WearMon: Reliability Monitoring Using Adaptive Critical Path Testing

Bardia Zandian*, Waleed Dweik, Suk Hun Kang, Thomas Punihaole, Murali Annavaram
Electrical Engineering Department, University of Southern California
{bzandian,dweik,sukhunka,punihaol,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 WearMon, 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

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, ultra-thin gate-oxide layers that breakdown under high thermal stress, Negative Bias Temperature Instability (NBTI), and Electromigration [2, 7, 8]. Wearout is one manifestation of these reliability concerns and is the primary focus of this paper. In this paper we define wearout as the gradual timing degradation of devices and their eventual breakdown. Timing degradation occurs extremely slowly over time and can even be reversed in some instances, such as when degradation is caused by NBTI effects. When individual device variations are taken into consideration, timing degradation is hard to predict or accurately model. Most commercial products solve the timing degradation problem by inserting a guardband at design and fabrication time. Guardband reduces performance of a chip during its entire lifetime just to ensure correct functionally during a small fraction of time near the end of chip's lifetime.

Our inability to precisely and continuously monitor timing degradation is one of the primary reasons for overprovisioning 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. In this paper, we argue that providing reliability monitoring to improve visibility into the timing degradation process is equally important to future processors. Such monitoring capability enables just-in-time activation of error detection and recovery methods, such as those proposed in [1, 12, 14, 15, 22].

The primary contribution of this research is to propose *WearMon*, 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 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 and higher frequency operation [15, 22].
3. **Low Overhead Monitoring:** The monitoring architecture should have low area overhead and design complexity. Furthermore, monitoring framework should be

implementable with minimal modifications to existing processor structures.

WearMon is designed to satisfy all of the above criteria. With continuous, adaptive and low-overhead monitoring, conservative preset guardbands can be tightened and processor can deploy preemptive error correction measures during in-field operations *only* when the measured timing margin is small enough to affect circuit functionality. The unprecedented view of timing degradation provided by WearMon 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 improving reliability of future processors.

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

2. Reliability Monitoring Framework

WearMon 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. The current timing margin of the CUT is measured using outcomes from these tests. In the following subsections we describe the architecture of WearMon. In particular, we describe one specific implementation of the WearMon approach called a Reliability Monitoring Unit (RMU). Using RMU, we will describe how critical path test vectors are used for checking the CUT.

2.1. Architecture of the Monitoring Unit

Figure 1(a) shows the overview of 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 of RMU. Test Vector Repository (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 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. Section 2.4 will describe DTC operation.

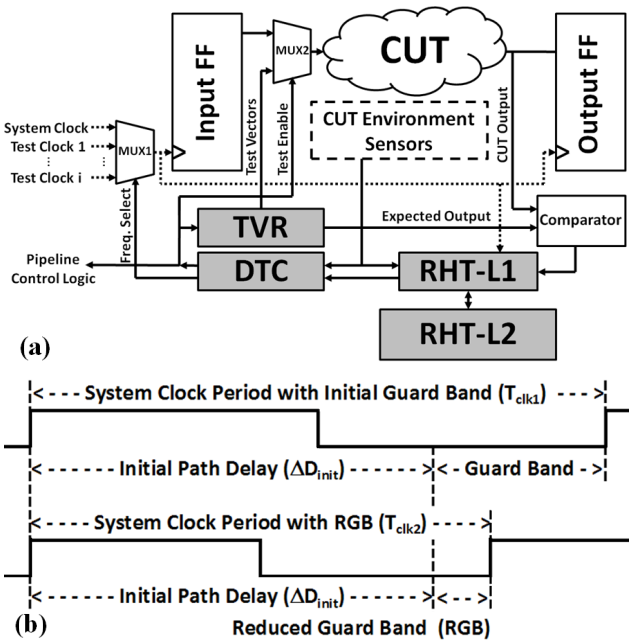

Figure 1. (a) RMU (b) reduced guardband

DTC selects a set of test vectors from TVR to inject into the CUT and the test frequency at which to test 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, testing frequency, pass/fail result, and CUT temperature. RHT is implemented as a two level 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. 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 and excessive thermal gradients over the entire lifetime of the chip. DTC 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 larger timing margin. Test and verification 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 them. 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.

2.3. Test Frequency Selection

In order to accurately monitor the remaining timing margin of a path in the CUT, it is tested at multiple test frequencies *above* its nominal operation frequency. The difference between the highest frequency at which the path passes a test and the nominal operation frequency determines the current remaining timing margin for the CUT. The test frequency range is selected between nominal operating frequency and frequency without a guardband since any test conducted at a higher frequency will fail. This range is then divided into multiple frequency steps. It is worth noting that when using WearMon, it would be possible to reduce the default guardband to a smaller value while still meeting the same reliability goals. This is shown on Figure 1(b), where $(T_{clk1} = 1/f_{clk1}) > (T_{clk2} = 1/f_{clk2}) > (\Delta D_{init})$. T_{clk1} is the original clock period when using the default guardband and T_{clk2} is the clock period with a reduced guardband (RGB). Note that in either case the initial path delay, ΔD_{init}, of the CUT is the same. The purpose of WearMon is to continuously monitor the CUT and check if CUT timing is encroaching into the reduced guardband. RGB is a performance improvement made possible by RMU, but it is not necessary to reduce the guardband for correct operation with WearMon.

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.

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 tested 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. 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 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 sudden large variation in threshold voltage which results in large path delay changes is nearly zero [16, 13]. 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.

Once the test complexity has been determined, then DTC

selects 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 such opportunities for testing a CUT. For example, on a branch misprediction the entire pipeline is flushed and instructions from the correct execution path are fetched into the pipeline. Execution, writeback, and retirement stages of the pipeline are idle waiting for new instructions since the newly fetched instructions take multiple cycles to reach the backend. When a long latency operations 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. 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 their duration 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 is increased and test complexity is reduced. 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. 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 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. We would also 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 [23]. Alternatively, an aging resilient delay generator such as the one proposed in [1] can be used with minimal area and power overhead.

While in our description we associated one DTC per CUT it is possible to have multiple CUTs share a single DTC which can make testing decisions for each CUT based on each CUT's own reliability history. This sharing of DTC logic can reduce the area overhead of implementing WearMon. In addition, a system-wide DTC can combine data from multiple local RHTs in a hierarchical manner to make an appropriate global testing decision.

3. Experimental Methodology

WearMon is not an error correcting mechanism. It provides monitoring capabilities which may in-turn be used to more effectively apply error detection or correction mechanisms. As such, monitoring by itself does not improve reliability. Hence, the aims of our experimental design are as follows: First, area overhead and design complexity of RMU are measured using close-to-industrial design implementation of the RMU and all the associated components. Second, we explored test intervals and test complexity state space. Third, performance overhead of monitoring is measured. 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 and the distance between two opportunities for testing. 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 show *worst case* overhead of monitoring compared to real implementation.

3.1. FPGA Emulation Setup

We modeled the 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. This FPU implements the 64-bit IEEE 754 floating point standard. The RMU and the FPU CUT are then mapped onto a Virtex 5 XC5VLX100T FPGA chip using Xilinx ISE Design Suite 10.1.

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 FPU temperature 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 on CompactFlash memory, and it's size is limited only by size of the CompactFlash. 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 in the FPU would be different between FPGA and ASIC 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, 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 XC5VLX100T FPGA chip. These DSP slices use dedicated combinational logic on the FPGA chip 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.

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 an approach to generate test vectors using benchmark driven input vector generation. We selected a set of five SPEC CPU2000 floating point benchmarks, Applu, Apsi, Mgrid, Swim, Wupwise, and generated a trace of floating point multiplies from these benchmarks by running them on Simplescalar [5] integrated with Wattch [10] and Hotspot [19]. 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, the expected correct output value, and the FPU temperature; we will shortly explain how the temperature is used to mimic processor aging. These traces are stored on a CompactFlash memory accessible to the FPU.

Initially the FPU is clocked at its nominal operating frequency reported by the ISE tool and all input traces produce correct outputs when the FPU runs at this frequency. The FPU is then progressively over-clocked 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

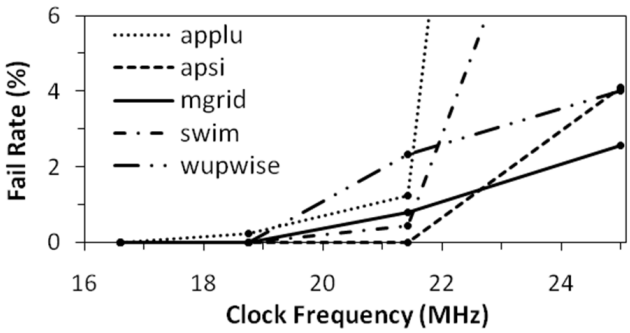

Figure 2. Timing margin degradation

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. Percentage of the FP multiply instructions that failed as the test clock frequency was increased are shown for multiple benchmarks in Figure 2. 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. We selected the top 1000 failing input operand pairs as the test vectors to fill the TVR. As CUT is overclocked beyond 25 MHz (simulating a large timing degradation), almost all input vectors fail across all benchmarks.

3.2. Three Scenarios for Monitoring

We emulated three wearout 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 condition 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, 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 phase. The early-stage mon-

itoring allows us to measure the overhead of WearMon in the common case when no errors are encountered.

Mid-stage monitoring: In mid-stage monitoring we emulated the conditions when the chip's timing margin has degraded just enough so that some of the circuit paths that encroach into the guardband 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 (closer to frequency without any guardband). 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 uses the number of failed tests seen in the last 8 test injections to determine when to activate the next test phase. We explored two different test interval selection schemes, 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 down 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 have 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 DTC adaptation policies will affect testing overheads.

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 failures 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 (e.g. supply voltage and temperature) 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.

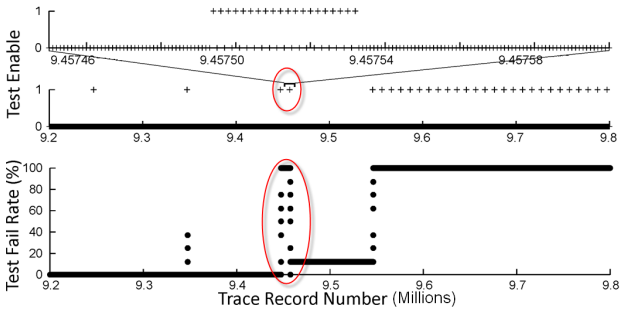

Figure 3. Dynamic adaptation of RMU

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^oC 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 and hence 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 test complexity of 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. 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. 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. The top plot zooms in on one test phase to show 20 back-to-back tests corresponding to the testing complexity 20.

4. Experimental Results

In this section results related to area and performance overhead of the WearMon framework are presented. Op-

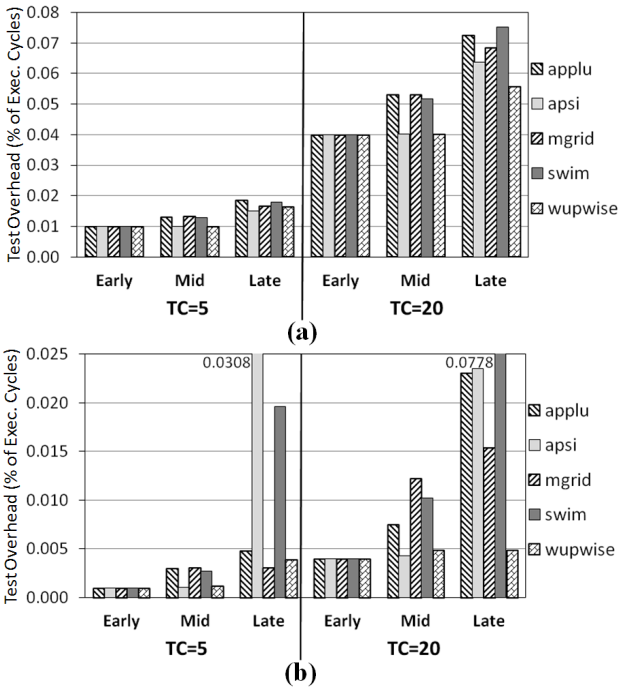

Figure 4. Overhead of (a) linear (b) exponential schemes

portunities to perform tests without interrupting the normal operation of the processor are studied in Section 4.3.

4.1. Area Overhead

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 fifteen dedicated DSP48E slices for building the double precision FP multiplier, while only one DSP48E slice is used by RMU logic. Note that this low area overhead of monitoring can be reduced even more by increasing the granularity of CUTs, i.e. increasing the CUT size from one FPU to the whole execution pipeline stage of a core. Since RMU size stays relatively constant irrespective of the CUT being monitored, one of the strengths of WearMon is this ability to select the CUT size so as to make the area overhead acceptable for any design.

4.2. Monitoring Overhead

Figure 4(a) shows the execution time overhead of testing 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)

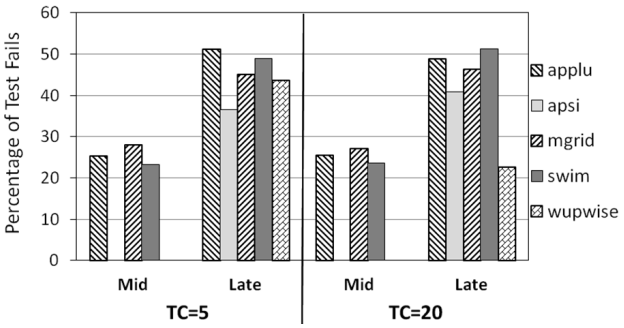

Figure 5. Test fail rates for linear scheme

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 will increase the monitoring overhead to test the CUT more frequently. The worst case overhead using late-stage monitoring scenario is only 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 conduct tests more frequently) 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 as 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.

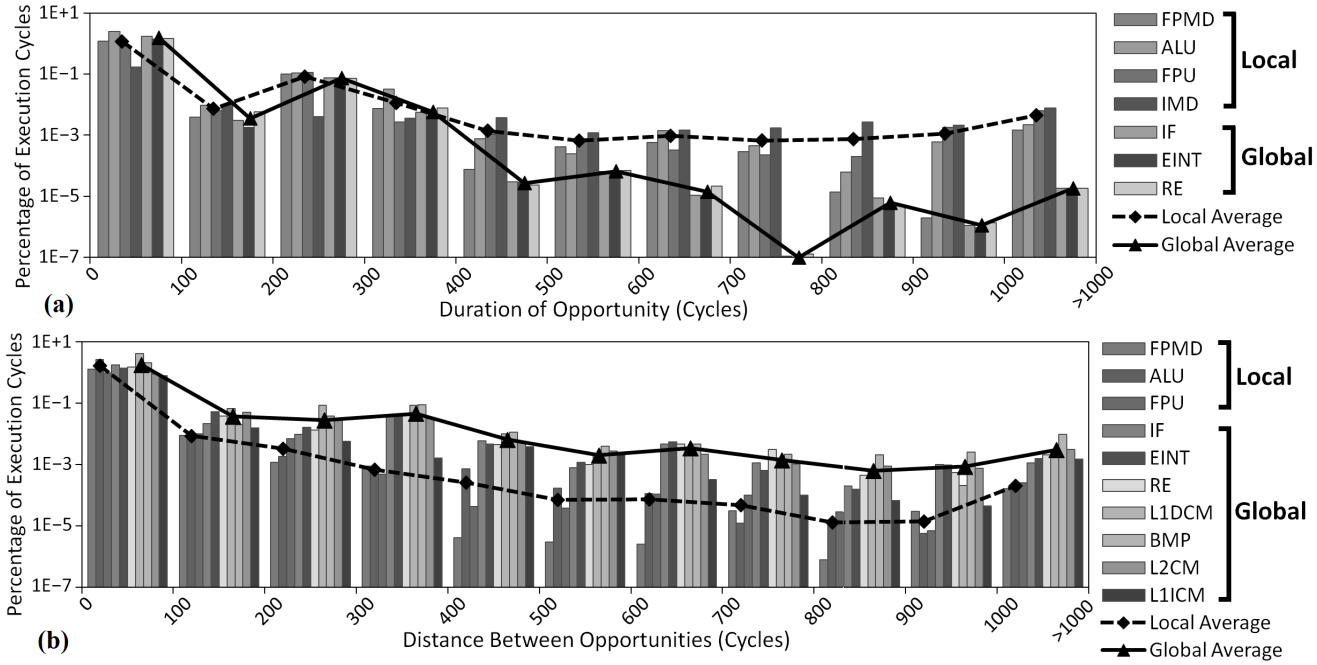

Figure 6. Distribution of (a) opportunity duration (b) distance between opportunities

4.3. Opportunistic Testing

Testing overhead can be reduced to zero if tests are performed opportunistically when a CUT is idle. Two important characteristics of testing opportunities have been studied in this section: 1) The duration of an opportunity 2) The distance between opportunities. For generating these two characterization results we used Simplescalar [5] 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 run 10 benchmarks from SPEC CPU2000 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. There are two types of opportunities studied: *local* and *global*

Local opportunities: These exist when only a particular CUT in the processor is idle. Difference in utilization of functional units caused by unequal distribution of different instruction types in the execution trace and variable execution latencies for different units in the processor will result in local opportunities. 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 periods and distance between successive idle periods for the above units to quantify local opportunities which exist for testing them.

Global opportunities: These exist when multiple CUTs in the 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. Cache misses, branch mispredictions, and exceptions will result in global opportunities. To quantify the global opportunities we measure duration and distance between idle periods at the pipeline stage level for instruction fetch (*IF*), integer execution (*EINT*), and the retirement stage (*RE*). Furthermore, distance between L1 data cache misses (*L1DCM*), branch mispredictions (*BMP*), L2 cache misses (*L2CM*), and L1 instruction cache misses (*L1ICM*) have also been measured.

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. 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). Please note that the vertical axis for both Figure 6 (a) and (b) are in logarithmic scale. The rightmost cluster of columns in these figures shows the cumulative value of all the opportunities with duration/distance above 1000 cycles. Data on integer multiplier/divider unit has not been shown 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.

The average percentage of local and global opportunities of each duration group are shown on Figure 6(a) as trend curves. Percentage of global opportunities with high duration rapidly drops for opportunity durations above 400 cycles. This behavior indicates that global opportunities must be used for shorter test phases. Local opportunities, on the other hand, have long durations making them ideal for longer test phases. Based on average trends shown in Figure 6(b), local opportunities are prevalent when the distance between opportunity is small. However, global opportunities are prevalent both at short distances as well as long distances. 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. These results show that there are more test opportunities than the number of tests needed in our monitoring methodology and hence only some of the opportunities are going to be taken advantage of when the monitoring unit schedules a test. 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. As shown earlier in Section 4.2 even in such a scenario the performance overhead of WearMon is low.

Monitoring multiple small CUTs instead of one large CUT might result in increased area overhead due to multiple RMUs. But it would provide the possibility to take advantage of more fine grain local opportunities while other parts of the processor continue to operate normally. Monitoring small CUTs may 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 interrupted, all other parts of the processor can still continue functioning.

5. Related Work

There have been prior works on modeling, detection, correction, and prediction of wearout faults. Modeling chip lifetime reliability using device failure models has been studied in [21, 17]. These models provide an architecture-level reliability analysis framework which is beneficial for prediction of circuit wearout at design time but they do not address actual infield wearout of the circuit which is highly dependent on dynamically changing operation conditions. Others have studied methods for error detection and correction. The mechanism proposed by Bower *et al.* [9] uses a DIVA [4] checker to detect hard errors and then the units causing the fault are deconfigured to prevent the fault from happening again. Shyam *et al.* [18] explored a defect protection method to detect permanent faults using Built in Self Test (BIST) for extensively checking the circuit nodes of a VLIW processor. BulletProof [11] focuses on comparison of the different defect-tolerant CMPs. Double sampling latches which are used in Razor [12] for voltage scaling can also be used for detecting timing degradation due to aging. A fault prediction mechanism for detecting NBTI-induced PMOS transistor timing violations have been studied in [1]. This failure prediction is done at runtime by analyzing data collected from sensors that are inserted at various locations in the circuit. It has also been shown that some of device timing degradation, such as those caused by NBTI, can be recovered from by reducing the device activity for sufficient amount of time [2]. Blome *et al.* [6] use a wearout detection unit that performs online timing analysis to predict imminent timing failures. FIRST (Fingerprints In Reliability and Self Test) [20], proposes using the existing scan chains on the chip and performing periodic tests under reduced frequency guardbands to detect signs of emerging wearout.

With WearMon, the optimal time for using many of the error detection and correction methods studied in the works mentioned above can be selected based on the information provided by the monitoring unit. [1, 6] 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. Monitoring sufficient number of circuit paths using many of mentioned approaches [9, 18, 12, 20] requires significant extra hardware and also lacks the flexibility and adaptability of the mechanism proposed in this work. WearMon not only has the adaptability 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 increased coverage without incurring significant modification to the circuit; monitoring of additional paths can simply be done by increasing the size of the TVR. Furthermore, the capability to dynamically select the circuit paths to test, results in a more targeted testing which significantly reduces the number tests that need to be performed.

6. Conclusions

As processor reliability becomes a first order design constraint for low-end to high-end computing platforms there is a need to provide continuous monitoring of the circuit reliability. Reliability monitoring will enable more efficient and just-in-time activation of error detection and correction mechanisms. In this paper, we present WearMon, a low cost architecture for monitoring a circuit using adaptive critical

path tests. WearMon dynamically adjusts the monitoring overhead based on current operating conditions of the circuit. Runtime adaptability not only results in minimal overhead but is also essential for robust monitoring in the presence of variations in operating conditions.

Our close to industrial strength RTL implementation effort shows that the proposed design is feasible with minimal area and design complexity. FPGA emulation results show that even in the worst case wearout scenarios the adaptive 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.

7. Acknowledgments

This work was supported by NSF grants CCF-0834798, CCF-0834799, and an IBM Faculty Fellowship.

References

[1] M. Agarwal, B. Paul, M. Zhang, and S. Mitra. Circuit failure prediction and its application to transistor aging. In *25th IEEE VLSI Test Symposium*, pages 277–286, May 2007.

[2] M. Alam and S. Mahapatra. A comprehensive model of pmos nbti degradation. *Microelectronics Reliability*, 45(1):71–81, 2005.

[3] M. Annavaram, E. Grochowski, and P. Reed. Implications of device timing variability on full chip timing. *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, pages 37–45, Feb 2007.

[4] T. Austin. Diva: A reliable substrate for deep submicron microarchitecture design. *Proceedings of the 32nd Annual International Symposium on Microarchitecture*, pages 196–207, 1999.

[5] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, Feb 2002.

[6] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 109–122, 2007.

[7] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Proceedings of the 38th Annual International Symposium on Microarchitecture*, pages 10–16, Dec 2005.

[8] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. *Proceedings of the 40th Annual Conference on Design Automation*, pages 338–342, 2003.

[9] F. Bower, D. Sorin, and S. Ozev. A mechanism for online diagnosis of hard faults in microprocessors. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pages 197–208, 2005.

[10] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Computer Architecture News*, 28(2):83–94, 2000.

[11] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. Bulletproof: a defect-tolerant cmp switch architecture. In *The Twelfth International Symposium on High-Performance Computer Architecture*, pages 5–16, Feb. 2006.

[12] D. Ernst, N. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. *Proceedings of the 36th Annual International Symposium on Microarchitecture*, pages 7–18, Dec. 2003.

[13] T. Fischer, E. Amirante, P. Huber, K. Hofmann, M. Ostermayr, and D. Schmitt-Landsiedel. A 65nm test structure for sram device variability and nbti statistics. *Solid-State Electronics*, 53(7):773–778, 2009.

[14] M. Li, P. Ramachandran, U. Karpuzcu, S. Hari, and S. Adve. Accurate microarchitecture-level fault modeling for studying hardware faults. In *Proceedings of the 15th International Symposium on High-Performance Computer Architecture*, Feb 2009.

[15] X. Liang, G. Wei, and D. Brooks. Revival: A variation-tolerant architecture using voltage interpolation and variable latency. In *35th International Symposium on Computer Architecture*, pages 191–202, June 2008.

[16] S. Rauch. Review and reexamination of reliability effects related to nbti-induced statistical variations. *Device and Materials Reliability, IEEE Transactions on*, 7(4):524–530, Dec. 2007.

[17] J. Shin, V. Zyuban, Z. Hu, J. Rivers, and P. Bose. A framework for architecture-level lifetime reliability modeling. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 534–543, June 2007.

[18] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, and T. Austin. Ultra low-cost defect protection for microprocessor pipelines. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 73–82, Oct 2006.

[19] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. *Proceedings of the 30th International Symposium on Computer Architecture*, pages 2–14, June 2003.

[20] J. C. Smolens, B. T. Gold, J. C. Hoe, B. Falsafi, and K. Mai. Detecting emerging wearout faults. In *In Proceedings of the IEEE Workshop on Silicon Errors in Logic - System Effects*, 2007.

[21] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. The case for lifetime reliability-aware microprocessors. *Proceedings of the 31st Annual International Symposium on Computer Architecture*, pages 276–287, June 2004.

[22] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Proceedings of the 41st International Symposium on Microarchitecture*, pages 129–140, Nov 2008.

[23] Q. Wu, P. Juang, M. Martonosi, and D. Clark. Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors. In *11th International Symposium on High-Performance Computer Architecture*, pages 178–189, Feb. 2005.