Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction

Matthew Fojtik, David Fick, Yejoong Kim, Nathaniel Pinckney, David Money Harris,
David Blaauw, *Fellow, IEEE*, and Dennis Sylvester, *Fellow, IEEE*

Abstract—We propose Bubble Razor, an architecturally independent approach to timing error detection and correction that avoids hold-time issues and enables large timing speculation windows. A local stalling technique that can be automatically inserted into any design allows the system to scale to larger processors. We implemented Bubble Razor on an ARM Cortex-M3 microprocessor in 45 nm CMOS without detailed knowledge of its internal architecture to demonstrate the technique's automated capability. The flip-flop based design was converted to two-phase latch timing using commercial retiming tools; Bubble Razor was then inserted using automatic scripts. This system marks the first published implementation of a Razor-style scheme on a complete, commercial processor. It provides an energy efficiency improvement of 60% or a throughput gain of up to 100% compared to operating with worst case timing margins.

Index Terms—Adaptive circuits, dynamic voltage and frequency scaling (DVFS), error correction, time borrowing, timing speculation, two-phase latches, variation tolerance.

I. INTRODUCTION

CONVENTIONAL synchronous digital systems require substantial timing guard bands to ensure proper operation across manufacturing and environmental variations. While manufacturing guard bands can be reduced by testing a part after production and adjusting voltage or frequency, this process is costly and still does not eliminate the guard bands for dynamic environmental variation. Traditional approaches to reduce margining at runtime include mimicking critical path delays with "canary" circuitry and using error prediction [1]–[7]. The Razor system [8], [9] proposed reducing these margins by employing in-situ timing error detection latches, and dynamically tuning the supply voltage during run time to the point where the circuit is on the edge of failure. Occasional timing failures are then corrected by replaying the operation

with greater margin. By always operating at the edge of failure, manufacturing and environmental guard bands are reduced to a minimum.

Multiple timing error detection techniques have been proposed, including Output Waveform Analysis [10], Time-Redundant Latches [11], Razor I latches[8], [9], Razor II latches [12], Transition Detector with Time Borrowing (TDTB) [13], and Double Sampling with Time Borrowing (DSTB) [13]. All focus on detecting data that arrives shortly after the clock edge and flagging it as an error. The earlier references focused on SEU detection and the later on Razor-style voltage tuning to eliminate margins.

Razor II, DSTB, and TDTB provide higher performance at lower cost than the earlier work. By reducing guard bands, they have demonstrated better than 30% energy savings [12], [13]. They also move metastability issues out of the datapath and into the error path, simplifying mitigation of this effect. These techniques have timing issues similar to pulsed latches in that they achieve high performance at the expense of a long hold time, increasing the risk of race failure. These significant hold time constraints are even more difficult to meet given worsening timing variability due to the link between speculation window and minimum delay. In addition, none of these methods have been applied to a complete commercial processor due to their architectural invasiveness.

To address these two issues we propose Bubble Razor, which uses a novel error detection technique based on two-phase latch timing and a local replay mechanism that can be inserted automatically in any design. The error detection technique breaks the dependency between minimum delay and speculation window, restoring hold time constraints to conventional values and allowing timing speculation of up to 100% of nominal delay. The large timing speculation makes Bubble Razor especially applicable to low voltage designs where timing varies exponentially with operating conditions.

The remainder of this paper is organized as follows. In Section II we review prior Razor approaches and their timing constraints. Section III presents the proposed Bubble-Razor approach. Section IV discusses a number of specific implementation issues in the Bubble-Razor method. Section V presents the silicon implementation of Bubble-Razor on an ARM Cortex-M3 processor, including silicon measurements. Finally, Section VI presents concluding remarks.

Conventional Flip-Flop Based Razor
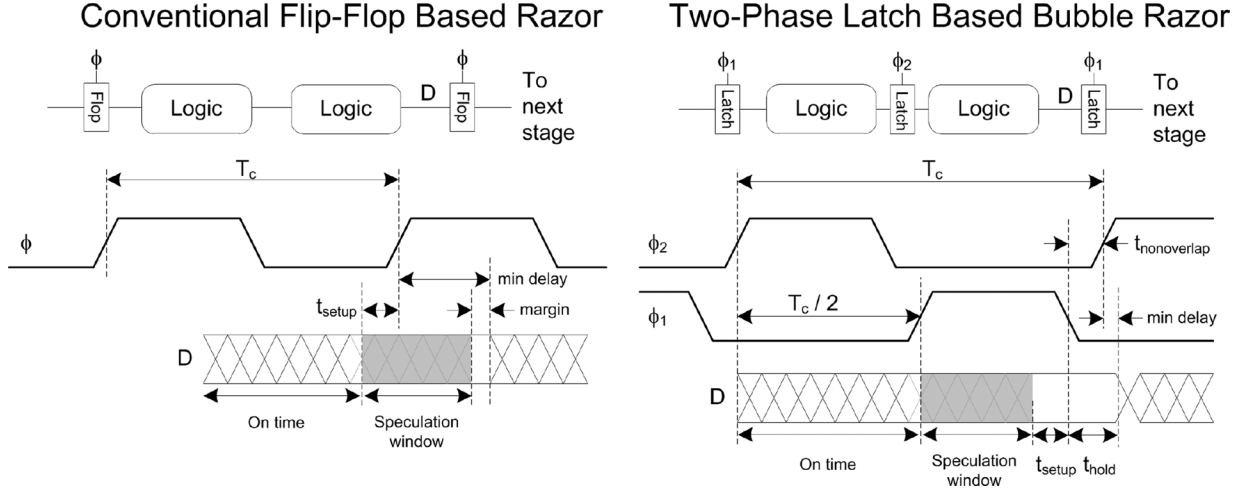
Two-Phase Latch Based Bubble Razor

Fig. 1. By using two-phase latch based timing, minimum delay constraints are restored to their conventional values allowing for large speculation windows.

II. REVIEW OF PRIOR RAZOR METHODS

A. Conventional Razor System Timing

In all conventional Razor-style systems [8], [9], [12]–[16], there is a fundamental tradeoff between speculation window and short path, or minimum delay, constraints (Fig. 1). In a system that allows 100 ps of timing speculation, when data arrives within 100 ps after the positive edge of a flip-flop or pulsed latch's clock, that data must be guaranteed to be from a long path launched from the previous clock edge. In order to ensure this, all short paths must be longer than 100 ps such that no short path can falsely trigger an error. This timing constraint must also be margined for any degree of timing variation. As Razor-style systems are targeted for situations with large timing variations, this constraint can be difficult to meet and causes large area and power increases due to buffers and other delay elements that are added to lengthen short paths. Further discussion of the timing constraints of conventional Razor-style systems is included in Appendix A.

B. Conventional Razor Error Correction Schemes

Upon detection of an error, some mechanism needs to correct for that error and allow the system to continue. Razor I [8] proposes two different styles of error correction, global clock gating and counterflow pipelining. Global clock gating involves stalling the entire processor and reloading each Razor flip flop with the correct value stored in its shadow latch. Counterflow pipelining has the error detecting stage send a bubble to downstream pipeline stages and a flush to upstream stages, which was propagated throughout the circuit one stage every clock cycle. Razor II [12] proposes another local signaling technique using architectural replay to flush the processor pipeline and replay the failing instruction, similar to how mispredicted branch instructions are handled. In order to guarantee forward progress the processor must be slowed during replay to ensure the same instruction does not repeatedly cause a timing error. In both counterflow pipelining and architectural replay, the architecture is designed with Razor in mind and the correction mechanism is built into the RTL of the design.
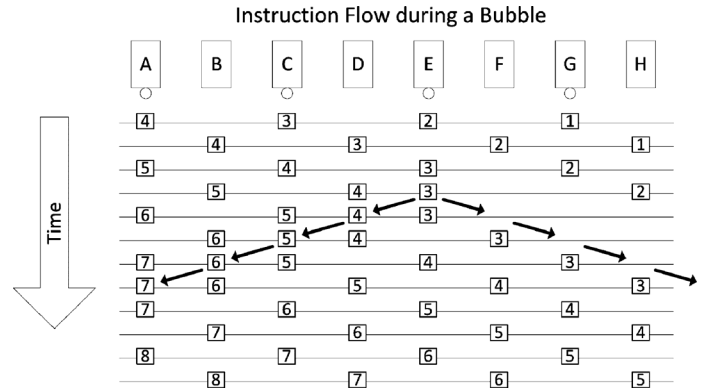
Fig. 2. In a two-phase latch based system, instructions can stall without immediately being overwritten.

Razor I's global clock gating technique is architecture independent, but its scale is limited to small designs without aggressive clock periods, as communicating a stall to the entire chip within one cycle can be impossible for large high performance designs. In all conventional Razor systems, if architecture independent stalling is used to correct for errors, it needs to be done at the global level. This is because the datapaths are based on edge-triggered flip-flops or pulsed latches, which have similar timing constraints to edge-triggered flip-flops. If one pipeline stage in a conventional Razor style system stalls, by gating its clock, the instruction held in the previous stage is lost, as every pipeline stage holds an instruction during every cycle and all of them update their state concurrently.

III. PROPOSED BUBBLE RAZOR APPROACH

A. Bubble Razor Timing

Unlike conventional Razor style systems, Bubble Razor uses a two-phase latch based datapath instead of a flip-flop based datapath. This has two main benefits: 1) it breaks the dependency between short path constraints and speculation window, enabling large speculation windows, and 2) it allows for architecture independent local correction, which can scale to large high performance systems. A flip-flop based datapath can be
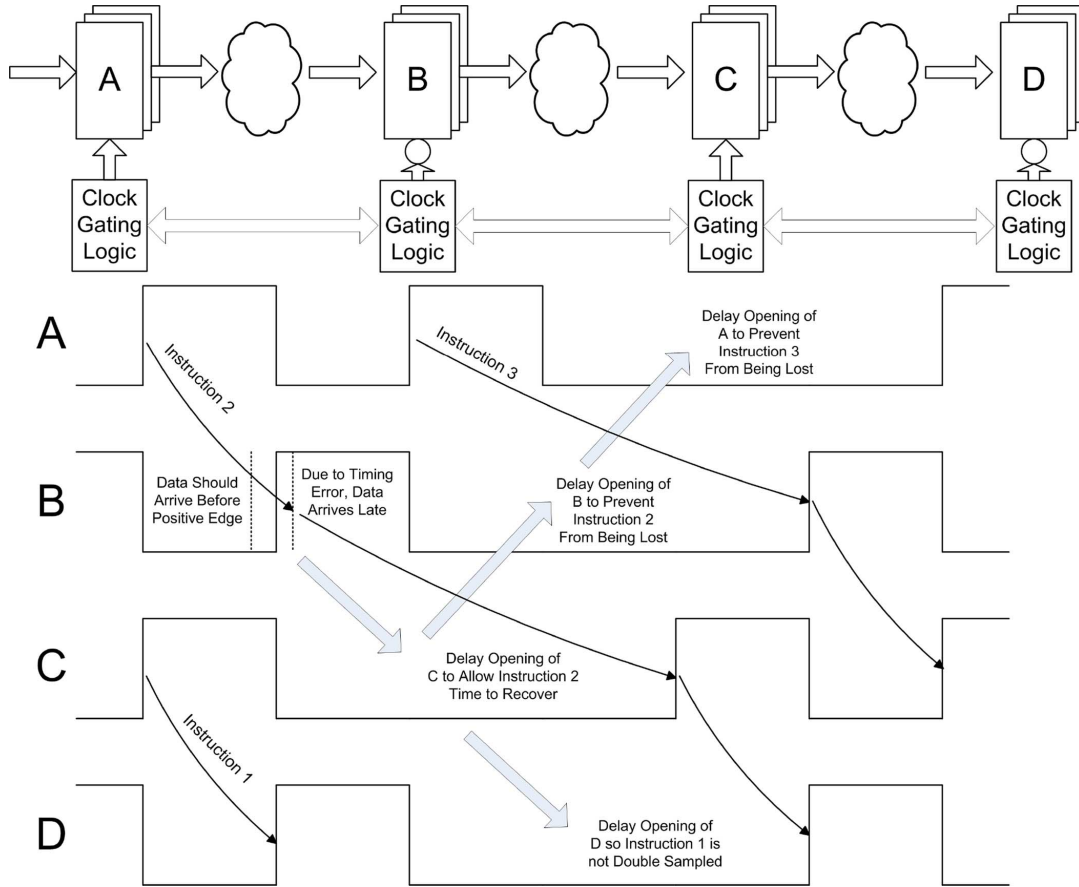
Fig. 3.   Timing errors are corrected by propagating bubbles which gate off clock pulses throughout the circuit.

converted into a two-phase latch based datapath by breaking the flip-flops into their constituent master and slave latches. By using commercial retiming tools to move the latches throughout the datapath, logic delay in each phase can be balanced such that no time borrowing occurs during error-free operation. Retiming can be performed to the same timing constraints, though the number of latches in the design may change due to retiming across gates with unequal fanins and fanouts.

During normal (error-free) operation data arrives at a latch input before the latch opens and no time borrowing occurs. If data arrives after the latch opens due to operating at the edge of failure, Bubble Razor flags an error. Unlike with flip-flop based systems, these errors are guaranteed to be caused by long paths taking more than a clock phase instead of by short paths, breaking the link between speculation window and short path constraints (Fig. 1). With a flip-flop based system, a flip-flop in one pipeline stage is clocked at the same time as the flip-flops in the preceding pipeline stage, creating the possibility of short paths being falsely flagged as timing errors. With two-phase latches, when one latch is opening the latches in the preceding stage are already closed. Thus, since new data is not being launched at that time, there is no possibility of short paths being falsely flagged as timing errors. The short path constraints in a Bubble Razor system are thus the same as in a conventional two-phase latch based system, which are easy to meet with non-overlapping clocks. This enables large speculation windows, up to 100% of circuit delay.

B. Bubble Razor Error Correction

Regarding error correction, the key observation is that errors do not immediately corrupt processor state as they borrow time from later pipeline stages. A failure will occur when data arrives after a latch closes, which can arise if the time borrowing effect is not corrected and compounds through multiple stages. Upon detection of a timing error, it is critical to recover quickly before time borrowing accumulates to a point of failure. Error clock gating control signals (bubbles) are propagated to neighboring latches (Fig. 3). A bubble causes a latch to skip its next transparent clock phase, giving it an additional cycle for correct data to arrive.

Unlike with flip-flop based systems, error correction can be accomplished by local stalling (Fig. 2). When a flip-flop stalls, data is immediately lost as its neighboring flip-flops transition their state at the same point in time. With two-phase latches, if a latch stalls, data is not immediately lost because its neighboring latches operate out of phase. In order to not lose data, neighboring latches must stall one clock phase later. Because of this time difference, the stalling can be distributed in time and only needs to be communicated to neighboring stages, stages with which data is already being communicated. Because stall signals need only be distributed to neighboring stages in the same amount of time given to communicate data, the system is scalable to processors of arbitrary size.

A key challenge lies in how to prevent bubbles from propagating indefinitely along loops and forwarding paths and bring
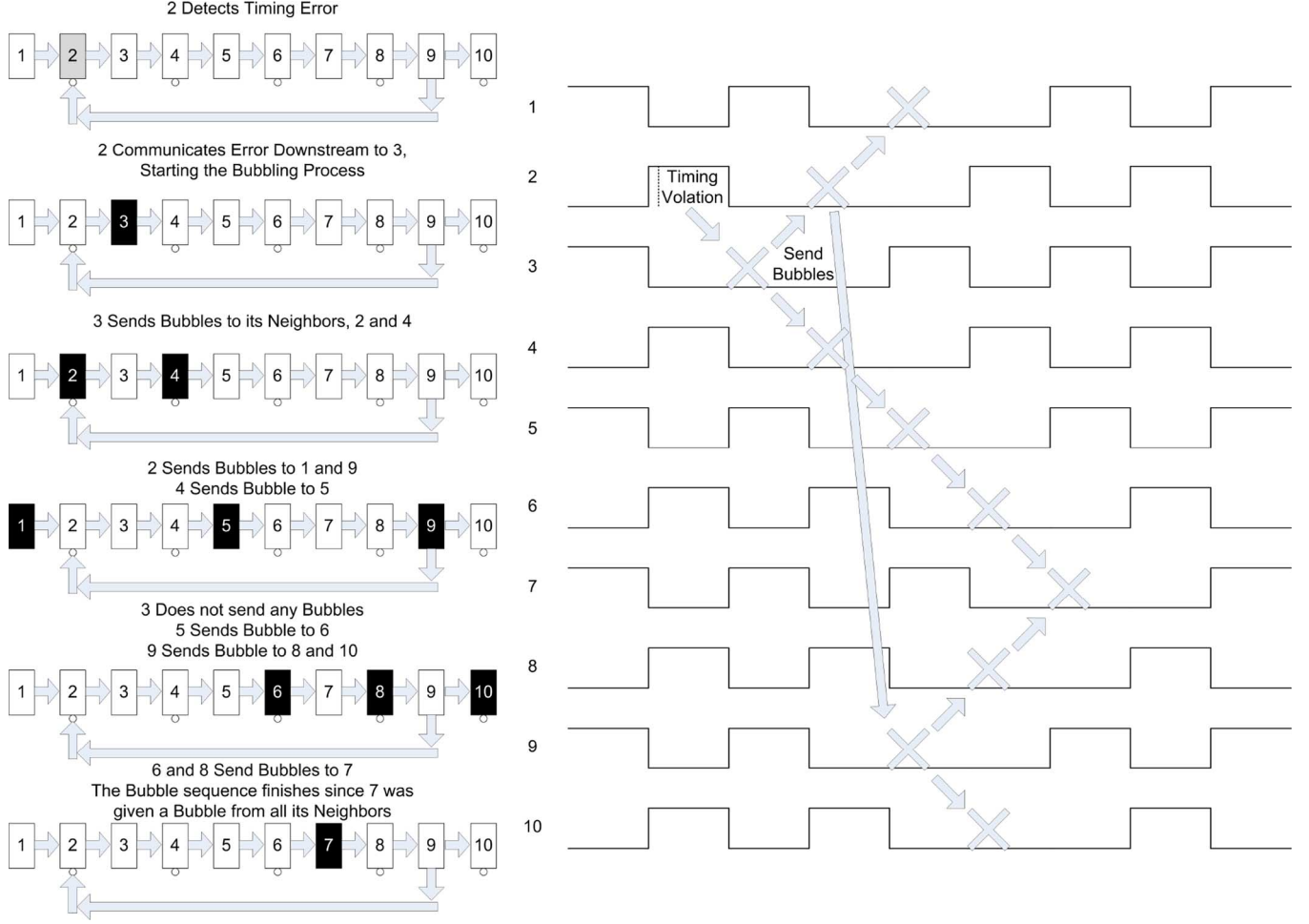
Fig. 4. Bubbles are communicated to neighboring latches. Upon error resolution, every latch has stalled for exactly one cycle.

the circuit back to a consistent, bubble-free state. To address this, we propose a novel bubble propagation algorithm: (1) a latch that receives a bubble from one or more of its neighbors stalls and sends its other neighbors (input and output) a bubble one half-cycle later; (2) a latch that receives a bubble from all of its neighbors stalls but does not send out any bubbles (Fig. 4). Despite the fact that latches stall at different times, the system maintains correct operation with every latch in the design stalling exactly once. The stalling technique is agnostic to state machine architecture or structure, allowing bubble clock gates and control logic to be automatically inserted. The only change to the external behavior of the system is an occasional single stall cycle on the inputs and outputs.

Other key questions include how the system behaves in the presence of multiple timing errors during the same cycle, the presence of multiple bubble sequences in flight at the same time, and whether forward progress is maintained during high error rates. The bubble algorithm does not need to be modified to address any of these concerns. Multiple errors during the same cycle will cause multiple bubble stalling sequences to take place at the same time, but when stall events collide they combine. The latches receiving bubbles are not aware of where the initial error occurred and they do not need to, as the stalling constraints

from each error sequence overlap. It is beneficial to have multiple bubble sequences combine as multiple timing errors can be corrected by a single stall cycle, reducing correction overhead. The algorithm is also guaranteed to make forward progress as a latch will never stall indefinitely. A latch stalls when sent a bubble by one or more neighbors and then sends a bubble to its other neighbors. An equivalent definition for bubble propagation is for the latch to send a bubble to all its neighbors but ignore bubbles if it stalled in the previous cycle. Since a latch will never stall two cycles in a row, it will always make forward progress. We have shown that the system operates correctly even with every latch reporting a timing error during every cycle. In this case, every latch spends exactly 50% of its cycles stalling.

IV. BUBBLE RAZOR IMPLEMENTATION ISSUES

A. Speculation Window Selection

A Razor-style system can be tuned such that it is running error-free but with no timing margins in order to increase system performance. At this point, the circuit is susceptible to timing errors if logic delay suddenly increases due to a voltage droop, temperature spike, or other transient event. The speculation window determines the amount this logic delay is

allowed to increase such that the system can still detect and correct errors and maintain correct operation. With Bubble Razor, as with other Razor systems, the speculation window can be limited by either the technique or the amount and location of latches with error checking. The maximum allowed speculation window is a full clock phase minus the delay of the error propagation circuitry. The theoretical maximum is therefore 100% of circuit delay, meaning correct operation could be maintained even if circuit delay suddenly doubles, although in practice the error correction circuits have non-zero delay. Because of the large allowable speculation windows, it is possible to tradeoff between speculation window and allowable time borrowing. Allowing some time borrowing can improve variation tolerance due to mismatch between stages, as well as reduce area overhead by limiting the number of latches introduced by retiming.

Placing error detection on every latch in the design is very costly in terms of area and power, and is not desirable. In addition, error detection is not required on all latches; if the critical path feeding into a latch is less than 50% of a clock phase, that latch will never experience a timing error even with a doubling of circuit delay.

Because the datapath is two-phase latch based, removing error detection from certain stages could allow time borrowing to occur without generating errors, complicating speculation window analysis. Fig. 5 shows an example of a system with a 30% speculation window. During error-free operation near the point of first failure (PoFF), the delay from Latch B to C is a full clock phase, 50% of a clock cycle, and the delay from C to D is only 20% of a clock cycle. Due to a voltage droop or other event, circuit delay becomes 130% of its nominal value such that the design is now operating at the edge of its speculation window. In this case, data does not arrive at C before it opens, however when looking at the combined path from B to D, the delay is only 91% of a clock period and data still arrives before Latch D opens. Because of the small delay between C and D, the path from B to C is able to borrow time and the timing error corrects itself without any need for bubbles. This would imply that error detection is not needed on C or D.

However, this analysis assumes that data is launched from Latch B at its opening edge. If the delay from A to B was nominally 50% of a clock cycle, and 65% after a voltage droop, then data arrives at B late and pushes back all the subsequent stages such that data arrives late at D. This multi-phase analysis is complex, even for the simple in-order pipeline shown. For a general finite state machine with loops and forwarding paths the analysis is substantially more complicated.

To simplify the process of determining where error detection is needed, we propose disallowing all undetected time borrowing. Thus, a latch assumes that data is launched at the opening edge of the latches preceding it, and determines that error detection is needed if its data arrives late under worst case conditions. In the above example, Latches B and C add error detection because the worst-case delay of their critical inputs paths are 65% of a clock cycle, which is greater than a clock phase. This analysis only requires looking at one path at a time, though it can produce a larger set of latches with error checking than strictly necessary.
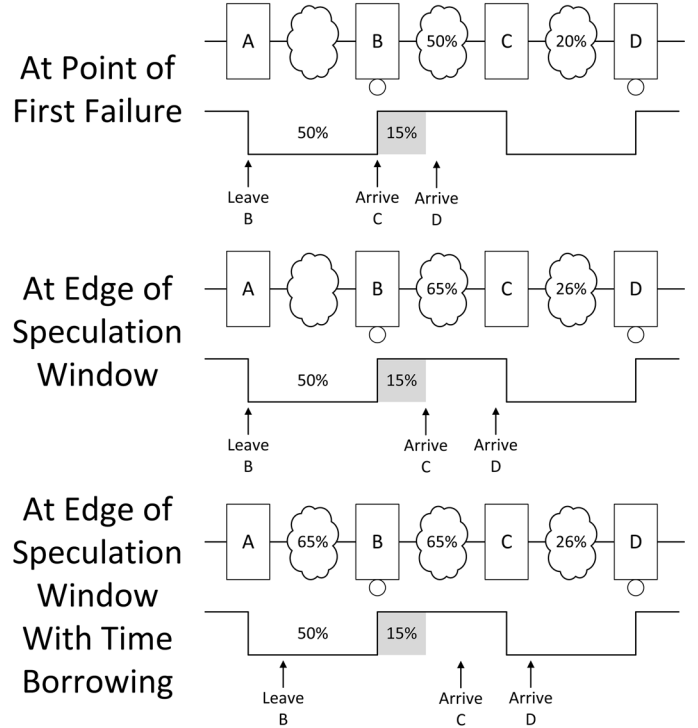

Fig. 5. When determining where error detection is needed for a given speculation window, time borrowing can complicate analysis.

B. SRAM Interface

The Bubble Razor algorithm works seamlessly for two-phase latches but adjustments need to be made when dealing with edge triggered peripherals such as SRAM. If speculative state was incorrectly written to memory that error could not be corrected for. SRAMs were treated as positive latches for the purpose of the Bubble Razor algorithm and wrapper logic was placed around SRAMs to make them behave similarly to level-sensitive latches when given a stall cycle (Fig. 6). In this implementation, the register file was synthesized logic and was transformed to two-phase latches along with the rest of the processor.

When retiming the design, negative latches are first placed on the outputs of the processor interface to memory such that the circuit is of legal configuration: all neighboring latches of the positive SRAM are negative latches. Assuming error checking occurs on the negative latches in the fanout of the SRAM, reads are constrained to operate in one clock phase. Depending on the configuration of the processor, this may introduce a tighter timing constraint. In the Cortex-M3 implementation timing was unaffected as SRAM was already operating in approximately 50% of a clock cycle with the other 50% of a clock cycle being used by combinational logic between the processor's inputs and first flip-flops.

To avoid writing incorrect data to SRAM, the system uses a commercial two-port, high-speed SRAM that separates read and write ports. Writes are clocked on the negative edge of the clock, after the speculation window, when data is guaranteed to be error free. A single entry store buffer could alternately be used to stabilize writes. Writes are disabled when the SRAM receives a bubble.
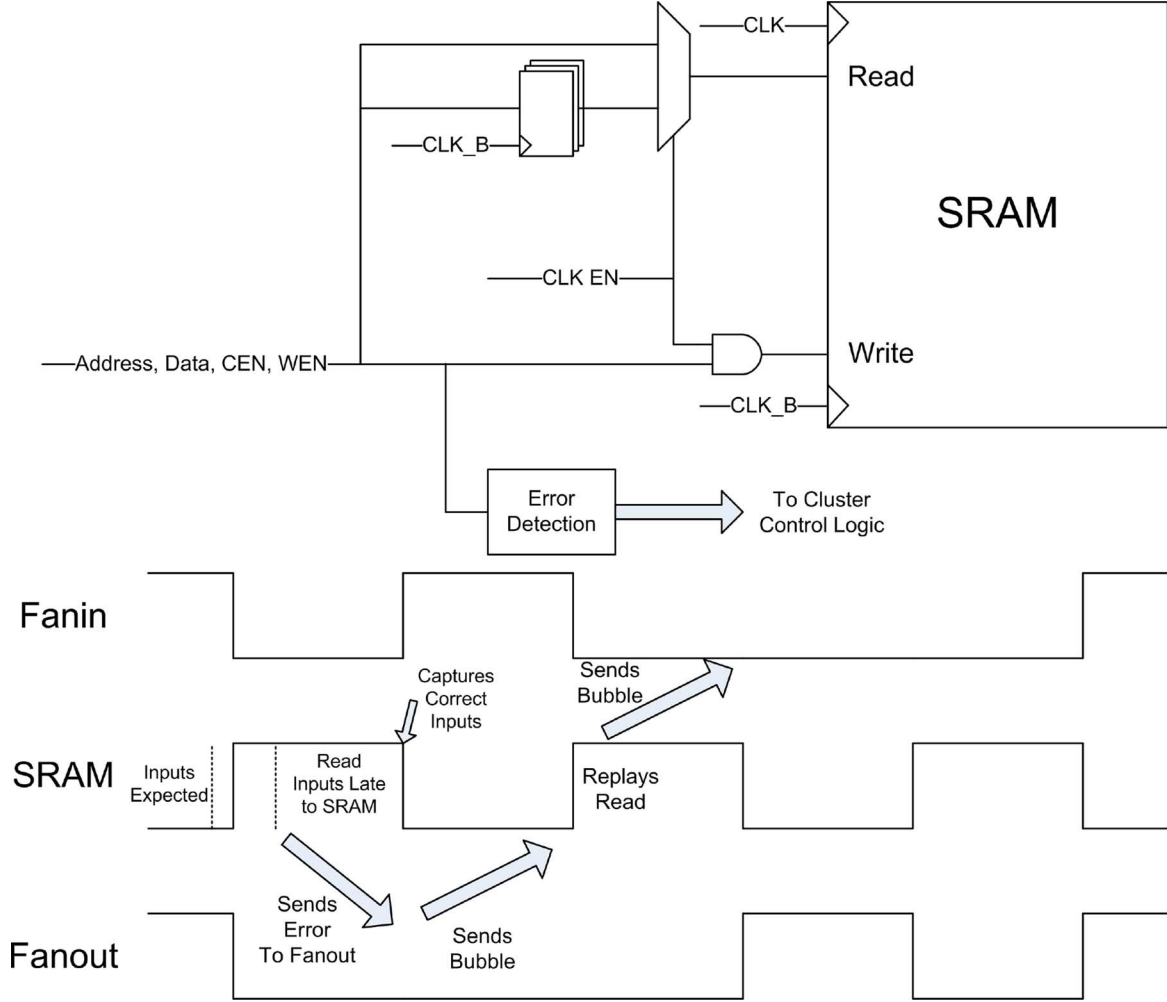
Fig. 6. Wrapper logic is placed around the SRAM such that it can be treated as a positive latch.

Since reads cannot be delayed without reducing system performance, they continue speculatively at the positive edge. If the read inputs to SRAM such as address arrive late, the SRAM would capture incorrect values at the positive edge of the clock and return the wrong data. Unlike with a level sensitive latch, this error will not automatically be corrected when given more time. Fortunately, due to the nature of the Bubble Razor algorithm, in all cases where the SRAM receives late inputs it will receive a bubble during the next cycle. Upon receiving this bubble, the SRAM uses the available cycle to repeat the read with the correct inputs that were captured by a bank of flip-flops on the negative clock edge. These approaches to handling SRAM can be automatically added to any system.

C. Latch Clustering

To reduce the logic area overhead of bubble propagation, latches that share neighbors were automatically grouped together into clusters. Latches in each cluster share a gated clock and combine their error signals into a common cluster error signal. A cluster then behaves as a single latch for the purpose of the Bubble Razor algorithm. It is possible for the designer to manually assign latches into clusters such as grouping together pipeline stages. Alternatively, we proposed an automated approach to assigning clusters. A positive and negative graph was extracted based on latch connectivity (Fig. 7). In each graph, the vertices represented the latches and the edge weights represented the number of paths through opposite polarity latches that connect the two vertices. Each latch was then assigned a cluster by inputting the graphs into a hypergraph partitioning tool [18].

Although the assignment of clusters is performed automatically, the designer choses the number of both positive and negative clusters. A tradeoff exists between the size of the OR gates needed to combine error signals within a cluster into a cluster error signal and the size of the OR gates needed to combine bubbles from neighboring clusters. With many clusters, the size of each cluster is small but each cluster has many neighbors. Alternatively, with few clusters each cluster has few neighbors but a large number of members. In the implemented design, 100 negative clusters and 70 positive clusters was chosen to balance the size of these competing OR gates.

V. BUBBLE RAZOR SILICON VERIFICATION

To demonstrate the automated and architecture independent nature of the Bubble Razor technique, it was implemented on an ARM Cortex-M3 microcontroller, a processor with which we have no knowledge of its internal architecture. Flip-flops in
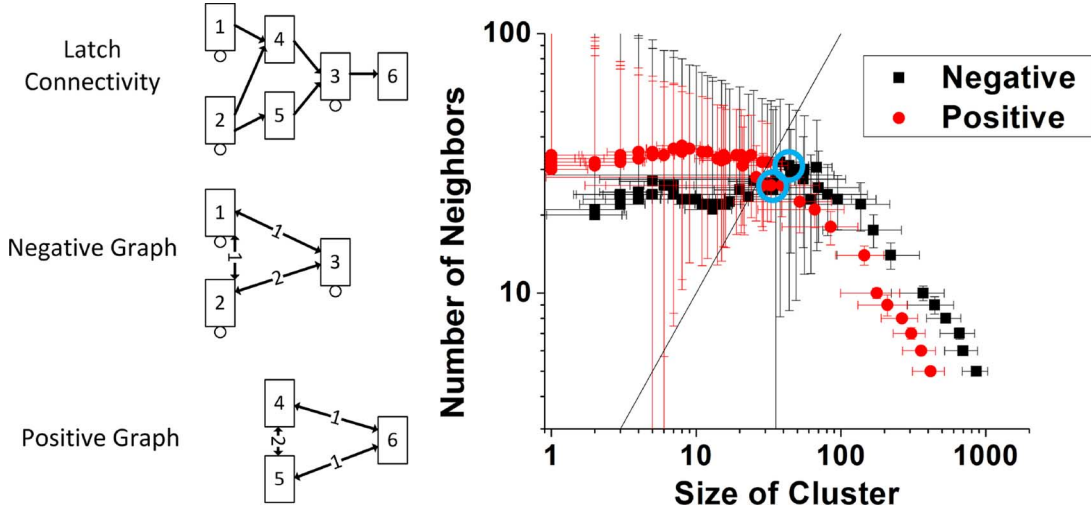
Fig. 7. Clustering was performed automatically by building two graphs based on latch connectivity. A tradeoff exists between the size of OR gates which is balanced by the choice of the number of clusters.
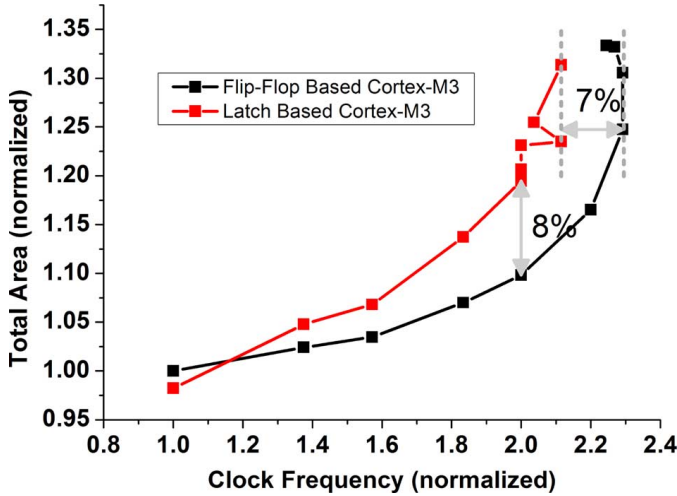

Fig. 8. Transforming the Cortex-M3 to two-phase latches can incur an 8% area penalty or 7% performance penalty.

the M3 were split into latches and the design was retimed and modified using scripts and automatic tools.

A. Retiming the Cortex-M3

Retiming the M3 was achieved by holding the positive latches in place and moving negative latches. Under ideal circumstances, this retiming can be performed with no performance penalty without modifying the combinational logic. In practice, the additional area resulting from the changing number of latches causes a small performance penalty for the design. Fig. 8 shows the results of topographical synthesis performed at various timing constraints. As the synthesis and retiming software uses heuristic based optimizations on large datasets, it is possible for it to produce non-optimal outputs as well as non-monotonic results when sweeping a variable such as target clock period. This effect is more pronounced when the software is unable to meet the target clock period and is seen in the rightmost datapoints of Fig. 8. The maximum possible operating frequency for the latch-based M3 is 7% lower than the flip-flop based M3 due to this area increase. At a reasonable

design point, the latch based M3 meets the same timing as the flip-flop based M3 but with an 8% area overhead. This operating point was chosen for further overhead analysis.

B. Speculation Window Selection for the Cortex-M3

The selection of latches which require error detection can be determined by examining the critical paths at the input of each latch. Fig. 9 shows the distribution of critical path delays for flip-flops in the original design and latches in the retimed design. As a result of only moving negative latches, 64% of the latches in design are negative. In addition, many negative latches have very low critical path delays. These low delays result from flip-flops in the original design with critical paths below 50% of a clock period, where latches do not need to be moved to meet timing.

We propose three different methods for selecting paths with error checking which make use of these timing characteristics: checking a subset of all latches, only positive latches, or only negative latches. When checking all latches, the maximum possible speculation window is 100%, while checking all positive or all negative latches would yield a speculation window of 50% since every other latch in a timing path has no error detection.

Fig. 10 shows the area overhead and speculation window results for these three techniques when applied to the Cortex-M3. When only checking positive latches, since most latches are near critical, most of the area overhead is present with small speculation windows, but futher pushing speculation window comes at low area cost. When only checking negative latches, area overhead drastically increases once the large number of latches with small critical paths require checking. Depending on the desired speculation window, either of the three techniques may be optimal. For a good design point, 30% speculation can be achieved with a retiming and error checking area overhead of 20%.

C. Implementation Circuitry

Error detection was performed using the Bubble Razor Latch, similar in design to the error detection flip-flop in [8]. A shadow latch captures data as the main datapath latch opens (Fig. 11). An XOR compares the two values and will flag an error if data

Flip Flops

All Latches

Negative Latches
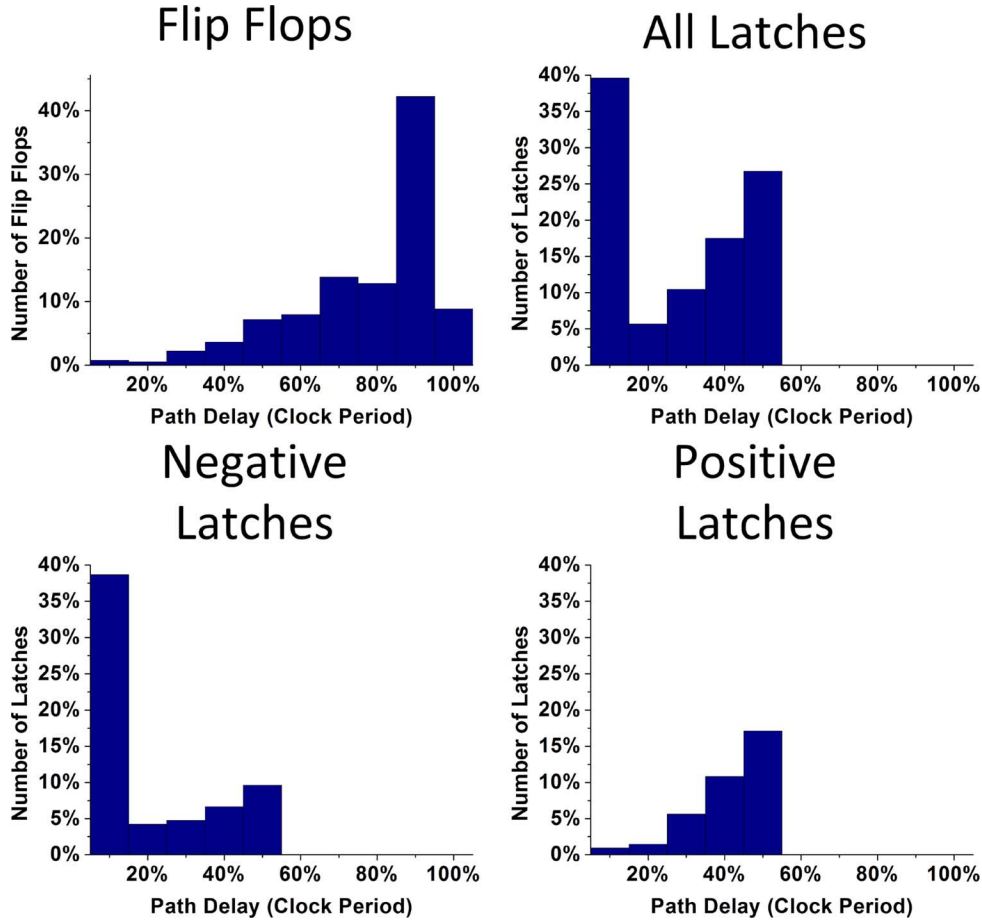

Positive Latches

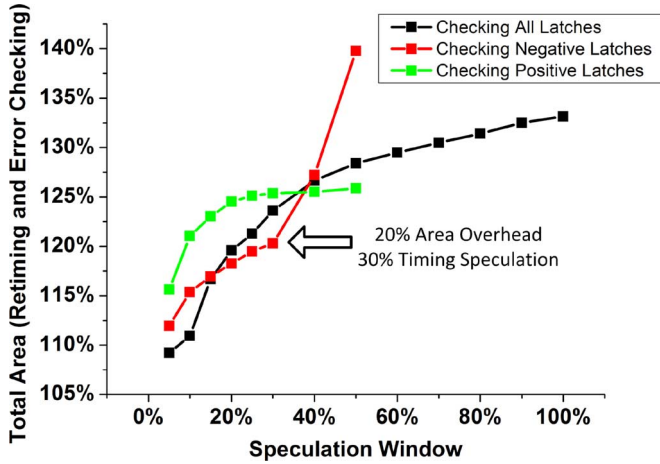Fig. 9. The Cortex-M3 has an imbalanced path distribution as a result of retiming.

Fig. 10. Error checking can be added to a subset of latches, or a subset of only positive or only negative latches, yielding different area overheads. This is due to the distribution of critical path delays at the inputs of each latch.

arrives late and changes the value in the main latch. The Bubble Razor algorithm is not dependent on the type of error checking latch, and hence transition detectors based on [12] could also be used.

Errors and bubbles are combined using wide dynamic OR gates, in our implementation made up of trees of 16 input dynamic ORs. Latches are used after trees of OR gates to hold the resulting values during the dynamic precharge phase.

Clock gating and bubble propagation is handled by the Cluster Control Logic blocks. This logic is based on the alternate definition of the Bubble Razor algorithm: when sent a bubble by one or more neighbors: stall and send a bubble to all neighbors if and only if you did not stall in the previous cycle. By using this approach, latches do not need to store which neighbors they received bubbles from, drastically reducing implementation area. Additionally, it was noted that upon initiating the bubble propagation sequence after detecting a timing error, the first clock gating event is optional, so clock gating does not take place during the first bubble.

Although the design uses dynamic cells and latch-based timing, the models given to synthesis, placement, and routing software are fully static and edge-based. Since the dynamic ORs are always followed by more ORs or a latch, the ORs are modeled as static and the latch is modeled as a flip-flop. Latches in the datapath are modeled as flip-flops, since time borrowing during error-free operation is disallowed. The resulting design appears to the tool chain as a standard, flip-flop based design with clock gating, allowing fully automated, standard integration with no designer intervention.

D. Silicon Test Chip

Bubble Razor was applied to the Cortex-M3 processor, a 1.25 DMIPS/MHz microcontroller [19], and implemented in a 45 nm SOI process. This silicon test chip is the first published Razor-style implementation to demonstrate a transformed
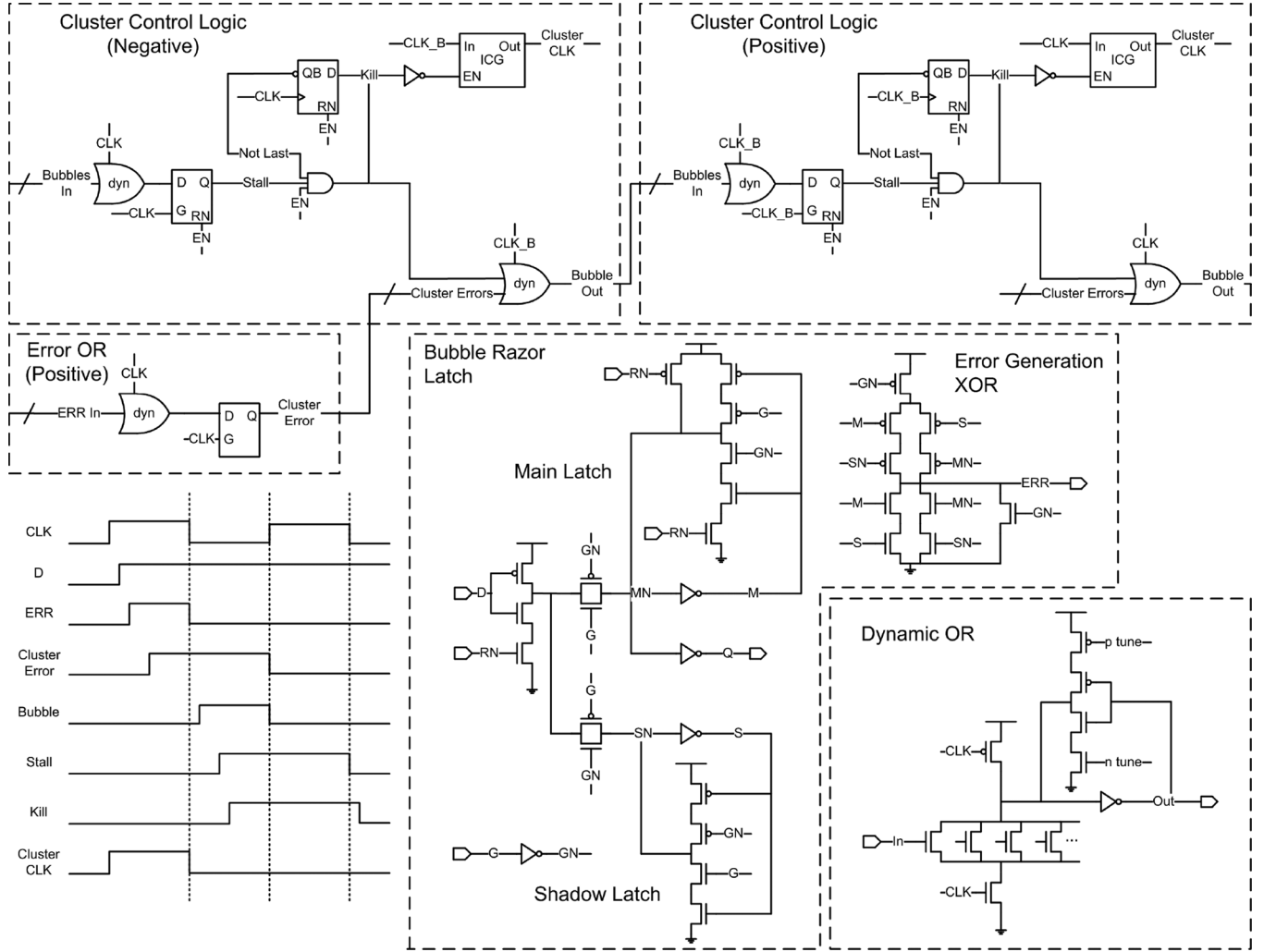
Fig. 11. Bubbles are combined using dynamic OR gates. A cluster ignores bubbles if it stalled in the previous cycle.

commercial processor operating correctly under the presence of timing errors. Several robust design decisions were made resulting in large area overheads for the silicon test chip. Timing error checking was added to all latches, even those which are not capable of failing timing, in order to allow us to find the maximum possible speculation window: one clock phase minus the propagation delay of the error detection circuits, which provides 55% timing speculation in this implementation. All latches in the design had an asynchronous reset although it is only strictly required for either all positive or all negative latches. Robust short path constraints were also put in place, and were met through buffer insertion.

These design decisions, when combined with retiming overhead, resulted in an artificially large cell area overhead of 87% for the latch based M3 compared to the original flip-flop based M3. This comprised a 21% increase in combinational logic area and a 280% increase in sequential area. The additional cluster control logic added 16% area compared to the original flip-flop design, resulting in a total area overhead over the flip-flop design of 103%. The number of gates increased from 32,805 to 36,206 when transforming to Bubble Razor, with the majority of the new cells comprising new latches as each flip-flop became

an average of 3 latches after retiming. Estimated clock loading increased by 230% with 88% of the loading coming from the Razor latches and the remainder coming from flip-flops in the JTAG test harness, latches in cluster control logic, and dynamic OR gates. Reducing the number of latches with error detection would drastically reduce the increase in sequential area, additional cluster control area, total area, and clock loading.

Synthesis results since the silicon implementation are shown in Sections V-A and V-B, which meet short path constraints with nonoverlapping clocks, only resets positive latches, and only uses Razor latches in timing critical locations. It is shown that error detection with a 30% speculation window can be achieved with 20% area overhead, which increases to approximately 25% when the additional cluster control logic is added. This area increase is for the core logic only and reduces when amortized over cache area.

E. Silicon Measurement Results

Because of the robust design decisions mentioned in Section V-D, a silicon comparison was not made between a conventional M3 and the test chip, so the silicon test chip compares against itself operating at worst case margins when
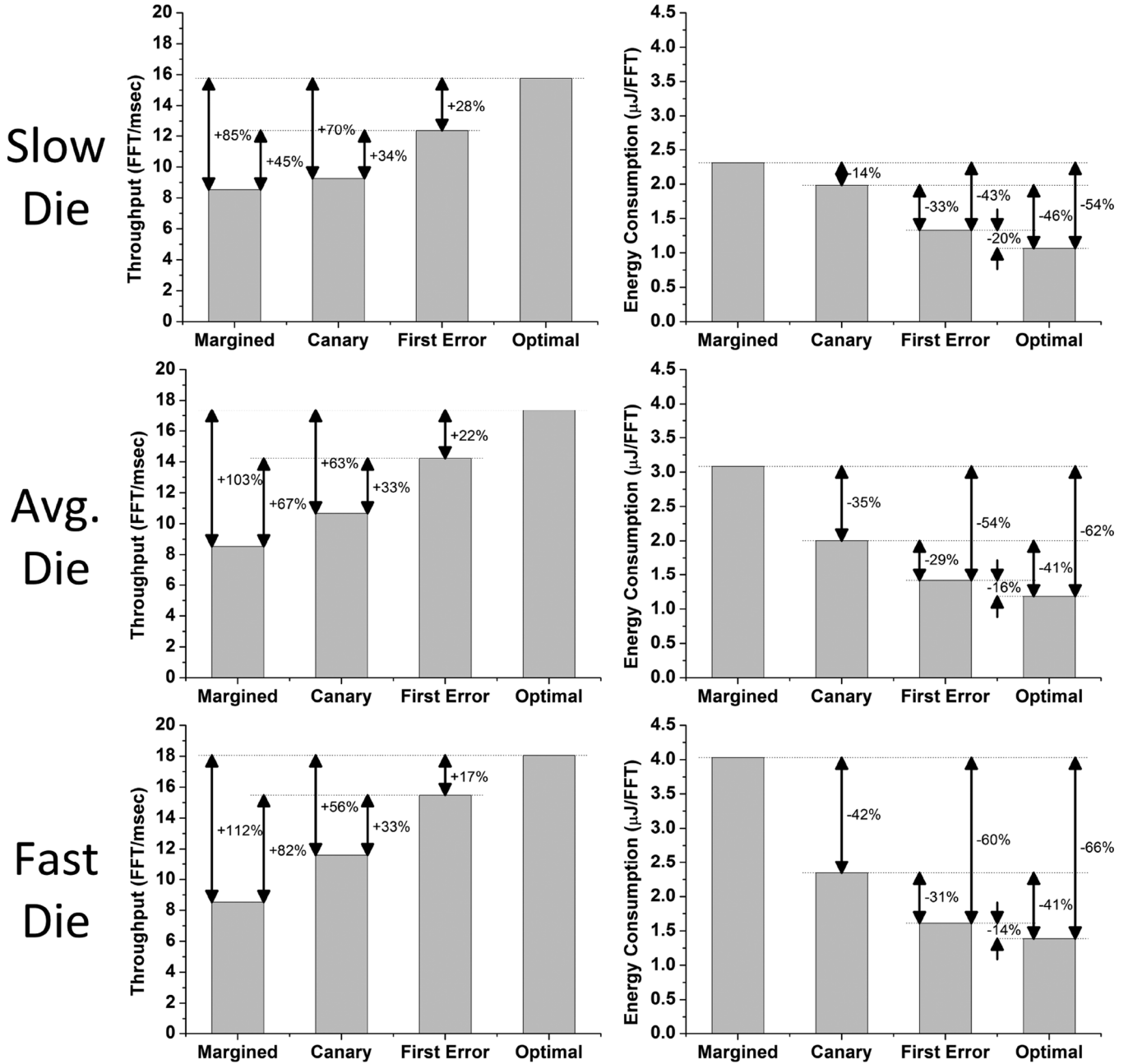
Fig. 12. By running the system under nominal conditions instead of with worst case margins, performance or energy can be improved.

calculating performance increases and energy savings. Synthesis results show the implemented test chip can operate at the same frequency as a conventional flip-flop based Cortex-M3 when designed to the same timing constrain, however the addition of Bubble Razor will come at a cost of area and power which is highly dependent on the desired speculation window as discussed in Sections V-A, V-B and V-D. Additionally, if DFT is added to a design it will come at a higher cost for the latch based Bubble Razor implementation as the scan chain will contain twice as many elements. These costs must be taken into account when calculating the energy savings from using Bubble Razor.

The silicon test chip was programmed to perform software FFT computations. At 85°C with 10% supply drop, 2σ process variation, and 5% safety margin, the maximum operating frequency of the M3 design is measured as 200 MHz, setting a frequency ceiling for a conventional margined design. With Bubble Razor the design can be tuned to the point of first failure (PoFF) which was 290/333/363 MHz for three shown chips, increasing throughput by 45, 67, and 82% (Fig. 12). Alternatively, supply voltage can be lowered at iso-performance, reducing M3 energy consumption by 43, 54, and 60%, respectively.

Fig. 13 shows system behavior when sweeping frequency or voltage beyond the PoFF. As clock frequency linearly increases, throughput initially linearly increases. As timing errors become more prevalent at higher frequencies, throughput improvement slows down and eventually reverses due to stall cycles consuming a large portion of processor runtime. Similarly, voltage scaling reduces energy consumption until timing errors become too common. When running at a voltage substantially lower than the PoFF, the large number of stall cycles cause the program to take longer to execute which increases total energy consump-
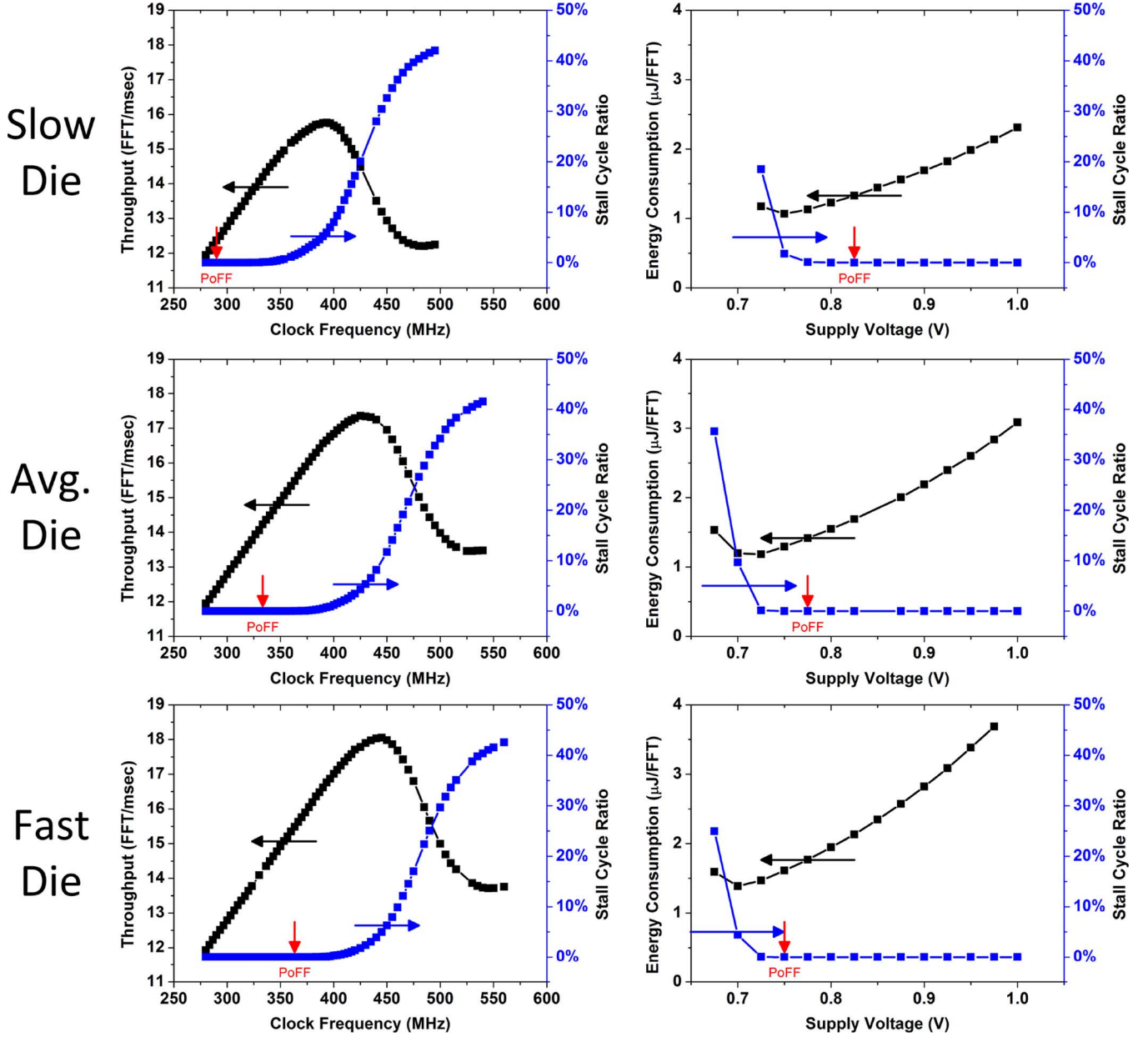
Fig. 13. Due to only a single cycle penalty for fixing timing errors, an additional 22% performance gain or 17% energy reduction can be made by running beyond the Point of First Failure.

TABLE I
TERMINOLOGY.

| Symbol	Description
T_c	Clock Period
$t_{pcq(f,l)}$	Clk-Q propagation delay for (flip-flop, latch)
$t_{ccq(f,l)}$	Clk-Q contamination delay for (flip-flop, latch)
t_{pdql}	D-Q propagation delay for latch
$t_{setup(f,l)}$	Setup time for (flip-flop, latch)
$t_{hold(f,l)}$	Hold time for (flip-flop, latch)
$t_{nonoverlap}$	Non-overlap time for two phase latch clocks
t_{pw}	Width of pulse in pulse latch system

tion. If frequency or voltage is scaled too far, the system will begin operating outside of its speculation window, timing errors will not be properly corrected for, and the system will fail.

All points in Fig. 13 represent the system executing its program correctly with the rightmost throughput and leftmost energy points representing limits of frequency and voltage scaling. Overall, an additional 22% performance or 17% energy reduction is obtained from running beyond the PoFF. This is significantly better than previous Razor approaches since only a single cycle is lost per corrected error, allowing beneficial operation at relatively high error rates. The combination of eliminating margins and running beyond the PoFF allows for a 100% throughput increase or 60% energy reduction when compared to operating with worst case timing margins.

We used ring oscillators on each chip as "canary" circuits to provide for a comparison of energy/performance gains from canary circuits and with Bubble Razor. Canary circuits allow some timing margins to be reduced, but cannot eliminate all margins as there may be mismatch between the canary and datapath. In

TABLE II
TIMING CONSTRAINTS FOR CONVENTIONAL SYSTEMS.

System	t_{pd}	t_{cd}	t_{borrow}
Flip-flop	$T_c - t_{pcqf} - t_{setupf}$	$t_{holdf} - t_{ccqf}$	0
Two-Phase Latch	$T_c - 2 * t_{pdql}$	$t_{holdl} - t_{ccql} - t_{nonoverlap}$	$T_c/2 - t_{setupl}$
Pulse Latch	$T_c - t_{pdql} - max(0, t_{setupl} - t_{pw})$	$t_{pw} + t_{holdl} - t_{ccql}$	$t_{pw} - t_{setupl}$
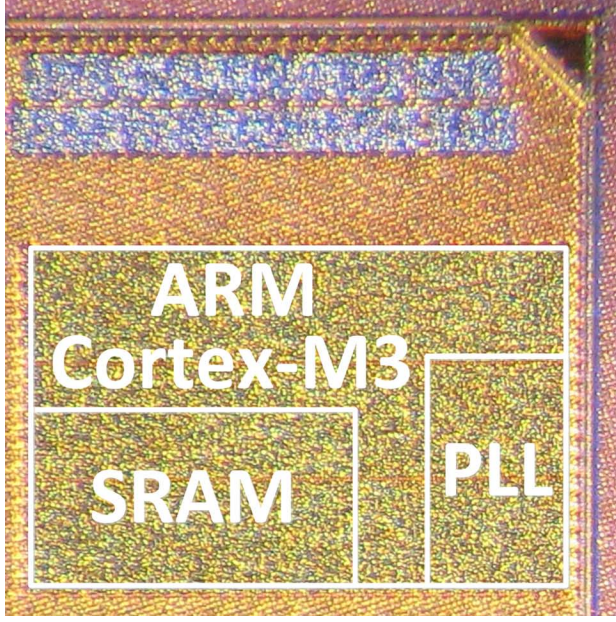
Processor Core	ARM Cortex-M3
Process Technology	IBM 45nm SOI12S0
Nominal VDD	1.0 V
SRAM Size	16 kB
Latches	7159
Positive Clusters	70
Negative Clusters	100
Speculation Window	55%

Fig. 14.   Die photo and system information.

addition, canary circuits can only adapt to slow changing operating conditions due to the time required to change the processor clock frequency, and thus canary circuits cannot eliminate the margins for supply droop. Adding margin for 3s of mismatch between the canary frequency and processor frequency, a margin for 10% supply droop, and an additional 5% safety margin, the design can be tuned to 217/250/272 MHz for the three shown chips. Running with Bubble Razor at the optimal throughput point provides gains of 70%, 63%, and 56% respectively when compared to running with canary circuits. Equivalently, Bubble Razor at the optimum energy point provides gains of 46%, 41%, and 41% over canary circuits.

VI. CONCLUSION

A novel Razor style technique was proposed that breaks the link between speculation window and minimum delay constraints, allowing large speculation windows. In addition, a local stalling technique was proposed that is independent of design architecture and scalable to designs of arbitrary size. Bubble Razor was successfully applied to the ARM Cortex-M3 microprocessor, the first Razor style implementation of a complete commercial processor. A test chip was fabricated in 45 nm CMOS to validate the technique and showed a 100% throughput improvement or 60% energy savings over running with worst-case timing margins.

APPENDIX

This appendix compares the timing constraints of prior Razor-style systems to Bubble Razor.

A. Conventional System Timing

The timing constraints of timing error detection sequencing systems have many similarities to conventional systems. The maximum logic delay (propagation delay t_{pd}), minimum logic delay (contamination delay t_{cd}), and maximum allowable time borrowing t_{borrow} for three conventional synchronization systems are summarized in Table II [20].

Two phase latch and pulse latch systems allow for time borrowing to help deal with unbalanced delays and clock skew. In all three systems, contamination delay is small and manageable, though in the pulse latch system it is proportional to t_{pw}, which creates a tradeoff between time borrowing and contamination delay.

B. DSTB Timing Constraints

This section explains the timing constraints for DSTB [13]. TDTB has similar timing, with the setup time of the flip-flop analogous to the setup time of the TD. Fig. 15 shows the timing diagram for a DSTB system.

The primary datapath resembles an ordinary pulse-latched sequencing system [21]. Each pipeline stage contains a pulsed latch followed by combinational logic. However, the data input D is also sampled by a flip-flop on the rising edge of the pulse ϕ_p. If D misses the flip-flop, it is considered late. If D is slightly late, as shown in the gray region, the pulsed latch will sample D correctly even though the flip-flop misses D. The difference is detected by the XOR, which generates an error signal. If D is too late, both the pulsed latch and flip-flop will miss the data and
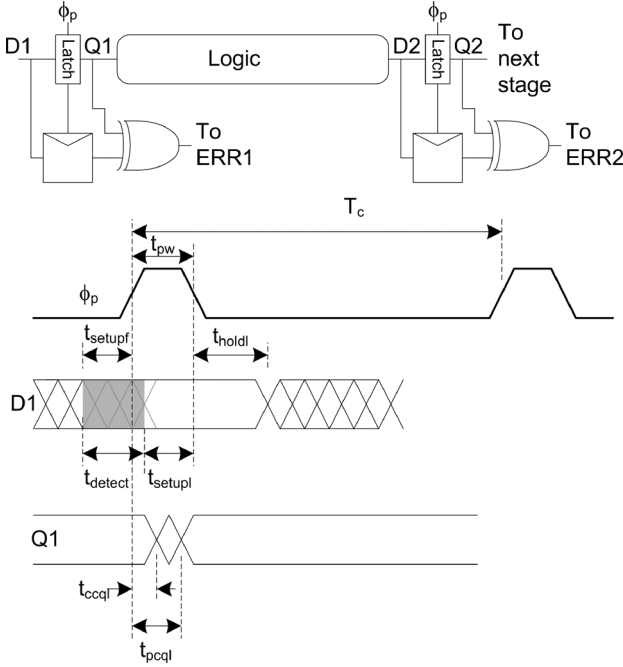
Fig. 15. Timing Diagram for DSTB System.

an undetected error will occur. The latch, flip-flop, and XOR together form a DSTB register. Errors from each register are combined to produce error signals for each pipeline stage and for the overall system.

1) Detection Window: D must stabilize at least t_{setupf} before the rising edge of ϕ_p to be sampled correctly by the flip-flop. But as long as it stabilizes t_{setupl} before the falling edge of ϕ_p, the late data will be detected. In summary, DSTB has a detection window t_{detect} in which it can detect late-arriving data:

$$t_{detect} = t_{pw} + (t_{setupf} - t_{setupl}). \tag{1}$$

Notice that this detection window grows with the pulse width. The detection window should be as wide as possible (e.g 25–30% of the cycle) to eliminate large guard bands.

2) Propagation Delay: In normal operation, there should be no late data. Hence, D should arrive safely before the rising clock edge and will propagate through the latch upon the rising edge of the clock. It will then propagate through the logic and setup at the next flip-flop before the next rising clock edge. The maximum logic delay in a cycle is

$$t_{pd} \le T_c - t_{pcql} - t_{setupf}. \tag{2}$$

This is similar to the logic delay in a flip-flop based system.

3) Contamination Delay: The input to the next pulsed latch must not change until at least a hold time after the end of the pulse. The minimum logic delay is

$$t_{cd} \ge t_{pw} + t_{holdl} - t_{ccql}. \tag{3}$$

Note that the contamination delay grows with the pulse width. This is the same as the hold time constraint in a pulsed-latch based system. Contamination delay problems are difficult to solve and catastrophic if they occur, so the pulse is normally

made quite narrow (ideally to permit $t_{cd} = 0$). This runs contrary to the desire for a wide detection window.

4) Clock Skew: Clock skew reduces the maximum propagation delay and increases the maximum contamination delay by t_{skew} in (2) and (3) [21].

5) Time Borrowing: Because the flip-flop samples on the same edge that the latch becomes transparent, no time borrowing is possible. If the clock to the flip-flop were delayed by t_d, the time available for borrowing (or skew tolerance) would become

$$t_{borrow} = t_d - t_{setupf}. \tag{4}$$

The detection window reduces by this delay:

$$t_{detect} = t_{pw} + (t_{setupf} - t_{setupl}) - t_d$$
$$= t_{pw} - t_{setupl} - t_{borrow}. \tag{5}$$

The delay also makes it possible to hide the flip-flop setup time from the critical path:

$$t_{pd} \le T_c - t_{pcql} - \max(t_{setupf} - t_d, 0). \tag{6}$$

C. Razor II Timing Constraints

This section explains the timing constraints for Razor II [12]. Fig. 16 shows the timing diagram for a Razor II system. As with DSTB, the primary datapath resembles a pulse-latched system. The error path involves feeding the internal latch node N into a transition detector T_D which is enabled by a detection clock DC. The transition detector is always enabled except for a small window t_{dc} in which N transitions during normal operation. If a transition on N occurs outside of this small window, an error is flagged. t_{dc} plays a role analogous to t_d. DC is kept high even when the latch is opaque in order to detect soft errors. If D arrives too late and misses the latch, the error will go undetected even though DC is high.

1) Detection Window: To ensure that normal transitions are not flagged as errors, t_{dc} must be greater than the longest clock-to-N delay, t_{pcnl}. The difference between these two values is the amount of time borrowing allowed by the system.

$$t_{borrow} = t_{dc} - t_{pcnl}. \tag{7}$$

A transition on D within t_{borrow} after the rising edge of ϕ_p will not be flagged as an error. To be correctly sampled by the latch, D must stabilize at least t_{setupl} before the falling edge of ϕ_p. Razor II therefore has a detection window t_{detect}:

$$t_{detect} = t_{pw} - t_{setupl} - t_{borrow}. \tag{8}$$

As with DSTB, this window grows with pulse width and shrinks with time borrowing.

2) Propagation Delay: In normal operation with no late data, D will arrive as the clock edge is rising, so the maximum logic delay in a cycle is

$$t_{pd} \le T_C - t_{pcql}. \tag{9}$$

3) Contamination Delay: The contamination delay for Razor II is identical to DSTB:
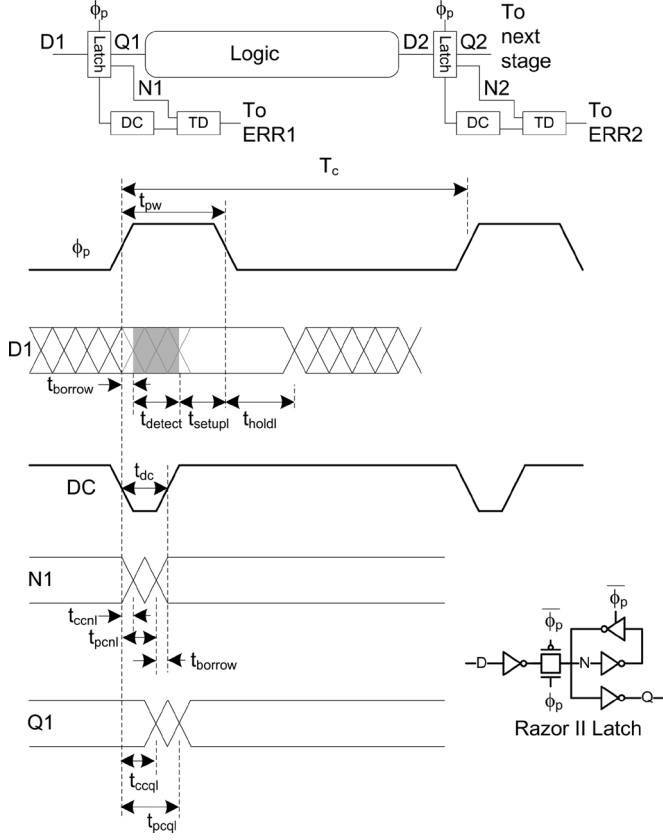
$$t_{cd} \ge t_{pw} + t_{holdl} - t_{ccql}. \tag{10}$$

Fig. 16.   Timing Diagram for Razor II System.

Fig. 17.   Timing Diagram for Bubble Razor System.

4) Clock Skew: Clock skew increases the maximum contamination delay by t_{skew} as with DSTB, but $t_{skew} < t_{borrow}$ can be tolerated for propagation delay.

5) System Operation: Suppose that the system normally operates correctly at some clock period T_1. However, under unusual circumstances, the worst-case period required is T_2. As long as $t_{detect} > T_2 - T_1$, the system can be clocked at T_1 and yet catch and replay the unusual timing errors.

Some systems offer time borrowing to balance logic between uneven pipeline stages and to opportunistically compensate for variations and skew [20]. However, both DSTB and Razor II have the poor hold time constraints of a pulsed-latch system. The detection window suffers from an unpleasant tradeoff between contamination delay and the detection window because both are linked to the pulse width. This severely limits the detection window.

D. Bubble Razor Timing Constraints

This paper proposes adding a mid-cycle latch to the DSTB system and using clocks with roughly 50% duty cycles. In the same way that two-phase latches eliminate the hold time problems in pulsed latch systems and provide time borrowing [21], the proposed sequencing methodology increases the detection window, eliminates or greatly simplifies hold time problems, and permits time borrowing to balance logic and compensate for further variation. The improvements come at the cost of the added latches in each pipeline stage.
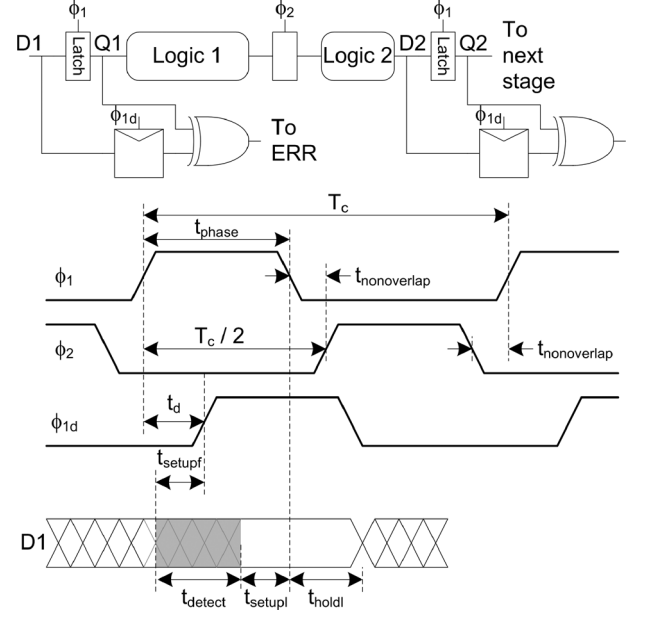
1) Two-Phase Timing Diagram: Fig. 17 shows a timing diagram for the proposed system. In the most general case, the latches are controlled by two-phase non-overlapping clocks that are high for $t_{phase} = T_c/2 - t_{nonoverlap}$ and the flip-flop clock is delayed by t_d. The primary datapath now resembles an ordinary two-phase latch sequencing system [21]. Each pipeline stage contains a ϕ_1 transparent latch, approximately half of the combinational logic, a ϕ_2 transparent latch, and the remainder of the combinational logic. However, D is also sampled some delay after the rising edge of phase 1. When the system is operating at low frequency, data arrives at each latch while it is opaque and waits until the latch becomes transparent. As the frequency increases, data may arrive at some latches after they become transparent. This is called time borrowing. As the frequency increases further, the data will miss the setup time of the flip-flop and an error will be detected. At even higher frequencies, both the latch and flip-flop will miss the data and an undetectable error occurs.

2) Detection Window: The detection window is now related to the width of phase rather than of a short pulse. However, the delay used for time borrowing cuts into the detection window:

$$t_{detect} = t_{phase} - t_d + (t_{setupf} - t_{setupl}). \qquad (11)$$

This detection window can be substantially wider than in Razor II because it does not trade against contamination delay.

3) Propagation Delay: In normal operation at maximum frequency, data arrives at each latch while it is transparent so it does not have to wait. In the absence of time borrowing, the sum of the propagation delays through the two blocks of logic, $t_{pd} = t_{pd1} + t_{pd2}$, must be less than one cycle minus the two latch delays. Hence, the maximum logic delay in a cycle is

$$t_{pd} \leq T_C - t_{pdql1} - t_{pdql2}. \qquad (12)$$

This is analogous to the logic delay in a two-phase latch based system and is similar to the performance of DSTB. The system

faces the same issues of balancing logic between phases that an ordinary two-phase latch based system has.

4) Contamination Delay: A pipeline stage has two hold time constraints, one at each latch. The data may depart a latch on the rising edge of the phase and arrive at the next latch after the contamination delay of the latch. This arrival time must be at least a hold time after the following latch became opaque:

$$t_{cd1} \geq t_{holdl2} - t_{ccql1} - t_{nonoverlap}$$
$$t_{cd2} \geq t_{holdl1} - t_{ccql2} - t_{nonoverlap}. \quad (13)$$

This is identical to the hold time constraint in an ordinary two-phase latch based system. Even if the two phases are complementary clocks with zero nonoverlap, the constraint is relatively easy to meet.

5) Time Borrowing: The system may only borrow time past the rising edge of ϕ_1 until the flip-flop misses its setup time:

$$t_{borrow} = t_d - t_{setupf}. \quad (14)$$

Hence, the detection window and time borrowing directly trade off against each other as with the other techniques.

$$t_{detect} = t_{phase} - t_{setupl} - t_{borrow}. \quad (15)$$

In an ordinary two-phase latch based system, no detection window is provided, so all of the time is available for borrowing. In this adaptive two-phase system, part of the phase is allocated for detection and part for time borrowing.

The system may only borrow substantially more time past the rising edge of ϕ_2.

$$t_{borrow-midcycle} = t_{phase} - t_{setupl}. \quad (16)$$

However, the designer should not exploit this full amount of time borrowing because a late input to a phase 2 latch cannot be detected. Specifically, the maximum borrowing should be reduced by t_{detect} so that timing errors detectable at the phase 1 latch do not result in undetected errors at the phase 2 latch. Subtracting (15) from (17) gives a more conservative borrowing limit that allows the full use of the detection window.

$$t_{borrow-midcycle} = t_d - t_{setupf}. \quad (17)$$

Note that this is exactly the same as the borrowing past the rising edge of ϕ_1 given in (14).

6) Clock Skew: As with Razor II, if $t_{borrow} > t_{skew}$, clock skew can be tolerated so that it does not cut into the propagation delay. Skew does increase the contamination delays necessary in each phase.

7) Summary: In summary, two-phase adaptive latches are much like regular two-phase latches. Their performance is better than flip-flops because they can tolerate some skew, but worse than pulsed latches because they have a second latch in the critical path. Their hold time difficulties are minor compared to pulsed latches. The first phase is divided into a first portion available for time borrowing and a second part portion for detecting late inputs.

REFERENCES

[1] 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 *Symp. VLSI Circuits Dig., 2009*, Jun. 2009, pp. 112–113.

[2] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala, "A distributed critical-path timing monitor for a 65 nm high-performance microprocessor," in *IEEE ISSCC 2007 Dig. Tech. Papers*, Feb. 2007, pp. 398–399.

[3] K. Hirairi, Y. Okuma, H. Fuketa, T. Yasufuku, M. Takamiya, M. Nomura, H. Shinohara, and T. Sakurai, "13% power reduction in 16 b integer unit in 40 nm CMOS by adaptive power supply voltage control with parity-based error prediction and detection (pepd) and fully integrated digital LDO," in *IEEE ISSCC 2012 Dig.*, Feb. 2012, pp. 486–488.

[4] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.

[5] M. Nakai, S. Akui, K. Seno, T. Meguro, T. Seki, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumano, and M. Shimura, "Dynamic voltage and frequency management for a low-power embedded microprocessor," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 28–35, Jan. 2005.

[6] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. I. Ishii, T. Y. Nguyen, and J. L. Burns, "A 32-bit powerpc system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1441–1447, Nov. 2002.

[7] S. Dhar, D. Maksimovi, and B. Kranzen, "Closed-loop adaptive voltage scaling controller for standard-cell ASICs," in *Proc. ISLPED '02*, 2002, pp. 103–107.

[8] 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 *Proc. 36th IEEE/ACM Int. Symp. Microarchitecture, 2003. MICRO-36*, Dec. 2003, pp. 7–18.

[9] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning DVS processor using delay-error detection and correction," *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 792–804, Apr. 2006.

[10] P. Franco and E. J. McCluskey, "Delay testing of digital circuits by output waveform analysis," in *Proc. Int. Test Conf., 1991*, Oct. 1991, p. 798.

[11] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *Proc. 17th IEEE VLSI Test Symp.*, 1999, pp. 86–94.

[12] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw, "Razor II: In situ error detection and correction for PVT and SER tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan. 2009.

[13] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S.-L. L. Lu, T. Karnik, and V. K. De, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 49–63, Jan. 2009.

[14] D. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. Blaauw, "A power-efficient 32 bit ARM processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 18–31, Jan. 2011.

[15] K. A. Bowman, J. W. Tschanz, S. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, Jan. 2011.

[16] R. Pawlowski, E. Krimer, J. Crop, J. Postman, N. Moezzi-Madani, M. Erez, and P. Chiang, "A 530 mV 10-lane SIMD processor with variation resiliency in 45 nm SOI," in *IEEE ISSCC 2012 Dig.*, Feb. 2012, pp. 492–494.

[17] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester, "Bubble Razor: An architecture-independent approach to timing-error detection and correction," in *IEEE ISSCC 2012 Dig.*, Feb. 2012, pp. 488–490.

[18] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. 34th Annu. Design Autom. Conf., DAC '97*, New York, NY, 1997, pp. 526–529.

[19] ARM Cortex-M3 [Online]. Available: http://www.arm.com/products/processors/cortex-m/cortex-m3.php
[20] D. Harris, *Skew-Tolerant Circuit Design*. San Francisco, CA: Morgan Kaufmann, 2001.
[21] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA: Addison-Wesley, 2010.

David Money Harris is a Professor of Engineering at Harvey Mudd College and Technical Director at Broadcom. He received the Ph.D. from Stanford University and the S.B. and M.Eng. degrees from the Massachusetts Institute of Technology.

Dr. Harris is the author of *CMOS VLSI Design*, *Logical Effort*, *Skew-Tolerant Circuit Design*, and *Digital Design and Computer Architecture*. He holds about a dozen patents in the field and has designed chips at Sun Microsystems, Hewlett-Packard, and Intel Corporation.

Matthew Fojtik received the B.S. and M.S. degrees in electrical engineering in 2008 and 2010 from the University of Michigan, Ann Arbor, where he is pursuing the Ph.D.

His research focuses on architecture independent timing speculation techniques. Additionally, he is interested in and has been involved in the design of several ultra-low power VLSI systems.

Mr. Fojtik is a recipient of an Intel Foundation/SRCEA Fellowship.

David Fick received the B.S.E. degree in computer engineering in 2006, the M.S.E. degree in computer science and engineering in 2009, and the Ph.D. degree in computer science and engineering in 2012, all from the University of Michigan, Ann Arbor.

He is currently a Research Associate at the Michigan Integrated Circuits Lab (MICL). He has authored three patents and published more than a dozen papers. His research interests include fault tolerance, adaptive circuits and systems, and 3D integrated circuits.

Yejoong Kim received the B.S. degree in electrical engineering from Yonsei University, Seoul, Korea, in 2008, and the M.S. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, in 2012. He is currently working toward the Ph.D. degree at the University of Michigan.

His research interests include subthreshold circuit designs, ultra low-power SRAM, and the design of millimeter-scale computing systems and sensor platforms.

Nathaniel Pinckney is pursuing the Ph.D. degree at the University of Michigan, Ann Arbor, specializing in low-power VLSI design. He received the B.S. degree in 2008 from Harvey Mudd College, and the M.S. degree in 2012 from the University of Michigan.

He has also worked for Sun Microsystems, Oracle, Qualcomm, and Applied Minds.

David Blaauw received the B.S. degree in physics and computer science from Duke University in 1986, and the Ph.D. in computer science from the University of Illinois, Urbana, in 1991.

Until August 2001, he worked for Motorola, Inc. in Austin, TX, were he was the manager of the High Performance Design Technology group. Since August 2001, he has been on the faculty at the University of Michigan where he is a Professor. He has published over 350 papers and hold 40 patents. His work has focussed on VLSI design with particular emphasis on ultra low power and high performance design.

Prof. Blaauw was the Technical Program Chair and General Chair for the International Symposium on Low Power Electronic and Design. He was also the Technical Program Co-Chair of the ACM/IEEE Design Automation Conference and a member of the ISSCC Technical Program Committee. He is an IEEE Fellow.

Dennis Sylvester (S'95–M'00–SM'04–F'11) received the Ph.D. in electrical engineering from the University of California, Berkeley, where his dissertation was recognized with the David J. Sakrison Memorial Prize as the most outstanding research in the UC-Berkeley EECS department.

He is a Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor and Director of the Michigan Integrated Circuits Laboratory (MICL), a group of ten faculty and 60+ graduate students. He previously held research staff positions in the Advanced Technology Group of Synopsys, Mountain View, CA, Hewlett-Packard Laboratories in Palo Alto, CA, and a visiting professorship in Electrical and Computer Engineering at the National University of Singapore. He has published over 300 articles along with one book and several book chapters. His research interests include the design of millimeter-scale computing systems and energy efficient near-threshold computing for a range of applications. He holds 12 US patents. He also serves as a consultant and technical advisory board member for electronic design automation and semiconductor firms in these areas. He co-founded Ambiq Micro, a fabless semiconductor company developing ultra-low power mixed-signal solutions for compact wireless devices.

Dr. Sylvester received an NSF CAREER award, the Beatrice Winner Award at ISSCC, an IBM Faculty Award, an SRC Inventor Recognition Award, and eight best paper awards and nominations. He is the recipient of the ACM SIGDA Outstanding New Faculty Award and the University of Michigan Henry Russel Award for distinguished scholarship. He has served on the technical program committee of major design automation and circuit design conferences, the executive committee of the ACM/IEEE Design Automation Conference, and the steering committee of the ACM/IEEE International Symposium on Physical Design. He has served as Associate Editor for IEEE TRANSACTIONS ON CAD and IEEE TRANSACTIONS ON VLSI SYSTEMS.