Blade – A Timing Violation Resilient Asynchronous Design Template

Dylan Hand, Benmao Cheng[1], Melvin Breuer, Peter A. Beerel

Computer Engineering Technical Report Number CENG-2014-04

Ming Hsieh Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, California 90089-2562

May 10th, 2014

[1] Benmao Cheng is a visiting scholar from China.

Abstract—This paper proposes a novel asynchronous design template, Blade, that uses single-rail logic, a reconfigurable delay line, and error detecting latches to reliably detect and recover from timing violations due to process variations and delay faults of single event upsets. The template employs a novel speculative handshaking paradigm that improves average-case performance by taking advantage of the fact that errors occur with low probability. We will analytically compares the performance of this template with both traditional synchronous designs and the state-of-the-art synchronous resiliency strategy Bubble Razor. Our results demonstrate the potential benefit of our approach as well as provide insight into how asynchronous designs should be optimized to achieve these benefits.

Keywords—Resilient design, variability, performance analysis.

I. Introduction

Traditional synchronous design must incorporate timing margin to ensure the correct operation under worst-case delay conditions. However, the ongoing increase in process variation compounded with aging effects is causing progressively larger delay variations, requiring more substantial timing margins reducing the performance and energy efficiency of traditional designs. To address this problem, many synchronous design techniques for resilient designs have been proposed that address delay variations. For example, canary FFs predict when the design is close to a timing failure (see e.g., [1]). Designs can then adjust their supply voltage or clock frequency either statically or dynamically to ensure correct operation at the edge of failure. In addition, Razor circuits [2] [3] [4] [5] have been proposed that feature in situ timing violation detection mechanisms, which allow recovery from timing errors via architectural replay or automatic pipeline stalling, further reducing margin.

Asynchronous circuits have been identified as a potentially more effective approach, particularly in the near-threshold regime (see e.g., [6] [7] [8]). The basic difference between asynchronous and synchronous design is that asynchronous designs utilize additional circuitry that indicates when individual blocks have finished computing instead of a global clock signal. There are two common asynchronous design styles that achieve this goal in very different manners. The first relies on dual-rail quasi-delay-insensitive (QDI) logic to embed the completion signal into the data representation. The basic problem with this design style is that implementations are much larger than the synchronous counterpart (often 4x larger) and have very high switching activity due to a return to zero paradigm (see e.g., [9]). The second design style, bundled-data (or micropipelines [10]), relies on delay lines that are matched to individual clouds of combinational single-rail logic. The advantage of this approach is that the switching activity within a logic cloud is essentially the same as in synchronous design and can be quite low. Moreover, the total area of the delay lines is similar to that of a clock tree and thus the overall area of bundled-data style is comparable to that of synchronous designs (see e.g., [11]). The Achilles heel of this design style is that the delay line must be conservatively designed to be longer than its corresponding logic under all possible process, temperature, and voltage corners. Consequently, in the presence of aggregated on-chip variations in the near threshold domain, the delay lines must be implemented with huge margins, taking away most if not all the advantages of asynchronous design.

The ideal ultra-low-power asynchronous design style would have the area close to that of bundled-data with the variation-tolerance and high-performance of QDI designs, and for many years this has been an elusive goal of asynchronous researchers. Researchers have proposed bundled-data designs coupled with layout techniques to mitigate variability such as duplicating the bundled-data delay lines [8] and constraining the design to regular structures such as PLAs [12] and soft latches [13]. Others suggest current-based completion sensing techniques (e.g., [14] [15]) that rely on analog current sensors that are themselves tricky to design when there is high transistor variability.

Our proposed approach is an all-digital asynchronous design template, Blade, which uses re-configurable delay lines that can be tuned and optimized to mitigate the impact of delay variations. The template consists of single-rail logic, reconfigurable delay lines, and razor-like [2] latches with asynchronous sampling circuitry that reliably handles errors even under the presence of metastability. The template employs a novel speculative handshaking paradigm that improves average-case performance by taking advantage of the fact that errors will have a low probability of occurrence.

The focus of this paper is to introduce the Blade template, characterize its unique features, and provide useful future extensions to this work and the tradeoffs involved in different design decisions. We will also provide an analytical model to quantify Blade's benefit over both traditional synchronous designs and the state-of-the-art synchronous resiliency strategy Bubble Razor. Finally, we will compare these analytical models to a Verilog model of the proposed template.

The remainder of this paper is organized as follows. Section 2 provides relevant background on Bubble Razor and its performance. Section 3 provides details of our proposed templates and their operation. Section 4 describes potential improvements to the Blade template. Sections 5 and 6 explain our model of performance and quantifies the potential benefits over both traditional synchronous design and Bubble Razor. Section 7 summarizes our results and outlines future work.

II. Background

A. Bubble Razor

Bubble Razor (BR) inherits the features of previous Razor techniques enabling real-time error detection and correction [4] [5]. Unlike other Razor architecture, it is based on a two-phase latch-based design, in which each traditional flip-flop is replaced with two latches that undergo retiming to have approximately equal amount of logic between each latch. It uses a novel bubble propagation algorithm that makes the approach applicable to any architecture and enables the

automatic application of this technique to legacy flip-flop based RTL designs, significantly reducing barriers to adoption.

Bubble Razor flags a timing violation when the data arriving at a latch varies after the latch opens using an error detecting latch (EDL). Upon detecting a timing violation, the circuit automatically recovers by stalling the subsequent latch, giving it an additional clock cycle to process the data. Half of the additional clock cycle is used to compensate for the unexpectedly large delay from the previous latch and the other half accounts for the delay from the current latch to the subsequent one. Thus timing violations are corrected as long as the real delay of each half clock-cycle *step* never exceeds one clock cycle of time.

However to ensure correct operation, stalling the subsequent latch is not sufficient. Upstream stages must be stalled to ensure valid data is not overrun and downstream stages must be stalled to ensure corrupt data is not accidently interpreted as valid. Previous Razor structures use counter-flow pipelining or architectural replay to recover from the stall [2] [16]; however, both techniques require the RTL to be designed with Razor in mind. The latch-based scheme in BR enables an automatic local stall propagation algorithm.

Consider the 2-stage ring in Figure 1 that consists of 4 latches with associated clock gating logic that implements the stall propagation algorithm. A timing violation causes an error signal to be sent to its Right Neighbor (RN) to tell it to stall. Then, the stalling spreads both forward and backward directions around the ring in a wave-like pattern. For example, in Figure 1, the timing violation occurs in latch 2 and this triggers a stall in latch 3. The clock gating logic for latch 3 then spreads the stall forward to stage 4 and backward to latch 2. Clock gating logic that receives stalls from both directions terminates the spreading of stalls. This is called *stall annihilation*. For example, in Figure 1, the stall is terminated by the clock gating logic of latch 1 because it receives stalls from both of its neighbors, i.e., latches 2 and 4.

Unlike other Razor schemes, one significant weakness of Bubble Razor is that it does not consider the impact of metastability in the error detecting logic. As the shadow latch closes at a time when errors are expected to happen at some frequency, metastability at the output of the shadow latch may occur. The metastable state may propagate through the error detection logic (XOR followed by a dynamic OR gate). If this state persists for longer than half a clock cycle, it will be latched into the control logic resulting in a system failure. This oversight significantly reduces the mean time before failure for many applications.

B. *Performance analysis of Bubble Razor*

To analyze the performance of Bubble Razor, the bubble propagation algorithm can be modeled using a Markov Chain. In particular, [17] considered an N-stage ring containing 2N latches with no primary inputs or outputs. There are two categories of states for a latch (and its corresponding clock gating logic): working and stalling. In a working state, the latch closes and opens normally in the current cycle. A latch in a stalling state does not open which prevents new data from propagating and keeps the output fixed in during the clock

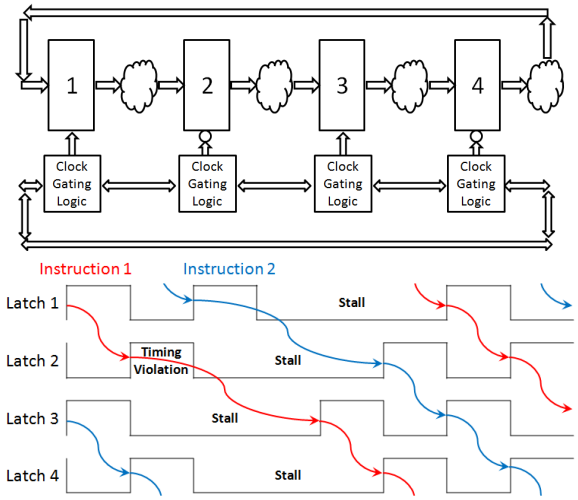

Figure 1. Bubble Razor block and timing diagrams

cycle. In other words, a latch can pass data only when it is in a working state.

The authors in [17] model the timing cost for error correction with the notion of an *Effective Clock Cycle Time* defined as the average time to process each instruction. Consider M clock cycles with a real clock cycle time C and a total time period of $M * C$. The effective clock cycle time (EC) can be expressed as follows:

$$EC = \frac{M * C}{M * \pi(working)} = \frac{C}{\pi(working)} \quad (1)$$

where π(working) is the steady state probability of a latch being in a working state obtained from their Markov Chain analysis.

It may be insightful to review the lower and upper bounds on EC. If every combinational cloud delay is shorter than half a clock cycle time (0.5C), no timing violation happens. This means π(working) = 1 and consequently the lower bound on EC = C. The upper bound on EC occurs when all combinational cloud delays are longer than 0.5C, but shorter than C to guarantee the circuit's correctness. In this case, $\pi(working)$ = 0.5 because every latch of the circuit stalls and works alternately, making EC = 2C.

C. *Delay Distribution*

Based on the Markov Chain model, EC can be expressed as a function of C (real clock cycle time), *p* (probability of timing violation for a latch) and N (number of stages referring to 2N latches in BR or N registers in traditional register-based circuits). It's obvious that *p* is influenced by C. The variable d is used to represent the real delay of a step, i.e., the logic delay from one latch to its Right Neighbor. So *p* can be expressed as follows:

$$p = P_R\{d > \frac{C}{2}\} \quad (2)$$

When considering process variation and aging, the variable *d* is a random variable with some distribution. We follow the approach in [17] and consider two different distributions –

normal and log-normal. Both require only two variables to describe them, i.e. a mean μ and standard deviation σ. The log-normal distribution has a heavy tail that has a basis in the underlying technology in near-threshold domains [18] [19].

In particular, we will explore the performance of traditional, bubble-razor, and Blade circuits with different amounts of variability, as quantified by different σ/μ ratios. The larger this ratio is, the higher the variation. However, when comparing BR and blade circuits to traditional synchronous circuits, i.e. circuits in which there is no dynamic error correction mechanism, we must also compare distributions for circuits that have different delay lengths, which is correlated to different mean delay length μ. Fortunately, reference [20] observes that for die-to-die variations σ/μ ratio is almost a constant for different logic depths, i.e. different delay lengths. For circuits with significant within-die variation, on the other hand, σ/μ ratio decreases for longer paths, i.e., larger μ (e.g., see [20]). Moreover to analyze the lower bound of C, it will also be important to reason about the distribution of the sum of two normal/log-normal variables. References [19] [21] prove that it is reasonable to use another normal/log-normal variable to represent the sum of two normal/log-normal variables.

D. Bubble Razor Systematic Error Rate

It is important to recognize that C cannot be too small because bubble razor must guarantee that every actual delay between adjacent latches must be shorter than one clock cycle or the additional timing compensation would not be sufficiently long to ensure correctness. Since normal/log-normal distributions do not have an upper limit, the authors set a rule that the systematic error rate (SER) should be smaller than some small fixed amount. For example, in their results, they assume SER ≤ 0.1%.

When comparing BR circuits to their traditional circuits, the authors ensure that their SER is the same. For traditional circuits, SER is calculated as:

$$SER = 1 - [P_R\{D \le C\}]^N \le 0.1\% \quad (3)$$

where D is a random variable with a mean twice as much as that of d, the delay between neighboring latches in BR circuits. For BR circuits, reference [17] showed that the error rate could be conservatively estimated to be

$$1 - [P_R\{d \le C\}]^{2N} \le 0.1\% \quad (4)$$

III. Proposed Blade Templates

A. Template Overview

The proposed Blade templates are based on the pipeline block diagram shown in Figure 2. The templates use single-rail logic followed by error detecting latches and two reconfigurable delay lines. The first delay line is of length δ and controls when the error-detecting latch first samples the data at the output of the combinational logic. In particular, it samples the data δ time units after the input request is received assuming no error has occurred in the previous stage.

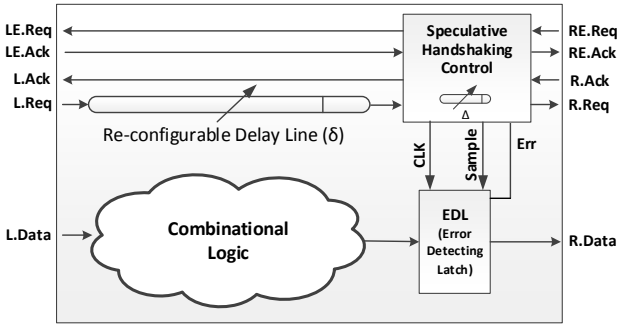

Figure 2. The Blade template

The second delay line is of length Δ and defines a time window during which errors are allowed, referred to as the resiliency window. If the combinational output changes during the resiliency window, the latches flag a timing violation by asserting the *Err* signal, which is sampled by the controller. The asynchronous control circuit then uses a novel protocol to communicate with its right neighbors to recover from the error by delaying the opening of the next latch until the new data has propagated through the combinational logic, as will be described in more detail later.

Each stage has four asynchronous channels that operate using a two-phase protocol. The first channel, L, is a typical bundled-data channel, comprised of Req, Ack, and Data. The second channel, LE, is a pull channel the handshaking controller uses to check if the previous stage suffered an error. It too has a Req and Ack, but no data value is required. Two additional channels, R and RE, will become the L and LE of the next stage.

B. Error Detecting Latch

As in bubble-razor [4], we propose using error-detecting latches that detect if signals are not valid upon the latch going transparent, and if so, generate an associated error signal to the controller. The latched value is valid as long as the data becomes valid before the latch becomes opaque. In other words, the pulse width of the latch, Δ, determines how much timing resiliency is allowed.

While there are many possible implementations of EDLs (e.g., [3] [22] [23] [24] [25] [26] [27]), we will focus on latches with the following properties: 1. The EDL is more sensitive than the combinational logic datapath to ensure small glitches are properly recorded as errors; 2. Once an error is detected, the Err signal will stay asserted for the remainder of the clock period; and 3. The latch will not enter a metastable state during the resiliency window. The TDTB latch in [26], with some minor modifications, fits these criteria. A general structure of an EDL is shown in Figure 3, consisting of a latch, error detector, and sampling circuit.

Metastability (MS) in the latch is not a concern as we will ensure Δ is set sufficiently large as to avoid sampling while the datapath is still evaluating. However, the possibility of the error signal itself becoming MS cannot be avoided. Therefore, a sampling circuit is used to ensure the *Err* output is always stable, even in the presence of MS, by coupling it with a MS

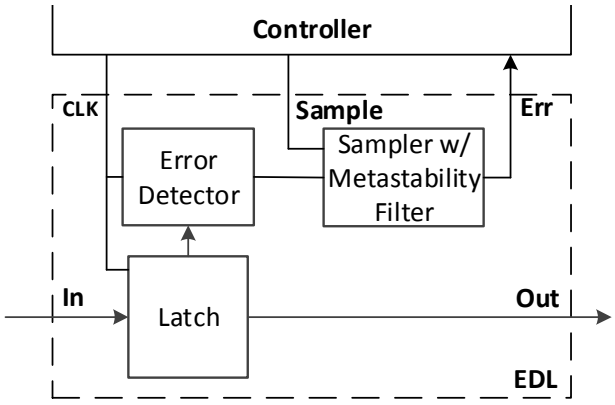

Figure 3. General structure of Error Detecting Latch

filter. MS filters are typically implemented using dual-rail outputs that remain neutral until MS has resolved. An example of such a circuit is the Q-Flop [28]. In rare cases, the output of the Q-Flop will take a long time to resolve while either its internal nodes are metastable due to an input transition as the flop closes or if the input itself is metastable. In a robust synchronous design, this resolution delay would translate into increased margins or extra clock cycles and synchronizers to wait-out this rare occurrence. However, because our design is asynchronous, it will gracefully wait for the MS state to resolve before allowing the next stage to open its latch, effectively stalling the stage and ensuring correct operation.

C. Speculative Handshaking Protocol

The proposed Blade template implements a new form of asynchronous handshaking called *speculative handshaking*, illustrated in Figure 4. A request signal between blocks is *speculatively* asserted assuming the delay line of length δ is sufficiently long and no timing violation occurs. A secondary *extend* channel, LE in Figure 2, is used to relay the error signal to the next stage which indicates if this assumption was incorrect and a violation was detected. Using this return channel, the previous stage, which in error, is in control of how long the next stage will need to wait for a clean data input. To implement this delay, we make use of the Req/Ack handshake that occurs on the extend channel. More specifically, the delay in receiving a request on the extend channel to sending an acknowledgement is variable: when no error occurs, the delay will be zero and the acknowledgement occurs immediately, while the acknowledgement will be delayed by Δ when a timing violation has occurred, as illustrated in Figure 4(a) and (b), respectively.

We propose two templates that employ these two delay lines differently. The first template called Blade-O is designed to tolerate mild to moderate variations and the second template called Blade-OC is designed to tolerate higher variations.

D. Blade-O: Delay Opening of Latch

The simplest Blade controller, referred to as Blade-O, is based on an assumption that each stage communicates on its extend channel *before* it opens its own latch and asserts its own output request signal. In particular, if the extend channel

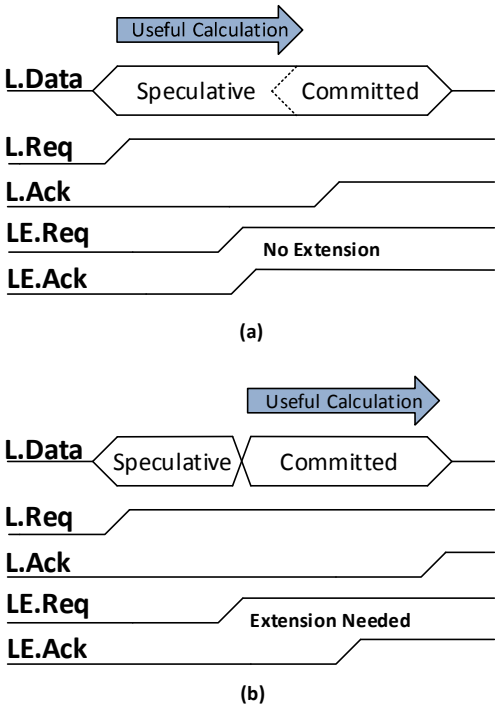

Figure 4. Two-Phase Speculative handshaking protocol when (a) no timing error and (b) a timing error occurs

indicates that the input data was invalid when the initial request was asserted, the control circuit will delay *both* the opening of the latch and the assertion of the output request by Δ as illustrated in Figure 5. The timing violation is identified at the falling edge of latch 2 and is used to delay the subsequent opening of latch 3 by Δ.

The underlying assumption in this template is that the previous stage knows if an error occurred before the short delay line δ is completed. Otherwise, the controller would not know whether or not to delay opening of the next latch. Because the Err signal is sampled Δ time units after the short delay line of length δ is triggered, this assumption can be formally stated as $\Delta \leq \delta$. More precisely, in order to use all of the time Δ to capture a timing violation, $\delta - \Delta$ must be greater than the propagation delay of the error and extend signals.

E. Blade-OC: Delay Opening or Closing of Latch

For systems with high-variance, the assumption of Blade-O that $\Delta \leq \delta$ can limit average-case performance. That is, for systems with high-variance the ideal nominal delay might be significantly less than half of the worst-case delay. For such systems, we propose a more complex Blade controller called *Blade-OC.*

In Blade-OC, the communication channels between controllers remain the same, but the controller itself becomes more complex. Instead of checking the previous stage for errors once, the Blade-OC controller makes two handshakes on the extend channel with the previous stage's controller. We will first describe the second handshake, as the first handshake is similar to Blade-O. Take for example the simple 3-stage

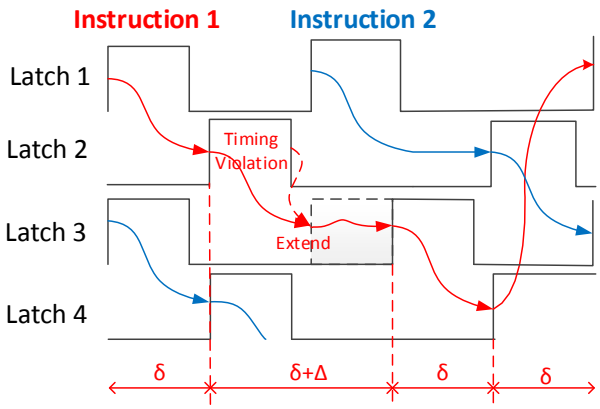

Figure 5. Timing diagram of the Blade-O template

pipeline in Figure 6. A request generated by Stage A arrives at Stage B after δ. Stage B's controller will accept the request and speculatively open its latch while speculatively forwarding the request to Stage C. Before the controller in Stage B closes its latch, it will send a request on its LE channel to Stage A. If Stage A has detected an error in its EDL, it will delay the acknowledgement of the extend request by Δ, which in turn delays the *closing* of Stage B's latch by Δ. This allows enough time for the correct data to propagate through the combinational logic between A and B, through B's latch, and into the B to C datapath. However, the request from B to C has already been speculatively sent at this time, so to ensure Stage C latches the correct data, the opening of its latch must be delayed in a manner similar to the Blade-O template. This is implemented using an additional handshake on the LE channel just as the request arrives through the nominal delay line. When Stage C receives the request, it will initiate a handshake on its LE channel to Stage B, which will then acknowledge the extend channel quickly if its latch closed on time (no error in Stage A) or Δ later if Stage A forced Stage B's latch to close late.

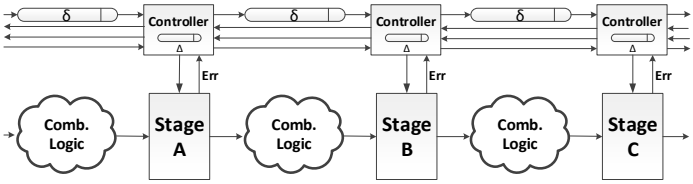

Figure 6. Three Blade-OC stages in a pipeline

Therefore, the difference between Blade-O and Blade-OC is twofold. First, the controller must delay the *closing* of its latch if the previous stage suffered an error. Second, the controller must delay the *opening* of its latch if the previous stage delayed the closing of its latch, or in other words, if an error occurred two stages prior to the current stage.

The timing diagram of the Blade-OC template is illustrated in Figure 7. A timing violation is identified at the falling edge of latch 2. This causes both the subsequent falling edge of latch 3 as well as the rising edge of latch 4 to be delayed by Δ. More specifically, latch 3's controller sends an extend request before closing latch 3, but latch 2's controller will delay the acknowledgement by Δ, forcing latch 3 to remain open for another Δ. Latch 4's controller then sends an extend request to latch 3's controller, which delays the acknowledgement by Δ, forcing latch 4 to remain closed for an additional Δ.Notice that the underlying assumption of the Blade-OC template is that $\Delta \leq 2\delta$ which guarantees the subsequent blade controller has time to delay the opening of its latch. In addition, we assume delaying latch 3 by Δ is sufficient to satisfy our basic SER assumption. Letting the delay of the three stages be d_1, d_2, d_3, with the same mean and variance, we assume that:

$$P_R\{d_1 + d_2 + d_3 \leq 3\delta + 2\Delta\} \leq P_R\{d_1 \leq \delta + \Delta\} \qquad (5)$$

Because of this assumption, as in the Blade-O template, the delay of a pipeline stage, as measured by the delay from input request to output request, is either set to δ or to $\delta + \Delta$. The difference is that the assertion of the extend signal from the Blade-OC controller that causes this extension can arise when the combinational delay of two stages back is larger than its nominal delay δ.

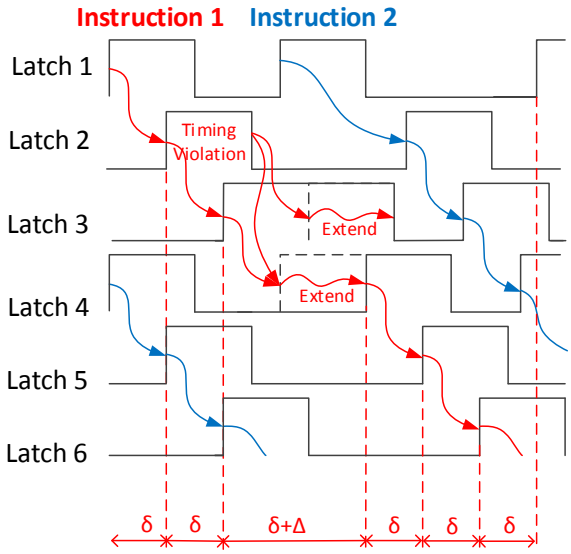

Figure 7. Timing diagram of the Blade-OC template

The advantage of Blade-OC over Blade-O is that the timing requirement $\Delta \leq 2\delta$ is more relaxed than the Blade-O requirement that $\Delta \leq \delta$. In particular, it offers significantly more flexibility in design because it allows the nominal delay of a pipeline stage δ to be as little as 1/3 of the worst-case delay $\delta + \Delta$.

In addition to a more complex controller, for the Blade-OC system to hide hold time and backward delay overheads, every loop in the design has to have at least three asynchronous pipeline stages. Otherwise, when a timing violation occurs, the stage that delays opening its latch would need to be further delayed to avoid violating the hold times of its right neighbor/predecessor whose latch closes much later. In a typical translation from a flop-based synchronous design, this means that each synchronous block of combinational logic (making up one synchronous pipeline stage) would be finer-grained pipelined into up to three blocks of logic, making up to three back-to-back asynchronous pipeline stages. This is why

Figure 7, the timing of a Blade-OC implementation of a 2-stage synchronous pipeline, shows the timing of six latches. Each of these stages will have its own bank of error-detecting latches. This costs a significant amount of area and power and thus should only be used when the higher resiliency to variations is required. That said, another benefit of this more complex architecture is that the nominal delay of each stage, δ, is now one-third the nominal delay of the comparable synchronous pipeline stage and thus the backward latency associated with the asynchronous handshaking protocol can take up to two-thirds of the nominal cycle time before becoming a bottleneck to the nominal handshaking scenario. Given the backward latency can often be reduced to a few gate delays (e.g., [29]), this can be easy to meet in practice.

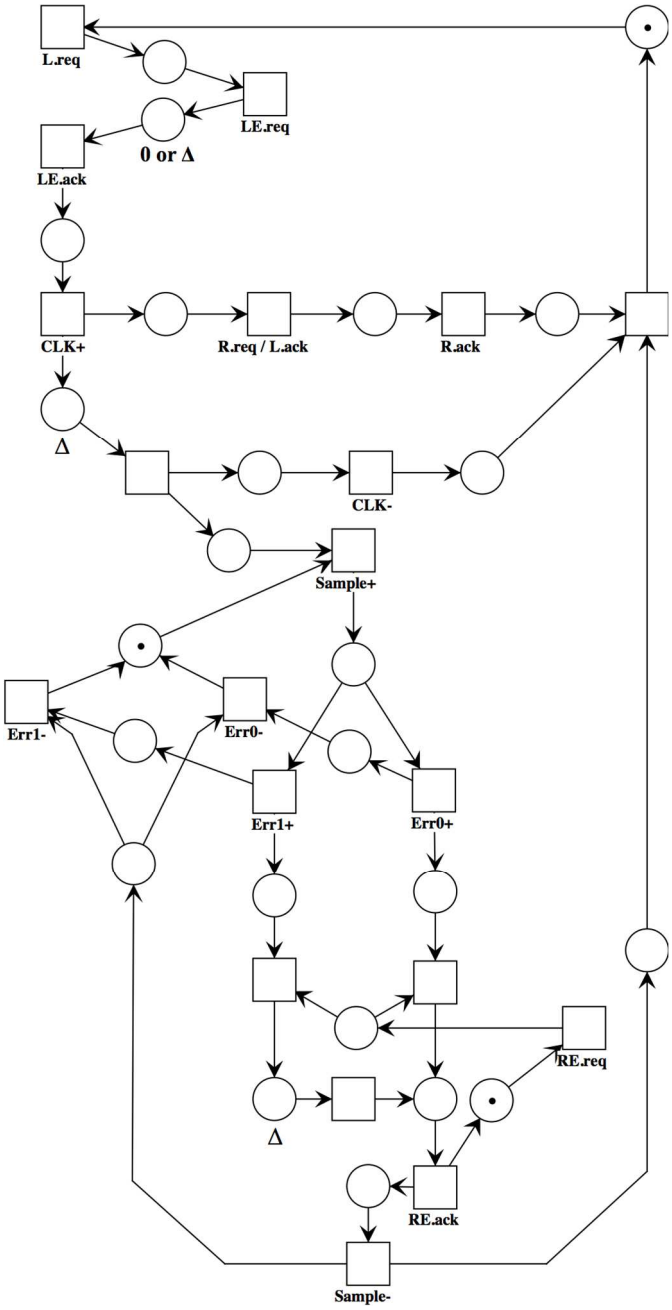

Figure 8. Petri Net description of the Blade-O controller

F. Petri Net of Blade-O Controller

The Blade-O controller can be automatically synthesized using a number of techniques (e.g., [30] [31] [32] [33]). One common method is to describe the controller as a Petri Net (PN), which can be formally analyzed for correctness and delay sensitivity. PNs can also be synthesized to library gates and C-Elements using well-known methods and tools, such as Petrify [31].

The PN in Figure 8 shows just one of many possible realizations of the Blade-O controller. Unlabeled transitions are internal states and shown only for completeness. Places with delay due to the Blade protocol are labeled with Δ, while unlabeled places have no additional delay. The place between LE.req and LE.ack is labeled *"0 or* Δ*"* to indicate the environment's variable delay of acknowledging the request on the extend channel, which is dependent on an error occurring in the previous stage, as described in Section III.C. This particular implementation uses a dual-rail *Err* input to the controller, which allows the stage to stall when resolving metastability as explained in Section III.B. We have implemented and tested this controller in behavioral Verilog to verify the protocol and analyze its performance. Synthesizing the Blade-O controller to gate-level PN is left for future work. In addition, the PN in Figure 8 can be adapted to describe the Blade-OC controller without much difficulty.

IV. Proposed Improvements

A. Reuse of Delay Lines

Delay lines are typically responsible for a significant portion of the area overhead associated with bundled-data designs. To mitigate this concern, we propose implementing only two tunable delay lines per stage, Δ and δ. A careful inspection of the Blade-O timing diagram will show that while there is a possibility of using a total delay of 2Δ per stage in the same cycle when an error occurs, each Δ delay occurs in series and does not overlap. Therefore, a single Δ delay line may be reused twice to implement the 2Δ delay.

Reusing delay lines in Blade-OC is not as straightforward. A single Δ delay line per controller may be reused twice: once to implement the initial resiliency window and a second time to delay the closing of the next stage. However, a second Δ delay line would be required to implement the Blade-O fallback feature to prevent the opening of the next stage's latch. With enough effort, a smaller delay line that is an integer factor of Δ may be used repeatedly to remove this extra delay line at the cost of a more complex controller. Furthermore, by making some reasonable timing assumptions, we may be able to remove this requirement in an entirely different way, as explained in the next section.

By the same argument, it may also be possible to re-use a single, short delay line in conjunction with a counter to repeatedly propagate signals through the delay line and create both the Δ and δ delays required in both Blade-O and Blade-OC. The tradeoff would once again be a more complex control

scheme versus increased area due to multiple delay lines. Depending on the implementation, fork-join structures between stages may also be more complicated to realize.

B. *Use local delay line*

The operation of Blade-O and Blade-OC as described Section III assumes that the Δ delay line used to detect an error is the same delay used to extend the next stage. This property allows for the value of Δ to vary amongst stages while maintaining correct operation in all cases. If we relax this requirement and assume all stages have roughly equal Δ delays, it then becomes possible to use the local delay line in each controller to implement the latch open/closing delays instead of relying on the delayed acknowledgement of the extend signal from the previous stage. For example, a Blade-OC stage could use its internal Δ delay once when an error is preventing the opening of its latch, again to regulate the width of the timing resiliency window, and finally to delay the closing of the latch when the previous stage detects an error. This optimization also allows for simplification of the extend channel, as it is no longer necessary to have a separate acknowledgement signal.

C. *Online Tuning of delay lines*

We envision each delay line (Δ and δ) to be post-silicon adjustable using well known techniques of building tunable delay lines in combination with at-speed or built-in test circuits [34] that monitor the collective or local error count in the system by recording the Err signal produced by the EDLs. Such a system could be activated infrequently to save power, only retuning the delay lines when it is suspected operating conditions have changed or at some predetermined regular rate. The simplest approach would be maintaining a difference in error count over a fixed period of time or number of asynchronous cycles. If the error rate exceeds the optimal threshold (as described in Section V), the delay between stages could be lengthened to ensure peak performance. Likewise, if the error rate is lower than expected, decreasing the delay of the system will also improve performance. Other implementations can be drawn from previous BIST and scan circuitry that have been extensively studied to implement delay testing in both synchronous [35] and asynchronous designs [36] [37].

D. *Alternative handshaking signals*

The proposed templates utilize two two-phase channels on each side of the controller for a total of 8 handshaking wires. Many physical implementations are possible while still implementing the same basic concept of the speculative handshaking protocol. For example, altering the protocol to use four-phase handshaking would be trivial.

Alternatively, the extend channel could also be flipped around to operate as a push channel and implemented using dual-rail signaling. Each stage would then send a “1” or “0” to its right neighbor stage to indicate when an error occurs. In this case, the subsequent controller must wait for a value to be received on the extend channel before making a decision to open the latch (Blade-O/OC) or close the latch (Blade-OC). In this particular implementation, the main channel’s acknowledgement signal could possibly be reused to acknowledge both the normal request and the extend channel’s data simultaneously, maintaining the same number of handshaking wires (8) as the proposed design.

Furthermore, the extend channel could even be implemented as a single wire with an associated timing assumption (the value of Extend must be stable at the next stage within δ time units), reducing the total number of wires from 8 to 6; however, ensuring this design meets all required timing assumptions can be difficult. It would also not be possible to allow extra time for metastability to resolve in the Q-Flop with a 1-bit extend signal.

v. Performance Model

A. *Systematic error rate*

The combination of δ and Δ should be set to ensure that all errors are caught. As in the analysis of bubble-razor, since normal/log-normal distributions do not have an upper limit, we set a rule that the combination of δ and Δ should satisfy the systematic error rate (SER). For example, in our results, we assume SER $\leq$ 0.1%.

In an N-stage ring using the Blade-O template with 2N latches, the equation that governs the SER is similar to that of bubble-razor:

$$\text{SER} = 1 - \left[P_R\left\{\delta + \Delta \leq \frac{C}{2}\right\}\right]^{2N} \leq 0.1\% \qquad (6)$$

In N-stage ring using the Blade-OC systems with 3N latches, the SER depends on a similar constraint

$$\text{SER} = 1 - \left[P_R\left\{\delta + \Delta \leq \frac{C}{3}\right\}\right]^{3N} \leq 0.1\% \qquad (7)$$

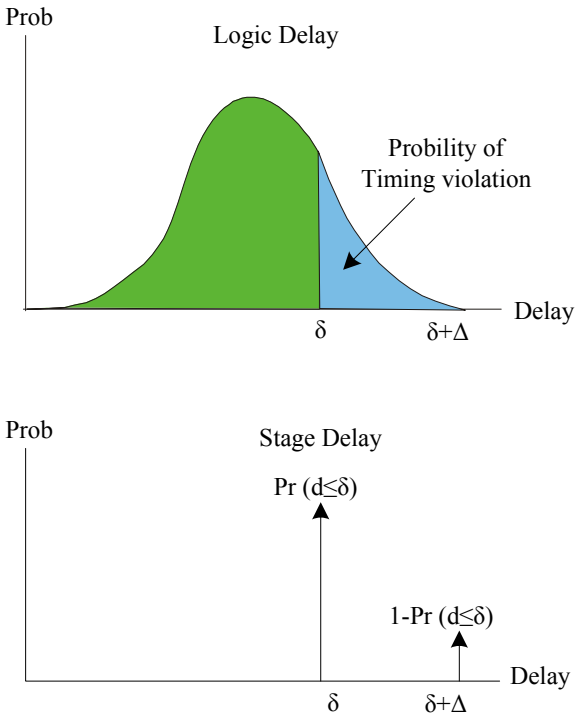

Figure 9. Two-valued discrete distribution of Blade

B. Optimal setting of the delay lines

A tradeoff in setting the delay lines emerges as decreasing δ allows the system to operate faster in the case of no errors, but the error rate increases so the subsequent pipeline stages spend more time being delayed by an amount Δ. To quantify this optimization problem, consider a delay distribution of a combination logic block between two latches as shown in Figure 9.

The area under the blue curve represents the probability that an error in a Blade system occurs at a previous output latch, referred to as p, such that the effective delay of this pipeline stage is δ+Δ. The area under the green curve is thus 1-p. We propose to model the performance of a pipeline stage as a discrete two-valued distribution as shown at the bottom of Figure 9.

It is important to notice that there is a fundamental tradeoff between the mean and the variance of these distributions. As described in Section II.D, because the weighted sum of δ and Δ should be set to achieve the desired SER, increasing δ causes the mean to increase and Δ to decrease. This decrease reduces the difference between the two discrete delay values and thus decreases the variance of the distribution. In complex systems the optimal setting of these delays is an interesting challenge, as latency-critical stages should be optimized to minimize average delay whereas non-latency-critical stages may be best set with a somewhat higher average delay that has lower variance.

Ignoring the interaction between different tokens in the ring, the optimal performance of an N-stage ring occurs when each latch stage's average-case delay is minimized. This can be formulated as

$$\text{Min}_{\delta}\ (\delta + p\ \Delta) \qquad (8)$$

Modeling the impact of the interaction between different tokens in the ring is left as future work.

Unfortunately, for normal and log-normal distributions this expression involves a transcendental equation for which we could find no closed form solution. However, for given parameters, numerical solutions are readily achievable.

VI. Performance Comparisons

This section compares the performance of Blade to both traditional synchronous and bubble-razor circuits. In particular, as in [17], we examine an N-stage ring. The traditional circuits have N flip-flops and bubble-razor and blade circuits have 2N or 3N latches.

All designs are set to have the same 0.1% SER. Moreover, without loss of generality we assume the nominal traditional clock cycle time is 1.0. To maintain a fair comparison, as in [17], the impact of the error/extend signal propagation delay on the error detection window is ignored. As mentioned above, we also assume the asynchronous ring is well pipelined such that the local handshaking and interaction of tokens in the Blade template is not a bottleneck.

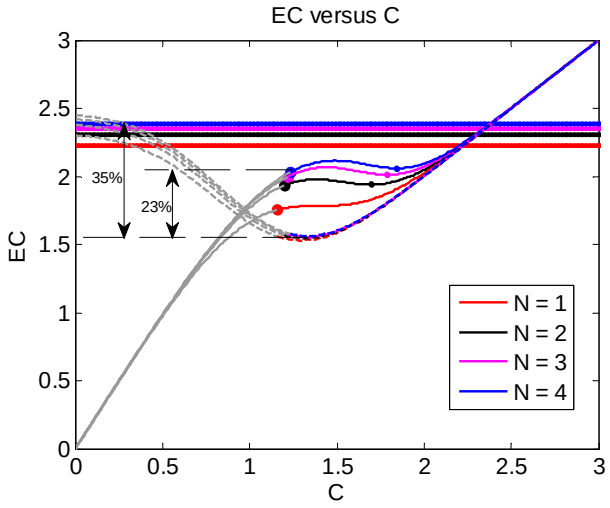

Figure 10. Blade-O comparisons with normally distributed delays

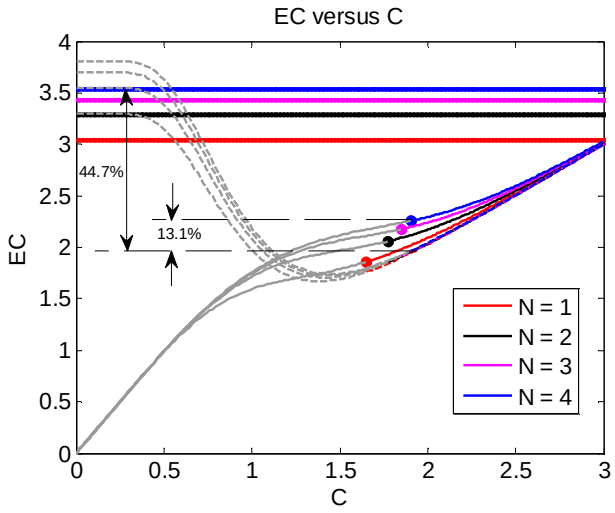

Figure 11. Blade-O comparisons with log-normally distributed delays

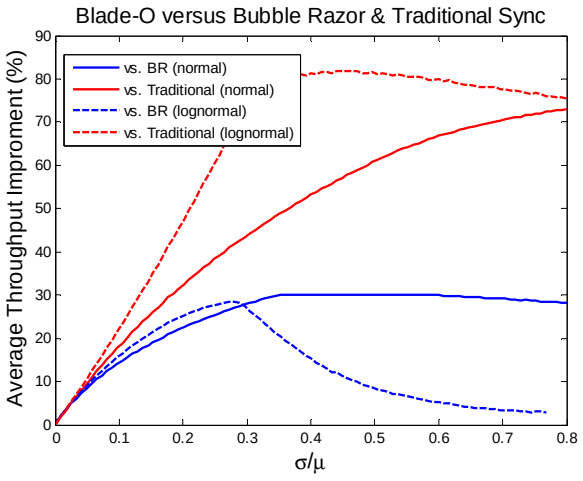

Figure 12. Blade-O performance improvement versus variability

A. *Blade-O comparisons*

Figure 10 and Figure 11 compare the Blade-O template with both traditional and BR when the underlying delays are normally and log-normally distributed with a moderate amount of variance ($\sigma/\mu = 0.2$) for rings of one to four traditional stages, i.e., N = 1 to 4. Curves corresponding to the same number of stages are drawn with the same color.

The horizontal lines represent the performance of traditional synchronous design restricted by the same SER. The solid curves represent the performance of bubble razor. Notice how the effective cycle time EC versus C curves have a slope of 2 for small C because when C is small timing violations always occur so that EC is twice as long as C. The curve is colored grey when the error rate exceeds that of the desired SER. The curve's slope is 1 for a very large C when the circuit is error-free. It is obvious that we should choose the point with the least EC while ensuring C meets the SER. These points are marked as large dots in Figure 10 and represent the operation points that achieve the best performance of the BR circuit for various ring lengths. The dashed somewhat v-shaped curves represent the performance of the proposed Blade template as a function of $C = 2\delta$. For example, when $\delta = 0$, each stage always delays the next stage and the cycle time is $\delta + \Delta$. The gray portion of the dashed line represents the portion of the line in which $\delta < \Delta$ that as described in Section V.B is not feasible. Figure 10 shows that for normal distributed delays at this variance the improvement Blade provides for a 4-state pipeline is 23% over bubble-razor and 35% over traditional synchronous designs. Figure 11 shows that for log-normally distributed delays with the same variance the improvement is 13.1% over bubble-razor and 44.7% over traditional synchronous design.

Figure 12 plots the performance improvement of Blade-O versus variance for a 4-stage (8-latch) ring. The chart shows that, compared to bubble razor, the potential performance improvement increases as the variance increases peaking at about 20% with σ/μ of about 25%. At this variance the δ is set such that the nominal delay is 65% of the worst-case delay. This is in contrast to bubble razor that is forced to have the nominal delay set to 50% of the worst-case delay. That is, at this point, the timing margin offered by bubble-razor is larger than it need be to achieve the SER. However, the performance improvement for log-normal distributions drops as variances further increase. This occurs because at such high-variances of the heavy tail of the log-normal distribution forces Blade-O to operate at its resilience limit, with its nominal delay δ set to 50% of the worst-case delay. In this region, Blade-O can offer no additional tolerance to variation over bubble razor and the improvement gains diminish.

B. *Blade-OC comparisons*

Comparisons of the Blade-OC templates show similar trends for moderate variations. Figure 13 and Figure 14 compare the Blade-OC template with both traditional and BR when the underlying delays are normally and log-normally distributed with a moderate amount of variance ($\sigma/\mu = 0.2$) for rings of one to four traditional stages, i.e., N = 1 to 4. Note here that the X-axis for Blade-OC is set such that $C = 3\delta$.Note that the systematic limits are higher than the Blade-O template

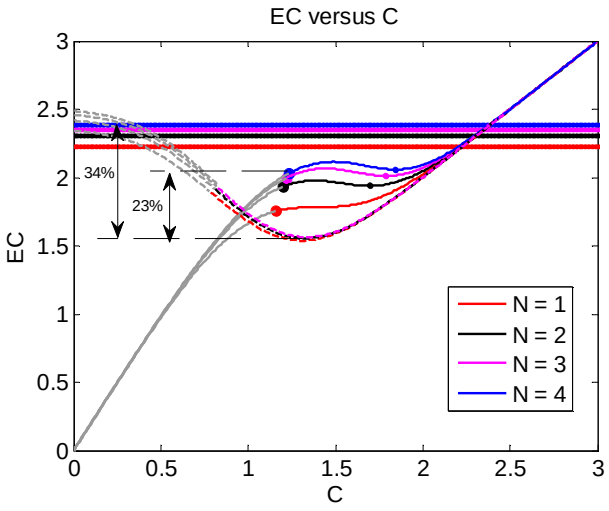

Figure 13. Blade-OC comparisons with normal distributions

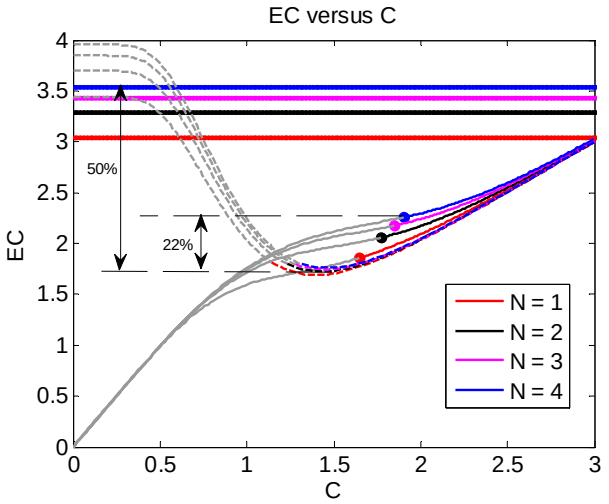

Figure 14. Blade-OC comparisons with log-normal distributions

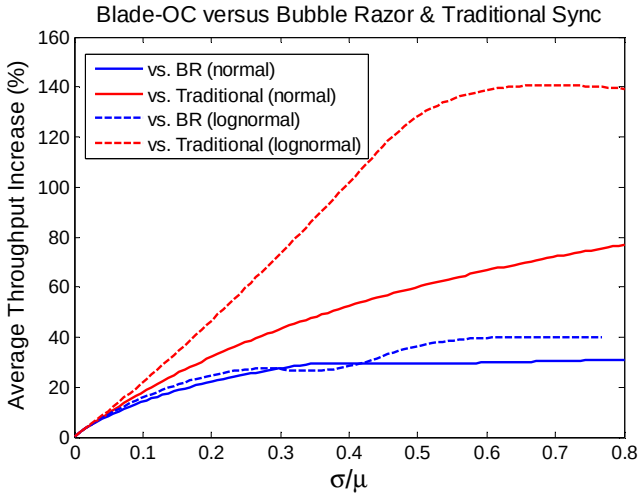

Figure 15. Blade-OC performance improvement versus variability

because of the presence of 3N latches instead of 2N. Also notice how the Blade curves change from grey to non-grey at a lower δ than in the Blade-O template. This is because this template allows for the nominal delay δ to be as little as $1/3^{rd}$ of the worst-case delay $\delta + \Delta$.

This extra flexibility helps when the design has high variability. To illustrate this, Figure 15 shows how the performance improvement of Blade-OC varies over a wide range of variances for a 4-stage (12-latch) ring. Unlike the Blade-O template, the performance improvement does not drop with higher variability. It is this region of the curve where the optimal nominal delay δ is less than ½ of the total delay $\delta + \Delta$.

VII. Conclusions and Future Work

This paper proposes two novel asynchronous design templates that use a bundled-data datapath with error-detecting latches as well as a new speculative handshaking paradigm that allows high average-case performance while maintaining low systematic error rates. The paper analytically compares the performance of this template with both traditional synchronous designs and the state-of-the-art synchronous resiliency strategy Bubble Razor for both normal and log-normally distributed delays. Our results demonstrate the potential benefit of our approach is significant.

Our future work includes gate-level designs of the proposed controllers and delay lines. Minimizing their complexity will be critical when trying to translate the performance improvements into power savings.

We believe that the benefits of the proposed template extend beyond that of the potential performance improvements shown in this paper. Namely, the observability that the proposed error-detecting latches provide offers several other advantages. First, the error signal not only can guide on-line tuning of the delay lines during test but also can enable on-the-fly re-configuration of the delay lines to account for temperate variations, aging, and/or changing workloads. Secondly, the increased observability and resilience makes the use of more complex delay lines whose delays depend on other data signals such as op-codes (e.g., Speculative Completion Sensing [38]) more practical, offering additional opportunities to optimize for the average case. Lastly, these templates inherently provide resilience to single event upsets in the datapath and assuming the control circuits can be efficiently designed to be single-event-upset tolerant, they can enable comprehensive resilience at a reasonable cost.

VIII. References

[1] T. Sato and Y. Kunitake, "A Simple Flip-Flop Circuit for Typical-Case Designs for DFM," in *8th International Symp. on Quality Electronic Design (ISQED)*, 2007.

[2] 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 pipelined based on circuit-level timing speculation," in *Proc. 36th IEEE/ACM Int'l. Symp. on Microarchitecture (Micro-36)*, 2003.

[3] S. Kim, I. Kwon, D. Fick, M. Kim, Y.-P. Chen and D. Sylvester, "Razor-lite: A side-channel error-detection register for timing-margin recovery in 45nm SOI CMOS," in *ISSCC*, 2013.

[4] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw and D. Sylvester, "Bubble Razor: An Architecture-independent Approach to Timing-error Detection and Correction," in *ISSCC*, 2012.

[5] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw and D. Sylvester, "Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 66-81, 2013.

[6] I. J. Chang, S. P. Park and K. Roy, "Exploring Asynchronous Design Techniques for Process-Tolerant and Energy-Efficient Subthreshold Operation," *IEEE Journal of Solid State Circuits*, vol. 45, pp. 401-410, 2010.

[7] C. Junchao, C. Kwen-Siong, G. Bah-Hwee and J. S. Chang, "An Ultra-low Power Asynchronous Quasi-delay-insensitive (QDI) Sub-threshold Memory with Bit-interleaving and Completion Detection," in *8th IEEE Int'l. NEWCAS Conference (NEWCAS)*, 2010.

[8] C. I. Joon, P. S. Phill and K. Roy, "Exploring Asynchronous Design Techniques for Process-Tolerant and Energy-Efficient Subthreshold Operation," *IEEE Journal of Solid-State Circuits*, vol. 45, pp. 401-410, 2010.

[9] P. A. Beerel, R. O. Ozdag and M. Ferretti, A Designer's Guide to Asynchronous VLSI, Cambridge University Press, 2010.

[10] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720-738, 1989.

[11] J. Cortadella, A. Kondratyev, L. Lavagno and C. P. Sotiriou, "Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, p. and Systems, 2006.

[12] N. Jayakumar, R. Garg, B. Gamache and S. P. Khatri, "A PLA-based Asynchronous Micropipelining Approach for Subthreshold Circuit Design," in *43rd ACM/IEEE Design Automation Conf.*, 2006.

[13] J. Liu, S. Nowick and M. Seok, "Soft MOUSETRAP: A Bundled-Data Asynchronous Pipeline Scheme Tolerant to Random Variations at Ultra-Low Supply Voltages," in *IEEE International Symp. on Asynchronous Circuits and Systems (ASYNC)*, 2013.

[14] O. C. Akgun, Y. Leblebici and E. A. Vittoz, "Current Sensing Completion Detection for Subthreshold Asynchronous Circuits," in *18th European Conf. on Circuit Theory and Design (ECCTD)*, 2007.

[15] L. Tsung-Te, L. P. Alarcon, M. D. Pierson and J. M. Rabaey, "Asynchronous Computing in Sense Amplifier-Based Pass Transistor Logic," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 2009.

[16] C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull and D. Blaauw, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 32-48, 2009.

[17] G. Zhang and P. A. Beerel, "Stochastic Analysis of Bubble Razor," in *Design, Automation & Test in Europe Conf.*, 2014.

[18] B. Zhai, S. Hanson, D. Blaauw and D. Sylvester, "Analysis and Mitigation of Variability in Subthreshold Design," in *ISLPED*, 2005.

[19] A. Chandrakasan and J. Kwong, "Variation-driven Device Sizing for Minimum Energy Sub-threshold Circuits," in *ISLPED*, 2006.

[20] T.-T. Liu and J. Rabaey, "Statistical Analysis and Optimization of Asynchronous Digital Circuits," in *18th IEEE International Symp. on Asynchronous Circuits and Systems (ASYNC)*, 2012.

[21] Y. S. Schwartz and S. C. Yeh, "On the Distribution Function and Moments of Power Sums with Log-normal Components," *Bell Syst. Tech. J., vol. 61, no. 7,* 1982.

[22] A. J. Drake, A. Kleinosowski and A. K. Martin, "A Self-Correcting Soft Error Tolerant Flop-Flop," in *12th NASA Symp. on VLSI Design, Coeur d'Alene*, 2005.

[23] A. Jagirdar, R. Oliveira and T. J. Chakraborty, "Efficient Flip-Flop Designs for SET/SEU Mitigation with Tolerance to Crosstalk Induced Signal Delays," in *IEEE Workshop Silicon Errors Logic System Effects*, 2007.

[24] R. Naseer and J.Draper, "The DF-dice storage element for immunity to soft errors," in *48th Midwest Symp. on Circuits and Systems*, 2005.

[25] M. Choudhury, V. Chandra, K. Mohanram and R. Aitken, "TIMBER: Time borrowing and error relaying for online timing error resilience," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2010.

[26] K. Bowman, J. Tschanz, N. S. Kim, J. Lee, C. Wilkerson, S. Lu, T. Karnik and V. De, "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance," *IEEE Journal of Solid State Circuits*, vol. 44, no. 1, pp. 49-63, 2009.

[27] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull and D. Blaauw, "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance," *IEEE Journal of Solid State Circuits*, vol. 44, no. 1, pp. 32-48, 2009.

[28] F. U. RosenBerger, C. E. Molnar, T. J. Chaney and T.-P. Fang, "Q-Modules: Internally Clocked Delay Insensitive Modules," *IEEE Transactions on Computers*, vol. 37, no. 9, pp. 1005-1018, 1988.

[29] M. Singh and S. Nowick, "MOUSETRAP: Ultra-high-speed Transition-signaling Asynchronous Pipelines," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 684-698, 2007.

[30] P. Beerel, C. Myers and T. Meng, "Covering Conditions and Algorithms for the Synthesis of Speed-Independent Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 3, pp. 205-219, 1998.

[31] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, "Methodology and Tools For State Encoding In Asynchronous Circuit Synthesis," in *Design Automation Conference*, 1996.

[32] R. M. Fuhrer, S. M. Nowick and M. Theobald, "Minimalist: An Environment for the Synthesis, Verification, and Testability of Burst-Mode Asynchronous Machines," CUCS-020-99, Columbia University, NY, 1999.

[33] K. Y. Yun and D. L. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits: Part II (Automatic Synthesis)," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 2, pp. 118-132, 1999.

[34] M. Abramovici, M. A. Breuer and A. D. Friedman, Digital Systems Testing and Testable Design, Wiley, 1994.

[35] B. I. Dervisoglu and G. E. Stong, "Design for Testability using Scanpath Techniques for Path-Delay Test and Measurement," in *Int'l. Test Conf.*, 1991.

[36] O. A. Petlin and S. B. Furber, "Built-in Self-testing of Micropipelines," in *Third International Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1997.

[37] M. Krstic and E. Grass, "BIST Technique for GALS systems," in *8th Euromicro Conf. on Digital System Design*, 2005.

[38] S. M. Nowick, "Design of a Low-latency Asynchronous Adder using Speculative Completion," *IEE Proceedings*, vol. 143, no. 5, pp. 301-307, 1996.