2015 21st IEEE International Symposium on Asynchronous Circuits and Systems

Performance Optimization and Analysis of Blade Designs under Delay Variability

Dylan Hand*, Hsin-Ho Huang*, Benmao Cheng‡, Yang Zhang*, Matheus Trevisan Moreira*†, Melvin Breuer*, Ney Laert Vilar Calazans†, and Peter A. Beerel*§

Ming Hsieh Dept. of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
†Faculty of Informatics, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil
‡Dept. of Automation, Tsinghua University, Beijing, China

Abstract—**As manufacturing processes continue to shrink and supply voltages drop, timing margins due to increased process, temperature, and voltage variability become a significant portion of the clock period. An asynchronous bundled data resilient template called Blade has recently been proposed to curb these margins and thereby outperform synchronous alternatives. This paper proposes a model to analyze the performance of Blade designs and an approach to optimize it. We validate the model against gate-level simulations of a resilient 3-stage MIPS CPU implemented with Blade and use it to compare the optimal performance of Blade designs with synchronous alternatives. The results show that Blade offers up to 44% higher performance than traditional designs and 23% higher performance than Bubble Razor, the synchronous resiliency strategy with the highest reported performance.**

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 variations compounded with aging effects is causing progressively larger delay variations, requiring increased timing margins which consequently reduce performance and energy efficiencies. Various forms of asynchronous circuits have been identified as promising solutions to this problem, particularly in the near-threshold regime [1], [2]. Quasi-delay-insensitive (QDI) logic tracks variations by embedding a completion signal into the data representation at the cost of higher area [3] and switching activity [4]. Bundled-data (e.g., micropipelines [5]), on the other hand, uses matched delay lines to track the delay of combinational single-rail logic.

Bundled-data (BD) designs have similar switching activity as their synchronous counterparts because the combinational logic is unchanged and the total area is also similar because the area of the control circuits and delay lines is comparable to that of a clock tree [6]. However, one challenge in BD designs is that the delay line must be conservatively designed to be longer than the worst case delay of its corresponding logic under all possible process, voltage, and temperature (PVT) corners, and this can take away much of its advantages. Researchers have proposed to mitigate this problem by duplicating the BD delay lines [1], constraining the design to regular structures such as PLAs [7], and using soft latches [8].

§ Peter A. Beerel is also a Chief Scientist, Technology Development at Intel, Calabasas, CA 91302.

Meanwhile, several synchronous *resilient* design techniques have been proposed that mitigate the impact of delay variations (e.g., [9], [10]). These techniques relax timing constraints by allowing timing violations to occur using error detecting flip-flops and latches that trigger recovery mechanisms to correcting errors caused by such violations. More recently, an approach that combines the benefits of synchronous resiliency and BD design called Blade has been proposed, which uses single-rail logic, reconfigurable delay lines, and error detecting latches with asynchronous sampling circuitry that reliably handles timing violations even under the presence of metastability (MS) [11].

The performance of Blade designs depends on stochastic characteristics of the data and environmental delay variations, the time for MS in the control circuits to resolve, and the size of the timing resiliency window supported. In this paper, we show that while the time to resolve MS may instantaneously slow down the circuit, the long term impact on average performance is likely quite low and can thus be ignored when optimizing Blade designs. We then provide an analytical model of Blade designs that enables us to optimally set the timing resiliency window for an expected degree of data and environmental variations using various types of distributions. An interesting side-result of our analysis is that while the optimal timing resiliency window is a function of the degree of variability, the optimal probability of error for normally distributed delays is proven to be constant. We then compare the optimal performance of Blade designs with synchronous alternatives, including using traditional flip-flop-based design as well as using Bubble Razor, the synchronous resiliency strategy with the lowest reported error penalty. The results show that Blade offers up to 44% higher performance than traditional designs and 23% performance over Bubble Razor. Finally, to apply our performance model to a real design, we compare the predicted performance with gate-level simulations on an OpenCore MIPS CPU, Plasma, and show that our model is typically accurate to within 5.4%.

The remainder of this paper is organized as follows. Section II provides relevant background on the Blade template, defines some commonly used delay distributions, and defines a notion of systematic error rate for unbounded distributions. Section III introduces the proposed Blade performance model and looks at its application to normal and log-normal distributions. Section IV documents non-ideal effects that may impact our

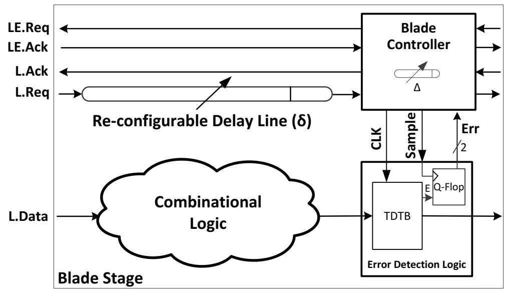


Fig. 1: Blade template

performance analysis. Section V quantifies Blade's benefits over both traditional and resilient, Bubble Razor, synchronous design approaches on abstract N-stage ring structures using both analytical models and a simulation framework that implements the Blade handshaking protocols, proving the accuracy of the proposed model. Section VI details how the proposed model can be applied to a real example, a MIPS OpenCore CPU, which has been implemented using Blade. We compare the analytical model to gate-level simulation results and show how the timing resiliency window can be optimized to achieve optimal performance. Finally, Section VII summarizes our results and outlines future work.

II. Background

A. Blade

1) Template overview: As shown in Figure 1, pipeline stages in Blade use single-rail logic followed by Transition Detecting Time Borrowing (TDTB) latches [12], Q-Flops [13], and two reconfigurable delay lines. The stage to stage delay line is of duration δ and controls when the TDTB first samples the data at the output of the combinational logic, assuming no timing violation had occurred in the previous stage. The second delay line is of duration Δ and defines a time window during which timing errors are allowed, referred to as the timing resiliency window (TRW). For the purposes of this paper, $\Delta = TRW$ and either may be used interchangeably.

If the outputs of the combinational logic change during the TRW, the TDTB flags a timing violation by raising its *E* signal. The TDTB prevents the introduction of metastable signals to the datapath, moving them to the control path, where they are filtered out by the Q-Flop, as will be described in detail in Section II-A3.

2) Speculative Handshaking: Each stage has four asynchronous channels that operate using a two-phase protocol. The first channel, *L*, is a typical bundled-data push channel, comprised of req, ack, and data. The second channel, *LE*, is a pull channel used to check if the previous stage suffered an error. It, too, has a req and ack, but no data value is required, because it is solely for control purposes. Two additional channels, *R* and *RE*, are tied to *L* and *LE* of the next stage.

Upon the opening of the latch, the Blade controller asserts its *R.Req* signal to its right neighbor speculatively, i.e. it sends the request signal before it can be determined if a timing violation occurred. Once the request is received by the right neighbor's controller (δ time units later), that controller will then send a signal back to the current stage on *RE.Req* to check if a violation has occurred. Because $\delta \geq \Delta$ in all stages, the error signal will have been properly sampled in the current stage's controller by this time. The next stage will then either be allowed to accept new data or be forced to wait a period of Δ to ensure the timing violation is resolved.

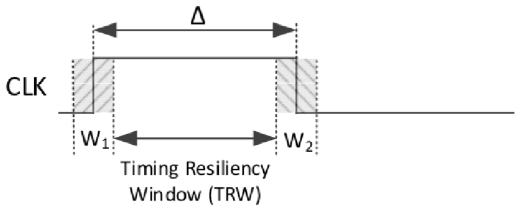


Fig. 2: Areas where metastability may occur in Blade

3) Metastability: Because the input data may stabilize sometime after the opening of the main latch, MS can exist and must be considered in Blade. Figure 2 shows the two main regions where the template is susceptible to MS. Timing violations in W_2 indicate the datapath is so slow that it exceeds the timing resiliency window and such circuits should be filtered out during post-fabrication testing. Therefore, it is only necessary to consider W_1. In this region, a data transition occurring near the rising edge of CLK can put the TDTB latch of the error detection logic in MS. For this reason the output of this latch is sampled by a Q-Flop, which is used to generate the *Err* signal. Similar to an arbiter, the Q-Flop uses a MS filter to resolve MS in an indeterminable amount of time. Its output is dual-rail, so the Blade controller will stall the stage until a strong '1' or '0' is present, preventing the MS state from propagating through the control logic. During the W_1 region, the value at which the Q-Flop resolves is only important from a performance perspective; it will not affect correctness. If the datapath transition that caused MS passes through a non-critical path in the next stage, there is no error and correction is not required for the next latch to correctly sample the value. Even if it passes through a critical path, the transition will appear at the next stage's TDTB while it is transparent, be flagged as a timing violation, and subsequently be corrected at the next stage. Lastly, small glitches at the output of the datapath during the TRW may also cause MS in the TDTB's *E* output, but not its data output [14]. The impact of these glitches can be further minimized by careful full-custom design of the TDTB [11]. Consequently, this MS scenario only effects performance, not correctness, and its performance impact is likely overshadowed by the impact of MS in W_1. Thus, for simplicity, it is not considered here.

B. Delay distributions

Delay variations in the datapath can be attributed to three main sources: global variation, local variations, and data dependency. It is common to model random local and global variations in circuits using normal distributions. However, it has been shown that heavy tail distributions, such as log-normal, are more suitable in near-threshold domains [15], [16]. Therefore, we analyze both normal and log-normal distributions with the proposed performance model. Data dependency, on the other hand, cannot be as well defined; it is determined by many factors, including architectural description, logic synthesis, and input data.

To simplify the analysis and abstract the various sources of variation, it is desirable to consider a single delay distribution. According to [16], [17], it is reasonable to represent the sum of two normal or log-normal random variables as another normal or log-normal random variable, respectively. In this way, the analyses presented in this paper are based on combined distributions with a σ/μ that can be considered to encompass all sources of variation.

C. Systematic error rate

In both the normal and log-normal distributions, there is a non-zero probability of experiencing an infinitely large delay value, i.e. it is impossible to set a traditional clock cycle time that would catch all variations with 100% probablility. Therefore, a notion of Systematic Error Rate (ξ) must be introduced to define an upper bound on the worst case performance of the circuit. ξ sets an acceptable amount of errors that may be allowed during operation of the circuit, which is typically a very small value, e.g. in [18] the authors assume $\xi \leq 0.1\%$. For traditional circuits, ξ is calculated as:

$$\xi = 1 - [P_R\{D \leq C\}]^N \tag{1}$$

where D is a random variable representing the delay of the worst case path between two sequential elements, C is the clock period, $P_R(x)$ is defined to be the probability of event x occuring, and N is the number of stages in the circuit.

III. Optimal Blade Performance

A. Performance model

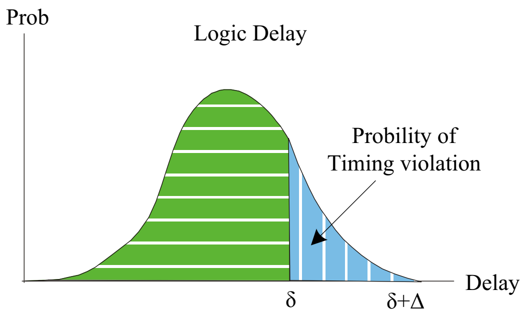


Fig. 3: Logic delay distribution model

There are two main timing parameters of Blade: the δ and Δ delay lines, where Δ sets the length of the TRW. Compared to a traditional synchronous circuit with clock period C, we can set $C = \delta + \Delta$. Therefore, a trade off in setting these values emerges as decreasing δ allows the system to operate faster if no timing violations (errors) occur; however, the shorter stage-to-stage delay means that more transitions will occur while the latch is transparent, thereby increasing the frequency of errors that force subsequent pipeline stages to be delayed by the now larger Δ value, as C remains constant. To quantify this optimization problem, consider a delay distribution of a combinational logic block between two latches as shown in Figure 3. The area of the blue vertically-lined region represents the probability that an error occurs at a previous output latch, defined as p, such that the effective delay of this pipeline stage is $\delta + \Delta$. The area of the green horizontally-lined region is thus $1-p$. We propose to model the performance of a pipeline stage as a discrete two-valued distribution, which yields the following equation for average delay of a Blade stage:

$$\bar{d} = \delta + p \cdot \Delta \tag{2}$$

The optimal performance of simple structures, such as N-stage rings, occurs when each stage's average-case delay is minimized, i.e. when $\bar{d} = \bar{d}_{min}$. Furthermore, in practice this equals the *effective cycle* time (EC) of the design, as introduced in [18]. In this way, the asynchronous and synchronous implementations can be compared directly by their ECs, where $EC = C$ for traditional synchronous designs.

B. Normal and log-normal distributions

We explore normal and log-normal distributions to analyze the impact on performance due to differences in distributions. Both are defined by two parameters: μ, the mean; and σ, the standard deviation. To generalize our analysis for any possible delay values, we define a distribution by its σ/μ ratio instead of these individual components. The probability of an error occurring depends on the chosen TRW and the expected ξ. Setting ξ according to Section II-C, we can then sweep the TRW from 0 to δ to plot the EC using (2). The results for both normal and log-normal distributions at three different σ/μ values are showin in Figure 4. In Blade, the TRW is limited to $\frac{\delta}{2}$; however, the proposed model can obtain the expected performance at all TRWs. This difference is denoted by the colored lines turning gray at the maximum TRW. The optimal EC is obtained by finding the minimum of these curves; from that, the optimal error rate, p_{opt}, can be trivially derived. By varying the σ/μ ratio and recomputing the minimum EC and p_{opt}, the relationship between variation and optimal error rate can be plotted, as shown in Figure 5. Interestingly, for normal distribution p_{opt} appears to be constant as variation increases. In fact, p_{opt} of normal distribution is independent of both σ and μ, and a proof is provided in the appendix.

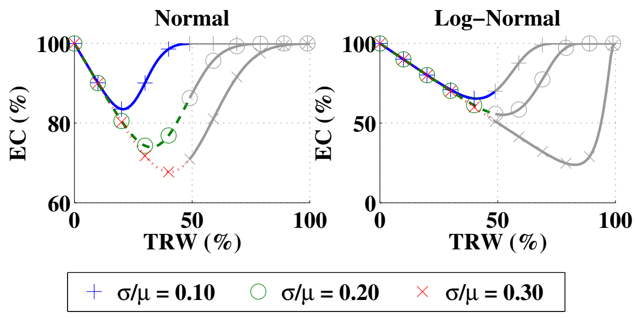


Fig. 4: Normalized expected cycle time versus size of timing resiliency window for Normal and Log-Normal Distributions

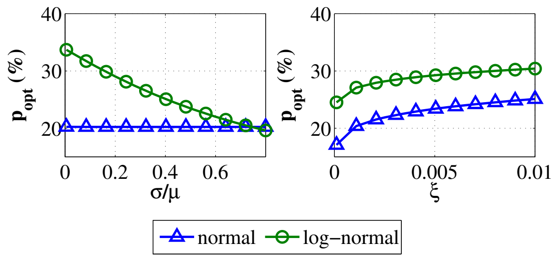


Fig. 5: p_{opt} versus variation and systematic error rate

This means that, given solely an expected ξ and knowledge that the distribution of delays resembles a normal curve, the optimal performance can be obtained by tuning the circuit to always achieve a pre-defined error rate.

IV. Performance Impact of Non-Ideal Effects

A. Robustness to delay line accuracy

In Blade, the δ and Δ delays are typically implemented using simple delay lines comprised of inverters or buffers, which imposes a limit to the accuracy of the delay line. In other words, the total delay of the delay line may be up to one gate delay off from the ideal δ value. Even if the delay lines are tunable, there will still be a quantization of the delay line such that the ideal delay is unobtainable. To quantify the impact, the variation in δ versus the resulting variation in EC is plotted in Figure 6. For a 10% variation in δ, we only see a 6.3% to 4.7% change in performance for normally and log-normally distributed delays, respectively. At 30% variation, the impact drops to 2.3% and 1.3%, respectively.

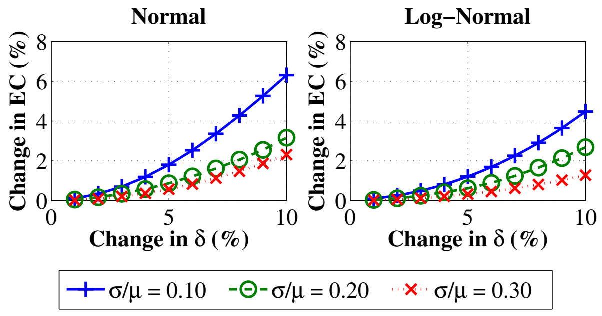


Fig. 6: Effect of delay line quantization on the Expected Cycle time for normal and log-normal distributions with σ/μ = 0.1, 0.2, and 0.3

B. Metastability

To analyze the impact of metastability on performance we analyze all possible scenarios, as illustrated in Figure 7, and create a weighted sum of expected stage delays based on the probability that each scenario will occur. We define an event, met, in which MS has occurred in the error detection logic, and thus the probability of this event as $P_R(met)$. Accordingly, the probability that MS does not occur is then $1 - P_R(met)$.

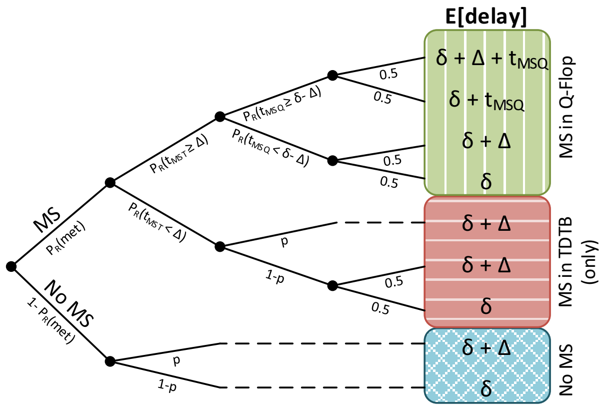


Fig. 7: Expected stage delays including metastability

1) Metastability scenarios: We define an expected delay associated with each of the nine scenarios. The expected delays of the two MS-free scenarios, highlighted in checkered blue, are easily obtained based on the analysis in Section III. The remaining scenarios are divided into two categories: MS occurs in the TDTB's E only and MS occurs in both the TDTB and Q-Flop. When MS occurs in the TDTB but resolves before the Q-Flop samples its output at time Δ, it should be noted that it is impossible to know whether MS resolved randomly or due to another datapath transition arriving at the TDTB's D input that set the E output to '1'. Therefore, three separate conditions, shown in the red horizontally lined region of Figure 7, should be evaluated: i) a new timing violation occurred with probability p; ii) no violation occurred but MS randomly resolved to '1' with 0.5 probability; or iii) no violation occurred and MS resolved to '0' with 0.5 probability. In the first and second conditions, the total stage delay will be $\delta + \Delta$, while the last condition has expected delay of δ.

If MS in the TDTB lasts longer than Δ, then the Q-Flop will sample the unknown value and become metastable itself. However, a stable output from the Q-Flop is not required until the *R.Req* signal propagates through the δ delay line and the next stage issues a request on its *LE* channel, as explained in Section II-A2. This allows up to $\delta - \Delta$ for MS in the Q-Flop to resolve before impacting the performance, shown in the green vertically lined region. Only when MS propagates from the TDTB to the Q-Flop and persists longer than $\delta - \Delta$ does the time to resolve, t_{MSQ}, appear in the expected delay value, shown in the purple region.

2) Analytical model: As demonstrated in Section II-A3, a transition in the datapath must occur during the W_1 time window to induce MS in the error detection logic. Therefore, we can define the probability of event met based on a normal distribution as:

$$P_R(met) = \int_{\delta-\frac{W_1}{2}}^{\delta+\frac{W_1}{2}} N(x,\mu,\sigma^2)dx \tag{3}$$

To analyze the individual components of this probability, we must define the probability that MS does not resolve in a certain amount of time. As shown in [19], this can be defined using two parameters: t_r, the time to resolve MS; and λ, a time constant that is derived from simulation of the circuit experiencing MS. Accordingly, we use t_{MST} and t_{MSQ} as the time to resolve MS in the TDTB and Q-Flop, respectively, and λ_T and λ_Q as the time constants, respectively. As an example, the probability that MS lasts longer than a time T in the TDTB conditioned on event met occurring is given by:

$$P_R(t_{MST} \geq T|met) = e^{-\lambda_C T} \tag{4}$$

Using the same form as (4), the probabilities of each of the branches shown in Figure 7 can be derived in a similar fashion. To simplify our results we set the time constants for the C-element and Q-Flop to be equal, i.e. $\lambda_T = \lambda_Q = \lambda$.

Taking all conditions into consideration and assuming delays are normally distributed, the expected delay per stage can then be calculated as:

$$E[delay] = (ab+1)\cdot\delta + a\Big[\frac{1-p(c-\frac{2}{a})}{2} - b\Big]\cdot\Delta + \frac{ab}{\lambda} \tag{5}$$

where

$$a = Q\Big(\frac{\delta - \frac{W_1}{2} - \mu}{\sigma}\Big) - Q\Big(\frac{\delta + \frac{W_1}{2} - \mu}{\sigma}\Big) \tag{6}$$

$$b = e^{-\lambda\delta} \tag{7}$$

$$c = e^{-\lambda\Delta} \tag{8}$$

The Q function in (6) is a well-known equation that computes the area under the tail of a normal distribution for a given value in the distribution. The difference between two Q functions is therefore the probability landing in the interval of the two parameters, in our case between $\delta \pm \frac{W_1}{2}$. To quantify the impact of MS, we look at the throughput ratio, defined as the expected delay with MS (5) divided by the nominal delay (2) versus variation. Here we set $\mu = 1$ and δ, p, and Δ according to the analysis presented in Section III. The time constant λ and MS window W_1 can be derived from either SPICE simulation or more accurately using a physical circuit, as shown in [20], where the authors obtained λ = 3 and W_1 = 0.07 using an older process. As an example, using these values we can compute that the expected impact on throughput for normally distributed data delays with σ/μ of 0.1, 0.2, and 0.3 is 1.5%, 1.1%, and 0.9%, respectively. In addition, modern processes will tend to feature a larger λ, smaller W_1, and greater variation due to PVT and unbalanced propagation delays, further reducing the performance impact of MS [21]. In other words, we conclude that it is reasonable to use (2) directly to model performance because the impact on stage delay due to MS is exceedingly small.

V. Performance comparison of N-stage rings

In this section, the performance improvements of Blade are compared to those of a synchronous resilient architecture, Bubble Razor (BR) [18], using generic N-stage rings. Both results are compared to the worst-case clock period of the traditional synchronous design to obtain a precise comparison between the two resiliency styles. The accuracy of the proposed model is also compared to performance results using a behavioral Verilog implementation of the Blade controller.

A. *Performance model for Bubble Razor*

Performance models for most synchronous resilient architectures are not readily available; fortunately, one exists for Bubble Razor [18]. BR is another architecturally independent resiliency scheme, in which a traditional N-stage synchronous design is converted into a 2N-stage retimed, latch-based design and augmented with error detection/correction control circuitry. While BR has been shown to be susceptible to MS [22], it can be considered as an upper bound on performance of synchronous resilient architectures due to its low, one-cycle penalty for recovering from errors. BR's EC for ring structures is derived through Markov Chain analysis by [18] as:

$$EC = C[2 - (1-p)^{2N}] \tag{9}$$

Implicit in this equation is that as the synchronous clock cycle C decreases, the probability of an error p increases, presenting a tradeoff whose optimal setting yields an optimal effective clock cycle time EC_{opt}.

B. *Model accuracy*

To verify the correctness of our performance model for Blade, a basic Blade controller was implemented based on [11] and a generic simulation environment was created using behavioral Verilog blocks to create N stage rings. Note that as in the BR design, an N-stage ring is implemented with 2N latch stages. By abstracting away the combinational logic, complex circuits can be modeled using simple structures with no loss of exactness as all delays are modeled probabilistically. In addition, the simulation framework accurately models the handshaking overhead and interaction between tokens in the control path, enabling the framework to capture the impact of overheads should they become sufficiently large as to affect performance.

Figure 8 shows performance results obtained using the analytical model presented in Section III and the simulation framework on a ring structure with a clock period (*CP*) of 2.5. Note that we plot *EC* versus *C*, where the clock period *C* for Blade is really $CP - TRW$, to match the notation used in [18]. In the region of operation where $\delta \geq \Delta$, i.e. the TRW is less than 50% of the original clock cycle, or 1.25, there is no appreciable difference between the two models. Outside of this region, when $\delta \leq \Delta$, the proposed model is showing

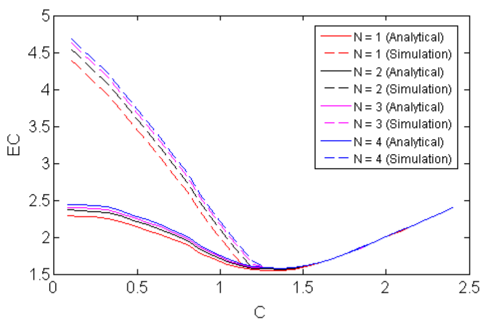
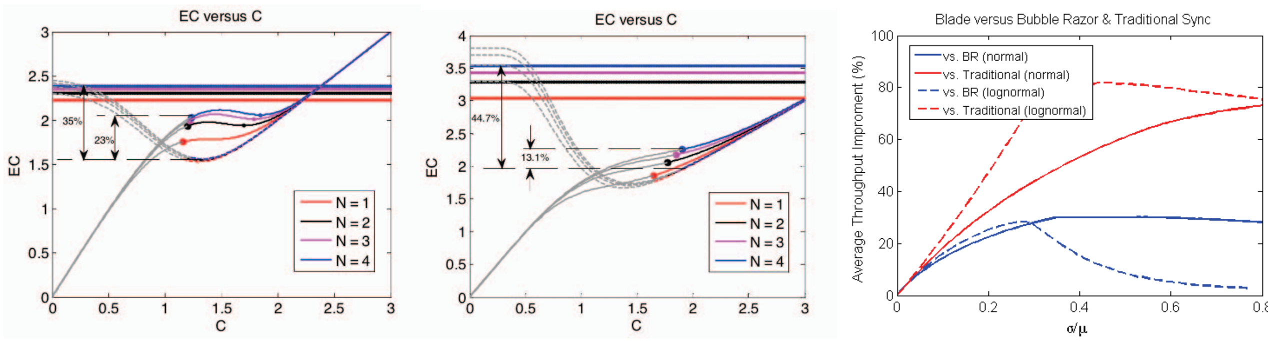


Fig. 8: Simulation Framework vs Performance Model for Ring



(a) (b) (c)

Fig. 9: Blade comparisons of (a) effective clock period with normally distributed delays, (b) effective clock period with log-normally distributed delays, and (c) performance improvements versus variability

the EC assuming the size of TRW can be further increased whereas the simulation framework is correctly modeling the Blade template slowing down the ring to ensure no errors occur outside of its normal region of operation.

C. Comparison results

We compare the performance improvements of both Bubble Razor and Blade over the traditional synchronous design on N-stage rings. To match the results presented in [18], ξ is fixed at 0.1%, $CP = 2.5$, and we sweep *C*, which is *CP* minus *TRW*, for all designs. Figure 9a and Figure 9b compare the EC obtained from each design when the underlying delays are normally and log-normally distributed, respectively, with a moderate amount of variance given both environmental [23] and data variability ($\sigma/\mu = 0.2$) 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 the traditional synchronous design and are at different heights because of the relationship between the clock period and the number of stages N in (1) when the SER is fixed. The solid curves represent the performance of BR, and the dotted curves shown results from Blade. The curve is colored gray when the maximum size of TRW for the Blade and BR designs is exceeded. In some cases, the size restriction on TRW occurs before the optimal EC is reached, therefore the optimal setting for TRW is $\Delta = \frac{CP}{2}$. For a 4-stage ring, Blade's performance improvement is 23% over BR and 35% over traditional synchronous designs assuming normally distributed delays. For log-normally distributed delays with the same variance, the improvement is 13.1% over BR and 44.7% over traditional synchronous design.

VI. Application to a MIPS CPU

To show an example of how the Blade performance model can be used, we present an application of the model on a 3-stage MIPS OpenCore CPU called Plasma [24]. We show that the optimal performance of the Blade design can be accurately predicted using only simulation data from the synchronous design. This allows designers to estimate the potential benefits of Blade without converting their existing designs and also provides insight in setting the timing resiliency window to achieve the optimal performance.

A. Delay distribution

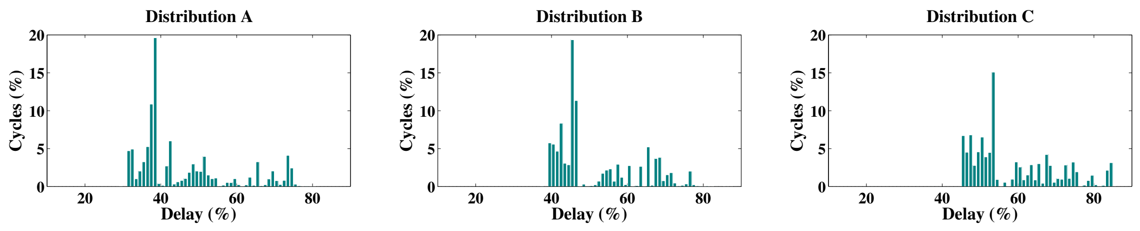
As discussed in Section II-B, normal and log-normal delays may not always be realistic when data dependency plays a large role in delay variability. To gain more insight into actual delay distributions and to apply our model directly to a real design, we analyze Plasma implemented using a standard synchronous flow in a 28nm process executing a benchmark program that computes π [24]. Delay values were recorded by altering the sequential Verilog modules in the library to record the difference in delay between the last transition at its input and the rising edge of clock, as shown in Figure 10a.

B. Impact of retiming and reducing error detection logic

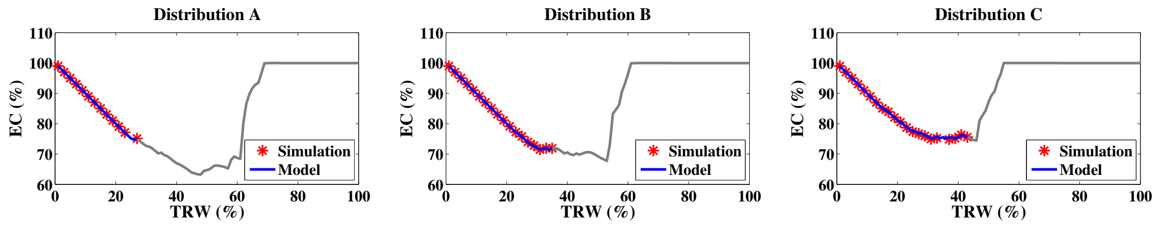
The conversion of a traditional synchronous FF-based design to Blade, which is latch-based, involves a retiming step where every flop in the original design is converted to two latches, only one of which is retimed to be placed near the middle of the original combinational logic. In this way, a critical path of length X will now ideally be split into two paths of length $X/2$ and $X/2$, enabling the hiding of Blade handshaking overheads. However, the path delays of the synchronous FF-based design cannot always be split equally; therefore, we collect data from the retimed latch-based synchronous design to obtain accurate distributions.

To minimize the area and power overheads of placing error detection logic on every retimed stage, it may also be desirable to apply Blade only to every other stage, i.e. at the original flop locations and not at the retimed latch locations. Non-error detecting stages do not monitor for timing violations; instead, they allow for time borrowing between the two neighboring stages. Fortunately, the proposed model also applies to such a design, with one caveat: the maximum TRW of the error detecting stage is now restricted by the degree of time borrowing allowed in the subsequent stage.

Determining the amount of time borrowing is therefore another avenue for optimization. To explore this avenue further, three distributions are shown in Figure 10a, each come from measuring the path delays of the synchronous latch-based



(a) Delay distribution histograms



(b) Effective cycle time versus timing resiliency window

Fig. 10: Experimental results of 3-stage MIPS CPU

design retimed to allow a different amount of time borrowing. In particular, Distribution A features the largest amount of time borrowing, while Distribution C allows the smallest amount of time borrowing. The datapath between these runs is only minimally changing (enough to accommodate the retiming algorithm); thus the differences are solely based on the final placement of the latches. Fortunately, our model easily handles differences in variation and thus we can obtain the EC vs TRW for each run, as shown in Figure 10b.

C. Optimizing TRW

Table I summarizes the optimal TRW, TRW_{opt}, and corresponding optimal EC, EC_{opt}, derived using the analytical model and actual Blade gate-level simulation normalized to the original synchronous clock period. The proposed model is within 5.4% of the simulated TRW_{opt} and more than 99% accurate at estimating EC_{opt}. Note that the potential performance benefit for these particular distributions are always limited by TRW_{max}, as set by the Blade template and chosen time borrowing allocation. The proposed model allows us to compute the truly optimal TRW, assuming no TRW_{max} limit exists. This region is shown by the extended gray curves in Figure 10b. For distributions A, B, and C, the optimal TRW sizes are then 0.48, 0.53, and 0.46, respectively, which achieve an EC of 0.632, 0.678, and, 0.745, respectively.

TABLE I: Normalized optimal TRW and EC of three data dependent distributions

	Distribution A		Distribution B		Distribution C	
	Model	Sim	Model	Sim	Model	Sim
TRW_{max}	0.27		0.35		0.43	
TRW_{opt}	0.26	0.27	0.34	0.35	0.39	0.37
EC$_{opt}$	0.748	0.751	0.713	0.719	0.751	0.748

VII. Conclusions

In this paper, we proposed an analytical model to accurately predict the performance of Blade designs based on generic delay distributions, such as normal and log-normal, given a particular degree of variation. This model can be used to optimally tune the size of the timing resiliency window used in Blade designs to achieve maximum performance. We also showed how the model can be adapted to work on more realistic delay distributions measured through simulation, where the final distribution of path delays is an amalgamation of variations from multiple sources. The optimal probability of error when delays are normally distributed, which tends to be true when considering PVT variations, was analytically proven to be 21% for a systematic error rate set to 0.1%, irrespective of σ or μ. This observation may lead to optimized methods of tuning stage delays in Blade circuits, as an online tuning circuit could simply adjust to maintain a pre-defined error rate to ensure near-optimal performance instead of computing more complicated statistics and timing analysis on-the-fly.

Non-ideal effects on the performance of Blade were also studied, including the quantization of delay lines and metastability. Our results show that the step size of a tunable delay line can be reasonably large without experiencing a significant impact in performance, allowing the designer to reduce the complexity, power consumption, and area of these circuits. The impact of MS on the performance was also derived and analyzed, showing that, while MS may lead to long instantaneous stage delays, the occurrence of these events is too rare to make an appreciable impact over the long term.

We then verified our performance model through a simulation framework using behavioral Blade controllers. Using this framework, we also compared Blade to Bubble Razor and traditional synchronous design with normally and log-normally distributed delays. Our results show Blade on N-stage rings to be up to 23% better than Bubble Razor and

35% better than traditional synchronous design for normal distributions with a reasonable amount of variation. Finally, the proposed model was applied to an actual gate-level implementation of Blade in a MIPS CPU to analyze irregular delay distributions, determine the optimal timing resiliency window, and gain insights into the impact of retiming on performance.

Appendix

It is possible to analytically find the optimal error rate for the normal distribution. Assume the worst case delay per stage is always constant, $K = \delta + \Delta$, where K is set based on the chosen ξ using the following equation:

$$K = \mu + m \cdot \sigma \tag{10}$$

For $\xi = 0.1\%$, m can be calculated to be ~3.2905. Rearranging and plugging back into equation (2) yields:

$$\bar{d} = (1-p)\delta + p \cdot K \tag{11}$$

In a normal distribtion, $(1-p)$ is defined as $\frac{1}{2}[1 + erf(\frac{\delta-\mu}{\sqrt{2}\sigma})]$ or $erf\big((\delta-\mu)/(\sqrt{2}\sigma)\big) = 1-2p$, where $erf(x)$ is the normal error function. Taking the inverse error function of each side would therefore give:

$$\delta = \sqrt{2}\sigma erf^{-1}(1-2p) + \mu \tag{12}$$

Replacing (12) in (11) yields

$$\bar{d} = (1-p)[\sqrt{2}\sigma erf^{-1}(1-2p) + \mu] + p \cdot K \tag{13}$$

To minimize $\bar{d}$, we can take the derivative of (13) and set it to zero:

$$\begin{aligned}\frac{\partial \bar{d}}{\partial p} = K - &[\sqrt{2}\sigma erf^{-1}(1-2p) + \mu] \\ &+ (1-p)\sqrt{2}\sigma \frac{\partial erf^{-1}(1-2p)}{\partial p} = 0\end{aligned} \tag{14}$$

Letting $y = 1-2p$ can simplify (14):

$$(1+y)\sqrt{2}\sigma \frac{\partial erf^{-1}y}{\partial y} - \sqrt{2}\sigma erf^{-1}y + K - \mu = 0 \tag{15}$$

A simple rearrangement of (10) and (15) yields:

$$(1+y)\sqrt{2} \frac{\partial erf^{-1}y}{\partial y} - \sqrt{2} erf^{-1}y + m = 0 \tag{16}$$

Inspection of equation (16) and the definition of inverse error function from [25], shows that neither y nor p have any dependence on σ or μ. Therefore, $\bar{d}$ is independent of σ/μ but not ξ (note the m).

Acknowledgements

We would like to thank Jordi Cortadella and Luciano Lavagno for fruitful discussions. We would also like to acknowledge Fang Bai, Frederico Butzke, Danlei Chen, Matheus Gibiluka, Dailin Li, Zhichao Li, Zhe Liu, Lan Lu, and Guan Wang for their help in testing the Blade template and supporting tools. This research was partially funded by a gift from Qualcomm.

References

[1] I. J. Chang, S. P. Park, and K. Roy, "Exploring asynchronous design techniques for process-tolerant and energy-efficient subthreshold operation," *IEEE JSSC*, vol. 45, no. 2, pp. 401–410, Feb 2010.

[2] J. Chen, K.-S. Chong, B.-H. Gwee, and J. S. Chang, "An ultra-low power asynchronous quasi-delay-insensitive (QDI) sub-threshold memory with bit-interleaving and completion detection," in *IEEE NEWCAS*, June 2010, pp. 117–120.

[3] M. Ferretti, "Single-track Asynchronous Pipeline Template," Ph.D. dissertation, University of Southern California, 2004.

[4] A. Yakovlev, P. Vivet, and M. Renaudin, "Advances in asynchronous logic: From principles to GALS & NoC, recent industry applications, and commercial CAD tools," in *DATE*, 2013, pp. 1715–1724.

[5] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[6] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. on CAD*, vol. 25, no. 10, pp. 1904–1921, Oct 2006.

[7] N. Jayakuma, R. Garg, B. Gamache, and S. Khatri, "A PLA based asynchronous micropipelining approach for subthreshold circuit design," in *ACM/IEEE DAC*, 2006, pp. 419–424.

[8] 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 *ASYNC*, May 2013, pp. 1–7.

[9] 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," in *IEEE/ACM MICRO-36*, Dec 2003, pp. 7–18.

[10] 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 JSSC*, vol. 48, no. 1, pp. 66–81, Jan 2013.

[11] D. Hand, M. T. Moreira, H.-H. Huang, D. Chen, F. Butzke, Z. Li, M. Gibiluka, B. M., N. L. V. Calazans, and P. A. Beerel, "Blade - a timing violation resilient asynchronous template," in *ASYNC*, May 2015.

[12] 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 JSCC*, vol. 44, no. 1, pp. 49–63, Jan 2009.

[13] F. Rosenberger, C. Molnar, T. Chaney, and T.-P. Fang, "Q-modules: internally clocked delay-insensitive modules," *IEEE Trans. on Computers*, vol. 37, no. 9, pp. 1005–1018, Sep 1988.

[14] M. T. Moreira, D. Hand, N. L. V. Calazans, and P. A. Beerel, "Tdtb error detecting latches: Timing violation sensitivity analysis and optimization," in *Quality Electronic Design, 2015. ISQED '15. International Symposium on*, 2015.

[15] B. Zhai, S. Hanson, D. Blaauw, and D. Sylvester, "Analysis and mitigation of variability in subthreshold design," in *ISLPED*, Aug 2005, pp. 20–25.

[16] J. Kwong and A. Chandrakasan, "Variation-driven device sizing for minimum energy sub-threshold circuits," in *ISLPED*, Oct 2006, pp. 8–13.

[17] S. C. Schwartz and Y. S. Yeh, "On the distribution function and moments of power sums with log-normal components," *Bell System Technical Journal*, vol. 61, no. 7.

[18] G. Zhang and P. Beerel, "Stochastic analysis of bubble razor," in *DATE*, March 2014, pp. 1–6.

[19] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Stanford Univ., CA., 1984.

[20] C. Foley, "Characterizing metastability," in *ASYNC*, Mar 1996, pp. 175–184.

[21] S. Beer, R. Ginosar, M. Priel, R. Dobkin, and A. Kolodny, "An on-chip metastability measurement circuit to characterize synchronization behavior in 65nm," in *ISCAS*, May 2011, pp. 2593–2596.

[22] S. Beer, M. Cannizzaro, J. Cortadella, R. Ginosar, and L. Lavagno, "Metastability in better-than-worst-case designs," in *ASYNC*, 2014, pp. 101–102.

[23] S. Seo, R. Dreslinski, M. Woh, Y. Park, C. Charkrabari, S. Mahlke, D. Blaauw, and T. Mudge, "Process variation in near-threshold wide SIMD architectures," in *DAC*, June 2012, pp. 980–987.

[24] Plasma CPU, 2014. available: http://opencores.org/project,plasma.

[25] A. Leon-Garcia, *Probability, Statistics, and Random Processes For Electrical Engineering*. Pearson, 2008.