Architecture Independent Timing Speculation Techniques in VLSI Circuits

by

Matthew R. Fojtik

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2013

Doctoral Committee:

Professor Dennis M. Sylvester, Chair
Professor David Blaauw
Associate Professor Jerome P. Lynch
Professor Trevor N. Mudge

to my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Sources of Delay Variation

Conventional digital circuits must ensure correct operation throughout a wide range of operating conditions. These operating conditions can comprise process, voltage, and temperature (PVT) variation and can be static or ambient. The primary effect these variations have on digital circuits is on circuit delays, determining a circuit's maximum operating speed [1].

Static timing variation is usually a result of process corners and mismatch. Global, chip-wide, variability in V_{th}, oxide thickness, or wire capacitance manifests as a static change in circuit delay that varies from chip-to-chip. Within-die, local, process variation is also present as a result of effects such as random dopant fluctiations and line edge roughness.

Ambient effects, such as supply noise and operating temperature, also affect circuit delay. These effects can also be global or local in scope. In the case of temperature, variation could be caused by global changes to the environment or local hot spots due to high power consumption of select blocks. These effects can also be classified in how quickly they change. Voltage droops can occur within a clock cycle, while temperature changes may gradually occur over thousands of cycles [1].

The final classification of delay variation is from the data being run through the circuit. The critical path in a circuit, and thus the maximum operating frequency, varies from cycle to cycle depending on the instruction being processed and the data being computed [2].

1.2 Conventional Approaches to Delay Variation

Microprocessor design in current CMOS technologies has become extremely challenging due to PVT variations [3, 4]. Traditionally, significant worst-case margin is added to the clock cycle to ensure that the targeted frequency is met over a wide range of PVT conditions and in the presence of fast transient timing errors (due to, e.g., fast/local supply voltage droops, soft errors, cross-coupling noise, clock jitter) [2, 5, 6]. Although this simplifies the design, the amount of margining required to achieve acceptable parametric yield has become very expensive in terms of performance and energy due to growing PVT variations. This poses a serious limit to the energy efficiency improvement in all market segments ranging from high performance (due to frequency reductions) to very low power (voltage cannot be scaled as larger margin is needed at low voltage) [7].

In recent years, a number of post-silicon techniques have been proposed to reduce design margin in microprocessors. For example, limited margin reduction has been achieved through VDD/frequency/body bias feedback compensation of slow variations that are captured through on-chip voltage/temperature sensors [5, 6, 8–11]. As an alternative, error prediction has been used through canary structures as well as lookup tables however these cannot fully capture within-die variations and still introduce some margin [7, 9, 12–14].

Table 1.1 compares several conventional approaches to addressing these types of variation. Characterization at tester time and table lookup can reduce the effect of static process variation, however speed testing all potentially critical paths can be costly. The addition of sensors to detect ambient conditions allows for slow, global ambient variation to be reduced but this comes at an even greater characterization cost. Canary circuits [12, 15–20] are a

Technique	Static		Ambient				Data
	Global	Local	Global		Local		
			Slow	Fast	Slow	Fast	
Table Lookup	X	X					
Table & Sensors	X	X	X				
Canary Circuits [12, 15–20]	X		X				
Razor-Style Designs [2, 7, 21–24]	X	X	X	X	X	X	X

Table 1.1: Types of Delay Variation

popular approach to reducing timing margins which involve creating a replica circuit with delay matched to the critical path. This circuit tracks the operating conditions but can not respond to fast changing ambient conditions, and can not reduce local mismatch since the canary is a separate structure from the design itself.

The Razor [24] system proposed eliminating all timing margins, including those caused by data variation, by running the circuit with occasional timing errors and correcting the errors when they occur. As the checking is performed *in situ* , margins for every classification of timing variation can be eliminated. Razor Style Designs [2, 7, 21–24] allow errors to occur in order to eliminate all types of timing margins, though the methodology for detection and correction varies from scheme to scheme.

1.3 Comparison of Razor-Style Designs

Although all Razor-Style Designs [2, 7, 21–24] eliminate timing margins by running with timing errors and correcting for those errors, they differ in their correction mechanisms.

The Razor [24] system proposed two approaches to correcting errors, global stalling and counterflow pipelining. Global stalling relied on the type of double sampling error detection circuit used where, upon detection of an error, all flip-flops in the processor stalled while all Razor flip-flops in the processor reloaded their values with the correct data stored in their shadow latch. The circuit then proceeded as normal in the next cycle. This technique is architecture independent, thus it can be applied to an existing design with minimal designer effort, however it is limited in its scale. Because the technique relies on communicating an error signal to the entire processor in the same cycle, it can not be scaled to large designs with aggressive clock periods.

To address this concern, Razor also proposed counterflow pipelining, which relied on the architecture to correct for errors. When a pipeline state detected an error, it communicated a stall with its downstream stage and a flush with its upstream stage. When a flush reached the first stage, it fetched the first instruction after the failing one and continued. Although this technique eliminated the need for a global stall signal, it requires the architecture to be designed with Razor in mind and puts the correction logic in the RTL adding to the work of the designer.

Further Razor work also used architecture specific correction mechanisms. Razor II [22] recognized that many processor architectures already support speculative execution when predicting branches. Thus, when a timing error was detected, it set a flag indicating that the instruction was in error and used the existing flushing mechanism to clear the pipeline and rerun the errant instruction. In order to ensure the failing instruction did not repeatedly fail and prevent the system from making forward progress, the instruction was rerun at half the original clock frequency. EDS [23] and ARM's Razor [21] corrected for errors in a similar way. Multiple Issue EDS [7] avoided adjusting the clock network when replaying failing instructions by issuing it multiple times in order to set up the correct values at the inputs of each pipeline stage. In these systems, it is necessary that no speculative state be written to memory or external peripherals, thus the configuration of the processor is limited to architectures with stabilization stages and write-back stages at the end of the pipeline.

Because of the architectural invasiveness of these techniques, no published Razor Style system had been applied to a complete existing commercial processor. If processor architectures are not designed with Razor in mind from the beginning, it can be difficult or impossible for a designer to adapt the system to allow Razor Style error detection and correction.

Additionally, in all conventional Razor-style systems [2, 7, 21–24], regardless of architecture dependence, there is a fundamental tradeoff between speculation window and short path, or minimum delay, constraints. When data arrives within the speculation window after the positive edge of a flip-flop'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 the speculation window such that no short path can falsely trigger an error. This timing constraint must also be margined for any degree of timing variation. Detailed analysis of this tradeoff is discussed in Appendix A.

Technique	Eliminates Margins	Slow Variation	Fast Variation	Scalable Size	Arch. Indep.	Min Delay Unchanged	Design Cost
Canary Circuitry [12, 15–20]		X		X	X	X	Low
Razor (Global Stall) [24]	X	X	X		X		Low
Razor (Counterflow) [24]	X	X	X	X			High
Razor II [22]	X	X	X	X			High
EDS [23]	X	X	X	X			High
ARM Razor [21]	X	X	X	X			High
Multiple Issue EDS [7]	X	X	X	X			High
Bubble Razor (Proposed) [25]	X	X	X	X	X	X	Low
Voltage Razor (Proposed)	X	X	X	X	X	X	Low

Table 1.2: Comparison of Timing Margin Reduction Techniques and Razor-Style Designs

1.4 Contributions of this Work

This thesis proposes two latch based Razor Style techniques that address the architecture dependence and scalability issues of previous Razor Style systems and also break the dependence between speculation window and minimum delay constraints.

Table 1.2 compares Canary Circuitry to existing Razor Style designs and the methods proposed in this thesis. All Razor Style designs eliminate timing margins and adapt to all types of timing variation but a tradeoff exists between scalability and architecture independence. All suffer from the link between minimum delay constraints and speculation window. The proposed systems offer the benefits of prior Razor Style techniques while also providing the scalability, architecture independence, and unaffected minimum delay constraints of canary circuits. In addition, the design effort of the proposed systems are low due to their automated nature.

Chapter 2 proposes the concept of latch based Razor and presents the Bubble Razor algorithm. It also explains the process of automatically transforming an existing design to a latch based Razor style system. Chapter 3 describes the specific implementation details and measurement results of the Bubble Razor algorithm applied to an ARM Cortex-M3 processor and implemented in silicon in a 45nm SOI process. Chapter 4 discusses Voltage Razor, another latch based Razor technique that uses voltage boosting to provide the circuit speed up needed for error correction. Chapter 5 discusses transforming a flip-flop based design which uses architectural clock gating to two-phase latches in which time borrowing can be used for error correction, allowing the application of either Bubble Razor or Voltage Razor to the target design. Bubble Razor is applied to a larger processor which makes use of clock gating, the ARM Cortex-R4, in Chapter 6. Chapter 7 summarizes the contributions in this thesis and proposes some direction for future work. A thorough analysis of the timing constraints in latch based razor is discussed in Appendix A

CHAPTER 2

Bubble Razor: An Architectural Independent Approach to Timing Error Detection and Correction

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.

2.1 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 [12, 15–20] The Razor system [2, 24] 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 [26], Time-Redundant Latches [27], Razor I latches[2,24], Razor II latches [22], Transition Detector with Time Borrowing (TDTB) [23], and Double Sampling with Time Borrowing (DSTB) [23]. 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 [22, 23]. 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.

2.2 Review of Prior Razor Methods

2.2.1 Conventional Razor System Timing

In all conventional Razor-style systems [2, 7, 21–24, 28], there is a fundamental tradeoff between speculation window and short path, or minimum delay, constraints (Fig. 2.1). In a system that allows 100ps of timing speculation, when data arrives within 100ps after the positive edge of a flip-flop or pulsed latchs 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 100ps 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.

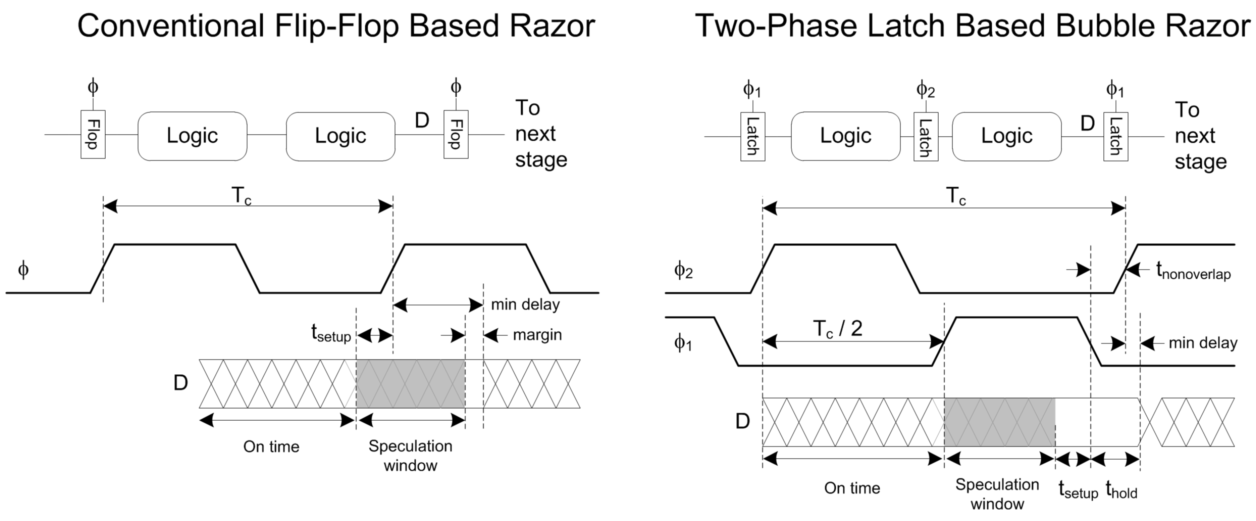


Figure 2.1: By using two-phase latch based timing, minimum delay constraints are restored to their conventional values allowing for large speculation windows.

2.2.2 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 [2] 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 [22] 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.

Razor Is 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.

2.3 Proposed Bubble Razor Approach

2.3.1 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 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. 2.1). 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.

2.3.2 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. 2.3). 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. 2.2). 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

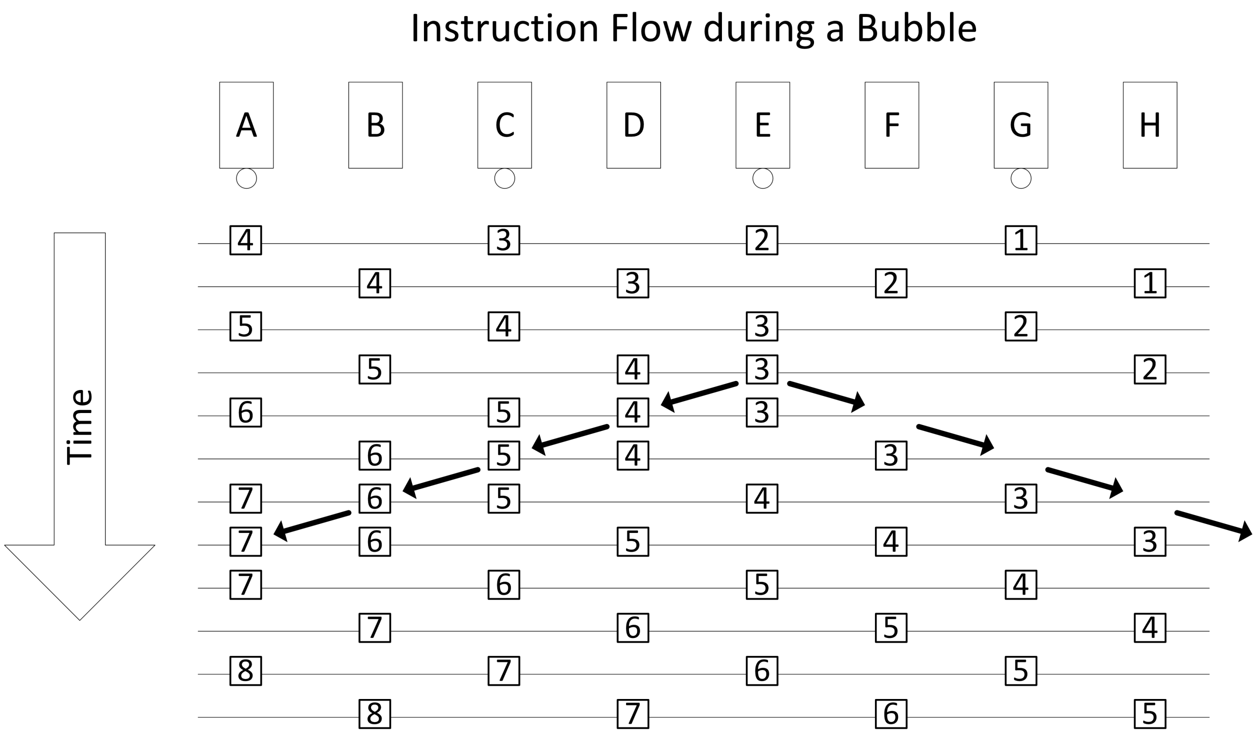


Figure 2.2: In a two-phase latch based system, instructions can stall without immediately being overwritten.

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 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. 2.4). 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

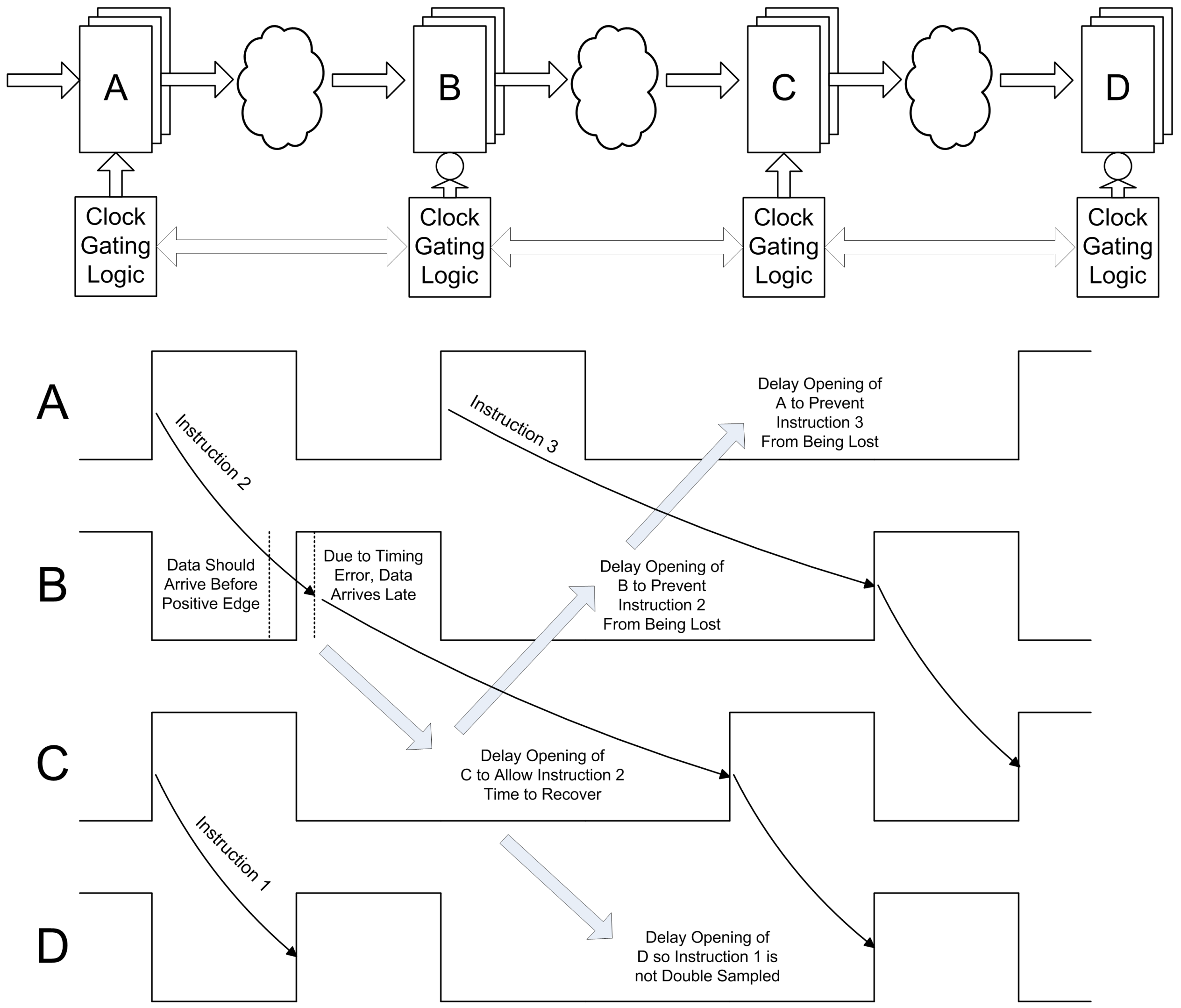

A
B
C
D
Clock Gating Logic
Clock Gating Logic
Clock Gating Logic
Clock Gating Logic
A
Instruction 2
Instruction 3
Delay Opening of A to Prevent Instruction 3 From Being Lost
B
Data Should Arrive Before Positive Edge
Due to Timing Error, Data Arrives Late
Delay Opening of B to Prevent Instruction 2 From Being Lost
C
Delay Opening of C to Allow Instruction 2 Time to Recover
Instruction 1
D
Delay Opening of D so Instruction 1 is not Double Sampled

Figure 2.3: Timing errors are corrected by propagating bubbles which gate off clock pulses throughout the circuit.

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

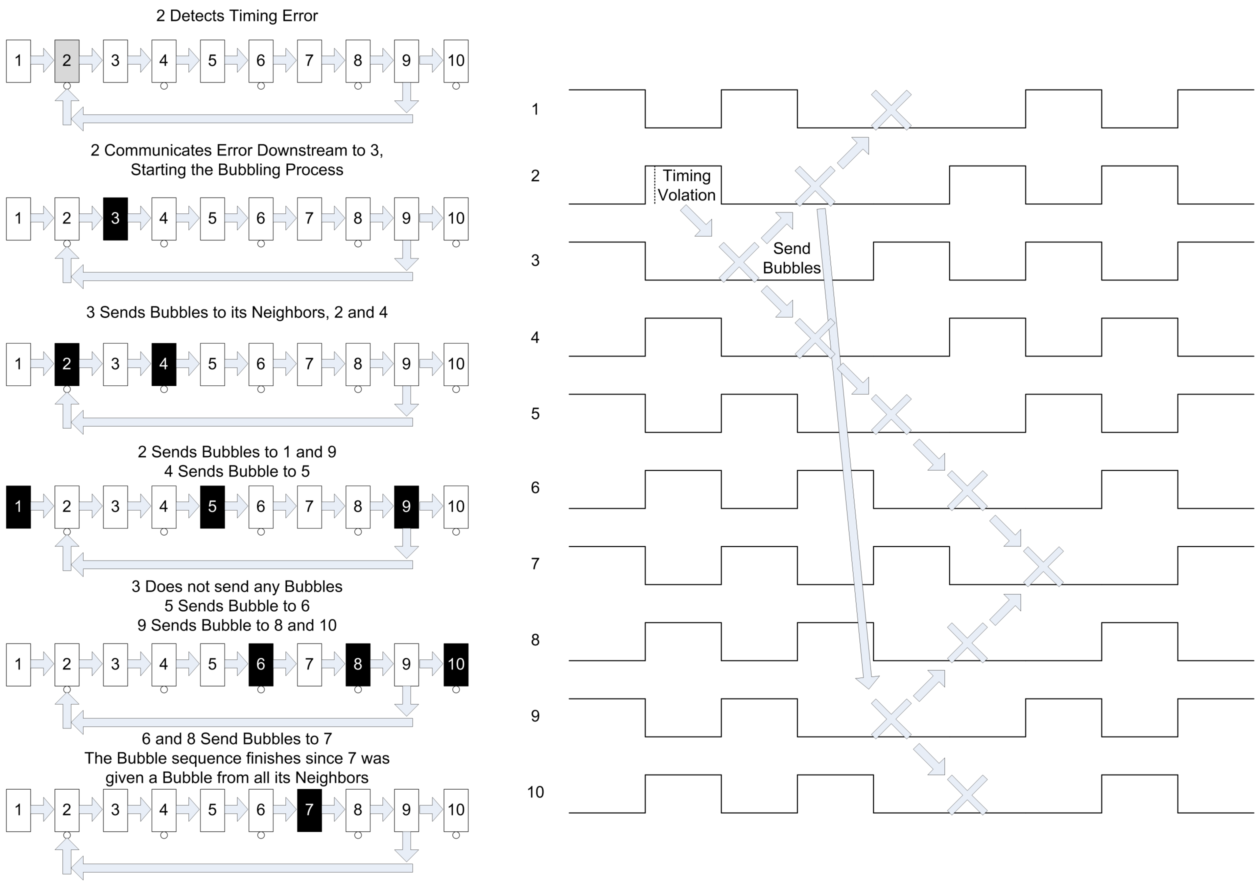

2 Detects Timing Error
1
2
3
4
5
6
7
8
9
10
2 Communicates Error Downstream to 3,
Starting the Bubbling Process
3 Sends Bubbles to its Neighbors, 2 and 4
2 Sends Bubbles to 1 and 9
4 Sends Bubble to 5
3 Does not send any Bubbles
5 Sends Bubble to 6
9 Sends Bubble to 8 and 10
6 and 8 Send Bubbles to 7
The Bubble sequence finishes since 7 was
given a Bubble from all its Neighbors
Timing
Volation
Send
Bubbles

Figure 2.4: Bubbles are communicated to neighboring latches. Upon error resolution, every latch has stalled for exactly one cycle.

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.

2.4 Bubble Razor Implementation Issues

2.4.1 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. 2.5 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

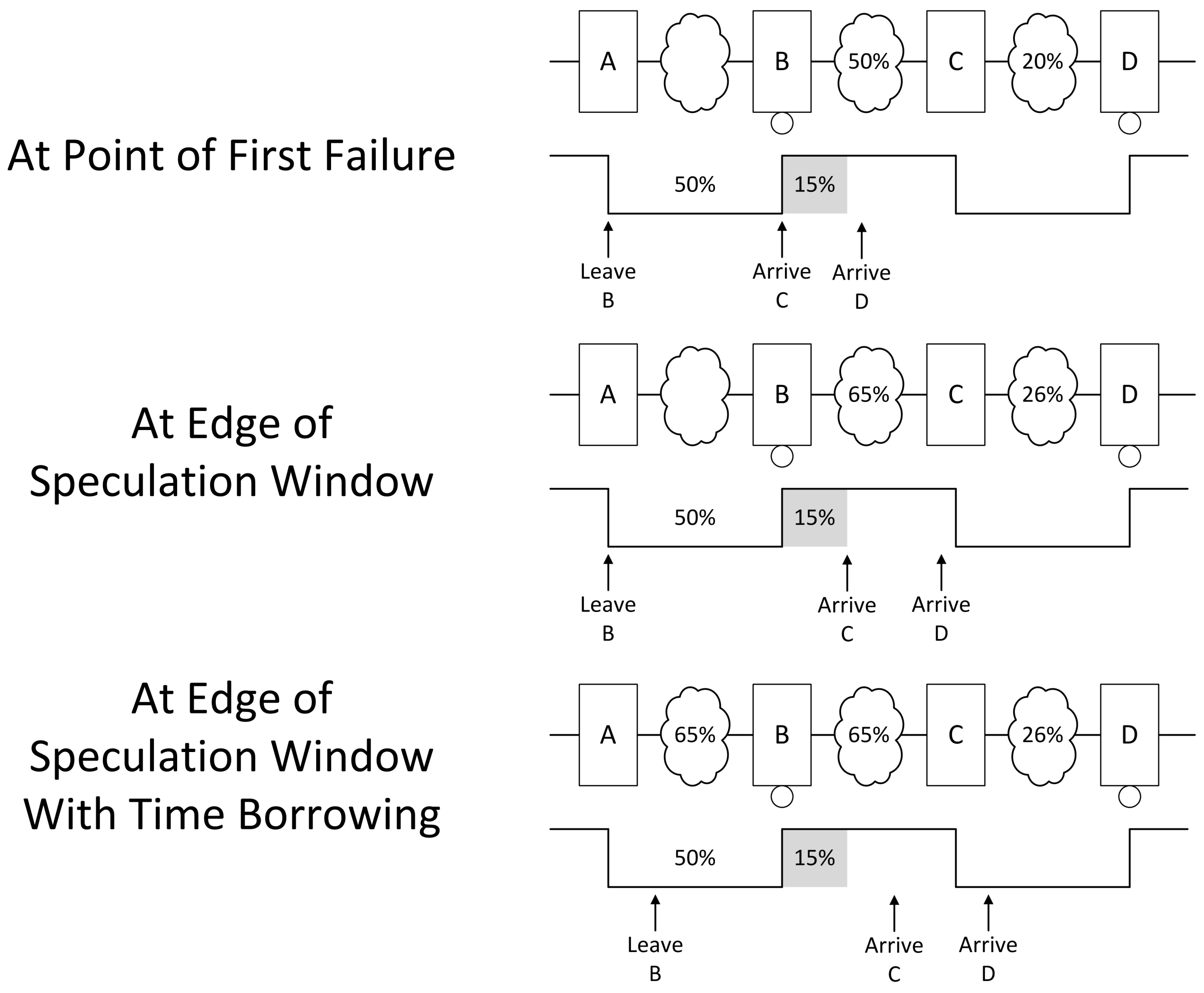


Figure 2.5: When determining where error detection is needed for a given speculation window, time borrowing can complicate analysis.

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.

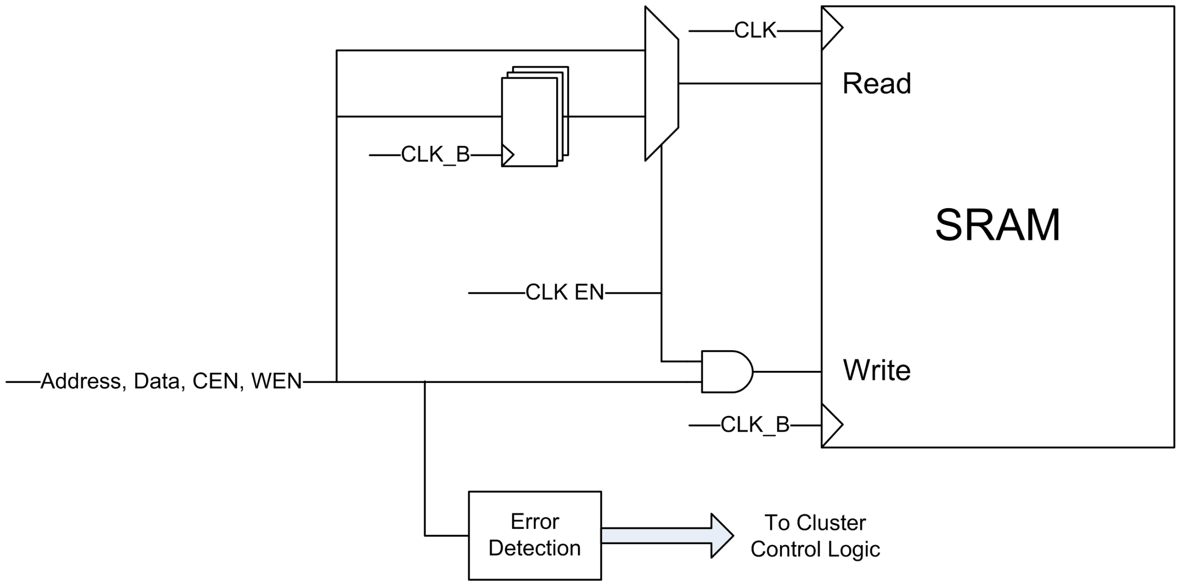
2.4.2 SRAM Interface

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. 2.6). 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 a Cortex-M3 implementation timing is unaffected as SRAM is already operating in approximately 50% of a clock cycle with the other 50% of a clock cycle being used by combinational logic between the processors 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.

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



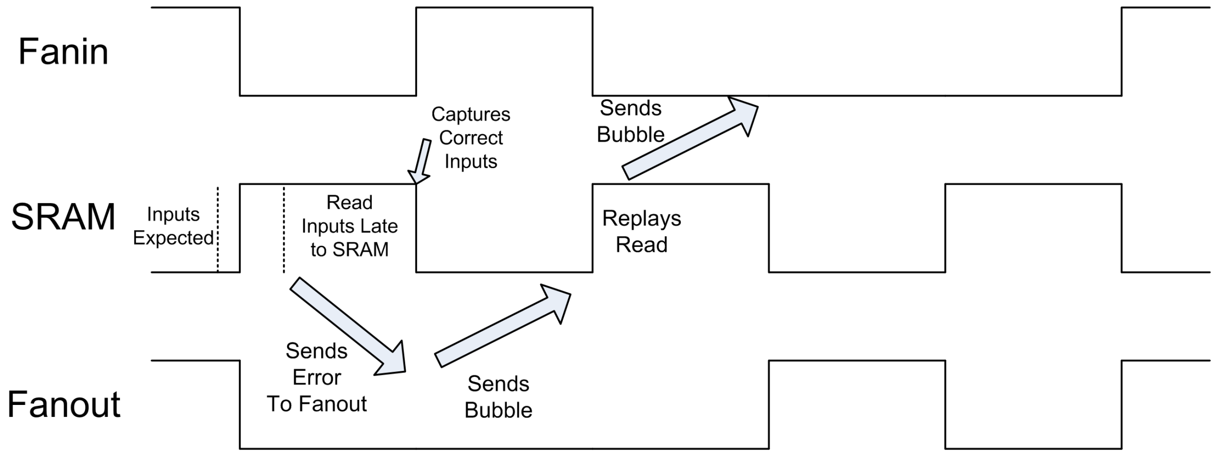


Figure 2.6: Wrapper logic is placed around the SRAM such that it can be treated as a positive latch.

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.

2.4.3 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. 2.7). 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 [29].

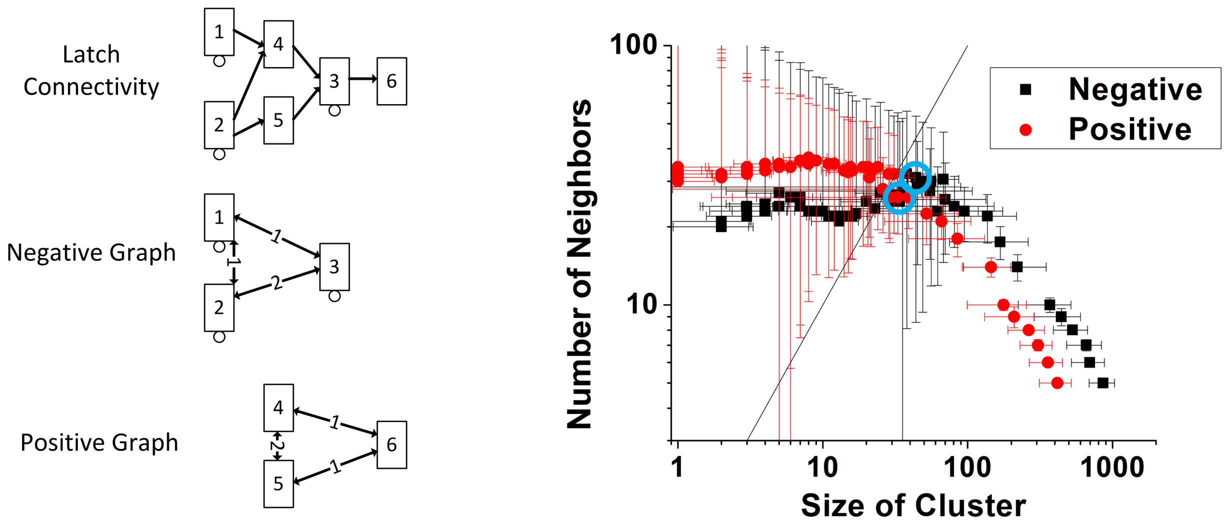


Figure 2.7: Clustering was performed automatically by building two graphs based on latch connectivity. A tradeoff exists between the size of OR gates which is balanced by the choice of the number of clusters. The data shown is for a Cortex-M3 processor.

Although the assignment of clusters is performed automatically, the designer chooses 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.

CHAPTER 3

Eliminating Timing Margins in an ARM Cortex-M3 Processor with Bubble Razor

We implemented Bubble Razor on an ARM Cortex-M3 microprocessor in 45nm 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.

3.1 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. 3.1 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. 3.1. 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.

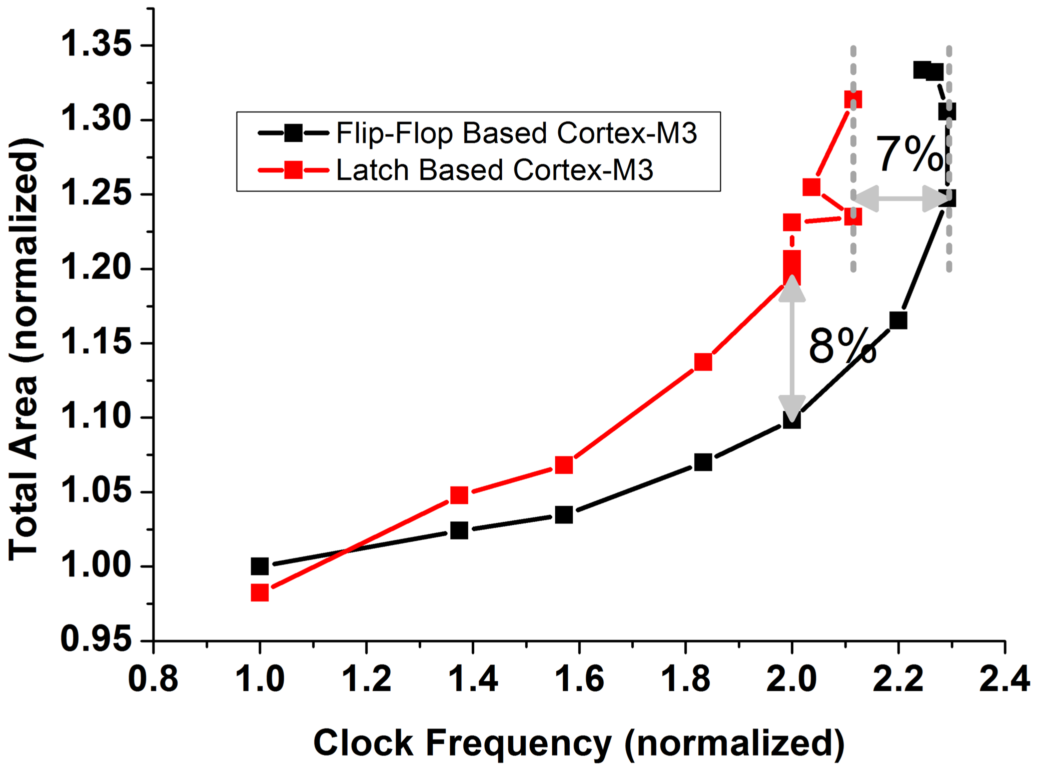
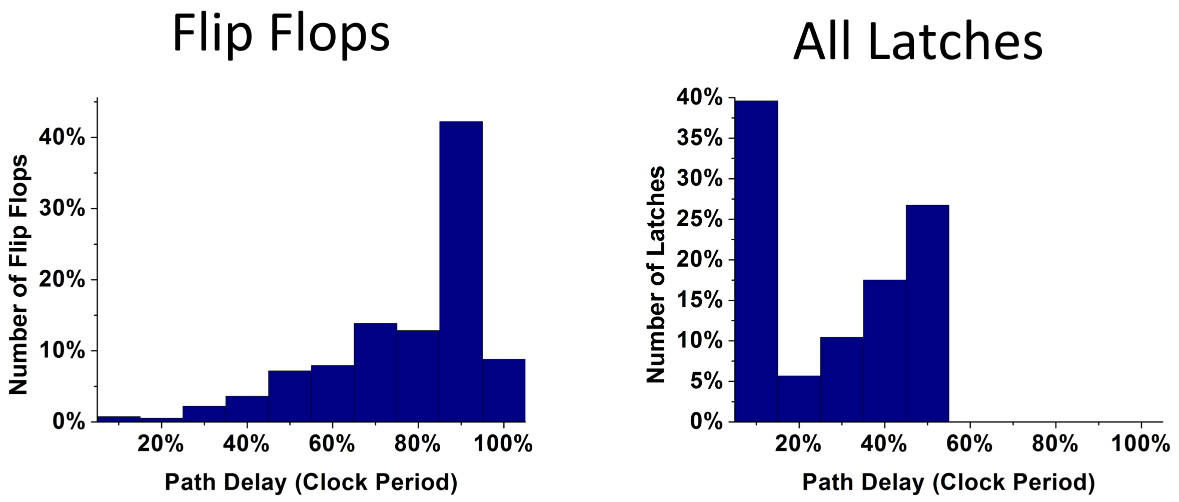


Figure 3.1: Transforming the Cortex-M3 to two-phase latches can incur an 8% area penalty or 7% performance penalty.

3.2 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. 3.2 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.



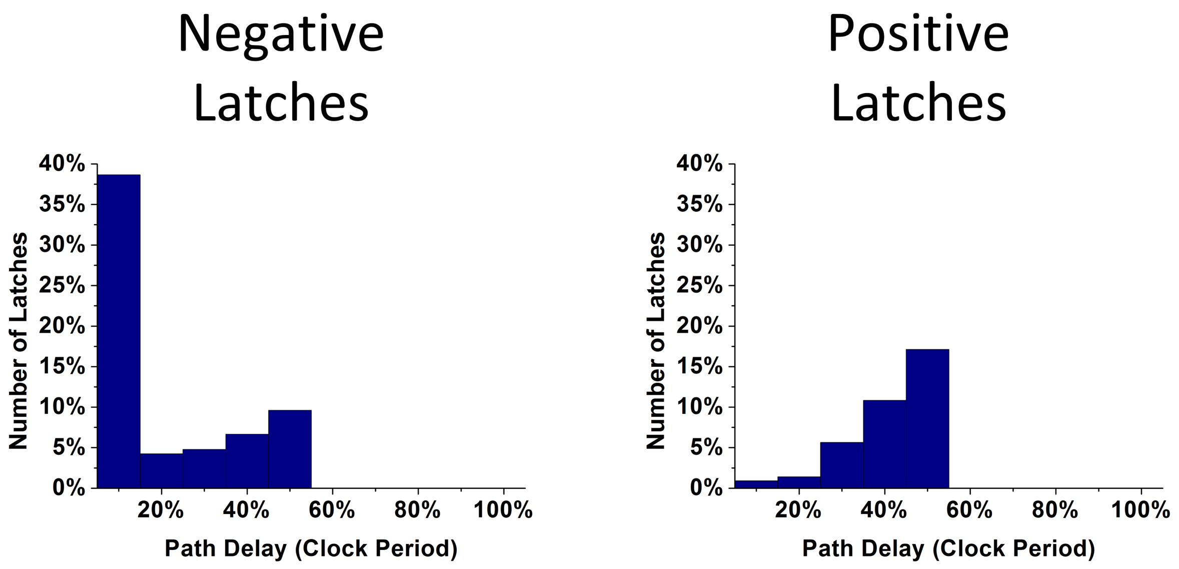


Figure 3.2: The Cortex-M3 has an imbalanced path distribution as a result of retiming.

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. 3.3 shows the area overhead and speculation window results for these three tech-

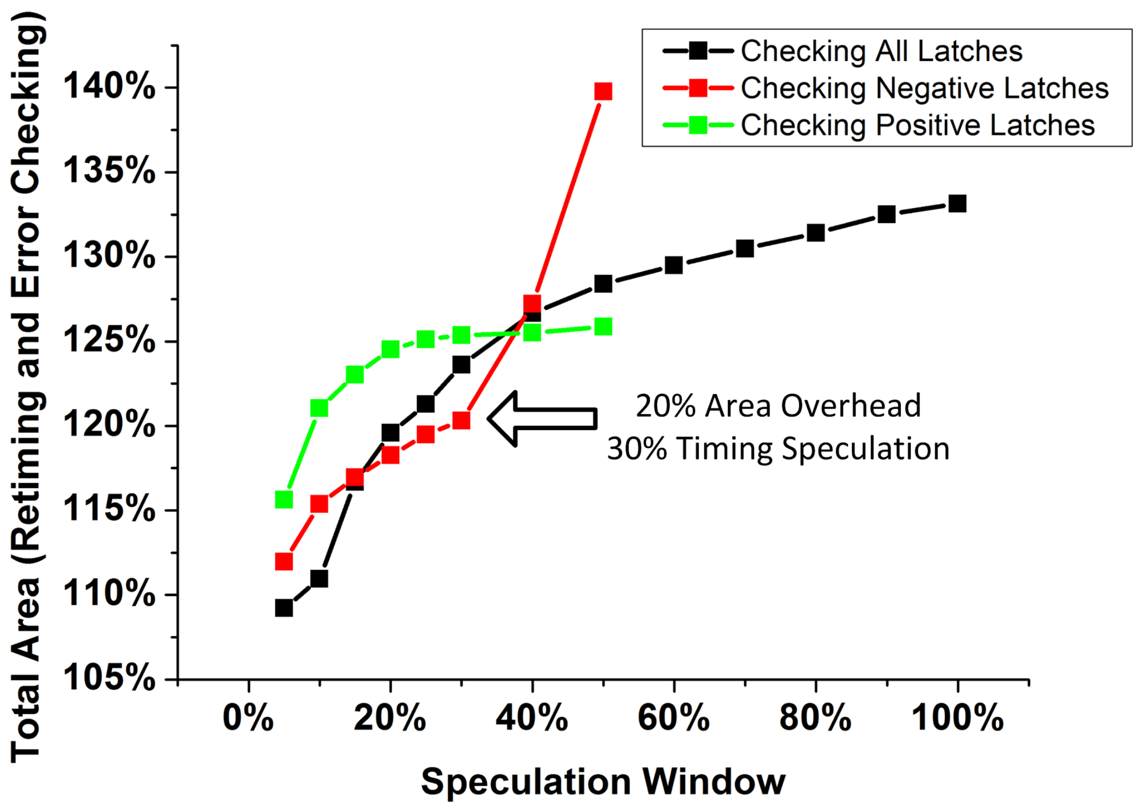


Figure 3.3: Error checking can be added to a subset of latches, or a subset of only positive or only negative latches, yielding different area overheads. This is due to the distribution of critical path delays at the inputs of each latch.

niques 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 further 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%.

3.3 Implementation Circuitry

Error detection was performed using the Bubble Razor Latch, similar in design to the error detection flip-flop in [2]. A shadow latch captures data as the main datapath latch opens (Fig. 3.4). An XOR compares the two values and will flag an error if data 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 [22] could also be used.

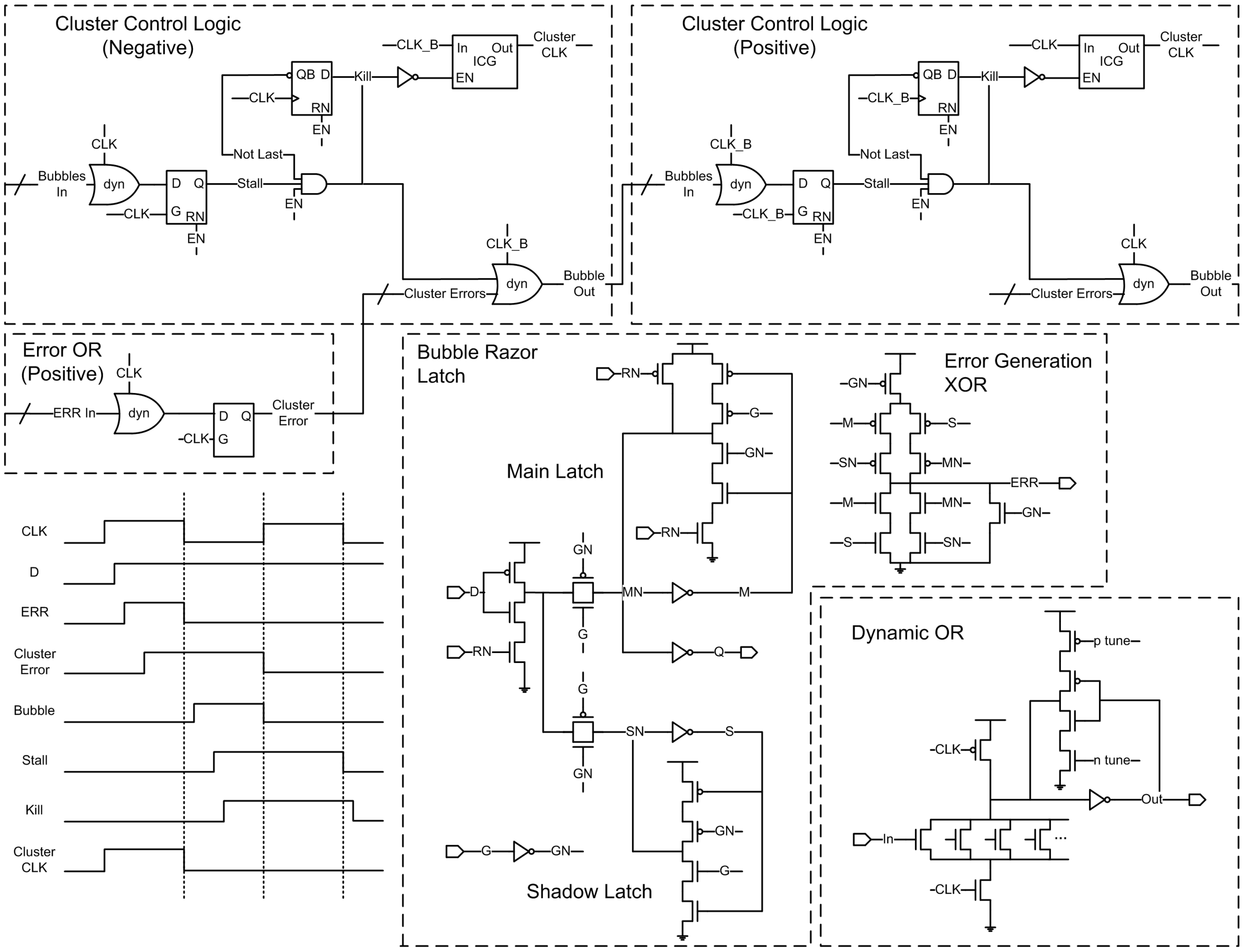


Figure 3.4: Bubbles are combined using dynamic OR gates. A cluster ignores bubbles if it stalled in the previous cycle.

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.

3.4 Silicon Test Chip

Bubble Razor was applied to the Cortex-M3 processor, a 1.25 DMIPS/MHz microcontroller [30], and implemented in a 45nm SOI process. This silicon test chip is the first published Razor-style implementation to demonstrate a transformed 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 Section and , 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.

3.5 Silicon Measurement Results

Because of the robust design decisions mentioned in Section , 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 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 , and . 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. 3.5). Alternatively, supply voltage can be lowered at iso-performance, reducing M3 energy consumption by 43, 54, and 60%, respectively.

Fig. 3.6 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 consumption. 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. 3.6. 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 addition, canary circuits can only adapt to slow changing operating conditions due to the time required to change the processor

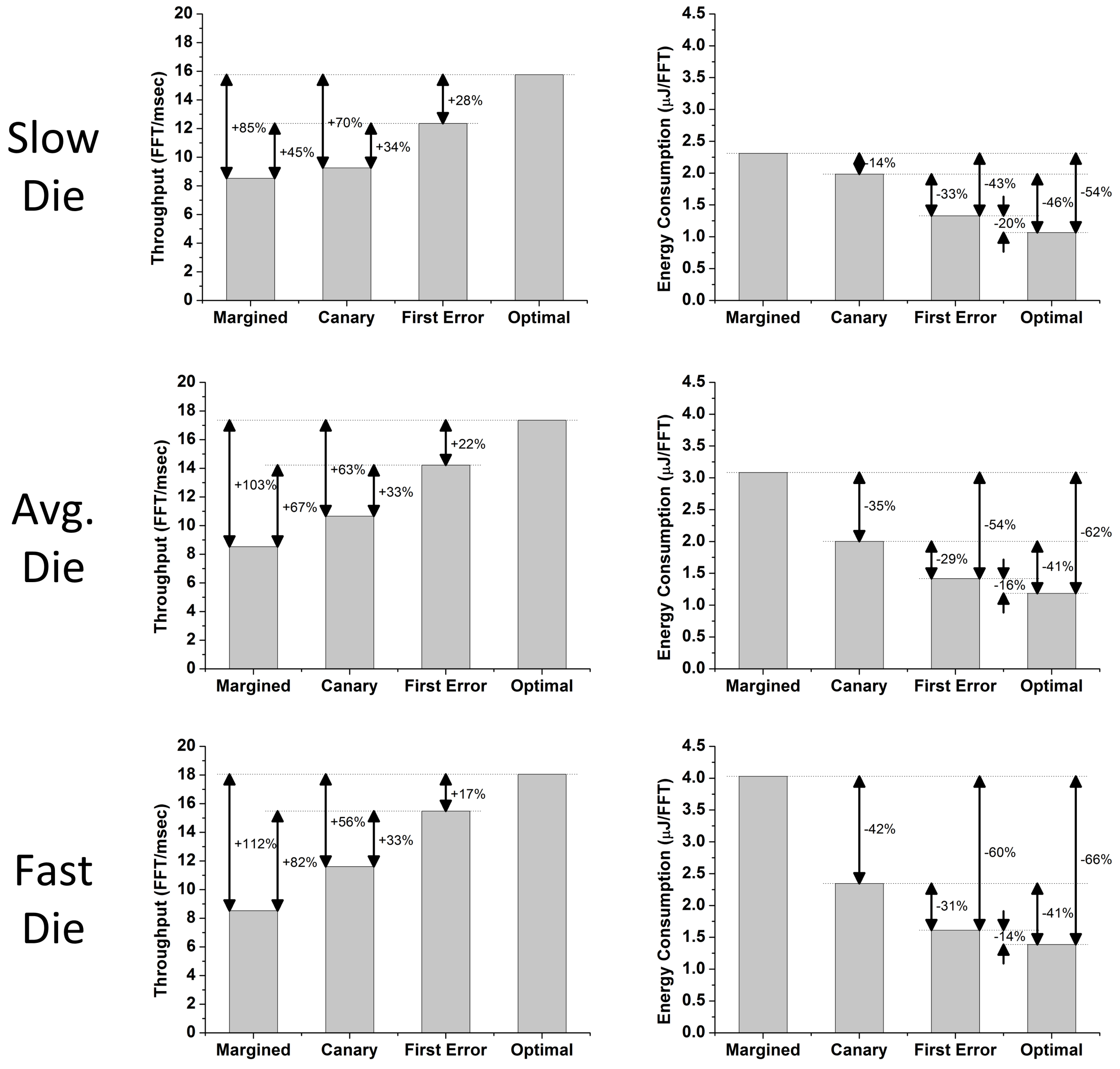


Figure 3.5: By running the system under nominal conditions instead of with worst case margins, performance or energy can be improved.

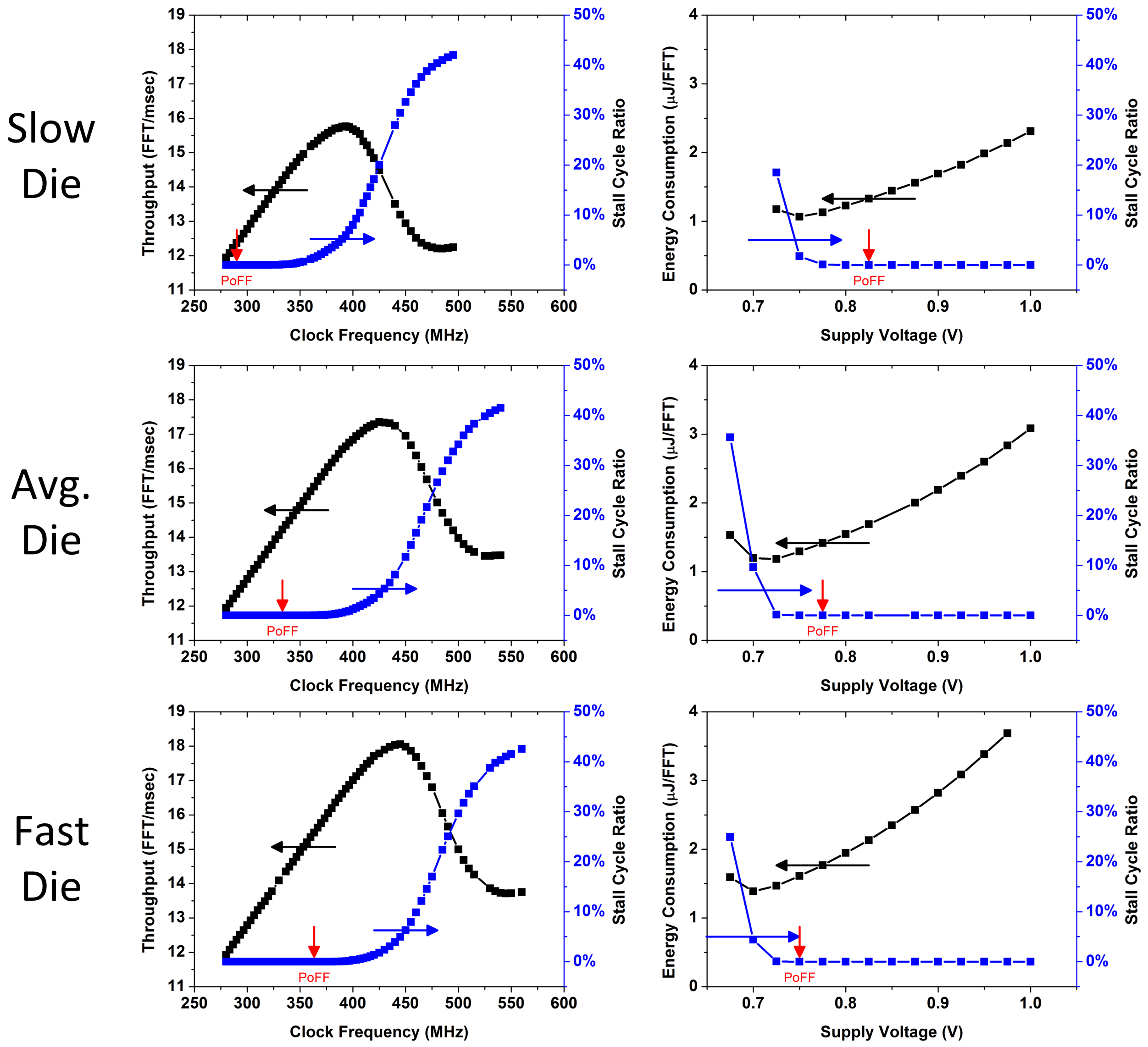
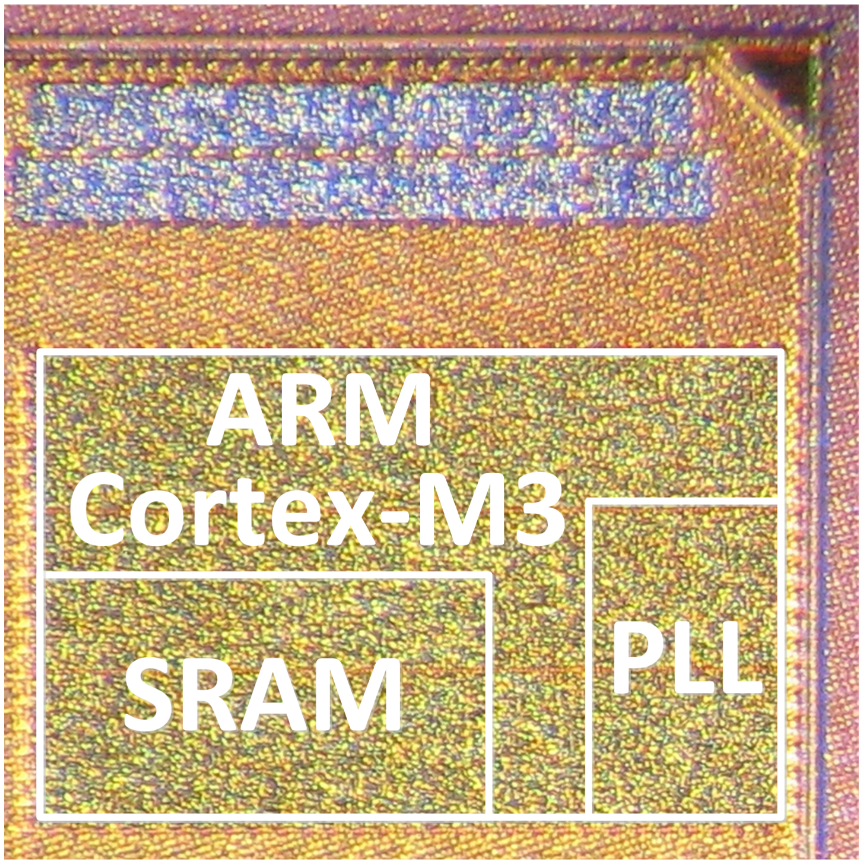


Figure 3.6: Due to only a single cycle penalty for fixing timing errors, an additional 22% performance gain or 17% energy reduction can be made by running beyond the Point of First Failure.

clock frequency, and thus canary circuits cannot eliminate the margins for supply droop. Adding margin for 3σ 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.

3.6 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 45nm CMOS to validate the technique and showed a 100% throughput improvement or 60% energy savings over running with worst-case timing margins.



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%

Figure 3.7: Die photo and system information.

CHAPTER 4

Voltage Razor: Using Supply Voltage Boosting as Error Correction

Similar to Bubble Razor, Voltage Razor uses a 2-phase edge-based latch clocking with transition detectors at latch inputs. The unique feature of Voltage Razor is that, upon detection of an error, the effective time available for each pipeline stage computation is increased by making logic faster, rather than stretching the cycle time. When a latch is transparent and detects a transition, an error is detected, and the supply voltage is quickly boosted to speed subsequent logic, allowing for the compensation of the late arriving input. Many prior Razor techniques have employed voltage regulation to keep average error rate within bounds, but this is always performed with a slow feedback loop that cannot correct incoming errors at run time. In Voltage Razor, we propose new circuit techniques to enable voltage boosting times that are comparable to the clock cycle time.

4.1 Principle and Motivation

As with Bubble Razor, in Voltage Razor the integrity of the state is preserved by the same latches within pipeline stages, as long as the error is corrected quickly enough to avoid state corruption. When a latch observes its input switching during its transparent phase, it flags an error. The failing latch still holds the correct data but is now effectively borrowing time from the next logic stage. Unless the error is corrected, this time borrowing can increase with each stage and will eventually cause a failure when the maximum amount of time borrowing is exceeded. In Voltage Razor, we avoid this failure by boosting the supply from the value VDD_L in normal operation to the boosted voltage VDD_H, which speeds up the logic of subsequent stages. If the voltage is boosted sufficiently fast, the time borrowing will be negated before its limit is reached and the circuit will return to normal operation where all max-delay constraints are met and no error is flagged. The supply voltage is then returned to VDD_L.

The key challenge in Voltage Razor is to boost the supply sufficiently fast (within a few clock cycles). This goal can be feasibly pursued if the amount of boosting is limited and the cycle time is not too short. Both conditions can be met when the core operates at near threshold (e.g., VDD_L 400-700 mV), since:

- At near threshold, gate delay is rather sensitive to VDD. For example, in 45nm SOI IBM technology, boosting the voltage from $VDD_L = 500$ mV to $VDD_H = 650$ mV leads to a significant gate delay (fanout-of-four, or FO4) reduction by 2.5.

- When VDD_L is close to the threshold, operating frequencies are typically much lower than the conventional low-GHz regime. Given these larger cycle times, it becomes much easier to boost the voltage in 1-2 clock cycles. For example, FO4 at $VDD_L =$ 500 mV is 4.6 greater than at VDD = 1 V.

The large speculation time that is enabled by Voltage Razor permits detection of the large variations that are observed near threshold, which are greater by almost one order of magnitude than at nominal voltage (at 45nm, process variations require a 0.7 margin at VDD = 1 V, which becomes 5.8 at 500 mV). Hence, near threshold is the natural operating range for Voltage Razor.

Within the speculation window, Voltage Razor can correct errors without introducing any stalling. This gives Voltage Razor two very unique properties:

- It corrects errors with no performance penalty, as opposed to previous techniques that require several or tens of cycles to recover from errors.

- External behavior is not affected by error detection. Hence, no modification of the interfaces with surrounding blocks is required, dramatically simplifying system integration.

Also, Voltage Razor is able to correct fully correlated errors, e.g., the occasional delay increase in adjacent pipeline stages due to a common voltage droop lasting multiple cycles. In this case, voltage boosting is activated by the first stage violating its timing constraint, and also compensates the delay increase of the second stage. The same consideration holds for multiple errors occurring in the same stage and adjacent cycles.

From a design point of view, the introduction of Voltage Razor into a microprocessor does not interfere with the cycle timing of the original circuit, as the latch error signals and the supply are orthogonal to the paths where data flows.

4.2 Error Correction Through Voltage Boosting

Figure 4.1 shows how timing errors are corrected in a Voltage Razor system. The green, yellow, and red regions signify where transitions are on time, late and borrowing time, and critically late past the point of correction respectively.

Data arrives at point A slightly after the negative edge of CLK and begins to borrow time from the next stage. This late transition is flagged as an error which starts the boosting process. VDD transitions from Vlow to Vhigh as a result of the flagged error. As this is happening, data continues to flow through the pipeline, initially borrowing more time when reaching point B. However as delays decrease due to the higher supply voltage, the time borrowed when reaching point C is less and the error has been completely corrected by the time data reaches point D. Alternatively, had the boost not occurred, data would transition as shown in the dashed lines, continualy borrowing more time. By the time the data reached point D, so much time would have been borrowed that data would have arrived critically late and caused a system failure.

Figure 4.2 shows simulation results of a voltage boosting correction. The top half of the plot shows the system clock while the bottom half shows the arrival of data values at latch inputs. Once a violation is detected in cycle 2, the boosting process begins. Voltage rapidly rises and overshoots the desired value for some time before ringing and eventually settling

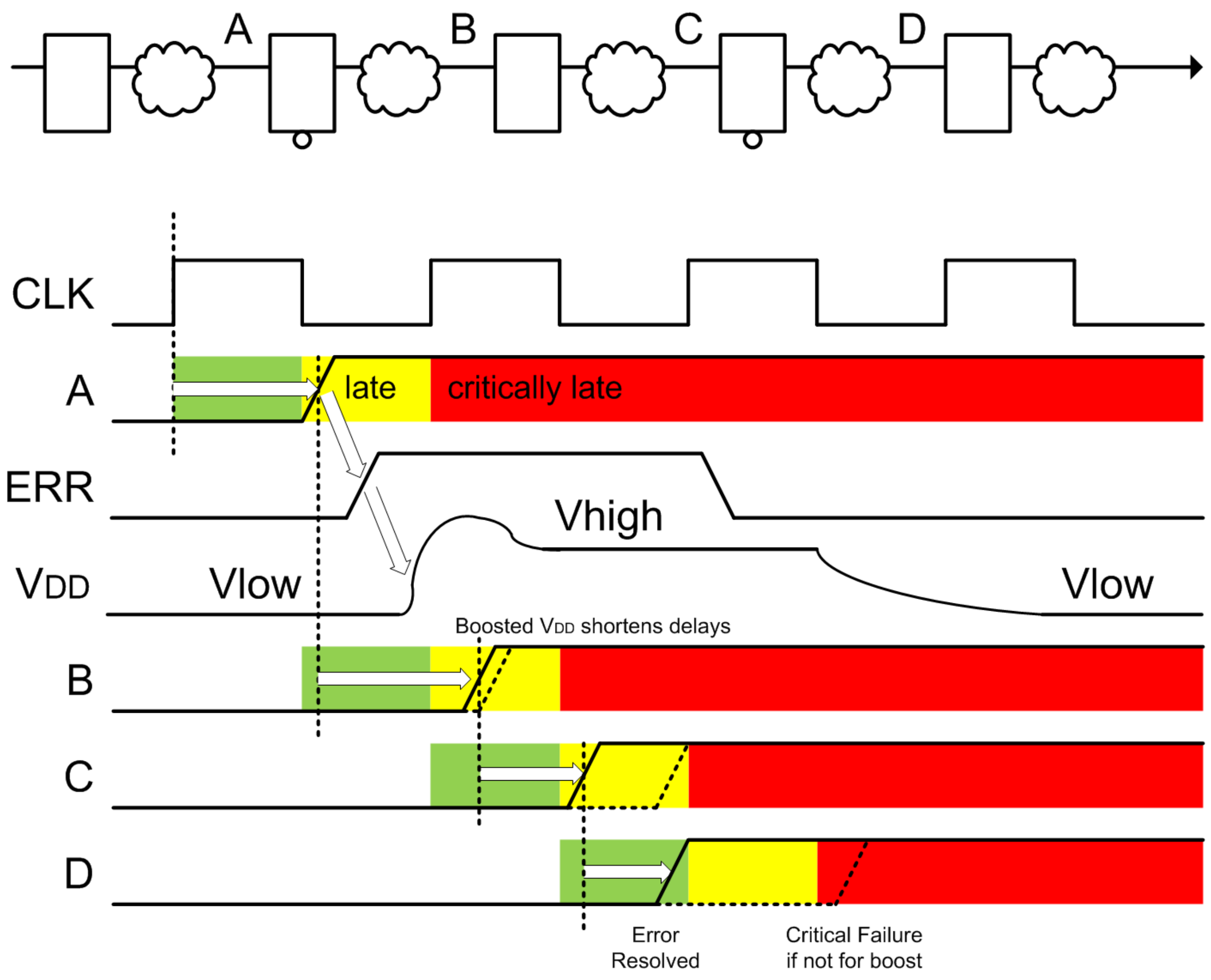


Figure 4.1: The boosting of VDD allows the circuit to run faster and for timing errors to recover

on Vhigh. Despite the fact that VDD has not yet settled on its final value, it remains higher than Vlow and thus decreases circuit delay allowing for the timing violation to be corrected.

Although in conventional systems, ringing and instability on VDD are considered extremely negative effects, in Voltage Razor, the initial overshoot of VDD while boosting is beneficial. This overshoot results in the lowest values for circuit delays and rapidly assists in recovery at a time when it is most needed.

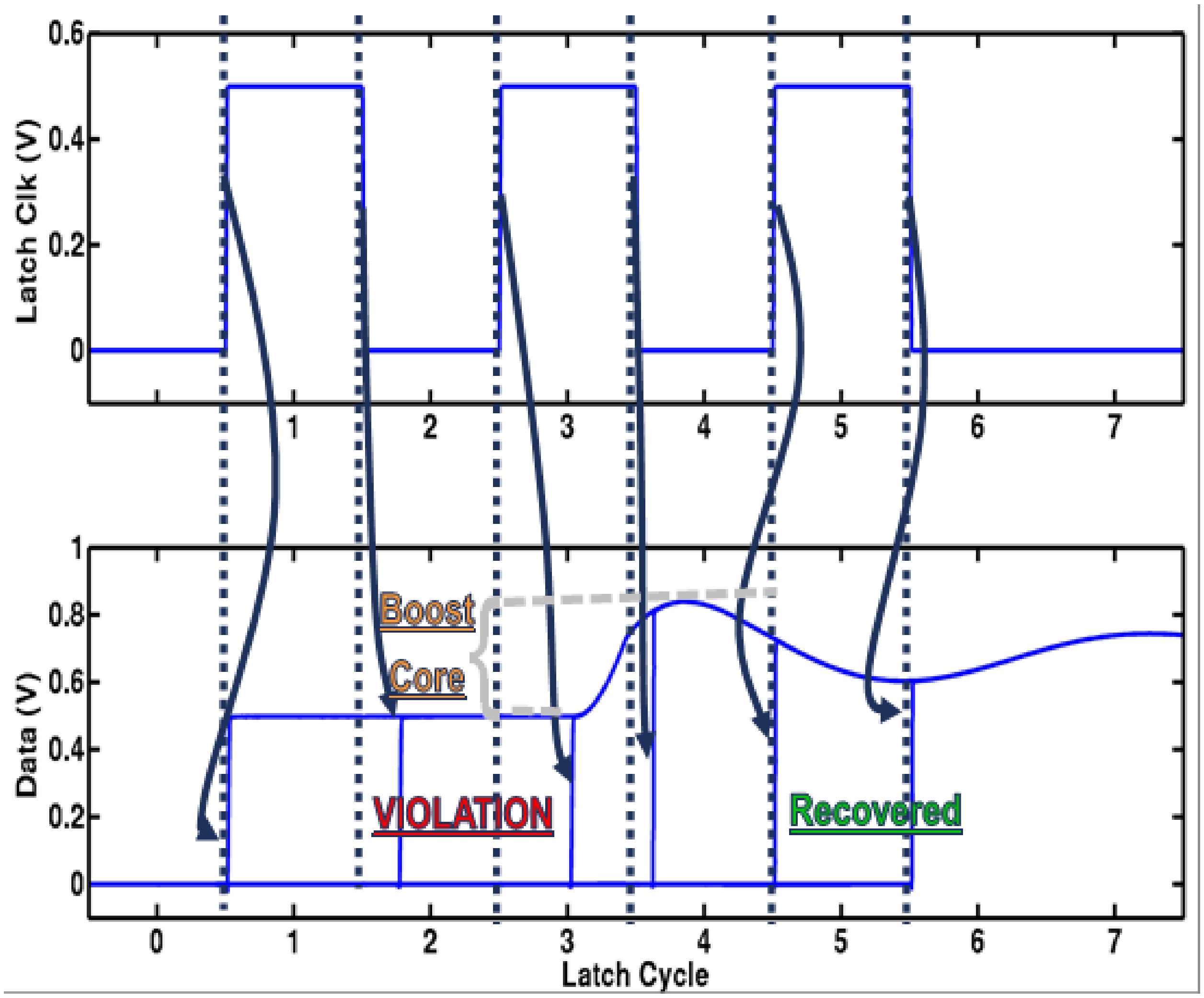


Figure 4.2: Boosting of VDD allows for the correction of the timing error even before VDD has stabilized and stopped ringing

4.3 28nm Test Chip Overview

Voltage Razor was implemented on an ARM Cortex-M3 and a test chip was implemented in a 28nm process. Figure 4.3 shows a block diagram of the system. Three ARM Cortex-M3 processors are included, a baseline flip-flop based core, a baseline two-phase latch based core, and the Voltage Razor core. Since there is no error detection on the baseline cores or bus interface logic, they are always powered by Vhigh. Level converters exist to boost bus signals from the Razor core to bus logic, as well as from the bus logic to the SRAM blocks.

Two sets of headers can be independently controlled to connect the supply of the Razor core to either Vlow or Vhigh. The Razor core outputs a signal to the voltage boosting error

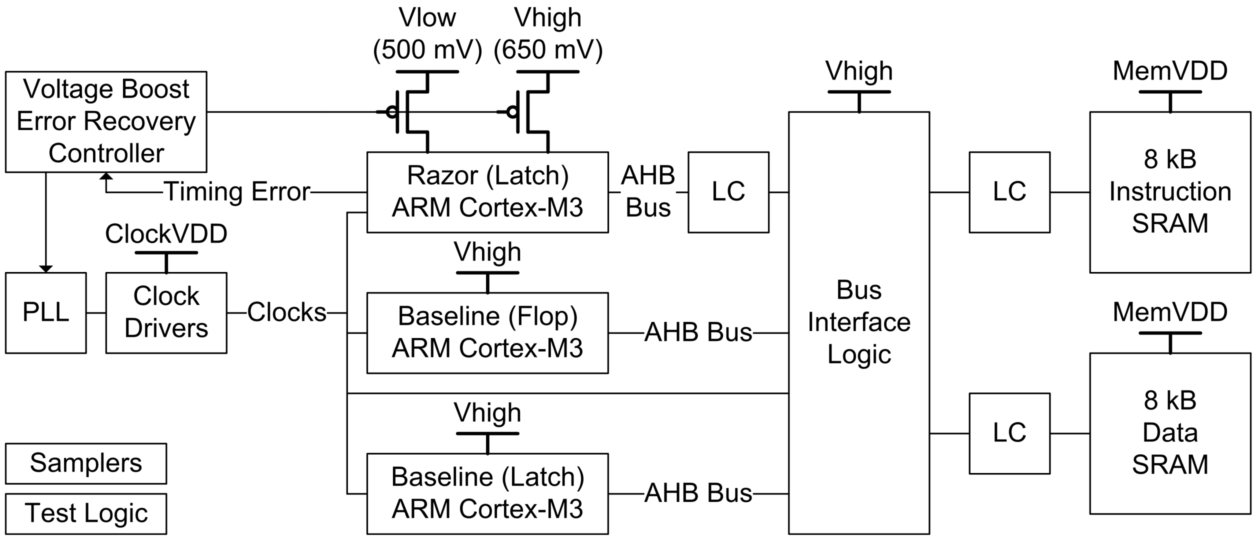


Figure 4.3: Block diagram of Voltage Razor test chip

recovery controller signifying that a timing error has occurred and requesting a boost in supply voltage. This error recovery controller contains several highly configurable timers which can control the exact sequence in which the supply is raised and lowered.

A PLL creates a master clock which is used to derive all the various clocks in the system which are computed and driven by logic running at ClockVDD. The boost controller also can control whether the PLL is in slow or fast mode and rapidly change the clock frequency in a glitch-free manner if needed. Samplers and test logic are present to control the system's various testing controls and sample voltage and clock waveforms.

4.4 Error Detection Latch

As full view layouts of standard cells were not available, the error detection latch was comprised of several standard cell gates as shown in Figure 4.4. The timing of this latch is shown in Figure 4.5.

Unlike with Bubble Razor, the shadow latch uses a separate detection clock DC than the datapath latch. This allows for the error detection window to be adjusted such that some amount of time borrowing is allowed without flagging errors as shown in Figure 4.6. Alternatively, DC could be shifted earlier in time such that a boost is triggered earlier, even

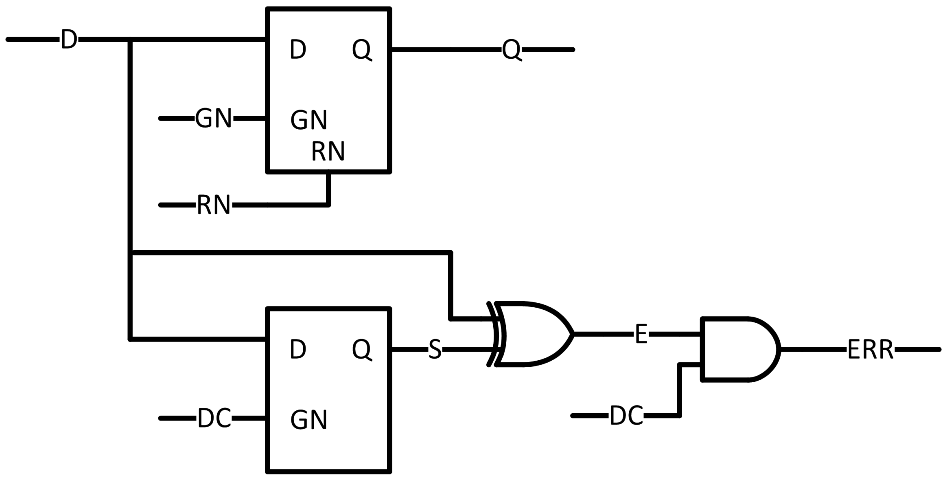


Figure 4.4: Standard cells comprising error detection latch

before time borrowing actually occurs.

As this error detection latch is made of standard cells, it is non-optimal in terms of area usage. Were full view layouts available, the cell could have been more aggressively optimized.

Error signals are combined using a tree of 16 input dynamic OR gates into a single signal representing whether any latch in the core has experienced a timing error. This signal is sent to the timers in the configurable boosting controller.

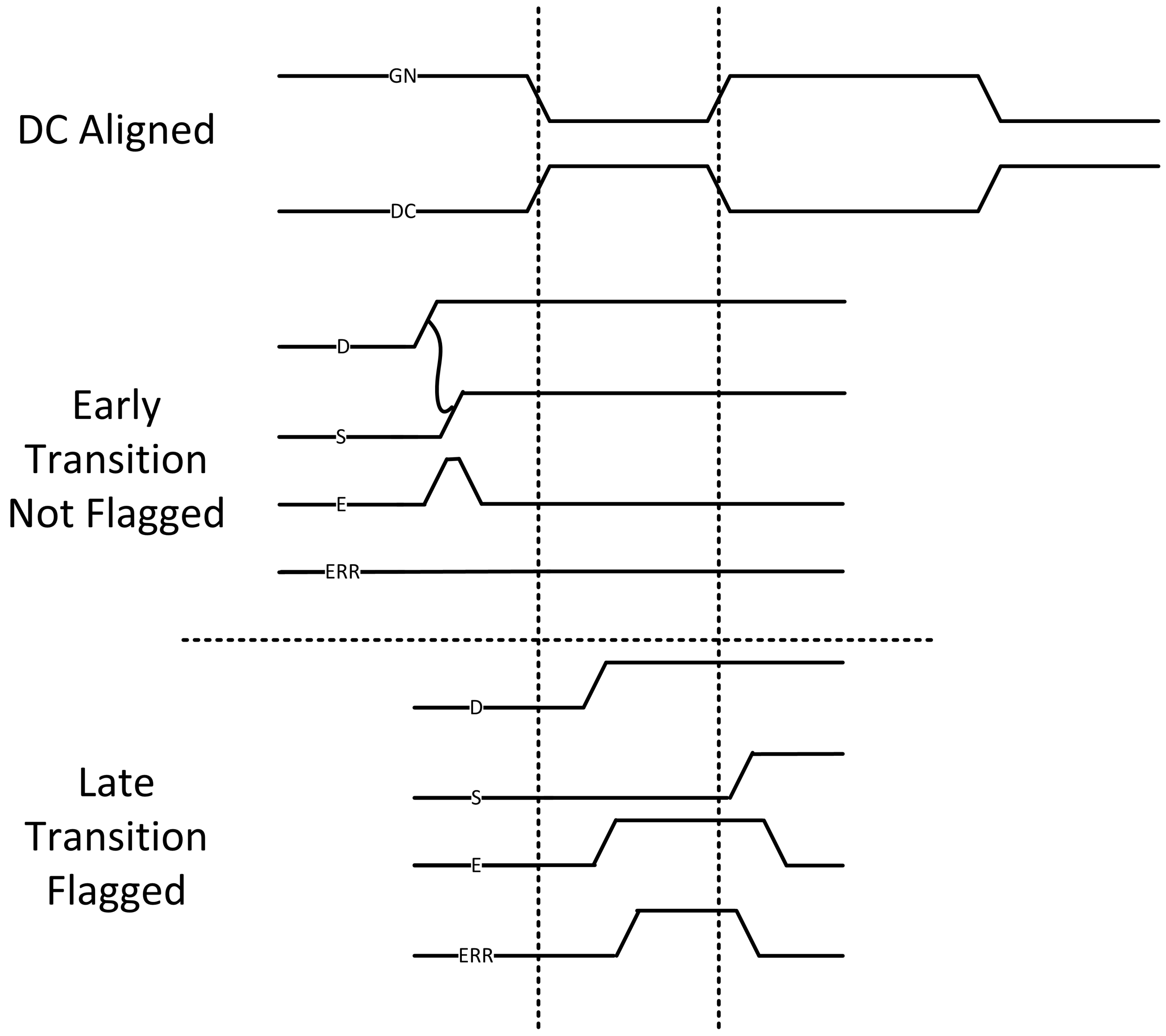


Figure 4.5: With the latch clock and detection clock aligned, the latch behaves like the Bubble Razor latch.

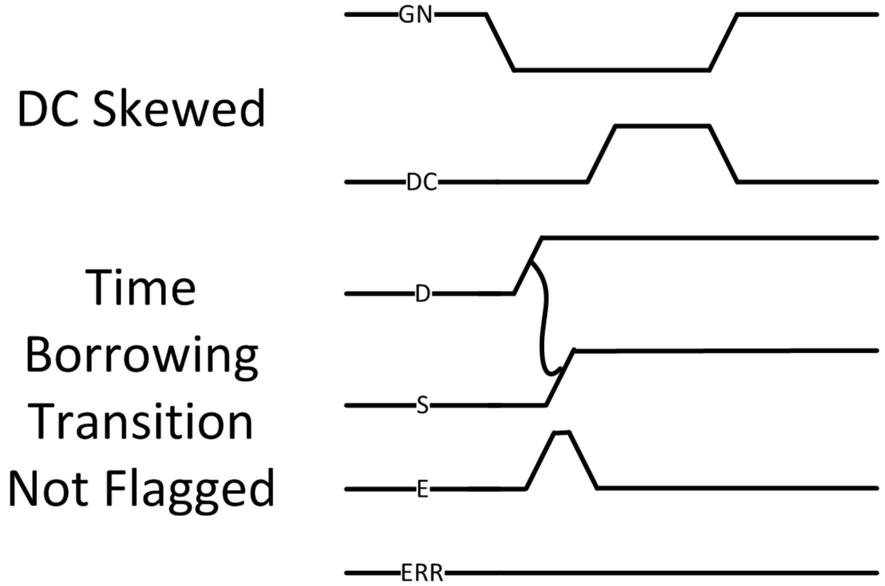


Figure 4.6: By shifting the detection clock DC, the window in which errors are flagged can be changed.

4.5 Three Cores

Three cores, a conventional flip-flop based, latch based, and Voltage Razor core, were implemented with identical timing constraints. For the Razor core, error detection latches were used in place of all positive latches in the design, providing 50% timing speculation with a 44% area increase over the baseline flip-flop design. The area breakdown of the three cores is shown in table 4.1. The majority of the area cost in the Razor core came from error detection, and as mentioned in Section 4.4, this cost could be reduced by optimizing the layout of the error detection latch.

	Flip-Flop	Latch	Razor
Total Area	26044	26302	37478
Combinational Area	15100	16136	17624
Sequential Area	10943	10165	10128
Error Detection Area	-	-	9725
Area from Error OR	-	-	785
Cell Count	22838	26828	28073
# of Flop/Latch	2483	6349	6349
% With Timing Detection	-	-	39.1%

Table 4.1: Comparison of Three Cortex-M3 Processors

Retiming resulted in each original flip-flop becoming an average of 2.55 latches. This did not increase sequential area due to the high area of the flip-flop standard cells in comparison to standard cell latches.

4.6 Clock Generation

The master system clock can be generated from an on-chip PLL, on chip ring oscillator, or be driven externally. After optionally dividing this master clock, it is sent to the logic shown in Figure 4.7 which generates the configurable clocks which are sent throughout the chip.

The clock generation logic contains several configurable delay elements eahc of which can be tapped off at 64 different points. Additional analog tuning is available on the transmission gates in the final 47 delay taps for expanded range of tunability.

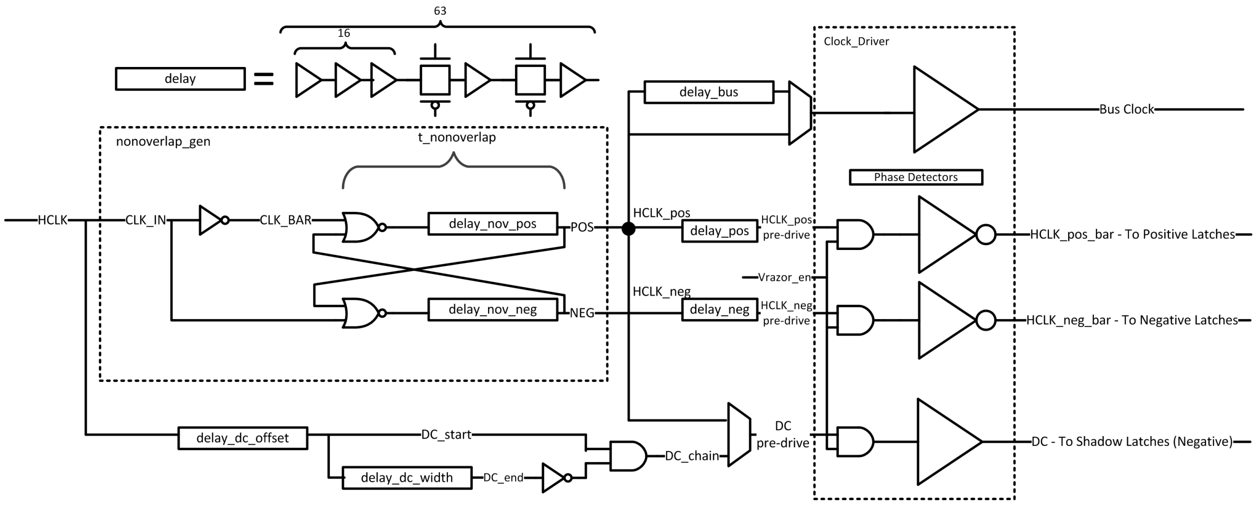


Figure 4.7: Clock generation logic for Razor Core

The master clock HCLK is sent to a non-overlapping clock generator which creates two non-overlapping clocks HCLK_pos and HCLK_neg. The detection clock DC has a tunable offset and width. All clocks have a tunable offset before being sent to clock drivers in the bus logic and Razor core. Similar logic exists for the other cores in the design, though they require a subset of the clocks needed by the Razor core.

A pair of phase detectors exist between every pair of different clocks allowing the determination of the order in which all clock edges arrive with respect to all other edges. Additionally, every clock signal is sampled by an analog waveform sampler which allows their waveforms to be sent off chip.

4.7 Clock Delivery

If clock trees were used, they would open up the potential to clock skew due to variation at low voltage. Additionally, if conventional clock tree cells were placed throughout the Razor core, the clock buffers would see their supply voltage boosted along with the rest of the logic, decreasing clock tree latency and reducing the benefit of boosting the logic. For these reasons, clock meshes are used to provide each latch or flip-flop its clock signal. The meshes are driven by a separate supply, ClockVDD, which is nominally the same voltage as Vhigh.

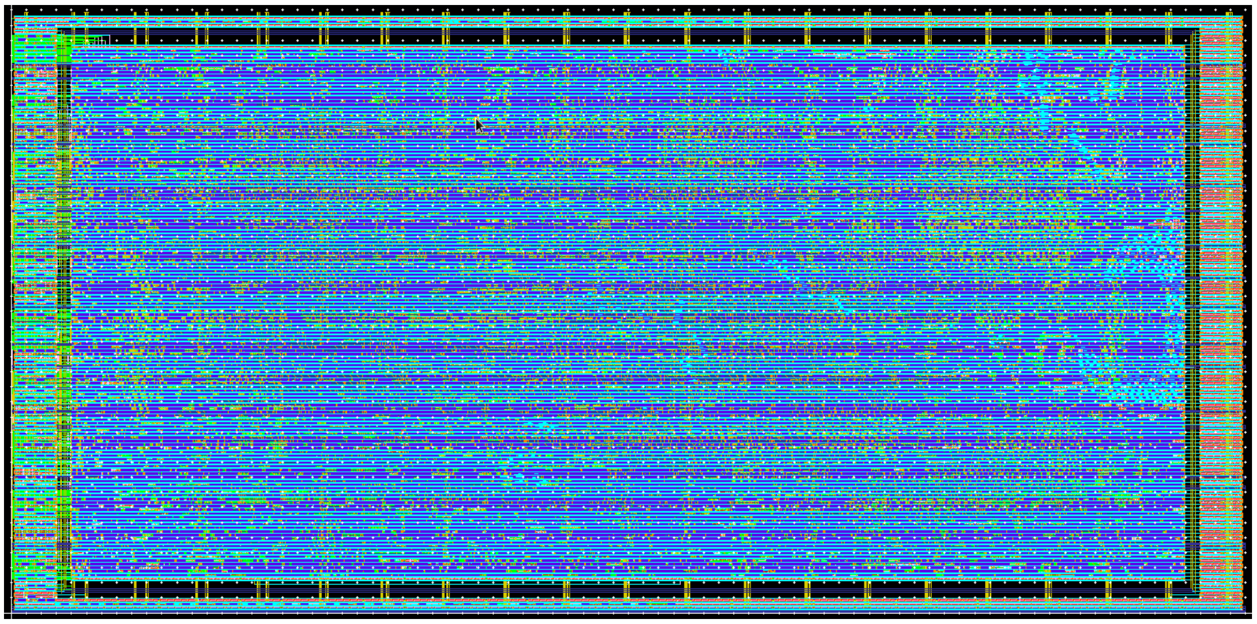


Figure 4.8: Razor Cortex-M3 layout with clock mesh drivers around border.

Figure 4.8 shows the layout of the Razor Cortex-M3. The core logic sits at the center and there is a ring of clock buffers surrounding the design. The supply rails are split between this ring and the main block of logic in the center. Figure 4.9 shows a closeup of the corner of the Razor core. The three rings between the blocks of logic deliver CoreVDD, ClockVDD, and VSS. The three smaller wires in a mesh pattern are HCLK_pos_bar, HCLK_neg_bar, and DC. Clock drivers and latches connect to this mesh at appropriate points based on clock mesh synthesis.

4.8 Interface to Bus Controller

Although detailed analysis of peripheral timing is not required in Voltage Razor as it was in Bubble Razor, the interface between the latch based logic and flip-flop based logic must still be considered. Specifically, if a timing error occurs in the output logic of the latch based core, it will not be able to be detected or corrected by boosting.

To ensure timing errors do not occur on this interface and corrupt system behavior, the timing constraints of the outputs of the latch based cores were tightly constrained. Additionally, it is possible to skew the bus clock relative to the core clock in a permanent

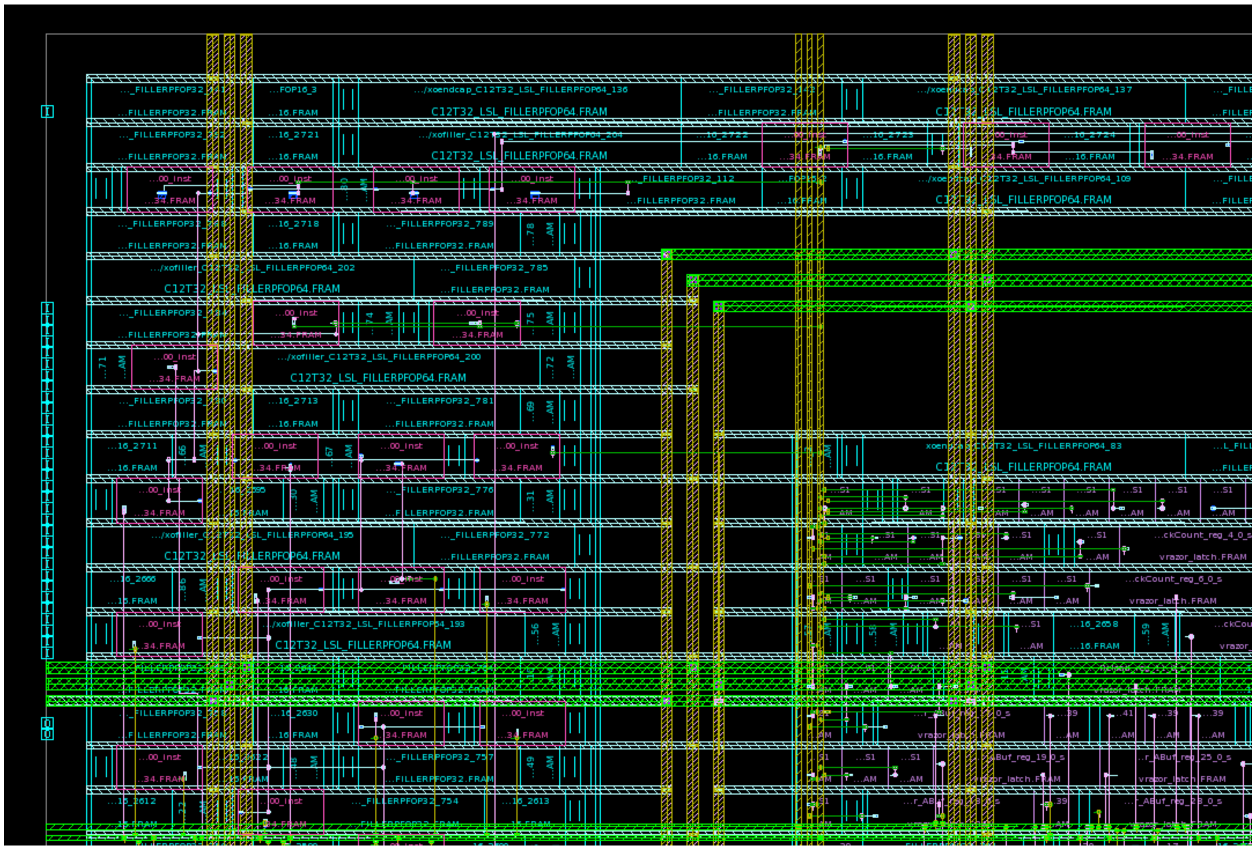


Figure 4.9: Closeup of Razor core's corner and clock mesh drivers

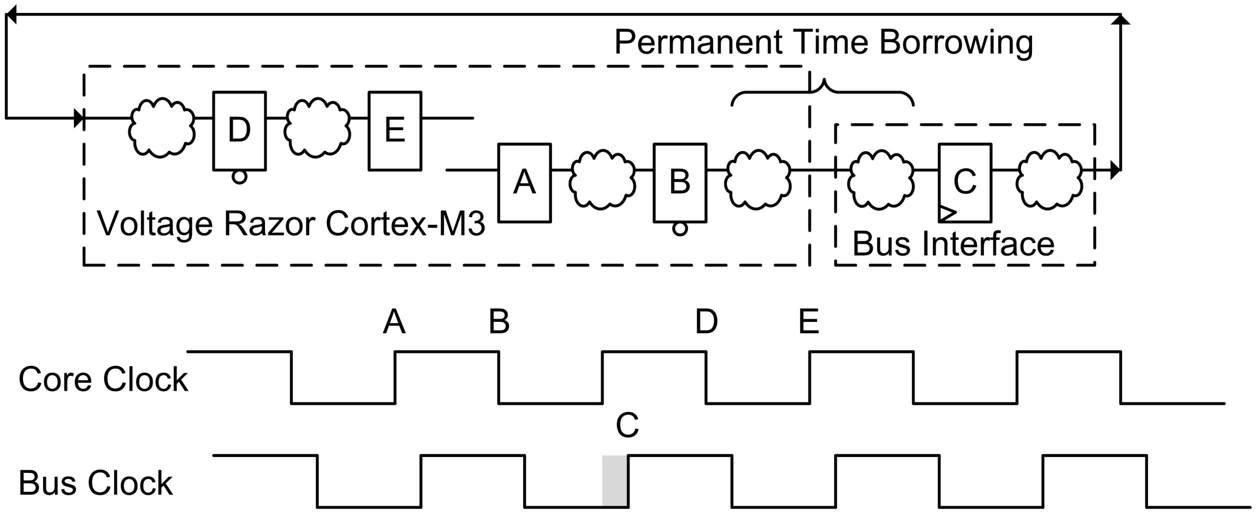


Figure 4.10: Allowing skew between the Bus clock and Core clock can allow for permanent time borrowing at the interface between latch and flip-flop based circuits.

fashion, as in Figure 4.10, allowing for permanent time borrowing. Because of the latch based core, this skew is tolerable and does not aggrivate minimum delay, or hold time, constraints.

4.9 Conclusion

A latch based Razor technique was proposed which uses time borrowing and voltage boosting as an error correction method. This system boosts supply voltage quickly upon detection of a timing error, allowing for the circuit delays to decrease and compensate for the timing error. As with Bubble Razor, the technique is independent of the target design's architecture and Voltage Razor has the additional benefit of not modifying any cycle timing or inserting any stall cycles within the pipeline.

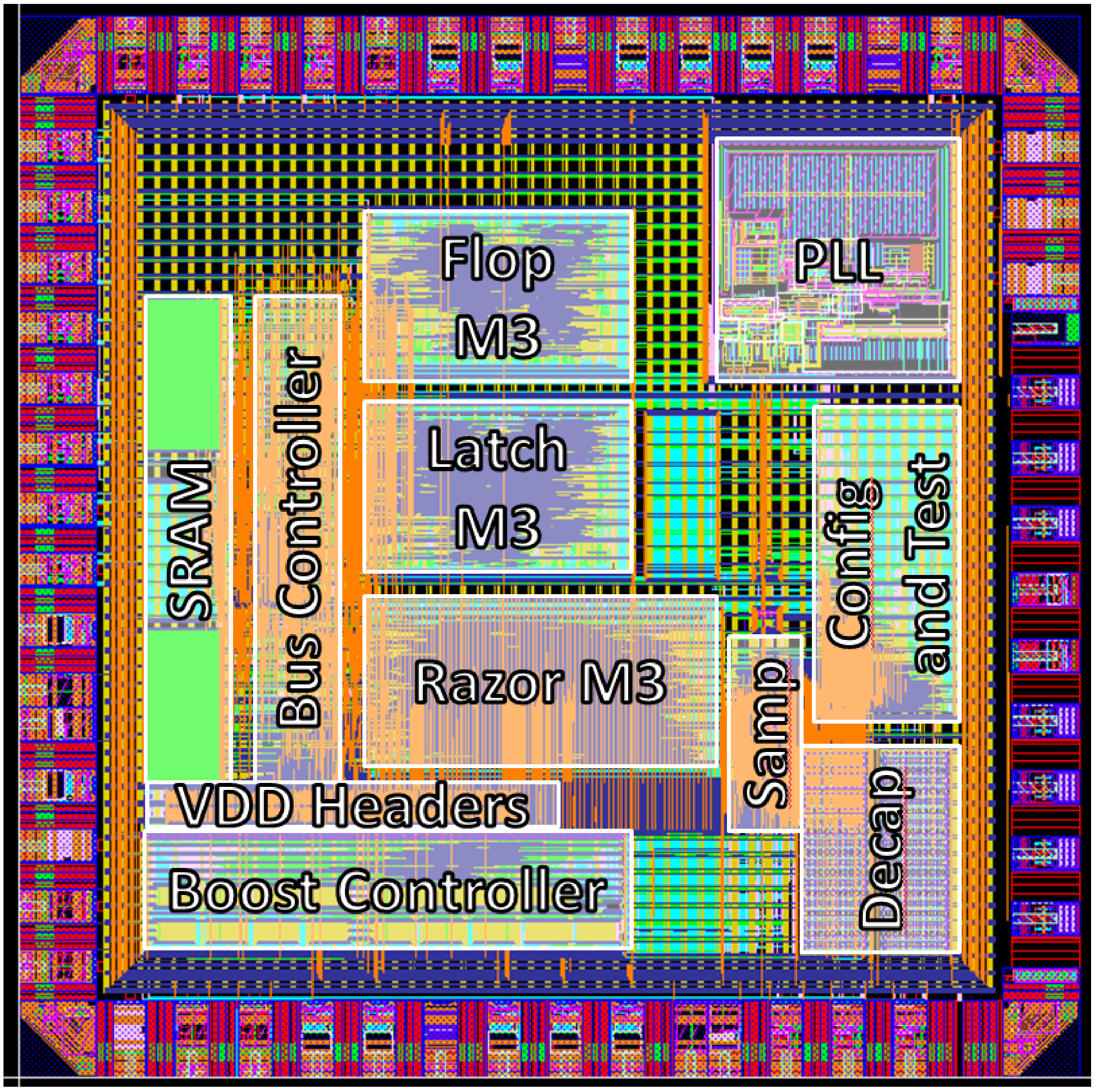


Figure 4.11: Layout picture of 28nm Voltage Razor test chip

CHAPTER 5

Transforming Designs With Clock Gating to Two Phase Latch Timing

Bubble Razor and Voltage Razor both took advantage of two-phase latch based timing, specifically the ability to use time borrowing as error correction, to correct for timing errors in an architecturally independent way. This relied upon the soft edges used in two-phase latch timing, where a timing error can be detected and corrected before it is compounded to the point of system failure. Target designs for Bubble and Voltage Razor only made use of a single clock and thus did not take advantage of clock gating, as clock gating introduces the idea of hard edges to the design in which timing errors immediately corrupt system state. This limited the scope of target designs to which Bubble and Voltage Razor could be applied, as many commercial designs make use of clock gating in order to reduce system power consumption in inactive blocks. Clock gating is likely the most commonly used power reduction technique in industry due to its ease of use and widespread toolchain support.

This chapter proposes a method to transform a flip-flop based traditional design which makes use of clock gating into a two-phase latch based design in which time borrowing can be used as error correction. After this transformation, the design can be easily input into a timing error detection and correction system such as Bubble Razor or Voltage Razor.

5.1 Removing Functional Requirement of Clock Gating

Figure 5.1 shows a diagram of a conventional system with clock gating. In this diagram, flip flops are shown as their constituent master and slave latches. Logic in the datapath computes the signal EN, which is ANDed with CLK to form $GCLK$. The signal EN thus determines whether a different block of logic clocked by $GCLK$ will be given a clock pulse or held in a low power clock gated mode.

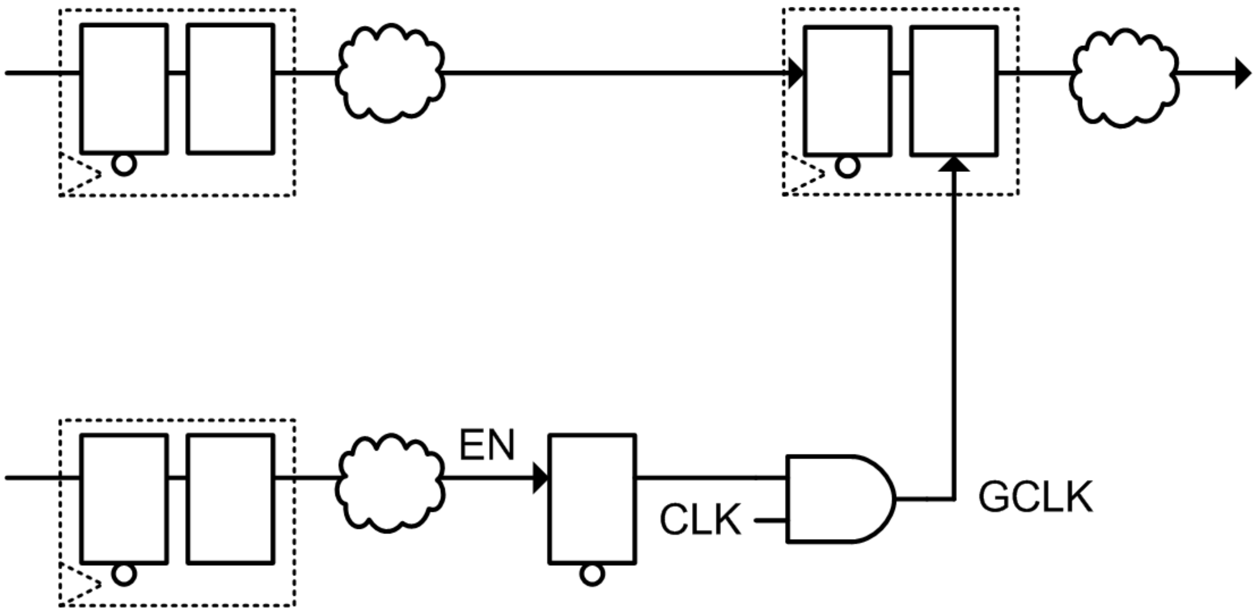


Figure 5.1: Clock Gating in a conventional system where Flip-Flops are drawn as their constituent latches.

If EN is low, $GCLK$ will remain low instead of pulsing and the flip-flops clocked by $GCLK$ will retain their previous values. In some designs this behavior is unnecessary as clock gating is merely used as a power saving technique. In other designs the behavior is required for proper functionality, for example if the EN signal is controlling whether an instruction is stalling in a pipeline stage. If EN were to falsely be high instead of low, $GCLK$ would pulse when it ought to hold low and the flip-flops clocked by $GCLK$ would incorrectly sample new data and lose their old data. If Bubble Razor or Voltage Razor were blindly applied to a design with clock gating, this would present a problem if a timing error occurred on EN. Timing errors in Bubble Razor and Voltage Razor do not immediately affect functionality as latches use soft edges and time can be borrowed from later stages. The signal EN however is sampled on a hard edge. If it is late, the system has already been corrupted as data is immediately lost upon the false $GCLK$ pulse.

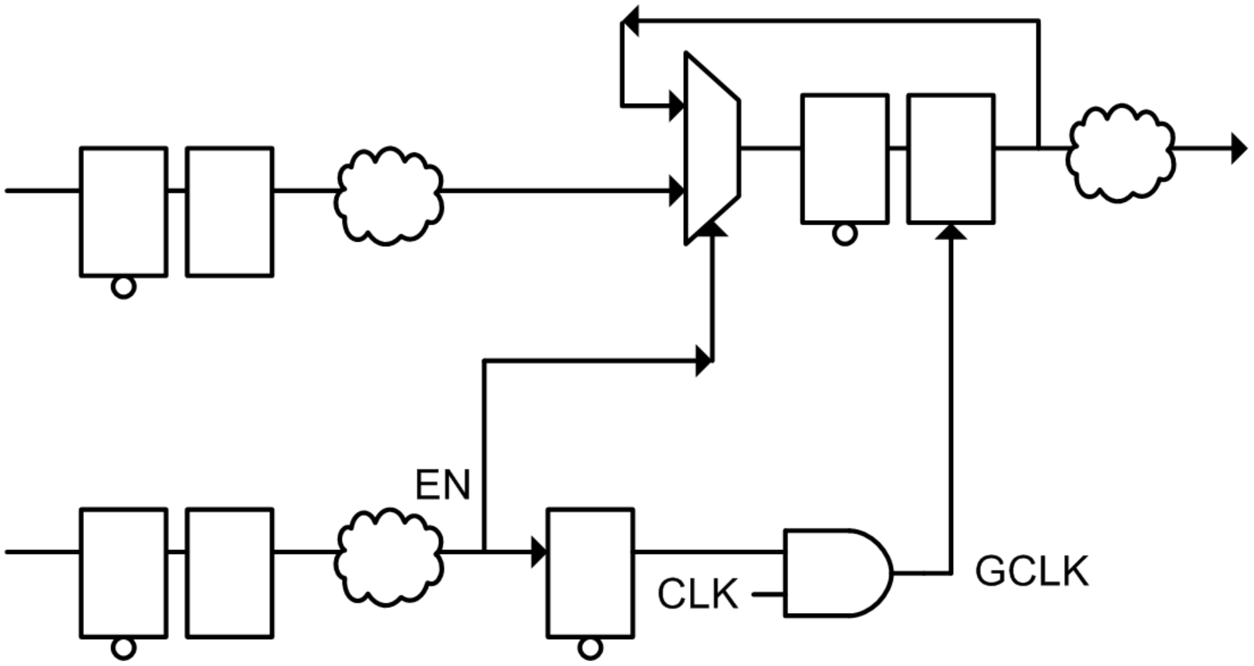


Figure 5.2: Clock Gating and its equivalent redundant logic.

If a stalling behavior is required and clock gating is not used, a loop must exist in the datapath to return data to the stalling flip-flop. Figure 5.2 shows the clock gating logic from Figure 5.1 in addition to a MUX which achieves the equivalent behavior if clock gating were not present by looping the data back to the input of the flip-flop. By adding this redundant path, clock gating is no longer required for functionality. This circuit forms the basis of the two-phase latch based approach to clock gating.

5.2 Retiming With Clock Gating Present

The first step in retiming is shown in Figure 5.3. The negative latch in the gated flip-flop is moved across the mux where it becomes three negative latches on each of the MUX's inputs.

The clock gating cell must be split into a separate negative latch and AND gate, such

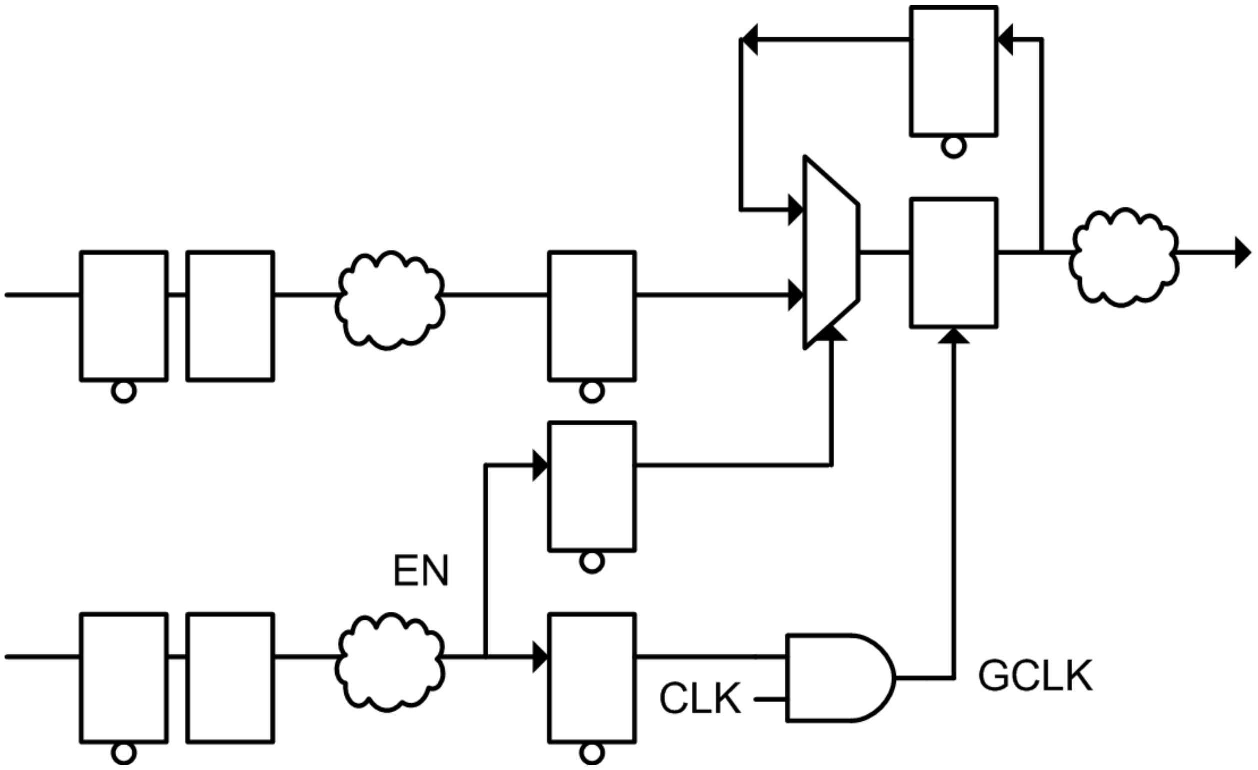


Figure 5.3: The first step of retiming, moving the negative latch across the MUX.

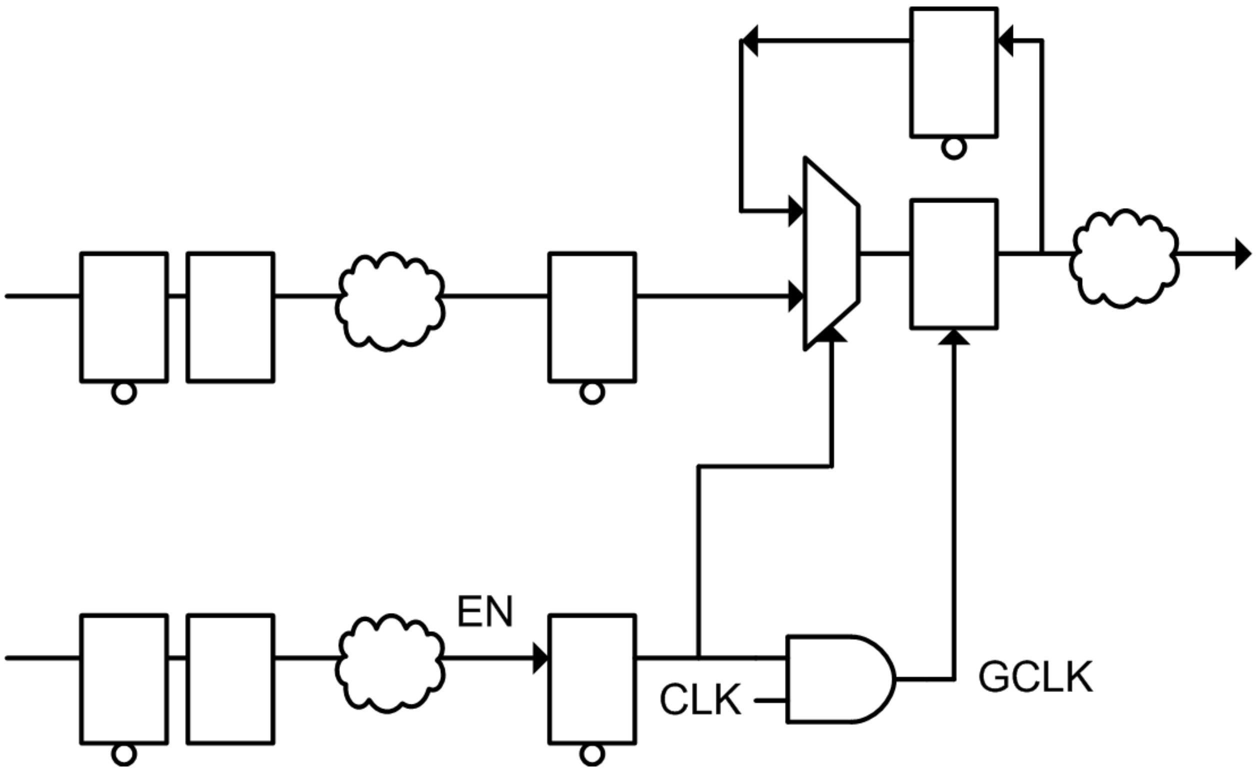


Figure 5.4: The circuit, ready to be retimed using a commercial retiming tool.

that the negative latch can be moved by retiming. The two latches capturing EN then are redundant and can be combined. In this case, negative latches originally clocked by $GCLK$ are instead being clocked by CLK. As clock gating is no longer required for functionality, gated latches can be given a different clock with more pulses than the original, or freely replaced with ungated latches.

If negative latches were given their original gated clocks and resets during retiming, the result could have extremely poor timing as some latches may not be able to be moved at all. This is because commercial retiming tools will not combine flip-flops or latches with different clocks or resets during retiming, leading to situations where the retimer must stop with a suboptimal result. To address this, the proposed retiming flow leaves positive latches in place, holding the spots of the original flip-flops. The negative latches resulting from doubling are not given a reset, as it is only needed in the positive latches, so long as the clock is either held low or ticking while resets are asserted. Each negative latch is then given the master, ungated clock during retiming. Secton 5.3 will discuss how to reassign gated clocks to these negative latches in order to reduce power consumption.

For the purpose of the Bubble Razor propagation algorithm, peripherals such as memories will be treated as a group of positive latches, so in order for the circuit to be of a valid configuration in which positive latches only neighbor negative latches and vice versa, a negative latch must be added to every primary output of the design going to such interfaces. Conceptually, these new negative latches result from splitting the peripherals into their positive and negative components and moving the negative latches into the target design. Primary inputs already neighbor negative latches since the flip-flops they used to neighbor have been split into a negative and positive latch, with the negative latch before the positive one. Interface timing is limited to half a clock cycle outside the target design, since the furthest the bordering negative latches can move are to the inputs or outputs themselves with no logic in-between. Depending on the target design, this may or may not impose a tighter timing constraint on the design.

At this point, the circuit is in a configuration where it can be given to commercial retiming software, as every latch that will be moved had no reset and shares a common clock. The retimer can then balance the latches in the design by either just moving latches, or by a combination of moving and synthesis.

5.2.1 Forward vs Backward Retiming

Although the previously described method performs retiming in the backward direction, it is also possible to perform retiming forward. This can be accomplished by placing negative latches after the positive latches instead of before them, by adding interface latches to the inputs of the design instead of outputs, and by removing the latches in the integrated clock gating cells. Regardless of whether forward or backward timing is performed, there will usually be more negative than positive latches since retiming almost always comes at an overhead and increases the total number of latches. Although both forward retiming and backward retiming are valid methods, from an area and power standpoint it may be more beneficial to perform in a certain direction.

Flip-Flops with critical paths in the original design will lead to negative latches that need to be moved half way between a pipeline stage, but for non-critical paths, retiming will stop moving latches once timing is met, before reaching the halfway point. For paths with a critical delay of less than half the timing constraint, the latches do not need to be moved at all. This means that if backward retiming is performed, there will be more positive latches with low critical delays and in general negative latches will be past the halfway point of their respective pipeline stages. If forward retiming is performed, negative latches will tend to have low critical delays and will appear early in pipeline stages. This will come into play when figuring overhead in two ways: the number of clock gated negative latches and the number of latches that require error detection.

5.3 Reassigning Gated Clocks to Negative Latches

After retiming is complete and the negative latches have been moved, the circuit is still functionally equivalent to the original, however from a power consumption standpoint it is non-optimal since no negative latches make use of clock gating. A post-processing step can be used to analyze the design and assign gated clocks to many negative latches.

It might be thought that each latch can simply be assigned the clock it had before being moved, however this is not possible. Negative latches after retiming can not be one-to-one corresponded to negative latches before retiming, as retiming increases and decreases the number of latches in the design as it moves them across logic gates, and latches are duplicated and combined throughout the retiming process. A negative latch after retiming may be a result of only a single latch from before retiming, or it may result from a combination of many.

This work proposes assigning gated clocks to negative latches by analyzing the latch based netlist. Each negative latch can determine which clock to use based on the clock(s) used either by its fanin positive latches or fanout positive latches. Since retiming can be performed correctly either forward or backward, one might suspect that gated clock assignment could also go either way. The following sections detail the implications of both approaches.

5.3.1 Assigning Gated Clocks to Negative Latches Based on Fanout

Conceptually, it makes sense for negative latches to look forward to determine how to gate their clocks. If one were to analyze a design immediately after splitting flip-flops into their constituent latches, it is readily apparent that each negative latch shares the gated clock of the positive latch immediately following it, thus if negative latches were to determine their clock assignment based on fanout, the netlist would remain unchanged.

Stalling Negative Latches Low vs High

If a single master clock is distributed throughout the design, it is clear that when a clock pulse is gated and the gated clock is held low, the positive latches which are driven by that clock are held in their closed state while the negative latches driven by that clock are held in their open, transparent state.

However, in latch based designs it is also common to distribute two non-overlapping clocks which go to positive and negative latches respectively, as this can relax the hold time constraints of the logic, reduce overhead from buffer insertion, and increase skew-tolerance. With separate, non-overlapping clocks, it is not readily apparent whether negative latches must be held in their open state when they are gated, or if it would also be possible for them to remain closed while gated.

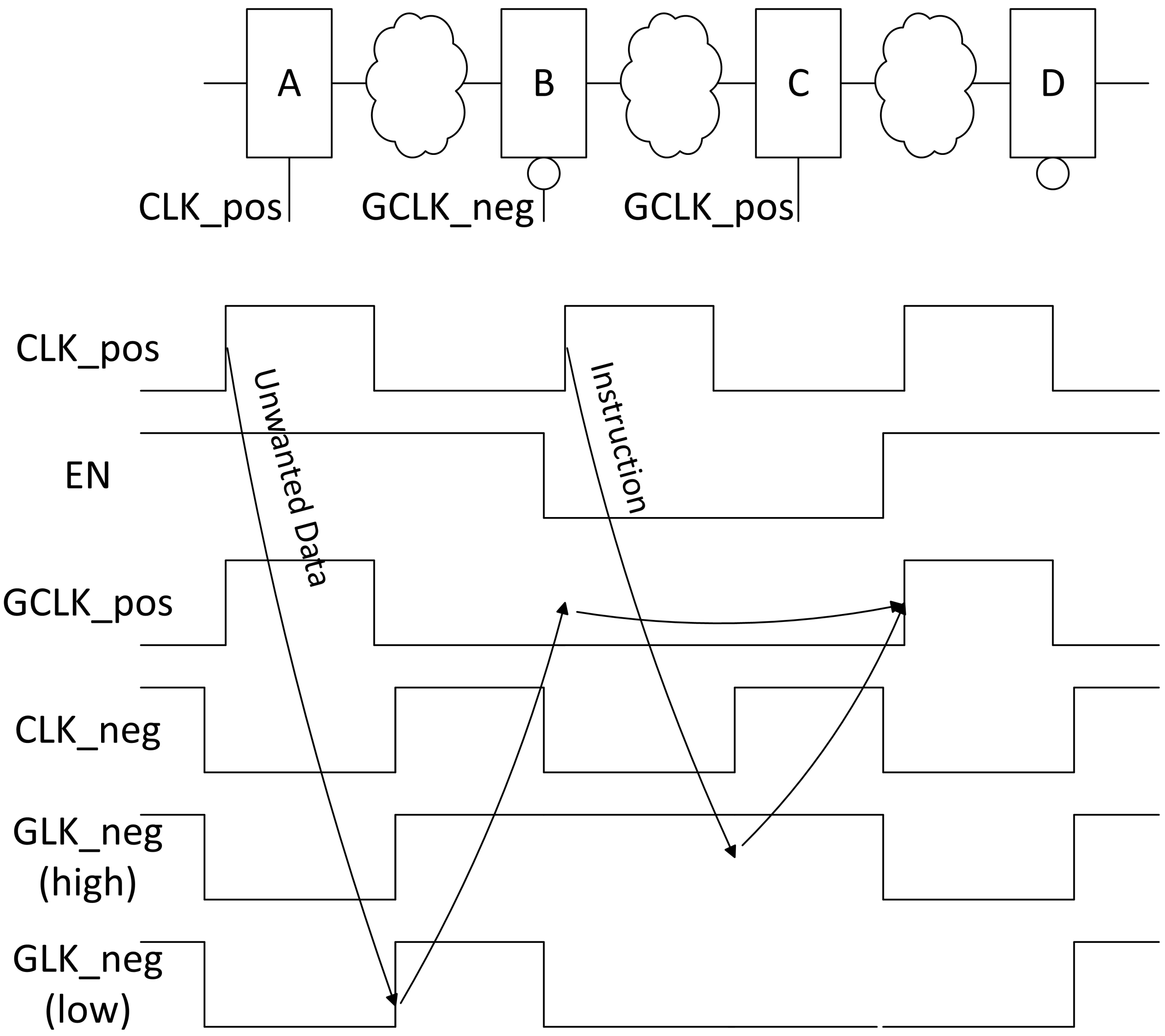


Figure 5.5: When Looking Forward to Assign Clocks to Negative Latches, They Must be Held Open When Clock Gated

Figure 5.5 shows the behavior of a pipeline in which negative latch B determined its clock based on its fanout latch C. The two possible scenarios of stalling open and stalling closed are considered. In this example, during the second clock cycle, EN is low and $GCLK$ is gated. The desired behavior is for latch C to hold its data during the second cycle, and in the third cycle capture the data that was held in latch A in the second cycle. The data held in latch A during the first cycle is to be ignored.

If $GCLK_{neg}$ is held open during the gated, second cycle, as would be the case if only a single $GCLK$ was distributed, the data held in latch A during the second cycle is passed through the open latch B and arrives at latch C during the third cycle, as desired. However, if $GCLK_{neg}$ is held closed while gated, latch B continues to hold the unwanted data that was in latch A in the first cycle. In this second case, latch C captures data from the wrong cycle, and the system behaves incorrectly. This example shows that when negative latches determine their gated clocks based on fanout. it is necessary for them to remain open while they are held in a gated state.

5.3.2 Preventing Race Conditions

Since negative latches are held transparent while gated, they are open during a time in which negative latches are normally closed. If gated clock assignment was done incorrectly, this could open up the possibility of race conditions which would compromise correct behavior.

A race condition could occur due to an incorrectly gated negative latch if a path which contained a gated negative latch existed between two open positive latches during the positive phase of the master clock. In other words, the race condition could occur if three latches in a row were transparent at the same time. At this time, were said negative latch ungated it would be closed and the two positive latches would be disconnected, thus the race would not occur.

To ensure this race condition can not occur, one must guarantee that at all times where a negative latch is gated open, either all positive latches in the fanin of this negative latch are gated closed or all positive latches in the fanout of the negative latch are gated closed. During timing error free operation, this is accomplished by basing the clock on fanout positive latches. If all of a negative latch's fanout positive latches use the same gated clock, the negative latch will also use that clock. If any fanout latches use the master clock, the negative latch must also use the master clock. If the fanout latches use several different gated clocks, then the negative latch must either use the master clock, or use a gated clock whose enable is at least the OR of all the fanout latches' clocks' enables.

Although race conditions cannot occur during timing error free operation, it is possible for them to occur on a late rising transition of EN as shown in Figure 5.6.

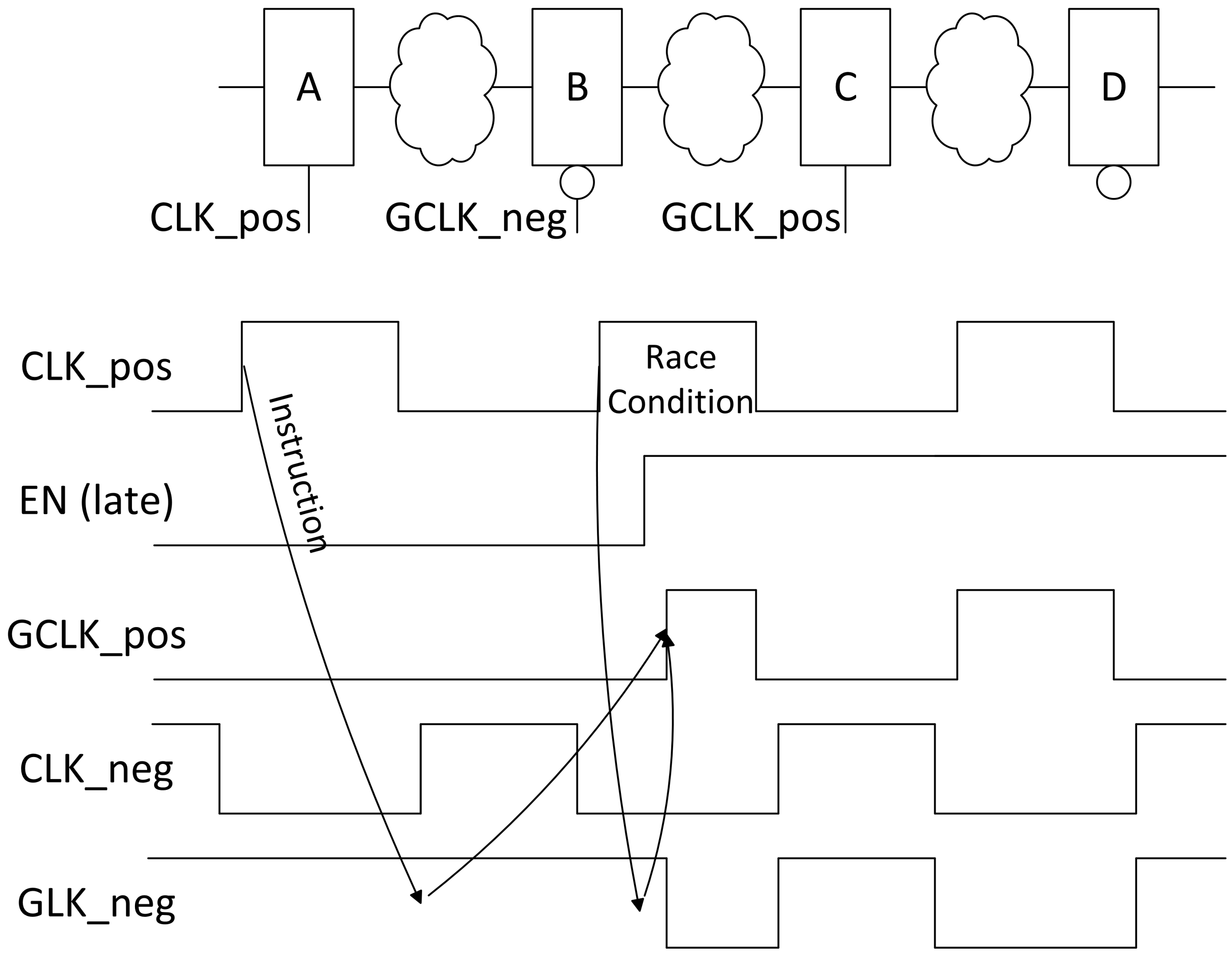


Figure 5.6: Late EN causes race condition

The late arriving EN signal delays the rising edge of $GCLK_{pos}$ and falling edge of $GCLK_{neg}$, however the rising edge of CLK_{pos} remains unchanged. Although the addition of muxes as mentioned in Section 5.1 allows the system to behave correctly if a late EN signal creates an undesired clock pulse, it does not allow for the system to behave correctly were a late EN signal to prevent a desired clock pulse. Therefore, the shortened pulse on $GCLK_{pos}$ is a requirement for the system to operate correctly.

Thus, when latch A opens during the second cycle, it can send data to latch B which is still transparent and overwrite the Instruction that was supposed to be held. This race condition can be overcome by hold time constraints in the datapath, however the magnitude of the constraint needed is linked to the speculation window. This is highly undesirable, as one of the main benefits of Bubble Razor and Voltage Razor is the breaking of the relationship between minimum delay and speculation window.

The fundamental source of this race condition is the fact that negative latches are held in their open and transparent state when gated, which is required for correct operation when negative latches look to their fanout positive latches to determine their clock.

5.3.3 Assigning Gated Clocks to Negative Latches Based on Fanin

The alternative, less intuitive, route is to assign gated clocks to negative latches based on their fanin positive latches. As with the previous case, in order to ensure correct operation, each negative latch must be ungated during any cycle in which any of its fanin positive latches are ungated.

Stalling Negative Latches Low vs High

As shown in Figure 5.7, there is no longer a change in circuit behavior if negative latches stall open or closed. The desired behavior is for latch C to capture the data held in latch A in the first cycle during both C's second and third cycle. Whether B is open or closed during the second cycle does not change the circuit's behavior.

As shown in Figure 5.8, late arriving EN signals which cause shortened clock pulses do not cause problematic race conditions. If $GCLK_{neg}$ holds open while gated, the non-overlap period between $GCLK_{pos}$ and $GCLK_{neg}$ has been lost, however a race condition could only occur in the conventional way due to clock skew. The potential race is not aggrivated by the speculation window. If $GCLK$ holds closed while gated, no race can occur at all.

Because of this, we propose that negative latches look to their fanin stages to determine whether to use a gated clock, that non-overlapping clocks are used, and that negative latches hold closed while clock gated. This also fits nicely into the Bubble Razor propagation algorithm in which negative latches must hold closed while stalled.

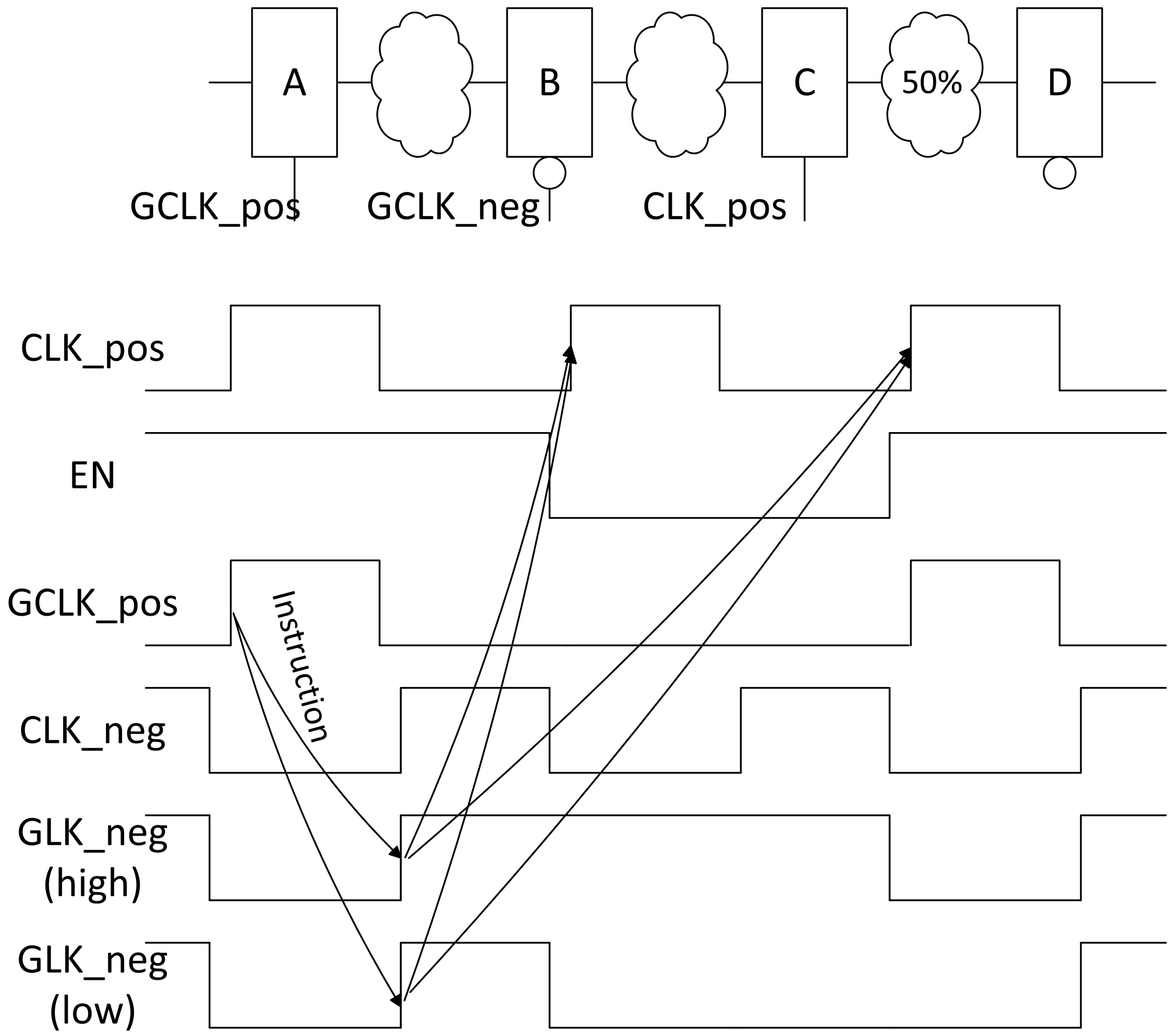


Figure 5.7: When Assigning Clocks Based on Fanin, Negative Latches can Hold Open or Closed when Gated

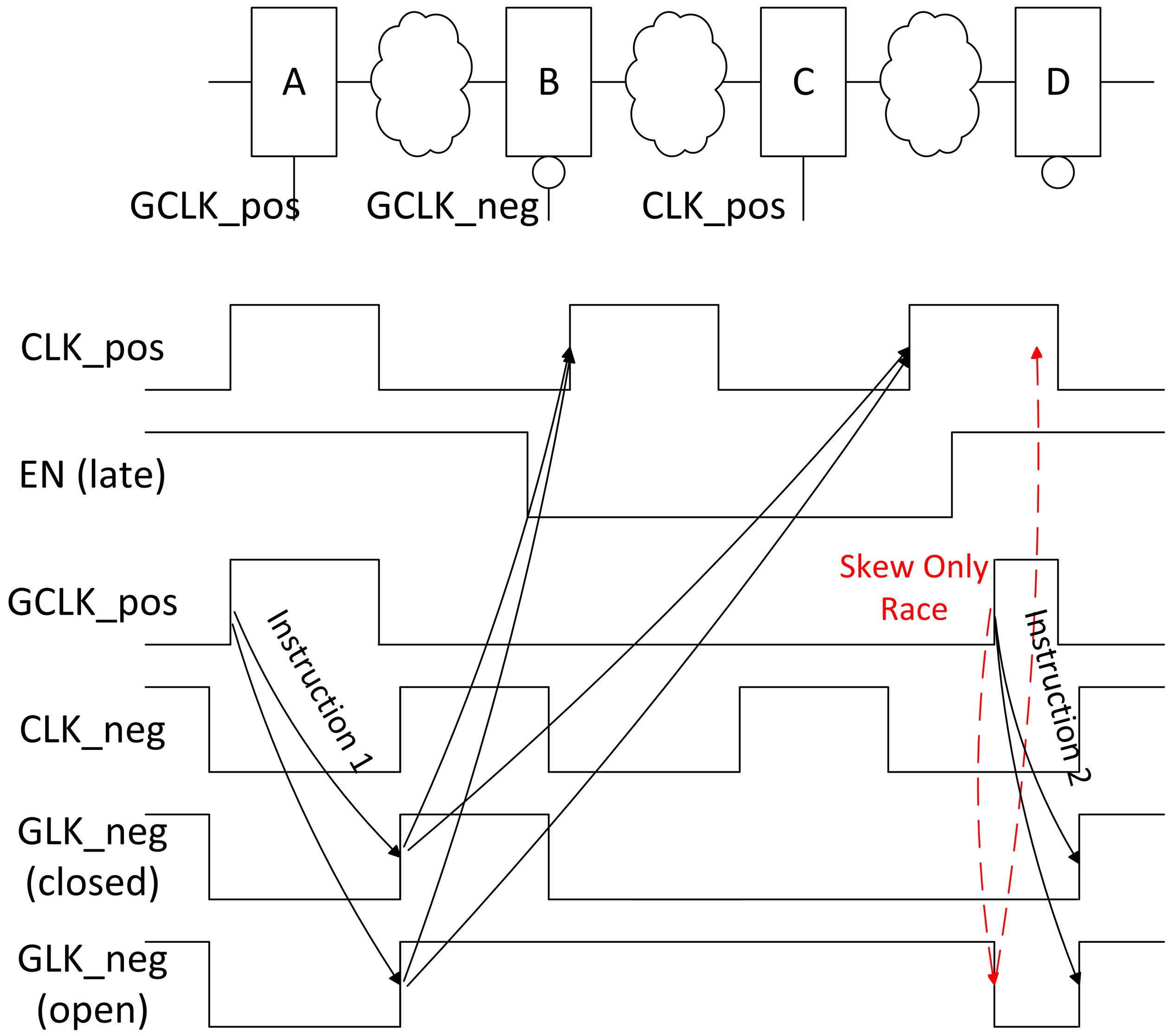


Figure 5.8: Late arriving EN signals do not cause race conditions.

5.3.4 Preventing Timing Loops

Another concern that could come into play when assigning gated clocks to negative latches is the formation of timing loops. This can result when a latch's output is used to calculate its own clock gate enable signal.

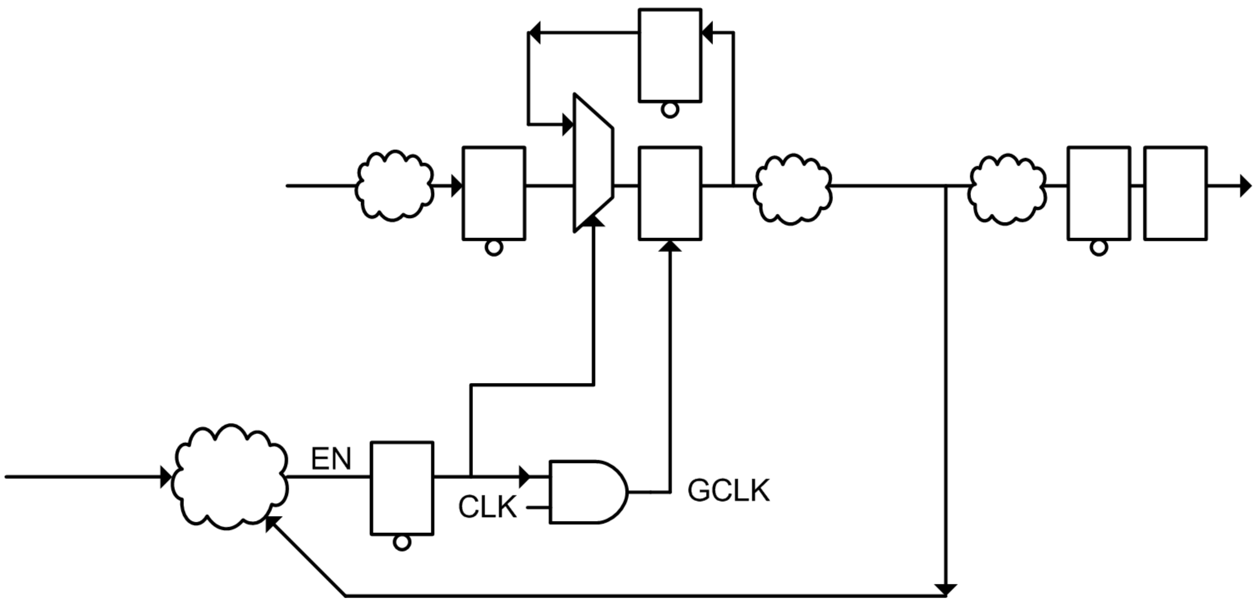


Figure 5.9: After transforming to latches, and prior to retiming.

As negative latches move back in the design, eventually a datapath latch is combined with a latch that originated in a clock gating cell. At this point, if it looks forward to its fanout and sees the clock gating AND gate, it assigns itself the master, ungated clock and no timing loop exists.

If the negative latch instead only looks backward to determine its clock, it will assign itself the gated clock and create a timing loop.

Depending on the clocking scheme used and circuitry used to perform the actual clock gating, the appearance of a timing loop may not actually be problematic to the circuit, although synthesis tools may still identify the loop. Specifically, if following the suggested method of non-overlapping clocks where negative latches hold closed while gated and using the proposed circuitry, the loop is not problematic. In this case, it is only activated when CLK_{pos} is high and thus by definition, CLK_{neg} is low, breaking the loop at the AND gate.

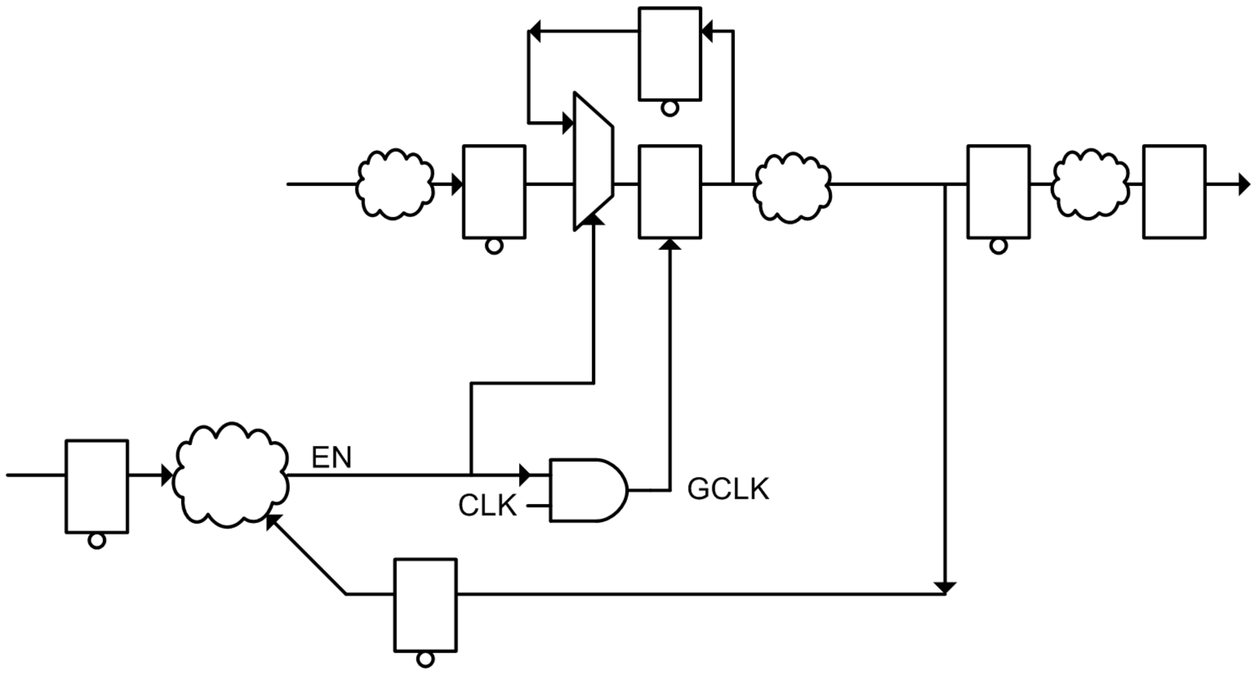


Figure 5.10: Mid-retiming with negative latches moving back

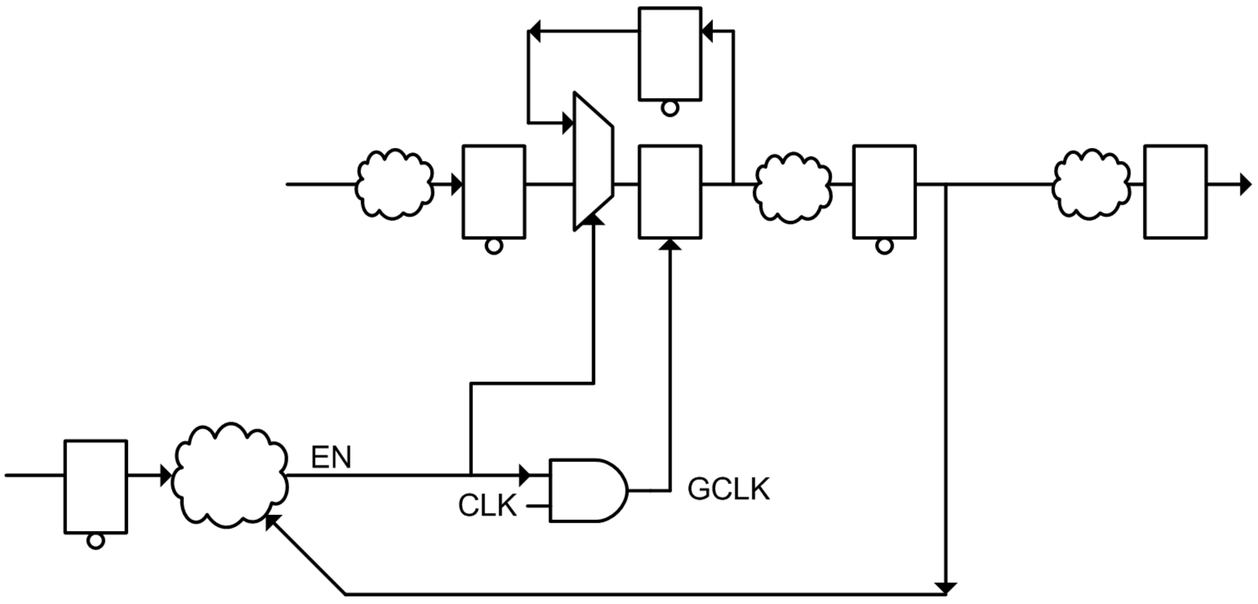


Figure 5.11: After retiming, if negative latches look forward to determine gated clocks, no timing loop exists since a clock gate is in the fanout of the moving latch.

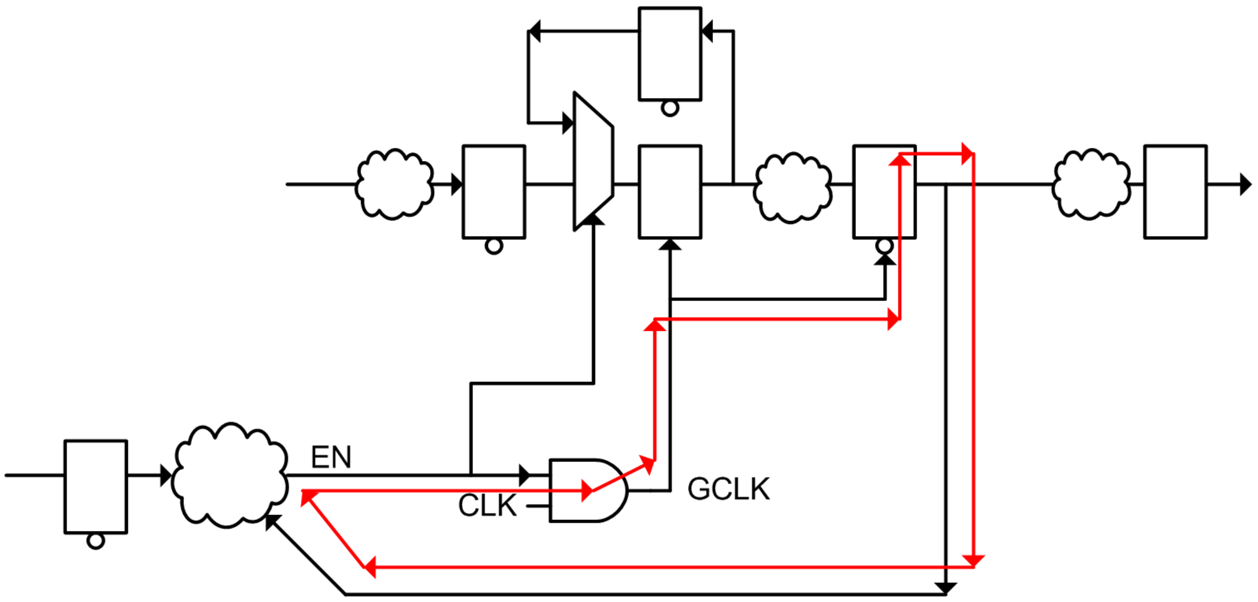


Figure 5.12: If negative latches do not look forward to determine clock assignment, timing loops can be created.

5.3.5 Algorithms for Looking Forward to Assign Clocks to Negative Latches

Therefore, to optimally assign gated clocks to negative latches, Algorithm 1 is performed.

Algorithm 1 Optimally Assign Gated Clocks to Negative Latches

```
for all $n$ = Negative Latches in Netlist do
  if There Are Clock Gates in the Fanout of $n$ AND Timing Loops are Not Acceptable then
    $n$ uses master ungated clock
  else
    for all $p$ = Positive Latches in Fanin of $n$ do
      Unique List $l$: Add $p$'s clock enable ($TRUE$ if $p$ ungated)
    end for
    $n$ uses clock enabled by OR of all elements in $l$
  end if
end for
```

Alternatively, to not add to the number of gated clocks in the system, Algorithm 2 is performed.

Algorithm 2 Assign Gated Clocks to Negative Latches Without Creating New Gated Clocks

```
for all $n$ = Negative Latches in Netlist do
  if There Are Clock Gates in the Fanout of $n$ AND Timing Loops are Not Acceptable then
    $n$ uses master ungated clock
  else
    for all $p$ = Positive Latches in Fanin of $n$ do
      Unique List $l$: Add $p$'s clock enable ($TRUE$ if $p$ ungated)
    end for
    if List $l$ only contains one element then
      $n$ uses clock enabled by element in $l$
    else
      $n$ uses master ungated clock
    end if
  end if
end for
```

As mentioned in Section 5.2.1, retiming in the forward or backward direction may be more beneficial from a power standpoint. This is now apparent since forward retiming tends to result in negative latches appearing earlier in pipeline stages, thus they will tend to have fewer fanin positive latches. This will tend to reduce the number of clock enables in list l and therefore reduce the amount of switching that must take place on the clock of each negative latch.

5.4 Clock Generation

Clock generation is accomplished through custom circuitry in order to deliver glitch free clocks to the various latches, even under the presence of timing failures.

Conventional flip-flop based circuits prevent glitches on gated clocks by using integrated clock gate cells, or ICGs, which are made up of an AND gate which shuts off the clock when not enabled, and a negative latch to sample EN during the low phase of the clock before the rising edge. With this setup, EN is guaranteed to arrive on time, before the positive edge of the clock, though it may transition at any time that does not violate setup or hold constraints. Without the latch, any transition or glitch on EN during the positive phase of the clock would cause a glitch on the gated clock, which would cause catastrophic failure of flip-flop operation.

With the non-overlapping two-phase clocking proposed in Section 5.3.3, glitches on the various positive and negative gated clocks will not cause incorrect system operation, however they are undesirable from a power standpoint as they represent unnecessary switching. Additionally, as EN is no longer required to arrive before the positive edge of the positive clock, the desired behavior is slightly different than in a conventional flip-flop based system.

Figure 5.13 shows the proposed clock timing for a two-phase latch based circuit in which time borrowing is allowed and EN is allowed to arrive late. The circuitry which generates these signals is shown in Figure 5.14. First, the two main ungated clocks, CLK_{pos} and CLK_{neg} are out of phase and non-overlapping, as in a conventional latch based design. They are generated from a standard non-overlap generator driven by a regular clock CLK. EN, a clock gating enable signal is computed from logic following a negative latch, and thus may transition any time between when CLK_{neg} opens and the end of the speculation window, some time after the positive edge of CLK_{pos}.

In the event EN arrives after the positive edge of CLK_{pos}, $GCLK_{pos}$ still needs to pulse,

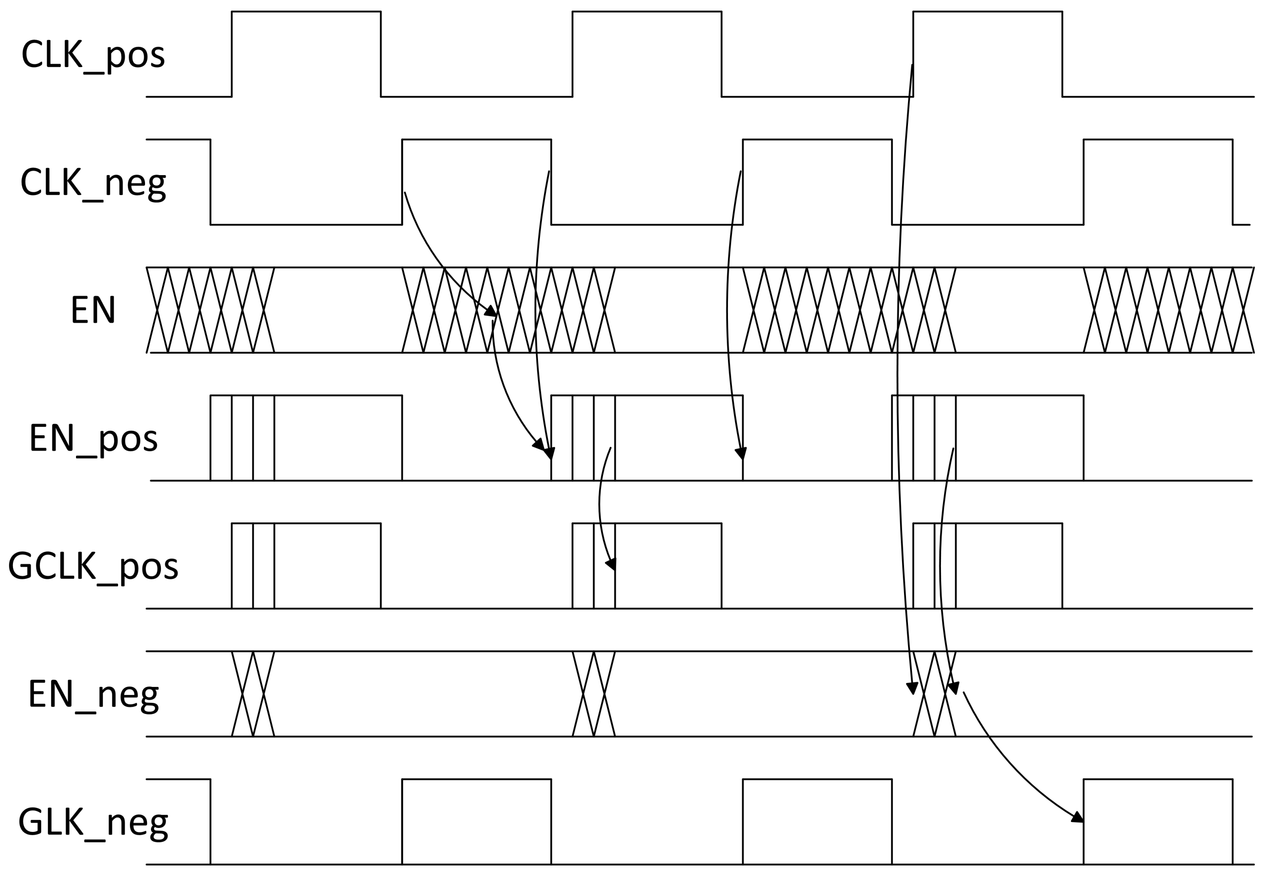


Figure 5.13: Proposed Clock Timing for Latch Based Design with Clock Gating

although it will be shorter than during error-free operation. This behavior is different than the conventional flip-flop case, where an ICG cell would ignore such rise on EN. In order to achieve this behavior, as well as prevent glitches on $GCLK_{pos}$, the EN De-Glitch circuit is used to generate a glitch-free signal EN_{pos}, which is ANDed with CLK_{pos} to create $GCLK_{pos}$. This circuit resets EN_{pos} every time CLK_{neg} is high and evaluates it any time EN is high while CLK_{neg} is low. Thus, any late glitches on EN will result in a glitch-free clock pulse on $GCLK_{pos}$, and a late arriving EN will result in a shortened pulse on $GCLK_{pos}$.

The signal ANDed with CLK_{neg} to create $GCLK_{neg}$, EN_{neg}, is sampled using a latch similarly to how an ICG works in a conventional circuit which uses clock gating. This ensures glitch-free behavior and a full pulse for $GCLK_{neg}$.

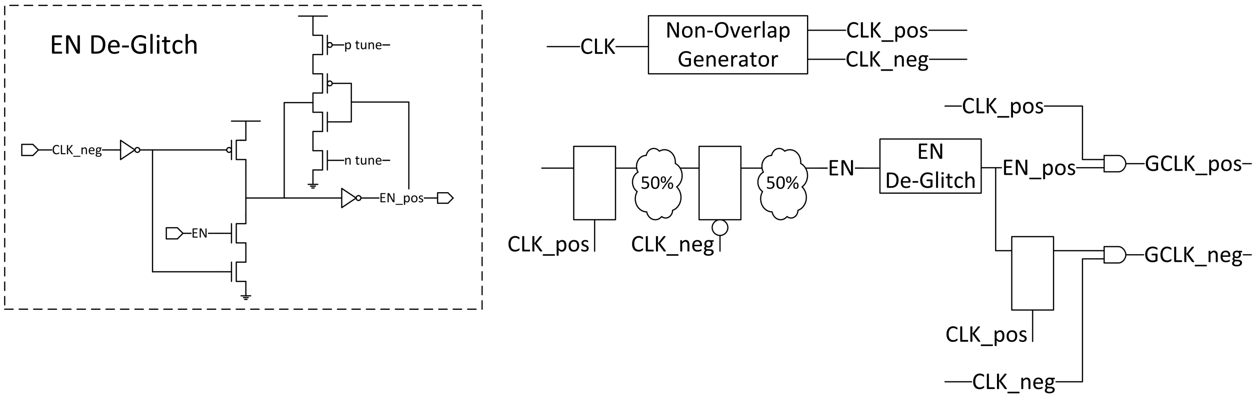


Figure 5.14: Proposed Circuits for Clock Generation

5.5 Timing When Late Arriving EN is Falling

As mentioned in Section 5.1, the addition of a MUX to a clock gated negative latch can feed back data in the event the clock incorrectly opens. Figure 5.15 details the case where a late arriving EN signal causes an unwanted pulse on $GCLK_{pos}$, but the addition of the MUX and latch E allow for correct operation.

The desired behavior is for EN to arrive before the positive edge of CLK_{pos} and gate off $GCLK_{pos}$ during the second cycle. The data that was captured by latch A in the first cycle is then supposed to be held through the second cycle and captured by latch C in the second and third cycles. Because of the late arriving EN signal (past the negative edge of CLK_{neg}), the clocking circuitry mentioned in Section 5.4 causes $GCLK_{pos}$ and $GCLK_{neg}$ to both experience full clock pulses as if they had not been clock gated.

Halfway through the first cycle, latch E opens and captures the data that was held in latch A during the first cycle. If not for latch E, this data would be lost when latch A incorrectly opens due to the timing error on EN. In addition to controlling the clocking, EN is also the select signal for latch A's input MUX. When EN transitions low, the MUX switches from selecting A's normal input to the value held in latch E, and latch A recaptures the value it had held in the first cycle. This is then passed onto latch B which also experiences an undesired pulse on its clock due to the timing error. At the start of the third cycle, latch C captures the data from latch B, thus the correct behavior is achieved despite the timing error causing an unwanted tick of $GCLK_{pos}$ and $GCLK_{neg}$.

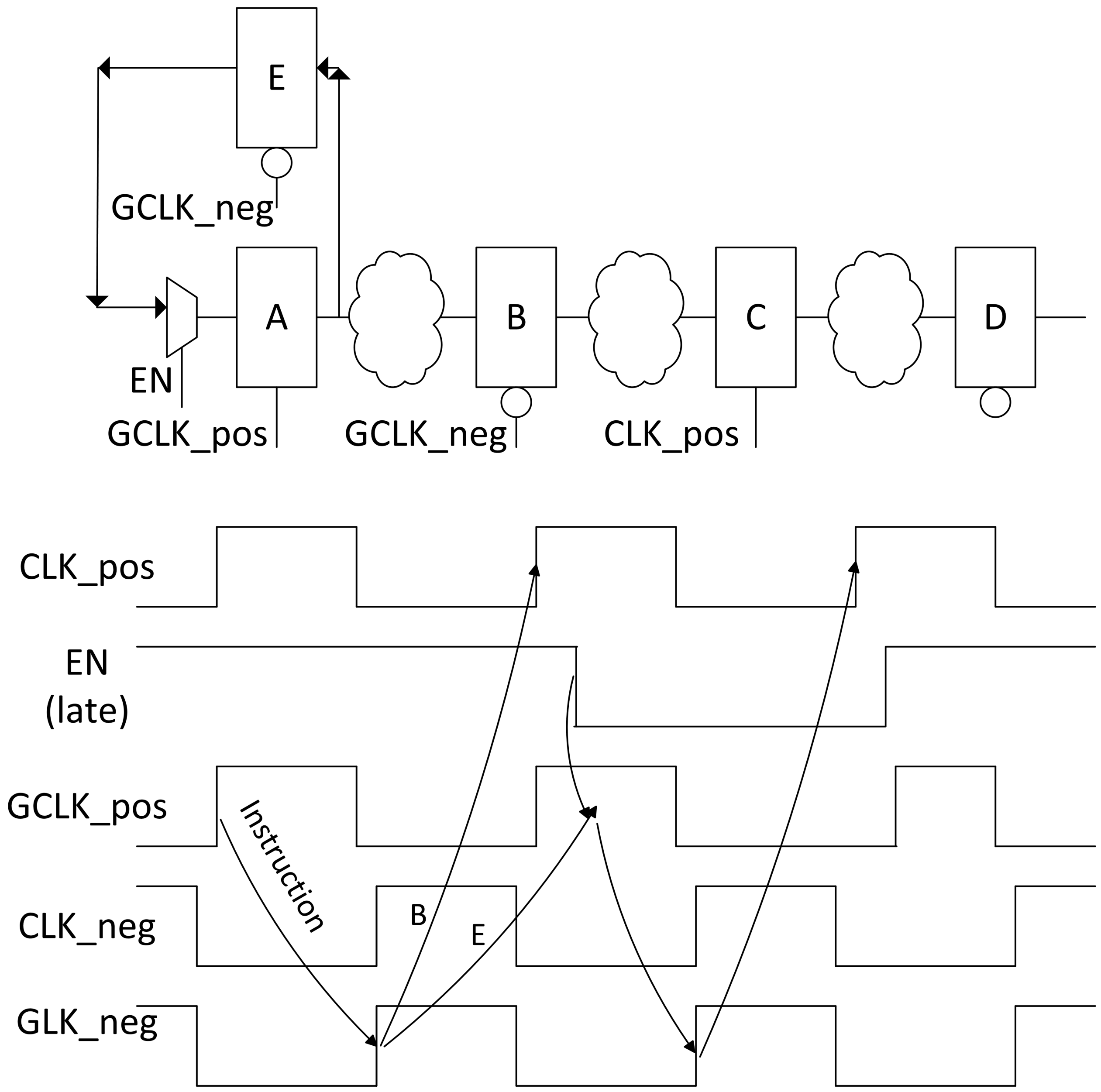


Figure 5.15: The addition of latch E and the MUX allows for correct behavior when $GCLK_{pos}$ opens incorrectly

5.6 Metastability on GCLK

Previously, it was shown that late arriving and early arriving rising and falling transitions on EN can be correctly accounted for. The only remaining case is a transition which results in metastability on the gated clock, which could potentially corrupt data. The troublesome transition is a falling transition on EN exactly as EN is sampled, causing a glitch or bad pulse on $GCLK$. In order to be immune to this metastability, the timing of the clock tree and MUX must be such that by the time the latches receiving the gated clock receive the glitch or bad pulse, both their input and the value they are storing are identical, therefore the clock signal has no effect.

Figure 5.16 shows the case in which GCLK may glitch or have a bad pulse due to metastability. In this case, EN is shutting off right as it is sampled.

If the delay from EN to the end of the $GCLK$ clock tree is greater than the delay of EN to $Dmux$, then both the input to the latch and output of the latch will hold the same value at the time the glitch or bad clock arrives, and the clock will not have an effect on the circuit. This is illustrated in Figure 5.17. Based on circuit simulation, the delay of the clock tree must be at least 50ps in 45nm SOI CMOS in order for the circuit to be metastability immune. This is a reasonable delay to expect from a clock tree, and the circuit can be constrained at a low cost to have at least this much delay in the clock buffering.

5.7 Timing Error Detection and Bubble Neighbors

With regards to timing error detection, dynamic OR gates that combine errors can be clocked by the gated clocks of the latches they are dealing with respectively, reducing power consumption while logic blocks are gated. Late falling transitions on EN will cause late transitions on positive latches and be detected, however late rising transitions on EN will not be detected inherently, so error detection should be added to all the clock gating enable signals in the design such that errors cannot compound to the point of a failure without being detected.

No modifications to Bubble Razor's neighbor finding are needed, since positive latches

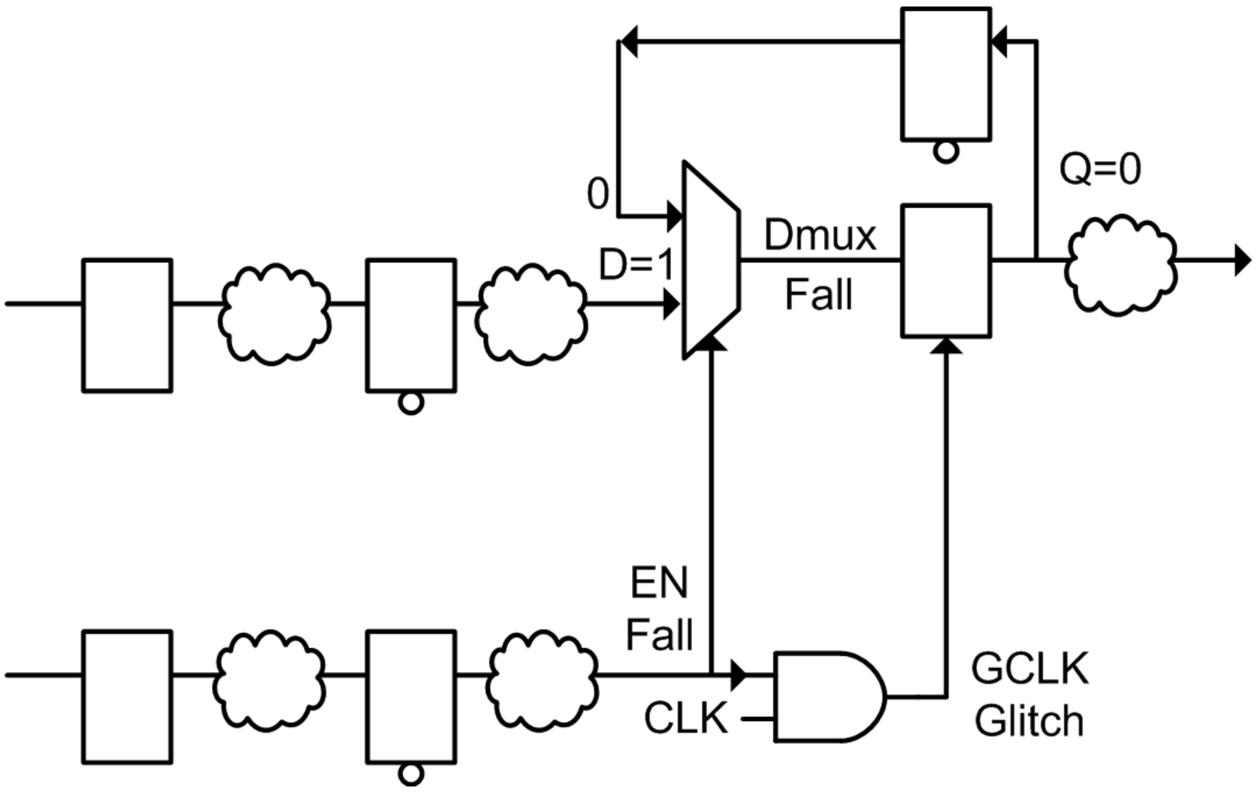


Figure 5.16: EN may be sampled at a time which causes metastability on GCLK

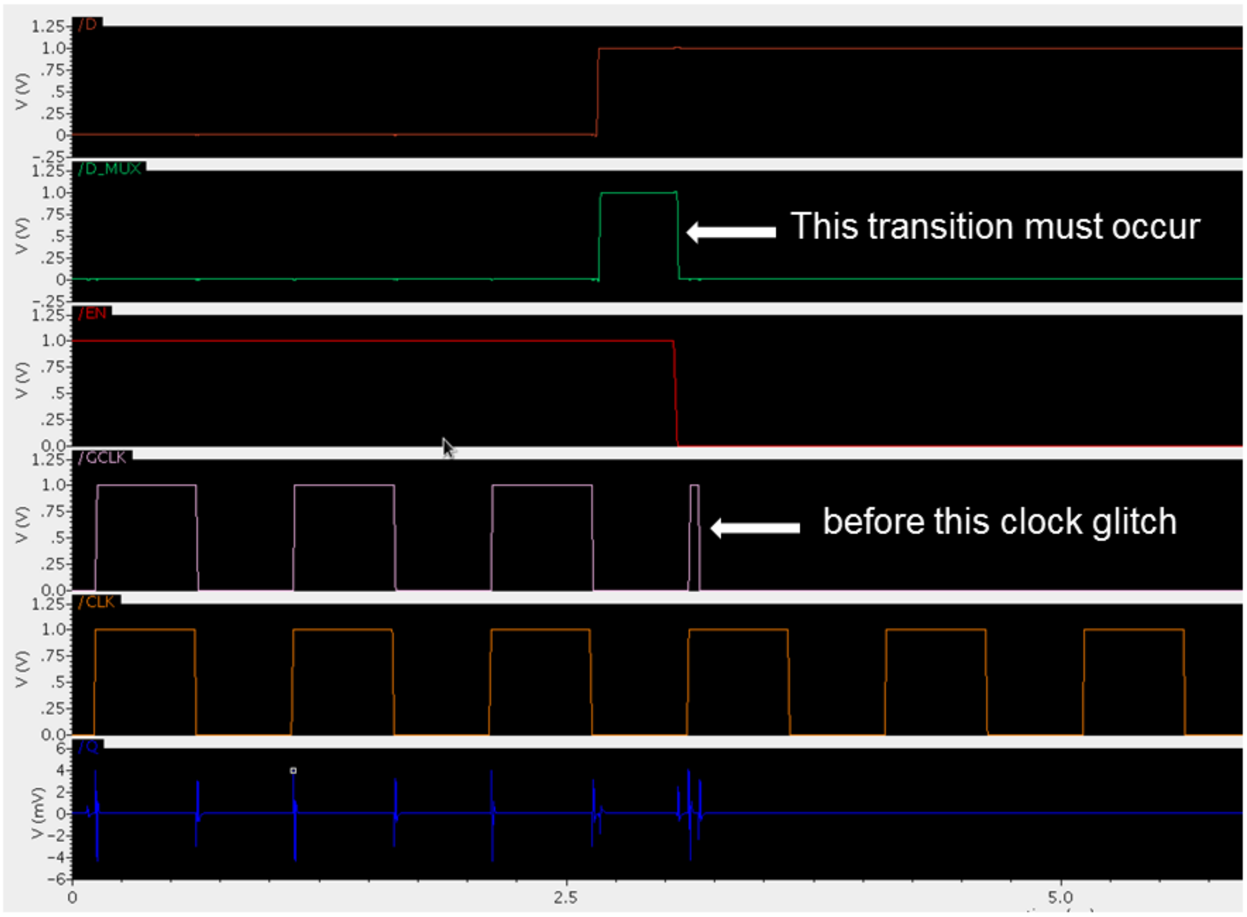


Figure 5.17: The circuit is metastability immune so long as the output of the MUX is stable before the glitch or bad pulse on GCLK

will automatically become neighbors with the negative latches driving their clock gate enable through the MUXes that were added. After transforming the clock gating as discussed in this chapter, the fact that clocks are not always ticking is somewhat irrelevant to the Bubble Razor transformation process.

5.8 Conclusion

The transformation of a flip-flop based design using clock gating into a two-phase latch based design which also uses clock gating and allows for time borrowing has been discussed. This process can be used to prepare traditional clock gated designs for use with either Bubble Razor or Voltage Razor, or simply be used for the skew-tolerant benefits of latch based design.

CHAPTER 6

Applying Bubble Razor to the ARM Cortex-R4

This chapter discusses the application of Bubble Razor on an ARM Cortex-R4, a larger commercial processor than the ARM Cortex-M3, as well as a processor that uses clock gating as a power saving technique. As Bubble Razor is targeted towards commercial processors, work has been done to address issues common to industrial circuit implementations. This chapter addresses the differences in the Bubble Razor flow and approach, compared to Chapter 3.

6.1 System Overview

The ARM Cortex-R4 processor [31] is a popular industrial processor used in a wide variety of applications, complete with caches, tightly coupled memory, and an AXI bus. Figure 6.1 shows a block diagram of the Bubble Razor Cortex-R4. The processor logic itself has been transformed with the clock gating and Bubble Razor transformation flow. This processor interfaces with several SRAMs using the interface described in Section 2.4.2. An asynchronous interface exists between the Coresight DAPLITE, the debug unit, and between the NIC-400, an AXI bus controller. The bus controller, the NIC-400, allows access to main memory as well as various memory mapped IO and GPIO pins. The register file has been pulled out of the processor in order to avoid transforming it, the interface for which is described in Section 6.5.

In order to allow for performance and power comparisons, four Cortex-R4 processors are

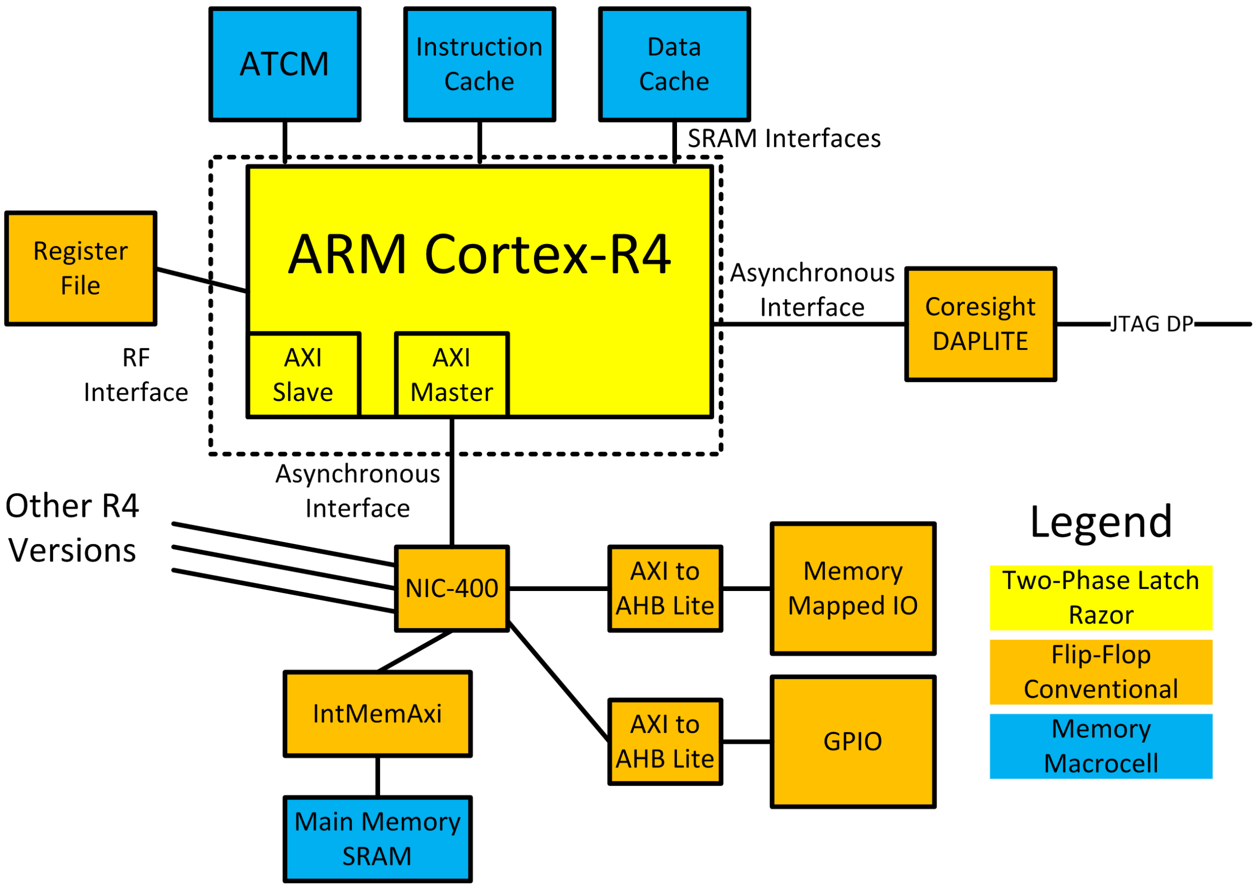


Figure 6.1: Block Diagram of ARM Cortex R4 based Bubble Razor Implementation

implemented on chip. All four processors are connected to the NIC-400 bus controller. The cores consist of a baseline flip-flop core, a baseline two-phase-latch core, a Bubble Razor core using an experimental low-overhead error detecting latch, as well as a Bubble Razor core using the proven latch from Chapter 3.

6.2 Retiming the Cortex-R4

Because of the clock gating transformation flow, the results of retiming are somewhat different than in the retiming of the Cortex-M3 in Section 3.5. Retiming is accomplished in four major steps:

1. Initial Flip-Flop based design, after using a script to pull out the register file, caches, and reset synchronizers into a different level of hierarchy

2. Input design to retiming tool. Muxes are added to clock gated latches, as discussed in Section 5.1. Retiming is performed using flip-flops in place of latches, resulting in a higher apparent area.

3. Post-retiming design, using flip-flops in place of latches and a clock twice the target frequency.

4. Final Two-Phase Latch based design, after replacing flip-flops with latches

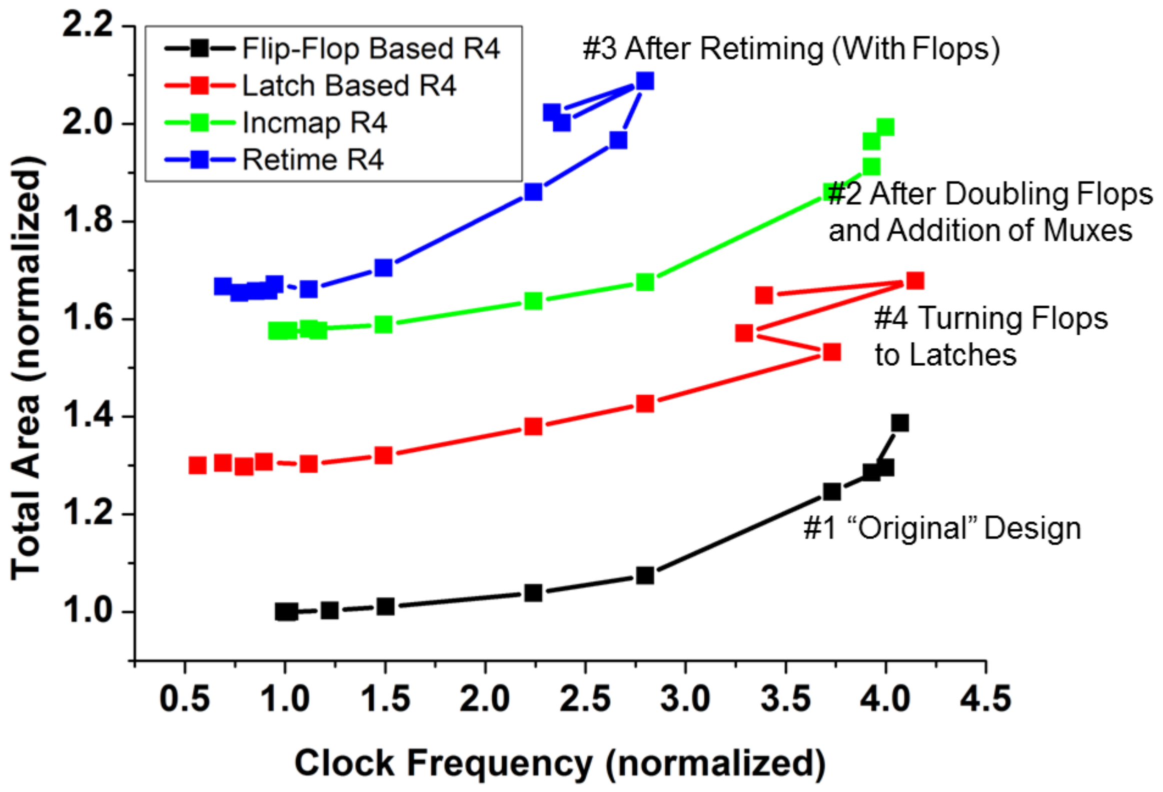


Figure 6.2: Total Area of Cortex-R4 at various steps in latch transformation

Figure 6.2 plots total area vs clock frequency for various timing constraints throughout the various steps in the transformation process. Figure 6.3 shows just the area resulting from combinational logic, ignoring the area of flip-flops or latches.

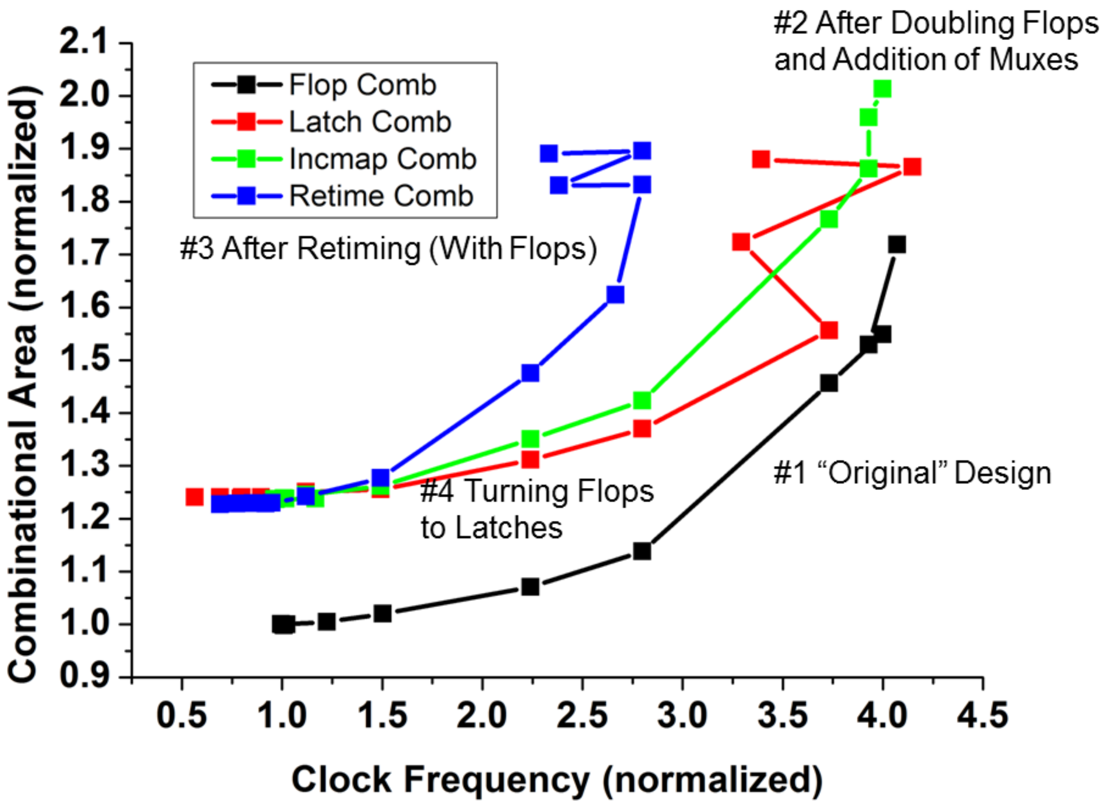


Figure 6.3: Combinational Area of Cortex-R4 at various steps in latch transformation

When not using an aggressive timing constraint, the addition of muxes on clock gated latches increases combinational area by 24%, and total area by 13%. However, as timing becomes more constrained, more combinational area is required to increase the performance of the flip-flop baseline design than the final two-phase latch based design. The combinational area of the best performing latch based design is only 8.5% higher than the best performing flip-flop based design, and the latch based design has 2% higher performance.

This effect can also be seen when considering total area. At the most relaxed timing constraint, the latch based design has 30% higher total area than the flip-flop based design, however with the highest performing designs, the area penalty is only 21%.

When reassigning gated clocks to negative latches, the approach follows Algorithm 2. After assignment, 83% of negative latches use a gated clock, while 13% of latches get assigned the master clock because they had fanin positive flops with differing clocks. The remaining 4% are assigned the master clock because all their fanin positive flops used the master clock.

6.2.1 Error Detection and Speculation Window

Figure 6.4 shows a histogram of critical delays for the positive latches when targeting a loose timing constraint, while Figure 6.5 shows the same data for the highest performing latch based design. To achieve 20% speculation on the loose design, 73% of positive latches require checking, while on the highest performing design 89% of the latches require checking. In either case, by checking 100% of positive latches, 50% speculation can be achieved. Although the distributions are very different in shape, both have most latches within the 60% to 100% of path delay, thus there are very few non-critical positive latches.

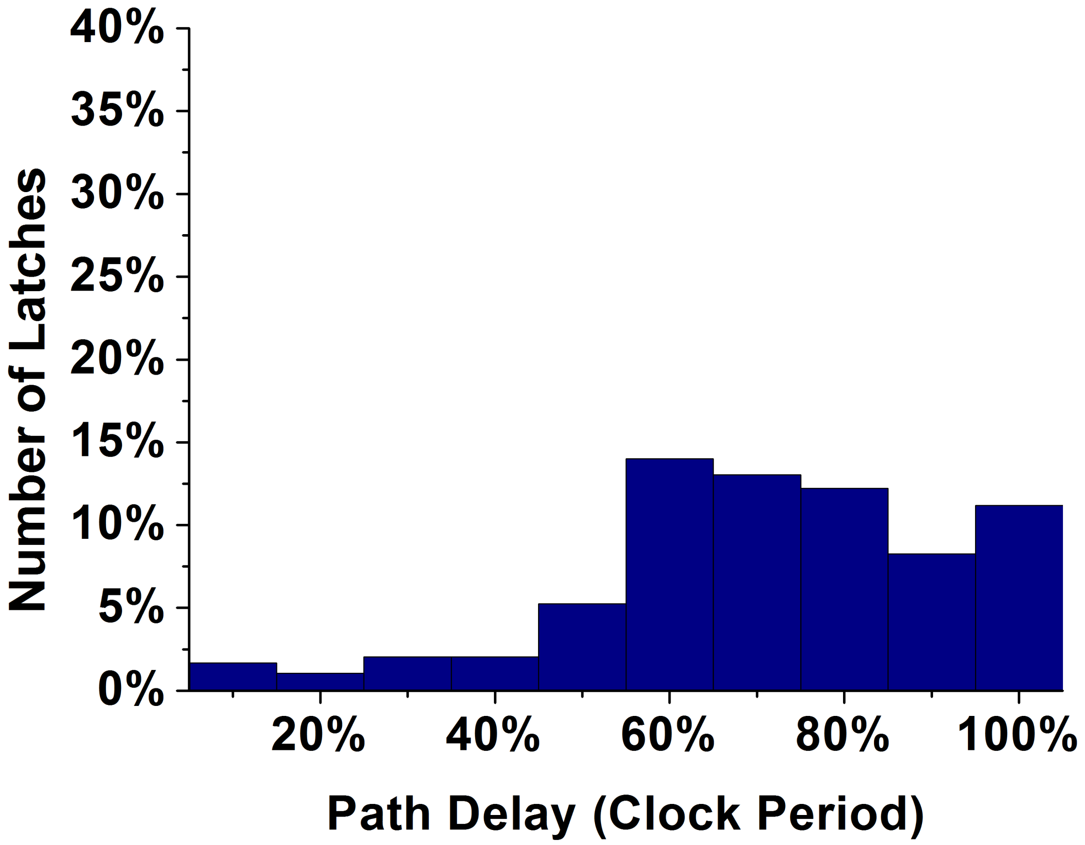


Figure 6.4: Criticality of Positive Latches at Relaxed Timing Constraint

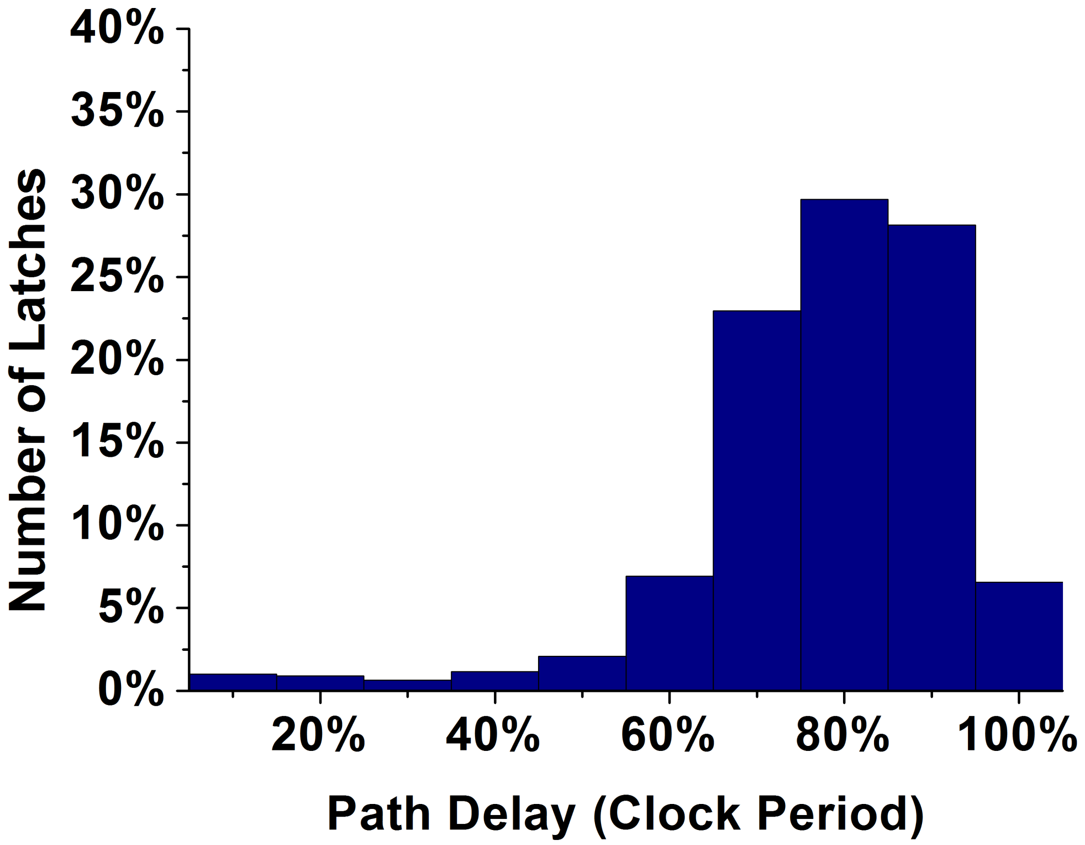


Figure 6.5: Criticality of Positive Latches at Tight Timing Constraint

6.3 Bubbling Algorithm and Logic

In this Bubble Razor implementation, different Bubble propagation circuitry is used than in Chapter 3. This circuitry implements the original definition of the Bubble propagation algorithm, to stall when told and send all other neighbors a Bubble.

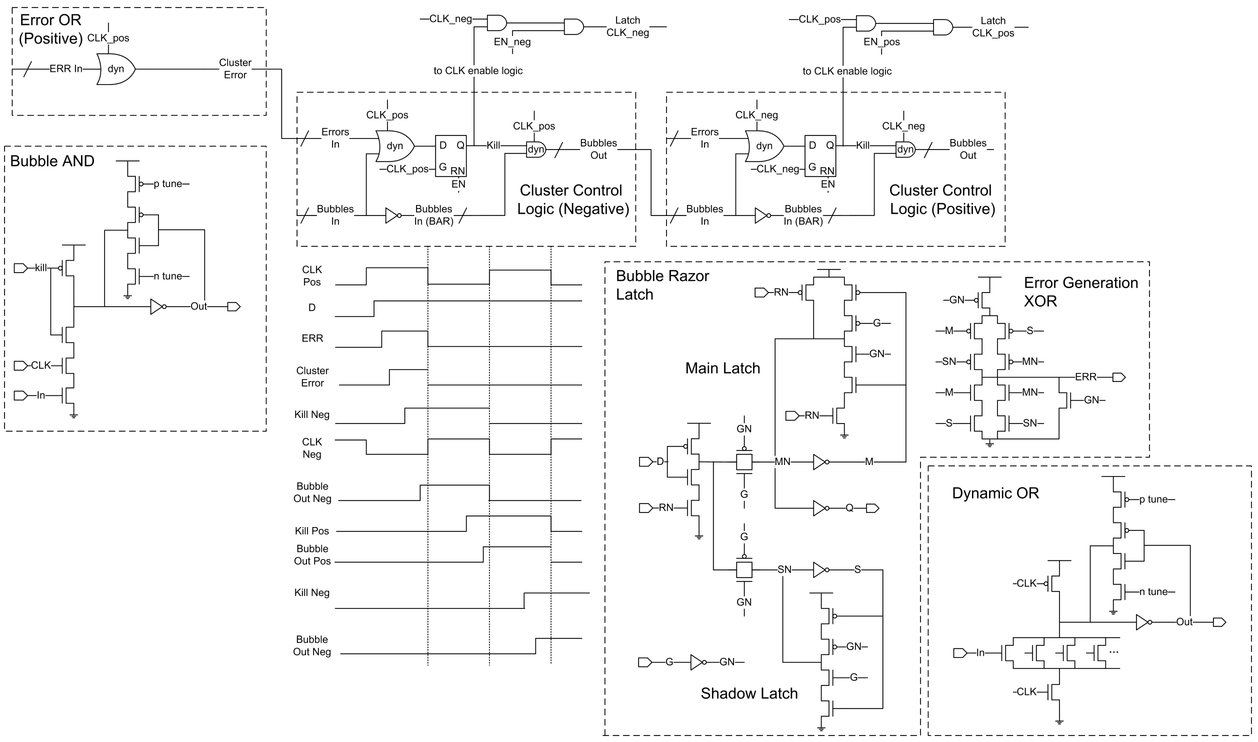


Figure 6.6: Bubble Razor logic for Cortex-R4 Implementation

This logic introduces a new logic gate, the Bubble AND, a dynamic AND which serves the purpose of generating output Bubble signals based on input Bubble signals, and holding their values during the appropriate time. The unique feature of this gate is that it is precharged by the data signal kill instead of a clock, reducing switching energy during error free operation.

6.4 Full Scan and Testability

The Bubble Razor implementation in Chapter 3 did not make use of a scan chain, an extremely common design-for-test practice in industry to allow for manufacturing test. A scan chain allows the state of every flip-flop in a conventional design to be loaded with a predetermined value. The circuit is then run for a set number of cycles and the resultant values are shifted out of the chip and compared to the expected results. This test methodology provides a high degree of confidence that the circuit is free from manufacturing defects.

Since Bubble Razor uses transparent latches instead of flip-flops, they cannot be simply chained together as data would race through two same polarity latches which are connected. A scan chain can be built however by carefully selecting the order with which latches are connected to each other. Only positive or negative latches require their values to be loaded, as opposite polarity latches will be transparent when same polarity latches are holding data. To reduce overhead only the polarity with lowest number of latches should be targeted. To prevent race-through, latch polarity in the scan chain must alternate. As only the polarity with the fewest latches is being targeted, there are enough opposite polarity latches in the design to insert between targeted latches. The sequentials in the additional Bubble Razor control logic (Figure 6.6) can also be added to the scan chain. As with any conventional design which uses clock gating, it must be ensured that all clock gating signals are enabled when in scan mode.

In the Cortex-R4 implementation, the area penalty for adding scan to the latch based design is 54% higher than the area penalty for of adding scan to the flip-flop based design.

6.5 Register File Timing

Figure 2.6 shows the special interface to integrate the Bubble Razor algorithm with typical SRAM timing in order to handle the edge triggered nature of SRAM. Typical processors also contain register files which often follow a common style of timing. Conventional register file timing differs from SRAM timing in that reads are performed combinationally where the address bits are never sampled by a clock.

In prior Bubble Razor work, the register files were considered part of the processor logic and were retimed accordingly. In this implementation, a register file wrapper has been designed to allow the register file to be removed from the part of the processor which is transformed, thus allowing conventional methods of optimizing register files to be used.

Traditional register files in flip-flop based processors are macros which are given write addresses and data, which are written to the register file at the positive edge of the clock, and

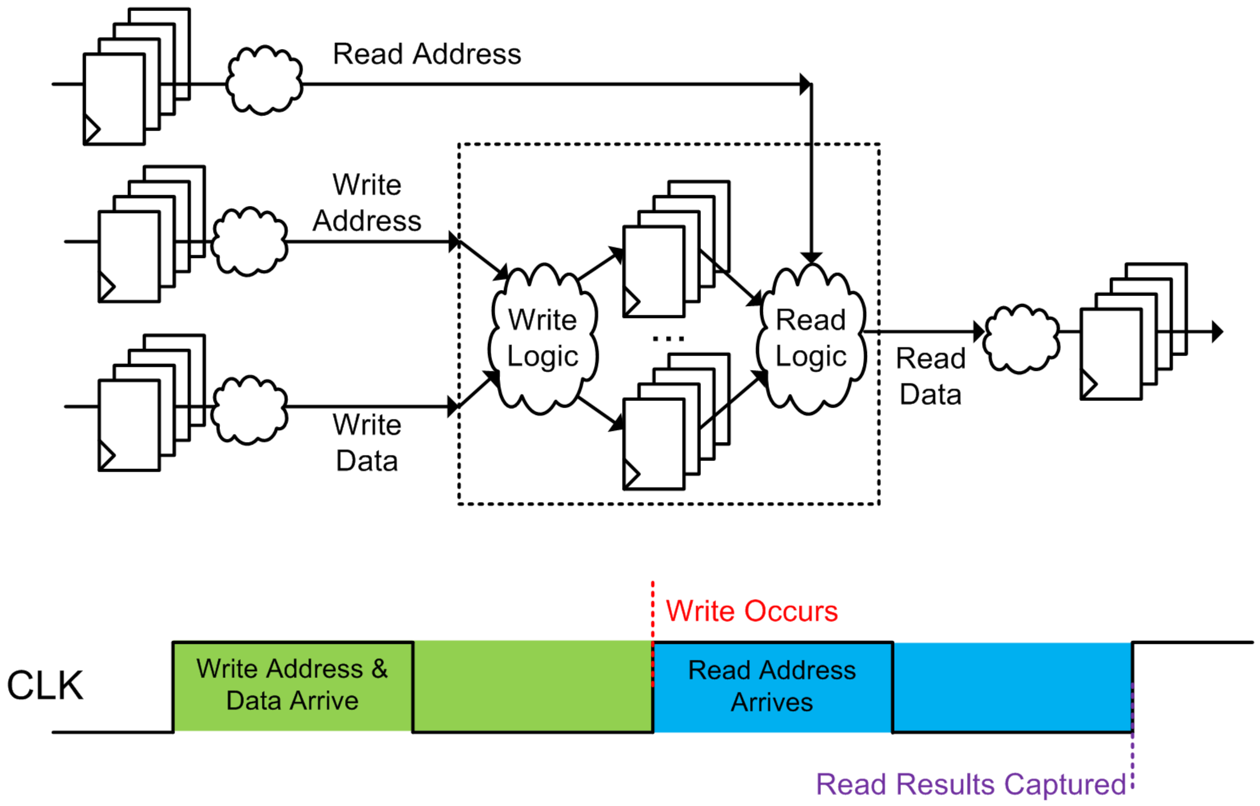


Figure 6.7: The timing and interface to a conventional register file

read addresses which access the register file to provide read data without any clock edges. Figure 6.7 shows a block diagram of this behavior, as well as the associated timing.

To interface the register file with Bubble Razor, the circuitry in Figure 6.8 is proposed. Retiming results in the neighboring latches to the register file which provide and capture addresses and data being negative, thus read data is now captured by the processor at a negative edge and write addresses and data are launched at a negative edge. This may or may not enforce tighter timing constraints on the system, as discussed in Section 5.2 whenever dealing with inputs or outputs to the design.

The first major change is delaying when writes take place. The proposed circuitry has the register file clocked on the negative edge of the clock instead of the positive edge, delaying writes by a half cycle. This ensures that when a write takes place, write data and addresses have stabilized to their correct values and are free of timing errors.

In the event a line of the register file is read immediately after being written, a forwarding path is needed to send the correct data to the output of the register file, as the delay in writing

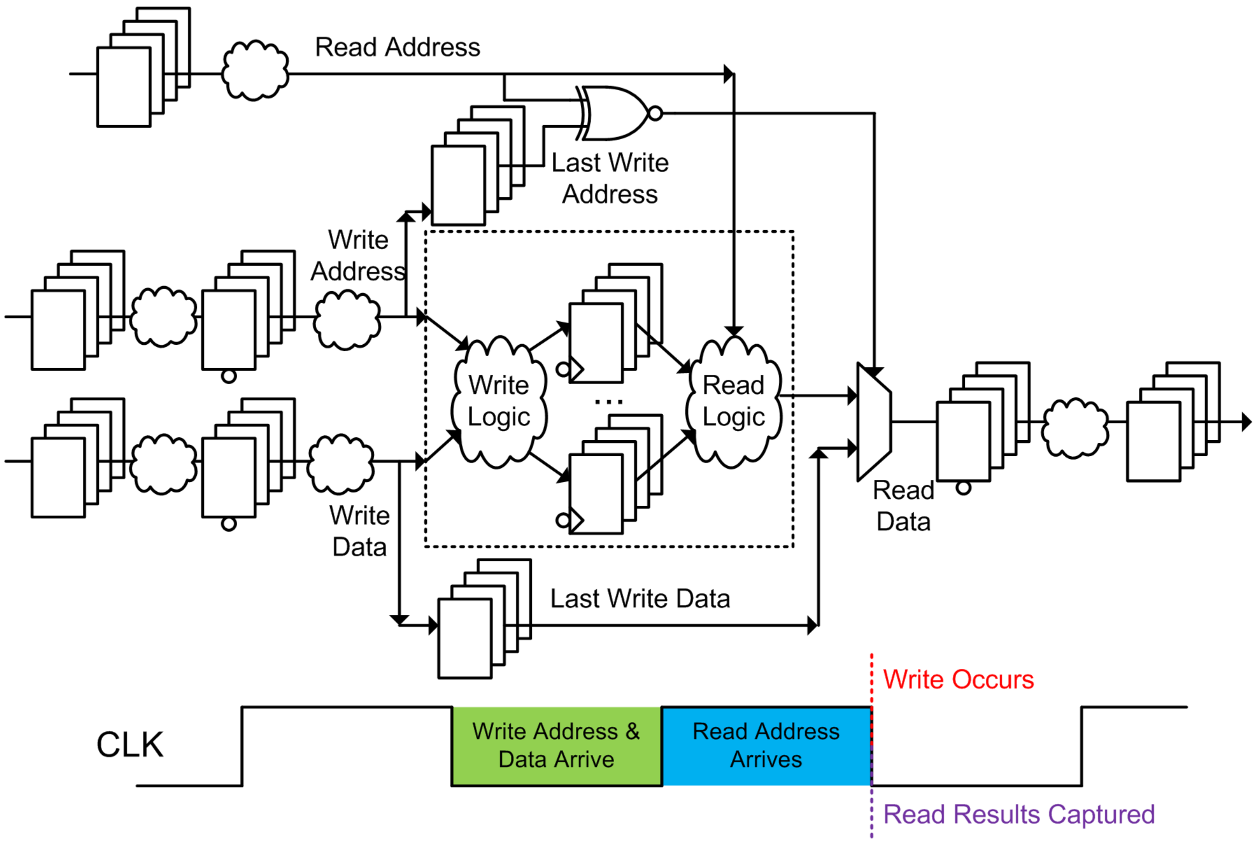


Figure 6.8: The timing, interface, and wrapper to a register file with Bubble Razor

it would have caused the previous value to be sent to the processor. For this reason, write address and data are captured by banks of positive latches parallel to the register file. The captured address is compared to the read address from the next cycle, and if the addresses match, the captured data is forwarded in place of the register file output.

This wrapper allows the original register file macro to be used unchanged, thus it can be optimized through custom design or other methods conventionally used in register file design.

6.6 Conclusion

The Bubble Razor flow was extended to alternatively use the original definition of Bubble propagation, include full scan and testability, and add a new wrapper for register file timing. The clock gating transformation flow discussed in Chapter 5 was implemented and applied to the retiming and transformation of the ARM Cortex-R4. A test system was designed in order to directly compare the performance and power consumption of baseline flip-flop and latch cores against Bubble Razor versions.

CHAPTER 7

Conclusion

7.1 Contributions of This Thesis

This thesis introduces the concept of Razor using two-phase latch based designs. By identifying and utilizing time borrowing as an error correction mechanism, it allows for Razor to be applied without the need to reload data or replay instructions. This allows for Razor to be blindly and automatically applied to existing designs without detailed knowledge of the internal architecture. Additionally, latch based Razor allows for large speculation windows, up to 100% of nominal circuit delay, because it breaks the connection between minimum delay constraints and speculation window.

This thesis also addresses how to transform conventional flip-flop based designs, including those which make use of clock gating, to two-phase latch based timing, allowing Razor to be automatically added to a large set of existing digital designs. Prior to this work, Razor could not be applied blindly or automatically to designs without a designer working with detailed knowledge of the architecture.

Chapter 2 introduces latch based Razor along with the Bubble Razor algorithm. It addresses how to ideally decide on speculation window size, interface with SRAMs, and cluster latches to reduce overhead.

Chapter 3 applies Bubble Razor to the ARM Cortex-M3 processor, the first ever application of a Razor technique to a complete, existing processor design. This proved in silicon for the first time that the Bubble Razor algorithm works, as the processor continued correct operation under the presence of timing errors.

Chapter 4 proposes a new latch based Razor technique which uses voltage boosting to correct for timing errors. While Bubble Razor addressed timing errors by stalling, thus allowing more time for circuits to process data, Voltage Razor solved these errors by speeding up the circuits. This had the benefit of not modifying any cycle timing, allowing for more peripherals to be easily integrated.

Chapter 5 expands the set of designs to which Bubble Razor or Voltage Razor can be applied to by discussing a process to transform designs which make use of clock gating to two-phase latch based timing. The transformation ensures that timing errors are non-critical during the time borrowing window, thus time borrowing can continue to be used as the error correction mechanism by various Razor techniques.

Chapter 6 details the application of Bubble Razor to the ARM Cortex-R4, a larger and more complicated processor than the M3 which makes use of clock gating. Care is taken to minimize various overheads and several new techniques are applied.

Appendix A discusses detailed timing constraints behind latch based Razor and compares them to conventional systems and existing Razor techniques.

7.2 Directions for Future Work

In this author's view, the largest barriers to widespread Razor adoption are incurred overheads from adding Razor and the complexities involved in interfacing with and verifying a Razor system. Although Bubble Razor and Voltage Razor do much to reduce the complexity of Razor addition, allowing it to occur automatically, there are still some interface issues left to the designer.

7.2.1 Reducing the Subset of Latches Which Require Error Checking

As documented in Chapters 3 and 4, the largest overhead resulting from Razor addition comes from the error detection needed on potentially critical paths. This issue applies to all

Razor style designs, although its effect is largest when a large speculation window is used. Due to the large speculation windows supported by Bubble Razor and Voltage Razor, this overhead can be extremely large.

Figure 3.2 shows the critical path distribution for the Cortex-M3 before and after retiming, with the resulting speculation window and area overhead tradeoff shown in Figure 3.3. For a desired speculation window, a certain set of latches are potentially critical and must have timing error detection placed on them. This error detection is costly in terms of area and power consumption, and it is desirable to minimize the number of latches which require error checking.

In a conventional system's synthesis, a timing wall tends to occur near the timing constraint. Paths are optimized until they meet timing, and then the synthesis or place and route tool stops optimization and moves on to a different failing path. To further optimize these paths would come at an area and power cost, yet offer no benefit, as the circuit already meets timing. Area savings can also occur if the synthesis tool downsizes gates in non-critical paths until they become critical. From a conventional standpoint, it is ideal for all paths in the system to be critical or near critical. However, if any style of Razor is added to a system that shows these timing characteristics, an extremely high number of latches or flip-flops would require error detection, as any one of them could be the first to fail.

When applying Razor to a design, the common approach is to add error detection to potentially critical latches or flip-flops. Though some paths may not be sped up through synthesis, it is possible to speed up some other paths to the point where they are non-critical and thus do not require error checking. It is possible for either error checking or a path speedup to come at a lower area and power cost and thus be more desirable.

The constraints provided to retiming can expand this search space. Retiming may allow some critical paths to become noncritical by moving delay slack from adjacent stages, however the process may also increase or decrease the total number of latches in the design.

This multi-variable tradeoff has potential for a large reduction in the overall area and power cost for transforming a design to use Razor. Because modern synthesis tools tend to create many critical paths, they are essentially timing the design in the worst possible way with regards to Razor addition. For this reason, this author believes there are large gains which may be found investigating this problem.

7.2.2 Verification

In many companies, there are more engineers devoted to verifying the correctness of designs than there are designers building them. Verification is a key component of any design process, however is often diminished in research settings. The addition of Razor, especially Bubble Razor or Voltage Razor, vastly expands the workload needed to verify the correctness of a design. Once timing errors are allowed to occur, the verification flow must not only consider whether the target design is correct but also whether the error correction process is flawless. When switching to two-phase latch based timing, one must verify whether the design behaves identically after being transformed.

There is much work to be done in the process of aiding the verification flow for Razor designs. Unless companies can believe with extreme confidence that a Razor based design is correct, they will likely not implement Razor even if there are large performance gains or energy savings.

7.2.3 Bus Protocols

Upon detection of an error, the Bubble Razor algorithm produces a bubble which causes clusters of latches to stall. When these stalls reach an interface to a non-transformed circuit, the designer must modify the RTL of the interface in order to handle this unexpected stall. For asynchronous interfaces, such as the JTAG debug access port in the Cortex-M3, nothing needs to be done as the protocol does not rely on cycle-to-cycle accurate timing. Other interfaces, such as buses to external memory, rely on cycle specific timing.

Many peripherals operate at much lower speeds than the processor clock, however they are synchronized to the processor clock. To address this, bus protocols support stalling or wait states. Essentially, every bus transaction comes with a valid signal signifying if it is to be exercised that cycle.

It may be possible to combine stalls resulting from Bubble Razor with popular existing bus protocol's wait states. This would allow for numerous peripherals to be easily added to the processor without any designer effort to deal with the interface.

APPENDIX

APPENDIX A

Timing Constraints of Razor Style Systems

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

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

Table A.1: Terminology

A.1 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 A.2 [32].

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

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	$T_c - t_{pdql} - max(0, t_{setupl} - t_{pw})$	$t_{pw} + t_{holdl} - t_{ccql}$	$t_{pw} - t_{setupl}$

Table A.2: Timing Constraints for Conventional Systems

tradeoff between time borrowing and contamination delay.

A.2 DSTB Timing Constraints

This section explains the timing constraints for DSTB [23]. TDTB has similar timing, with the setup time of the flip-flop analogous to the setup time of the TD. Figure A.1 shows the timing diagram for a DSTB system.

The primary datapath resembles an ordinary pulse-latched sequencing system [33]. 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 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.

A.2.1 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:

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

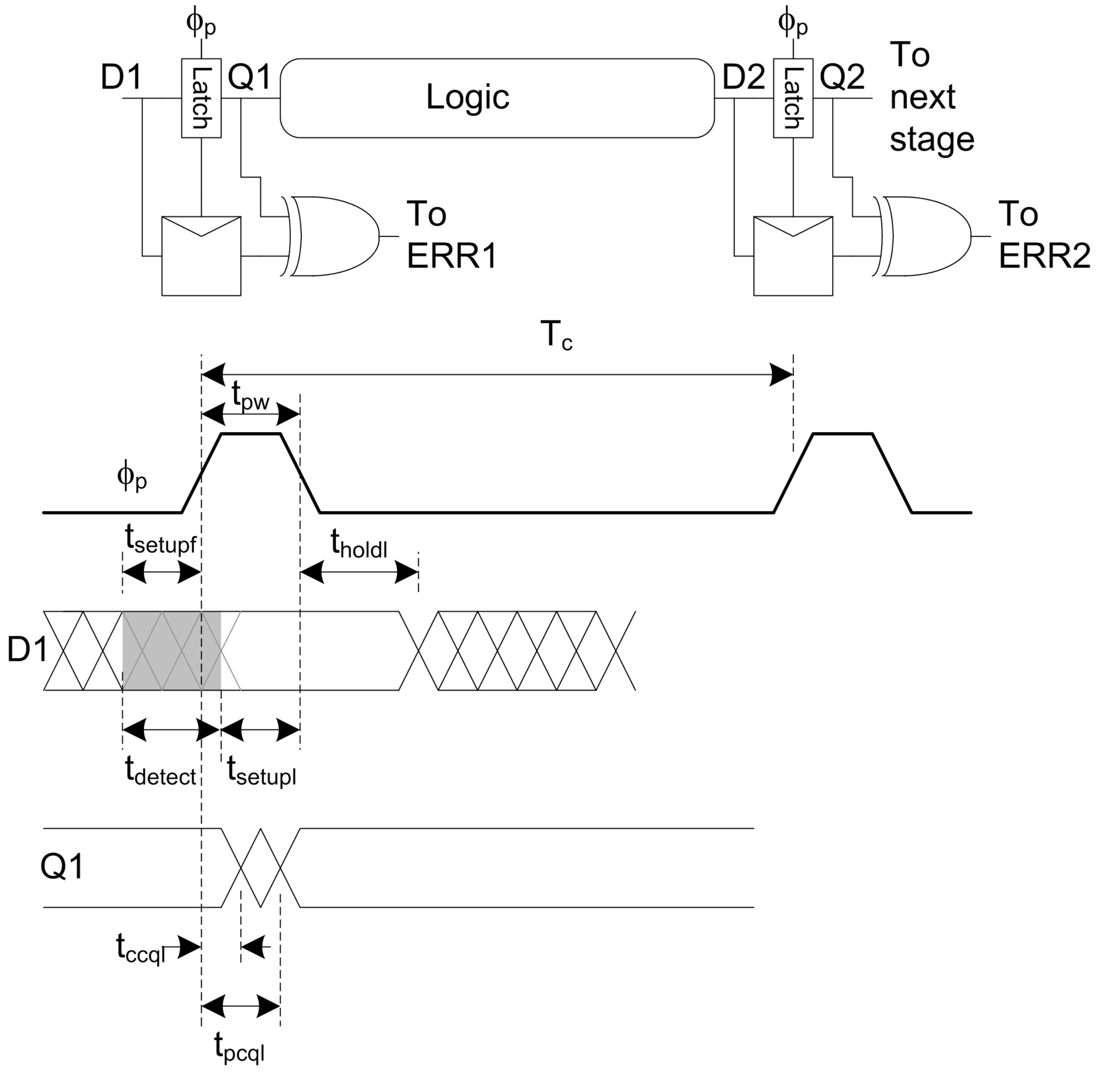


Figure A.1: Timing Diagram for DSTB System

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.

A.2.2 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

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

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

A.2.3 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

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

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.

A.2.4 Clock Skew

Clock skew reduces the maximum propagation delay and increases the maximum contamination delay by t_{skew} in EQs A.2 and A.3 [33].

A.2.5 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

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

The detection window reduces by this delay:

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

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

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

A.3 Razor II Timing Constraints

This section explains the timing constraints for Razor II [22]. Figure A.2 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.

A.3.1 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

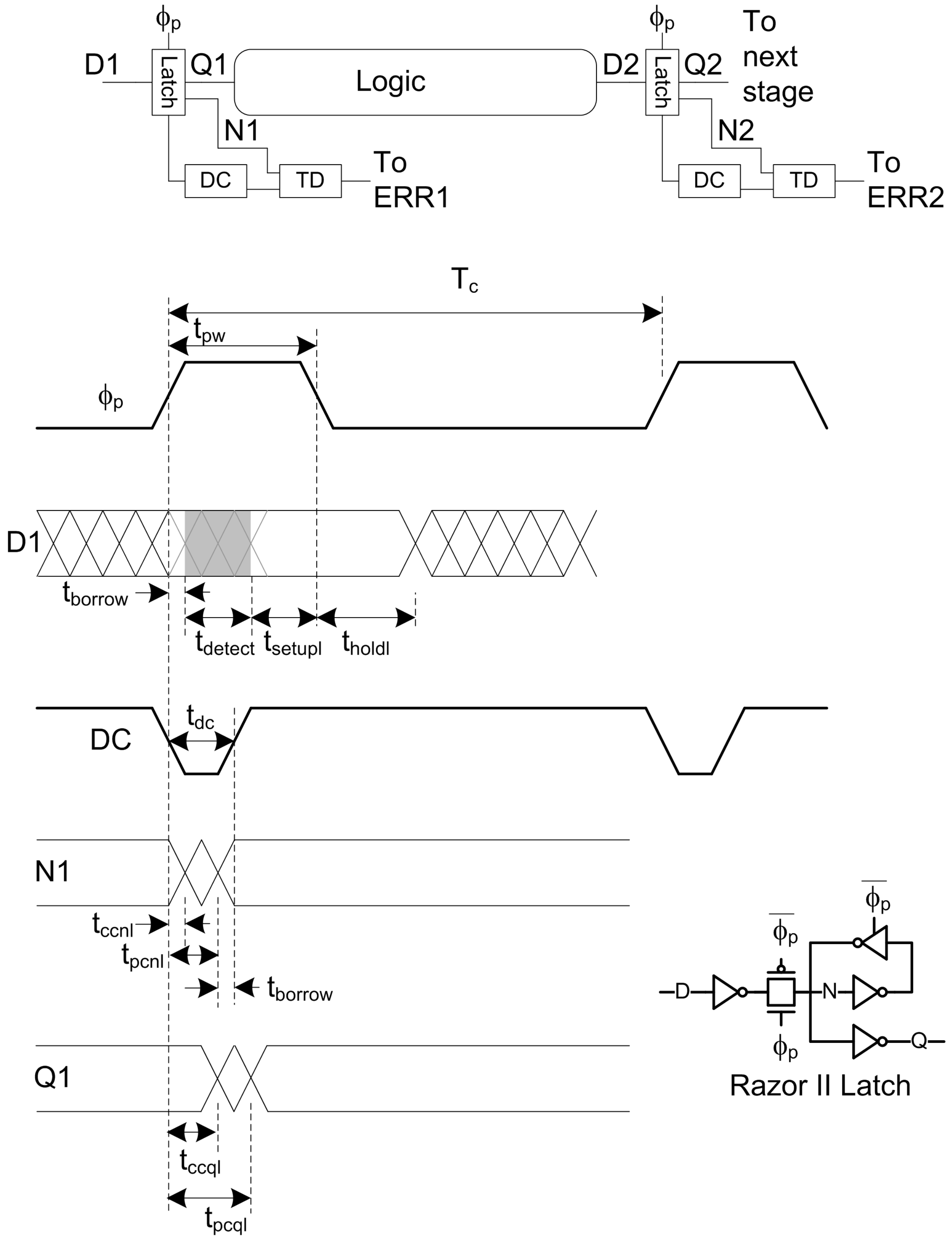


Figure A.2: Timing Diagram for Razor II System

borrowing allowed by the system.

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

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}:

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

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

A.3.2 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

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

A.3.3 Contamination Delay

The contamination delay for Razor II is identical to DSTB:

$$t_{cd} \geq t_{pw} + t_{holdl} - t_{ccql} \tag{A.10}$$

A.3.4 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.

A.3.5 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 [32]. 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.

A.4 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 [33], 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.

A.4.1 Two-Phase Timing Diagram

Figure A.3 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 [33]. 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

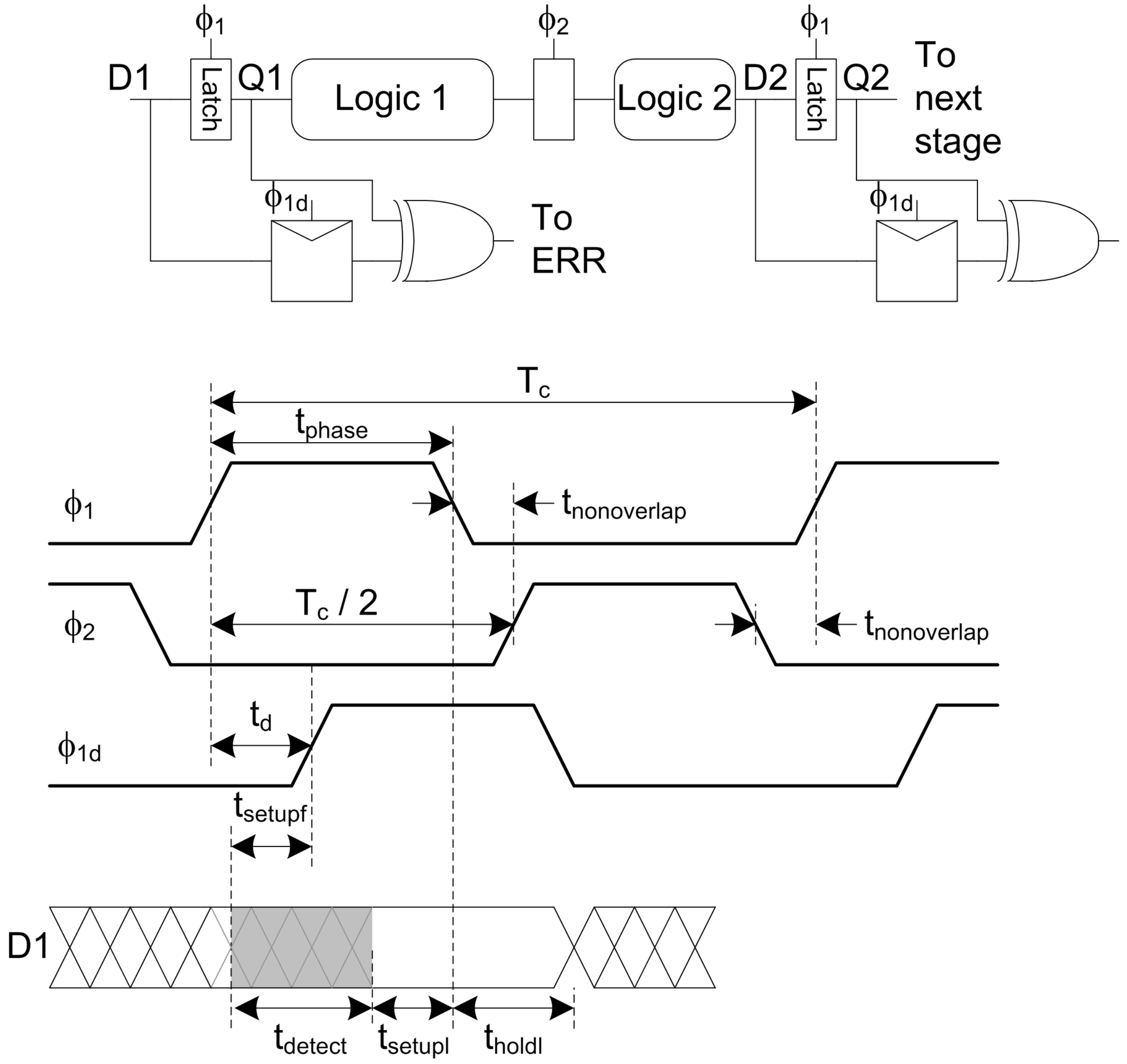


Figure A.3: Timing Diagram for Bubble Razor System

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.

A.4.2 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:

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

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

A.4.3 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

$$t_{pd} \leq T_C - t_{pdql1} - t_{pdql2} \tag{A.12}$$

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.

A.4.4 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:

$$\begin{aligned} t_{cd1} &\geq t_{holdl2} - t_{ccql1} - t_{nonoverlap} \\ t_{cd2} &\geq t_{holdl1} - t_{ccql2} - t_{nonoverlap} \end{aligned} \tag{A.13}$$

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.

A.4.5 Time Borrowing

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

$$t_{borrow} = t_d - t_{setupf} \tag{A.14}$$

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

$$t_{detect} = t_{phase} - t_{setupl} - t_{borrow} \tag{A.15}$$

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.

$$t_{borrow-midcycle} = t_{phase} - t_{setupl} \tag{A.16}$$

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 EQ A.15 from A.17 gives a more conservative borrowing limit that allows the full use of the detection window.

$$t_{borrow-midcycle} = t_d - t_{setupf} \tag{A.17}$$

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

A.4.6 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.

A.4.7 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.

BIBLIOGRAPHY

[1] Shidhartha Das, David Roberts, David Blaauw, David Bull, and Trevor Mudge. Architectural techniques for adaptive computing. In Alice Wang and Samuel Naffziger, editors, *Adaptive Techniques for Dynamic Processor Optimization*, Integrated Circuits and Systems, pages 175–205. Springer US, 2008.

[2] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7 – 18, dec. 2003.

[3] Executive Summary. Platform 2015 : Intel processor and platform. *Systems Technology*, 21(4):419–424, 2006.

[4] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4.5):433 –449, july 2006.

[5] R. McGowen, C.A. Poirier, C. Bostak, J. Ignowski, M. Millican, W.H. Parks, and S. Naffziger. Power and temperature control on a 90-nm itanium family processor. *Solid-State Circuits, IEEE Journal of*, 41(1):229 – 237, jan. 2006.

[6] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie. Comparison of split-versus connected-core supplies in the power6 microprocessor. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 298 –604, feb. 2007.

[7] 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. *Solid-State Circuits, IEEE Journal of*, 46(1):194 –208, jan. 2011.

[8] T. Fischer, J. Desai, B. Doyle, S. Naffziger, and B. Patella. A 90-nm variable frequency clock system for a power-managed itanium architecture processor. *Solid-State Circuits, IEEE Journal of*, 41(1):218 – 228, jan. 2006.

[9] J. Tschanz, Nam Sung Kim, S. Dighe, J. Howard, G. Ruhl, S. Vanga, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 292 –604, feb. 2007.

[10] J.W. Tschanz, S. Narendra, R. Nair, and V. De. Effectiveness of adaptive supply voltage and body bias for reducing impact of parameter variations in low power and high performance microprocessors. *Solid-State Circuits, IEEE Journal of*, 38(5):826 – 829, may 2003.

[11] J.W. Tschanz, J.T. Kao, S.G. Narendra, R. Nair, D.A. Antoniadis, A.P. Chandrakasan, and V. De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *Solid-State Circuits, IEEE Journal of*, 37(11):1396 – 1402, nov 2002.

[12] James Tschanz, Keith Bowman, Steve Walstra, Marty Agostinelli, Tanay Karnik, and Vivek De. Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In *VLSI Circuits, 2009 Symposium on*, pages 112 –113, june 2009.

[13] Xiaoyao Liang, B.C. Lee, Gu-Yeon Wei, and D. Brooks. Design and test strategies for microarchitectural post-fabrication tuning. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, pages 84 –90, oct. 2009.

[14] T. Nakura, K. Nose, and M. Mizuno. Fine-grain redundant logic using defect-prediction flip-flops. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 402 –611, feb. 2007.

[15] 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 65nm high-performance microprocessor. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 398 –399, feb. 2007.

[16] K. Hirairi, Y. Okuma, H. Fuketa, T. Yasufuku, M. Takamiya, M. Nomura, H. Shinohara, and T. Sakurai. 13% power reduction in 16b integer unit in 40nm cmos by adaptive power supply voltage control with parity-based error prediction and detection (pepd) and fully integrated digital ldo. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 486 –488, feb. 2012.

[17] T.D. Burd, T.A. Pering, A.J. Stratakos, and R.W. Brodersen. A dynamic voltage scaled microprocessor system. *Solid-State Circuits, IEEE Journal of*, 35(11):1571 –1580, nov 2000.

[18] 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. *Solid-State Circuits, IEEE Journal of*, 40(1):28 – 35, jan. 2005.

[19] 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. *Solid-State Circuits, IEEE Journal of*, 37(11):1441 – 1447, nov 2002.

[20] S. Dhar, D. Maksirnovi, and B. Kranzen. Closed-loop adaptive voltage scaling controller for standard-cell asics. In *Low Power Electronics and Design, 2002. ISLPED '02. Proceedings of the 2002 International Symposium on*, pages 103 – 107, 2002.

[21] 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. *Solid-State Circuits, IEEE Journal of*, 46(1):18 –31, jan. 2011.

[22] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D.M. Bull, and D.T. Blaauw. Razorii: In situ error detection and correction for pvt and ser tolerance. *Solid-State Circuits, IEEE Journal of*, 44(1):32 –48, jan. 2009.

[23] K.A. Bowman, J.W. Tschanz, Nam Sung 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. *Solid-State Circuits, IEEE Journal of*, 44(1):49 –63, jan. 2009.

[24] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning dvs processor using delay-error detection and correction. *Solid-State Circuits, IEEE Journal of*, 41(4):792 – 804, april 2006.

[25] M. Fojtik, D. Fick, Yejoong Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester. Bubble razor: An architecture-independent approach to timing-error detection and correction. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 488 –490, feb. 2012.

[26] P. Franco and E.J. McCluskey. Delay testing of digital circuits by output waveform analysis. In *Test Conference, 1991, Proceedings., International*, page 798, oct 1991.

[27] M. Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *VLSI Test Symposium, 1999. Proceedings. 17th IEEE*, pages 86 –94, 1999.

[28] Robert Pawlowski, Evgeni Krimer, Joseph Crop, Jacob Postman, Nariman Moezzi-Madani, Mattan Erez, and Patrick Chiang. A 530mv 10-lane simd processor with variation resiliency in 45nm soi. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 492 –494, feb. 2012.

[29] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: application in vlsi domain. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 526–529, New York, NY, USA, 1997. ACM.

[30] ARM Cortex-M3. `http://www.arm.com/products/CPUs/ARM_Cortex-M3.html`.

[31] ARM Cortex-R4. `http://www.arm.com/products/processors/cortex-r/cortex-r4.php`.

[32] David Harris. *Skew-tolerant circuit design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[33] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective.* Addison-Wesley Publishing Company, USA, 4th edition, 2010.