2015 European Conference on Circuit Theory and Design (ECCTD)

A Path Towards Average-Case Silicon via Asynchronous Resilient Bundled-data Design

Peter A. Beerel*, Ney Calazans†
*University of Southern California - Los Angeles, United States
†Pontifícia Universidade Católica do Rio Grande do Sul - Porto Alegre, Brazil
pabeerel@usc.edu, ney.calazans@pucrs.br

***Abstract*—The periodic nature of the global clock in traditional synchronous designs forces circuits to be margined for the worst possible case of process, voltage, temperature, and data conditions. This constrains the silicon to operate at worst-case frequencies and at conservative supply voltages. Resilient architectures promise to remove these margins, by detecting and correcting timing errors when they occur, thereby creating the potential to achieve real average-case operation. However, synchronous resilient schemes previously proposed can suffer from multiple issues, including being susceptible to metastability and requiring often complex changes to the architecture to support replay-based recovery from timing errors. These problems respectively lead to circuit failures and/or incur high timing penalties when errors occur. This paper reviews a recently proposed asynchronous bundled-data resilient template called Blade that is robust to metastability issues, requires no replay-based logic, and has low timing error penalties. It also describes some open issues and new research opportunities this template presents, including automation problems to target average-case operation, specific circuit optimizations to minimize resiliency overhead, and the need for new test procedures to tune delay lines and screen out bad chips.**

***Index Terms*—Asynchronous circuits, resiliency, bundled-data.**

I. Introduction

Traditional synchronous designs must incorporate timing margins to ensure correct operation under worst-case delays caused by process, voltage, and temperature (PVT) variations as well as data-dependency [1]. This is particularly important in low-power low-voltage designs, as performance uncertainty due to PVT variations grows from around 50% at nominal supply to around 2,000% in the near-threshold domain [2]. To address this problem, many synchronous design techniques for resilient circuits have been proposed that address delay variations. For example, canary FFs predict when the design is close to a timing failure (see e.g., [3]). Designs can then adjust their supply voltage or clock frequency either statically or dynamically to ensure correct operation at the edge of failure. Other resilient design techniques use extra logic to detect and recover from timing violations [4]–[6]. However, many of the proposed techniques are susceptible to metastability issues [7], exacerbate hold time problems [4], and/or require adding replay-based logic, often at an architectural level, to recover from these violations. This can be a challenge to implement in modern processors and often leads to high timing error penalties, which limit their benefits.

This work focuses on a recently proposed asynchronous design template that couples the architectural benefits of resilient techniques with the flexibility of asynchronous bundled-data pipelines. The template, called Blade, minimizes hold time issues, requires no replay-based logic, and is supported by an automatic translation flow from synchronous RTL specifications. It is not only safe from metastability issues but also takes advantage of the low *average* metastability resolution times, which leads to low timing error penalties compared to synchronous alternatives. It thus provides significantly higher potential performance and voltage scaling power benefits.

The paper reviews Blade principles and operation, comparing and contrasting the approach to synchronous alternatives. Its recent application to the design of a MIPS OpenCore processor illustrates techniques to reduce overheads and maximize performance and power benefits. The paper also discusses the range of designs for which this technique is likely to provide the biggest overall benefit, as well as some of the open problems that must be solved to maximize the opportunity to use Blade and make the technique commercially attractive.

II. Synchronous Resiliency and its Pitfalls

It is possible to identify in the literature two ways of achieving resiliency in synchronous systems: architecturally dependent, or "replay-based" approaches, and architecturally independent. The former include works like Razor II [5] and the Intel approach described in [1]. The problem with these approaches is that they work much like pulsed latch circuits: the wider the pulse, the more resiliency is obtained, at the cost of worsening hold time characteristics [6]. Moreover they require synchronizers in the control path, incurring long delays to know an error occurred, and demand complex replay and recovery mechanisms [1], [5]. Granted, the area overhead of these can be amortized by reusing existing recovery logic (e.g., for resuming after a mispredicted branch), but the techniques remain architecturally invasive and thus a design challenge. In contrast, architecturally independent approaches like Bubble Razor [6] require no architectural changes and can be automatically generated from standard RTL specifications. The flow involves replacing FFs with re-timed latches that have non-overlapping clocks, solving hold time problems. Bubble Razor avoids replay and recovery by immediately stalling neighboring stages via clock gating, and solves timing errors on the fly and locally. However, the template assumes that metastability can be resolved within one clock period, which is often unrealistic and leads to poor MTBF figures [7]. Another issue is that the conversion of flip-flops to error-detecting

978-1-4799-9877-7/15/$31.00

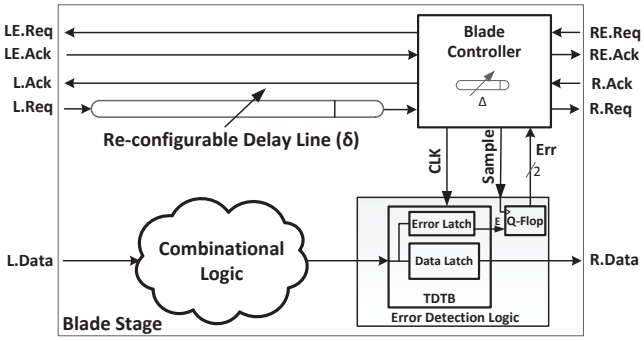

Fig. 1. The Blade architecture typical stage structure.

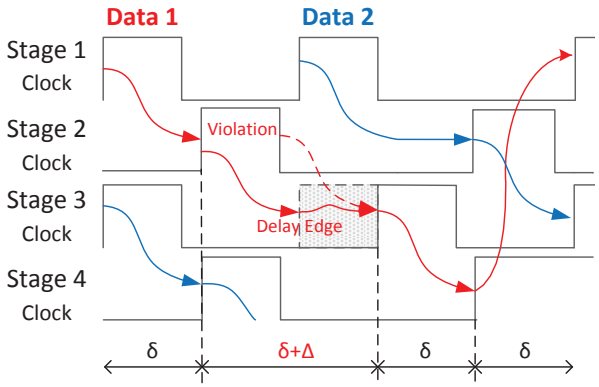

Fig. 2. Typical timing diagram for the Blade template.

latches can bring a significant increase in area and power, reducing the overall benefit.

III. The Blade Bundled-Data Architecture

As Figure 1 shows, pipeline stages in Blade use single-rail logic followed by Transition Detector with Time Borrowing (TDTB) error detecting latches (EDLs) [1], [8], Q-Flops [9], and two reconfigurable delay lines. The stage-to-stage delay line is of duration δ and controls when the TDTB goes transparent and begins to propagate data at the output of the combinational logic to the next stage. According to the timing diagram depicted in Figure 2, the asynchronous controller speculatively assumes data at the output of the TDTB latch is stable and triggers the request to the next stage via the standard bundled data request channel consisting of $R.req$ and $R.ack$. The second delay line is of duration Δ and defines a time window during which late transitions that violate this assumption (i.e. timing errors) are allowed, which is called the *timing resiliency window* (TRW). While Δ is elapsing, CLK is high (i. e. the Data Latch is transparent).

Error detecting latches are responsible for triggering an error if a timing violation occurs during the TRW. While there are several EDL implementations (e.g., [1], [4], [6], [8]), Blade employs a custom design [8] based on TDTB latches [1]. The basic design requirement is this component triggers an error on its E output in response to any transition or glitch during the TRW that is significant enough to also propagate to its data output [8]. In this way, no timing violation is missed.

In addition to the push data channel L, Blade uses a second pull *error channel* formed by signals $RE.req$ and $RE.ack$ to manage potential timing violations. Near the end of the TRW, after receiving a request on the $RE.req$ signal, the controller will trigger a signal that directs the Q-Flop to sample the E signal, determining whether or not a timing error occurred during the TRW. If an error did not occur $RE.ack$ is immediately asserted, else Δ is triggered and only after that $RE.ack$ is asserted. Because the setup time of the TDTB Error Latch may be violated, the E signal may be metastable during sampling. To cope with this, the Q-Flop has a built-in metastability filter that guarantees metastability does not propagate to its Err output. In fact, this output is intentionally made a dual-rail signal that only becomes valid after the Q-Flop has safely determined if an error occurred or not. The controller simply waits for this to happen before acknowledging the error channel request via the $RE.ack$ signal. This ensures that metastability, while possibly causing an instantaneous cycle slowdown, does not propagate to the main control path. This is in stark contrast to synchronous schemes, which must wait for a fixed, larger metastability resolution time set to guarantee a sufficiently large mean time between failures (MTBF).

There are two main delay lines that affect the performance of Blade, δ and Δ. Compared to a traditional synchronous circuit, with clock period C, we set $C = \delta + \Delta$. The TRW (defined by Δ) must be large enough to capture even the worst-case datapath delay. However, a trade off in setting these values emerges, as increasing Δ allows δ to be smaller and the system to operate faster if no timing violations (errors) occur; on the other hand, 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. The optimal Δ depends greatly upon the amount of total variation (due to data and PVT variations) that can be expected in the design, and can range from 20% to over 60% [10] of the stage total delay.

When Δ is sufficiently smaller than δ, the next stage has time to check whether the previous stage has an error before it makes its own latch transparent, delaying the transparency phase if the previous stage had an error. Stage clocks will thus remain non-overlapping, as illustrated in Figure 2, making it easy to satisfy hold times. This is again in contrast to most synchronous resiliency schemes that make meeting hold time margins harder. Supporting larger values of Δ (w.r.t. δ) is also possible and is beneficial when data/process yield high variability. However, the result is that the transparency phases of neighboring stages clocks will overlap, and this may cause hold time issues similar to those seen in synchronous approaches (see [6] for an encompassing analysis). Managing these hold time issues in synchronous resiliency approaches is particularly challenging, as they cannot be fixed by slowing down the clock. Accordingly, hold times need to margined to a higher degree than setup times. These hold margins are typically satisfied by adding hold buffers to the datapath,

but the higher margins may make the number of added buffers impractically large for designs with high variability. In contrast, an asynchronous solution like Blade can easily add programmable delays to the backward control path, actively managing the degree of transparency overlap, which makes such extra margins unnecessary. In both cases the flexibility of the asynchronous solution makes managing hold time issues far more practical.

Lastly, note that Blade also uses programmable delay lines, because under significant PVT variations it may be difficult to achieve the optimal TRW, which captures the delay of all worst-case paths via static design analysis and optimization. Programmable delay lines allow customizing the actual delay post-silicon. In particular, the authors expect that during chip characterization delay lines are analyzed and optimally configured for every chip produced, subject to some quantization error. In particular, quantization errors in δ may lead to a non-optimal expected error rate, but the overall performance will remain close to optimal [10]. Any additional margin needed to account for worst-case paths under PVT variations can be added only to the Δ delay line. Given the average frequency of timing violations can be in the range of 20%-40%, the impact of the added margin is only experienced 20-40% of the time, greatly reducing the percentage drop in performance. This is in contrast to non-resilient bundled-data designs (e.g., [11]) in which the added margin affects performance 100% of the time. As an example, a 10% increase in variation due to PVT can result in up to 30% margin penalty in synchronous designs; however, even considering a 40% rate of timing violations, the computed performance impact on Blade is less than 13% [12].

IV. CAD Flow, Challenges, and Opportunities

The authors' teams developed a preliminary flow to automatically convert single CLK domain, synchronous RTL designs to the Blade template using industry standard synthesis and P&R tools. The flow consists of various Tcl and shell scripts that drive the tools and a library of custom cells (e.g., the TDTB error latch), needed to make the template efficient.

In addition, to further reduce area and power overheads of the error detection logic, two microarchitectural optimizations are used. First, not every pipeline stage need be error-detecting, and non error-detecting stages can time borrow. Time-borrowing stages permit data to pass through the latch during the entire time it is transparent without flagging violations. The authors found that alternating between error-detecting and time-borrowing stages can work well as this effectively halves the overhead of error detection logic while still providing sufficient resiliency. Secondly, only latches that terminate near-critical paths [12] need to be error detecting, further reducing the number of EDLs in the entire design.

As Figure 3 illustrates, the flow has five main steps:

1) **Synchronous Synthesis:** The synchronous RTL is synthesized to a flip-flop (FF-based) design for given clock.
2) **FF to Latch Conversion:** FFs are converted to master-slave latches by synthesizing the design using a fake library of standardized D flip-flops (DFFs) that can be easily mapped to standard cell latches.

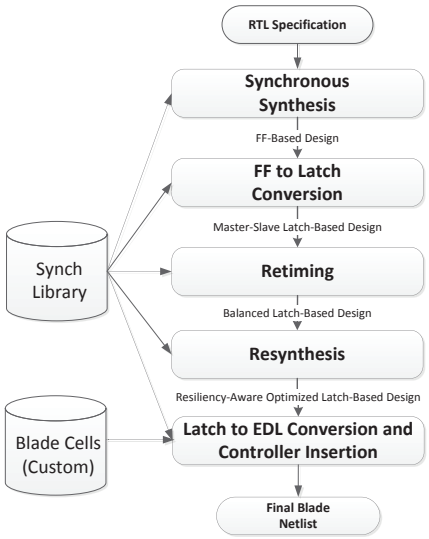

Fig. 3. The Blade design flow.

3) **Latch Retiming:** The latch-based netlist is retimed using a target TRW, where the combined path delay constraint of any two stages equals the given clock period. The purpose is to split the critical path in two parts, which enables hiding inter-stage Blade handshaking overheads.
4) **Resynthesis:** The retimed netlist is then resynthesized to reduce the number of TDTBs and increase performance of the final resilient netlist. In particular, re-synthesizing the logic happens such that the delay to a subset of latches is sufficiently fast to guarantee that data is stable before the latches go transparent (i.e., is not near-critical). This means that the latches do not need to be error-detecting, reducing the EDL overhead, and potentially reduces the error rate at the expense of increasing the datapath logic area. Targeting latches that cause the most errors in typical applications can lead to significant reductions in error rates with marginal increase in area. In [12] the authors employ a simple brute-force search, but more powerful means of identifying which subset of latches to speed up is an interesting area of future work.
5) **Blade Conversion:** The resynthesized latch-based netlist is then converted to the Blade template, by removing clock trees and replacing these with Blade controllers. The control logic, delay lines, and error detection logic are also inserted to create a final Blade netlist. There are many ways to implement the control logic [13]; using burst-mode specifications has been explored in [12]. In addition, there are many ways to design delay lines, as the authors and others explore in [14]. However, ensuring these have their proposed delay during P&R, when gate sizing, wire delay, and cross-couping capacitance play a role is non-trivial. Some works suggest ways to enable commercial P&R tools to automate this step [15]. More generally, however, creating a comprehensive P&R flow that optimizes the average-case delay of Blade designs is another interesting area for future work.

The authors' preliminary pre-P&R flow was tested and evaluated on a 3-stage version of Plasma [16], a MIPS OpenCore CPU, targeting a 28nm FD-SOI technology. The gate-level Blade design was compared to the equivalent synchronous design, and post-synthesis results demonstrate that for an area overhead of 8.4%, the Blade version of Plasma achieves a 19% average performance boost with a timing resiliency window of 30%. Out of the 8.4% area overhead, 32% is due to the use of EDLs and to the FF to latch conversion. With the removal of synchronous PVT margins, it led to an estimated 30%-40% improvement in performance [12]. Another aspect that requires additional research is configuration and test. In particular, it is necessary to develop test methods for optimally tuning the programmable delay lines based perhaps on *in situ* error rate monitoring, as well as find means to identify and discard chips with delay variations too large to correct.

V. Discussion and Conclusions

Asynchronous design has become an increasingly attractive alternative to synchronous design in several applications for a variety of reasons. For example, Intel showed that high-performance quasi-delay-insensitive (QDI) design is sufficiently robust and effective for high performance networking chips [17]. Moreover, the challenges of managing a global clock in large neuromorphic chips, have driven IBM [18] and Stanford [19] to adopt an asynchronous mostly QDI interconnect. Other academic researchers have found that built-in flow-control in bundled-data network-on-chips lead to significant benefits in terms of latency and area compared to synchronous counterparts [15]. However, efforts to commercialize bundled-data pipelines for processors demonstrated only marginal performance benefits [11]. We hope that adding resiliency opens the door for much larger performance advantages for a broader range of applications.

Generally speaking, the range of architectures and applications for which resiliency adds value depends on two factors: the overhead one can expect from the error-detecting latches and the variance of the data and PVT dependent delays [20]. The benefits of a resilient design are higher when the fraction of combinational to sequential area is large, because the relative overheads of the TDTBs is smaller. Thus, resilient design favors less pipelined designs. Moreover, an architecture where the difference between average and worst-case delay is large will likely benefit more than a well balanced architecture and even more likely if the worst-case paths are rarely executed [20]. For example, architectures that involve complex logic with rarely executed long carry chains will benefit more than balanced designs consisting of many regular structures (e.g., memories). Fortunately, there are many architectural decisions that can be made to favor timing resilient templates [20].

Lastly, it is important to emphasize that the advantages of asynchronous resilient designs are difficult to approximate in synchronous architectures. In particular, asynchronous resilient designs adapt to the quite low average-case time it takes for metastability to resolve, which in principle can be unbounded. In contrast, the periodic nature of the clock forces synchronous alternatives to be designed for a much larger fixed resolution time, set by an acceptable MTBF. This difference enables our solution to be architecturally-independent, whereas existing robust synchronous solutions are forced to be based on recover and replay logic to obtain reasonable MTBF.

Thus, we believe asynchronous resiliency is a promising research direction to obtain efficient designs which adapt to the combination of PVT and data variations - in short providing a path to achieving average-case silicon.

References

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

[2] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.

[3] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De, "Tunable Replica Circuits and Adaptive Voltage-Frequency Techniques for Dynamic Voltage, Temperature, and Aging Variation Tolerance," in *VLSI*, 2009, pp. 112–113.

[4] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, "Timber: Time borrowing and error relaying for online timing error resilience," in *DATE*, March 2010, pp. 1554–1559.

[5] S. Das, C. Tokunaga, S. Pant, M. Wei-Hsiang, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance," *IEEE JSCC*, vol. 44, no. 1, pp. 32–48, Jan 2009.

[6] 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 JSCC*, vol. 48, no. 1, pp. 66–81, Jan 2013.

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

[8] M. T. Moreira, D. Hand, N. L. V. Calazans, and P. A. Beerel, "TDTB Error Detecting Latches: Timing Violation Sensitivity Analysis and Optimization," in *ISQED*, 2015, pp. 379–383.

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

[10] D. Hand, H. Huang, B. Cheng, Y. Zhang, M. Moreira, M. Breuer, N. Calazans, and P. Beerel, "Performance Optimization and Analysis of Blade Designs under Delay Variability," in *ASYNC*, May 2015.

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

[12] D. Hand, M. Moreira, H. Huang, D. Chen, F. Butzke, Z. Li, M. Gibiluka, B. M., N. Calazans, and P. Beerel, "Blade - A Timing Violation Resilient Asynchronous Template," in *ASYNC*, May 2015.

[13] P. Beerel, R. Ozdag, and M. Ferreti, *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010.

[14] G. Heck, L. Heck, A. Singhvi, M. Moreira, P. Beerel, and N. Calazans, "Analysis and Optimization of Programmable Delay Elements for 2-Phase Bundled-Data Circuits," in *VLSI Design*, 2015, pp. 321–326.

[15] A. Ghiribaldi, D. Bertozzi, and S. Nowick, "A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems," in *DATE*, Mar. 2013, pp. 332–337.

[16] Plasma CPU, 2014. Available: http://opencores.org/project,plasma.

[17] M. Davies, A. Lines, J. Dama, A. Gravel, R. Southworth, G. Dimou, and P. Beerel, "A 72-Port 10G Ethernet Switch/Router using Quasi-Delay-Insensitive Asynchronous Design," in *ASYNC*, May 2014, pp. 103–104.

[18] P. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[19] P. Merolla, J. Arthur, R. Alvarez, J.-M. Bussat, and K. Boahen, "A Multicast Tree Router for Multichip Neuromorphic Systems," *IEEE TCAS-I*, vol. 61, no. 3, pp. 820–833, Mar. 2014.

[20] J. Sartori and R. Kumar, "Exploiting Timing Error Resilience in Processor Architecture," *ACM ToECS*, vol. 12, no. 2, pp. 89:1–89:25, May 2013.