2015 21st IEEE International Symposium on Asynchronous Circuits and Systems

Blade - A Timing Violation Resilient Asynchronous Template

Dylan Hand*, Matheus Trevisan Moreira*†, Hsin-Ho Huang*, Danlei Chen*, Frederico Butzke‡, Zhichao Li*, Matheus Gibiluka†, 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
‡Computer Science Department, Universidade de Santa Cruz do Sul, Santa Cruz do Sul, Brazil

***Abstract*—Resilient designs offer the promise to remove increasingly large margins due to process, voltage, and temperature variations and take advantage of average-case data. However, proposed synchronous resilient schemes have either suffered from metastability or require modifying the architecture to add replay-based logic that recovers from timing errors, which leads to high timing error penalties and poses a design challenge in modern processors. This paper presents an asynchronous bundled-data resilient template called Blade that is robust to metastability issues, requires no replay-based logic, and has low timing error penalties. The template is supported by an automated design flow that synthesizes synchronous RTL designs to gate-level asynchronous Blade designs. The benefits of this flow are illustrated on Plasma, a 3-stage OpenCore MIPS CPU. Our results demonstrate that a nominal area overhead of the asynchronous template of less than 10% leads to a 19% performance boost over the synchronous design due to average-case data and a 30-40% improvement when synchronous PVT margins are considered.**

I. Introduction

Traditional synchronous designs must incorporate timing margin to ensure correct operation under worst-case delays caused by process, voltage, and temperature (PVT) variations as well as data-dependency [1]. Different asynchronous templates have been proposed to address this problem (e.g., [2]). Quasi-delay-insensitive (QDI) templates use completion signal logic, which makes them robust to delay variations at the cost of increased area and high switching activity due to a return to zero paradigm [3]. Bundled-data templates (e.g., micropipelines [4]) use delay lines matched to single-rail combinational logic, providing a low area, low switching activity asynchronous solution (e.g., [5]). However, the delay lines must be implemented with sufficiently large margins in the presence of on-chip variations, reducing the advantages of this approach. Researchers have proposed different solutions to mitigate these margins, such as duplicating the bundled-data delay lines [6], constraining the design to regular structures such as PLAs [7], and using soft latches [8].

Meanwhile, the synchronous research community have investigated various methods to reduce timing margins in clocked designs. Among these efforts, we highlight resilient design techniques, which rely on extra logic to detect and recover from timing violations [9]–[11]. However, many of the proposed techniques are susceptible to metastability [12] or require adding replay-based logic, often at an architectural level, to recover from these violations, which can be a challenge in modern processors and lead to high timing error penalties.

This paper presents a new asynchronous bundled-data template called Blade, which couples the architectural benefits of resilient techniques with the flexibility of asynchronous pipelines. In particular, Blade enables average case performance, is robust to metastability issues, requires no replay-based logic, and has very low timing error penalties.

Blade uses single-rail logic, reconfigurable delay lines, and error-detecting latches [1] that reliably detect timing violations. The template implements 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. Moreover, it is supported by an automated design flow that synthesizes synchronous RTL designs to gate-level Blade designs. The flow includes automatic FF to latch conversion, retiming, and resynthesis to further improve average-case performance while minimizing area. The potential benefits of Blade and this flow are explored in a case study using a 3-stage MIPS OpenCore CPU, Plasma [13], targeting an FDSOI 28nm technology. We compare the gate-level Blade design 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 the removal of synchronous PVT margins, we estimate a 30%-40% improvement in performance.

The remainder of this paper is organized as follows. Section II introduces the Blade template, explores the main components, and details the associated timing assumptions and overheads. Section III documents the automated conversion process used to synthesize Plasma and compares the performance between the asynchronous and synchronous designs. Finally, Sections IV and V provide discussion of the case study results, general observations, conclusions, and several opportunities for future optimizations and applications.

II. Blade Template

The proposed Blade template, as shown in Figure 1, uses single-rail logic followed by error detecting latches (EDLs), two reconfigurable delay lines, and an asynchronous Blade controller. The first delay line is of duration δ and controls when the EDL becomes transparent, allowing the data to propagate through the latch. The Blade controller speculatively assumes that the data at the input of the EDL is stable when it becomes transparent and thus sends an output request along

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

1522-8681/15 $31.00 © 2015 IEEE
DOI 10.1109/ASYNC.2015.13

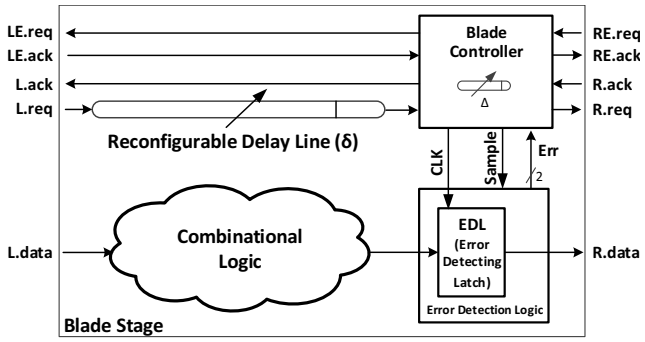


Fig. 1: The Blade template

the typical bundled data channel L/R. The second delay line, with duration Δ, defines the time window during which the EDL is transparent. If data changes during this window, but stabilizes before the latch becomes opaque, it is recorded as a timing violation, which can subsequently be corrected. Consequently, Δ defines a timing resiliency window (TRW) after δ during which the speculative timing assumption may be safely violated.

In particular, if the combinational output transitions during the TRW, the error detection logic flags a timing violation by asserting its Err signal, which is sampled by the controller. The Blade controller then communicates with its right neighbor using a novel handshaking protocol implemented with an additional error channel (RE/LE) to recover from the timing violation by delaying the opening of the next stage's latch, as will be described in more detail in Section II-B.

A. Error Detection Logic

As illustrated in Figure 2, the error detection logic consists of EDLs, generalized C-elements, and Q-Flops [14]. While there are many possible implementations of EDLs (e.g., [1], [9], [11], [15]), we implemented a custom design based on the Transition Detecting Time Borrowing (TDTB) latches proposed in [1], a functional block diagram of which is shown in Figure 2. The already low overhead of the TDTB is further reduced by integrating the transition detector into the pass-gate latch circuit, where inherit internal latch delays are repurposed to replace the t_{TD} delay line connected to the XOR gate. The XOR gate itself is also optimized at the transistor level to improve the transition detector's sensitivity [15].

The generalized C-elements in Figure 2 are also designed at the transistor level using the flow proposed in [16] and act to temporarily remember violations detected by the EDL during the high phase of *CLK*. While the input connected to *CLK* is symmetric, i.e. required for both low-to-high and high-to-low output transitions, the *X* signal from the EDL feeds a positive asymmetric input, which can only affect low-to-high transitions. Accordingly, the generalized C-element will switch to 0 if *CLK* is at 0 and to 1 only if both *CLK* and the *X* input are at 1. This creates a memory cell that temporarily stores any violation detected by the EDL during the high phase of *CLK*, i.e. during the TRW. Note that a compensation delay is added by the t_{comp} delay line, the purpose of which will be explained in Section II-E.

Under normal operation, the pulse on *X* will be sufficiently large to guarantee the output node of the C-element is fully charged, indicating an error has occurred while *CLK* is high, as outlined in [15]. However, because the data may violate the setup time of the EDLs, the *X* signal and the C-element may exhibit metastablity, as will be further discussed in Section II-C. To ensure safe operation, this metastability must be filtered out before reaching the main controller. In synchronous designs, the filtering would be handled through multi-stage synchronizers increasing the latency of error detection dramatically. In contrast, the output of the C-element in the Blade template is sampled at the end of the TRW using a Q-Flop, which contains a metastability filter that prevents the dual rail output signal, *Err*, from ever becoming metastable, even if the C-element is in a metastable state. The Blade controller simply waits for the dual-rail *Err* signal to evaluate to determine whether or not an error occured, gracefully stalling until metastabiilty is resolved.

To minimize area overheads due to error detection, it is desirable to amortize the cost of the C-elements and Q-Flops across multiple EDLs. As shown in Figure 2, a 4-input generalized C-element can combine the *X* signals of 3 EDLs using parallel inputs such that an error from any of the three EDLs triggers the C-element output to fire. An OR gate can further combine 4 C-elements before reaching a Q-Flop. In this scenario, a single Q-Flop will accurately catch errors and filter metastability from 12 EDLs. Counterintuitively, this added delay provides timing benefits in addition to multifaceted area savings, as will be further explored in Sections II-E and II-F. Note that the C-element's static implementation [3] makes it undiserable to have more than 4-inputs as the PMOS stack grows too large.

To further reduce area and power overheads of the error detection logic, two additional micro-architectural optimizations are considered. 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 any violations. In particular, we found alternating between error-detecting and time-borrowing stages can work well as this

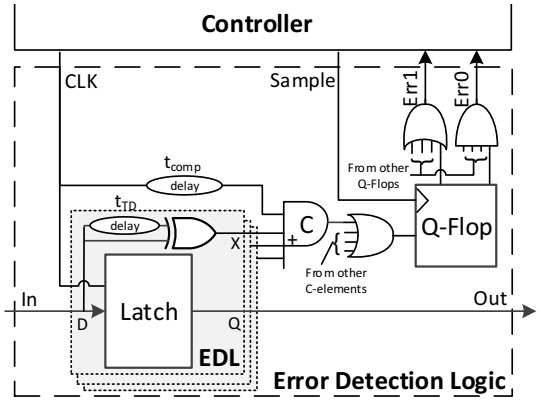


Fig. 2: Error detection logic including block level diagram of error detecting latch

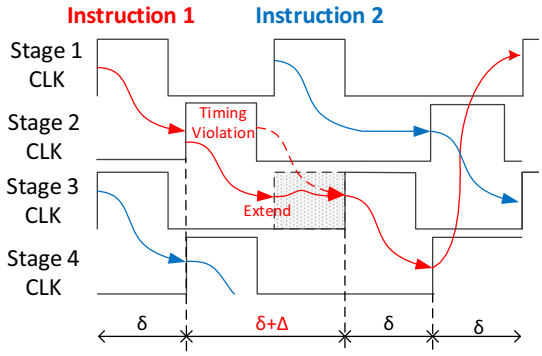
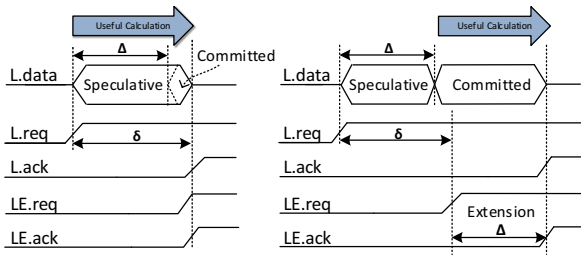


Fig. 3: Timing diagram of Blade template

effectively halves the overhead of error detection logic while still providing sufficient resiliency. Secondly, we define a stage's critical path as the longest possible input to output path in the combination logic, which sets the endpoint of the TRW. If another path has delay within the TRW it is said to be "near-critical". Only latches that terminate near-critical paths[1] need be error detecting, further reducing the number of EDLs required in the entire design.

B. Speculative Handshaking Protocol

The proposed Blade template implements a new form of asynchronous handshaking: *speculative handshaking*. To understand this protocol, we first introduce the expected behavior of the *CLK* signals of four Blade stages in a pipeline, shown in Figure 3. As Instructions 1 and 2 flow through the pipeline, the arrows indicate the dependency of one clock signal on another. Instruction 1, shown in red, launches from Stage 1 at time zero. While Stage 2's latch is transparent, a timing violation occurs indicating the δ delay line in Stage 1 was shorter in duration than the combinational logic path. The rising edge of Stage 3's *CLK* signal is nominally scheduled to occur δ time units after Stage 2's, shown as the dotted gray region; however, the timing violation extends this time, giving Instruction 1 a total of $\delta + \Delta$ to pass from Stage 2 to Stage 3. Conversely, Instruction 2 does not suffer a timing violation in Stage 2, which allows Stage 3's *CLK* signal to activate δ time units after Stage 2's.



(a) Without extension (b) With extension

Fig. 4: Speculative handshaking protocol

An example of the speculative handshaking protocol that achieves this behavior using two-phase signaling is shown in Figure 4. Here, a Blade stage speculatively receives a request and data value on its L channel. The request passes through the δ delay line before reaching the Blade controller while the speculative data propagates the combinational logic. The Blade controller then checks with the previous stage's controller if the speculative request was sent before the input data was actually stable, i.e., if the previous stage experienced a timing violation. This action is implemented via a second handshake on the pull-channel LE. When no timing violations occur in the previous stage (Figure 4a), the *LE.req* signal is immediately acknowledged by *LE.ack*, indicating the speculative request was correct and no extension is required. In Figure 4b, on the other hand, a timing violation occurs in the previous stage causing the *LE.ack* signal to be delayed by Δ time units while the final, committed input data passes through the stage's combinational logic. In both cases this stage is given a nominal delay of δ to process stable data. In addition, notice that the information of whether a timing violation occured is not directly transmitted between stages; rather, this information is encoded into the variable response time between *LE.req* and *LE.ack*. Additionally, the *R.req* signal of the controller, not shown in Figure 4, is coincident with the arrival of *LE.ack*, which forces the R channel request to be delayed by Δ as well when an extension is necessary.

C. Metastability Analysis

Since the input data may stabilize sometime after the opening of the latch, Blade's susceptibility to metastability (MS) must be examined. MS in the datapath is not a concern as we ensure Δ is set sufficiently large as to avoid closing the latch while the datapath is still evaluating. However, certain internal nodes of the error detection logic can become metastable due to several different scenarios:

- *Scenario M1*: A data transition occurring near the rising edge of *CLK* will cause a pulse on the *X* output of the EDL to occur before the rising edge of *CLK* arrives at the generalized C-element. In this case, the C-element may only partially discharge its internal dynamic node, resulting in metastability at the output. Fortunately, the width of the timing window in which this can occur is sufficiently small that timing violations caused by these transitions are short in duration and their impact can be absorbed by the following stage. Consequently, the value to which metastability resolves is not critical and the circuit will work correctly regardless of the value to which the Q-flop eventually resolves.
- *Scenario M2*: Late transitions in the datapath can cause pulses on the EDL's *X* output that are coincident to the falling edge of *CLK*. Similarly, the rising edge of the C-element's output may coincide with the rising edge of the Q-Flop's sampling signal. Timing violations in this case indicate the datapath is so slow that it exceeds our timing resiliency window and such circuits should be filtered out during post-fabrication testing.
- *Scenario M3*: Datapath glitches that occur in the middle of the TRW may also induce metastability in the C-element. However, through careful design of the EDL, these input glitches will only cause glitches on the *X* output and not the data output [15], i.e. the transition

[1]Note that by definition a critical path is also "near-critical".

detector is more sensitive to glitches than the data latch itself. Consequently, metastability in this scenario only affects performance but not correctness, just as MS in Scenario M1. Moreover, the probability of entering MS can be reduced by making the generalized C-element more sensitive to glitches than the transition detector.

In rare cases, the output of the Q-Flop will take an arbitrarily long time to resolve due to internal MS. In a robust synchronous design, similar resolution delays translate directly into increased margins or extra clock cycles and synchronizers to wait for this rare occurrence to resolve. However, due to the asynchronous nature of our template, the Blade controller will gracefully wait for the metastable state to resolve before allowing the next stage to open its latch, effectively stalling the stage and ensuring correct operation. This is a significant benefit of asynchronous design which, to the best of our knowledge, cannot be easily approximated in synchronous alternatives.

D. Blade Controllers

The Blade controller is implemented as a set of three interacting Burst-Mode state machines [17] and synthesized using the tool 3D [18]. Figure 5 shows these state machines for pipeline stages with EDLs. Note that intermediate signals *goL*, *goR*, and *goD* are communication signals between the three individual state machines, and signals *delay*, *edi*, and *edo* are used to add the Δ delay line into the controller. For simplicity, the delay line is duplicated between *CLK*→*delay* and *edo*→*edi*. Consolidating these to a single delay line is left as future work.

We have extended this controller to a token version, which generates an output request after reset, as well as simplified versions for stages without error detection logic, creating four distinct Blade controllers. We added reset to the unmapped netlists and manually mapped them to our 28nm library of gates. For all cases, the implicit fundamental mode timing assumption [17] was validated using a simulation environment with random environmental delays.

E. Timing Constraints

The datapath in Blade most closely resembles a standard time borrowing design [19]. However, the introduction of error detecting stages as well as the error detection logic itself alters these constraints making the analysis of Blade timing constraints similar to that of Bubble Razor [11].

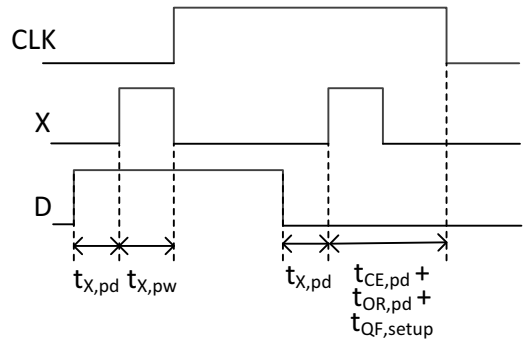


Fig. 6: Timing constraints in Blade

The annotated timing diagram of the *CLK*, *X*, and *D* signals for a single error detecting Blade stage in Figure 6 shows the overheads associated with our error detection logic. The delay through the error detection logic is comprised of five components: (i) propagation delay from *D* to *X* of the EDL, $t_{X,pd}$; (ii) output pulse width of pin *X*, $t_{X,pw}$; (iii) C-element propagation delay, $t_{CE,pd}$; (iv) Q-Flop setup time, $t_{QF,setup}$; and (v) propagation delay of the OR gate between the C-elements and Q-Flop, $t_{OR,pd}$.

Note that $t_{X,pd}$ and $t_{X,pw}$ would enforce a large setup time before the EDL becomes transparent to ensure a transition before the rising edge of *CLK* is not flagged as a timing violation. Therefore, a small compensation delay $t_{comp} = t_{X,pd} + t_{X,pw}$ is added to the *CLK* input of the C-element, as seen in Figure 2, to prevent these unintended errors.

1) Timing Resiliency Window: The actual size of the timing resiliency window is affected by each of the error detection logic delays. In particular, the TRW can be defined as:

$$TRW = \Delta + t_{X,pw} - (t_{CE,pd} + t_{OR,pd} + t_{QF,setup}) \quad (1)$$

Note that $t_{X,pd}$ impacts the TRW in two ways: positively for transitions occurring near the rising edge of the CLK and negatively for transitions at the falling edge. Hence this term cancels out in (1).

2) Propagation Delay: When using the optimizations described in Section II-A, there are three potential logic path end points. First, pipeline stages that do not have error detection use regular latches that allow time borrowing. Second, latches in error detecting pipeline stages that are not on near-critical paths are not converted to EDLs and have constraints similar to flops. Finally, the EDLs in error detecting stages are the end points for paths with delay longer than δ.

For paths ending at non-error detecting stages, the propagation delay is simply:

$$t_{pd,TB} \leq \delta + \Delta - t_{latch,CQ} - t_{latch,setup} \quad (2)$$

where $t_{latch,CQ}$ is the clock to Q delay of the source latch and $t_{latch,setup}$ is the setup time of the sink latch[2]. For paths ending at non-error detecting latches in an error detecting stage, the propagation delay is also straightforward:

$$t_{pd,NE} \leq \delta - t_{latch,CQ} \quad (3)$$

Note that latch setup time is not included in this constraint because the data is arriving at the rising edge of clock, i.e. when the latch becomes transparent.

Finally, the propagation delay of paths ending at EDLs can be derived as:

$$t_{pd,E} \leq \delta + TRW - t_{latch,CQ} \quad (4)$$

where TRW is defined as in (1). Note that latch setup time does not appear here either as the requirement to meet the TRW is always stricter than the latch's setup time.

[2]This equation assumes that each stage can borrow the maximum amount of Δ, which occurs when time borrowing and non-time borrowing stages are alternated. See [19] for the more general time borrowing constraints.

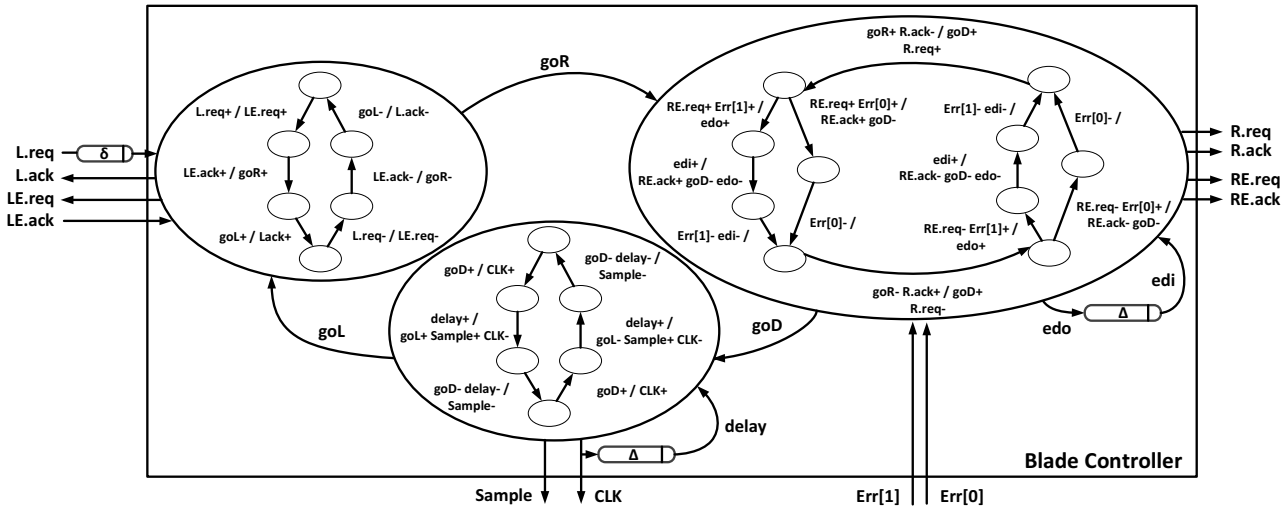


Fig. 5: Burst-mode state machines for Blade controller with error detection

3) Contamination Delay: The Blade controller enforces a condition that latches of neighboring stages cannot be transparent at the same time, which provides significant hold time margin. When including the clock tree delays, $t_{CLK,pd}$, the hold time constraint between two stages is:

$$t_{cd} \geq (t_{CLK_R,pd} - t_{CLK_L,pd}) - t_{ack\_to\_clk} \tag{5}$$

where *L* and *R* represent two neighboring stages and $t_{ack\_to\_clk}$ is the delay from R's controller generating an acknowledgement signal to L's controller raising its clock signal. In practice, $t_{ack\_to\_clk}$ is around 4 gate delays, making t_{cd} small or even negative for balanced local clock trees. This is in contrast to many resiliency schemes which exacerbate hold time issues (e.g. [11]).

4) Hiding Handshaking Overhead: After a request is received at a Blade controller, a full two-phase handshake must occur on its *LE* channel to check if the previous stage suffered a timing violation. Even when no violations occur, this process takes a non-zero amount of time, t_{EC}, due to gate delays in the two controllers. Fortunately, this delay can be hidden completely by shortening the stage to stage delay, δ, by t_{EC}. If δ is not shortened, the circuit will still operate correctly but with slower performance.

F. Maximum Timing Resiliency Window

To compute the maximum width of the timing resiliency window, TRW_{max}, we first define a few additional delays:

- $t_{QF,pd}$: the nominal propagation delay from the sample input to the outputs of the Q-Flop without metastability.
- $t_{ET,pd}$: the maximum propagation delay of the AND and OR trees that collect the individual dual-rail error signals from the Q-Flops.

To find TRW_{max}, it is also helpful to first define Δ_{max}, the maximum clock pulse width for a Blade stage. Because opening the latch of one stage depends on checking if an error occurred in a previous stage, Δ cannot be equal to δ and still achieve the expected cycle time including overheads. Therefore, Δ_{max} is conservatively set as:

$$\Delta_{max} = \delta - t_{ET,pd} - t_{QF,pd} - t_{Err[0]\_to\_clk} \tag{6}$$

where $t_{Err[0]\_to\_clk}$ is the internal controller delays from receiving Err[0] one controller to raising the clock signal in the subsequent stage. Combining (1) and (6) we find:

$$\begin{aligned} TRW_{max} = \delta - t_{ET,pd} - t_{QF,pd} - t_{Err[0]\_to\_clk} \\ + t_{X,pw} - (t_{CE,pd} + t_{OR,pd} + t_{QF,setup}) \end{aligned} \tag{7}$$

In some cases, a large TRW may not be ideal and setting it to 20-30% may be sufficient, as was done in [11]. In addition, reasonable estimates of $t_{CE,pd}$ and $t_{QF,setup}$ in a modern process are on the order of tens of ps. However, the magnitude of $t_{ET,pd}$ and $t_{OR,pd}$ depend on multiple factors, including the number of EDLs per stage and the degree to which the EDLs are amortized across Q-Flops. This presents an interesting optimization problem in which reducing the number of EDLs may also maximize the potential performance of the design.

III. Case Study: Plasma 3-Stage CPU

A. Automatic Translation to Blade Template

An automated flow to convert single CLK domain synchronous RTL designs to asynchronous Blade using industry standard tools, including DesignCompiler and PrimeTime from Synopsys (for synthesis and STA) and NC-Sim from Cadence (for simulation), was developed to analyze the benefits of the proposed template on a 3-stage version of Plasma [13], a MIPS OpenCore CPU, targeting a 28nm FD-SOI technology. The flow consists of various Tcl and shell scripts, a library of custom cells, and a Verilog co-simulation environment for verification and analysis that are wrapped in a Makefile system, which provides multiple configuration knobs to control the synthesized frequency, TRW, compensation for overheads, and other aspects of the design. The flow has 5 main steps:

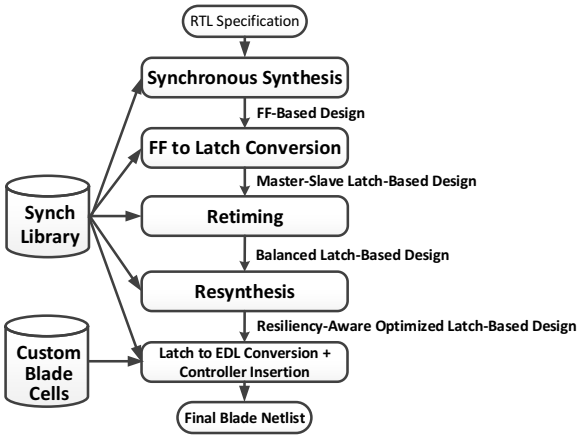


Fig. 7: Blade design flow

1) **Synchronous Synthesis:** The synchronous RTL is synthesized to a flip-flop (FF) based design at a given clock frequency with preset I/O delays and output load values.
2) **FF to Latch Conversion:** The 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.
3) **Latch Retiming:** The latch-based netlist is then retimed using a target TRW that defines the maximum time borrowing allowed, where the combined path delay constraint of any two stages equals the given clock period.
4) **Resynthesis:** The retimed netlist is then resynthesized to optimize the expected area and performance of the final resilient netlist, as will be described in Section III-C.
5) **Blade Conversion:** The resynthesized latch-based netlist is then converted to the Blade template by removing clock trees and replacing them with Blade controllers. The control logic, delay lines, and error detection logic are also inserted to create a final Blade netlist.

The final Blade netlist is validated via co-simulation with the synchronous netlist from step 1 to verify correct operation and measure performance. In particular, to verify correct operation the stream of inputs is forked to both the synchronous and Blade netlists and the stream of outputs is compared.

B. Handling Macros

In many designs there may be logic blocks that are either implemented using hard macros or would be problematic to convert to the Blade template directly. Therefore, it is beneficial to capture errors at the inputs to these cells and ensure the timing for the macro is satisfied at the ideal target clock frequency, i.e. the given clock period minus the TRW. Fortunately, an important advantage of asynchronous design is that we can add new pipeline stages to the design without changing functionality. For Blade, we take advantage of this feature by adding an error-detecting pipeline stage at the input of the macro controlled by a non-token-buffer pipeline controller. These controllers only pass tokens through the system; unlike token controllers, they do not generate tokens on reset. Therefore, the functional behavior of the design is unchanged. In synchronous designs, this would not be possible without major architectural modifications as adding a pipeline stage changes the functionality greatly.

As an example of this process, the Plasma CPU contains a 32 entry register file (RF) that can be implemented using a memory generator or synthesized directly as 32 flip-flops per register. It is not uncommon for either the input or output of the RF to be on a critical path in the CPU; however, it is often the case that the majority of this critical delay occurs outside of the macro boundary (e.g. an ALU's result being stored into the RF). With Blade, if a near-critical path ends at the RF, all internal registers would need to be converted to EDLs, resulting in large area overheads. But we can exploit the fact that the decoding logic inside the RF macro is quick in comparison to the rest of the input path by adding a non-token Blade stage on the data and address inputs to the RF. We therefore achieve the same resiliency benefits while reducing the number of EDLs drastically without changing the macro itself; for a 32-bit RF, only 37 EDLs are required when placed at the input (32 for data, 5 for address) instead of 1024 when the internal flops are converted to EDLs. The nominal datapath delay from the added error detecting Blade stage, through the RF, and to the subsequent Blade stage must be faster than the ideal target frequency for this method to be effective, which was easily met in our case.

C. Resynthesis

Each EDL adds overhead in timing and area in multiple ways: i) the EDL itself is larger than a latch; ii) the number of C-elements and Q-Flops increase; and iii) the size of the OR/AND trees needed to combine error signals also increases. Therefore, it is desirable to minimize the number of EDLs while maintaining both the robustness to timing violations and the expected performance increases. One method to achieve these goals is through resynthesis. The retiming step of the Blade design flow generates a report of latches that should be converted to EDLs, i.e. all latches that are on a near-critical path, such that the static timing analysis indicates a timing violation would occur when running at the ideal target frequency. Constraining the delay to one of these latches to be no greater than the target frequency and resynthesizing the design would therefore remove the selected latch from the EDL report, allowing it to be implemented using a standard

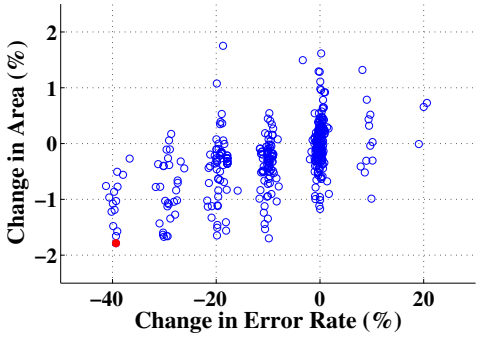


Fig. 8: Resynthesis to improve area and decrease error rate

latch rather than an EDL. Although the combinational area may increase due to tighter constraints on certain paths, this overhead can be offset if multiple latches that were slated to become EDLs are no longer on near-critical paths as well. Unfortunately, the high degree of shared paths in the combinational logic makes it challenging to estimate the the reduction in EDLs, i.e. constraining one latch may also speed up shared paths to many other latches. Moreover, the reduction of EDLs combined with faster combinational logic may lead to a reduced frequency of timing violations during simulation, which affects the maximum performance of the circuit.

Without reliable methods of estimating these two effects, it is difficult to know a priori which latch(es) in the EDL report to further constrain; therefore, a brute-force approach in which all latches marked EDL are tested one by one is employed to find a suitable candidate latch. Figure 8 shows the results of this approach on the Plasma CPU, with a given frequency of 666MHz and a target frequency of 952MHz. After retiming, there 456 latches required to be converted to EDLs. A max delay constraint equal to the target clock period was placed on each latch separately to ensure no timing violations would occur. Then the netlist was resynthesized, converted to Blade, and simulated in the co-simulation environment to obtain both the post-conversion area and error rate, i.e. the frequency of timing violations averaged over the entire simulation. The best point, highlighted in red in Figure 8, yields a 27% decrease in number of EDLs with a 1.79% decrease in overall area, and 39% improvement in error rate. Note that the potential benefits of this resynthesis approach will depend heavily on the initial starting frequency, i.e. a design that is already heavily constrained cannot easily be constrained further to achieve area and performance benefits.

D. Area and Performance Comparisons

Using the flow described in Section III-A, Plasma was converted from a 666MHz synchronous flop-based design to Blade with a timing resiliency window of 30% in a 28nm FDSOI process. New library cells were created and characterized for the EDLs, C-elements, and Q-Flops to obtain accurate area and timing information for the synthesis tools and our simulations. While a behavioral model of the burst-mode Blade controller, described in Section II-D, was used for simulation, a preliminary gate-level design was also mapped to our technology to estimate controller area and timing. The timing information generated through synthesis was then used to inform delays in our behavioral controllers and delay lines. The final asynchronous control logic and error detection overheads are depicted in Figure 9. The overall area overhead from the original synchronous design is 8.4% after one pass of the resynthesis method presented in Section III-C.

To compare the performance between the synchronous and asynchronous designs, we executed one iteration of an industry standard benchmark, CoreMark [13], on both CPUs. The Blade design achieved an average frequency of 793MHz with a peak frequency of 950 MHz, an increase of 19% and 42%, respectively. A plot of the performance over time is shown in Figure 10, where average performance is measured across the entire benchmark while the instantaneous performance is measured only over the previous 1,000 cycles. The Blade design quickly switches operating frequencies, benefiting from large variations in data dependent delays near the beginning of the benchmark before the overall performance averages to just under 800MHz.

IV. Discussion

A. Retiming

The retiming step of the Blade conversion flow may reduce the performance of Blade and increase area overhead of the final netlist. This opens the door to optimization problems that involve retiming to maximize average case performance. For example, a traditional synchronous retiming algorithm may prefer unbalanced paths between time-borrowing latches in order to save area without sacrificing performance. However, the final placement of the latches also affects the number of near-critical paths in the circuit. For resilient designs, poor latch placement could unnecessarily inflate the number of EDLs, resulting not only in larger area overheads but also higher error rates and lower performance. Finding ways to exploit the positive benefits of retiming in resilient architectures such as Blade is an area of on-going research.

B. Performance with Margins

Because Blade utilizes programmable delay lines, it is expected that, after tuning, these delays will reasonably track the delay of datapath combinational logic even in the presence of variations due to process and environmental factors [20]. Therefore, we can reduce the amount of margin required in our timing assumptions compared to traditional synchronous designs. The δ delay line impacts the start of the timing resiliency window, and thus may lead to fluctuations in expected error rate under variation, but timing violations will still be identified and corrected. Accordingly, the majority of margin can be added to the Δ delay line, which controls the clock pulse width and delay penalty when a timing violation occurs. In our simulations with Plasma, the average frequency of timing violations were 20% - 40% in the benchmarks we considered. Thus, the impact of the added margin is only experienced 20-40% of the time, greatly reducing the percentage drop in performance compared to synchronous designs. This is

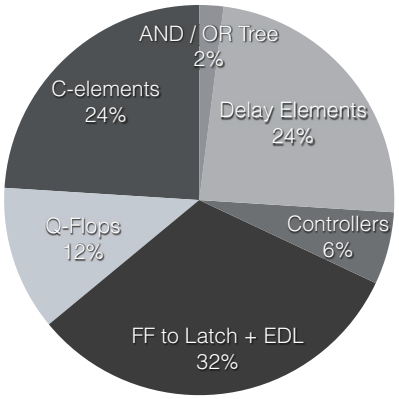


Fig. 9: Area overheads as percentage of total overhead

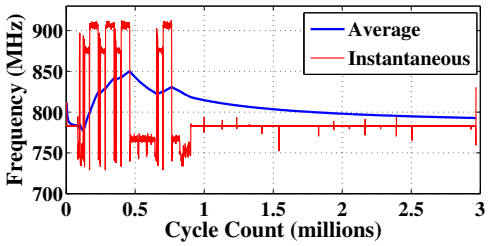


Fig. 10: Average case performance over time for Plasma CPU using Blade. Original synchronous frequency of 666MHz

in contrast to non-resilient bundled-data designs (e.g., [5]) 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 for synchronous designs; however, the performance impact on Blade is less than 13%, when considering even a 40% rate of timing violations.

V. Conclusions

This paper presents a novel asynchronous resilient design template that achieves modest performance improvements due to variations in data dependency alone. When combined with expected variation due to PVT, the performance benefits can be significant, at the cost of a less than 10% increase in area. The Blade template excels compared to other synchronous resiliency schemes and previous asynchronous approaches in the following key ways:

- Some synchronous resiliency approaches either do not handle metastability or handle it unsafely. For example, Razor has no protection from metastability, which Razor II fixes at the cost of adding synchronizers in the control path [10]. Likewise, Bubble razor fails to account for metastability, which leads to poor MTBF [12]. On the other hand, the metastability filter in the Q-Flop of the Blade stage guarantees correct operation of the circuit under metastability at the expense of performance. The stage will stall indefinitely until metastability has resolved, which is simply not possible in synchronous designs.
- As was shown in Section III-B, adding pipeline stages in an asynchronous design is straightforward and requires no architectural modifications to the original RTL. This allows enormous freedom in how the impact of difficult to handle timing paths can be mitigated. In the Plasma case study, adding a pipeline stage to the input to the RF reduced the area overhead in EDLs alone by ~67%.

As ongoing work we are exploring power characterization for Blade designs and improvements that can be obtained via voltage scaling. Because the template allows performance improvements when compared to synchronous designs, designers can trade-off these improvements with power savings through voltage scaling, achieving lower power at iso-throughput. Furthermore, Blade also motivates new areas of future work, including avenues for optimization for the average-case at the logic and architectural levels as well as new challenges in the area of automated physical design to realize these benefits post-layout. In addition, new testing strategies could be developed to both optimally tune the programmable delay lines based on in situ error rate monitoring and identify chips with delay variations too large to correct.

Acknowledgements

We would like to thank Jordi Cortadella and Luciano Lavagno for fruitful discussions. We would also like to acknowledge Fang Bai, Dailin 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 and scholarships from CAPES and CNPq.

References

[1] 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.

[2] 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*, March 2013, pp. 1715–1724.

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

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

[5] 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.

[6] 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.

[7] N. Jayakuma, R. Garg, B. Gamache, and S. Khatri, "A PLA based asynchronous micropipelining approach for subthreshold circuit design," in *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] 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.

[10] S. Das, 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 JSCC*, vol. 44, no. 1, pp. 32–48, Jan 2009.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] M. Moreira, B. Oliveira, J. Pontes, F. Moraes, and N. Calazans, "Adapting a C-element design flow for low power," in *ICECS*, Dec 2011, pp. 45–48.

[17] R. Fuhrer, B. Lin, and S. Nowick, "Symbolic hazard-free minimization and encoding of asynchronous finite state machines," in *ICCAD*, Nov 1995, pp. 604–611.

[18] K. Yun, D. Dill, and S. Nowick, "Synthesis of 3D asynchronous state machines," in *ICCD*, Oct 1992, pp. 346–350.

[19] K. Sakallah, T. Mudge, and O. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," *IEEE Trans. on CAD*, vol. 11, no. 3, pp. 322–333, Mar 1992.

[20] J. Tschanz *et al.*, "Tunable Replica Circuits and Adaptive Voltage-Frequency Techniques for Dynamic Voltage, Temperature, and Aging Variation Tolerance," in *VLSI*, 2009, pp. 112–113.