Self-Checking Self-Repairing Computer Nodes Using the Mirror Processor

Yuval Tamir, *Member, IEEE*

Abstract—**Fault-tolerant systems often rely on self-checking computing nodes. Such nodes detect errors as soon as they occur, thus preventing the spread of erroneous information throughout the system. System performance and reliability is increased if the nodes can recover locally from most errors caused by transient faults without requiring system-level recovery. The circuitry added for concurrent error detection usually reduces performance. Using a technique called *micro rollback*, it is possible to eliminate most of the performance penalty of concurrent error detection. Error detection is performed in parallel with intermodule communication, and erroneous state changes are later undone. We report on the design and implementation of a VLSI RISC microprocessor, called the *Mirror Processor* (MP), which is capable of micro rollback. In order to achieve concurrent error detection, two MP chips operate in lockstep, comparing external signals and a signature of internal signals every clock cycle. If a mismatch is detected, both processors roll back to the beginning of the cycle when the error occurred. In some cases the erroneous state is corrected by copying a value from the fault-free processor to the faulty processor. We describe the architecture, microarchitecture, and VLSI implementation of the MP, emphasizing its error-detection, error-recovery, and self-diagnosis capabilities.**

Index Terms—**Micro rollback, self-checking modules, self-diagnosis, self-repair, VLSI RISC processor.**

I. Introduction

THE DESIGN of computer systems involves trade-offs between average performance, real-time performance, and reliability. In order to provide a high probability of meeting real-time constraints, the system is underutilized most of the time, leading to reduced average performance. In order to meet reliability requirements, fault tolerance is often necessary. The goal of our research is to design fault-tolerant systems that achieve high average performance as well as a high probability of meeting real-time constraints.

Fault-tolerant systems which include multiple processors often rely on self-checking computing nodes [14], [19]. These nodes detect errors as soon as they occur, thus confining the erroneous information to the failed node and minimizing the scope of recovery actions. This reduces system unavailability following an error, while recovery is in progress. Since transient faults are more likely to occur than permanent faults [2], system average performance and ability to meet real-time constraints can be increased if the computing nodes recover locally from most errors caused by transient faults without requiring system-level recovery.

In order to implement concurrent error detection for the self-checking nodes, checkers are usually connected in the communication paths between modules. The checkers reduce performance by requiring either longer clock cycles or additional pipeline stages. As we have previously described [21], [22], this performance overhead can be minimized if the checks are performed in parallel with intermodule communication. Each module processes its inputs immediately when they become available. If the data are erroneous, they are followed, after a delay of a few cycles, by an error indication. If the data processed by the receiver are later flagged as erroneous, any changes to the state of the system due to this information must be undone. Hence, it is necessary to back up processing to the state that existed just before the error first occurred. We call the process of backing up a system several cycles in response to a delayed error signal *micro rollback* [22].

We report on the design and implementation of a VLSI RISC microprocessor, called the *Mirror Processor* (MP), which can serve as a building block for self-checking self-repairing computing nodes [23]. The MP implements the instruction set of the Berkeley RISC II chip [11], [16]. The basic error-detection mechanism of the MP is duplication and comparison [5]. Following the scheme supported by several commercial microprocessors [4], [7], [8], two MP chips, a *master* and a *slave*, operate in lockstep, with the slave comparing their results every clock cycle. Local recovery from an error caused by transient faults is accomplished by undoing the erroneous state changes and repeating the clock cycles that resulted in the error. When necessary, the corrupted state in one MP chip is repaired by copying values from the other MP chip. The self-repair features are unique to the MP. Each MP is capable of micro rollback of up to four cycles. Hence, the latency for error detection does not require slowing down normal operation.

The MP chip has been laid out using MOSIS scalable CMOS design rules. The chip contains 52 644 transistors

Manuscript received March 5, 1991; revised August 29, 1991. This research was supported by the SDIO Innovative Science and Technology Office, managed by the Office of Naval Research, contracted to the Jet Propulsion Laboratory under task plan 80-2984; and by Hughes Aircraft Company and the State of California MICRO program.

The author is with the Computer Science Department, University of California, Los Angeles, CA 90024-1596.

IEEE Log Number 9104047.

and, with 2-μm technology, the size of the chip is approximately 8.4 mm $\times$ 6.7 mm. The operation of the chip has been checked using simulations at the circuit level, switch level, and register-transfer level. Based on these simulations, the chip will operate at a peak execution rate of 10 MIPS.

The basic techniques used in the MP have been presented in previous papers. The main contribution of this paper is in presenting how these techniques can be incorporated in a real implementation of a complete microprocessor. The MP uses a combination of techniques that, together, result in a practical building block for high-performance fault-tolerant systems. This paper describes the architecture and microarchitecture of the MP, emphasizing its error-detection, error-recovery, and self-diagnosis capabilities. We demonstrate the benefits of micro rollback in a real system, show that it can be implemented in a practical VLSI chip, and evaluate the overhead and design issues encountered.

Section II is a brief description of micro rollback and related terminology. The basic microarchitecture and timing of the MP are presented in Section III. The data path, with its special features for micro rollback, error detection, and state repair, are described in Section IV. We have found that much of the added design complexity of the MP, relative to a conventional RISC microprocessor, was due to the control unit. Section V describes the MP control unit. With most self-checking modules, there are difficulties in verifying that the self-checking features are operational once the module is integrated with the rest of the system. Section VI presents special instructions that were added to the MP in order to facilitate self-diagnosis. The VLSI implementation of the MP is described in Section VII. This includes the floor plan of the chip, the overhead for detection, repair, and micro rollback, as well as a brief description of the design process.

II. Micro Rollback

Micro rollback is based on the idea that a valid system state can be restored by rolling back to a previously saved checkpoint [13]. A micro rollback of a module (subsystem) consists of bringing the module back a few cycles to a state that it had reached in the past [22]. In order to be able to perform such an operation, it is necessary to save a "snapshot" of the state of the subsystem (*checkpoint*) at each cycle boundary [6]. Micro rollback restores the state of a subsystem by overwriting the current state with a "snapshot" taken in the past (Fig. 1).

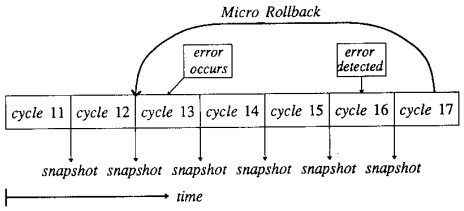


Fig. 1. Micro rollback of a module—restoring a saved snapshot.

Unlike traditional checkpointing and rollback [13], with micro rollback both checkpointing and rollback are performed entirely in hardware. This allows checkpointing to be performed in parallel with normal operation while recovery is performed in a few clock cycles. Such rapid checkpointing and rollback is essential in systems with real-time constraints, where long delays for recovery are intolerable. In [22] we presented a comparison between micro rollback and traditional instruction retry [3] as well as between micro rollback and schemes used for precise interrupts [6], [18]. Micro rollback is performed at a lower level, on the basis of clock cycles rather than instructions. This allows rollback to be executed at the logic level, without keeping track of instruction semantics and instruction pipeline conditions. As a result, the micro-rollback capability can be independently implemented in each module of a synchronous system. Building blocks that are capable of micro rollback can be interconnected in arbitrary ways to construct systems capable of micro rollback. Such flexibility is difficult to achieve if the semantics of rollback are tightly coupled to the specific function of each module.

Shedletsky [15] proposed the use of rollback of a few cycles for achieving self-checking with respect to permanent faults in modules which are self-testing but not fault-secure [1]. With such modules errors are likely to be detected within a few cycles after they occur so a rollback of a few cycles is likely to restore the system to a valid state. The goal of this scheme is not to reduce the performance penalty of adding checkers, but rather to deal with "imperfect" checkers. While this proposed scheme may be quite useful for permanent faults, it is not applicable to transient faults, which are more likely to occur [2]. There is no discussion in [15] of how the rollback will be performed.

The state of a module (subsystem) is the contents of all storage elements which carry useful information across cycle boundaries. When a rollback occurs, the number of cycles to be undone must be provided to a rollback controller. This number is called the rollback distance. One of the design parameters of a system with support for micro rollback is the rollback range—the maximum rollback distance that modules in the system must support. The rollback range is determined by the number of stored snapshots.

III. The Architecture of the Mirror Processor

The choice of the basic processor architecture for the MP was guided by several considerations: 1) since we were not interested in instruction set design, it was preferable to use an existing instruction set; 2) in order to have a chance of successful implementation in an academic environment, the basic processor architecture had to be simple; and 3) in order to be a convincing "proof of concept," the performance of the processor could not

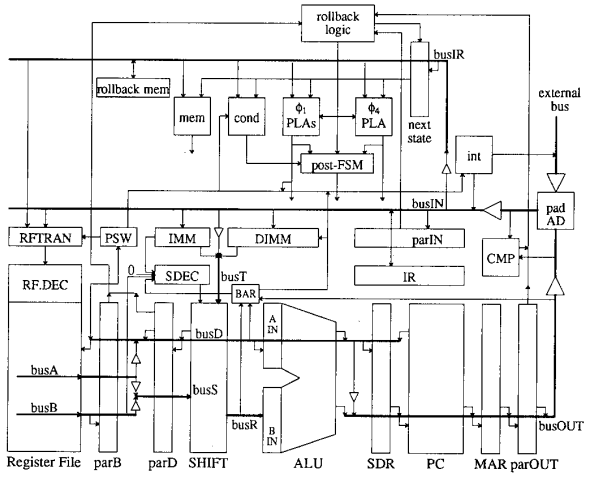


Fig. 2. The Mirror Processor. All the modules below busIN are part of the data path. The control unit is shown above busIN.

be orders of magnitude below the performance of contemporary microprocessors. Based on these considerations, we chose to use the Berkeley RISC II processor [11], [16] as the basis for the MP. The RISC II was the basis of the popular Sun SPARC architecture and is similar to other "reduced instruction-set computers." It is a load/store architecture, where only explicit load and store instructions access memory. All ALU and shift operations obtain their operands from the register file and store their results back to the register file. One-, 2-, and 4-byte integers are supported. The memory is accessed as a single, flat 4-gigabyte segment. The register file has multiple register banks to support fast procedure calls [11]. In the MP, the register file consists of four banks of sixteen 32-b registers for procedure stack frames plus ten global general-purpose registers.

The data path of the MP, shown below the busIN bus in Fig. 2, is almost identical to the original RISC II data path [16]. The differences are related to the fault-tolerance features of the MP and will be described later. The memory interface is over a multiplexed data/address bus, which is used to reduce the number of pins and thus the cost of fabrication and testing. Register–register instructions are executed at a rate of one instruction per cycle with the bus used every cycle to fetch the next instruction. Due to the multiplexed external bus, two cycles are needed to execute load and store instructions.

Pipelining is used in order to maintain an execution rate of one instruction per cycle. Four nonoverlapping clock phases are used to sequence operations within each cycle. As shown in Fig. 3, the multiplexed external bus is used to transfer addresses during ϕ_2 and ϕ_3, and transfer an instruction or data in ϕ_4 and ϕ_1. A typical ALU instruction requires reading two registers from the register file, performing an ALU operation, and writing the result back to the register file. The delayed write buffer [22], which is added for micro rollback (Section IV), allows the result to be written to the register file module in the same phase (ϕ_1) that the operands for the next instruction are read.

IV. Fault-Tolerance Features in the Mirror-Processor Data Path

The MP was designed for use in self-checking nodes where error detection is accomplished by error-detecting codes (EDC's) in the memory and duplication and comparison [5], [7], [19] for the processor. Two MP chips are used in each self-checking node. In order to reduce the board chip count and complexity, a comparator for all the key output pins is implemented in the MP chip. Based on the value of a dedicated input pin, the chip operates as a master or a slave [4], [7], [8]. The slave operates in lockstep with the master, performing the same operation at every clock phase. However, whenever the master produces an output (data or address), the slave instead reads the value from the bus and compares it to its own internally generated value. A mismatch indicates an error.

With the scheme above, there is latency inherent in the error-detection process. Specifically, once the master generates some value, the value must be driven to the output pins, received by the slave, and compared with the slave's internally generated values. If a mismatch is found, the error signal must be driven from the slave to the output pin and back to the master. Hence, an error can corrupt the processor state before it is detected. The MP uses micro rollback to undo such state changes and restore the processor to its state prior to the instruction when the error first occurred.

A. Support for Micro Rollback

Efficient implementation of micro rollback involves delaying commitment of state changes until the new values are known to be correct (checking is complete). For the MP, it is sufficient to delay commitment by four cycles since the latency of the error-detection mechanisms used is less than four cycles. As described in [22], this is accomplished using delayed write buffers (DWB's), as shown in Fig. 4. Updates are made by writing to the left stage of the DWB and setting the corresponding *valid* bit. The DWB is shifted right every cycle. After four cycles, the updated value is shifted into the "permanent storage" for the register (the rightmost register). Updating of the permanent storage is performed only if the rightmost valid bit is set. On a read, the *select* circuitry determines the most recent (the leftmost) value for which the valid bit is set. A rollback of n cycles is accomplished in a single phase by clearing the first (leftmost) n valid bits so that a subsequent read will obtain an older value.

In [22] it was shown that the basic technique described above can be used for the register file. The "permanent storage" consists of 74 registers, as described in Section III. In the DWB, the select circuit includes a *tag* with the register number corresponding to the value stored in the data part. On a write to the register file, the new value and register number are written into the DWB. A read

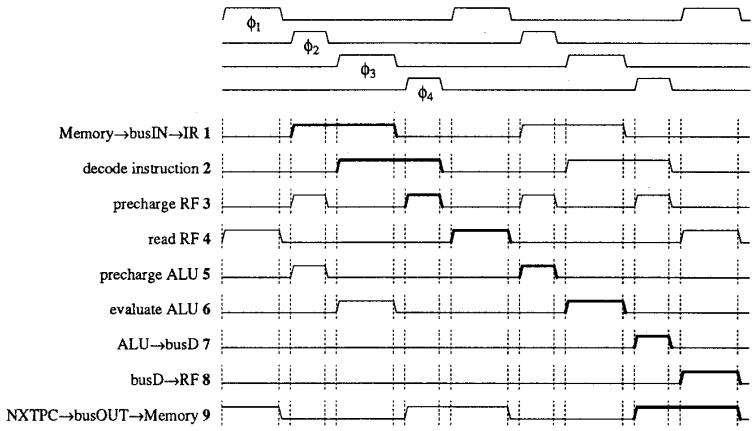


Fig. 3. The timing of register–register ALU instructions. The instruction is fetched in one clock cycle and executed in the following clock cycle. The result is stored in the register file in phase 1 of the cycle following the execution cycle.

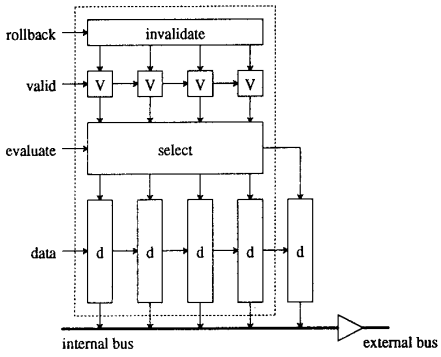


Fig. 4. A single register with delayed commitment of updates, using a four-stage delayed write buffer (DWB). Writes modify the leftmost stage of the DWB. Recent updates are undone by clearing the valid bits. Reads obtain the most recent valid value.

from the register file involves comparing the register number to all the tags to determine if a recent value should be obtained from the DWB. Since the look-up in the DWB is performed in parallel with the permanent register file read, there is little additional delay [22].

As in the Berkeley RISC II processor, there are three registers used for storing the next, current, and last values of the program counter (PC) [9]. In the RISC II design these registers are organized as a small FIFO queue and the following transfers occur during each cycle:

$$\text{new value} \rightarrow \text{next\_PC} \rightarrow \text{PC} \rightarrow \text{last\_PC}$$

Micro rollback of the PC unit could be supported by treating the three registers as individual state registers (Fig. 4). However, this would result in high area overhead of 12 DWB stages. For the MP we developed a special, more efficient, mechanism for supporting micro rollback in the PC. As explained below, this mechanism takes advantage of the original FIFO organization of the PC unit.

The basic organization of the MP PC unit is shown in Fig. 5. As with the single register (Fig. 4), there are four valid bits, corresponding to the four DWB stages. A rollback of n cycles is accomplished by clearing the n leftmost valid bits. The key difference between the PC module and the module of a single register is the selection circuitry. Depending on whether the read access is to the next_PC, PC, or last_PC, pass transistor logic, shown in Fig. 5, is used to select the registers corresponding to the first, second, or third valid bits, respectively. Following a rollback, when several of the valid bits are cleared, this selection process may select one of the permanent registers (npc, pc, or lpc) instead of a value in the DWB.

Starting with a microarchitecture that does not include support for micro rollback, it is clear that DWB's are needed for registers that hold values across cycle boundaries [22]. Many VLSI implementations depend on the ability to store values for a few phases or cycles using the inherent parasitic capacitance of the circuits for dynamic storage. For example, once a value is asserted on an internal bus, it may be assumed that the value will remain stable for several cycles, even if the driving circuit is disconnected. Whenever such dynamic storage is used to store values across cycle boundaries, support for micro rollback requires the ability to restore the value when rollback is performed.

The problem of rolling back dynamic storage was encountered in the design of the MP. One case led to the addition of a memory address register (MAR), which was not part of the original microarchitecture. As shown in Fig. 3, the instruction fetch involves driving an address to busOUT and to the external bus during ϕ_4 of a cycle and ϕ_1 of the next cycle. This value is obtained by reading next_PC during ϕ_1 and incrementing it during ϕ_3 of the first cycle. When a rollback is performed to the cycle boundary (between ϕ_4 and ϕ_1), the instruction fetch that was in progress at the time must be re-executed. This requires the address of the instruction to be restored to the external bus. Furthermore, the processor must allow the memory the same amount of time to perform the restored instruction fetch as the time to perform a normal instruction fetch. In order to meet these requirements, a memory

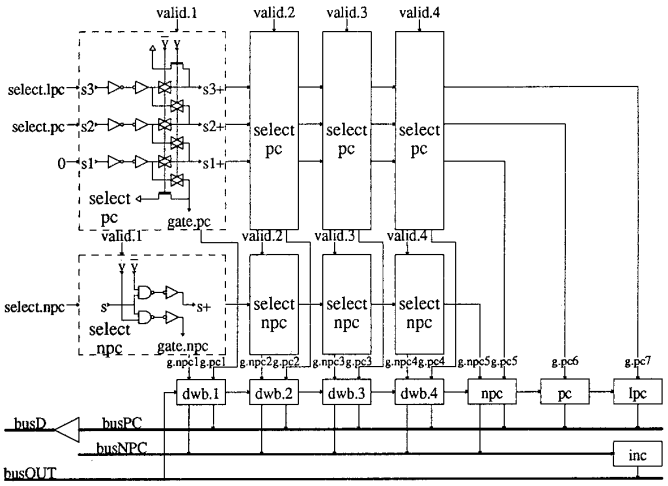


Fig. 5. Support for micro rollback in the program counter. Three values are available at any instant: next_PC, PC, and last_PC. A four-stage DWB is used to delay commitment of updates to a three-stage FIFO queue of permanent storage registers.

address register, with the structure shown in Fig. 4, is connected to busOUT. The value on busOUT is written into the MAR during ϕ_1 of every cycle. During ϕ_3 of the rollback cycle, the appropriate number of MAR DWB stages are invalidated and the most recent remaining MAR value is read. This value is driven onto busOUT, thus correctly restarting the instruction fetch operation.

Two other registers with DWB's had to be added to the data path of the MP to support micro rollback: 1) the instruction register (IR) is used for restoring the instruction that was fetched in the cycle preceding the cycle boundary to which rollback is performed, and 2) the store data register (SDR) is used to restore the data of a store instruction if the rollback is to the boundary between the two cycles of the store. A DWB (as in Fig. 4) is also required to support micro rollback of the processor status word (PSW) register. It should be noted that the external memory (or cache) must also include a DWB [22] so that recent stores to the memory can be rolled back to maintain a consistent state with the processor.

B. Support for Error Detection and Error Recovery

As discussed earlier, the MP was designed as a building block of self-checking nodes where the primary error-detection mechanism is duplication and comparison. The basic support for this mode of operation is the ability of the chip to operate in slave mode, where it does not drive values onto the external bus. Instead, the slave chip compares the values on the external bus, generated by the master, to internally generated values. In addition to the external bus lines, the comparison includes the memory control signals, a mode bit (system/user), and the interrupt acknowledge signal.

Effective detection of errors in the external memory (or cache) can be implemented using EDC's at a lower cost than using duplication and comparison. Hence, the master and slave processors share (read from) the same memory (or cache). The MP uses single-bit parity on the external bus and memory for error detection. The external bus is thus 33 b wide; all addresses and data generated by the processor include a parity bit. A "compressed" tree of static XOR gates [26] connected to busOUT (parOUT) is used to generate the parity. Data and instructions read from the external memory must include a parity bit. The parity of these values is checked by a similar circuit (parIN).

The two techniques described above are sufficient if the only requirement is error detection. However, one of the goals of the MP is to be able to *recover* from all errors caused by a single transient fault in the processors. When an error is detected, both processors (master and slave) roll back to the cycle boundary preceding the likely cause of the error. In the MP, both of the above errors require a rollback of two cycles. If the error is incorrect parity on the value read from the external bus, the rollback results in re-executing the memory read operation. If the error is a mismatch in the comparison, for example due to a transient fault in one of the ALU's, the two-cycle rollback results in the re-execution of the ALU operation. Unfortunately, the above error detection schemes and micro rollback are not sufficient for recovering from *all* errors caused by single transient faults. One problem is caused by the potential for long latencies in detecting errors. For example, if the instruction executed is a register–register operation, a transient fault in one of the ALU's can result in an incorrect value stored in the register file. Many cycles later, a store of that register to memory will bring the value to the master/slave comparator and an error will be detected. At this point, a rollback will not restore a valid state and there is no simple way for the system to recover.

Error-detection latency can be reduced by including in the master/slave comparison internal values that are normally not accessible from the pins. In particular, it can be

useful to compare the ALU result as it is stored into the register file. In order to reduce the number of pins used to facilitate the comparison of internal values, a much smaller signature [10] of the internal values can be used. As shown in Fig. 6, one possible way to generate a 4-b signature is to use interleaved parity, where each signature bit is the parity of the set of bits consisting of every fourth bit of the original data [26]. A comparison of such signatures will detect all single-bit errors, as well as adjacent bit errors, and numerous other multiple-bit errors. Compact high-performance VLSI implementation of this signature generation circuitry is possible using chains of switching cells [17] to implement the multiple-input XOR for each signature bit [26]. In the MP, this basic technique is used to generate, every cycle, a 4-b signature of the value on busD (32-b ALU or shifter result), the 7-b "physical" register number for the destination register of the operation, a bit indicating whether a value is written into the register file (the valid bit for the DWB), the 11 b of the PSW, and all four state bits from the controller. The 4-b signature is driven, by the master, onto output pins and is included in the master/slave comparison every cycle.

The above use of signatures does not ensure recovery from all possible transient faults in the processor. Specifically, a transient fault can invert the value of a bit in the register file. Many cycles later, a read from the register file will obtain the corrupted value. The corrupted value will, of course, cause a mismatch between the master and slave so that the error will be detected. A rollback of a few cycles will not correct this error and there is no way to determine which processor, the master or slave, has the correct value. In order to allow the two processors to determine which one has the corrected value, a single parity bit is stored with each register in the register file. Whenever the register file is read, the parity is checked (parB and parD in Fig. 2). If a parity error is detected, a rollback of one cycle is initiated. In addition, the master and the slave communicate to each other the results of the respective parity checks using dedicated pins. Following rollback, if the results are clear regarding which processor has the corrupted data, two cycles are used to transfer the value from the fault-free processor to the faulty processor. If both processors detect parity errors in the same register, no state repair is performed following rollback (see Section V). Since a transient error in the register file decoder can cause a register to be stored in the wrong location, the parity stored with each register is calculated over the physical register number as well as the data. This error will be detected when there is an attempt to read this register. The state repair mechanism will allow recovery from this error.

The state repair mechanism with the dedicated error detection, as used for the register file, is not required for any other register on the MP chip. The reason for this is that the other registers are all modified *every cycle*. Since there is an update every cycle, all stages of the DWB's of these registers generally contain valid values. If an error occurs due to a corrupted value in one of these registers (e.g., the value of a condition code bit in the PSW is inverted, causing a conditional branch to behave incorrectly) the resulting micro rollback will invalidate the corrupted DWB stage. Thus, when the instruction is re-executed, a valid value will be obtained from the DWB.

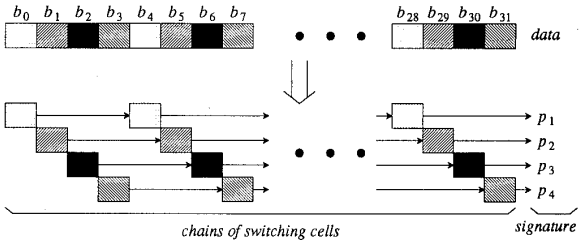


Fig. 6. Generating a "signature" of a 32-b word using four interleaved parity bits. The parity bits are computed using four interleaved chains of switching cells.

V. The Mirror-Processor Control Unit

The Berkeley RISC II processor was characterized by a simple small control unit [9], [16]. A single PLA was used to decode the opcode, producing 39 control bits, which were ANDed with different clock phases to produce approximately 100 control signals. The MP data path is more complex than the RISC II data path due to the support for micro rollback, error detection, and state repair. Hence, the number of control bits required by the MP is approximately double the number of control bits in RISC II. Furthermore, many of the control bits are dependent on rollback and repair signals in addition to the opcode. As a result, the MP control unit (Fig. 7) is significantly more complex than the RISC II control.

Instructions are read from the external bus through busIN and are latched onto busIR. A four-state finite state machine keeps track of whether the processor is executing a normal instruction, executing the second cycle of a two-cycle instruction, or performing the first or second cycle of state repair. The next-state logic block computes the next state based on the incoming opcode and on the repair signals generated by the rollback logic. The next state, plus the opcode and the repair signals, are sent to three PLA's that generate the majority of the control signals used by the data path.

Fig. 8 shows the sequence of operations involved in micro rollback, starting from ϕ_4 of the normal cycle preceding the rollback cycle. Each module in the rollback domain may detect an error by ϕ_4 of a normal cycle. Specifically, the MP slave determines the results of the master/slave comparison during ϕ_3 of each cycle and the results of the parity checks in both MP chips are determined during ϕ_4 of the cycle. In the simplest case, if a comparison or busIN parity error is found, the chip requests a rollback of two cycles. If a register file (parB or parD) parity error is found, a rollback of one cycle is signaled. If an error is detected, at the beginning of ϕ_1 of the fol-

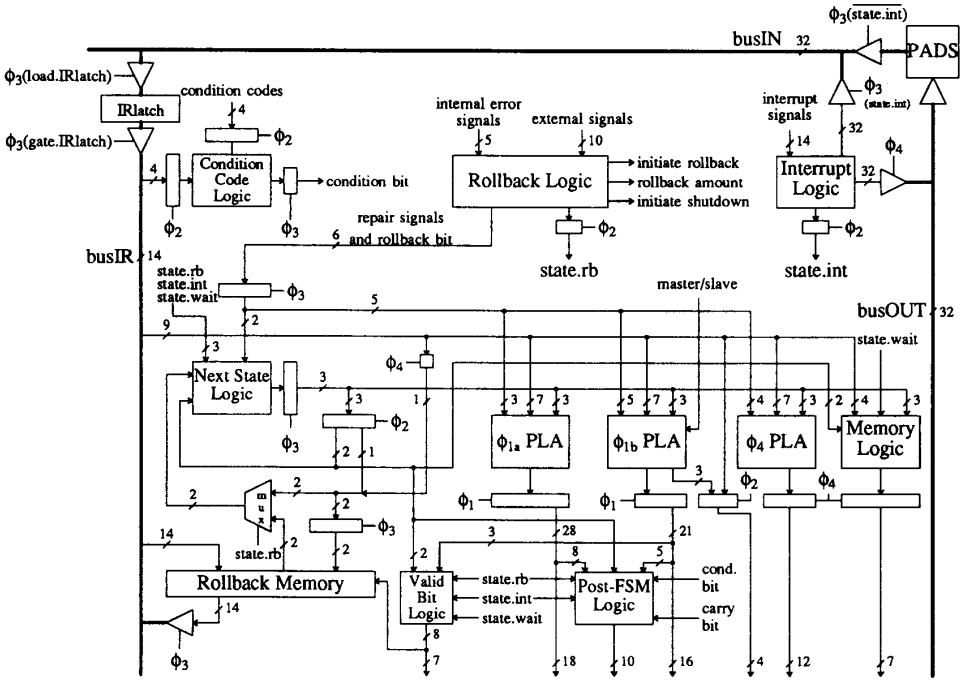


Fig. 7. The MP control unit. Most of the control signals are generated by the three PLA's. Where necessary, random logic is used for speed.

lowing cycle the module pulls down the rollback line and pulls down the rollback-amount lines according to the number of cycles that it needs to roll back. All four lines are connected to all the modules, which are part of the same rollback domain. Each module connects to these lines through a bidirectional open-drain pad driver. The lines are held high by external pullup resistors. During ϕ_2 of every cycle, the control reads the value on the rollback pin and the three rollback-amount pins and determines whether the current cycle is a rollback cycle as well as the number of cycles to be rolled back.

Since the system is synchronous, when there is a rollback all the modules in the rollback domain must roll back the same distance. If several modules simultaneously request rollbacks by different amounts, the entire system must determine the maximum rollback distance requested and roll back by that amount. The maximum rollback distance requested is determined by a straightforward implementation of the Futurebus arbitration protocol [24]. By ϕ_2 of a rollback cycle, the rollback distance for all the modules is available on the rollback-amount lines.

During a normal cycle, a maximum of two values are read from the register file of each processor onto the internal buses, busA and busB (Fig. 2). Four pins on the chip are dedicated to coordinating state repair between the master and slave. They signal possible parity errors in the values read from the register file. Two of the pins are driven by the master and indicate possible parity errors in the values it reads from its register file. The other two pins are driven by the slave based to the parity checks of the values read from the slave's register file. These four signals are set during ϕ_1 of a rollback cycle and are read during ϕ_2.

During ϕ_3 of the rollback cycle, the appropriate valid bits in the various DWB's are cleared in order to perform the rollback. At the same time, each processor's control unit independently determines whether state repair should be initiated. If none of the repair bits are set, no repair is initiated. If one of the processors detected an error in the value read on busA while the other did not, the busA repair will occur regardless of possible errors detected in the values read on busB. If both processors detect errors on busA, a repair of busA is impossible and none is attempted. If a busA repair is not needed, a busB repair may be initiated, as appropriate. Two points should be noted: 1) if repair is needed for both busA and busB, only the busA repair will be done and then normal operation will resume. When the instruction is re-executed following the repair, the busB error will be detected and, since there is no error on busA, the busB repair will be done. 2) If both the master and slave detect an error on the same internal bus, no repair will be done. However, rollback will be done so that the operation will be retried. If the error in at least one of the processors was not permanent (e.g., a transient on an internal bus), at least one of the values will be correct when the instruction is re-executed.

Fig. 9 shows the sequence of operations involved in state repair, starting from the rollback cycle preceding the first repair cycle. During ϕ_1 of the first repair cycle, both processors reread the same registers that were read in the

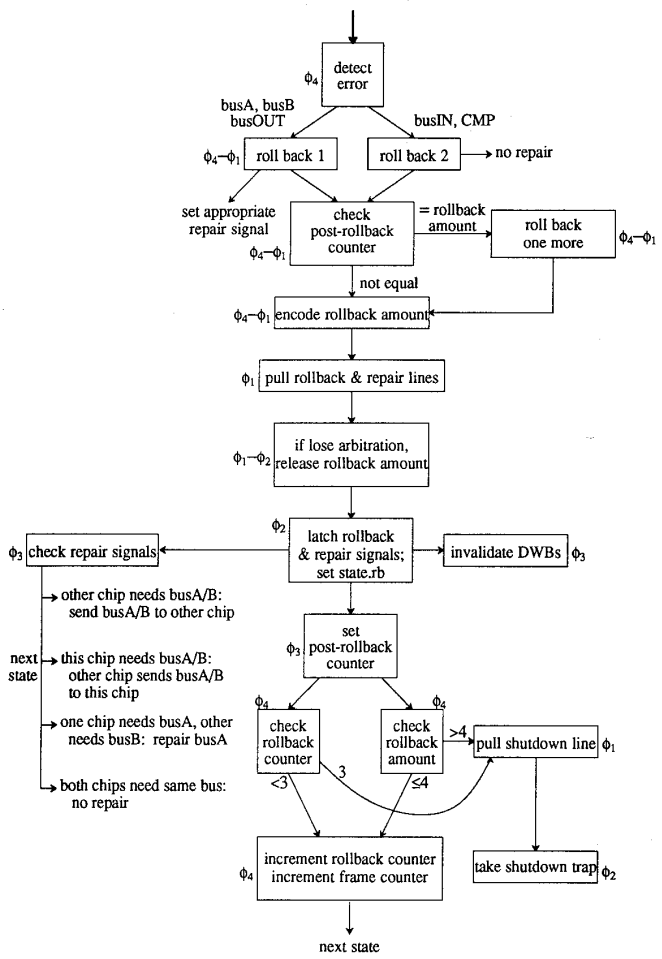


Fig. 8. The sequence of operations for micro rollback.

cycle preceding the rollback cycle. During ϕ_3 and ϕ_4, the register containing the value to repair is then gated onto busD through the shifter. During ϕ_2 of the second repair cycle, the processor with the correct value assumes master mode. Then, during ϕ_2 and ϕ_3, the master gates busD onto busOUT and the address/data pins while the other processor (the slave) reads the correct data in from the address/data lines. The slave gates the correct value through the shifter during ϕ_3 and finally to the register file during ϕ_4 and ϕ_1 of the next (normal) cycle. During ϕ_4 of the second repair cycle, both processors restore their respective master/slave modes and resume with the instruction that was rolled back to.

Fig. 10 shows the rollback and repair controller, which is part of the MP control unit. It controls the operations described above and includes many features for handling multiple faults and system-level recovery from massive errors. The rollback controller consists of two parts. The first part monitors the error signals generated by the parity checkers and comparators and pulls the external rollback line if an error is detected. It also determines the distance to roll back and recognizes conditions where local recovery is not possible (see below). The second part of the rollback controller monitors the external rollback, rollback-amount, and repair lines and sets internal rollback signals.

As discussed earlier, the MP was designed to recover from errors caused by single transient faults. However, several features were added to allow recovery from some multiple faults. For all the registers the DWB storage is implemented using dynamic latches. Hence, this storage is susceptible to transient faults. If the value in a DWB stage is corrupted, it is possible for a rollback to restore an erroneous state in the master or slave. This will be detected one or two cycles later, triggering a rollback. Unfortunately, without a special mechanism for dealing with this case, the second rollback may restore the erroneous state. In order to prevent this situation, the rollback controller includes the post-rollback counter (Fig. 10), which counts the number of cycles since the last rollback until it exceeds three cycles. Based on the value of this counter, the internal rollback block can determine whether

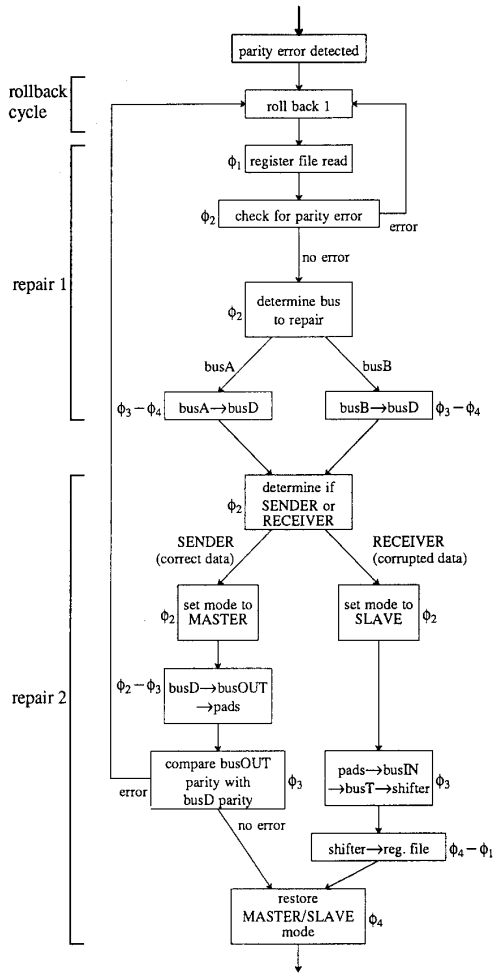


Fig. 9. The sequence of operations for state repair.

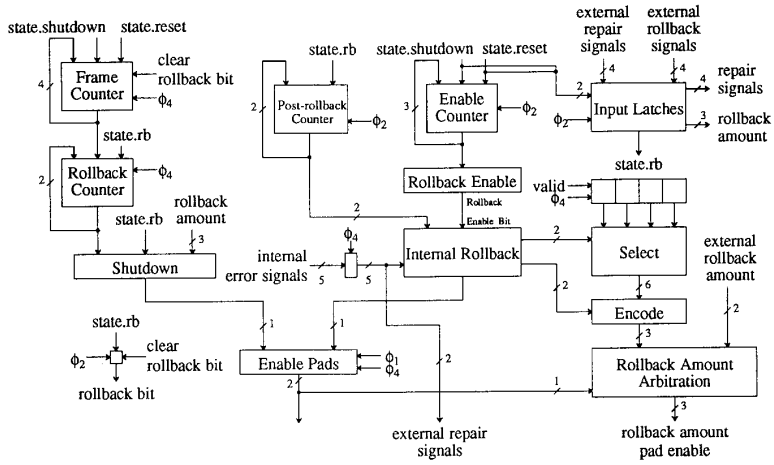


Fig. 10. Rollback and repair controller.

a rollback will restore the state just restored, several cycles earlier, by a previous rollback. If this situation is detected, the select logic is used to determine how many DWB entries must be invalidated in order to invalidate the state restored by the previous rollback. This determination is done based on a 4-b shift register, into which a 1 is shifted every normal execution cycle. During rollback of n cycles, the n most recent entries in this shift registers are cleared. Hence, the shift register contains a record of which of the last four cycles was a normal execution cycle that has not been rolled back.

It is possible for the master/slave comparison to detect errors from which recovery is not possible using micro rollback and the state repair mechanism discussed above. For example, if 2 b in one of the registers of the register file are inverted, a store of that register will bring the erroneous value to the output pins, causing a master/slave mismatch to be detected. In this situation, the parity checks on values read from the register file cannot indicate which register file is at fault. The mismatch will trigger a rollback. However, when the instruction is re-executed, the same error will be repeated. Without some additional mechanism, the processors will continuously execute the same two cycles followed by a rollback. No repair will be made and there is no way for the node to participate in system-level recovery that might allow normal operation to resume.

In order to better handle the situation of useless repeated rollbacks, the node must have the capability of detecting when local recovery is impossible. In the MP, this is handled by detecting that the fourth rollback is being attempted within the last 16 cycles and causing the processor to trap to an error handling routine instead of performing the rollback. In the rollback controller, the frame counter is a simple 0 to 15 counter which is incremented every cycle. The rollback counter counts the number of rollbacks within the current frame. The rollback counter is cleared whenever the frame counter reaches 0. If a rollback is initiated and the value of the rollback counter is 3, the shutdown logic initiates a shutdown trap. As with any interrupt or trap, the shutdown trap causes the processor to begin executing code in a predetermined address. The code stored in this address can, for example, perform self-diagnosis and then, if the node is operational, integrate the node back with the rest of the system. In a multiprocessor system this self-resetting capability [19] is essential since system-level recovery procedures require active cooperation from every operational node.

VI. Support for Periodic Self-Diagnosis

In any design of self-checking modules, there is the problem of preventing latent faults from remaining undetected for long periods. If faults remain undetected, multiple faults can eventually exist in the system simultaneously and cause the concurrent error-detection mechanism to fail, leading to an undetected error. One solution to this problem is for the system to periodically perform self-diagnosis whose purpose is to flush out latent faults. Generally, the probability of multiple faults occurring between diagnosis runs can be reduced to the required level by adjusting the frequency of the self-diagnosis runs appropriately. In a general-purpose processor, self-diagnosis can be accomplished by an operating system that periodically suspends normal execution and runs programs that were designed specifically to exercise all the features of the processor. This approach can be quite successful for simple RISC processors, where relatively short functional tests can be expected to achieve high-fault coverage [25].

With the MP, there is a special problem of how to test the circuitry for error detection, micro rollback, and state repair. This circuitry is designed to be transparent to the application and is generally not exercised unless there is an error and rollback is initiated. Special hardware and dedicated instructions were added to the MP in order to permit self-diagnosis of the error-detection, repair, and rollback circuitry. The new instructions are all *privileged* instructions which can only be executed in *kernel mode*. An unusual feature of these instructions is that they perform *different* actions on the master and slave processors. Furthermore, a primary consideration in the design of these instructions was to minimize the complexity of hardware modifications needed to support them. Low priority was given to the generality or "elegance" of these instructions.

The additional instructions provide two key facilities: 1) the ability of code to force apparent errors by storing incorrect parity or performing a different operation on the master and slave, and 2) the ability of code to determine whether a micro rollback has occurred since a flag was last cleared. A single rollback bit register was added to the control unit. This register can be explicitly cleared (see below) and is set to 1 every time there is a rollback. The dedicated instructions for self-diagnosis are as follows:

Clear Rollback Bit: clrrbm clrrbs

Clears the rollback bit in the rollback controller. **clrrbm** clears the rollback bit on the master, while **clrrbs** clears the bit on the slave.

Add with Bad Parity:

addbpm *S1,S2,Rd* **addbps** *S1,S2,Rd*

Functions as a normal **add** instruction, except that an *incorrect* parity bit is stored in the destination register of one of the processors. **addbpm** stores the bad parity in the master, while **addbps** stores the bad parity in the slave. This instruction is used to force parity errors on busA and busB.

Jump if Rollback Bit Is Set:

jmprbm *Rs1,Rs2,Rd* **jmprbs** *Rs1,Rs2,Rd*

If the rollback bit is set, a PC-relative jump is performed, using the contents of register *Rs2* as the offset for the jump, while, at the same time, the contents of register *Rs1* are stored in the destination register *Rd*. If the rollback bit is not set, the branch is not taken, and

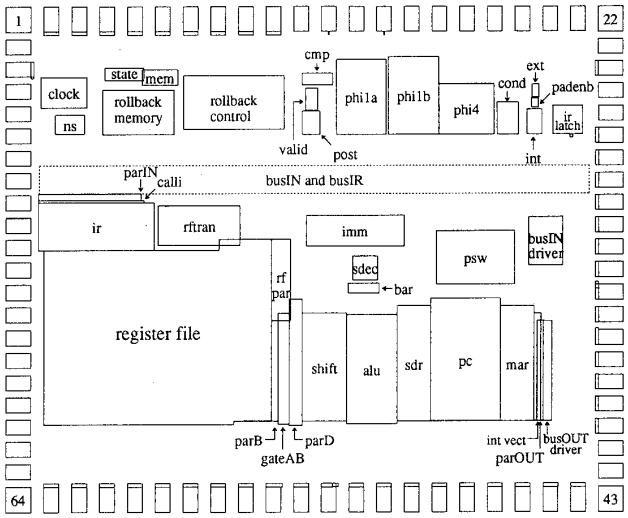


Fig. 11. The floor plan of the complete MP chip. This floor plan was extracted from the actual layout. All modules are drawn to scale. The "white space" is occupied by interconnections between modules and to the pads.

either *Rs*1 or *Rs*2 is gated onto busD and stored in *Rd*, depending on the instruction and mode: for **jmprbm,** the master stores *Rs*1, and the slave stores *Rs*2; for **jmprbs,** the slave stores *Rs*1, and the master stores *Rs*2. Since the branch offset can have only a few limited values, there are two separate instructions to allow both the master and the slave to gate *any* value onto busD in order to exercise the busD state compression logic. This instruction is used to force state compression comparison errors as well as to verify that a rollback has occurred.

Store Bad Data: strbdm *Rs,X* **strbds** *Rs,X*

Similar to a normal PC-relative store instruction. If the rollback bit is set, both processors will store the contents of *Rs* into location *X*; if the rollback bit is cleared, one of the processors will store the contents of the MAR instead. For **strbdm,** the master will store the bad data, while for **strbds,** the slave will do so. This instruction is used to force a comparison error on the data portion of a memory write.

Load with Bad Parity: ldrbpm *X,Rd* **ldrbps** *X,Rd*

Similar to a normal PC-relative load instruction. If the rollback bit is set, both processors will load *Rd* with the contents of location *X*; if it is cleared, one of the processors will gate the loaded data onto busIN with an incorrect parity bit. For **ldrbpm,** the master will load a bad parity bit, while for **ldrbps,** the slave will do so. This instruction is used to force a parity error on busIN.

As an example of using these special instruction, we consider testing of the handling of a parity error in the register file. The test program can use the **addbpm** or **addbps** instructions to store incorrect parity in a register of the master or slave, respectively. The rollback bits of the master and slave are cleared using **clrrbm** and **clrrbs.** Then, an instruction that reads the register stored with incorrect parity is executed. If everything is operating correctly, this instruction should cause a rollback of one cycle followed by the repair of the register. To check whether a rollback has occurred, the **jmprbm** or **jmprbs** instruction is used. In this use of the "jump if rollback" instruction, the *Rs*1 field is set to the same register number as the *Rs*2 field, so that there will not be a master/slave mismatch if the rollback bit was no set. If a rollback has occurred this is an indication that the rollback mechanism is working. The next step of this test is to clear the rollback bits again, read the register originally stored with bad parity, and check the rollback bits. In this case a rollback should *not* have occurred if the state repair mechanism operated correctly the first time.

It is also possible to test the master/slave comparison of the signature of internal results (Section IV). This is done by clearing the rollback bits and then using the **jmprbm** or **jmprbs** instruction to store different values to the destination registers. Since different "ALU results" will be transmitted over busD, the signatures of the master and slave will differ, triggering a rollback of two cycles. After the rollback, the execution cycle of the **jmprbm** or **jmprbs** instruction will be repeated. However, now the rollback bits will be set, causing the branch to be taken. Thus, reaching the target of the "jump if rollback" instruction is an indication that the signature comparison operated correctly.

VII. VLSI Implementation of the Mirror Processor

We have completed a full-custom CMOS VLSI layout of the MP, using the MOSIS scalable CMOS design rules (SCMOS). All the features discussed in this paper are fully implemented. The chip contains 52 644 transistors, fits in an 84-pin package (76 pins are used), and, assuming 2-μm technology, the size of the chip is approximately 8.4 mm $\times$ 6.7 mm (Fig. 11).

We have used the switch-level simulator *bdsim* to simulate the entire chip executing several test programs. A phase-by-phase register-transfer-level (RTL) description of the chip was written in Zycad's ISP' hardware description language. The results of RTL simulation of the chip were compared, phase by phase, with the results of the switch-level simulations in order to verify the operation of the chip.

All the modules in the chip were simulated, at the circuit level, using a version of the SPICE circuit simulator (HSPICE), assuming 2-μm technology and using pessimistic (slow) values for the device parameters. The timing analyzer *crystal* was used to determine the critical paths for each phase. Based on SPICE simulations of the critical paths, the chip will operate correctly using a 100-ns clock, with 25-ns ϕ_1 and ϕ_3, 15-ns ϕ_2 and ϕ_4, and a 5-ns nonoverlap distance between phases. At this clock rate, power consumption is expected to be approximately 0.9 W per chip. The chip includes circuitry to generate the clock phases described above from an external 50% duty cycle, 40-MHz signal. This circuitry also requires an external 10-MHz signal, which is used to ensure that the master and slave execute the different phases in lockstep.

The area overhead for the error detection and recovery capabilities of the chip is significant. In particular, we estimate that by removing all the features for fault tolerance, the width of the chip could be reduced from 8.4 mm by approximately 1.8 mm and the height could be reduced from 6.7 mm by approximately 1.5 mm. Thus, approximately 39% of the chip area is devoted to supporting fault tolerance. Very little performance overhead is incurred for the fault tolerance capabilities. Specifically, the only additional delay is caused by a slightly larger capacitance to be charged during register-file reads due to longer buses across the register-file DWB and parity circuitry. This increases the clock cycle by less than 3 ns.

Regarding the area overhead, it should be noted that with modern microprocessors a large fraction of the chip is devoted to on-chip caches [12]. Our techniques would require using error-correcting codes in the caches and adding a DWB to the data cache. The relative area overhead for fault tolerance will thus be much lower for caches than for the rest of the chip, leading to a significantly lower relative area overhead for the entire chip.

VIII. Summary and Conclusions

We have presented the design and implementation of the Mirror Processor—a VLSI RISC microprocessor with unique capabilities for fault tolerance. A self-checking computer node, implemented with two MP chips operating in lockstep, can recover from most errors caused by transient faults. For those cases where a transient fault may permanently modify a stored value, we introduce the use of rapid hardware-supported state repair. When local recovery is impossible, the use of duplication and comparison results in a very high probability of detecting errors, so that system-level recovery can be initiated. As with other systems that use duplication and comparison, there is a low probability of undetected errors due to common-mode failures [20].

With traditional duplication, triplication, and voting [27], or even quad modular redundancy [4], a transient fault can corrupt the state of one of the copies, requiring lengthy system-level resynchronization [27] to restore the original redundancy to system. A duplex node constructed with Mirror Processors has the advantage that, in most cases, both processors maintain a valid state. Hence, the node can maintain its full detection and/or recovery capabilities without resorting to system wide recovery. An additional difficulty with schemes that involve voting [27] is that the voters cannot be "hidden" on the processor chip and thus require a higher chip count and more interchip wiring.

Our implementation of the MP demonstrates that micro rollback is a practical technique for minimizing the latencies normally associated with concurrent error detection. Micro rollback is used for re-execution of cycles (as opposed to instructions) during which errors are generated. Since local recovery is not sufficient for all possible errors, our design supports node self-reset, which guarantees that the node will reestablish a "sane" state from which it can participate in system-level recovery. The problem of latent faults in the detection and recovery mechanisms is addressed by special instructions that facilitate periodic self-diagnosis. The extensive fault-tolerance features of the Mirror Processor involve significant chip area overhead but only negligible performance overhead.

Acknowledgment

The Mirror Processor was implemented by M. Liang and T. Lai. M. Tremblay contributed preliminary layouts of some of the circuits supporting micro rollback and error detection.

References

[1] D. A. Anderson and G. Metze, "Design of totally self-checking check circuits for m-out-of-n codes," *IEEE Trans. Comput.*, vol. C-22, pp. 263-269, Mar. 1973.

[2] X. Castillo, S. R. McConnel, and D. P. Siewiorek, "Derivation and calibration of a transient error reliability model," *IEEE Trans. Comput.*, vol. C-31, pp. 658-671, July 1982.

[3] M. L. Ciacelli, "Fault handling on the IBM 4341 processor," in *Proc. 11th Fault-Tolerant Computing Symp.* (Portland, ME), June 1981, pp. 9-12.

[4] *80960MC Hardware Designer's Reference Manual*, Intel Corp., 1988.

[5] R. W. Downing, J. S. Nowak, and L. S. Tuomenoksa, "No. 1 ESS maintenance plan," *Bell Syst. Tech. J.*, vol. 43, no. 5, pp. 1961-2019, Sept. 1964.

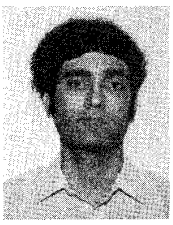
[6] W. W. Hwu and Y. N. Patt, "Checkpoint repair for out-of-order execution machines," in *Proc. 14th Annual Symp. Computer Architecture* (Pittsburgh, PA), June 1987, pp. 18-26.

[7] D. Johnson, "The Intel 432: A VLSI architecture for fault-tolerant computer systems," *Computer*, vol. 17, pp. 40-48, Aug. 1984.

[8] M. Johnson, "System considerations in the design of the Am29000," *IEEE Micro*, vol. 7, no. 4, pp. 28-41, Aug. 1987.

[9] M. G. H. Katevenis, "Reduced instruction set computer architectures for VLSI," CS Div., Univ. California, Berkeley, Rep. UCB/CSD 83/141, Oct. 1983.

[10] E. J. McCluskey, "Built-in self-test techniques," *IEEE Design Test*, vol. 2, no. 2, pp. 21-28, Apr. 1985.
[11] D. A. Patterson and C. H. Séquin, "A VLSI RISC," *Computer*, vol. 15, pp. 8-21, Sept. 1982.
[12] T. S. Perry, "Million-transistor microchip—Intel's secret is out," *Spectrum*, vol. 26, pp. 22-28, Apr. 1989.
[13] B. Randell, P. A. Lee, and P. C. Treleaven, "Reliability issues in computing system design," *Computing Surveys*, vol. 10, no. 2, pp. 123-165, June 1978.
[14] D. A. Rennels, "Architectures for fault-tolerant spacecraft computers," *Proc. IEEE*, vol. 66, pp. 1255-1268, Oct. 1978.
[15] J. J. Shedletsky, "A rollback interval for networks with an imperfect self-checking property," *IEEE Trans. Comput.*, vol. C-27, pp. 500-508, June 1978.
[16] R. W. Sherburne, M. G. H. Katevenis, D. A. Patterson, and C. H. Séquin, "A 32-bit NMOS microprocessor with a large register file," *IEEE J. Solid-State Circuits*, vol. SC-19, no. 5, pp. 682-689, Oct. 1984.
[17] M. Sievers and D. A. Rennels, "An LSI totally self-checking Hamming coded memory interface," in *Proc. Int. Symp. Circuits Syst.* (Rome, Italy), May 1982, pp. 1176-1179.
[18] J. E. Smith and A. R. Pleszkun, "Implementing precise interrupts in pipelined processors," *IEEE Trans. Comput.*, vol. 37, pp. 562-573, May 1988.
[19] Y. Tamir and C. H. Séquin, "Self-checking VLSI building blocks for fault-tolerant multicomputers," in *Proc. Int. Conf. Computer Design* (Port Chester, NY), Nov. 1983, pp. 561-564.
[20] Y. Tamir and C. H. Séquin, "Reducing common mode failures in duplicate modules," in *Proc. Int. Conf. Computer Design* (Port Chester, NY), Oct. 1984, pp. 302-307.
[21] Y. Tamir, M. Tremblay, and D. A. Rennels, "The implementation and application of micro rollback in fault-tolerant VLSI systems," in *Proc. 18th Fault-Tolerant Computing Symp.* (Tokyo), June 1988, pp. 234-239.
[22] Y. Tamir and M. Tremblay, "High-performance fault-tolerant VLSI systems using micro rollback," *IEEE Trans. Comput.*, vol. 39, pp. 548-554, Apr. 1990.
[23] Y. Tamir, M. Liang, T. Lai, and M. Tremblay, "The UCLA mirror processor: A building block for self-checking self-repairing computing nodes," in *Proc. 21st Fault-Tolerant Computing Symp.* (Montreal, Canada), June 1991, pp. 178-185.
[24] D. M. Taub, "Arbitration and control acquisition in the proposed IEEE 896 futurebus," *IEEE Micro*, vol. 4, no. 4, pp. 28-41, Aug. 1984.
[25] S. M. Thatte and J. A. Abraham, "Test generation for microprocessors," *IEEE Trans. Comput.*, vol. C-29, pp. 429-441, June 1980.
[26] M. Tremblay and Y. Tamir, "Support for fault tolerance in VLSI processors," in *Proc. Int. Symp. Circuits Syst.* (Portland, OR), May 1989, pp. 388-393.
[27] J. F. Wakerly, "Microcomputer reliability improvement using triple-modular redundancy," *Proc. IEEE*, vol. 64, pp. 889-895, June 1976.



Yuval Tamir (S'78-M'85) received the B.S.E.E. degree from the University of Iowa, Iowa City, in 1979, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1981 and 1985, respectively.

Since 1985 he has been on the faculty of the Computer Science Department at the University of California, Los Angeles, where he is currently an Assistant Professor. He established and is currently directing the Computer Science VLSI Systems Laboratory. His research interests include scalable parallel architectures, fault-tolerant computing, and VLSI systems.

Dr. Tamir is a member of the IEEE Computer Society and the Association for Computing Machinery.