Cross-layer Resilience Using Wearout Aware Design Flow

Bardia Zandian, Murali Annavaram
Electrical Engineering Department, University of Southern California
Los Angeles, CA
{bzandian,annavara}@usc.edu

***Abstract*—As process technology shrinks devices, circuits experience accelerated wearout. Monitoring wearout will be critical for improving the efficiency of error detection and correction. The most effective wearout monitoring approach relies on continuously checking only the most critical circuit paths to detect timing degradation. However, circuits optimized for power and area efficiency have a steep critical path wall in some designs. Furthermore, wearout depends on dynamic conditions, such as processor's operating environment, and application-specific path utilization profile. The dynamic nature of wearout coupled with steep critical path walls may result in excessive number of paths that need to be monitored. In this paper we propose a novel cross-layer circuit design flow that uses path timing information and runtime path utilization data to significantly enhance monitoring efficiency. The proposed methodology uses application-specific path utilization profile to select only a few paths to be monitored for wearout. We propose and evaluate four novel algorithms for selecting paths to be monitored. These four approaches allow designers to select the best group of paths under varying power, area and monitoring budget constraints.**

Keywords-Wearout; Timing margin; Cross-layer design

I. INTRODUCTION

As devices scale to nanometer dimensions, processor's lifetime reliability is reduced due to increased stress factors such as higher current density, electric field, and operation temperature [1]. Reliability degradation manifests in the form of many electro-physical phenomena such as Electromigration, Time Dependant Dielectric Breakdown (TDDB), Hot Carrier Injection (HCI), and Negative Bias Temperature Instability (NBTI) [2]. The net result of these phenomena is gradual timing degradation and eventual breakdown of circuits [3-5], which is referred to as wearout or aging. Designers estimate the expected wearout during the lifetime of a processor and use guardbands to proactively reduce the clock frequency (and increase supply voltage) to account for worst-case wearout. But wearout prediction is becoming increasingly challenging as process variations lead to random device characteristics both within and across chips. Dynamically changing environmental conditions and workload-dependent circuit path utilization further exacerbate the problem of wearout estimation. While these uncertainties existed even before, the severity of their impact is increased as devices are scaled [6, 7]. There are different models which explain the rate of wearout and its dependence on different static and dynamic parameters. All these models and physical device level experiments show a gradual degradation which happens over long time periods [3-5]. Given that wearout occurs at a glacial time scale compared to processor cycle time, it is best to monitor wearout first before deploying expensive error correction mechanisms. With continuous monitoring the occurrence of a wearout related error is predicted before the error occurrence and preventive adjustments are made to the circuit's operation point (e.g. changing clock frequency and supply voltage; or deploying modular redundancy) in order to avoid errors [8, 9]. High accuracy error prediction allows for designing a truly reliability-aware circuit which can adapt to the in-field reliability state of the hardware.

Given the promise of the prediction approach several prediction techniques have been proposed. Some prediction techniques use "canary" circuits which are designed to fail before the actual circuit [10]. Other techniques use sensors inserted into the circuit at design time which are capable of detecting wearout by sensing increased circuit delay [11, 12] or changes in other parameters, such as threshold voltage (V_{th}) [13]. Canary circuits do not test the actual signal paths in the circuit; rather they only act as proxies for the primary circuit wearout. More recently researchers proposed wearout prediction based on monitoring the signal paths in the primary circuit itself. In WearMon [14], stored test vectors that are specifically selected to sensitize the critical paths of the circuit are used for runtime tests that capture the timing margin (also called timing slack) of these paths. Another *in situ* circuit checking method [15] uses Built-in Self Test (BIST) mechanism to perform runtime circuit tests. The main advantage of monitoring actual signal paths using stored test vectors compared to static sensory circuit insertion is the ability to capture the effects of *actual* circuit lifetime utilization at a lower cost and with higher flexibility for online adaptation. For instance, test coverage can be optimized during in-field operation with little or no overhead by simply updating the stored test vectors.

Wearout monitoring mechanisms generally make the following two basic assumptions: (1) In any given circuit there are only a few circuit paths that have critical timing margins. Hence, to accurately predict imminent timing failures only a few circuit paths with the least timing margin need to be monitored. (2) Circuit paths with least timing margin have a higher probability of being among the first to violate timing. Hence, monitoring prioritizes paths purely based on the timing margin measured at design time. The first assumption indicates that selection of only a few paths for monitoring would be sufficient for robust monitoring. This assumption may hold well in some designs that use

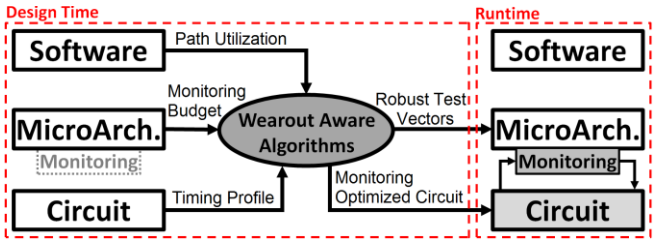

Figure 1. Design time and runtime cross-layer interaction.

automatic design tools to synthesize, place and route the design. In the absence of knowledgeable designer's input these tools typically do not create steep critical path walls [16, 17], where a large number of paths have small timing margin. However, custom design optimizations for maximizing power and area efficiency, particularly employed in high performance processors, may result in the creation of a steep critical path wall in several circuit blocks. In the presence of a steep critical path wall the number of paths that need to be monitored can be very large, thereby increasing the monitoring overhead. The second assumption made by *in situ* approaches results in the selection of paths purely based on design stage timing margin. However, it has been shown that wearout depends on dynamic runtime utilization of the processor and many of the causes of wearout get exacerbated with increased circuit utilization [6, 7]. Path selection purely based on timing margin neglects this important dependence. Thus the robustness of the monitoring approach that relies purely on timing margin can be compromised due to the dynamic nature of path utilization. Hence, we conclude that in order for monitoring approaches to be more broadly applied (beyond low-cost computing segment) there is a need for a symbiotic interaction between circuit design tools, monitoring hardware and the high-level application software. Only through such an interaction it is possible to identify circuit paths which are the slowest at design time and also have higher lifetime utilization resulting in most wearout induced timing degradation.

In this paper we propose a novel cross-layer circuit design flow methodology that combines static path timing information with runtime path utilization data to significantly enhance monitoring efficiency and robustness. Fig. 1 shows the layered framework consisting of two phases:

(1) Cross-layer design flow (CLDF) phase: This phase (marked as "Design Time" in the figure) uses representative application inputs to derive circuit path utilization profile. The microarchitecture specification provides monitoring budget, such as the amount of chip area or the power consumption allocated for monitoring. CLDF also derives timing profile from static timing analysis of circuit's design. The wearout aware algorithm then combines information from software, microarchitecture and circuit layers to drive circuit design optimizations with the explicit goal of making a circuit amenable for robust and efficient monitoring. The algorithm selects a refined group of paths along with a robust set of input vectors for wearout monitoring.

(2) Wearout monitoring phase: A runtime wearout monitoring phase, similar to that proposed in WearMon [14], continuously monitors the paths selected from the CLDF phase. The information about the circuit paths which need to be monitored, obtained from the CLDF phase, is used in the runtime phase for wearout detection.

The focus of this research work is to develop the CLDF framework. As such, we assume that a wearout monitoring mechanism exists in the underlying microarchitecture. CLDF significantly enhances the applicability of existing runtime monitoring approaches. For example, where wearout sensors or canary circuits are used for monitoring, CLDF will identify circuit paths that are most susceptible to failure thereby allowing the designer to select the most appropriate location of the wearout sensors or canary circuitry. When *in situ* monitoring approaches are used [14, 15, 18, 19] only the most susceptible circuit paths reported by the CLDF framework are monitored. It should be noted that although the CLDF framework can be used with all the above mentioned reliability monitoring approaches, throughout this paper we assume that the underlying microarchitecture uses an *in situ* monitoring approach similar to WearMon [14] to illustrate how our design phase optimizations can enhance runtime monitoring efficiency.

The main contributions of this work are:

1. We design and implement a novel cross-layer circuit design flow methodology that combines static path timing information with runtime path utilization data to significantly enhance monitoring efficiency. This framework uses path utilization profile, path delay characteristics, and number of devices in critical paths to optimize the circuit using selective path constraint adjustments (i.e. increasing the timing margin of selected group of paths). This optimization results in a new implementation of the circuit which is more amenable for low overhead monitoring of wearout-induced timing degradation.

2. We propose four algorithms for selecting the best group of paths to be observed as early indicators of wearout induced timing failures. Each of these algorithms allows the designer to tradeoff area and power overhead of monitoring with robustness and efficiency of monitoring.

3. We develop a hybrid hierarchical emulation/simulation infrastructure to study the effects of application level events on gate-level utilization profile. This setup provides a fast and accurate framework to study system utilization across multiple layers of the system stack using a combination of FPGA emulation and gate-level simulation.

In an era when computers are built from increasing number of components with decreasing reliability, multi-layer resiliency is becoming a requirement for all computer systems. In this work we design and implement a low cost and scalable solution in which different layers of the computer system stack can communicate and adapt both at design phase and during the runtime of the system. Our proposed cross-layer design flow approach is discussed in Section II. Section III shows our hybrid cross-layer evaluation infrastructure, followed by the evaluation results in Section IV. Section V describes the most relevant prior work, followed by conclusions in Section VI.

II. CROSS-LAYER DESIGN FLOW

In this section we describe the cross-layer circuit design flow (CLDF) methodology. At the core of CLDF is a novel approach that modifies the distribution of path timing margins, so as to create a group of critical paths that are more likely to fail before any other paths fail. The paths that are likely to fail first are referred to as *wearout-critical* paths. Wearout-critical paths would be ideal candidates for being monitored as early indicators of wearout. CLDF receives a monitoring budget, in terms of the area and power overhead allowed for monitoring, as input from the designer. CLDF uses three characteristics of the circuit, namely path timing, path utilization profile, and number of devices on the path, to select a limited number of wearout-critical paths to satisfy the monitoring budget constraints specified by the designer.

Paths which are selected to be monitored at runtime are going to be checked regularly using approaches like [14, 15, 18-20]. Detailed description of runtime monitoring techniques is outside of the scope of this paper. However, to put the work presented in this paper in context, we will highlight a key testing method used by these runtime monitoring approaches. Many of the runtime monitoring frameworks test the circuit (or canary circuit) at a test frequency, f_{test}, which is higher than the normal operation frequency, $f_{0+GB}=1/T_{0+GB}$. T_0 is the delay of slowest paths in the circuit at design time and hence ideally the circuit can operate at that clock period at fabrication time. As mentioned earlier, designers add a guardband (increase the clock period) to deal with wearout. T_{0+GB} is the clock period of the system with added guardband, which is the usual operational clock period of the circuit. If multiple tests, each at a clock period that falls within the T_0 and T_{0+GB} range ($1/T_{0+GB}<f_{test}<1/T_0$), were preformed the test results would provide information about the exact amount of timing degradation in paths tested. Throughout the paper we assume that the above described approach is used for wearout monitoring.

We first provide an overview of the algorithmic steps for the proposed CLDF approach. Detailed description of the key steps will follow immediately.

Step 1. The circuit is first synthesized using conventional design flow. Performance, power, and area constraints are provided as inputs to the synthesis tool. The synthesis tool generates the implementation of the design and an initial static timing report that shows the timing margin of each circuit path. The first step in CLDF takes this synthesized design as input and sorts all the circuit paths in the timing report based on their timing margin. It then selects some number of paths, say *nLong*, with least timing margin. These *nLong* paths are further analyzed in the rest of the steps.

Step 2. The second step is where the cross-layer aspect of design flow comes into effect. In this step, CLDF selects a representative set of workloads and runs them on the synthesized design. Utilization profile of the *nLong* paths selected in step 1 is collected. The profile provides information regarding how frequently each path has been exercised during the execution of the selected workloads.

Step 3. One of the four approaches discussed in subsection II.C is used to select two groups of paths from the *nLong* paths: a) Path to be optimized further. b) Paths to be monitored at runtime.

Step 4. Paths selected in group 3(a) are optimized to be faster which results in more timing margin for these paths. By optimizing paths in 3(a) the approach creates a distinct separation of timing criticality between the two path group. This separation causes paths in the group 3(b) to be wearout-critical paths that allow for robust monitoring. It should be noted that groups 3(a) and (b) are not mutually exclusive and depending on the approach selected by the CLDF framework there might be paths which are in both groups and are optimized and also selected for being monitored.

Step 5. This step collects necessary data to enable robust runtime monitoring of paths in group 3(b). This step is dependent on the monitoring framework used. For example if a runtime wearout monitoring such as [14] is used, the input vectors that would sensitize the paths in group 3(b) are created in this step. These inputs are then stored in a test vector repository to enable runtime monitoring. If approaches like [10, 20] are used for runtime monitoring, then location of the paths in group 3(b) and their structure should be stored so that canary circuits can be designed for them or wearout sensors can be inserted at appropriate locations. As stated earlier, in this paper we assume a monitoring approach based on test vector injection for path testing [14] is used.

A. Step 1: Selection of the Analysis Group

The first step in CLDF is to use a traditional synthesis tool to synthesize the design and perform static timing analysis. The hardware description language (HDL) code for the design in addition to performance, area, and power constraints are provided as inputs to the synthesis tool. The output of this initial synthesis will be the gate-level implementation and a timing report that indicates the amount of timing margin for each circuit path. CLDF then generates a sorted path list based on timing margin and selects a group of *nLong* longest paths (paths with the least timing margin). These paths are considered for optimization and/or runtime monitoring as we will describe later. The selection of *nLong* paths is done as follows.

CLDF selects *nLong* paths based on an initial cut-off criteria (*InitCutoff*) given as input to the algorithm. CLDF selects only those paths whose delay is larger than *InitCutoff* percentage of the maximum path delay. For example if the delay of the longest path in the circuit is 10ns and if *InitCutoff* is selected as 75%, then CLDF picks all paths with delay of 7.5ns or higher; this approach ensures that all paths within 75% of the worst-case delay are selected for analysis. The cutoff parameter is selected by the designer based on the worst-case wearout expected in a design within the typical lifetime of the processor. It has been shown in prior studies that all wearout causing phenomena, such as NBTI, and Electromigration, reach a maximum wearout level beyond which they cause device failure [6, 7]. In fact, this knowledge is what is used by conservative circuit design approaches for selecting a guardband to prevent premature failures; when a designer selects a 10% guardband the assumption is that no path with more than 10% timing

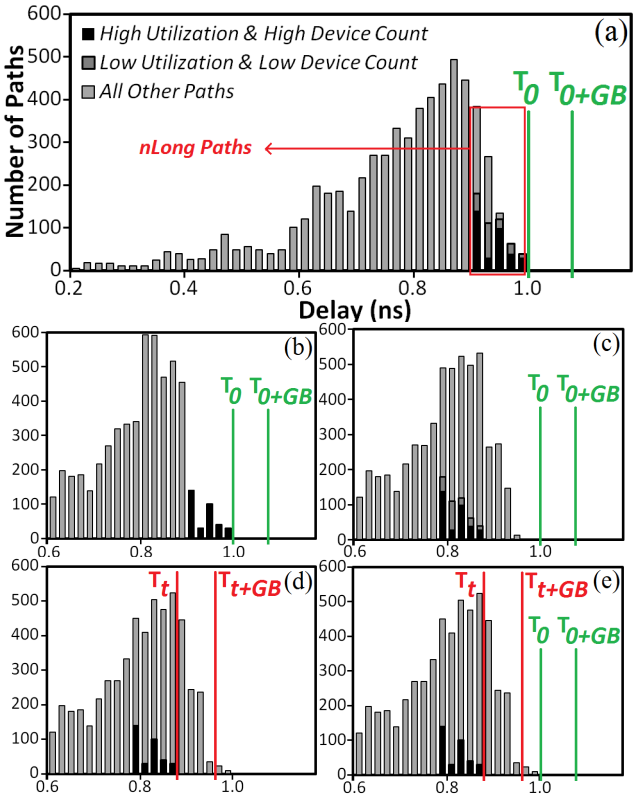

Figure 2. Path delay distribution (a) before optimization and after (b) Approach 1 (c) Approach 2 (d) Approach 3 (e) Approach 4.

margin will fail before the expected lifetime of the processor. Hence, *InitCutoff* is simply the conservative guardband that has already been estimated at design time.

It is worth noting that for circuits with steep critical path timing walls using *InitCutoff* may result in selection of a large group of paths for further analysis, thereby making *nLong* a very large number. Large *nLong* values do not create any impediment in the next steps of CLDF algorithm. Similarly, for circuits with shallow critical path timing walls *nLong* may be small. If *nLong* is too small (smaller than the number of paths which can be monitored efficiently), then there is no need to even conduct further analysis since the circuit does not have many critical paths and it may be possible to monitor all critical paths without further analysis or need for CLDF. The main goal of this work, however, is to make circuits with steep critical path timing walls (large *nLong* values) still amenable for monitoring.

B. Step 2: Utilization Based Path Prioritization

Step 2 generates utilization profile of *nLong* paths. The utilization data is collected while executing a representative set of applications that are expected to run on the design. During execution of representative applications the number of times each of the *nLong* paths is utilized is saved. Then *nLong* paths are sorted based on the cumulative number of times each path was utilized during profile runs; we call this sorted list the utilization profile.

CLDF uses *HighUtilCutoff* parameter given as input to CLDF to identify paths that have utilization greater than

TABLE I. COMPARISON OF APPROACHES

Path Utilization	Path Device Count	App. 1		App. 2		App. 3		App. 4	
		Opt.	*Mon.*	*Opt.*	*Mon.*	*Opt.*	*Mon.*	*Opt.*	*Mon.*
High	High (1)	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Low (2)	Yes	No	No	Yes	No	No	No	Yes
Low	High (3)	Yes	No	No	Yes	No	No	No	No
	Low (4)	Yes	No	Yes	No	No	No	No	No

HighUtilCutoff percent of the maximum utilization reported for the *nLong* paths. These paths are demarcated as high utilization paths. CLDF also uses a *LowUtilCutoff* parameter and any path with utilization lower than this cutoff is demarcated as a low utilization path. The rationale behind using two cutoffs is to create two distinct groups of paths with very different utilization levels. As explained shortly, this clear separation between high and low utilization is used to create robust and efficient monitoring mechanisms.

Timing degradation of a circuit path is a sum of the degradation of all the devices on that path. Hence, if all other parameters are the same, more devices on a path result in more susceptibility to wearout induced timing degradation. As such CLDF uses device counts on a path to further differentiate between paths. CLDF uses a single input parameter called *DevCutoff* to demarcate paths with high or low device counts. The criteria for specifying these parameters are described later.

After gathering the utilization profile, CLDF divides *nLong* paths into three categories based on *HighUtilCutoff*, *LowUtilCutoff*, and *DevCutoff*. Timing margin of one category of paths will be increased; we refer to these as the optimized group. Another category contains those paths that are monitored for wearout at runtime, which is referred to as the monitored group. The third category contains paths that are neither optimized nor monitored. We have explored four path categorization algorithms in this research. These algorithms provide different tradeoffs between performance, power, area, and reliability.

Illustrative example: While describing the four algorithms, we will use an illustrative example to show how path categorization is done. For this purpose in Fig. 2(a) we show the initial delay distribution of a sample circuit taken from OpenSPARC T1 processor [21]. This sample circuit is the instruction decode block of the instruction fetch unit (sparc_ifu_dec). Section IV presents more quantitative details for this circuit but they are not necessary here for understanding the algorithms. The timing constraint used for synthesis is 0.95ns (T_0 or zero timing margin path delay). We assume there is a 0.09ns timing guardband added by the designer to deal with wearout. Hence, the resulting system clock period is 1.04ns (T_{0+GB}). In this discussion we assume that we use 90% as the *InitCutoff* value. Hence, we select *nLong* paths that are within 90% of the longest timing paths. All paths in the right most five columns of Fig. 2(a) form the *nLong* paths. There are three types of paths highlighted with shades of black in Fig. 2(a): *high utilization & high device count*, *low utilization & low device count*, and all other paths. The group marked *high utilization & high device count* are the paths that have utilization that exceeds the *HighUtilCutoff* and device count that exceeds the *DevCutoff*

parameters. Similarly, *low utilization & low device count* are the paths that have utilization that is below the *LowUtilCutoff* and device count that is below the *DevCutoff* parameter.

Intuitively, the separation of paths into three types based on utilization and device count provides an opportunity to shift steep critical timing walls by not treating all paths with the same timing margin as equally important. Instead we create path heterogeneity with device count and utilization information derived from application level information. By exploiting this crucial runtime information through design time utilization analysis we can avoid critical path timing walls, as we will show in the next step.

C. Step 3: Approaches for Selecting Monitored Paths

The output from this step is the identification of paths that are used for monitoring. We assume that a designer has a fixed budget to monitor only *nMonitor* paths (based on the area, power, and performance budget allocated for monitoring). Hence, the goal is to select a total of *nMonitor* paths. In this section we describe four approaches that we designed for path selection.

1) Approach 1: Monitor Least Reliable

The goal of this approach is to create a distinct group of paths which, with high probability, are the paths that are going to have wearout induced timing failure before the rest of the paths. These paths will be monitored and used as predictors of imminent timing violations. Approach 1 achieves this goal by reshaping the path delay distribution of the circuit as follows. A group of paths that are most susceptible to wearout are selected for monitoring. Concurrently, all the paths that are not monitored are removed from the critical path wall by increasing the timing margin of these paths. Since paths that are not monitored have higher timing margin the probability of path not monitored failing before the monitored group is reduced. Fig. 2(a) shows the distribution of a sample circuit before using Approach 1 and Fig. 2(b) shows the redistribution of the paths after applying Approach 1. The paths with the most delay in the redistributed plot, highlighted in black on Fig. 2(b), are the group left for monitoring while all other paths are moved away from the critical path wall. Detailed description of Approach 1 is given below.

Paths optimized: This approach starts with the utilization profile generated in Step 2 of the algorithm, which sorts *nLong* paths based on path utilization. The *HighUtilCutoff* parameter is used to select paths with high utilization, i.e. paths with utilization greater than the cutoff parameter. We then sort the high utilization paths based on the number of devices on each path. We further divide this newly sorted list by using *DevCutoff* parameter and identify the high device count and low device count paths. At the end of this process we end up with three sets of paths: *high utilization & high device count, high utilization & low device count,* and the remaining paths without any concern for their device count. We then separate *high utilization & high device count* paths from the *nLong* paths. The remaining paths (*nLong* paths excluding *high utilization & high device count* paths) are optimized to have a larger timing margin. The increase in the margin is equal to the initial circuit guardband. Path optimization is done by resynthesizing the design using stricter timing constraint for the paths selected. The delay of the optimized paths can be reduced, for instance, by increasing the size of devices used on these paths. Since the optimized paths have more timing margin they are also significantly less likely to cause timing violations.

Paths monitored: The *high utilization & high device count* paths which are not optimized (black bars in Fig. 2(b)) will form the set of paths which are going to be continuously monitored for wearout. These paths have a higher probability of suffering the most wearout. These paths are utilized more frequently and utilization has a first order effect on many of the wearout causing phenomena. These paths also have more devices on them and are more susceptible to timing degradation caused by wearout of their devices. Runtime monitoring would check the path delay degradation of these paths between T_0 and T_{0+GB} and will alert the system if any monitored path delay gets critically close to T_{0+GB}.

Discussion of Approach 1: The goal is to select a total of *nMonitor* paths where all paths have the characteristic of *high utilization & high device count*. Our main motivation for using *HighUtilCutoff* selection criteria is to pick a subset of *nLong* paths with a distinctly higher utilization compared to the rest of the *nLong* paths in that circuit. To satisfy this goal *HighUtilCutoff* can be selected in the range of 75% to 85% of the maximum utilization in the *nLong* path group. If a smaller percentage is selected, the relative utilization difference between the paths selected and the ones not selected would become smaller and hence the goal of leveraging utilization differences between paths will not be satisfied.

A few special cases are worth mentioning. First, if the number of paths in the *high utilization & high device count* category are more than the monitoring budget we simply select the most utilized *nMonitor* paths from this category and optimize the remaining paths even in this category. On the other hand, in some circuits the number of paths categorized as *high utilization & high device count*, after applying *HighUtilCutoff* and *DevCutoff*, may be fewer than *nMonitor*. In this case we fill the remaining paths for monitoring from *high utilization & low device count* category as well thereby removing these paths from further optimization.

It should be noted that the goal of this work is to deal with circuits which have many more paths than the *nMonitor*. If the paths selected to be in the *nLong* path group are fewer than *nMonitor* paths, then it is not necessary to use the CLDF approach and all paths in the *nLong* group can simply be monitored.

The value used for *nMonitor* has a direct impact on the area overhead of Approach 1. If *nMonitor* is small then the number of paths which are not monitored will be large and hence the area overhead of the optimization is going to increase. Recall that all the paths in *nLong* group that are not monitored will be optimized, which usually requires increasing device sizes. Furthermore, paths optimized with larger device sizes also lead to higher dynamic power

consumption whenever these paths are exercised. These overheads can be reduced if the monitoring overhead is increased, by selecting larger *nMonitor*. Of course there is the tradeoff that more paths being monitored would mean more overhead for the monitoring setup.

One advantage of Approach 1 is that it does not perturb paths with *high utilization & high device count* which typically are the most power hungry paths in a circuit. On the other hand, since it does not perturb the *high utilization & high device count* paths the optimization effort and the resulting area overhead would not improve circuit's susceptibility to timing failures since the *high utilization & high device count* paths still have a small timing margin. In other words, this approach only has the benefit of making any circuit with any path distribution suitable for monitoring and will increase robustness and effectiveness of monitoring but it does not change the fundamental wearout behavior of the circuit.

It should be noted that for designs which have stringent power and area constraints but can tolerate some performance degradation (e.g. mobile device chips that are more constrained by power and area than maximum frequency) Approach 1 can be implemented in an alternative way. Paths which are selected for monitoring can be deoptimized while keeping all other paths the same. In other words, we increase the clock period and reduce the speed of *high utilization & high device count* paths to match the lower timing demands. All the other paths remain untouched and hence they will all gain additional timing margin while the *high utilization & high device count* paths will be the wearout critical paths used in monitoring.

2) Approach 2: Two Monitoring Groups

Goals of this approach are twofold: (1) monitor the paths which are most susceptible to wearout but also make sure that the optimization effort of CLDF results in a longer lifetime of the circuit in presence of wearout. (2) Increase robustness of monitoring even if the path utilization during in-field operation varies from the utilization profile collected from representative applications. In order to achieve the first goal of improving reliability, first the paths most susceptible to wearout are monitored as in Approach 1. In addition these monitored paths are also optimized to improve the lifetime of the circuit. To achieve the second goal we also monitor a second subset of paths that are not necessarily the most wearout susceptible during the profile run. The redistribution is shown on Fig. 2(c).

Paths monitored: The monitoring budget is split equally into two groups. First group of paths to be monitored, called Group 1, is the same as those selected in Approach 1, namely, *high utilization & high device count* paths selected using the selection polices described in Approach 1, except that we only select *nMonitor/2* paths. The paths selected for monitoring are removed from the *nLong* paths. The remaining paths are then sorted in descending order based only on utilization without any constraint on device count. The second half of monitored paths, called Group 2, is selected from the top of this newly sorted list. Group 2 increases the robustness of monitoring since it selects half the paths that are categorized as not as susceptible during profile run.

Paths optimized: All paths except those in Group 2 are optimized. By optimizing paths in Group 1 the most susceptible paths will have more timing margin and the overall circuit lifetime is increased. By not optimizing paths in Group 2, which are not as susceptible to wearout, we create a distinct group of paths with very different timing margin profile that are simultaneously monitored thereby further improving monitoring robustness. In particular, by not optimizing Group 2 while at the same time monitoring Group 2, the reliance on profile data accuracy is reduced.

Discussion of Approach 2: In Approach 2 every path in the *nLong* group is either monitored or optimized or both. In particular, there are no paths that are neither optimized nor monitored. This approach is particularly suitable for designs with larger monitoring budget (*nMonitor*) and circuits with clustering of a large number of paths in the *low utilization & low device count* paths.

3) Approach 3: Virtual Critical Paths

In the first two approaches there is no limit on the number paths optimized which may lead to unacceptable area and/or power overheads for some circuits. Approach 3 focuses on limiting area and power overheads from optimization while still retaining monitoring efficiency of prior approaches. The approach relies on a small change to monitoring process itself to achieve its goal. Monitoring is done using a higher testing frequency than the previous two approaches.

Paths monitored and optimized: We use Approach 1 to select *nMonitor* paths for monitoring. We then optimize only the paths selected for monitoring and all other paths are untouched. Note that in Approach 1 we optimized all other paths that are not monitored, where as in Approach 3 we optimize *exactly* the same group of paths that are also monitored. Thus the area and/or power overhead associated with optimizing remains fixed (based on the *nMonitor* paths) independent of the number of *nLong* paths.

Modifications to monitoring hardware: In this approach the monitoring hardware itself has to be modified. Testing of the critical paths selected for monitoring needs to be done using a different testing clock frequency, f_{test}, than the one described early in Section II, which is $1/T_{0+GB}<f_{test}<1/T_0$. When Approach 3 is employed, a test clock period that is shorter than the actual clock period of the system is going to be used for monitoring; we will refer to this as a virtual test clock. Since paths monitored are also the paths optimized, monitored paths no longer have the smallest amount of timing margin. For monitoring purposes, however, these paths are treated as if they are still the most critical paths (virtually critical). Note that the monitored paths are in *high utilization & high device count* category even though they are optimized. These paths will suffer the most wearout during in-field operation. Thus Approach 3 still monitors most wearout susceptible paths. Since these paths are also optimized they have more timing margin and hence they would not threaten the systems performance or lifetime. Fig. 2(d) illustrates the new test clock range in which these paths are monitored. The new test clock period

is between T_t and T_{t+GB} instead of between T_0 and T_{0+GB}. T_t is the delay of the slowest optimized path and T_{t+GB} is T_t plus the same guardband. The paths highlighted in black are the ones most susceptible to wearout and have been optimized and are also monitored.

4) Approach 4: Two Monitoring Domains

Approach 3 creates a set of paths that are monitored at an elevated test clock frequency with the assumption that monitoring the most utilized paths that are also optimized will be sufficient to detect wearout. After the path redistribution of Approach 3 there will be a new set of critical paths which are not going to be monitored. These are the paths which have a larger delay than T_t as shown on Fig. 2(d). Note that these paths have lower *predicted* utilization than the paths monitored. Hence the *assumption* is that these paths are less susceptible to wearout. However, during in-field operation if the utilization varies from the utilization profile collected from representative applications then the prediction may not be accurate. In this case, paths that have a smaller timing margin may become susceptible to failure. Approach 4 eliminates this susceptibility by adding additional paths to monitor from these smaller timing margin paths.

Paths optimized and monitored: This approach monitors two groups of paths. Paths in Group 1 are selected the same way as Approach 3. However, Approach 4 selects half the number of paths (*nMonitor*/2) to monitor using the virtual test clock (between T_t and T_{t+GB}). The Group 1 paths selected for monitoring are also optimized as in Approach 3 (number of paths optimized is half of *nMonitor* paths).

Paths selected in Group 1 are removed from *nLong* paths. The second half of the monitored paths, called Group 2, is selected from the paths remaining in *nLong* paths. We sort the remaining paths in descending order based on their utilization and select the top *nMonitor*/2 paths. Group 2 paths are monitored using a test clock with period between T_0 and T_{0+GB} (this is the original test frequency range used by Approach 1 and 2). Thus Approach 4 uses two monitoring test frequency ranges. Group 2 paths are the ones with the least timing margin after optimizing Group 1 paths. Group 2 paths have a delay above T_t as shown on Fig. 2(e).

Modifications to monitoring hardware: The additional monitoring cost incurred in this approach would be due to the need for additional control hardware to enable monitoring at two different test frequency ranges. This slightly more complex monitoring hardware would reduce the sensitivity of monitoring to path utilization profiling accuracy since two distinctly different sets of paths are monitored.

D. Summary of Approaches

Table I summarizes the four approaches discussed. As shown in Table I *nLong* critical circuit paths in the analysis set can be grouped into four categories based on utilization and devices count: (1) *High utilization & high device count* (2) *High utilization & low device count* (3) *Low utilization & high device count* (4) *Low utilization & low device count*. Each of the four approaches (labeled as App. 1 to 4) are going to select a subset of each of the above four path categories to be optimized (*Opt.*) and/or to be monitored (*Mon.*).

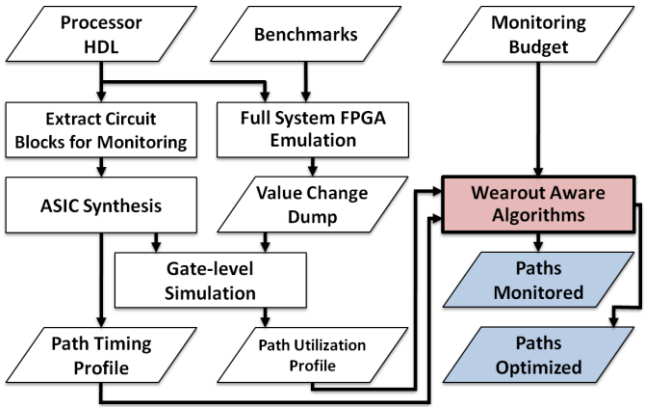

Figure 3. Flow chart of evaluation methodology.

III. EVALUATION METHODOLOGY

In order to implement CLDF and evaluate various approaches discussed in the previous section the evaluation setup must have the following capabilities: First, it should synthesize the circuit design to generate the timing report used in Step 1 discussed in the last section. Any traditional ASIC synthesis tool has this capability. Second, the evaluation setup must take the *nLong* paths obtained from Step 1 and collect utilization profiles of these paths while running representative applications of interest. Running applications, particularly complex applications with system interactions, requires full system simulation capability with support for running an operating system. Hence the circuit being analyzed must be part of a complete processor design to collect utilization profile. Full system simulation can then be run on top of the processor design. Conducting a full system simulation on top of a gate-level processor design is an extremely slow process. Hence, collecting utilization profile from application runs on a gate-level full system simulator is impractical and a new solution is thus required for collecting utilization data. Finally the evaluation setup must be capable of using the utilization profile data to generate a list of optimized and monitored paths.

Taking into consideration these complex requirements, we have designed and implemented a novel hybrid emulation infrastructure. Fig. 3 shows the flow chart of our evaluation setup. There are three inputs; the design code for a full processor, the representative benchmarks that should be used for collecting path utilization data, and the monitoring budget available in term of number of paths which can be monitored. There are two outputs from the design flow. First output is a set of paths that need to be optimized such that the circuit is better suited for wearout monitoring. Second is a list of critical circuit paths that can be used in monitoring hardware. Various blocks in the flowchart will be explained in detail below.

ASIC Synthesis: The experimental setup takes as input complete Verilog HDL design of a processor. In our implementation, we used OpenSPARC T1 processor HDL code [21]. The designer then identifies a select few

functional unit blocks (FUBs) of this processor and marks them as candidates for wearout monitoring. Verilog HDL code of these FUBs is then extracted from the processor HDL. In our implementation, we demonstrate the effectiveness of CLDF framework using four different FUBs from OpenSPARC T1 processor. These FUBs are: (1) Instruction decoder (sparc_ifu_dec) (2) Execution control logic (sparc_exu_ecl) (3) Store buffer controller (lsu_stb_ctl) (4) Register management logic (sparc_exu_rml). The extracted FUBs are then synthesized using ASIC design flow to generate two outputs: gate level circuit implementation, and path timing characterization. We use Synopsys Design Compiler to generate the gate level circuit implementation. IBM 90nm design library is used to synthesize the gate-level netlist of the circuits used in this study. We also use Synopsys PrimeTime timing analysis tools to collect timing characterization of each studied circuit.

FPGA Emulation: The full processor HDL is mapped on a Xilinx ML509 FPGA evaluation platform that uses a Virtex 5 XC5VLX110T FPGA chip. ML509 is specifically designed with enough resources so that the OpenSPARC T1 processor (1 core, 4 threads) can be fully implemented on it. OpenSolaris 11 operating system is booted on top of OpenSPARC T1 on the ML509 platform. This FPGA-based full system emulation can execute unmodified SPARC binaries of most workloads.

The second input to the experimental setup is a collection of representative benchmarks to run. We selected ten unmodified SPEC CPU 2000 benchmarks compiled to run on T1 processor. One important requirement for CLDF is the ability to generate utilization profiles of the selected FUBs while executing the selected benchmarks. For this purpose, we collect inputs signals to the four FUBs selected for our evaluation using Xilinx ChipScope Pro Integrated Logic Analyzer (ILA) IP cores [22]. ChipScope probes the input signals to the selected FUBs while the benchmarks are executing on the FPGA based full system emulation. These inputs are then saved as value change dump (VCD) files.

Gate-level Simulation: The next goal of our experimental setup is to collect path utilization within the selected FUBs while running the benchmarks. We perform gate-level simulation of the selected FUBs using VCD inputs collected from the FPGA emulation. By using VCDs we skip gate-level simulations during those cycles when the inputs do not change. It should be noted that skipping these uninteresting events provides reduced gate-level simulation time without any loss of accuracy in the collected utilization profile. Gate-level simulation is used to collect detailed path utilization information.

During gate-level simulation a Switching Activity Interchange Format (SAIF) file is generated for each simulation run which counts the exact number toggles for each of the gates in the simulated circuit. The utilization of each circuit path is the number of toggles of a node on the path which had the minimum toggles. This approach takes into consideration the fact that many gates are shared among multiple paths and ensures that when utilization of a path is calculated, the toggles for gates which are shared among multiple paths are accounted for accurately.

Cadence NC-Verilog tool suite is used for gate-level simulation of the synthesized netlist generated for selected FUBs. The gate-level netlist is generated using the Synopsys Design Compiler circuit synthesis. The input trace to the FUB is obtained from FPGA emulations which use ChipScope to collect FUB inputs. The output from this simulation is a utilization profile of all the nodes in the netlist.

Wearout Aware Algorithms: The last step in the flowchart shown in Fig. 3 is to execute wearout aware algorithms. This step implements the four algorithms described in Section II.C. This step takes the following inputs: path timing profile collected from ASIC synthesis, path utilization data obtained gate-level simulation, and designer specified constraints such as the number of paths to be monitored. This step identifies paths that need to be optimized and paths that need to be monitored based on each of the four algorithmic approaches.

Discussion of the Experimental Setup: Our innovative multi-tool and multi-phase experimental setup allows us to objectively evaluate the design flow methodology proposed in this paper. It should be noted that the *exact* Verilog HDL design code of OpenSPARC T1 is used for both FPGA and ASIC synthesis. Hence there is an exact match of the functional behavior of the circuit between FPGA emulation and gate-level simulation. Although the internal implementation details of FUBs on a FPGA are significantly different than their implementation in ASIC design, the inputs, outputs, and function of FUBs are identical both in FPGA and ASIC design. The difference is that on FPGA the FUB logic is implemented using lookup tables (LUTs) made of SRAM, while in ASIC the circuit is implemented using gates from a design library. Since FUB inputs in both implementations are identical, the VCD file generated from FPGA emulation can be used as input to simulation.

The fact that the same HDL code for the circuit is used for both FPGA and ASIC implementations makes this setup significantly more flexible and accurate than other alternatives which can be used for cross-layer research such as hierarchical simulation (e.g. the framework used in [17, 23]). This setup is useful for prototyping (pre-fabrication) phase of development of new circuits where cross-layer study of the circuit's behavior can lead to significant reliability, power, area, and performance benefits. This setup allows for accurate evaluation of correlated solutions at different levels of the system design to achieve benefits which are not obtainable by dealing with them individually.

IV. Evaluation Results

We now demonstrate the effectiveness of the CLDF framework using the four selected FUBs from OpenSPARC T1 processor. CLDF provides the designer with the opportunity to tradeoff area and power for reliability in a fully-automated design flow chain. All the designer has to do is to provide benchmarks of interest, a monitoring budget (in terms of number of paths to be monitored) and some guidelines on setting parameter such as *HighUtilCutoff*, *LowUtilCutoff*, and *DevCutoff*.

For the results presented in this paper the monitoring budget (*nMonitor* parameter) is set to 50 paths. We will compare the four approaches to a baseline which simply monitors the 50 slowest paths in the FUB regardless of the utilization profile.

FUBs are synthesized using Synopsys Design Compiler and then Synopsys PrimeTime is used for timing characterization the paths in the FUBs. These FUBs are synthesized for a 1.25ns system clock with OpenSPARC's default 20% guardband added to the clock period. As explained earlier, we use *InitCutoff* parameter to be the same as the clock period guardband which is set to 20%. In other words, paths with less than 20% timing margin are selected to form the *nLong* path group. All three selection parameters *HighUtilCutoff*, *LowUtilCutoff*, *and DevCutoff* are also set to 20% for these evaluations.

A. Detailed Results from sparc_ifu_dec

We first present detailed results and analysis for one FUB from OpenSPARC T1 processor, namely sparc_ifu_dec. Fig. 4(a) shows the initial timing margin distribution for all paths in the sparc_ifu_dec FUB. Total number of paths in the FUB is 5988. These paths are divided into four groups based on their timing margin (*s*): (A) *s*<20% (B) 20%$\leqslant$*s*<40% (C) 40%$\leqslant$*s*<60% (D) 60%$\leqslant$*s*<80%. Using 20% *InitCutoff* value resulted in 2974 paths as *nLong* paths which are shown in group (A). There were 2274 paths in group (B), 537 in group (C), and 203 in group (D). The primary observation from this figure is that there is a relatively steep critical path wall; nearly 50% of all paths have less than 20% margin.

In order to consider the effect of path utilization on their wearout, as described earlier, it is necessary to generate the utilization profile. Utilization profile of all paths is shown in Fig. 4(b). The horizontal axis is divided into three categories. Paths with utilization less than 20% are in low utilization group and paths with utilization within 20% of the maximum utilization of any path in this FUB (which is 55% utilization in this case) are in the high utilization group and finally all paths with utilizations in between the above cutoff values are classified as medium utilization. On the vertical axis the distribution of the paths from each of the four groups A, B, C, and D is shown for each utilization category. Fig. 4(b) shows that most of the paths in the group A have a low utilization while the highest utilized paths are mostly in groups B, C, and D. This result shows that for this FUB only a small group of the critical paths (less than 20% margin) have high utilization. This group of paths is more susceptible to wearout due the high utilization and hence is going to be selected by our algorithms to be optimized and/or monitored. Next, we will look at how each one of the four approaches modifies the implementation of the above FUB.

Approach 1: Initially all paths in the *nLong* group are sorted based on their amount of utilization. As clearly seen in the path utilization profile Fig. 4(b), only a small subset of critical paths has high utilization. Hence when we select paths within the top 20% (*HighUtilCutoff*) of the maximum utilization, the selection results in just 31 paths with utilization between 35.2% and 44.6%. Recall that Approach 1 selects *high utilization & high device count* paths for

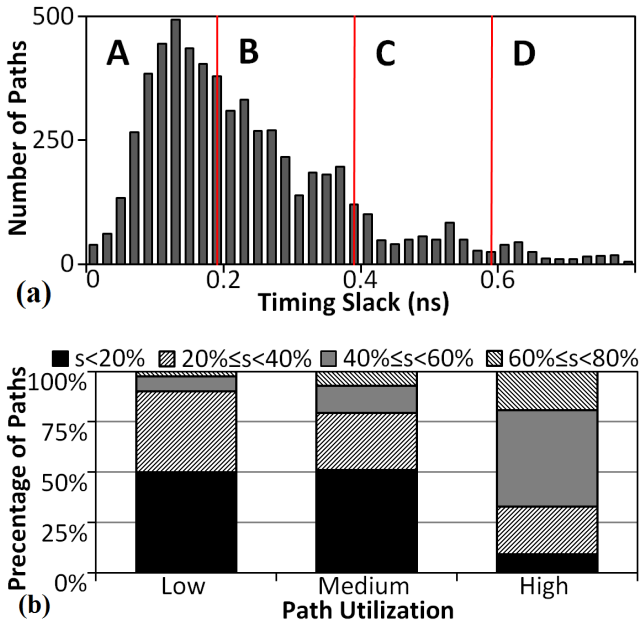

Figure 4. (a) Path timing and (b) path utilization profile of sparc_ifu_dec.

monitoring while all the remaining paths in the *nLong* group are optimized. In this FUB just applying *HighUtilCutoff* parameter resulted in selection of too few paths even before applying *DevCutoff* parameter. Since the monitoring budget allows 50 paths to be selected for monitoring, *DevCutoff* parameter is not applied. In fact, this approach ended up selecting 31 high utilization paths. The remaining 19 paths selected are simply the most utilized paths from the *nLong* paths after the above 31 paths are excluded. These 19 paths are selected for monitoring even though they are not as prone to wearout as the first 31 selected. The remaining 2924 out of the *nLong* paths (2974 paths from group A – 50 monitored paths) are optimized. In order to visually demonstrate how this approach shifts critical path walls, the top graph in Fig. 6(a), labeled "App. 1", shows the timing margin distribution of the circuit paths before (dotted line) and after (solid line) applying Approach 1.

Approach 2: Approach 2 divides the 50 path monitoring budget into two equal parts of 25 paths each. The first group of paths for monitoring is selected using the same policy used for Approach 1. The approach picks the top 25 out of the 31 paths that are in the *high utilization & high device count* category. The 25 selected paths are removed from the *nLong* paths and the remaining paths are resorted purely based on utilization. The top 25 paths are selected as the second group from this newly sorted list for monitoring. All paths are optimized except for the second group of paths in this approach. The first 25 paths plus 2924 form a group of 2949 paths which get optimized in this approach.

Approach 3: Approach 3 selects the same 50 paths as Approach 1 for monitoring. But it also optimizes the same 50 paths to have 20% more timing margin. The same 50 paths that are optimized are also monitored but using an elevated test frequency range. Since vast majority of paths are untouched, the timing distribution graph before and after

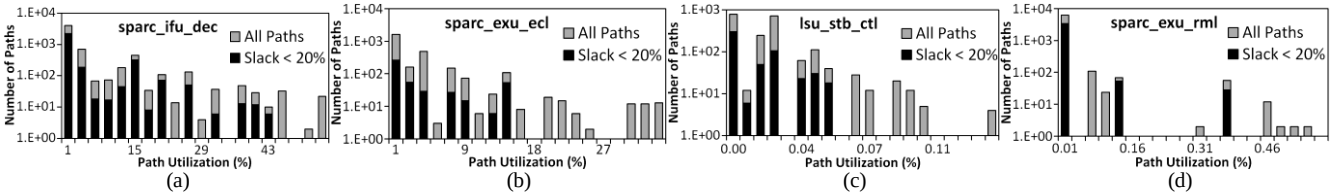

Figure 5. Path utilization profile for (a) sparc_ifu_dec (b) sparc_exu_ecl (c) lsu_stb_ctl (d) sparc_exu_rml (Vertical axis has logarithmic scale).

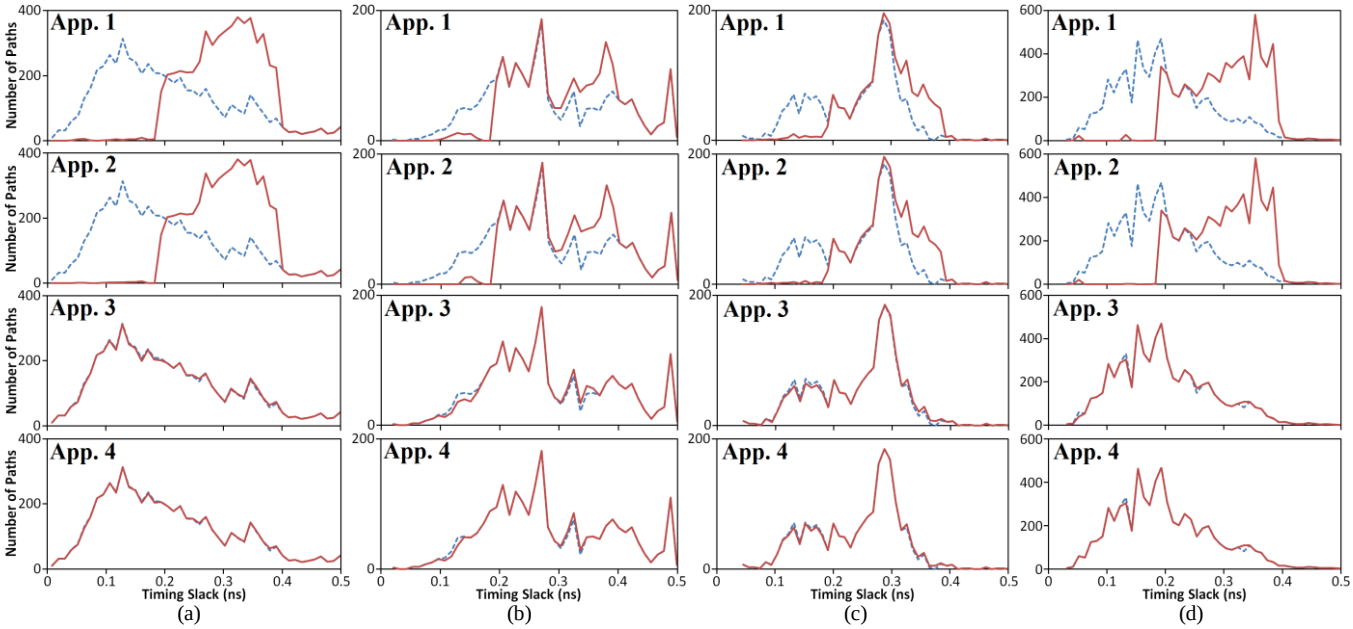

Figure 6. Slack distribution before (dotted line) and after CLDF optimizations on (a) sparc_ifu_dec (b) sparc_exu_ecl (c) lsu_stb_ctl (d) sparc_exu_rml.

applying Approach 3 remain nearly the same, as shown in Fig. 6(a) (graph labeled App. 3)

Approach 4: Approach 4 selects the first group of 25 paths to optimize using the same policy as in Approach 1, namely *high utilization & high device count* paths. These 25 paths are also monitored in Approach 4. However, the second group of 25 paths that need to be monitored are selected purely based on high utilization, but the second group is not optimized. But as described earlier, Approach 4 uses two different test frequency ranges for monitoring. Approach 4 selects 25 paths for optimization. Similar to Approach 3, timing distribution remains nearly the same after Approach 4 optimizations, as shown in Fig. 6(a) (graph labeled App. 4).

B. Path Utilization Profile Analysis of All FUBs

Fig. 5 shows the path utilization profile of four FUBs selected. Note that the vertical axis is in logarithmic scale. First observation from comparison of these plots is that the path utilization profile in each FUB is quite different. For example there are paths in sparc_ifu_dec which are utilized almost half of the time (the columns around 50% on Fig. 5(a)). On the other hand for sparc_exu_rml shown on Fig. 5(d) there are no paths which are utilized more than 1% of the time. This extreme difference in the utilization profile between FUBs further reinforces the need for consideration of application-driven utilization profile in path selection. Furthermore, these utilization profile differences also motivate the need for applying multiple algorithms for path selection.

Utilization distribution of the subgroup of paths with less than 20% margin follows almost the same general trend as all the paths in the circuit, particularly at low utilization level (as shown using black columns on Fig. 5). In other words, if we consider all the paths in the circuit and observe that the number of paths with 1% utilization is much more than the number of paths with 10% utilization then the same observation holds if we just use the paths with less than 20% timing margin. In the FUBs studied, there is a limited group of paths with high utilization and many of these paths have more than 20% timing margin.

Fig. 5 shows that there is a notable clustering of paths with low utilization. It is not surprising that even though a circuit has many paths, majority of paths have low utilization in these FUBs. In other words, only a limited number of paths are frequently utilized. Since utilization is the primary driver for wearout, identifying these paths will enhance monitoring efficiency without decreasing confidence.

C. CLDF Overheads

Table II shows the number of paths selected for optimization (labeled *Opt paths*) by each of the four approaches and the area overhead of optimizations. Each approach leads to different area overheads depending on the

TABLE II. COMPARISON OF FOUR APPROACHES FOR DIFFERENT FUBS

Circuit block	Description	Area (μm^2)	*nLong*	Approach 1		Approach 2		Approach 3		Approach 4	
				Opt Paths	*% Area overhead*	*Opt Paths*	*% Area overhead*	*Opt Paths*	*% Area overhead*	*Opt Paths*	*% Area overhead*
sparc_ifu_dec	Instruction decode	87352	2974	2924	15	2949	25	50	3	25	2
sparc_exu_ecl	Exec. control logic	255900	449	399	0	424	0	50	0	25	0
lsu_stb_ctl	ST buffer control	95131	533	483	0	508	0	50	0	25	0
sparc_exu_rml	Reg. management logic	175666	3449	3399	7	3424	7	50	2	25	1

structural, timing, and utilization characteristics of the circuit. sparc_exu_ecl and lsu_stb_ctl FUBs do not show a steep critical path wall. Hence the area overhead of all approaches for these FUBs is almost zero. In essence, our approaches do not introduce any additional overhead when the FUB is already well suited for monitoring. On the other hand, sparc_ifu_dec and sparc_exu_rml have a steep critical path wall as shown on Fig. 6 (a) and (d). As a result there are overheads associated with all 4 approaches for these FUBs. The area overhead of Approach 1 is always higher for these FUBs since majority of paths are optimized. In Approach 2 increased correlation between the paths monitored and paths optimized results in a more robust monitoring that has less reliance on path utilization profile. However, it increases the area overhead in some cases, since 25 additional paths are optimized. To increase monitoring robustness Approach 1 and 2 optimize many paths. As a result these two approaches can dramatically alter the timing distribution. This shift is also clearly seen in Fig. 6 (a) and (d) for Approaches 1 and 2 where timing redistribution looks significantly different.

Approaches 3 and 4 limit the number of paths to be optimized and hence the area overhead is significantly reduced. Approaches 3 and 4 enhance monitoring robustness by jointly optimizing the paths and relying on elevated test frequency to control the area overhead. Since Approaches 3 and 4 only change the timing distribution of a fixed number of paths (*nMonitor* paths) they do not fundamentally alter the initial timing distribution. This can be clearly seen in Fig. 6 where the path distribution after Approaches 3 and 4 are very similar to the initial distribution.

It summary, it has been shown that different FUBs benefit from different approaches. These differences are due to the differences in the timing margin distribution and utilization profile of FUBs. Aggressive optimization of the FUB is not always an outcome the algorithms presented and in scenarios where a FUB does not have a steep critical path wall the ideal group of paths to be monitored can be selected without the adding any additional optimization overhead. One interesting aspect of these four different approaches is that their overheads are dependent on the initial timing and utilization profile of the circuit and hence each approach is more suitable for a category of circuits with specific characteristics.

V. RELATED WORK

Predictions by the International Technology Roadmap for Semiconductors (ITRS) [24] for more severe wearout in future technology generations has resulted in increased research efforts in modeling, detecting, and predicting wearout. Some methods have been specifically developed for prediction of wearout related timing failures [11, 12, 14]. These methods tackle the problem of wearout at microarchitecture level or by making circuit level enhancements. While these approaches focus only at one layer of the system design, in this work we presented a novel approach which correlates microarchitectural wearout prediction techniques with the circuit design implementation. The resulting circuit design is aware of the presence of wearout monitoring and hence can make monitoring more robust and efficient. Our work takes the utilization of the circuit paths driven by application level information to change circuit implementation.

Research in using runtime behavior during circuit design time has expanded in the recent years. Many of these efforts target improved power consumption or operation at reduced error rates. Design time error rate analysis has been used for improving reliability in presence of variations [25]. Circuit modifications proposed in [25] make the implementation of the circuit more suitable for timing speculation [26]. In Blueshift [27] targeted acceleration for frequently exercised path is used to change circuit implementation with the goal of improving performance of timing speculation even in the presence of a critical path wall. In [23], the authors presented a design time optimization with the goal of reducing error rates even when the circuit is operating at a reduced operation voltage. These approaches allow for more aggressive voltage scaling and increased power savings without impacting reliability.

In many of these prior studies the circuit is intentionally operated at a higher than nominal clock frequency resulting in some circuit paths not meeting the timing constraint. In contrast our approach does not do timing speculation. Rather the goal is to continuously monitor circuit wearout efficiently. Hence, the design changes necessary for wearout monitoring are quite different than those necessary for timing speculation. We have exploited runtime path utilization information, as was done in [27] which has a different end goal of improving timing speculation. We use runtime information about how the design is used to reduce the number of paths needed for monitoring. We have taken advantage of the graduate nature of wearout and its dependence on utilization to correlate design time optimization efforts with runtime wearout monitoring enhancements. The resulting cross-layer resiliency framework improves the effectiveness and efficiency of *in situ* circuit monitoring techniques.

VI. CONCLUSIONS

As device sizes in a processor continue to shrink with each new process technology, there is a growing concern for

reliability. While reliability issues can take different forms, wearout is a prevalent degradation where circuit timing margins gradually decrease over the lifetime of the circuit. Continuously monitoring wearout will become critical. By monitoring the amount of timing margin left in a circuit it is possible to enable just-in-time error detection and correction solutions. Since monitoring itself will be done continuously it is necessary to improve monitoring efficiency. Selecting only the most critical paths to monitor can reduce monitoring overhead. But in the presence of critical path timing wall, monitoring overhead can be significant due to the need to monitor many paths. This research addresses this serious bottleneck to monitoring in the presence of critical path timing walls.

We present a cross-layer design flow that uses application knowledge to separate more frequently used critical paths from the ones with low utilization. Since wearout is a function of utilization, using application level information to derive path utilization provides new ways to improve monitoring efficiency and robustness. We describe four approaches to redistribute the timing of circuit paths that take advantage of this cross-layer utilization information. All these approaches provide the designer the ability to tradeoff monitoring robustness with power and area overheads.

The proposed design is implemented in a novel evaluation framework that allows application level information and circuit design tools to interact and exchange information. Our evaluation framework provides an automated mechanism to generate the best set of paths that need to be monitored given the design constraints. Using OpenSPARC T1 processor FUBs we evaluated the four proposed approaches. Our results show that all four approaches have unique capabilities that allow them to be applied to FUBs with different characteristics.

ACKNOWLEDGMENT

This work was supported by National Science Foundation grants CAREER-0954211, CCF-0834798, CCF-0834799, and an IBM Faculty Fellowship.

REFERENCES

[1] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," Microarchitecture, vol. 25, pp. 10-16, 2005.

[2] S. Borkar, "Electronics beyond nano-scale CMOS," Design Automation Conference, pp. 807-808, 2006.

[3] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," International Symposium on Computer Architecture, pp. 276-287, 2004.

[4] J. Shin, V. Zyuban, Z. Hu, J. A. Rivers, and P. Bose, "A framework for architecture-level lifetime reliability Modeling," Dependable Systems and Networks, pp. 534-543, 2007.

[5] W. P. Wang, V. Reddy, A. T. Krishnan, R. Vattikonda, S. Krishnan, and Y. Cao, "Compact Modeling and simulation of circuit reliability for 65-nm CMOS technology," IEEE Transactions on Device and Materials Reliability, vol. 7, pp. 509-517, 2007.

[6] C. Kenyon, A. Kornfeld, K. Kuhn, M. Liu, and A. Maheshwari, "Managing Process Variation in Intel's 45nm CMOS Technology," Intel Technology Journal, vol. 12, pp. 93-109, 2008.

[7] J. Hicks, D. Bergstrom, M. Hattendorf, J. Jopling, and J. Maiz, "45nm Transistor Reliability," Intel Technology Journal, vol. 12, pp. 131-144, 2008.

[8] J. Abella, X. Vera, and A. Gonzalez, "Penelope : The NBTI-Aware processor," International Symposium on Microarchitecture, pp. 85-96, 2007.

[9] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "Adaptive Techniques for Overcoming Performance Degradation due to Aging in Digital Circuits," Asia and South Pacific Design Automation Conference, pp. 284-289, 2009.

[10] A. C. Cabe, Z. Y. Qi, S. N. Wooters, T. N. Blalock, and M. R. Stan, "Small Embeddable NBTI Sensors (SENS) for Tracking On-Chip Performance Decay," International Symposium on Quality Electronic Design, pp. 1-6, 2009.

[11] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra, "Circuit failure prediction and its application to transistor aging," VLSI Test Symposium, pp. 277-286, 2007.

[12] J. Blome, S. Feng, S. Gupta, and S. Mahlke, "Self-callibrating online wearout detection," International Symposium on Microarchitecture, pp. 109-122, 2007.

[13] E. Karl, P. Singh, D. Blaauw, and D. Sylvester, "Compact in situ sensors for monitoring negative-bias-temperature-instability effect and oxide degradation," Solid State Circircuts Conferance, pp. 410–411, 2008.

[14] B. Zandian, W. Dweik, S. H. Kang, T. Punihaole, and M. Annavaram, "WearMon: Reliability Monitoring Using Adaptive Critical Path Testing," Dependable Systems and Networks, pp. 151-160, 2010.

[15] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, and T. Austin, "Ultra low-cost defect protection for microprocessor pipelines," ACM Sigplan Notices, vol. 41, pp. 73-82, Nov 2006.

[16] J. Patel, "CMOS Process Variations: A Critical Operation Point Hypothesis," Online Presentation, 2008.

[17] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack Redistribution for Graceful Degradation Under Voltage Overscaling," Asia and South Pacific Design Automation Conference, pp. 825-831, 2010.

[18] Y. J. Li, S. Makar, and S. Mitra, "CASP: Concurrent Autonomous chip self-test using Stored test Patterns," Design, Automation and Test in Europe, pp. 885-890, 2008.

[19] A. H. Baba and S. Mitra, "Testing for Transistor Aging," VLSI Test Symposium, pp. 215-220, 2009.

[20] D. Ernst, N. S. 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," International Symposium on Microarchitecture, pp. 7-18, 2003.

[21] http://www.opensparc.net/opensparc-t1/index.html.

[22] http://www.xilinx.com.

[23] A. Kahng, S. Kang, R. Kumar, and J. Sartori, "Designing Processors from the Ground Up to Allow Voltage/Reliability Tradeoffs," High Performance Computer Architecture, pp. 1-11, 2010.

[24] http://www.itrs.net/.

[25] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas, "EVAL: Utilizing Processors with Variation-Induced Timing Errors," International Symposium on Microarchitecture, pp. 423-434, 2008.

[26] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, "Opportunities and challenges for better than worst-case design," Asia and South Pacific Design Automation Conference, pp. 2-7, 2005.

[27] B. Greskamp, L. Wan, U. R. Karpuzcu, J. J. Cook, J. Torrellas, D. M. Chen, and C. Zilles, "BlueShift: Designing Processors for Timing Speculation from the Ground Up," High Performance Computer Architecture, pp. 213-224, 2009.