001	10 bytes
1010	11 bytes
1011	12 bytes
1100	13 bytes
1101	14 bytes
1110	15 bytes
1111	16 bytes

The B field indicates the number of bytes of information to be transferred by an interconnect access. If B=0, 1 byte of Interconnect register data will be transferred. When B=0, the least significant bit of the 24-bit interconnect address indicates whether the upper or lower byte of the Interconnect register is to be transferred. If B=1, 2 bytes of Interconnect register data will be transferred. When B=1, the 24-bit interconnect address must be aligned to a double byte boundary; i.e., the least significant bit of the interconnect address must be zero.

The following tables summarize the information sequences for the three types of memory bus (or MACD bus) messages. Table F-1 describes the memory bus request message formats. Table F-2 describes the reply message formats. Table F-3 describes the bus notification message formats.

Table F-1. Memory Bus Request Message Formats

CCC	Upper MACD Byte	Lower MACD Byte	T	Request Message
2 0	15 8	7 0		
000	0000LLLL	MADRB(2)	0	Memory Read Request
001	MADRB(1)	MADRB(0)	1	
000	0001LLLL	MADRB(2)	0	Memory RMW Read Request
001	MADRB(1)	MADRB(0)	1	
001	0010LLLL	MADRB(2)	0	Memory Write Request
001	MADRB(1)	MADRB(0)	1	
001	DATAB(1)	DATAB(0)	2	
.	.	.	.	
.	.	.	.	
000	DATAB(N-3)	DATAB(N-4)	C-2	
001	DATAB(N-1)	DATAB(N-2)	C-1	
001	0011LLLL	MADRB(2)	0	Memory RMW Write Request
001	MADRB(1)	MADRB(0)	1	
001	DATAB(1)	DATAB(0)	2	
.	.	.	.	
.	.	.	.	
000	DATAB(N-3)	DATAB(N-4)	C-2	
001	DATAB(N-1)	DATAB(N-2)	C-1	
000	0100000B	IADRB(2)	0	Interconnect Read Request
001	IADRB(1)	IADRB(0)	1	
001	0101000B	IADRB(2)	0	Interconnect Write Request
000	IADRB(1)	IADRB(0)	1	
001	DATAB(1)	DATAB(0)	2	

Table F-2. Memory Bus Reply Message Formats

CCC	Upper MACD Byte	Lower MACD Byte	T	Request Message
2 0	15 8	7 0		
Read Replies				
010	DATAB(1)	DATAB(0)	0	Read Reply
111	IDLE	IDLE 1		1 or 2 Bytes
011	DATAB(1)	DATAB(0)	0	Read Reply
.	.	.	.	More than 2 Bytes
.	.	.	.	
011	DATAB(N-5)	DATAB(N-6)	C-3	
010	DATAB(N-3)	DATAB(N-4)	C-2	
011	DATAB(N-1)	DATAB(N-2)	C-1	
RMW Read Replies				
010	DATAB(1)	DATAB(0)	0	Memory RMW Read Reply
101	IDLE	IDLE	1	
011	DATAB(1)	DATAB(0)	0	Memory RMW Read Reply
.	.	.	.	
.	.	.	.	
011	DATAB(N-3)	DATAB(N-4)	C-3	
010	DATAB(N-1)	DATAB(N-2)	C-2	
101	IDLE	IDLE	C-1	
Other				
110	11111111	11111111	0	Write Acknowledge
111	IDLE	IDLE	1	
110	11111111	11111110	0	Memory RMW Write Acknowledge
111	IDLE	IDLE	1	
110	11111111	11111100	0	Memory RMW Locked Read Reply
111	IDLE	IDLE	1	

Table F-3. Memory Bus Notification Message Formats

CCC 2 0	Upper MACD Byte 15 8	Lower MACD Byte 7 0	T	Bus Notification Message
100 111	00000000 IDLE	DDDDDDDD IDLE	0 1	Interprocessor Communication
100 111	11111111 IDLE	11111111 IDLE	0 1	RMW Enqueue

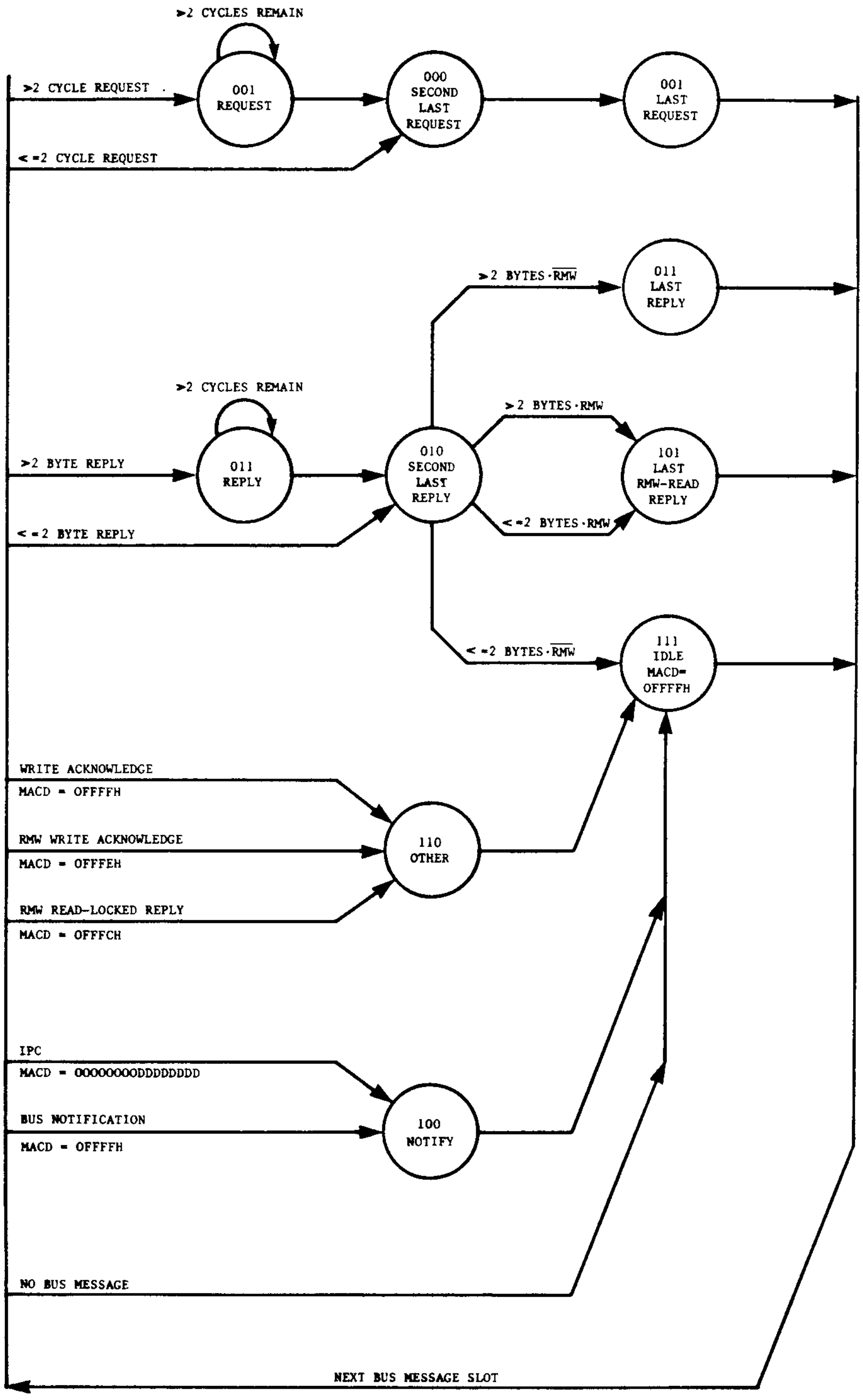


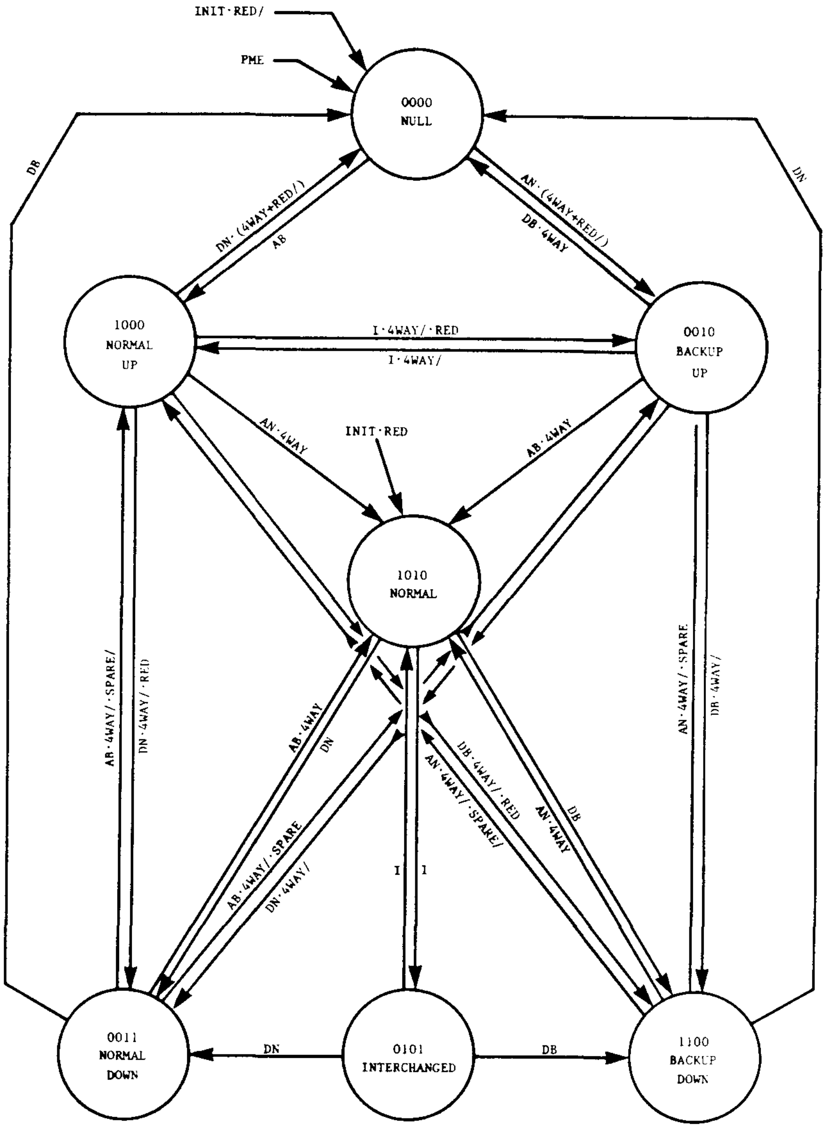
Figure F-1. State Diagram for MACD Bus Messages/Control Codes F-0004

BUS SWITCHING

The BIUs and MCUs in a system are capable of automatically switching memory buses when an error occurs. Each BIU and MCU contains a state machine that monitors the current state of the bus and of the error report logs to determine if and when to switch to an alternate bus. Interconnect components may be commanded to switch buses by means of the Attach, Detach, and Bus Interchange commands.

The three diagrams on the following pages present state transition diagrams that describe bus switching activity. In Figure F-2, the fully general version of the bus switching state machine is provided. Figure F-3 shows simplified versions of bus switching activity in a large continuous processing system and in a minimum continuous processing system. In Figure F-2, F-3, and F-4 the four-digit binary number inside each state element corresponds to the 4-bit bus state code located in the State register (BIU Register 06 and MCU Register 06).

Other configurations are possible, but they will not perform automatic bus switching. In these cases, bus switching must be performed under program control by direct access of the Bus State bits in the State register. This direct control of bus switching must be performed only when the interconnect system is in a quiescent state.



LEGEND: AN - ATTACH FROM NORMAL BUS
AB - ATTACH FROM BACKUP BUS
DN - DETACH FROM NORMAL BUS OR PERMANENT BUS ERROR
DB - DETACH FROM BACKUP BUS OR PERMANENT BUS ERROR
RED - REDUNDANT BUS IS ENABLED

I - INTERCHANGE BUSSES
SPARE - SPARE BUS AVAILABLE
4WAY - FOUR WAY INTERLEAVING
PME - PERMANENT MODULE ERROR
INIT - INITIALIZATION

Figure F-2. General State Machine for Memory Bus Switching F-0003

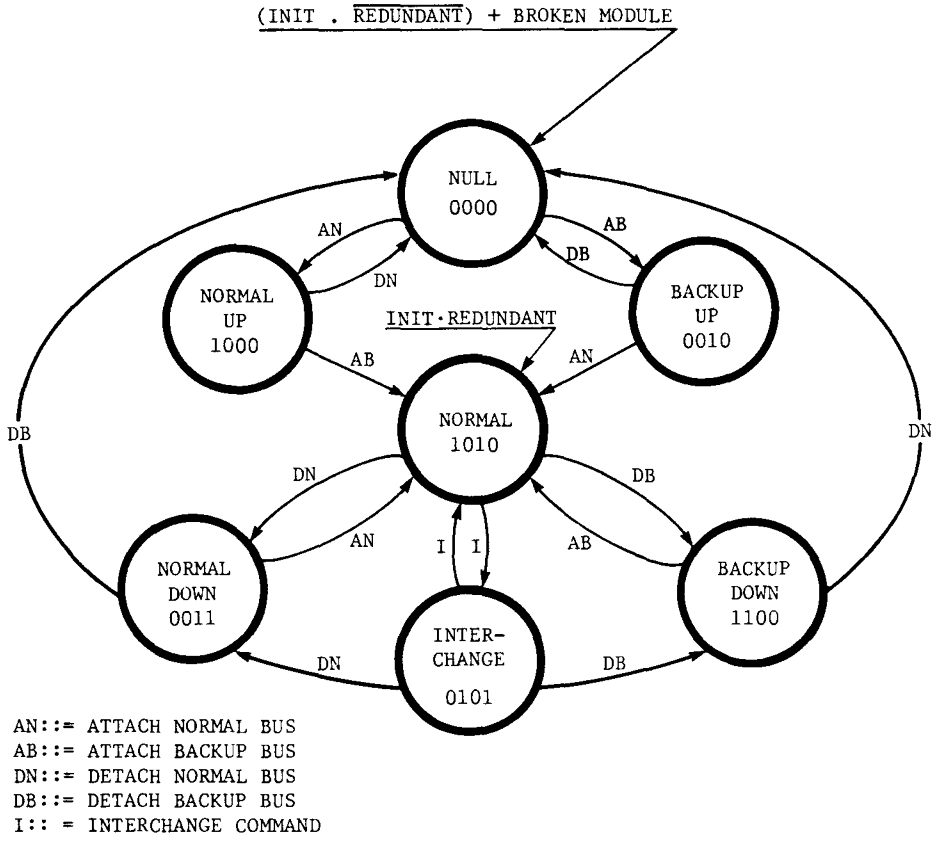
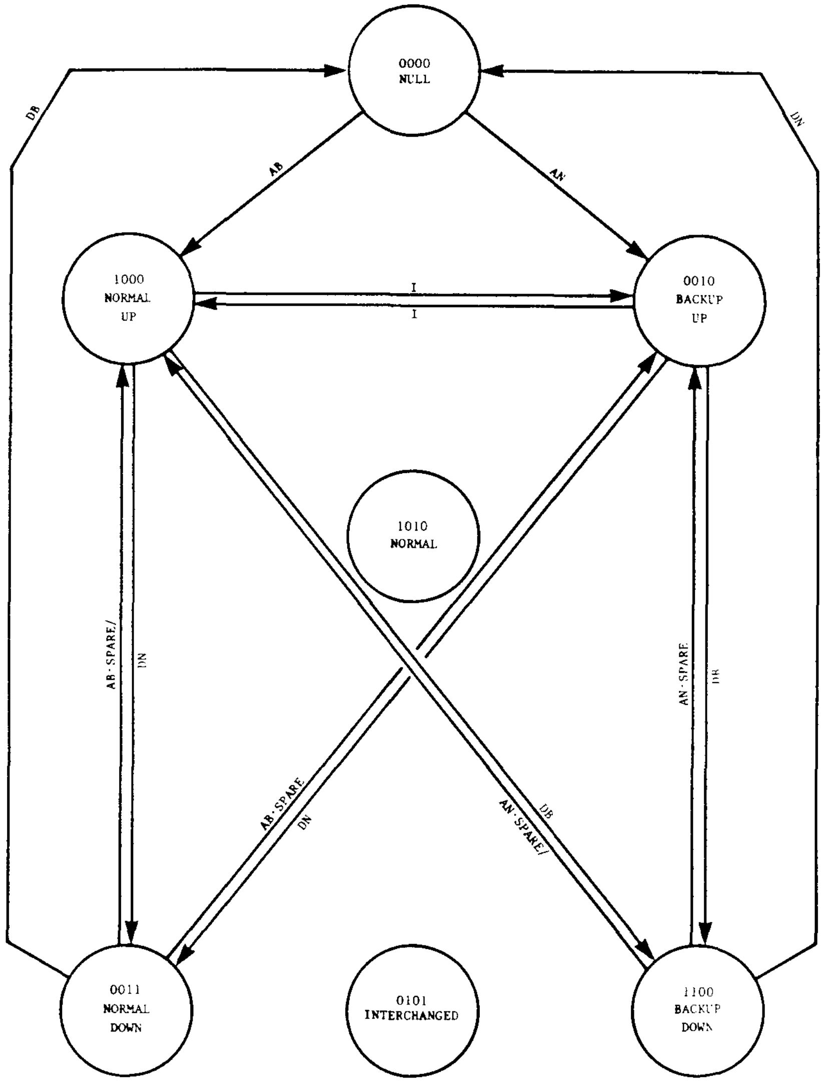


Figure F-3. State Machine for Large Continuous Processing System

F-0450



LEGEND: AN - ATTACH FROM NORMAL BUS
AB - ATTACH FROM BACKUP BUS
DN - DETACH FROM NORMAL BUS OR PERMANENT BUS ERROR
DB - DETACH FROM BACKUP BUS OR PERMANENT BUS ERROR
I - INTERCHANGE BUSSES
RED - REDUNDANT BUS IS ENABLED
SPARE - SPARE BUS IS AVAILABLE

2 BUSSES (0 AND 2)
1 GDP AND 1 IP
REDUNDANT BUS ENABLED
NO INTERLEAVING

Figure F-4. State Machine for Minimum Continuous Processing System

F-0002

intel

INTERCONNECT
ARCHITECTURE REFERENCE MANUAL
172487-001

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

____
____
____
____
____
____

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

____
____
____
____
____

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

____
____
____
____
____
____

4. Did you have any difficulty understanding descriptions or wording? Where?

____
____
____
____

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). ______________

NAME ______________________________ DATE ______________

TITLE ____

COMPANY NAME/DEPARTMENT ______________________________

ADDRESS ____

CITY ______________ STATE ______________ ZIP CODE ______________
(COUNTRY)

Please check here if you require a written reply. □

:'D LIKE YOUR COMMENTS . . .

s document is one of a series describing Intel products. Your comments on the back of this form help us produce better manuals. Each reply will be carefully reviewed by the responsible son. All comments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
SSO Technical Publications WW1-487
3585 SW 198th Ave
Aloha OR 97007

INTEL CORPORATION, 3585 S.W. 198th Avenue, Aloha, Oregon 97007 • (503) 681-8080

Printed in U.S.A./S001/1K/1282/AP