Maintenance Techniques of a Microprogrammed Self-Checking Control Complex of an Electronic Switching System

HERBERT YU-PANG CHANG, GARY W. HEIMBIGNER, DANIEL J. SENESE, AND THOMAS L. SMITH

Abstract—This paper describes the various fault detection, routine exercise, and diagnostic techniques used in the design of a switching complex. Unlike other electronic switching system (ESS) processors, the processor is entirely self-checking. By carefully utilizing the capabilities of microprogramming and excess real time, most of the faults in the processor can be concurrently detected with little additional hardware. Integrating such self-checking features into the design of the processor is frequent. Error-detection techniques using the microprogram control include a full match of register-to-register transfers, a sequencing check to insure that instructions are executed in the proper order, and read-after-write checks of the memory. A great majority of the processor faults are detected concurrently by one of the above techniques. Those faults that cannot be detected concurrently (as identified by the computer simulation) as well as the maintenance logic are checked by frequent routine exercise. Routine exercise is performed by the processor at instruction level and microinstruction level and by the local maintenance center. Routine exercise also plays a part in determining the active unit when there is a fault in the call store (CS). Much consideration has also been given to simplifying the diagnostic procedure. The packaging of the processor and the test access provided facilitate the diagnostic resolution for many faults. The detailed gate-level design of the processor implementing all of these features has been completed. All of the circuits have been simulated on IBM 360/67 to verify the completeness of the self-checking. It has been demonstrated that it is possible to design a processor with a high degree of maintainability with only a moderate increase in processor size while keeping on-line maintenance programs to a minimum.

Index Terms—Fault detection and diagnosis, maintainability and reliability, microprogramming, self-checking processor, switching systems.

II. MAINTENANCE PLAN

II. Design Philosophy

There are a number of maintenance aspects of the system that are philosophically different from those employed in other electronic switching systems (ESS) [2], [3]. The control complex (processors and memories) is duplicated, but the duplicate copies are not matched, meaning that each central control (CC) must be capable of detecting its own malfunctions. This self-checking philosophy will minimize the cross coupling between the active and the standby units, and will also reduce the need to resolve conflict, as in the case of a duplicate-and-match approach whenever a mismatch occurs. Processor faults are automatically detected during normal call processing by the built-in check circuits. To minimize the cost of the processor, excess real time is used for checking whenever possible. Design guidelines for high maintainability are carefully followed [1], [4], including minimum use of sequential logic. The added check circuits are centralized to minimize cost and are designed to be easily exercisable on a routine basis. A regular structure (e.g., uniform width-data paths throughout the processor) is maintained to optimize circuit packaging as well as to facilitate fault diagnosis.

A second major difference from previous ESS machines is in the nature of the LMC. The LMC is a programmable unit separate from the two CC's that performs many maintenance and administrative functions. The complexity of the LMC is reduced by allowing it to work in conjunction with the standby CC. This also allows many maintenance and administration programs to be stored off-line and loaded whenever needed. The removal of these programs simplifies the CC executive and thus, results in a fairly straightforward program structure.

The fact that the LMC interacts only with the standby control reduces the chance of manual errors creeping into an online CC. Ample access and control are provided from the LMC.
to simplify the diagnosis of the (standby) processor. The LMC also determines the active and standby status of the control complex and the peripheral frames.\footnote{The peripheral frame circuitry that communicates serially with the central control is switched as a group along with the control complex.} Upon detection of a switch request from a faulty processor, the LMC decides on the proper restart procedure based upon the severity of the switch request and previous fault indications.

III. FAULT DETECTION

Recent advances in integrated circuit (IC) technology development have resulted in an increasing emphasis on fault detection. In real-time applications, faults must be detected immediately in order to facilitate smooth recovery before normal operations are affected. This requirement precludes the use of program testing as the primary fault-detection mechanism. Self-checked circuit design thus becomes a major topic in the planning of the processor.

Past experience indicates that if self-checked circuits are designed without considering their routine exercise sequences and additional maintenance access, overall maintainability will usually suffer. Furthermore, in many cases, the degree to which a circuit is self-checked cannot be readily determined by functional level (block diagram level) analysis because of the specific design constraints associated with any given logic family or alternative implementations. Consequently, extensive computer simulation of the processor logic has been performed in order to obtain a quantitative measure of its self-checking capability. Simulation is done by using a digital fault simulator running on IBM 360/67; it proves to be an invaluable mechanism for providing design verification and fault analysis [5].

In any switching processor, the various functions performed can be categorized into three areas: data transfer, data manipulation, and control. Concurrent fault-detection techniques for circuits in each of those categories of the processor have been devised and are described next.

A. Fault Detection in Data Transfer Circuits

Data transfer refers to moving unaltered data from a source location to a destination location where the source and destination can be registers or memory locations. Fig. 1 shows the block diagram of the register-bus system where each register has 2 check bits (parity on address and parity on data) and 16 information bits. The processor register-bus system is very regular with all data transfer paths (i.e., registers, buses, and memories) of uniform width.

The primary tools for detecting faults associated with the register-to-register gating functions are the check register and the parity-check circuit [1], [6]. In gating a source register (e.g., R 0) to a destination register (e.g., R 13) via the normal bus, the contents of the source register (R 0) are also toggled into the cleared check register. If all the leads work correctly, both the check register, a toggle register whose initial state is all zeros, and the destination register (R 13) should contain the same data. In the next step, when the destination register (R 13) is gated into the check register, the check register should return to an all-zeros state. A nonzero indication upon interrogation of the check register will cause a maintenance alarm. Since this procedure performs a full match of register-to-register gating, multiple errors are detected. In addition, the bus-parity circuit is interrogated each time data is gated onto the bus in order to detect transient errors (odd number of errors only) that might have occurred during the interval between gate-in and gate-out from a certain register.

Address parity is used as a means of detecting memory access circuitry faults. The translation store (TS) and call store (CS) are accessed via the address stored in the memory address register (MAR). During each memory read, the address parity bit retrieved from memory is compared with the parity of the address of the memory location being read. For each memory write (for CS only), the data to be written (16 data bits) are toggled into the cleared check register and the address parity is generated and toggled into the address parity bit. Next, the data in the check register is written to the CS. After the write cycle, an automatic microprogrammed read-after-write check
is performed with the data just read gated into the check register. The two patterns are thus compared when the check register is interrogated for an all-zeros indication. During all data manipulation and transfer operations other than writing the CS, the address parity bit is carried along unaltered.

B. Fault Detection in Data-Manipulation Circuits

Data manipulation is a general term used here to denote all the logic and arithmetic operations, such as AND, OR, shift, and count. As opposed to data transfers, data are usually "altered" when being manipulated. There are basically three data manipulation circuits in the processor: count (increment by one), rotate (steer logic), and insertion mask (IM). Parity prediction and checking is the primary mechanism for detecting faults associated with the count, rotate, and IM circuitry. To facilitate self-checking and diagnostic testing, the realization of these circuits is totally combinational; the manipulation logic contains no sequential circuits. The inputs are taken directly from the input bus (see Fig. 2) and the outputs of the manipulation logic either directly or indirectly (i.e., via a register for storing intermediate results) feed the output bus.

A block diagram of the count circuit is shown in Fig. 3(a). Data from the input bus are incremented by one, the new parity is independently computed (by parity prediction), and the result is then checked by the bus-parity circuit. Similarly, Fig. 3(b) shows the block diagram of a rotate circuit. Data from the input bus can be rotated from 0 to 15 positions and the results stored in the accumulator. Since rotation preserves parity, the operation is checked by the bus-parity check circuit when the results are gated onto the bus. The IM operation is also checked as shown in Fig. 3(c), by parity prediction and checking. The parity-predict function is generated using independent circuitry to compute the mod-2 sum of the individual bit positions that will cause parity to change and, again, the result is checked on the bus. All of the circuits have been designed and simulated to evaluate their self-checking capability. The results are summarized in the first three lines of Table I. The percentage of immediately detectable faults is interpreted as follows. For a given input, all those faults that affect
the data but are not detected by the hardware error-detection circuits are called nondetectable faults. Every other fault is considered detectable. Detectable faults for a given input include those faults that do not affect the output and those faults that are detected by the error-detection circuits. The fault analysis is exhaustive; for each input pattern, all faults have been simulated and simulation has been repeated for every possible input.

The range given in Table I is the percentage of detectable faults for those two inputs that have the largest and smallest number of detectable faults. The average percentage of detectable faults for all possible inputs, thereby assuming that all inputs are equally likely, is also given. All these circuits exhibit a high degree of self-checking capability.

### C. Fault Detection in Control Logic

Techniques for detecting faults in the control logic are generally different from those used for data transfer and data-manipulation circuits. This is partly due to the irregularity of the structure and partly due to the sequential nature of the circuitry. Both characteristics pose a problem in self-checking design. In this processor, irregularity is minimized by employing microprogramming and sequential logic is reduced by the expanded use of memory and decoders for controlling instruction fetching and other operations.

In this section, fault-detection techniques for the microprogrammed control section are discussed. This is followed by a discussion of the overall timer and the self-checked clock system design.

1) Microprogram Control: An important consideration in using microprogramming techniques for the implementation of the processor control section is ensuring that the sequencing (or linkage) between instructions and microinstructions is properly maintained. As discussed in [11], the sharing of hardware between the instruction register (IR) and the microprogram data register (MPDR) and the use of microcode provides a powerful means to check the proper linkage between microsteps and macrosteps.

Furthermore, there is also a very strong check of the sequencing from microinstruction to microinstruction inherent in the check register sequences. For improper sequencing to occur and not be detected, there would have to be a correct mesh of the check register events. If a “check register = 0” (ck = 0?) test is performed after a “gate-into-check” (g.c.) or any odd number of g.c. operations, an error will be noted. The sequential series of check register operations is backed by the address-parity technique. There are two parity bits associated with the next-address field of the MPDR (Fig. 4). One is associated with the address base (AB), the other with the operation code (OP). When accessing the microprogram memory, the parities of these two subfields are compared and stored in two flip-flops. Upon retrieving the next microinstruction from memory, the outputs of these two flip-flops are compared with the parity bits PH and PL which have been computed by the microprogram assembler and stored in the fetched word. A mismatch indicates that a memory accessing error has occurred.

A parity bit is also associated with each of the remaining fields of the MPDR: TO, FROM, bus select, check register, and memory preread. The bus select errors, if not detected by parity (e.g., 01 changes to 10) will generally result in the failure of a check register sequence or an overall time out when the wrong bus is selected. Errors associated with the check register field (g.c. or ck = 0?) not caught by the parity bit are likely to cause the checking steps of the microcode to be out of step, and thus will be detected. Memory preread errors, either the wrong memory (i.e., among TS, CS, and PS) being accessed or no memory being accessed will generally cause an address-parity error or a data-parity error on the bus. In short, it can be said that the MPDR and its associated circuitry are protected by many levels of checking mechanisms: group-parity checks, sequencing, data-parity checks, and checkers designed for other circuits.

The outputs of the TO and FROM (1-out-of-N) decoders are checked by the 1-out-of-N check circuit shown in Fig. 5. The outputs of the decoder are divided into two groups based on the parity of the decoder input. Since exactly one output should be active, the check circuit identifies the no-active and multiple-active output cases as errors. Dual-rail implementation of the decision logic allows the 1/N check circuit to also be used for detection of decision logic failures.

2) Overall Timer and Self-Checked Clock System: Almost all of the program tasks performed in a switching system are of a periodic nature. Tasks are usually arranged in a cyclic loop and thus, can be designed with appropriate check points. The time between check points is predictable when the program structure is defined. To guard against having the processor indefinitely trapped in a program loop, either due to a hardware fault or a software bug, an overall timer is provided. The timer is reset when the program passes through a check point. The time between check points should be short enough that the timer will not normally time out. If the timer automatically times out before a check point is reached, a maintenance alarm will be issued. The timer is not intended to be the primary fault-detection mechanism, but rather serves as backup to other checking logic discussed in previous sections.

To insure completely reliable operation, the self-checking principle is also extended to the processor clock system. A four-phase clock is used to control the various register gating, data manipulation, and sequencing functions. The clock sys-
Fig. 4. MPDR and its associated checking circuitry. (Note: *The other two error indicators associated with microprogram section are group-parity indicators.)

Fig. 5. 1-out-of-N (1/N) check circuit.

Fig. 6. Clock-phase generation and checking.

detector. In this way, all errors in pulsewidths and relative spacing of clock phases can be detected.

D. Fault-Detection Summary

Extensive self-checking capabilities have been incorporated in the processor design to provide immediate detection of errors. The various techniques employed can be summarized as follows.

Data Transfer: Bus parity, check register, and microcode.
Memories: Address parity, data parity, read-after-write, and group parities on microprogram memories.

Data Manipulation: Parity predict, bus parity.
Microprogram Control: Instruction/microinstruction sequence check, microprogram data register group parities, 1-out-of-N check of decoders, microprogram address parities, and overall timer.
Clock System: Duty cycle check and critical edge check.

With the combined use of these techniques, all operations are concurrently checked not only at instruction (or macro) level, but also at microinstruction level. The following example illustrates the various checking steps taking place in performing a register-to-register transfer instruction, REGA → REGB:

Step 1: IR → NOP, g.c.
Step 2: IAR + 1 → IAR, ck = 0?
Step 3: REGA → REGB, g.c.
Step 4: REGB → NOP, g.c., start PS read.
Step 5: RPS → IR, ck = 0?, g.c.

The first step and the last step are the proper instruction/microinstruction linkage check associated, respectively, with the preceding and the succeeding instructions (see [1]); the check register is interrogated for an all-zeros (ck = 0?) indication in Step 2. Once this is successful, the instruction address register (IAR) is incremented using the count bus (Step 2),
parity is predicted, and the results are checked by the bus-parity circuit. In Steps 3 and 4, register-to-register gating takes place with the proper transfer of data checked by the check register, which again is interrogated for an all-zeros indication in Step 5. The gating of the next instruction, which has been accessed in Step 4, from the PS into the instruction register (IR) takes place in Step 5 and the address parity is examined. Other checking steps that are not explicit in the microcode are the data-parity checks whenever information is gated on the bus, the group-parity checks of various fields of the microprogram data register, the 1-out-of-N checks of the TO and FROM decoders, and the clock-phase checks. These checks are performed every time a microinstruction is executed.

The designs of various self-checked circuits mentioned in Sections III-A-C are complete and have been verified by computer simulation. Fault analysis results indicate that all circuits exhibit a very high degree of self-checking. The results are summarized in Table I. Details of the simulation work are discussed in [5].

IV. Routine Exercise

A. General Considerations

The processor self-checking philosophy just described provides adequate protection against all but a few faults that cannot be concurrently checked using error-detection hardware. Consider, for example, the circuit that checks data parity of the bus. This parity circuit can be tested by operating on normal data, with one major exception. Since all of the input vectors have the correct parity, the circuit output should constantly indicate a "pass" condition, and some faults in the critical last stages of logic are not detected. To maintain a high degree of confidence in the concurrent check circuit and its ability to identify other faults, these last logic stages must be exercised on a periodic basis to show that they can detect errors.

The major problem posed by routine exercise of any constant-output check circuits is that the circuit must be driven into a "fail" state. There are two aspects of the problem. First, the facility to introduce deliberate errors or to simulate errors must be provided in the initial design phases and this additional access must be maintainable. A second aspect concerns system reaction to these deliberate errors. The reaction should exercise the normal response circuitry to the fullest extent possible, but undesirable results must be inhibited. For instance, it would be undesirable to have every deliberate error introduced during routine exercise cause a switch of central controls. The number of tests required would cause a considerable amount of unnecessary configuration switching and would be extremely inefficient.

From the preceding discussions, it is quite apparent that a number of problems exist that can be solved by routine exercise. The process of designing routine exercise tests is one of defining the required access and supplying the necessary maintenance microlevel code (i.e., the TO field and the FROM field data). This ability to carefully specify every activity at a true gate level of control and to use the microdecision capability allows a very efficient routine exercise implementation. The implementation takes specific advantage of the self-checked decision facilities of the processor to perform a large part of the exercise in parallel. The impact of the microprogram level implementation of the exercise procedures will be made more obvious in the routine exercise summary (Section IV-D3).

In the processor, an attempt has been made to handle the constant-output check circuits in a manner most compatible with microprogram control. Each constant-output lead is tied to a bit of a microprogram addressable register called the status register, as shown in Fig. 7. Also evident in this figure is the double-rail circuit that monitors the status register for the all-zeros condition. The use of a double-rail realization provides two independent maintenance communication links that are fault tolerant under single-failure conditions, between the processor and the LMC. Any perturbation from an all-zero state in the status register causes the immediate generation of a pair of independent error signals that will cause the LMC to switch processors (unless the processor has inhibited both error signals during the routine exercise microprogram).

The major advantage of this organization is that it allows microprogram controlled use of the processor's self-checked decision logic for analysis of the routine exercise results. Since the status register data is bitwise independent, it is feasible to exercise the check circuits in parallel and analyze the results in parallel. This increases the efficiency of the exercise process and reduces the penalty usually associated with a large number of check circuits.

In the next section, the constant-output check circuits in the processor and the special access required to exercise them are discussed. Section IV-C discusses the organization of the routine exercise subroutine and microroutine. Section IV-D describes the use of a routine exercise for the CS which is required for system reconfiguration and recovery rather than
error detection. Finally, the role of the LMC in routinely exercising the system is described.

B. Exercise of CC Check Circuits

The following discussion pinpoints examples of constant-output portions of circuitry requiring routine exercise. For each of the identified circuits, the method of exercise is described and the required microprogram access is discussed.

1) Address- and Data-Parity Circuits: The normal flow of data in the processor exercises the parity circuits on a semi-regular basis. The remaining problem is to introduce an improper parity word and observe the circuit response. The method of doing this involves reading one of the memories without first having preread it. The result of this operation is the idle state of the selected memory, which is of improper parity. By selecting a memory address requiring an address-parity bit opposite the idle memory state, both the data- and the address-parity circuits can be exercised by this single test.

2) MPDR Parity Circuits: To exercise the MPDR checks, several microprogram words must be supplied with incorrect parity bits. The microassembler has the option of setting these bits as desired. There are a total of three independent indicators associated with the group parity and access checks of the MPDR: one associated with the TO and FROM fields, one with the control field (i.e., the check register, the bus select, and the memory preread), and one with the next-address access. Since this section is not bit sliced, the increased number of checks should lead to an easing of the problems of diagnostic resolution. Also, since the checks can be performed in parallel, the cost in terms of hardware and real time is minimal.

3) 1/N Checks: The 1/N check is another constant-output check circuit. The routine exercise is to artificially introduce the “none-active” and the “two-active” input states and to observe the circuit responses. The none-active state is simulated by providing a decoder state that is not wired to either side of the 1/N check circuit. Selection of this state should yield a failure indication. Similarly, a second decoder output is wired to both sides of the check circuit. This simulates the two-active situations. The execution of both of these maintenance instructions will verify the ability of the 1/N checks to detect failures.

4) Check Register: During normal usage, the check register may present problems at two points. First, the double-rail circuit that monitors for the all-zeros condition has two "pass/fail"-type constant outputs. Verification of this circuit is carried out by putting a single one in each position of the check register and executing a zero test (ck = 0?). To ease the problems of verification, a separate status register bit is provided for each output rail of the monitor circuit. This overall approach makes the monitor circuit nearly immune to single failures.

A second aspect of the check register that could present problems is the state of the individual flip-flops which normally function in a return-to-zero mode. Thus, if a flip-flop becomes "stuck-at-zero," it will not be concurrently detected. There are two major exceptions to this mode of operation. The first is the execution of the short transfer instruction, which involves the formation of the target address in the check register [1]. The second exception is the process of writing data into the CS, which is done from the check register. The data from the check register in both of these cases is examined by parity. Both instructions are frequently used (e.g., more than 30 percent of the time) in call processing programs. Thus, while the routine exercise for the monitor circuit also verifies the operation of the individual flip-flops, the probability of detecting a check register flip-flop stuck-at-zero on a concurrent basis (i.e., by the data-parity check) is extremely high. The fact that the check register is bit sliced strengthens this assumption.

5) Clock Circuit: The self-checked clock for the processor has a single constant output and five routine exercise inputs. Two inputs exercise the gates that monitor the clock phases for improper overlap. Selection of one of these leads should result in the clock-error bit of the status register being set due to clock-phase overlap. The other three maintenance leads are used to exercise the circuit that checks the duty cycle of the clock phases. A major part of this clock circuit is a filter arrangement, which means that the real-time response of this circuit is not immediate. To initiate a duty cycle failure, a ground must be applied to one of the three maintenance inputs for a duration of three machine cycles. When the clock circuit responds and the maintenance signal is removed, a delay must be initiated to allow the filter section of the check circuit to recover. This is done by taking advantage of the decision and counting capability of the processor to implement a 30-machine-cycle delay loop.

6) Decoder Clock Leads: A good example of the usefulness of the status register approach is the problem of the clock-lead input diode open on the TO decoder gates. This problem was originally viewed as a routine exercise problem and, as such, would have required an extensive amount of microcode (at least one word per decoder output). By monitoring the state of two nodes internal to the 1/N check (the ODD side and the EVEN side of the 1/N check, see Fig. 5) on the TO decoder when the clock phase is off, this problem is reduced to one of concurrent detection. The required hardware is two gates, I b of the status register and two unclocked decoder outputs for exercise. The added access and trouble of a special check circuit are thus seen to be rather minimal under the adopted exercise procedure.

The preceding examples are not intended to be complete. They are intended to provide insight into the types of undetectable faults encountered in the initial design phases. They are also examples of how careful initial designs limit the need for special access.

C. Organization of Exercise Routine

1) Instruction Level Routine: In the previous sections, the methods of exercising the processor have been discussed. The actual exercise procedure uses a single subroutine that performs the macrolevel exercise and provides the required data input to the routine exercise microprogram. This subroutine also analyzes some of the results. A flow chart is shown in Fig. 8. The major goal of the instruction code is to handle as much of the bookkeeping as possible. This routine checks defensive
programming flags and exercises those circuits that are fully exercisable without dedicated or unusual access.

2) Microlevel Routine: The microlevel exercise is initiated by the execution of a single special instruction or microprogram. The flow chart of this instruction is shown in Fig. 9. The actual exercise procedure is performed in five distinct phases, where each phase consists of sequentially driving a number of check circuits into a failed state. The response of each check circuit is trapped in the status register yielding a binary pattern. The previously exercised decision logic is then used to compare the status register pattern with a data pattern loaded by the initializing subroutine. If the generated pattern matches the desired pattern, a progress pointer is updated and the next exercise phase proceeds. The actual circuits exercised by each phase are shown in Table II.

Several points about the exercise instruction should be made. First, the instruction is quite specialized and depends upon data loaded by the calling subroutine. Accidental execution of the instruction will appear as a routine exercise failure (a hardware response) with a progress pointer stored in a specific register. Second, as shown in Table II, a number of tests executed appear redundant. These allow reduction of the number of data patterns that must be prestored. Finally, the requirement of five sequential test phases is dictated by the need to apply tests to the clock circuit and the MPDR control-field parity-check circuits.

If, at any time during the exercise procedure a test does not pass, an abnormal termination is invoked. This consists of placing a unique code (all ones) in the status register, placing the progress pointer in a specific general register, and signaling the LMC of the trouble. This is also the signal used if one of the defensive checks in the subroutine fails. By using an easily recognized primary pattern in the status register and the progress pointer, the diagnosis required to resolve failures in the check circuits can be made quite small.

3) Summary of Exercise Routine Requirements: The present version of the routine exercise instruction (microprogram) is composed of approximately 140 microprogram words. During normal operation, this routine requires approximately 200 microcycles for its execution with 90 cycles being consumed in delay loops waiting for recovery of the clock duty cycle checks. The supporting program is approximately 100 instructions and would require about 800 cycles to complete. The total exercise requirements are about 1000 machine cycles which, for the 300-ns clock cycle, is about 300 μs. The short length of time required for routine exercise is rather important. Since the procedure is short, it can be run quite often. Assuming relative independence of the faults, this means that the probability of failure in a check circuit output masking a subsequent failure can be kept quite low, which should lead to very high confidence in the integrity of the check circuits.

D. CS Recovery Procedures

The CS, unlike other stores in the system, can be both read and written by the CC. The active CC must update the infor-
TABLE II
INDICATORS TESTED BY ROUTINE EXERCISE PHASES

<table>
<thead>
<tr>
<th>Phase</th>
<th>Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1) Data parity (idle TS states)</td>
</tr>
<tr>
<td></td>
<td>2) Address parity (TS address)</td>
</tr>
<tr>
<td></td>
<td>3) Clock-lead open (even half)</td>
</tr>
<tr>
<td></td>
<td>4) MPDR1 (from field—test 1)</td>
</tr>
<tr>
<td></td>
<td>5) MPDR2 (preread parity)</td>
</tr>
<tr>
<td></td>
<td>6) MPDR3 (access check—low—test 1)</td>
</tr>
<tr>
<td>2</td>
<td>1) Data parity (idle CS state)</td>
</tr>
<tr>
<td></td>
<td>2) Address parity (CS address)</td>
</tr>
<tr>
<td></td>
<td>3) Clock-lead open (odd half)</td>
</tr>
<tr>
<td></td>
<td>4) Clock duty cycle (clock phase B)</td>
</tr>
<tr>
<td></td>
<td>5) MPDR1 (from field—test 2)</td>
</tr>
<tr>
<td></td>
<td>6) MPDR2 (check register parity)</td>
</tr>
<tr>
<td></td>
<td>7) MPDR3 (access check—low—test 2)</td>
</tr>
<tr>
<td>3</td>
<td>1) Data parity (idle PS state)</td>
</tr>
<tr>
<td></td>
<td>2) Address parity (PS address)</td>
</tr>
<tr>
<td></td>
<td>3) Clock lead open (odd half)</td>
</tr>
<tr>
<td></td>
<td>4) Clock duty cycle (clock phase C)</td>
</tr>
<tr>
<td></td>
<td>5) MPDR1 (to field—test 1)</td>
</tr>
<tr>
<td></td>
<td>6) MPDR2 (bus control parity)</td>
</tr>
<tr>
<td></td>
<td>7) MPDR3 (access check—high—test 1)</td>
</tr>
<tr>
<td>4</td>
<td>1) Data parity (single &quot;1&quot; in word)</td>
</tr>
<tr>
<td></td>
<td>2) ck = 07 (&quot;1&quot; rail)</td>
</tr>
<tr>
<td></td>
<td>3) ck = 07 (&quot;0&quot; rail)</td>
</tr>
<tr>
<td></td>
<td>4) 1/N—FROM field (two active)</td>
</tr>
<tr>
<td></td>
<td>5) 1/N—TO field (two active)</td>
</tr>
<tr>
<td></td>
<td>6) MPDR1—(to field—test 2)</td>
</tr>
<tr>
<td></td>
<td>7) MPDR2 (check register—test 2)</td>
</tr>
<tr>
<td></td>
<td>8) MPDR3 (access check—high—test 2)</td>
</tr>
<tr>
<td></td>
<td>9) Clock (overlap of phase B and phase E)</td>
</tr>
<tr>
<td>5</td>
<td>1) Data parity (single &quot;0&quot; in word)</td>
</tr>
<tr>
<td></td>
<td>2) ck = 07 (&quot;1&quot; rail)</td>
</tr>
<tr>
<td></td>
<td>3) ck = 07 (&quot;0&quot; rail)</td>
</tr>
<tr>
<td></td>
<td>4) 1/N—FROM field (zero active)</td>
</tr>
<tr>
<td></td>
<td>5) 1/N—TO field (zero active)</td>
</tr>
<tr>
<td></td>
<td>6) MPDR1 (arbitrary)</td>
</tr>
<tr>
<td></td>
<td>7) MPDR2 (bus selection—test 2)</td>
</tr>
<tr>
<td></td>
<td>8) MPDR3 (access check—arbitrary)</td>
</tr>
<tr>
<td></td>
<td>9) Clock (overlap of phase B and phase D)</td>
</tr>
</tbody>
</table>

* Included to allow minimization of the number of prestored patterns.

Fig. 10. CS–CC organization.

shows that CC 0 cannot work with its own store, CC complex 0 is faulty. If this test does not detect any faults, CC 0 makes a routine test of CS 1. If this test detects a fault, some further check must be made to determine whether the fault is in call store 1 or the circuitry connecting CC 0 and CS 1. \( \text{CC} 1 \) can most easily make this test. If CC 1 can operate correctly with its own store, the fault is probably in the circuitry connecting CC 0 to CS 1. If CC 1 cannot operate correctly with its own store, the fault is probably in CS 1.

Whenever a CC does not operate correctly with its mate store, a coordinated effort of the two CC's is required to determine which CC complex is faulty. Four words in the CS are used to coordinate this effort. The first word is set when CC 0 finds CS 1 faulty and reset otherwise. The second word is set when CC 1 finds CS 1 nonfaulty and reset otherwise. The other two words provide similar functions for CS 0.

For example, when CC 0 first finds a fault in CS 1, it assumes the fault is in CC complex 0, sets the appropriate memory word, and initiates a switch of CC's. CC 1 will now determine if CS 1 is faulty and mark the appropriate memory word. If the store is faulty, there will be another switch of CC's. CC 0 will again find CS 1 faulty, but before initiating a switch, it will check the control words. The first control word indicates that this fault has been previously found. The second word indicates whether the CC 1 can work with the apparently faulty CS. CC 0 will initiate a switch depending on whether or not CC 1 can work with its own CS.

The routine test of the CS's and subsequent recovery procedures described above are used when an error is detected in the CS and are also used whenever a CC becomes active. The recovery procedure would normally be used after a fault was detected. However, because part of this routine involves a switch of CC's and subsequent testing in the other CC, the recovery procedure must be performed by a CC after it is made active but before it processes any calls.

The routine exercise program must be included in each CC for error detection in the mate CS and for the CS recovery procedure. The self-checking techniques are adequate for fault detection in the home CS. However, since the routine exercise program is already on-line, and since there is adequate real time available, faults will be detected more quickly if a thorough exercise of both CS's is performed routinely.
E. LMC

The LMC is connected to both CC's and interacts directly with the standby CC to perform maintenance and administrative functions.

The primary function of the LMC is to diagnose the standby CC as described in Section V. Some additional functions are to determine the active CC, to prevent the standby CC from interfering with the active CC, and to perform routine tests.

1) Non-detectable CC Faults: The CC has a very high degree of self-checking as previously described, but it is not possible to make a machine 100 percent self-checking. A few faults will always be undetectable. The processor relies on the LMC to routinely check some of these faults.

One of the most critical pieces of circuitry associated with error detection is the one that requests a switch of CC's. Because of its importance and because it is impossible to detect errors in this circuitry from the CC, this circuitry is duplicated. This mechanism is exercised by routinely requesting a switch of CC's to insure that it will work correctly in case of a CC failure. Because of the redundancy in this mechanism, a single fault will not cause it to fail, but these single faults must be detected. If they go undetected for long periods of time, a subsequent fault could occur in the same circuit. The resulting multiple faults could prevent a necessary switch of CC's.

The CC cannot test for the types of faults described above because they are too intimately associated with the operation of the CC and because the test access is inadequate. The LMC must test for these faults on a routine basis, using the same circuitry required for diagnostic testing.

2) LMC-CC Interface: The interface between the CC and the LMC is another area for potential problems with error detection. This interface is necessary to allow the LMC access to test and control the CC. It is not possible to completely test this interface from the CC, but some faults within the interface will be detected by the CC. Error detection can be accomplished by dividing faults in this access circuitry in two types, those which cause data mutilation while the CC is operating, and those which cause incorrect information to be transferred between the CC and LMC. Those faults that may cause data mutilation will be detected by the self-checking mechanisms within the CC. The other faults in the access circuitry will not have any effect until the access circuitry is used. These faults will be detected through routine exercise by the LMC before any access is made to the CC.

V. Diagnosis Considerations

A. General Objectives

The primary objective of the processor diagnosis procedure is to be able to “efficiently” resolve the location of a fault to a small set of replaceable circuit packs (e.g., one to three packs). The facets of efficiency must include minimization of manual intervention, minimization of diagnostic production effort, and maximization of correct resolution. By considering the problems of diagnosis currently with those of logic design and packaging, a number of problems have been avoided. The result should be a diagnostic procedure requiring a minimum of LMC program storage and a minimum of fault simulation to prepare trouble locating manual (TLM) information [7]. In this section, several important aspects of diagnostic techniques will be discussed.

B. LMC Interface

The LMC is responsible for diagnosing the faulty processor which is in the standby mode. This aspect of maintenance is not assigned to the processor because of the difficulty of a faulty processor diagnosing itself. The LMC-processor interface must be considered sufficiently early in the design of the machine to ensure that complete access to the processor is provided.

The LMC, when called upon for diagnosis, is faced with the problem of examining and manipulating a stopped processor. To accomplish this, the LMC must have ability to exercise complete control of the processor clock circuit. The LMC can, under program control, step the processor through any number of microinstructions or instructions.

The LMC also has several ports that are used to supply data to and retrieve data from the processor (see Fig. 11). The major write ports are to the translation store (18 b), the program store (18 b), the MPDR (36 b), and the input side of the processor bus (18 b). To complement this write access, the read ports are the status register (18 b), the MPDR (36 b), and the output of the processor bus (18 b). This test interface was chosen to provide complete access to the processor. Using this interface and control of the timing, the processor can be set into any desired state. The bus interface provides access to store information into the registers, and the MPDR access allows the state of the microprogram section to be set. This interface also provides sufficient test points to determine the state of the machine. The bus interface is used to determine the state of the register, the MPDR access shows the state of the microprogram section, and the status register provides information about the state of the check circuitry.

When a hardware error is signaled to the LMC, a 72-b vector is immediately accessible. In most cases, this vector cannot completely pinpoint the fault, but in many cases, it can localize the fault to a small subset of the total possible faults in the machine. Some imprecision arises in limiting the faulty area of the machine because the clock is not stopped immediately when the error occurs. Generally, a full microcycle elapses before the clock is turned off. In addition, some of the available information is data dependent and cannot be directly related to a fault. However, the information in the status register is always useful, and the microprogram data register should give an indication of what function was underway when the fault occurred.

C. Partitioning the Processor

One of the major advantages of increased levels of circuit integration is the relaxation of the diagnostic resolution problems. To take full advantage of this characteristic requires careful attention to packaging during the design phases. The major packaging effort has been directed toward strengthening the single-fault assumption by “bit slicing.” This implies that as much circuitry as possible associated with 1 b is packaged on a single circuit pack, thus minimizing the dependency between bits. With this approach, the only major connections between
data pattern is small, the subsequent diagnostic procedure to isolate the fault to a single pack should also be small. Any improvements in the fault resolution obtained by observing these patterns will be reflected in an improvement in the diagnostic program.

An example (a failure in the TO decoder) of the procedure described above should clarify many details. The TO decoder in the microcontrol section is functionally packaged as a TO decoder (even parity) board, a TO decoder (odd parity) board, and the TO field 1/N check circuit board. Failures in this circuitry are generally indicated by the TO decoder 1/N check-failure bit or clock-lead open bit in the status register. The first notice of trouble that the LMC receives is than an error has occurred. This message is initiated when the status register is set to a nonzero state. The LMC reads the status register and finds a pattern consisting of the TO decoder clock-lead open bit set and the TO field 1/N check-bit set. Recognizing this pattern, the LMC pages in the proper diagnostic program from off-line storage. In this case, the suspect subsystem is the TO decoder and the TO field 1/N check circuit. The LMC program then clears the status register and observes its next state. If the status register cannot be cleared, the problem is immediately isolated to the check circuit or the status register. This one test may then resolve the problem to a board. If the status register remains zero, the LMC access to the MPDR is verified by a write-read-match sequence. This test checks the TO field in the MPDR. Next, the TO field is loaded with data that should yield an even-parity lead active. The clock is operated through a single cycle and the status register is observed. If the clock-lead open bit is set but not the 1/N check bit, the failure is in the even TO decoder gates. If both bits are set, the failure is in the odd TO decoder gates. Here, three tests have resolved the fault to a circuit pack.

Not all of the faults can be diagnosed as easily as the TO decoder chosen for this example, but the same techniques can be used for many of the faults. In general, the size of the diagnostic program required for a particular fault pattern will be small.

E. Some Diagnostic Remarks

From a diagnostic standpoint, the worst time for the machine to fail is during the routine exercise of the checking circuits. A failure at this time is likely to create unusual patterns because the machine is in an unusual state. In particular, part of the error detection hardware may be inhibited. As a defense against these problems, the processor routine exercise program saves a progress pointer in one of the registers. By interrogating that register, the local maintenance center can determine whether a routine exercise procedure was in progress and which phase of the program was in progress when the fault occurred.

VI. CONCLUSION

A primary objective of the project was to demonstrate the feasibility of a low-cost self-checking processor. This has been achieved by the careful integration of maintenance techniques into the overall processor architecture. Maintenance considerations resulted in a very regular structure with minimum use of
sequential logic, in order to enhance concurrent fault detection capability and to simplify processor diagnosability. Cost considerations dictated the desirability of centralizing the checking operations, and of trading real time for hardware whenever possible. Microprogramming plays an important role in all of these considerations.

Faults in data transfer circuits are detected by using the check register gating steps along with some microcode; this self-checking capability is equivalent to that of a full-match process. Parity prediction and checking techniques are used to detect faults in the data-manipulation circuits (e.g., count, rotate, and insertion mask); simulation results indicate that a high degree (more than 90 percent) of concurrent fault detectability has been obtained. The use of the check register and parity checking has been extended to the microprogram control section for checking the instruction-to-instruction transition and microstep sequencing. The 1-out-of-N detection scheme is employed for checking the decoders and the decision functions. Careful design techniques have maximized the amount of circuitry checked by each dedicated check circuit.

The small number of faults not detectable via concurrent hardware checks has been isolated via computer simulation. These faults and the maintenance logic are checked by routine exercise sequences. The ability to implement these sequences in microcode allows an on-line processor to detect any fault in its internal operation without resorting to special, hard-to-check circuitry. Frequent execution of these routine exercises will insure the integrity of the checking circuits, which in turn reinforce the overall processor self-checking ability. The small size of the processor routine exercise program (~ 140 microinstructions and 100 instructions) is a reflection of a comprehensive design plan where hardware, software, and maintainability tradeoffs were continuously evaluated.

Diagnostic considerations for fault isolation and repair have also been incorporated into the processor design, but the implementation of diagnostic programs has not been completed. Bit slicing is used in circuit packaging to minimize the effect of unforeseen failure mechanisms and to ease the fault-resolution problems. Ample diagnostic access has been provided for information transfer and for sequencing control from the LMC. The access has been carefully designed to prevent propagation of errors due to a failure and to take full advantage of the microprogram structure. The use of a programmable LMC for processor diagnosis minimizes the coupling between CC's.

Experience with this project has demonstrated that the design of a highly self-checking processor is an iterative process requiring continuous evaluation of circuit, program, and equipment design tradeoffs and making changes accordingly. It also indicates that early consideration of techniques for fault detection, diagnosis, and routine exercise on an overall basis is an absolute necessity. The computer simulation has proven to be a valuable tool for fault analysis. It has also been successfully used as a rapid means of incorporating design improvements.

ACKNOWLEDGMENT

Contributions made by R. C. Dorr and R. A. Elliott on processor design and fault simulation are gratefully acknowledged.

REFERENCES


Herbert Yu-Pang Chang (S'62-M'65), for a photograph and biography, see this issue, page 500.

Gary W. Heimbigner (S'68-M'71) was born in Auburn, Wash., on February 11, 1947. He received the B.S.E.E. degree from the University of Washington, Seattle, in 1969, and the M.S.E.E. degree from Northwestern University, Evanston, Ill., in 1971.

During the summer of 1968 he worked for the McDonnell Douglas Missile and Space Division. He became a member of the technical staff of Bell Telephone Laboratories, Inc., Naperville, Ill., in June 1969. His current work is in the area of design automation, systems reliability, and fault diagnosis.

Mr. Heimbigner is a member of Tau Beta Pi and an associate member of Sigma Xi.

Daniel J. Senese, for a photograph and biography, see this issue, page 500.

Thomas L. Smith (S'67-M'69) was born in Benton Harbor, Mich., on September 30, 1946. He received the B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1967 and 1969, respectively.

While at the Massachusetts Institute of Technology he was a Teaching Assistant in the Department of Electrical Engineering. He joined the technical staff of Bell Telephone Laboratories, Inc., in 1969, and has worked in both Naperville, Ill., and Holmdel, N.J. He is currently involved in designing minicomputer real-time monitor and control systems at the Holmdel Laboratory.

Mr. Smith is a member of the Association for Computing Machinery, Tau Beta Pi, and Eta Kappa Nu.