Intel iAPX 432—VLSI building blocks for a fault-tolerant computer

by DAVE JOHNSON, DAVE BUDDE, DAVE CARSON, and CRAIG PETERSON
Intel Corporation
Aloha, Oregon

ABSTRACT

Early in 1983 two new VLSI components were added to the iAPX 432 family of components. The 43204 Bus Interface Unit (BIU) and the 43205 Memory Control Unit (MCU) extend the logical flexibility and robustness of the 432 processors into the physical implementation of 432 systems. The BIU and MCU provide a range of fault-tolerant system options. The components provide comprehensive detection facilities for processor operations as well as for the operation of buses and memories. Recovery is possible from permanent as well as transient errors. Detection and recovery are done totally in the VLSI components; there is no need for additional TTL logic or diagnostic software. This range of fault-tolerant capabilities is achieved by replication of VLSI components. VLSI replication provides software transparent migration over the full range of fault-tolerant options without any penalties for unused fault-tolerant facilities in low-end systems.
INTRODUCTION

The range and nature of computer applications is changing dramatically during the 1980s. Software, maintenance, and downtime are dominating the life-cycle costs of computer systems. At the same time, application flexibility (to grow with the user's needs and to migrate the application as new opportunities occur) is becoming a key system requirement. Matching such expanding requirements are the unfolding capabilities of VLSI technology. VLSI offers high functionality at a low cost and high reliability per function. The 432 responds to this changing environment by applying VLSI technology to significantly reduce system life-cycle costs.

The iAPX 432 was introduced in early 1980. At that time the 432 consisted of three components, a two-chip (43201, 43202) Generalized Data Processor (GDP) and a single-chip (43203) Interface Processor (IP). These processors provide an object-based architecture and modular performance growth via transparent multiprocessing. The object-based architecture of the 432 provides a robust and flexible environment for cooperating, concurrent software systems. The 432 processors use a cooperative 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 43204 Bus Interface Unit (BIU) and the 43205 Memory Control Unit (MCU) extend the logical flexibility and robustness of the 432 processors into the physical implementation of 432 systems. The BIU and MCU allow the 432 hardware to modularly and transparently expand the processing power from 1 to 63 modules—processors or memories), bus bandwidth (from 1 to 8 backplane busses), and fault-tolerant capabilities of the system.

In the area of fault tolerance, the 432 provides a software-transparent, VLSI solution. The system may grow and migrate over a wide range of capabilities without any change to software or any cost burden for unused fault-tolerant functionality. At the low end, small low-cost systems that offer the inherent high reliability of VLSI and the ability to recover from transient errors may be configured. A range of fault-tolerant capabilities are available that increase the robustness of the error-detection and recovery facilities available in the system. At the high end, a 432 system can be configured to provide rigorous error detection over every aspect of the central system and recovery facilities for any single fault in the system. All of these capabilities are implemented in VLSI. No additional TTL logic is required, and no changes need to be made to the software system. Finally, the 432 provides an architecture that supports and encourages the development of reliable software systems. The 432 exploits VLSI technology to shift the burden of fault tolerance from software to VLSI.

The 432 provides a new level of flexibility for fault-tolerant applications that has been missing in the past. Traditionally, fault-tolerance has been considered a special and isolated set of applications. Applications were forced into an early decision either to pay for the more expensive fault-tolerant system or to choose the cheaper but less reliable standard system. The 432 eliminates this barrier by making expansion of fault-tolerant capabilities a simple and software-transparent configuration capability. The 432 is more than a fault-tolerant computer. The 432 offers cost effective solutions over a wide range of applications needs and can be modified at any time to meet the changing demands of the application.

FUNDAMENTAL PRINCIPLES

Three basic principles form the foundation for the implementation of the 432 fault-handling mechanisms. First, the fault-tolerant functionality is achieved by replication of standard VLSI components, not special purpose parts. 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 achieve effective use of VLSI technology. To be successful, each VLSI component must reach high-volume production. In the 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 that 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 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 significantly reduces the need for diagnostic probing as a method of fault isolation.
All communication in the 432 system is done over busses. 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 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 1 shows the four types of confinement areas in a 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 busses are the natural building blocks for 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 are checked as they flow 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 busses. 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 (GDP and BIUs) and placed in checker mode. Any disagreement is detected and signaled to the rest of the system. Thus, this duplicated module is 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 busses in the system, plus an interface to an external I/O subsystem. The interfaces to the memory busses 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 leave 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 fire wall between the central system and the I/O subsystem. The IP confinement area checks data as they leave the IP; the application HW and SW must check data that leave the I/O subsystem and enter the IP module. Error detection is performed by a duplicate checker as it was in the GDP module.

The memory confinement area includes the MCU, the RAM array, and the busses and support logic inside the module. A memory module has interfaces to two of the memory busses 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 and 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 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 a MCU. Error detection is performed by two interlaced parity bits that cover the control and address/data lines. Error detection for the arbitration lines is achieved by duplication. A timeout is used to detect protocol violations on the bus.

An example processor memory operation may help to clarify the operation of the confinement areas. (This example is shown graphically in Figure 2.) Assume a GDP makes a read request to a memory location. That request will be mapped through the BIU on the addressed memory bus. As the information flows onto the memory bus it will be checked by the BIU. If there was 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 con-

Figure 1—432 confinement areas
flows into the GDP module from the memory bus it is checked for correctness by the BIV before being used by the GDP module. As data flows onto 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. These confinement areas were formed by applying five different detection mechanisms: duplication, parity, hamming error-correction codes, timeouts, and loop back checks (used to detect errors in the TTL bus drivers). The only remaining question concerns checking the detection mechanisms. Some 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, which 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 busses that are totally independent from the busses used during normal operation. This error-reporting network is fully fault tolerant. No single failure in the error-reporting network can prevent the correct and timely reporting of an error in the system. This network of serial busses follows exactly the same topology as the busses used for normal operation. A failure on one of these busses is limited to one of the confinement areas discussed earlier. The failure of one error-reporting bus does not compromise the fault-tolerant capabilities of the system. Other error-reporting busses (in the other confinement areas) take over its reporting responsibilities. 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 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 432. Retry and single-bit correcting ECC codes are used for recovery from transient errors. Shadowed modules, backup busses, and spare memory bits are used for recovery from permanent errors. 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 that occur in dynamic RAMs.

For permanent errors: Every self-checking 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 of the state information in the module. This mechanism is known as module shadowing because the shadow is ready for immediate recovery should the primary fail, or vise versa. A fault-tolerant module is also called a quad modular redundant (QMR) module because the 432 VLSI components are replicated four times (two self-checking modules, with a master and checker in each module).

Each memory bus in the system may be paired with another memory bus. During normal operation the busses 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. 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 are retried and the MCUs 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.
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 the node becomes active and takes over operation for the faulty unit. After the resource switch is complete, all of the outstanding memory accesses are automatically 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. The software may choose to maintain full capabilities of the system and switch in a spare resource, or maintain fault tolerance and degrade performance by switching out the unit that no longer has a shadow, or maintain performance and run with an increased probability of failure by keeping the shadow unit in operation without bringing a spare on line. All of these configuration options are under software control and require no manual intervention or hardware changes. These policy decisions are carried out while normal system operation continues. Figures 3 and 4 show two examples of recovery operations.

**SUMMARY**

The 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 to provide recovery for only transient errors (no checkers for error detection or shadows...
for recovery). This functionality comes without extra cost in the VLSI interconnect system. To reduce maintenance costs and increase system availability, a system may use all of the detection mechanisms but may not add any extra recovery capability (add checker modules, but not form shadow pairs from the self-checking modules). Fully fault-tolerant systems (continuous operation in the presence of any single failure) are available to the user by adding the extra recovery capabilities. This expansion of fault-tolerant capabilities is accomplished solely by replicating VLSI components, not by the addition of new component types.

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 (either in cost, size, or performance) for the unused fault-tolerant capabilities. Increased levels of fault tolerance are achieved by replicating the 432 VLSI components. The hardware fault tolerance in the 432 is transparent to application software. The system's fault-tolerant capabilities may be changed without any changes to the application software system.

ACKNOWLEDGMENTS

The 432 fault-tolerant design would not have been possible without the support and design refinement by many talented people within Intel's Special Systems Operation. In particular, we would like to thank Tony Cornish (who was a co-architect of this work) and Bill Corwin for their important contributions to the design work.

REFERENCES
