A Concurrent Fault Detection Method for Superscalar Processors

Alberto Palacios Pawlovsky and Makoto Hanawa

Hitachi Central Research Laboratory (HCRL), Hitachi Ltd.

Abstract
This paper describes a new method for the concurrent detection of faults in instruction level parallel (ILP) processors. This method makes use of the No Operation (NOP) instruction’s slots that sometimes fill some of the pipelines (stages) in an ILP processor. We show the practical application of this method to a superscalar RISC processor. For this processor, branch addresses, execution of certain instructions (store/load) and resource conflicts that force the inclusion of NOPs, are the cases exploited to test its pipelines. The NOPs are replaced by an effective instruction running in another pipeline. This allows the checking of the processor’s pipelines by the comparison of the outputs of their stages during the execution of the replicated instruction.

1 Introduction

Computer systems has evolved in the last 20 years from general purpose mainframes to mini and supercomputers and instruction processing has accordingly changed from sequential execution and pipeline processing to vector, superpipelining and superscalar processing. Some of these techniques can even be found implemented in some state-of-the-art RISC designs [1-3]. RISC processors appeared at the early 80’s as a counter trend to the development of complex instruction set computers (CISC).

The complexity previously found in CISC processors with control sections covering more than 50% of the chip area (as compared to the nearly 10% of the RISC processors) is now found in RISC processors in the form of multiple functional blocks embedded in one chip. The problem faced in CISC processor to test them, is found in RISC processors in the testing of their multiple functional units. Some test methodologies [4-9] have been studied and developed for the online testing of CISC designs, but they require an excessive redundant area or lead, due to delays in the data paths, to a degradation of the processing speed [10].

Recent RISC [1,2] and CISC [11] designs incorporate fault tolerant capabilities to connect them in a master-checker or active-shadow mode configuration through a lock-step operation. Also during power up, some chips [12] implement a BIST (Built-In Self Test) approach to test under a reset state all the registers and cache memory circuits.

Traditional concurrent error detection techniques have often employed hardware N-modular redundancy, N-version programming, recomputing with shifted operands (RESO), parity and SEC/DED codes [27]. Unfortunately, most redundancy techniques lead to large overhead and the error code-based techniques in most cases have limited applicability and restricted error detection capability.

In all ILP (e.g. pipelined, superscalar, VLIW, etc.) processors the degree of resource parallelism is fixed, while the instruction level parallelism available in an application varies from program to program and within a program during its execution. Therefore, due to limitations of the compiler or hardware to identify and exploit parallelism, there will be always execution cycles when resources of the processor are idle. For example, there are idle resources during execution of long latency operations in a RISC processor, in stalled cycles in a superscalar processor, and during scheduling of NOP instructions in a very long instruction word (VLIW) processor. Hence, during certain cycles, some of the functional units in a processor will be idle.

Attempts have been made in the past to exploit the unused resources of ILP processors or similar systems to make them fault-tolerant. A technique, called saturation, has been used in multitasking multiprocessor systems [13]. In this technique processors not needed to execute a task are used for error checking. Such processors execute replicated versions of a task running on the system. Then, the group of the replicated task uses majority-voting to determine the output of the group. Another approach [14] used the fact that the scalar computation unit in the CRAY-1 is rarely utilized in two consecutive instructions, to schedule RESO [15] tests and perform fault detection. Unutilized resources are also used in some signature monitoring techniques [16]. In these techniques, signatures are embedded in the application object code that follows unconditional branches. These signatures are fetched by the processor and used by the monitoring hardware to detect control-flow errors (CFE). Recently one technique that exploits the unused slots in ILP processors to run tracking and checking operations has been proposed for the detection of CFEs [17]. The available resource driven control-flow monitoring (ARC), as is called this technique, requires the modification of the original program code to embed tracking and checking operations in program’s segments.

Here we explores a new approach that uses the idle resources in ILP processors for concurrent fault detection in the hardware of the processor. It is worth to notice that even though the processor we use as a running example in the following sections is a RISC design and only has two pipelines, the testing method itself can be extended to any other ILP processor.

The following section describes our test approach in detail and the conditions under which it is applied. Section 3 briefly describes the CPU core used to evaluate our approach and shows details of its implementation. The evaluation of its requirements in area and those required to implement a gracefully degrading scheme are briefly discussed in
2 Fault Detection Method

As indicated above the approach to be described is not restricted to a specific ILP processor, but we will explain it for the superscalar processor described in the next section.

2.1 Conditions that Drive a Test Cycle

For the processing of instructions adopted in the superscalar processor shown in fig. 1 (two at a time) delay slots (or, equivalently, NOPs) will be needed when the address of a branch is odd, since in these cases the first instruction in the pair to be executed after the branch will have to be replaced by a NOP, or in cases that force the sequential execution of instructions, such as the store of data in the same address.

The method we propose makes use of the occurrence of these delay slots (NOPs) to exercise the pipelines and carry fault detection at some stages of them. This is accomplished substituting the NOPs by the instructions that will run in the pipeline that is not filled with a NOP. This results in one instruction and its copy running in two pipelines at the same time temporarily making the CPU to function, for the processing cycle of the replicated instruction, as a 2-redundant pipeline system. During this cycle, the comparison of the outputs of the pipeline stages will eventually led to the detection of errors and therefore faults in any one of them.

In multiple-pipeline ILP processors, if more than one pipeline happen to be filled with NOPs, the running instruction and its copies can be used to implement a majority-voting scheme and concurrently perform fault detection and location.

2.2 Implementation Requirements

The main requirement to implement the above fault detection scheme is to have the CPU pipelines identical (for the general case of more than two pipelines, all them must be identical).

For a processor implemented in this way fault detection is carried as follows. Under a branch condition and for the case of an address of jump that is odd, the instruction that forms a pair with the address of jump and usually is replaced by a NOP, will be replaced by a copy of the instruction pointed by the branch instruction. This as shown in fig. 2, is accomplished by using the multiplexer controlled by the Copy2 signal. Since both pipelines are identical the comparison of their outputs provides a way to determine if there exists a transient or permanent fault in any of them. In this way are tested not only the data and control paths of the pipelines, but also indirectly their associated logic (general registers, register file, input (output) latches, etc.).

For the case of a resource conflict (same I/O port, same address location, some kinds of data dependencies), the conflicting pair of instructions are forced to execute in sequence. The fault detection circuitry of fig. 2 is configured such that it allows the execution of two consecutive test
cycles in these cases. The first test cycle will duplicate the first instruction in pipeline 2 (under control of the Copy1 signal), and compare the results of the decoding, execution (ALU's results) and the write back stages of the pipelines. The logic in charge of performing the comparison and record any fault indication is the TSC (Totally Self-Checking) comparator and error buffer on the right (center) side of fig.2. The error buffer will hold a '0' for a no fault indication and '1' for a fault. The bits for the ID, EXE (ALU) and WB stages will be consecutively shifted from left to right during the execution of a test. The second test cycle will repeat this for the second instruction and its duplicate. In this cycle the fault/no fault bit indicator will be generated and stored in the error buffer on the left (center) side part of the fault detection circuitry. If in any of the test cycles an error occurs, the corresponding holding and error indicator will has its output at '1'. If both of the holding and error indicators have their outputs at '1' the output comparator will generate a '1' at the 'fault' output, signaling the occurrence of a possibly permanent fault.

![Circuit Diagram](image)

Figure 3. Test control signal generator of fig.2. DL.Y signals the sequential execution of instructions under a data dependency condition or the collision for the use of a common resource.

2.3 Fault Detection

Under a fault indication, the only comparison of the results will not suffice to determine which of the pipelines has caused the indication. The comparison of the results of a given test with subsequent ones could help to determine the nature of the fault. To isolate the faulty stage, the data in the error buffers can be used to determine the faulty stage(s). In fig.2 the fault detection circuitry is configured to permit the comparison of two consecutive tests when a pair of instructions is forced to execute sequentially. For those cases that only trigger one test cycle, one possible option was to alternatively switch between the right and left sides of the MUX, TSC comparator and error buffer pair and permit two single consecutive test cycles to compare. Instead, we opted for using the BOCopy2 signal to set the right holding and error indicator such that any error in a single test cycle will always set the 'fault' output. This approach requires a very simple logic for the clear of the error buffer and allows an immediate fault indication without having to wait for a second single test cycle.

3 The Application RISC Processor

The application circuit we used to quantitative evaluate our approach is the modified design of an experimental chip that has two 32-bit RISC processors, each one with a 500 MIPS (Million Instructions Per Second) peak performance [18]. The chip also has dedicated instruction and data caches, a 64-bit wide on-chip bus and shared (on-chip) 4-way interleaved 8Kbytes secondary memory. The CPU core of this chip is shown in figure 1. This figure also shows the blocks that handle the detection of data dependencies. The application processor has an instruction set with conditional branch and jump instructions.

![Timing Diagram](image)

Figure 4. Timing diagram that shows the execution of one test cycle for the occurrence of a branch with an odd address.
3.1 Fault Detection Logic

The fault detector is a symmetrical circuit. It has standard blocks for the multiplexors and its TSC comparators are a combination of 2-rail checkers and input logic that configure a circuit that for its test needs only 4 test patterns [19]. Due to the possibility of missing an error signal of short duration, holding and error indicators keeps track of the output signals of the TSC comparators. This holding and error indicator [28] is a less complex version of previous published designs [20-23] and is synchronized to the system clock by the Clock_T and Clock_T2 signals generated by the test control signals generator of figure 3. Between each TSC comparator and holding error indicator, we placed an error buffer to record the results of each phase of the test cycle. This logic will hold the results of comparing the outputs of the ID, EX (ALU) and WB stages. This data will serve as reference for the identification of the faulty functional stage and/or pipeline. For each clock of a test cycle a 1 (fault) or a 0 (no fault) will be stored and shifted in the error buffer.

3.2 Fault Detection Test Cycles: Examples

3.2.1 Single test cycle.- Figure 4 shows the time diagram of the execution of a single test cycle. This diagram shows two cases. The first one correspond to the case where the instruction read from the location pointed by PC+8 (second clock cycle) is an unconditional branch or a branch instruction with its condition found true. In this case the address of jump is shown to be even so there is no test cycle triggered by the branch. The second case (fourth clock cycle) illustrates the case where not only the branch is taken but also it has an odd jump address. These conditions will cause the rise of the BOCopy2 and Copy2 signals. The last one will copy the PC” instruction in pipeline 1 and create a window for clock_T2 that consecutively will shift the results of the test in the corresponding error buffer. The A_R_T2 and ID_T2 signals control the pass of the outputs of each stage through the multiplexers and into the TSC comparators. The DWR1 signal serves to avoid the writing of the results obtained in the run of the replicated instruction in pipeline 1.

3.2.2 Double test cycle.- Figure 5 shows the case where the data dependency between two instructions triggers the execution of two consecutive test cycles. In the figure the pair of instructions that has as address PC shows an interdependency that causes the program counter to temporarily stop (for the DLY ‘1’ cycle). The instructions coming from the instruction memory (Fig.1) will be sequentially executed and at the same time copied in their counterpart pipeline. In the last half of the 2nd. clock cycle, the data dependency detection logic (DDPI or SD of fig.1) rises DLY and triggers Copy1. This last one will copy, in the third clock cycle, the first instruction in pipeline 2 and start the test cycle for the first instruction. In the fourth cycle Copy2 will copy the second instruction in pipeline 1 and will start the test cycle for this instruction. The DWR2 as DWR1 is used to disable the change of registers or memory locations.

4 Evaluation of the Test Method

We have quantitatively evaluated the implementation of the concurrent fault detection method for the CPU core described in section 3. The requirements in area for the CPU without any fault detection capability is shown in table 1. The requirement in area for the implementation of our fault detection methodology is shown in table 2. In this table is also indicated its percentage respect to the CPU core area. As shown, it represents only a 3.7% increase in area.

<table>
<thead>
<tr>
<th>Functional Blocks</th>
<th>area (sq.um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode decoder</td>
<td>21,241.4</td>
</tr>
<tr>
<td>ID selecting logic</td>
<td>14,398.6</td>
</tr>
<tr>
<td>ID and EX registers</td>
<td>757,278.7</td>
</tr>
<tr>
<td>Kill registers</td>
<td>37,930.6</td>
</tr>
<tr>
<td>&quot;R&quot;, reg. decoder</td>
<td>110,626.6</td>
</tr>
<tr>
<td>&quot;S&quot;, reg. decoder</td>
<td>163,933.6</td>
</tr>
<tr>
<td>&quot;D&quot;, reg. decoder</td>
<td>174,493.5</td>
</tr>
<tr>
<td>Control of ALU’s</td>
<td>7,975.4</td>
</tr>
<tr>
<td>Clock enable logic</td>
<td>3,396.3</td>
</tr>
<tr>
<td>Program counter</td>
<td>1'673,517.7</td>
</tr>
<tr>
<td>ALU1</td>
<td>2'061,646.6</td>
</tr>
<tr>
<td>ALU2</td>
<td>2'061,646.6</td>
</tr>
<tr>
<td>Register file</td>
<td>1'993,449.6</td>
</tr>
<tr>
<td>Data dependency detection</td>
<td>91,024.6</td>
</tr>
<tr>
<td>Operand feedforward logic</td>
<td>170,058.2</td>
</tr>
<tr>
<td>Branch control logic</td>
<td>7,769.3</td>
</tr>
<tr>
<td><strong>CPU core area:</strong></td>
<td><strong>9372,147.5</strong></td>
</tr>
</tbody>
</table>

Figure 5. Time diagram that shows the application of our test method for the case of data dependency or collision for a common resource. The DWR1 and DWR2 signals are used to invalidate the results of the duplicated instruction.
4.1 Fault Location

Fault location is possible for the cases of two consecutive test cycles. Under a 'fault' indication and after the corresponding interrupt execution, the Scan-outE pin (fig.2) can be used to scan out the content of the error buffers. According to the content of these buffers (3-bit registers), it is possible to determine in some cases the faulty stage and/or pipeline. Suppose that the bits held in the buffers were respectively 100 and 100 (WB, ALU, ID test result from left to right). In this case it is possible to infer that there is a permanent fault in the write-back stage of any of the pipelines. If the bits were 110 and 100, it is not possible to say that any of the WB stages is faulty since a transient fault in one of the ALUs would have trigger the second '1' in the first triple and consequently cause the third bit to be '1' and, another transient fault in the WB stage could have trigger the third '1' in the second triple. Based in this kind of analysis it is possible to construct a fault table that would facilitate the analysis of a fault indication.

4.2 Fault Coverage

The kind of faults that can be detected is not restricted to any particular type but to any (transitory or permanent) that cause a different output at any stage of a given test cycle. As indicated in a previous section besides the paths of the pipelines, other paths and functional blocks are also tested indirectly. The instructions itself (instruction memory) are tested in the ID stage of a test cycle and the content of the register file is tested during the ALUs' outputs comparison. The test of the ALUs at the EXE stage of a test cycle, since it is based on the comparison of outputs of duplicated hardware, can be expected to have a better fault detection rate than RESO [15] or similar approaches [24] used to test arithmetic units. Of course, since the tests are not permanently executed, there will be always some intermittent faults that will not be detected. However, the detection of permanent faults can be expected to be comparable to that of a totally self-checking approach.

4.3 Gracefully Degrading Implementation Requirements

The fact of having the pipelines identical opens the possibility of configure the CPU core as a subsystem with gracefully degrading capability. Of course, any particular implementation will be subject to the way a 'fault' indication is handled and how is processed the corresponding interrupt. The design described in section 3 does not implement interrupts. The adoption of a reorder buffer, a history buffer, a future file [25] or any other configuration [26] will have to take into consideration that for two consecutive test cycles, the program counter value, corresponding to the instruction following the one that drove the test, will have to be preserved to execute an interrupt precisely. This will not add to the redundant logic since for the processor considered here and for any of the above interrupt schemes the PC value will have to be saved.

The setting of the 'fault' pin (figure2) will cause an interrupt that in turn will start the execution of the corresponding fault location routines, the isolation of the fault, and structure of the pipelines if necessary.

If one of the pipelines of the CPU of section 3 is found to contain a permanent fault, the sequential execution of instructions, in the remaining nonfaulty pipeline, can be enforced by the modification of the execution scheme adopted for the two cycle tests. Under a permanent fault, for every pair of instructions the DLV signal should be made to go high and the signals Copy1 and Copy2 redefined such that all the instructions can execute sequentially in the nonfaulty pipeline. This obviously will decrease the performance of the system but at the same time permit the continuous operation of the CPU.

5 Conclusions

We have showed a new concurrent method for fault detection in ILP processors. We have evaluated it for a two-pipeline RISC superscalar processor. For this processor the requirement in redundant area, to implement fault detection, is only of the order of 4%. This approach allows the detection of permanent and transitory faults. The detection rate of permanent faults can be expected to be comparable to a totally self-checking approach but without having the restriction of being limited to a certain kind of faults (e.g. unidirectional faults). Of course, it will not be real-time detection in all cases. The coverage of transient faults could be expected to be smaller than the one performed by a totally self-checking approach and it will depend on the frequency of the test cycles. The requirement of having identical pipelines, brings the benefit that the processor can be structured such that under a permanent loss of one of its pipelines it can function in a degraded way processing the instructions sequentially in the remaining pipeline.

The method described here does not introduce additional delays in the data or control paths since it exploit time slots that are usually filled with NOP instructions. The fault detection method also supports fault location. A primary approach to fault location has also been briefly introduced in the paper.

It is now under study the implementation of the fault detection method shown here to test the CPU core at the fabrication stage. The scheme we have shown here, can be extended to any ILP processor that handles cases were NOPs are used to fill its pipelines.

Table 2. Area requirements of the fault detection circuitry.

<table>
<thead>
<tr>
<th>Additional Blocks for Testing</th>
<th>area (sq.um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equalization of the WB registers</td>
<td>570.2</td>
</tr>
<tr>
<td>Equalization of the decoders</td>
<td>570.2</td>
</tr>
<tr>
<td>Logic to duplicate instructions</td>
<td>28,226.9</td>
</tr>
<tr>
<td>Fault detection logic</td>
<td>309,133.4</td>
</tr>
<tr>
<td>Test Control Signal Generator</td>
<td>24,449.0</td>
</tr>
<tr>
<td>CPU + testing logic area:</td>
<td>9,735,097.2</td>
</tr>
<tr>
<td>Percentage of redundant area:</td>
<td>3.7%</td>
</tr>
</tbody>
</table>

The design described in section 3 does not implement interrupts. The adoption of a reorder buffer, a history buffer, a future file [25] or any other configuration [26] will have to take into consideration that for two consecutive test cycles, the program counter value, corresponding to the instruction following the one that drove the test, will have to be preserved to execute an interrupt precisely. This will not add to the redundant logic since for the processor considered here and for any of the above interrupt schemes the PC value will have to be saved.

The setting of the 'fault' pin (figure2) will cause an interrupt that in turn will start the execution of the corresponding fault location routines, the isolation of the fault, and structure of the pipelines if necessary.

If one of the pipelines of the CPU of section 3 is found to contain a permanent fault, the sequential execution of instructions, in the remaining nonfaulty pipeline, can be enforced by the modification of the execution scheme adopted for the two cycle tests. Under a permanent fault, for every pair of instructions the DLV signal should be made to go high and the signals Copy1 and Copy2 redefined such that all the instructions can execute sequentially in the nonfaulty pipeline. This obviously will decrease the performance of the system but at the same time permit the continuous operation of the CPU.

5 Conclusions

We have showed a new concurrent method for fault detection in ILP processors. We have evaluated it for a two-pipeline RISC superscalar processor. For this processor the requirement in redundant area, to implement fault detection, is only of the order of 4%. This approach allows the detection of permanent and transitory faults. The detection rate of permanent faults can be expected to be comparable to a totally self-checking approach but without having the restriction of being limited to a certain kind of faults (e.g. unidirectional faults). Of course, it will not be real-time detection in all cases. The coverage of transient faults could be expected to be smaller than the one performed by a totally self-checking approach and it will depend on the frequency of the test cycles. The requirement of having identical pipelines, brings the benefit that the processor can be structured such that under a permanent loss of one of its pipelines it can function in a degraded way processing the instructions sequentially in the remaining pipeline.

The method described here does not introduce additional delays in the data or control paths since it exploit time slots that are usually filled with NOP instructions. The fault detection method also supports fault location. A primary approach to fault location has also been briefly introduced in the paper.

It is now under study the implementation of the fault detection method shown here to test the CPU core at the fabrication stage. The scheme we have shown here, can be extended to any ILP processor that handles cases were NOPs are used to fill its pipelines.

Table 2. Area requirements of the fault detection circuitry.

<table>
<thead>
<tr>
<th>Additional Blocks for Testing</th>
<th>area (sq.um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equalization of the WB registers</td>
<td>570.2</td>
</tr>
<tr>
<td>Equalization of the decoders</td>
<td>570.2</td>
</tr>
<tr>
<td>Logic to duplicate instructions</td>
<td>28,226.9</td>
</tr>
<tr>
<td>Fault detection logic</td>
<td>309,133.4</td>
</tr>
<tr>
<td>Test Control Signal Generator</td>
<td>24,449.0</td>
</tr>
<tr>
<td>CPU + testing logic area:</td>
<td>9,735,097.2</td>
</tr>
<tr>
<td>Percentage of redundant area:</td>
<td>3.7%</td>
</tr>
</tbody>
</table>
Acknowledgments:

We want to thank the leader of the 6th.Dept., 602 research unit, Mr.Kunio Uchiyama for his critical review of the paper.

References: