Continuous Reliability Monitoring Using Adaptive Critical Path Testing

Bardia Zandian*, Waleed Dweik, Suk Hun Kang, Thomas Puniahole, Murali Annavaram
Electrical Engineering Department, University of Southern California
Mail Code: EEB200, University of Southern California, Los Angeles, CA 90089
Tel:(213)740-3299, {bzandian,dweik,sukhunka,puinhaol,annavara}@usc.edu

Abstract

As processor reliability becomes a first order design constraint, this research argues for a need to provide continuous reliability monitoring. We present an adaptive critical path monitoring architecture which provides accurate and real-time measure of the processor’s timing margin degradation. Special test patterns check a set of critical paths in the circuit-under-test. By activating the actual devices and signal paths used in normal operation of the chip, each test will capture up-to-date timing margin of these paths. The monitoring architecture dynamically adapts testing interval and complexity based on analysis of prior test results, which increases efficiency and accuracy of monitoring. Experimental results based on FPGA implementation show that the proposed monitoring unit can be easily integrated into existing designs. Monitoring overhead can be reduced to zero by scheduling tests only when a unit is idle.

Keywords: Reliability, Critical Paths, Timing Margins
Submission Category: Regular Paper for DCCS

1. Introduction

Reduced processor reliability is one of the negative repercussions of silicon scaling. Reliability concerns stem from multiple factors, such as manufacturing imprecision that leads to several within-in die and die-to-die variations [7, 8, 10], ultra-thin gate-oxide layers that breakdown under high thermal stress, Negative Bias Temperature Instability (NBTI) [2], and Electromigration. Many of these reliability concerns lead to timing degradation first and eventually lead to processor breakdown [7]. Timing degradation occurs extremely slowly over time and can even be reversed in some instances, such as those caused by NBTI effects. When individual device variations are taken into consideration, timing degradation is hard to predict or accurately model analytically. Most commercial products solve timing margin degradation problem by inserting a guardband at design and fabrication time. Guardband reduces the performance of a chip during the entire lifetime just to ensure correct functionally in a small fraction of time during the late stages of the chip lifetime.

Our inability to precisely and continuously monitor timing degradation is the primary reason for over-provisioning of resources. Without an accurate and real-time measure of timing margin, designers are forced to use conservative guardbands and/or use expensive error detection and recovery methods. Processors currently provide performance, power, and thermal monitoring capabilities. These capabilities are exploited by designers and developers to understand and debug performance and power problems and design solutions to address them. In this paper, we argue that providing reliability monitoring to improve visibility into the timing degradation process is equally important to future processors as there will be an ever increasing need to monitor device reliability. Such monitoring capability enables just-in-time activation of error detection and recovery methods, such as those proposed in [1, 13, 16, 17, 24, 25].

The primary contribution of this research is to propose a runtime reliability monitoring framework that uses adaptive critical path testing. The proposed mechanism injects specially designed test vectors into a circuit-under-test (CUT) that not only measure functional correctness of the CUT but also its timing margin. The outcomes of these tests are analyzed to get a measure of the current timing margin of the CUT. Furthermore, a partial set of interesting events from test injection results are stored in flash memory to provide an unprecedented view into timing margin degradation process over long time scales. For monitoring to be effective, we believe, it must satisfy the following three criteria:

1. Continuous Monitoring: Unlike performance and power, reliability must be monitored continuously over extended periods of time; possibly many years.
2. Adaptive Monitoring: Monitoring must dynamically adapt to changing operating conditions. Due to differences in device activation factors and device variability, timing degradation rate may differ from one CUT to the other. Even a chip in early stages of its expected lifetime can become vulnerable due to
aggressive runtime power and performance optimizations such as operating at near-threshold voltages [15] and higher frequency operation [17, 24]. Some effects such as NBTI related timing degradation are even reversible [2]. Thus there is a need to adapt the monitoring mechanisms to match the operating conditions at runtime.

3. Low Overhead Monitoring: The monitoring architecture should have low area overhead and design complexity. Obviously, monitoring framework should be implementable with minimal modifications to existing processor structures, and should not in itself be a source of errors.

The reliability monitoring approach introduced in this paper is designed to satisfy the above criteria. Using the proposed monitoring mechanism has many benefits. With continuous, adaptive and low-overhead monitoring, conservative preset guardbands can be tightened. Unlike current approaches where error detection and correction mechanisms are continuously enabled, processor can deploy preemptive error correction measures during in-field operations only when the timing margin is small enough to affect circuit functionality. Furthermore, the unprecedented view provided by continuous monitoring will enable designers to correlate predicted behavior from analytical models with the in-field behavior and use these observations to make appropriate design changes for future processors.

The rest of this paper is organized as follows: Section 2 explains our proposed monitoring architecture and how we achieve the three desired criteria mentioned above. Section 3 shows the experimental setup used to evaluate the effectiveness of our proposed architecture. Results from our evaluations are discussed in Section 4. Section 5 describes related work and compares the proposed approach to prior work. We conclude in Section 6.

2. Reliability Monitoring Framework

The reliability monitoring framework proposed in this paper is based on the notion of critical path tests. Specially designed test vectors are stored in an on-chip repository and are selected for injection into the CUT at specified time intervals. Using outcomes from these tests, the monitoring unit measures the current timing margin of the CUT. In the following two subsections we describe the architecture of the proposed reliability monitoring unit and how critical path test vectors are selected in detail.

2.1. Architecture of the Monitoring Unit

Figure 1(a) shows the overview of the proposed Reliability Monitoring Unit (RMU) and how it interfaces with the CUT. In our current design we assume that a CUT may contain any number of data or control signal paths that end in a flip-flop, but it should not contain intermediate storage elements. The four shaded boxes in the figure are the key components in our proposed RMU. The first component is the Test Vector Repository (TVR). TVR holds a set of test patterns and the expected correct outcomes when these test patterns are injected into the CUT. TVR will be filled once with CUT-specific test vectors at post-fabrication phase. We describe the process for test vector selection in more detail in Section 2.2. Multiplexer, MUX1, is used to select either the regular operating frequency of the CUT or one test frequency from a small set of testing frequencies. Test frequency selection will be described in Section 2.3. Multiplexer, MUX2, on the input path of the CUT allows the CUT to receive inputs either from normal execution trace or from TVR. MUX1 input selection is controlled by the Freq. Select signal and MUX2 input selection is controlled by the Test Enable signal. Both these signals are generated by the Dynamic Test Control (DTC) unit.

DTC selects a set of test vectors from TVR to inject into the CUT. After each test vector injection the CUT output will be compared with the expected correct output and a test pass/fail signal is generated. For every test vector injection an entry is filled in the Reliability History Table (RHT). Each RHT entry stores a time stamp of when the test is conducted, test vector index, testing frequency, pass/fail result, and environment conditions such as CUT temperature. RHT is implemented as a two level FIFO structure where the first level (RHT-L1) only stores the most recent test injection results on an on-die SRAM structure. The second level RHT (RHT-L2) is implemented on a flash memory that can store test history information over multiple years and can be implemented on the same package but on a different die, similar to multi-chip-packages. While RHT-L1 stores a complete set of prior test injection results within a small time window, RHT-L2 stores only interesting events, such as test failures, excessive thermal gradients. DTC only reads RHT-L1 data to determine when to perform the next test as well as how many circuit paths to test in the next test phase.

2.2. Test Vector Selection

Prior studies using industrial CPU designs, such as Intel Core 2 Duo processor [3], show that microarchitectural block designs often result in three groups of circuit paths. First group contains few paths (< 1%) with zero timing margin; second group contains several paths (about 10%) with less than 10% timing margin, followed by a vast majority of paths (about 90%) with a large timing margin. Verification and test engineers spend significant amount of effort to analyze the first two sets of critical paths and generate test vectors to activate these paths for pre and post-fabrication testing purposes. Obviously, these paths are the ones with the least amount of timing margin and are most susceptible to timing failures. Even in the absence of manual effort to
identify these paths, standard place-and-route tools can also classify the paths into categories based on their timing margin and then generate test vectors for activating the paths. We propose to exploit the effort already spent either by the test designers or the CAD tools to create critical path test vectors. TVR is thus initially filled with test vectors that test paths with less than 10% timing margin. Based on results shown in [3], even for very large and complex CUTs such as cache controllers, TVR may store in the order of 50-100 test vectors. In our current implementation, TVR stores vectors in the sorted order of their timing criticality once during the post fabrication phase. Compression techniques can be used to further reduce the already small storage needs of TVR.

DTC makes decisions regarding which vectors to inject and when to inject. Note that the minimum number of test vectors needed for one path to get tested for a rising or a falling transition is two. Instead of using just two input combinations that exercises the most critical path in the CUT, multiple test vectors that exercise a group of critical paths in the CUT will be used in each test phase. Thus, test vectors used in each test phase are a small subset of the vectors stored in TVR. This subset is dynamically selected by DTC based on the history of test results and CUT’s operating condition. Initially DTC selects test vectors in the order from the most critical (slowest) path at design time to less critical paths. As path criticality changes during the lifetime of the chip, cases might be observed where paths that were initially thought to be faster are failing while the expected slower paths are not. Hence, the order of the critical paths can be dynamically updated by the DTC by moving the failing input patterns to the top of the test list. To account for the unpredictability in device variations, DTC also randomly selects additional test vectors from TVR during each test phase making sure that all the paths in the TVR are checked frequently enough. One simple test vector selection approach is to use top eight critical path test vectors stored in TVR followed by two test vectors that are selected in a round-robin fashion from the remaining test vectors in TVR. This multi-vector test approach allows more robust testing since critical paths may change over time due to different usage patterns, device variability, and difference in the number of devices present on each signal path.

While our approach ensures that dynamic variations of critical paths are accounted for, the fundamental assumption is that a non critical path that is not tested by vectors in TVR will not fail while all the critical paths that are tested through TVR are still operational. It is important to note that due to the physical nature of the phenomena causing aging related timing degradation, the probability of a path which is not in the TVR failing before any of the circuit paths that are being tested is extremely low. Studies on 65nm devices show that variations in threshold voltage due to NBTI are very small and gradual over time. In particular, the probability of a large variation in threshold voltage which result in large path delay changes is nearly zero [18, 14]. Thus the probability of an untested path failing is the same as the probability of a path with timing margin greater than the guardband suddenly deteriorating to the extent that it violates timing. Chip designers routinely make the assumption that such sudden deterioration is highly unlikely while selecting the guardband. Hence we believe our fundamental assumption is acceptable by industrial standards.

2.3. Test Frequency Selection

In order to accurately monitor the remaining timing margin of a path in the CUT, a paths gets tested at multiple test frequencies above its nominal operation frequency. The highest frequency at which the path passes a test, provides us with information regarding the current level of timing degradation that the path has experienced. The test frequency range is selected between nominal operating frequency and frequency without a guardband. This range is then divided into multiple frequency steps. When using the proposed RMU, it would be possible to reduce default guardband to a smaller value while still meeting the same FIT goal. This is shown on Figure 1(b), where \( T_{\text{clk1}} = 1/f_{\text{clk1}} \) > \( T_{\text{clk2}} = 1/f_{\text{clk2}} \) > \( (\Delta D_{\text{init}}) \). \( T_{\text{clk1}} \) is the original clock period when using the default guardband and \( T_{\text{clk2}} \) is the clock period with a reduced guardband (RGB). Note that in either case the initial path delay of the CUT is the same, \( \Delta D_{\text{init}} \). The purpose of RMU is to continuously monitor the CUT and check if CUT timing is encroaching into the reduced guardband. For instance, if \( \Delta D_{\text{init}} \) is 0.33 nanoseconds (3GHz) and RGB is 10% then...
the nominal operating cycle time of the CUT will be 0.36 nanoseconds (2.7 GHz). Since the CUT is operating at 2.7 GHz with a 10% timing margin, RMU uses the RGB of 10% as the test frequency range. The test frequency range is then divided into 10 equal frequency steps between 2.7 GHz and 3 GHz. For each test vector selected by DTC, the CUT is tested at each of the 10 test frequency steps to measure the current timing margin of the CUT. It should be noted that using RGB instead of the more conservative guardband is not necessary for correct operation of the proposed monitoring unit, and RGB is a performance improvement that is made possible with RMU. If this potential opportunity to increase the operation frequency is not taken advantage of, the default processor guardband can be used in place of RGB in all of the above discussion.

Our approach uses a narrow testing frequency range which is subdivided into multiple frequency steps. Providing such small frequency steps is not a difficult challenge. Prior research has shown that it is feasible to provide internal clocking capabilities in increments of 2 MHz with limited design effort [26]. Alternatively, an aging resilient delay generator such as the one proposed in [1] can be used with minimal area and power overhead.

2.4. DTC and Opportunistic Tests

DTC is the critical decision making unit that determines the interval between tests (called interval throughout the rest of the paper) and the number of test vectors to inject (called complexity) during each test phase. These two design choices would exploit tradeoffs between increased accuracy and decreased performance due to testing overhead.

DTC reads the most recent RHT entries to decide the interval and complexity of the next testing phase. The 100 most recent RHT entries are analyzed by DTC to see if any tests have failed during the testing phase. Each failed test entry will indicate which test vector did not produce the expected output and at what testing frequency. DTC then selects the union of all the failed test vectors to be included in the next phase of testing. If no prior tests have failed, DTC simply selects a set of test vectors from the top of TVR. We explore two different choices for the number of vectors selected in our results section.

Once the test complexity has been determined DTC then decides the test interval. There are two different approaches for selecting when to initiate the next test phase. In the first approach DTC initially selects a large test interval, say 1 million cycles between two test phases and then DTC dynamically alters the test interval to be inversely proportional to the number of failures seen in the past few test phases. For instance, if two failures were noticed in the last eight test phases then DTC decreases the new test interval to be half of the current test interval.

An alternate approach to determine test interval is opportunistic testing. In this approach DTC initiates a test injection only when the CUT is idle thereby resulting in zero performance overhead. Current microprocessors provide multiple opportunities for testing a CUT. We describe a few such opportunities. On a branch misprediction the entire pipeline is flushed and instructions from the correct execution path are fetched into the pipeline. Since the newly fetched instructions take multiple cycles to reach the backend, the execution, writeback and retirement stages of the pipeline are idle waiting for new instructions. When a long latency operation such as a L2 cache miss is encountered, even aggressive out-of-order processors are unable to hide the entire miss latency thereby stalling the pipeline. On an I-cache miss the fetch stage of the pipeline is stalled waiting for the cache miss, which provides an opportunity to test the fetch stage of the pipeline. In addition to these common events, there are several rare events such as exceptions that cause a pipeline flush which provide testing opportunities. Finally, computer system utilization rarely reaches 100% and the idle time between two utilization phases provides an excellent opportunity to test any CUT within the system. We quantify the distance between idle times and duration of idle time for a select set of microarchitectural blocks in the experimental results section.

DTC can automatically adapt to the reliability needs of the system. For a CUT which is unlikely to have failures during the early stages of its in-field operation, test interval and complexity can be reduced. Even if an opportunistic testing approach is used one can simply ignore most opportunities when the CUT is working well and use only a few opportunities for testing. As the CUT ages or when the CUT is vulnerable due to low power settings, DTC can increase testing. Note that the time scale for test interval is extremely long and for instance, NBTI related timing degradation occurs only after many seconds or minutes of intense activity in the CUT. Hence testing interval will be in the order of seconds even in the worst case. We will explore performance penalties for different selections of test intervals and test complexities in the experimental results section.

2.5. Design Issues

We would like to note that there are design alternatives to several of the RMU components described. For implementing variable test frequencies, there are existing infrastructures for supporting multiple clocks within a chip. For instance, DVFS is supported on most processors for power and thermal management. While the current granularity of scaling frequency may be too coarse, it is possible to create much finer clock scaling capabilities as described in [26].

The RHT-L2 size can also be quite small. For instance, each test vector generates 2 bytes of data in our current implementation which can be reduced further by using compression techniques. Since testing will be done rarely, say
once every minute, even a small 64KB RHT-L2 can store nearly 45 days worth of continuous test data. By selectively storing only failed test vectors in RHT the storage capacity can be significantly prolonged to store multiple months of data using a relatively small flash.

While in our description each RMU is associated with one CUT, it is possible to have multiple CUTs share a single RMU. This RMU can maintain multiple RHTs and TVRs for different CUTs and use a single DTC logic block to make the testing decisions. In addition, a system-wide DTC can combine data from multiple local RHTs in a hierarchical manner to make an appropriate global testing decision.

Note that using opportunistic testing a CUT is never taken offline. Rather we use the idle time to inject test vectors. Even if we do not use opportunistic testing, we should emphasize that testing the CUT does not in itself lead to noticeable increase in aging of the CUT. The percentage of time a CUT is tested is negligible compared to the normal usage time. For instance, if the CUT is tested once a second with 1000 tests (an extremely aggressive testing scenario) for a system which has a 1GHz clock, our approach increases the CUT usage by 0.0001%. This is a negligible penalty we pay for providing continuous monitoring.

3. Experimental Methodology

Our proposed approach is not an error correcting mechanism. It provides monitoring capabilities which may in-turn be used to more effectively apply prior error detection or correction mechanisms. As such, monitoring by itself does not improve FIT rate. The following are the aims of our experimental design: First, area overhead and design complexity of RMU are measured using close-to-industrial design implementation of the RMU and all the associated components. Such a design also provides us an opportunity to verify that RMU does not negatively impact the CUT performance during the its normal operation. Second, the state space of DTC and RHT interactions is explored and different test interval and test complexity parameters are studied. Third, overhead of monitoring in measured in terms of how much monitoring may reduce performance. As described earlier, the overhead of testing can be reduced to zero if testing is done opportunistically. We use software simulation to measure the duration of each opportunity (in cycles) and the distance between two opportunities. In all our experiments instead of waiting for the occurrence of rare stochastic events, effects of timing degradation are deliberately accelerated to measure the overheads and the results produced by this assumption only show worst case overhead of monitoring compared to real implementation.

3.1. FPGA Emulation Setup

We modeled the proposed RMU design, including TVR, DTC and RHT using Verilog HDL. We then selected a double-precision Floating Point Multiplier Unit (FPU) as the CUT that will be monitored. The FPU selected in our design implements 64-bit IEEE 754 floating point standard. It is a single stage combinatorial block that contains thousands of circuit paths. Hence, the FPU gives us credible enough results due to its size and complexity. We focus most of our experimental design effort on the implementation of RMU.

RHT-L1 is implemented as 256 entry table using FPGA SRAMs. During each test phase the test vector being injected, testing frequency, test result and operating conditions are stored as a single entry in RHT-L1. When a test fails the test result is also sent to the RHT-L2, which is implemented as an unlimited table on CompactFlash memory. When RHT-L1 is full the oldest entry in the RHT-L1 will be overwritten, hence, DTC can only observe at most the past 256 tests for making decisions on selecting the test interval. In our implementation DTC reading results from RHT does not impact the clock period of the CUT and it does not interfere with the normal operation of the CUT. Furthermore, reading RHT can be done simultaneously while test injection results are being written to RHT.

One important issue that needs to be addressed is the difference which exists between FPU’s implementation on FPGA as compared to ASIC (Application Specific Integrated Circuit) synthesis. The critical paths would be different between these two implementations. These differences would also be observed between instances of ASIC implementations of the same unit when different CAD tools are used for synthesis and place and route (PAR), or between different fabrication technology nodes. However, to maximize implementation similarity to ASIC designs, the FPU is implemented using fifteen DSP48E Slices which exist on the Virtex 5 XC5VLX110T FPGA chip. These DSP slices use dedicated combinational logic within the FPGA rather than the traditional SRAM LUTs used in conventional FPGA implementations. Using these DSP slices increases the efficiency of the FP multiplier and would make its layout and timing characteristics more similar to industrial ASIC implementation. Recall that in our RMU each CUT will have a corresponding TVR that stores test vectors that exercise most critical paths specific to that CUT. Since TVR stores CUT-specific test vectors, the potential differences in critical paths between different implementations do not negatively impact any of our qualitative observations and results presented here.

Section 2.2 described an approach for selecting test vectors to fill TVR by exploiting designer’s efforts to characterize and test critical paths. For the FPU selected in our experiments we did not have access to any existing test vector data. Hence, we devised a simple approach to generate test vectors using benchmark driven input vector generation as described in this section. We selected a set of
5 CPU2K floating point benchmarks, Applu, Apsi, Mgrid, Swim, Wupwise, and generated a trace of floating point multiplies from these benchmarks by running them on SimpleScalar [4] integrated with Watch [11] and Hotspot [21]. The simulator is configured to run as a 4-way issue Out-Of-Order processor with Pentium-4 processor layout. The first 300 million instructions in each benchmark were skipped and then the floating point multiplication operations in the next 100 million instructions were recorded. The number of FP multiplies recorded for each trace ranges from 4.08 million (Wupwise) to 24.39 million (Applu). Each trace record contains the input operand pair and the expected correct output value and the temperature of the chip; we will shortly explain how the temperature is used to mimic processor aging. These traces are stored on a CompactFlash memory that is accessible to the FPU.

Initially the FPU is clocked at its nominal operating frequency reported by the CAD tool and all input traces produce correct outputs. The FPU is then progressively overclocked in incremental steps to emulate the gradual timing degradation that the FPU experiences during its lifetime. We then feed all input traces to the FPU at each of the incremental overclocked step. At the first overclocking step, above the nominal operating frequency, input operand pairs that exercise the longest paths in the FPU will fail to meet timing. At the next overclocking step more input operand pairs that exercise the next set of critical paths will fail, in addition to the first set of failed input operand pairs.

In our design, FPU’s nominal clock period is 60.24 nanoseconds (16.6 MHz) where all the input operand pairs generate the correct output. We then overclocked the FPU from 60.24 nanosecond clock period to 40 nanosecond clock period in equal decrements of 6.66 nanoseconds. These correspond to 4 frequencies: 16.6 MHz, 18.75 MHz, 21.42 MHz, 25 MHz. The clock period reduction in each step was the smallest decrement we could achieve on the FPGA board. The percentage of paths failing to meet timing gradually increase at each step and the fail rate increases dramatically above 25 MHz. Percentage of the FP multiply instructions that failed as the test clock frequency was increased are shown in Figure 2. Results show the number of input vectors that failed to produce correct output for one CUT but with multiple benchmarks. When the CUT is initially overclocked by a small amount the number of input vectors that fail is relatively small. These are the vectors that activate the critical paths with the smallest timing margin. However, as CUT is overclocked beyond 21.42 MHz (essentially simulating a large timing degradation), not surprisingly, almost all input vectors fail across all benchmarks. The knee in the curves indicates that, almost all paths in the FPU have combinational delays that would not be met at frequencies above 21.42 MHz and hence nearly all input test vectors will fail at that frequency. Finally, as to

![Figure 2. Timing Margin Degradation](image)

be expected, CUT timing violations are mostly independent of the benchmarks. Hence, we do not need to select a large number of benchmarks to demonstrate the results of RMU.

We selected the top 1000 failing input operand pairs as the test vectors to fill the TVR. Since the minimum clock period decrease is 6.66 nanoseconds the number of input test vectors that fail increases in very large increments with each overclocking step and we were forced to fill TVR with many more test vectors than needed. Input test vectors could be differentiated more precisely with finer changes to the clock frequency, thereby reducing the size of TVR.

### 3.2. Three Scenarios for Monitoring

We emulated three scenarios for measuring the monitoring overhead. The three scenarios are: Early-stage monitoring, Mid-stage monitoring, Late-stage monitoring.

**Early-stage monitoring:** In early-stage monitoring we emulated the conditions of a chip which has just started its in-field operations, for instance, the first year of chip’s operation. Since the amount of timing degradation is relatively small, it is expected that the CUT will pass all the tests conducted by RMU. To emulate this condition, our RMU tests the CUT at only the nominal frequency, which is 16.6 MHz in our experimental setup. The test vectors injected into the CUT do not produce any errors and hence DTC does not change either test interval or test complexity. We explored two different test intervals, where tests are conducted at intervals of 100,000 cycles and 1,000,000 cycles. At each test injection phase we also explored two different test complexity setting. We used either a test complexity of 5 test vectors or 20 test vectors for each test injection phase.

**Mid-stage monitoring:** In mid-stage monitoring we emulated the conditions when the chip’s timing margin has degraded just enough such that some of the circuit paths that encroach into the reduced guardband (RGB as described in Section 2.3) of the CUT start to fail. These failure are detected by RMU when it tests the CUT with a test frequency near the high end of the test frequency range. We emulated this failure behavior by selecting the frequency for testing the CUT to be higher than the nominal operation frequency of the CUT. In our experiments we used 18.75 MHz for testing, which is 12.5% higher than the nominal
frequency. Since this testing frequency mimics a timing degradation of 12.5% some test vectors are bound to fail during CUT testing. In mid-life monitoring DTC’s adaptive testing is activated where it dynamically changes the test interval. In our current implementation DTC simply computes the number of failed tests seen in the last 8 test injections and then uses that count to determine when to activate the next test phase. We explored two different test interval selection mechanisms, the first scheme uses linear decrease in the test interval as the fail rate of the tests increase. The maximum (initial) test interval selected for the emulation purpose is 100,000 cycles and this will be reduced in equal steps up to 10,000 cycles when the fail rate is detected as being 100%. For instance, when the number of test failures in the last eight tests is zero then DTC makes a decision to initiate the next test after 100,000 cycles. If one out of the eight previous tests has failed then DTC selects 88,750 cycles as the test interval (100,000-(90,000/8)). An alternative scheme uses initial test interval of 1 million cycles and then as the error rate increases the test interval is reduced in eight steps by dividing the interval to half for each step. For instance, when the number of test failures in the last eight tests is zero then DTC makes a decision to initiate the next test after 1,000,000 cycles. If two out of the eight previous tests have failed then DTC selects 250,000 cycles as the test interval (1,000,000/2^2). The exponential scheme uses more aggressive testing when the test fail rate is above 50% but it would do significantly more relaxed testing when the fail rate is below 50%. These two schemes provide us an opportunity to measure how different assumptions about DTC adaptation policies will translate into testing overheads. Note that many of the parameters mentioned here, such as test interval, test complexity and number of RHT entries to analyze, are design space options.

Late-stage monitoring: In the late-stage monitoring scenario the chip has aged considerably. Timing margin has reduced significantly with many paths in the CUT operating with limited timing margin. In this scenario, path failure detected by RMU are not only dependent on the test frequency but are also dependent on the surrounding operating conditions. For instance, a path that has tested successfully across the range of test frequencies under one operating condition may fail when the test is conducted under a different operating condition. The reason for the prevalence of such a behavior is due to non uniform sensitivity of the paths to effects such as Electromigration and NBTI that occur over long time scales.

The goal of our late-stage monitoring emulation is to mimic the pattern of test failures as discussed above. One reasonable approximation to emulate late-stage test failure behavior is to use temperature of the CUT at the time of testing as a proxy for the operating condition. Hence, at every test phase we read the CUT temperature which has been collected as part of our trace to emulate delay changes. For every 1.25°C raise in temperature the baseline test clock period is reduced by 6.66 nanoseconds. We recognize that this is an unrealistic increase in test clock frequency with a small temperature increase. However, as mentioned earlier our FPGA emulation setup restricts each clock period change to a minimum granularity of 6.66 nanoseconds. This assumption forces the RMU to test the CUT much more frequently than would be in a real world scenario. The adaptive testing mechanism of DTC is much more actively invoked in the late-stage monitoring scenario.

3.3. Dynamic Adaptation of RMU

Figure 3 shows how DTC dynamically adapts the test interval to varying wearout conditions in Late-stage monitoring scenario with TC=20 and using the linear test interval adaption scheme. The data in this figure represents a narrow execution slice from Apsi running on FPGA implementation of FPU which is being monitored by RMU. The horizontal axis shows the trace record number, which represents the progression of execution time. The first plot from the bottom shows the test fail rate seen by DTC. Recall that DTC reads the last 8 test injection results from RHT and counts the number of failures. The highlighted oval area shows a dramatic increase the test fail rate and the middle plot shows how DTC reacts by dynamically changing the interval between Test Enable signals. Focusing again on the highlighted oval area in the middle plot, it is clear that the test interval has been dramatically reduced since gap between successive test phases has shrunk. The top plot zooms in on one test phase to show 20 back-to-back tests that were injected into the CUT corresponding to our TC=20 setting. Figure 3 confirms that our proposed RMU design dynamically adapts to changing wearout conditions.

4. Experimental Results

In this section results related to area and design overhead of RMU, in addition to performance overhead of monitoring are presented. Opportunities to perform tests without interrupting the normal operation of the processor are studied in Section 4.3.
4.1. Area Overhead of RMU

RMU and FPU implemented on FPGA utilize 4994 FPGA slices out of which 4267 are used by the FPU and 727 slices are used for the RMU implementation. Out of the 8818 SRAM LUTs used in our design the RMU consumes 953 LUTs while the remaining 7865 LUTs are used by the FPU. The FPU also uses 15 dedicated DSP48E slices for building the double precision FP multiplier, while only one DSP48E slice is used by RMU logic. Note that overall area overheads of monitoring can be reduced by increasing the granularity of CUTs being monitored, say increase the CUT size from FPU to the whole execution pipeline stage of a core. Since RMU size stays relatively constant irrespective of CUT being monitored, the strength of the proposed scheme is this ability to adjust the CUT size so as to make the RMU area overhead acceptable for any design.

4.2. Monitoring Overhead

Figure 4(a) shows the execution time overhead of test- ing compared to the total execution time of the benchmark traces. The horizontal axis shows the three monitoring scenarios, Early-stage (labeled Early), Mid-stage (Mid) and Late-stage (Late) monitoring. Vertical axis shows the number of test injections as a percentage of the total trace length collected for each benchmark. DTC uses linear decrease in test interval from 100,000 cycles to 10,000 cycles depending on the test fail rates. Results with test complexity (TC) of 5 and 20 test vectors per test phase have been shown. The early-stage monitoring overhead is fixed for all the benchmarks and depends only on the test interval and complexity. Testing overhead varies per benchmark during Mid and Late testing. The reason for this behavior is that benchmarks which utilize the CUT more frequently, i.e. benchmarks that have more FP multiplications, will increase CUT activity factor which in turn would accelerate CUT degradation. Degradation would be detected by DTC and it may increase the monitoring overhead to test the CUT more frequently. The worst case overhead using late-stage monitoring scenario is still 0.07%.

Figure 4(b) shows the same results when DTC uses exponential test intervals. Note that the vertical axis scale range is different between Figure 4 parts (a) and (b). Comparing the percentage of time spent in testing between the linear and exponential interval adjustment schemes, it is clear that exponential method results in less testing overhead in almost every one of the emulated scenarios. This observation is a direct result of the fact that exponential test intervals start with a higher initial test interval setting and DTC only decreases the testing interval to conservative low values when test failures are detected. Figure 5 shows the percentage of the tests that have failed in each of the emulation schemes described earlier for Figure 4. The vertical axis shows only mid-stage and late-stage monitoring schemes since early-stage does not generate any test failures. Only the linear test interval scheme results are shown for both the test complexities emulated, TC=5 and TC=20. Test fail rates increase dramatically from mid-stage to late-stage scenario, as expected during in-field operation. Benchmarks such as Apsi and Wupwise have zero test failures in the mid stage because they don’t stress the FPU as much the other cases and hence the emulated timing degradation of FPU is small. However, in the late-stage emulations FPU’s timing degrades rapidly and hence the test fail rates dramatically increase. There is direct correlation between the fail rate observed in Figure 5 and the test overhead reported in Figure 4 which is a result of DTC’s adaptive design. Similar observations hold for the exponential test interval scheme.

4.3. Opportunistic Testing

Testing overhead can be reduced to zero if tests are performed opportunistically when CUT is idle. There are two
types of opportunities: local opportunities exist when a particular block within a processor is idle. On the other hand, global opportunities exist when many CUTs in a processor are all idle at the same time. For instance, after a branch misprediction the entire pipeline is flushed leaving most of the CUTs in the processor idle. In general, cache misses, branch mispredictions, exceptions, imbalance distribution of different instruction types, variable execution latencies for different units in the processor can contribute to a variety of CUT idle periods.

We studied two important characteristics of testing opportunities for a range of CUTs: 1) The duration of an opportunity 2) The distance between opportunities. For generating these two characterization results we used Simplescalar [4] simulator configured as a 4-wide issue out-of-order processor with 128 in-flight instructions and 16KB L1 I/D cache. Since these opportunities depend on the application running on the processor, we decided to run 10 benchmarks from CPU2K benchmark suite, each for one billion cycles. The benchmarks used are Gzip, Crafty, Bzip, Mcf, and Parser in addition to the five used in our FPGA emulations. Our experiments are carried out using execution-driven simulations with a detailed processor model which has one floating point multiplier/divider (FPMD), four integer ALU’s, one floating point ALU (FPU), and one integer multiplier/divider unit (IMD). We measure the duration of idle times and distance between successive idle time to quantify local opportunities. To quantify the global opportunities we collected two sets of data. We measure the duration and distance between idle time periods at the pipeline stage level for instruction fetch (IF), integer execution (EINT), floating point execution (EFP), and the retirement stage (RE). We also measured the distance between L1 data cache misses (L1DCM), branch mispredictions (BMP), L2 cache misses (L2CM), and L1 instruction cache misses (L1ICM).

Figure 6(a) shows the distribution of opportunity durations in terms of percentage of total execution cycles for the above mentioned local and global opportunities. The first four bars are local opportunities and the second four bars are global opportunities. Data collected for the four ALUs has been averaged and shown as one data set.

Distribution of distance between consecutive opportunities has been shown in Figure 6(b). In this figure the first three bars are local and rest are global opportunities. Please note that the vertical axis for both Figure 6(a) and (b) are in logarithmic scale. The last cluster of columns on these figures show the cumulative value of all the opportunities with duration/distance above 1000. Data on integer multiplier/divider unit has not been measured on Figure 6(b) because all the opportunities for this unit (0.2% of execution cycles) happened within 100 cycles of each other.

Most local and global opportunities studied have a duration of 10-100 cycles, which is sufficient time for testing with multiple test vectors. More detailed breakdown of the low duration opportunities (below 100 cycles) indicates that for the eight units studied on average, cases with duration lower than 10 clock cycles account for 0.8% of the total simulated execution cycles. Opportunities of duration 10-50 cycles account for 0.4% of the execution cycles and then opportunities of 50-100 cycles duration account for 0.01%. Since these opportunities occur very close to each other it is even possible to spread a single test phase across two consecutive opportunity windows.

Based on data shown in Figure 6(b), local opportunities (the first group of three bars) are prevalent when the distance between opportunity is small. The number of local opportunities decrease as the distance increases. However, global opportunities (the remaining group of bars) are prevalent both at short distances as well as long distances. Hence, we believe that local opportunities are better candidates for opportunistic testing. While Figure 6(a) provides us with valuable information on the duration of opportunities available for testing Figure 6(b) shows if these opportunities happen often enough to be useful for our monitoring goals.

There are far more test opportunities than the number of tests needed in our monitoring methodology and hence only when the monitoring unit requires a test phase the opportunities are going to be taken advantage of. To better utilize the idle cycles, DTC would specify a future time window in which a test must be conducted, rather than an exact time
to conduct a test. When a CUT is idle within the target
time window the test is conducted. In rare occasions when
there are insufficient opportunities for testing, DTC can still
force a test by interrupting the normal CUT operation. Even
in such a scenario the performance overhead of using our
methodology is extremely low.

Monitoring multiple CUTs of small size instead of one
large CUTs might result in increased area overhead due to
presence of multiple RMUs but it would provide the possi-
bility to take advantage of more fine grain local opportuni-
ties while the other parts of the processor continue to fully
function. This is going to be beneficial even if the testing
is not performed opportunistically because even if a test is
forced on a unit and the normal operation of that unit is in-
terrupted all other parts of the processor can continue func-
tioning. Centralized test control with distributed TVRs and
RHTs can be implemented so that monitoring of multiple
CUTs can be controlled by one DTC units. Detail study of
such architectures is part of our future work.

5. Related Work

There have been prior works on modeling, prediction,
and detection of wearout faults. Modeling chip lifetime
reliability using device failure models has been studied in
[23, 19]. These models provide an architecture-level reli-
ability analysis framework which is beneficial for predic-
tion of circuit wearout at design time but they do not address
actual infield wearout of the circuit which is highly depen-
dant on dynamically changing operation conditions. The
mechanism proposed by Bower et al. [9] uses a DIVA [5]
checker to detect hard errors and then the units causing the
fault are deconfigured to prevent the fault from happening
again. Shyam et al. [20] explored a defect protection method
to detect permanent faults using Built in Self Test (BIST)
for extensively checking the circuit nodes of a VLIW pro-
cessor. BulletProof [12] focuses on comparison of the dif-
ferent defect-tolerant CMPs. Fault prediction mechanism
for detecting NBTI-induced PMOS transistor aging timing
violations have been studied in [1]. The circuit failure pre-
diction proposed in this paper is done at runtime by analyz-
ing data collected from sensors that are inserted at various
locations in the circuit. Blome et al. [6] use a wearout de-
tection unit that performs online timing analysis to predict
imminent timing failures. Double sampling latches which are
used in Razor [13] for voltage scaling can also be used for
detecting timing degradation due to aging. FIRST (Fin-
gerprints In Reliability and Self Test) [22], proposes using
the existing scan chains on the chip and performing periodic
tests under reduced frequency guardbands to detect signs of
emerging wearout.

It has been shown that some of device timing degrada-
tion due to aging, such as those caused by NBTI, can be re-
covered from by reducing the device activity for sufficient
amount of time [2]. Using our method, the optimal time
for using these recovery methods can be selected based on
the information provided by the monitoring unit. Some of
the related works mentioned above have suggested methods
which use separate test structures that model the CUT such
as buffer chains or sensing flip flops and main advantage of
our method compared to these previous approaches is that
our test vectors activate the actual devices and paths that are
used at runtime, hence each test will capture the most up-
to-date condition of the devices taking into account overall
lifetime wearout of each device.

Implementing dedicated sensing circuitry proposed in
some of the prior works for monitoring sufficient number of
circuit paths requires significant extra hardware and would
lacks the flexibility and adaptability of the mechanism pro-
posed in this work. Our method not only has the adapt-
ability advantage to reduce performance overheads due to
testing (specially during the early stages of the processors
lifetime) but also is more scalable and customizable for in-
creased coverage without incurring significant modification
to the circuit; monitoring of additional paths can simply
be done by increasing the size of the TVR. In addition to
the above mentioned key points, the capability to dynami-
ically select the circuit paths to test would result in a more
targeted testing of the actual devices in the circuit which
would significantly reduce the number tests that need to be
performed.

6. Conclusions

As processor reliability becomes a first order design con-
straint for low-end to high-end computing platforms there
is a need to provide continuous monitoring of the circuit
reliability. Just as processors currently provide power and
performance monitoring capabilities, this paper argues that
reliability monitoring will enable more efficient and just-
in-time activation of error detection and correction mecha-
nisms. This paper presents a low cost architecture for mon-
itoring a circuit using adaptive critical path tests. The pro-
posed architecture dynamically adjusts the monitoring over-
head based on current operating conditions of the circuit.
Many of the parameters controlling the monitoring over-
head, such as the interval between tests, the number of tests
performed at each test phase, the clock frequency used for
testing, and the selection test vectors can be dynamically
adapted at runtime. This runtime adaptability not only pro-
vides continuous monitoring with minimal overhead but is
also essential for robust monitoring in the presence of unre-
liable devices and rapid variations in operating conditions.

Our close to industrial strength RTL implementation ef-
fort shows that the proposed design is feasible with min-
imal area and design complexity. FPGA emulation results
show that even in the worst case wearout scenarios the adap-
tive methodology would work with negligible performance
penalty. We also showed that numerous opportunities which are suitable for multi-path testing exist when different parts of the processor are not being utilized.

References


