A High Throughput Multiplier Design Exploiting Input Based Statistical Distribution in Completion Delays

by

Ravi Tej Uppu

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
Aug 3, 2013

Keywords: Wallace Multiplier, Completion Sensing, Delay Distribution, Clock Period Scaling, Throughput, Hold and Release

Approved by

Adit D.Singh, Chair, James B Davis Professor, Electrical and Computer Engineering
Vishwani Agrawal, James J Danaher Professor, Electrical and Computer Engineering
Fa Foster Dai, Professor and Asso Director, Electrical and Computer Engineering

Abstract

The primary goal of this work is to ensure that optimum performance is achieved for a Multiplier Design, while reducing as much static power dissipation as possible or atleast equal to their slower counterparts. This design tries to exploit the input based Statistical Distribution of Completion Delays of a circuit in optimizing the performance. Design methodologies such as Razor [3,4,5] minimize power dissipation by slowing down circuits so as to eliminate timing slacks to the point where occasional timing errors are observed. The main challenge is the design of efficient mechanisms to detect and recover from these infrequent errors.

We present a novel design for widely used Wallace multiplication using 4:2 compressors, where because of the highly skewed input based statistical distribution in completion delays, the potential for power and performance gains is significantly higher; clock periods can be potentially reduced by a factor of 3 or more, with very rare timing violations for random input distributions. For this we present a novel low cost error recovery approach that latches and holds logic values at key internal circuit nodes during every clock cycle beyond the next clock edge. This allows generation of the correct outputs for that clock period one clock cycle later in case of a timing error. Meanwhile, very fast error evaluation, exploiting a unique characteristic of carry ripple addition, allows this hold to be quickly released if no error is detected, ensuring no impact on the circuit timing in error free operation. While an additional area overhead of 10% was observed after implementing the design in a 32x32 Wallace Multiplier a 2.5x improvement in the average performance was achieved. Spice simulation results with varied clock period for 10000 vectors shows an optimum average performance improvement can be achieved at a reduced clock period of 3.75ns against the actual clock period of 9.5ns, the vectors which can trigger the critical path were obtained

from [8]. Also, this design when deployed in a logic circuit would prove to be a Variation Tolerant Design.

Acknowledgments

My advisor and committee were the people most directly involved with the completion of my thesis. I would like to express my appreciation and sincere thanks to my advisor Dr. Adit Singh, who patiently shaped this work right from the beginning. I benefited greatly from his ability to approach problems from many different directions. His advise and attitude towards life would remain a guiding light for me throughout my career. I also wish to thank my advisory committee members, Dr. Vishwani Agrawal and Dr. Fa Foster Dai for their guidance and advice on this work.

My work could not have been completed without the excellent course work offered in VLSI design and Testing and other related courses at Auburn University, for which I am grateful to each and every professor.

Graduate study at Auburn University has been a learning experience. I thank my brother Ravi Kanth for his constant support and colleagues Suraj and Praveen for their valuable inputs and patients. Without them my graduate experience would not have been this enriching.

I am indebted to my parents, brother, sister and all friends for their love and support. I also thank everyone who directly or indirectly helped and inspired me during the course of my graduate study.

Finally, I would like to thank AMD for giving me an opportunity to implement the design in TSMC 28nm technology and letting me use the relative numbers, besides giving a start off to my career.

Table of Contents

Abstract ii

Acknowledgments iv

List of Figures viii

List of Tables x

1 Introduction 1

1.1 Background 2

1.2 Motivation 4

1.3 Problem Statement 7

1.4 Contribution of This Thesis 7

1.5 Organization of Thesis 8

2 Literature Survey 9

2.1 Requirement for BTWC designs 9

2.1.1 Previous BTWC designs 11

2.2 Razor 12

2.2.1 Razor I Overview 13

2.2.2 RazorII 14

2.2.3 Razor Limitations 16

2.3 CRISTA 19

2.4 CSCD for High Speed and Area Efficient Arithmetic 20

2.5 Need for Fast and Resilient Error Detection Circuit 22

3 Brief Overview on Adders 23

3.1 Basic Adders 23

3.1.1 Half Adder 23

3.1.2 Full Adder 24
3.2 Ripple Carry Adder (RCA) 25
3.2.1 Performance Analysis of a 4-bit RCA 26
3.2.2 Best Case Performance 26
3.2.3 Worst Case Performance 27
3.2.4 Average Case Performance 28
3.3 Carry Look Ahead Adder 29
3.3.1 Performance Analysis of 4-bit CLA 29
3.4 Carry Select Adder 31
4 Overview on Multipliers 33
4.1 The Multiplier 33
4.2 Partial Product Generation 34
4.3 Partial Product Accumulation 35
4.3.1 Array Multiplier 35
4.3.2 Carry-Save Multiplier 37
4.3.3 Wallace Multiplier 38
4.4 Final Addition 42
5 Proposed BTWC Design 44
5.1 The Internal signal Hold until Completion Confirmed Scheme 44
5.2 Timing Error Detection 46
5.3 Fast Comparator to monitor output completion 48
5.3.1 Equality Comparator 50
5.3.2 A Pass Logic Single Stage Comparator 51
6 Implementation of 32x32 Wallace Multiplier with the proposed BTWC Design . 53
7 Simulation Results and Conclusion 56
7.1 Simulation Results using TSMC 180nm technology 56
7.2 Simulation Results using TSMC 28nm technology 58

7.3 Conclusion 59
Bibliography 60

List of Figures

1.1 Delay Distribution of 32 bit RCA 3

2.1 Razor flop-flop 13

2.2 Pipeline augmented with Razor latches and control lines 15

2.3 Current Sensor Circuitry 21

3.1 Half Adder 23

3.2 Full Adder Block 24

3.3 Full Adder using two Half Adders 25

3.4 Full Adder with fast carry out 25

3.5 4-bit Ripple Carry Adder 26

3.6 4-bit Ripple Carry Adder 26

3.7 4-bit adder best case performance 27

3.8 4-bit adder worst case performance 27

3.9 Illustrating the probability of carry rippling through a 5 stage RCA 28

3.10 Partial Full Adder 29

3.11 4-bit Carry Look Ahead Adder 31

3.12 Carry Select Adder 32

4.1 Binary Multiplication - an example 34

4.2 4x4 Array Multiplier 35

4.3 Delay Distribution of an Array Multiplier for 50,000 vectors 36

4.4 A 4x4 carry-save multiplier 37

4.5 Transforming a Partial-Product tree (a) into a Wallace tree (b,c,d), using an iterative covering process. 39

4.6 Wallace Multiplier 40

4.7 Illustration of Wallace Multiplier 40

4.8 4:2 Compressor 41

4.9 Illustration of Wallace Multiplier with 4:2 compressors 41

4.10 Delay Distribution of a Ripple Carry Adder 42

4.11 Delay Distribution of 32-bit Wallace Multiplier with final RCA distribution for 100000 vectors 43

5.1 Pipelined Schemtic for Wallace Multiplier Operation 45

5.2 Timing Model Waveforms for Hold Until Completion 46

5.3 Timing Error Detection Circuitry 47

5.4 Implementation of the above verilog code by Mentor Graphics Synthesis tool . . 49

5.5 4-bit Equality comparator 51

5.6 4-bit Equality comparator 51

6.1 Illustration of Wallace Multiplier with the Proposed Design 53

List of Tables

7.1 Spice Simulation Results with Reduced Clock Periods 57

7.2 Relative numbers for 28nm Multiplier Design (Performance and Area Optimized) 58

Chapter 1

Introduction

Many factors introduce variation into the behavior of CMOS-based processor designs. Non-idealities such as voltage fluctuations in the power supply network, temperature fluctuations in the operating environment, manufacturing variations in parameters such as gate length and doping concentration, and cross-coupling noise all affect the timing behavior of a processor, making it statistical in nature rather than deterministic.

In traditional processor designs, great pains are taken to ensure that a processor always produces correct results, even when subjected to a worst-case combination of non-idealities.This means that conservative guardbands are incorporated into design parameters to ensure correct behavior in all possible scenarios. Design for such a conservative operating point incurs a considerable overhead in terms of power spent to ensure correctness. Making matters worse, variation in CMOS-based circuits is expected to increase in comming technology generations [International Technology Roadmap for Semiconductors 2008, http://public.itrs.net.],resulting in the need for even more conservative designs. Consequently, the already expensive cost of pristine computation will continue to increase in the future.

Although processors are traditionally designed to tolerate worst-case conditions, it is unlikely that all nonidealities will take effect at once, pushing a processor to the brink of erroneous behavior. Thus, there exists a considerable potential to increase the power efficiency or performance of processors by relaxing traditional, conservative requirements for correctness in the worst-case and instead designing processors for the average-case. Such better-than-worstcase designs work normally in the average case and have recovery mechanisms to ensure correct operation when errors occur.

1.1 Background

Digital circuits are classified into synchronous and asynchronous circuits. Synchronous circuits (and systems) include clocked elements like flip flops, latches and registers. The input of these elements should be stable before the next active edge of the clock. One major problem with these systems is the distribution of clock signals which must be in phase without significant skew every place in the chip. This makes designing a complex synchronous circuit operating at high frequency difficult due to timing skew and delays. Besides, a design should also accommodate some timing window for PVT induced variation during which the system is completely inactive, which indeed contributes to the already significant leakage power.

Having mentioned the above constraints, it is imperative to design an innovative technique which can not only exploit the timing window to accommodate PVT induced variation but also the slack available in completion delays. While the above mentioned solutions seems possible if the design has a mechanism which can not only sense the failing paths but also effectively recover them by providing an additional clock cycle.

Authors in [ref] deploy voltage scaling and frequency scaling techniques to reduce the power dissipation or improve the performance by exploiting the non-critical paths completion delays timing slack. The Razor approach, proposed in [3], aims at reducing the power consumption by minimizing the timing margin to zero and beyond (used to accommodate PVT induced timing margin), by building in a system capability to detect occasional errors due to slow signal paths and recover from them. The timing margins are removed by reducing the supply voltage to slow down the circuit to a point where a small, acceptable number of errors are observed. As long as the power saving from the reduced voltage operation exceeds the extra power needed, on average, by the occasional error detection and recovery cycle, such a scheme can provide a net power saving. The challenge clearly is in designing an efficient low cost error detection and recovery capability to support this approach. The original Razor design [3] has changed and evolved [4,5,7] significantly over time in an attempt to achieve practicality. Even so the potential power savings from eliminating timing margins

appear limited. Besides CRISTA [6], tries to exploit the input based statistical distribution in completion delays and use an algorithm to synthesize a circuit such that a critical path is triggered occasionally i.e. aiming to obtain a skewed distribution in completion delays, this technique uses shannon's expansion theorem to synthesize a circuit and upsize the gates in a critical path to make it furthur critical.

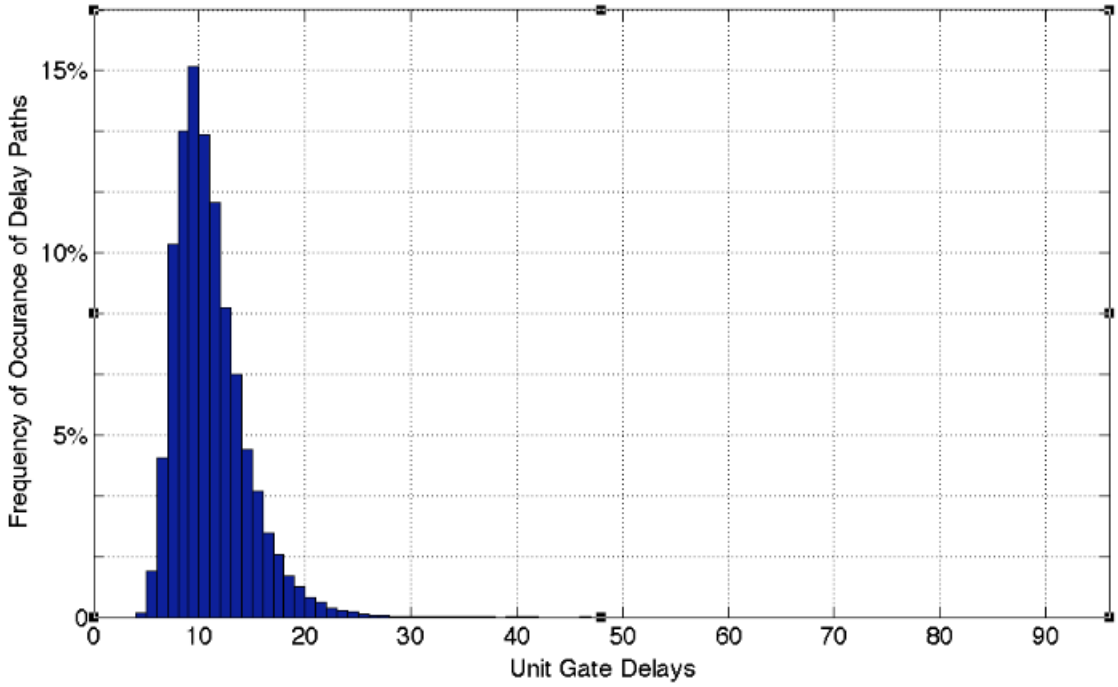

Figure 1.1: Delay Distribution of 32 bit RCA

Instead of applying this approach of minimizing circuit timing slack with recovery from occasional errors to general purpose logic as attempted by the Razor team and others, we have implemented it in specific widely used computations such as addition and multiplication as in [1] where, because of the inherent characteristics in the applications, the potential for power savings can be much greater. Depending on the hardware implementation, these computations can display a wide range of completion delays depending on the applied inputs. For example a simple low cost (in hardware and power) 32-bit ripple carry adder (RCA) can generate a result with no carry propagation delays for some input cases, while requiring 32 stages of carry propagation in the worst case. Importantly, for random inputs, this delay distribution is highly skewed, which can allow a significant speedup in the operating clock

period while still ensuring only a small error and recovery rate. This is seen in Figure 1.1, which simulates addition completion delays for a 32bit RCA designed at the gate level and, for simplicity, assumes fixed unit delay for every gate. Observe that the median addition delay is only 10 units when compared to the worst-case delay of 64 units. Note this does not mean that only 3-4 full adder stages generate a carry for the median case because strings of carries are generated and propagate in parallel at the same time. From the point of generation, a carry needs to see 01 or 10 inputs in the subsequent full adder stage for propagation, resulting in only a 50% chance that it will propagate through the next stage. The probability of a carry propagating n stages is 2^n , which dies out quickly with increasing n. Consequently, it if observed in Figure 1 that compared to a worst case delay of 64 units, less than 0.1% of random inputs result in an addition delay of greater than 25 units. This suggests that the addition can potentially be run at 2.5X the worst case speed, with only one in a thousand operations, on average, requiring correction and recovery. Of course the overall system, utilizing such an adder must be designed to support and work with this occasional recovery operation. Fortunately, architectural level handing of exceptions through pipeline stalls and out of order instruction issue are now well understood and commonplace in processors to handle other forms of speculative execution; our design can be considered just another speculative implementation. Note the greatly reduced average clock period results in significant energy saving per computation because of the saving in the static power wasted during the mostly inactive part of the longer clock period in a traditional design. Moreover, this performance gain can be traded off for lower power consumption by reducing the supply voltage to slow down the circuitry instead of speeding up the clock as in the Razor[3] approach.

1.2 Motivation

Current trends in the CMOS VLSI design methodologies show steady scaling of feature sizes to meet the speed and performance requirements for complex applications in the areas

of communication, aerospace, defence and consumer electronics. Extensive scaling of CMOS process has given rise to many problems such as leakage current, short circuit current, large process variations, etc [2]. It is proving to be difficult to achieve higher performance with transistor channel length scaled to few nanometres. These problems have to be addressed while designing circuits for low power and high speed applications. Also, with CMOS technology approaching its limit, new approaches for processes and design architectures are needed. Apart from scaling, there has been much focus on very low power consumption and area efficient design methodologies in developing consumer applications.

As performance and battery life requirements are becoming more crucial in small and lightweight systems like laptops and hand-held devices, power consumption is becoming a critical problem. For example, processors and microcontrollers for these small portable systems should provide higher system performance while keeping power consumption within specified limits. The performance of these chips is mainly dependent on the Arithmetic Logic Units (ALUs) used in them to perform complex mathematical operations such as addition, subtraction, division and multiplication. To save silicon area and power, some chips do not include dedicated arithmetic circuits for multiplication and other complex computations which are considered to be very area intensive and power hungry; multiplication is performed through repeated addition. In such scenarios, there is a critical need for fast and power efficient adder circuits which are almost constantly active during computation, and can have a major impact on overall system performance. Current Sensing Completion Detection for High Speed and Area Efficient Arithmetic is one such design [1] which try to exploit a unique characteristic of a Ripple Carry Adder in improving the performance and power of a Booth Multiplier.

Our focus is low cost, high throughput integer multiplication. To portray the efficacy of our approach we present the design of a 32x32 Wallace Multiplier. Arithmetic circuits, particularly the multiplier, often have a decisive impact on overall timing in electronic systems. Moreover, multiplication being a very important and commonly used operation consumes

a significant portion of the system's power in modern processors. Traditionally, state-of-the-art multiplier designs have focused on the performance over hardware cost and power, with numerous high-speed designs proposed in the literature. Many high speed multipliers employ a carry save approach, based on designs proposed by Wallace [10] and Dadda [11], that first quickly reduces the input based partial products to two numbers without any carry propagation delays, and then quickly adds these two numbers using high speed adders. Enhancements speed up each of these two steps at considerable expense in hardware and power. For example, the high speed multiplier described in [9] employs Booth encoding on the operands to reduce the number of partial products to be reduced by the carry save tree, specialized compressor circuits for carry save partial product reduction, and a mixed carry-select, carry look-ahead topology to add the final two accumulated partial products [12,13,14]. Our goal is to achieve comparable speed for almost all the multiplication operations without many of these enhancements for partial product reduction, and the use of a much lower cost carry ripple adder to generate the final product; only rarely requiring recovery for a timing error due to the activation of a long path. Furthermore, it is critical for the timing error detection and correction circuitry to be fast, low cost and robust. Developing such a timing exception recovery capability is a major contribution of this thesis.

To achieve this error recovery we present a novel low cost approach which involves latching and holding logic values at key internal circuit nodes during every clock cycle beyond the clock edge to allow recreation of the correct outputs for that clock period one clock cycle later in case of a timing error. Then very fast error evaluation, exploiting a unique characteristic of carry ripple addition, allows this hold to be quickly released without any impact on the circuit timing in the next period cycle in error free operation. The rare detection of a timing error (long carry propagation path) extends the hold signal for additional one or more clock cycles to give the computation time to complete. *Note that this approach is very different from those previously proposed for Razor or other work in the literature, and is particularly well suited to exploit the architecture of low cost Wallace multipliers.*

1.3 Problem Statement

While in the earlier work timing slack due to PVT variation was addressed and overcome in Razor, CRISTA a circuit design technique where the clock frequency can be increased by a factor of two by making the delay paths skewed was addressed. In this thesis we try to improvise the following by exploiting the innate skewed distribution property of a Wallace Multiplier with Ripple Carry Adder as a final vector merger.

1) A better recovery mechanism to quickly recover from the occasional timing errors.

2) Exploit the Skewed distribution property and PVT timing margin slack of a circuit.

3) Increase the clock frequency by a factor of two or more.

1.4 Contribution of This Thesis

In this thesis we present a novel design technique to leverage the input based statistical distribution of a Ripple Carry Adder, further this design technique is implemented on a Wallace Multiplier with a Ripple Carry Adder as a final Adder. The key contribution of this thesis is achieving a performance benefit of close to 2.5x compared to the actual design i.e., clock period determined by the worst case delay. The performance achieved was comparable to that of the fastest multiplier which is supposed to be hardware intense.

An analysis of how the delays are spread in a Ripple Carry Adder was done using unit delay simulation in Modelsim. The skewed delay distribution of the Ripple Carry Adder was as expected and this delay distribution was leveraged after implementing a Wallace Multiplier with Ripple Carry Adder as final adder. The key concept is to hold the internal signal at the final adder beyond the next clock edge and release them quickly if there are no active carry signal propagating within the Ripple Carry Adder thus, providing additional two/three clock cycles in the case of occasional timing paths which go beyond the already set clock period. Spice simulation results on this design for 10,000 random vectors with progressively decreasing clock period shows a performance benefit of 2.36x.

1.5 Organization of Thesis

The thesis is organized as follows. In Chapter 2, we discuss a general background of the earlier Better than Worst Case Design techniques, their pros and cons, and their use in narrower but widely used designs such as Adders and Multipliers. Chapter 3 presents a detailed overview of the performance of Ripple Carry Adder, Carry Look Ahead adders and other complex adders. Chapter 4 presents a brief overview of various stages in a Multiplier and how we leverage the unique characteristics of a RCA. In Chapter 5, the proposed BTWC design is discussed. Implementation of the proposed BTWC design in a Wallace Multiplier is discussed in Chapter 6. Simulation results with Conclusions and future work are discussed in Chapter 7.

Chapter 2

Literature Survey

This chapter presents a brief study on the present and past research work on better than worst case delay designs. Though BTWC designs were implemented earlier either for performance improvement or for power aware computation the design has to make sure that a circuit operates without any errors even in the worst case scenario. Thus, the design of a efficient Error resilient logic and a mechanism which can recover from a timing critical error can be thought of as a metric for the designs based on BTWC delay. The completion detection and correction in this work is novel, and can be implemented on designs whose delay distribution is skewed. Also, it is important to keep in mind that power or performance is a compromising design metrics i.e. one needs to trade power for performance or vice versa.

2.1 Requirement for BTWC designs

After decades of astonishing improvement in integrated circuit performance, digital circuits have come to a point in which there are many problems ahead of us. Two main problems are power consumption and process variations.

In the past few decades, circuit designs have followed Moores law and the number of transistors on a chip has doubled every two years. As we fit more transistors into a given area and clocked them faster and faster, power density increases exponentially. The most effective way to reduce power density is to minimize the supply voltage, as predicted by CV^2f. Currently, we have been successful in containing the power density within tolerable levels, but this will not last. One barrier comes from the threshold voltage. In order to maintain the same performance, we have to reduce the threshold voltage together with the

supply voltage. However, reducing threshold voltage leads to an exponential increase in off-state leakage current. Leakage current has become so significant that a further reduction in threshold voltage and supply voltage has slowed or even stopped. Without voltage scaling, the power density of a chip will increase without bound. If a solution to this ever increasing power consumption cannot be found, Moores law will come to an end and we will no longer be able to experience the tremendous increase in performance experienced in the past.

The other problem of the IC industry is process variations [17,18]. As transistor sizes approach atomic levels, it is very difficult to fabricate exactly what we specify. For example, a variation of dopant implantation on order of a few atoms may translate to a huge difference in dopant concentration, and may cause a noticeable shift in the threshold voltage. Because traditional designs dictate that our circuit must always function correctly in all circumstances, the huge process variations present today forces designers to allocate more voltage margin on top of the typical case in order to ensure proper operation. To make things worse, other temperature, die-to-die, and IR drop variations further increases safety margins needed. The general practice of conservative over design has become a barrier to low power design. Because these large margins are used only to prevent the rare worst case scenario from failing, a large amount of energy can be saved if these margins are eliminated and instead utilize an error-resilient logic that can dynamically tune itself for all kinds of variations. We will then be able to run our chip at the lowest possible energy consumption or highest possible performance .

Power consumption and variations have become two of the most important roadblocks for extending Moores law and these problems must be addressed before we can continue to improve the performance of electronics at the amazing pace we enjoyed in the past decades. Thus even a minor improvement in performance without trading off power shall be a significant contribution in the face of the above mentioned bottlenecks of digital circuits.

2.1.1 Previous BTWC designs

A number of better-than-worst case (BTWC) designs have been proposed in the past to allow circuits to save power by operating under normal conditions rather than conservative worst case limits. One class of BTWC techniques specifies multiple safe voltage and frequency levels that a design may operate at and allows for switching between these states. Examples in this class are correlating VCO [23, 24] and Design-Time DVS [27]. As voltage changes, correlating VCO adapts the frequency of a circuit to a level slightly below the critical frequency. The clock frequency for a given voltage is selected to incorporate a margin for process and temperature variations, as well as noise in the power supply network. Thus, scaling past the critical point is not allowed.

Similarly, Design-Time DVS provides the capability to switch between multiple voltage / frequency operating points to match user or application requirements. As with correlating VCO, each operating point incorporates a conservative margin to ensure that errors do not occur.Another class of BTWC designs uses canary circuits to detect when arrival at the critical point is imminent, thus revealing the extent of safe scaling. Delay line speed detectors [25] work by propagating a signal transition down a path that is slightly longer than the critical path of a circuit. Scaling is allowed to proceed until the point where the transition no longer reaches the end of the delay line before the clock period expires. While this circuit enables scaling, no scaling is allowed past the critical path delay plus a safety margin.

Another similar circuit technique uses multiple latches which strobe a signal in close succession to locate the critical operating point of a design. The third latch of a triple latch monitor [26] is always assumed to capture the correct value, while the first two latches indicate how close the current operating point is to the critical point. A design is considered to be tuned when the values in the first two latches do not match but the values in last two latches do match, indicating that the setup time of the third latch is longer than the critical delay of the circuit by a small margin.

All the BTWC techniques mentioned above have similar limitations. They allow for scaling up to, but never beyond,the critical operating point. However, with increasing variability in circuits, there is also high potential for benefit (in terms of power e.g.) when scaling is allowed to proceed past the critical point. Razor [3,4,5] actually allows voltage scaling past the critical point, since it incorporates error detection and correction mechanisms to handle the case when errors occur. While CRISTA [6] allows aggressive voltage scaling by Isolating and predicting the set of possible paths that may become critical under process variation and ensure that they are activated rarely, it avoids possible delay failures in the critical paths by dynamically switching to two-cycle operation.

On the other hand CSCD [1] provides carry completion signaling in low cost ripple carry adders which allows the control logic to schedule the next addition as soon as an earlier one is complete, thereby achieving the average case, rather than worst case addition delay over a set of computations.

While all the above mentioned designs can be considered as BTWC. Only Razor, CRISTA and CSCD fall under a less conservative design as each design has unique way to sense the timing paths completion, and the challenge is to design a least conservation design technique. Instead of applying this approach to general purpose logic as attempted by the Razor team and others, we have explored its use in specific widely used computations such as addition and multiplication.

2.2 Razor

The Razor approach, proposed in [3], aims at reducing the power consumption by minimizing the PVT timing margin to zero and beyond (since timing critical paths are not activated in every cycle), by building in a system capability to detect occasional errors due to slow signal paths and recover from them. The timing margins are removed by reducing the supply voltage to slow down the circuit to a point where a small, acceptable number of errors are observed. As long as the power saving from the reduced voltage operation

exceeds the extra power needed, on average, by the occasional error detection and recovery cycle, such a scheme can provide a net power saving. The challenge clearly is in designing an efficient low cost error detection and recovery capability to support this approach. The original Razor design [3] has changed and evolved [4,5] significantly over time in an attempt to achieve practicality. Even so the potential power savings from eliminating timing margins appear limited.

2.2.1 Razor I Overview

The key concept in Razor I [3] is to sample the input data of the flip-flop at two different points in time. The earlier, speculative sample is stored in a conventional positive-edge triggered, master-slave flip-flop. This main flip-flop is augmented with a so-called shadow latch which samples at the negative edge of the clock.

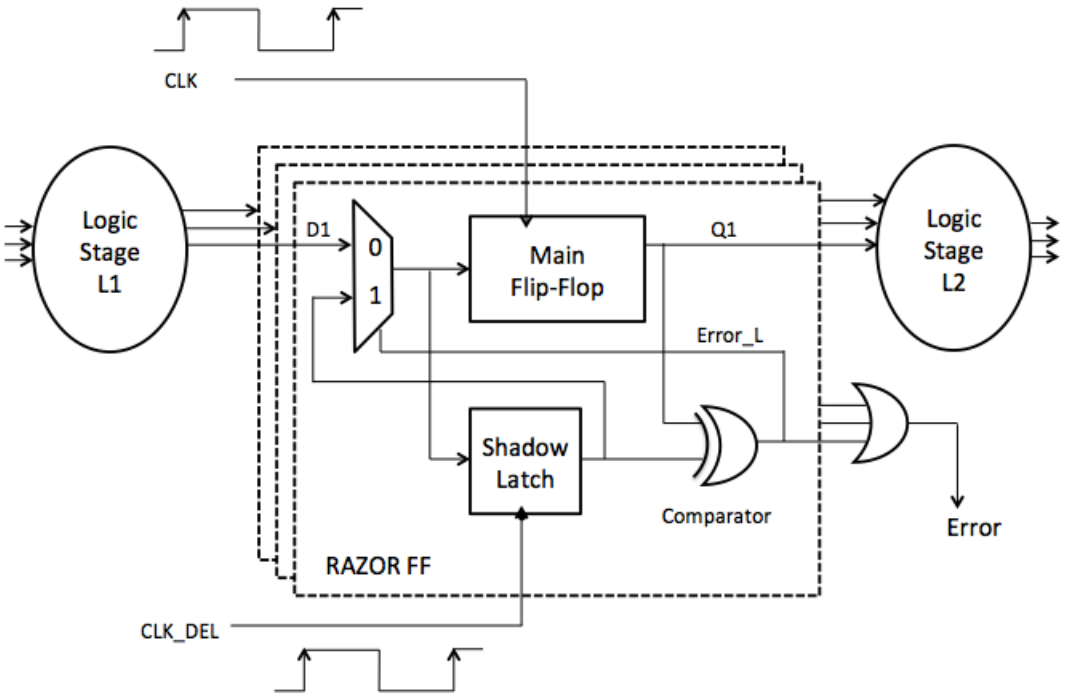

Figure 2.1: Razor flop-flop

Thus, the shadow-latch gets additional time equal to the high-phase of the clock to capture the correct state of the data. An error is flagged when data captured at the main

flip-flop differs from the shadow-latch data. As the setup and hold constraints for the main flip-flop are allowed to be violated, an additional detector is required to flag the occurrences of metastability at the output of the main flip-flop. The error-pins of individual RazorI flip-flops are then OR-ed together to generate a pipeline restore signal which overwrites the correct data in the shadow-latch into the main flip-flop, at the next positive edge of the clock. Since the shadow latch data is used to overwrite state in the main flip-flop, it is required to ensure using conventional worst-case techniques that the data in the shadow latch is always correct.

There are key design issues that complicate the deployment of RazorI in high-performance, aggressively-clocked microprocessors. The primary difficulty is the generation and propagation of the pipeline restore signal. The restore signal is evaluated at the output of a high fan-in OR-tree and is suitably buffered and routed to every flip-flop in the pipeline stage before the next rising edge of the clock. This imposes significant timing constraints on the restore signal and the error recovery path can itself become critical when the supply voltage is scaled. This limits the voltage headroom available for Razor, especially in aggressively clocked designs. The design of the metastability detector is also difficult under rising process variations as it is required to respond to metastable flip-flop outputs across all process, voltage and temperature corners. Consequently, it requires the use of larger devices which adversely impacts the area and power overhead of the RazorI flip-flop. There is the additional risk of metastability at the restore signal which can propagate to the pipeline control logic, potentially leading to system failure.

2.2.2 RazorII

Razor II [3] was implemented to effectively address the design and timing issues in RazorI which moves the responsibility of recovery entirely to the micro-architectural domain. The RazorII approach introduces two novel components which are described in the following paragraphs.

Instead of performing both error detection and correction in the flip-flop, RazorII performs only detection in the flip-flop, while correction is performed through architectural replay. This allows significant reduction in the complexity and size of the Razor flip-flop. although at the cost of increased IPC penalty during recovery. Architectural replay is a conventional technique which often already exists in high-performance microprocessors to support speculative operation such as out-of-order execution and branch prediction. Hence, it is possible to overload the existing framework to support replay in the event of timing errors. In addition, this technique precludes the need for a pipeline restore signal, thereby significantly relaxing the timing constraints on the error recovery path. This feature makes RazorII highly amenable to deployment in high-performance processors.

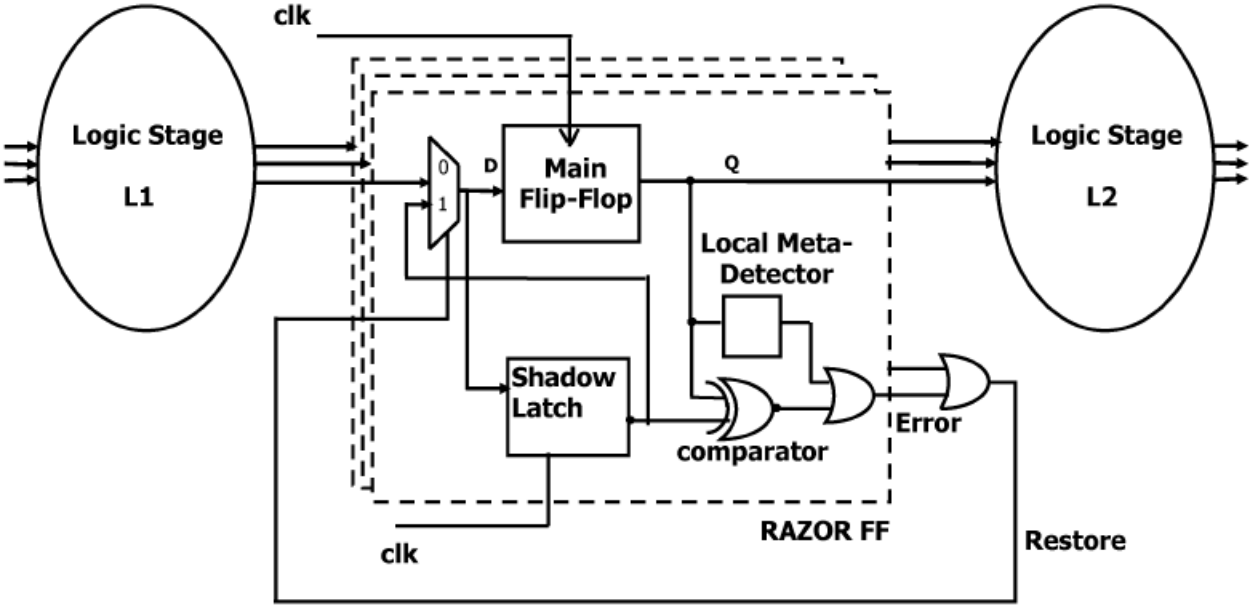

Figure 2.2: Pipeline augmented with Razor latches and control lines

Besides, the design of the RazorII flip-flop uses a positive level-sensitive latch instead of a master-slave flip-flop. The flip-flop operation is enforced by flagging any transition on the input data in the positive clock-phase as a timing error. Elimination of the master latch significantly reduces the clock pin capacitance of the flip-flop bringing down its power and area overhead.

2.2.3 Razor Limitations

Designs such as Razor that allow scaling past the critical operating point [ref (rl) J. Patel. Cmos process variations: A critical operation point hypothesis, 2008.] must be mindful of two aspects of error recovery - error detection and correction. Razor detects an error when the value latched by the shadow latch differs from the value latched by the main flip-flop. This happens when the logic signal has not settled to its final value before the setup time of the main flip-flop. If the signal transitions again before the shadow latch latches, an error will be detected.

For error correction, the Razor flip-flop must not only detect a timing violation, but must also latch the correct value in the shadow latch. This simply implies that the correct value must arrive by the setup time of the shadow latch for all Razor flip-flops in a design. So, Razor may not be able to correct errors if a) detection fails (i.e., both the main flip-flop and the shadow latch have the same incorrect value), or b) detection succeeds, but the value latched in the shadow latch is not the correct value.

To guarantee correctness, Razor requires two conditions to be met on the circuit delay behaviour the short path constraint and the long path constraint. The long path constraint (eqn. 2.1), states that the maximum delay through a logic stage protected by Razor must be less than the clock period (T) plus the skew between the two clocks (the clock for the main flip-flop and the clock for the shadow latch).

$$delay < T + skew \tag{2.1}$$

If the long path constraint is not satisfied, false negative detections can occur when a timing violation causes both the main flip-flop and shadow latch to latch the incorrect value. The short path constraint (eqn. 2.2) states that there must not be a short path through a logic stage protected by Razor that can cause the output of the logic to change before the shadow latch latches the previous output.

$$delay_{min} > skew + hold \tag{2.2}$$

Failure to satisfy the short path constraint leads to false positive error detections when the logic output changes in response to new circuit inputs before the shadow latch has sampled the previous output. Combination of the short and long path constraints (eqn. 2.4) demonstrates that Razor can only guarantee correctness when the range of possible delays for a circuit output falls within a window of size

$$skew + hold < delay < T + skew \tag{2.3}$$

$$delay_{max} - delay_{min} < T - hold \tag{2.4}$$

Note that equation 2.4 implies a tradeoff between the limit of Razor protection and the range of Razor usability. While increasing skew can reduce the number of uncorrectable errors by protecting longer path delays, this also leads to a reduction in the range over which Razor can be applied to correct errors due to violation of the short path constraint.

The authors of [22] try to demonstrate the voltage scaling limitations of Razor based design using two circuits which are chosen to be canonical examples of two contrasting design philosophies i.e., a Kogge-Stone adder (KSA) and a Ripple Carry Adder (RCA)

The KSA architecture has timing paths of nearly the same length, and therefore exhibits a critical operating point akin to traditional high performance processor designs. Due to this property of KSA, it was observed that in [ref] scaling beyond a certain voltage point leads to a catastrophic failure of the adder (i.e., 100% error rate). Aggressive voltage scaling, therefore, is not possible for such designs even with an efficient error correction mechanism

as the power consumption may actually increase drastically in spite of voltage scaling. This is because the absolute error rate is high (close to 100%) and the overhead of error recovery for Razor is roughly an order of magnitude more expensive than the overhead of executing an instruction normally [ref]. So, for designs like KSA where timing paths are bunched up (like in traditional high performance processor designs), Razor may not be very effective in terms of power reduction through undervolting (1.e., scaling beyond the voltage for which the first timing violation appears). While some power can be saved by eliminating the voltage guardband, scaling past the critical operating point results in nearly 100% erroneous computations.

The second circuit i.e., ripple carry adder (RCA architecture is not subject to catastrophic failure in response to scaling past the point of first error. The delay distribution show in fig 1 of chapter 1 implies that the error rate increases gradually as voltage decreases. Although the minimum delay for any path of the RCA equals the delay of the sum path of a full adder, operational delay ultimately depends on adder inputs, which generate carry chains from lower to higher order bits. The RCA exhibits maximum delay when the carry chain extends from the least significant bit to the most significant bit. However, on average, carry chains are much shorter, leaving extensive room for aggressive scaling past the point where errors begin to occur. In fact, the error rate reaches close to 100% only at very low voltages.

The behaviour of RCA may be a suitable desired behaviour for high performance processor designs to enable significant power savings through undervolting. Recent attempts [20, 21] at processor designs that produce graceful degradation in reliability in the face of voltage scaling try to mimic this behaviour.

It might be obvious that Razor should perform well for architectures that fail gracefully, since such designs do not have a wall of criticality. However, analysis of the results in [22] reveals some serious limitations of using Razor, even in such architectures.

Limitations arise due to the potential short path and long path constraint violations as discussed earlier. If the long path constraint is not satisfied, false negative detection can occur when the main flip-flop and shadow latch both latch the incorrect value. Similarly, the failure to meet short path constraints makes Razor unusable over a range of voltages.

In fact, the same factor that makes the error behavior of RCA graceful (skewed delay distribution) makes Razor less effective. This is because Razor relies on the variation in delay to be less than a threshold. The variation in delay is significantly larger for an RCA design than a KSA design. In order to make Razor work for circuits that fail gracefully, buffering must be used to increase the delay of short paths, thus shifting them into the window of correction. This buffering adds area and power overheads in a design, negating some of the power savings afforded by better than worst-case design. Secondly, required buffering increases the delay on short paths, transforming a circuit from one that fails gracefully to one that fails catastrophically, thus limiting the extent of possible scaling.

So, while Razor is ineffective for circuits like KSA because of massive timing violations in the face of undervolting, it is also not very effective for circuits like RCA due to a large span between the maximum and minimum circuit delays. The results in [ref razor limits] demonstrate the inadequacies of current better than worst-case design methodologies (like Razor) in terms of voltage/ frequency scaling, motivating the need for new techniques for processor design and error handling.

2.3 CRISTA

CRISTA [6] is a low-power variation-tolerant circuit design called CRitical path ISolation for Timing Adaptiveness [ref crista], which allows aggressive voltage scaling. The principal idea includes the following:

1.) isolates the critical paths and makes them predictable (based on few primary inputs) under parametric variation so that with reduced supply voltage, possible delay errors under single-cycle operation are deterministic and can be avoided by a two-cycle operation.

2.) restricts the occurrences of the previous two-cycle operations by reducing the activation probability of critical paths.
3.) increases the delay margin between critical and noncritical paths by both logic synthesis and proper gate sizing for improved yield, reliability of operations, and aggressive voltage scaling.

As mentioned above CRISTA induces the skewed distribution in completion delays for a given pipeline stage logic and further downsizes the critical path gates to increase the timing slack between non-critical and critical paths for performance/power benefits in a circuit. The occurrence of an error is detected by a Decode logic which monitor the activation probability of critical path vectors.

It can be understood intuitively that the performance or power benefit from such a design technique for designs such as a 32-bit RCA can be minimal. The primary reason being RCA has an innate skewed distribution as shown in [fig chap 1] besides the decode logic used to flag for two cycle operation could impose additional hardware penalty.

Therefore, though the design can be considered as a less conservative design for generic circuits with balanced critical paths it might not gain huge performance benefits from designs with skewed delay distribution such as RCA.

2.4 CSCD for High Speed and Area Efficient Arithmetic

This design [1] equips a Ripple Carry Adder with a current sensing capability which observes late settling carry signal nodes in the circuit and indicates when they reach a quiescent state. The incorporation of a computation completion signal into a RCA offers a way for improving the "average case" RCA to signal to the higher level circuitry controlling it that it has completed the operation. Thus, for example, if 32 repeated additions are to be performed to multiply two 32 bit numbers, using completion signalling to initialize the next addition can cut down the total multiplication time from 32 worst case addition delays, to 32 average case delays.

The proposed method for implementing the current sensor involves using a sense-inverter such as the one shown in fig-2.3 . A CMOS inverter draws current from the power supply as long as its input is midrange between a high and a low voltage, such that both the P and N transistors see a gate source voltage above their respective threshold voltages. The Supply current does not flow once the input reaches within a threshold of either the high or low supply voltages. Thus monitoring the supply current provides a means to determine if the input has stabilized close (within a threshold voltage) of a high or low.

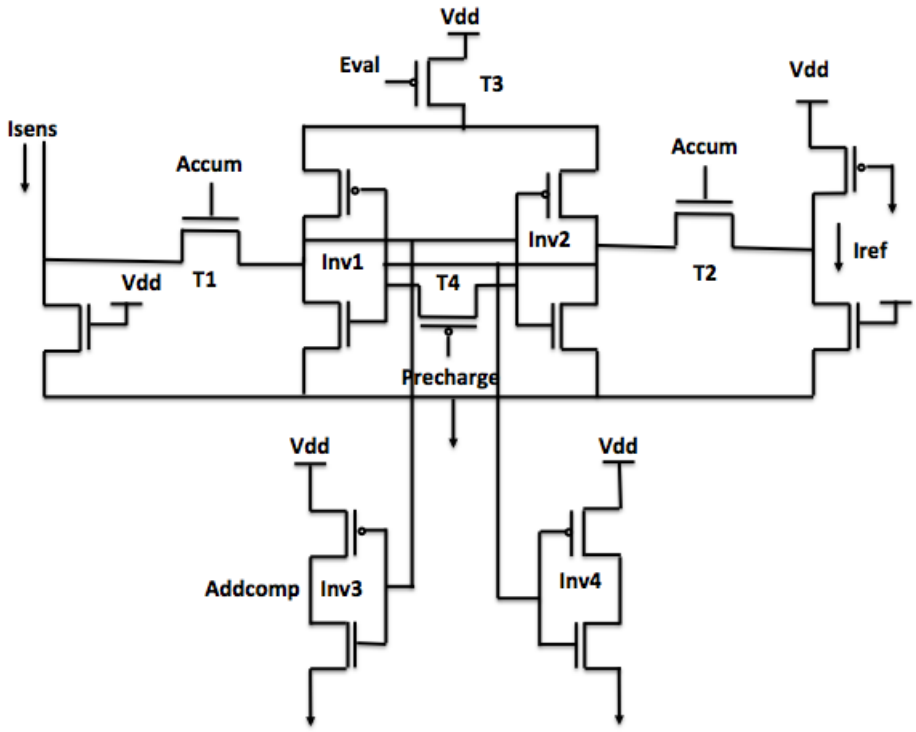

Figure 2.3: Current Sensor Circuitry

In fig-2.3 the sense current is drawn from a transient current generator i.e., an inverter whose IDD quiescent current is monitored with respect to the rising clock edge of each capture flop. The Addcomp signal is flagged in the case of an addition completion.

Though this design seems to be promising for circuits with a skewed delay distribution such as a RCA, the loading on the sense circuitry increases linearly with the number of

capture flops to be monitored for transition completion. Thus, imposing an upper bound on performance or power benefits.

2.5 Need for Fast and Resilient Error Detection Circuit

Though designs such as RAZOR, CRISTA and CSCD are less conservative compared to other BTWC designs the performance/power benefits for a design with skewed delay distribution (like a RCA) using any of the above techniques is limited.

Thus, to exploit the innate skewed distribution of a much narrower but widely used arithmetic designs i.e. Multiplier (With a RCA as a final Vector Merger) it is important that an Error detection technique which is fast and resilient is required. The major contribution of the work in this thesis is the design of a Fast Error Detection logic which makes the performance of a Multiplier with a low cost RCA comparable to that of the fastest Multiplier.

Chapter 3

Brief Overview on Adders

In this chapter the performance of a Ripple Carry Adder for delay in the best and worst case scenarios is discussed. Also ripple carry adder performance and area is compared to a different flavors of adders such as Carry Look Ahead (CLA), Carry Save Adder (CSA), Kogge Stone Adder etc..

3.1 Basic Adders

The basic building blocks for any adder are Half Adder and Full Adder (3:2 compressor), there are several complex compressors in the literature but a good insight in Half Adder and Full Adder will give an intuitive understanding in characterizing the performance of complex adders.

3.1.1 Half Adder

The half adder adds two single binary digits A and B. It has two outputs, sum (S) and carry (C). The carry signal represents an overflow into the next digit of a multi-digit addition. The simplest half-adder design, pictured in Fig 3.1, incorporates an XOR gate for S and an AND gate for C.

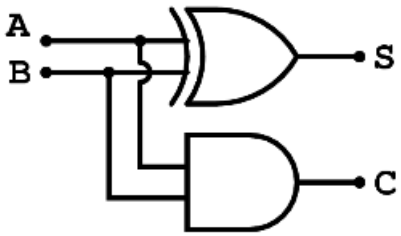

Figure 3.1: Half Adder

In the above figure the Worst case delay is one XOR gate delay.

3.1.2 Full Adder

A full adder performs additions of three 1-bit numbers, A, B, Cin, and gives outputs Sum, and Carry out, Cout. The expressions for sum and carry are given by

$$S = A \oplus B \oplus Cin \tag{3.1}$$

$$Cout = A \cdot B + Cin(A \oplus B) \tag{3.2}$$

$$Cout = A \cdot B + A \cdot Cin + B \cdot Cin \tag{3.3}$$

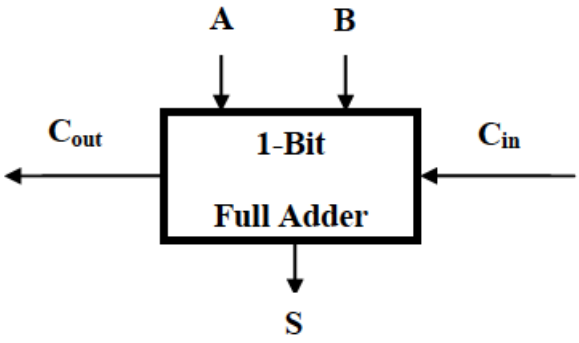

Figure 3.2: Full Adder Block

Two half adders can be combined to make a full adder with the addition of an OR gate to combine their carry outputs. Using Eqn 3.1 and 3.2 we can represent a Full Adder at gate level as shown in Fig 3.3. It can be observed from the figure that the critical path for Sum output is determined by two XOR gates, whereas the Critical path for a Carry out is determined by an XOR, AND and an OR gate. Since, carry output is the one which

propagates in adders with bit length > 1 it is important to build a full adder which can generate a carry output faster as shown in fig 3.4.

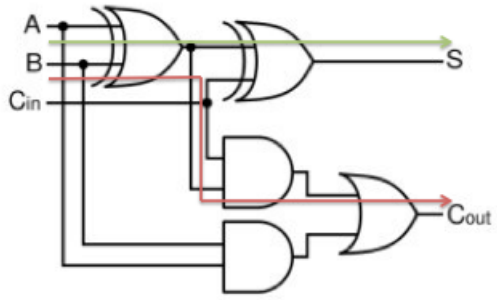

Figure 3.3: Full Adder using two Half Adders

In Fig 3.4 the carry out delay is determined by an AND gate and an OR gate (3-input) which is much faster than the fig 3.3. Henceforth, a full adder is considered to be the one in fig 3.4 in this thesis.

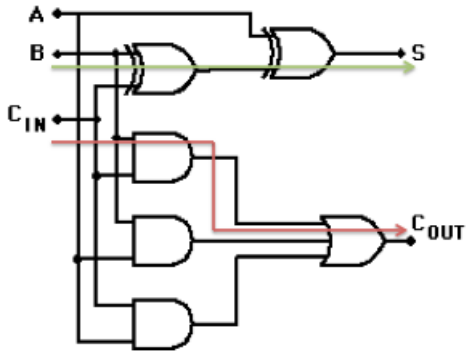

Figure 3.4: Full Adder with fast carry out

3.2 Ripple Carry Adder (RCA)

A Ripple carry adder consists of blocks of 1-bit full adders connected in series with the carry out of one block serving as carry in to the next block. Fig 3. shows the interconnection of a 4-bit ripple carry adder. It can be observed that the critical path for this design is the path from C0 to C4.

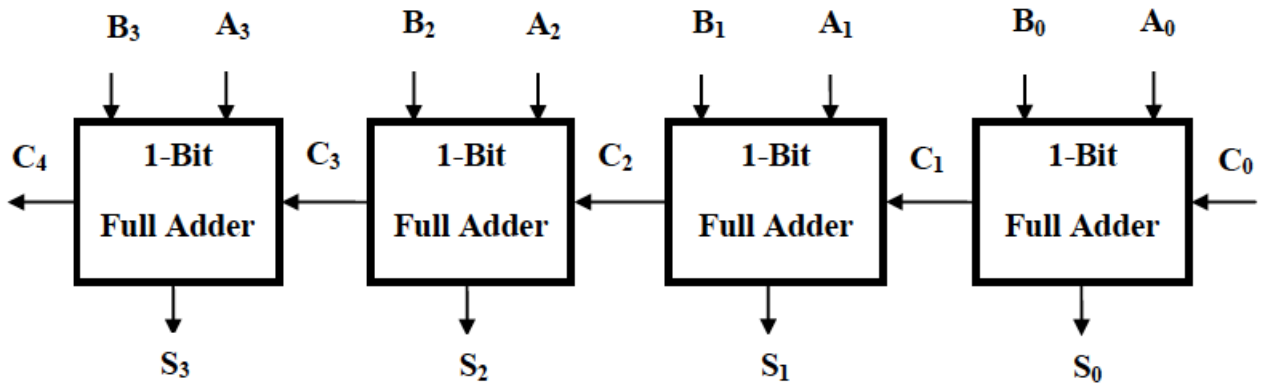

Figure 3.5: 4-bit Ripple Carry Adder

3.2.1 Performance Analysis of a 4-bit RCA

Figure 3.6 shows the gate level representation of 4-bit full adder circuit with sum and carry outputs. The addition of two bits at any stage depends on the carry generated by the addition of the two bits in the previous stage. Thus, the sum of the most significant bit is only available after the carry signal has rippled through the adder from the least significant stage to the most significant stage. The critical path for this design is the carry propagation path from C0 to C4 which is 8 gate delays. Extending the design concept for 32- bit ripple carry adder gives us the critical path delay of 64 gates.

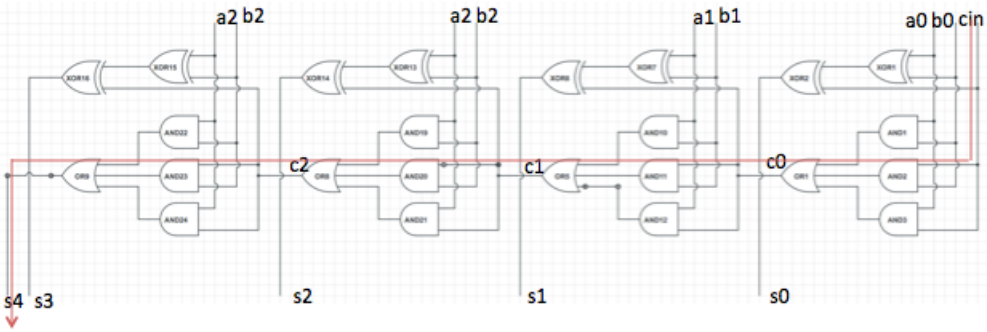

Figure 3.6: 4-bit Ripple Carry Adder

3.2.2 Best Case Performance

The best case performance for a ripple carry adder is seen when there is no carry generated at each bit position. In the case of the 4-bit adder in Figure 20 the sum outputs

are available just after two xor gate delays. This can be observed with input values A = 1111, B = 0000 and C0=0, which gives C1=C2=C3=C4=0 at each bit position. For this case there are no carry bits generated and since no propagation takes place the sum outputs are valid after two gate delays.

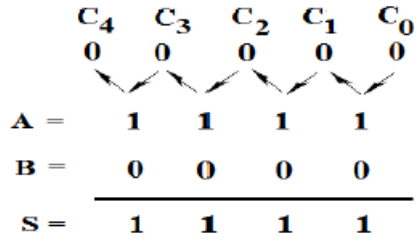

Figure 3.7: 4-bit adder best case performance

Delay

S_0 = 2 gate delays

S_1 = 2 gate delays

S_2 = 2 gate delays

S_3 = 2 gate delays

3.2.3 Worst Case Performance

The worst case performance for a ripple carry adder is seen when the input carry, Cin is propagated to next stage by each bit position. Because of carry propagation from the least significant position to most significant position the sum bits generation takes variable time delays.

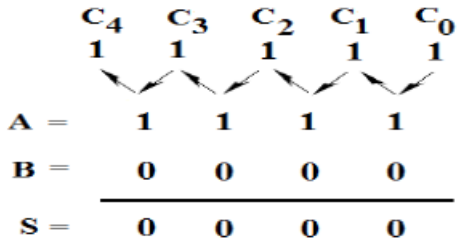

Figure 3.8: 4-bit adder worst case performance

Delay

S_0 = 2 gate delays

S_1 = 4 gate delays

S_2 = 6 gate delays

S_3 = 8 gate delays

In the case of the 4-bit adder in Figure 3.6, this can be observed with A = 1111, B = 0000 and C0=1, which gives C1=C2=C3=C4=1 at each bit position. Since the carry is propagated the delays for the sum outputs is observed to be two to eight gate delays.

3.2.4 Average Case Performance

The average case performance of a ripple carry can be obtained from the delay distribution of the RCA. To look at this from a statistical perspective let us consider the probability of having a '10' or '01' combination at the input of the Full Adder i.e., two out of four cases the Full Adder inputs can have the above combination (A carry can only propagate to the next full adder only if the input bits are either '10' or '01') so the probability that a carry propagates to the next Full Adder is $\frac{1}{2}$. Hence, the probability that a carry propagates beyond the n^{th} stage is $\frac{1}{2^n}$ i.e., for a 5 bit adder the critical path is triggered once in 32 cases as shown in Figure 3.9. Hence, the skewed nature of RCA becomes more prominent with the RCA bit length.

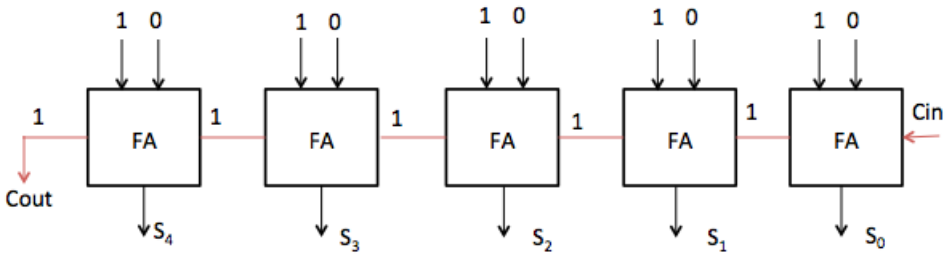

Figure 3.9: Illustrating the probability of carry rippling through a 5 stage RCA

In figure 3.9 we try to illustrate the probability of a carry being propagated all the way down to C_{out} in a 5 stage Ripple Carry Adder.

3.3 Carry Look Ahead Adder

The carry look ahead adder calculates carry bits simultaneously using complex logic circuitry. This minimizes the worst case delay to calculate the sum bits. Due to large fan out on the gates, the carry generation and propagation bits are calculated by making groups of carry blocks (usually four bits) as shown in Figure

The primary block of the carry look ahead adder is Partial Full Adder (PFA). Each block of PFA takes three bits A, B, C; as inputs and generates three outputs G, P and S. These are

Generate, G = A . B

Propagate, P = A⊕B

Sum, S = A ⊕ B ⊕ C

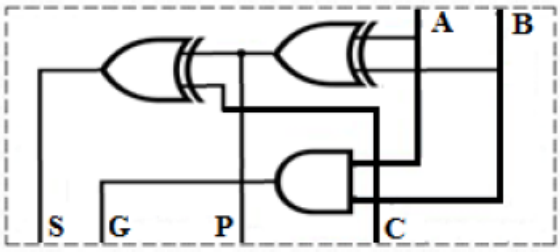

Figure 3.10: Partial Full Adder

3.3.1 Performance Analysis of 4-bit CLA

Fig 3.10 shows the gate level architecture of a 4-bit carry look ahead adder. In order to construct a 4-bit CLA four PFAs are needed to generate the signals. When Pi = 1, an incoming carry is propagated to the next bit position from C_i to C_i+1. For $P_i = 0$, carry propagation to the bit position is blocked. Regardless of the value of P_i, when $G_i = 1$, the carry output from the current position is "1.

The carry output has the following logic:

C1 = G0 + P0C0

C2 = G1 + P1C1 = G1 + P1G0 + P1P0C0

C3 = G2 + P2C2 = G2 + P2G1 + P2P1G0 + P2P1P0C0

C4 = G3 + P3C3 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0C0

The carry look ahead block takes only two gate delays to generate carry bits from C_1 through C_3. The implementation of C_4 is becomes more complicated when this 4-bit carry look ahead adder is extended to multiples of 4 bits, such as 16 bits and 32 bits. The carry look ahead adder for 4 bits computes carry bits at all the bit positions simultaneously. The longest delay in the 4-bit carry look ahead adder is 4 gate delays, compared with 8 gate delays in the ripple carry adder. The improvement is very modest but at the cost of much additional hardware. From Figure 22 it can be observed that the fan in for generating the C_4 is 4, if we extend the same concept to generate more carry bits simultaneously then the high fan-in for generating the carry bits could contribute to additional adder delays. So in such cases the carry bits are limited to groups of 4 and are extend to next higher level of blocks to generate the carry bits. For a 16 bit carry look ahead adder the higher numbered bits 4 - 7, 8 - 11, and 12 - 15 are grouped together. For this in positions 4, 8, and 12 we would like the carry to be produced as fast as possible without using excessive fan-in. The estimated worst case delay for a 32-bit CLA is 8 gate delays but high fan-in in the carry generation block could add additional delays in the adder circuit.

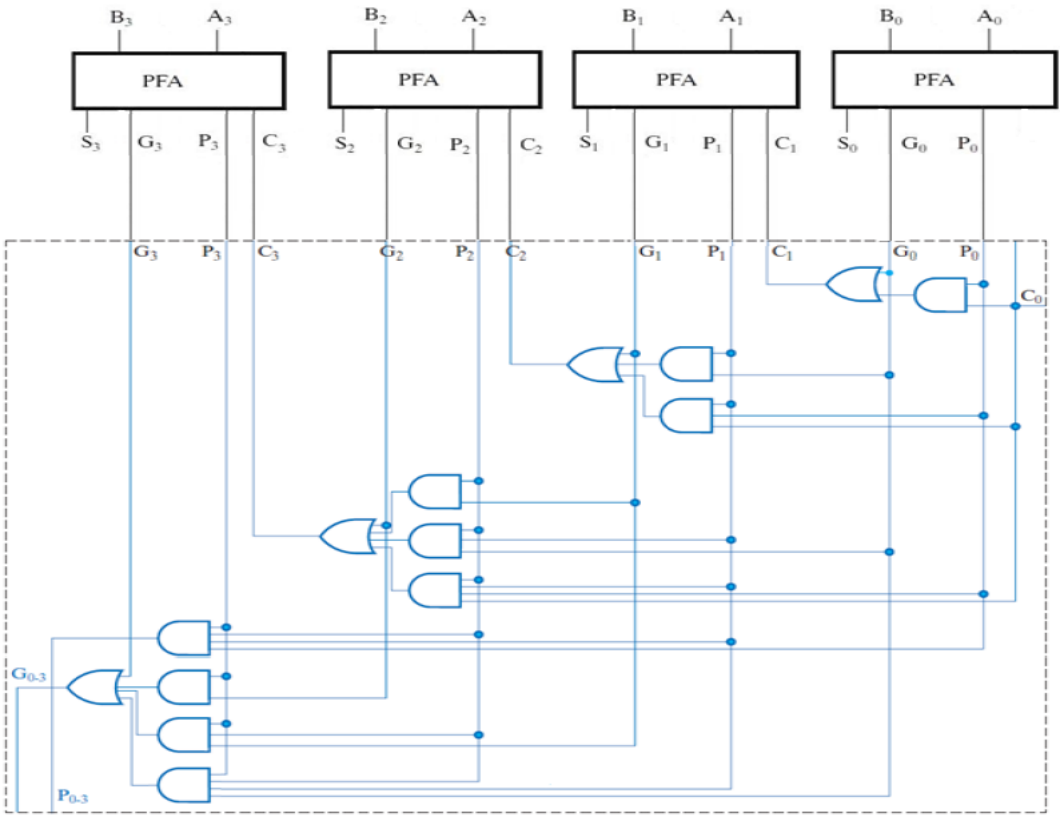

Figure 3.11: 4-bit Carry Look Ahead Adder

3.4 Carry Select Adder

In a Ripple Carry Adder, every full-adder cell has to wait for the incoming carry before an outgoing carry can be generated. One way to get around this linear dependency is to anticipate both possible values of the carry input and evaluate the result for both possibilities in advance. Once the real value of the input carry is known, the correct result is easily selected with a simple multiplexer stage. An implementation of this idea, appropriately called the *carry-select adder* is demonstrated in Figure 3.11

Consider the block of adders, which is adding bits k to k+3. Instead of waiting on the arrival of the output carry of bit k-1, both the 0 and 1 possibilities are analysed. From a circuit point of view, this means that two carry paths are implemented. When C_{in} finally at the last stage finally settles, either the result of the 0 or the 1 path is selected by the multiplexer, which can be performed with a minimal delay. As is evident form the Figure

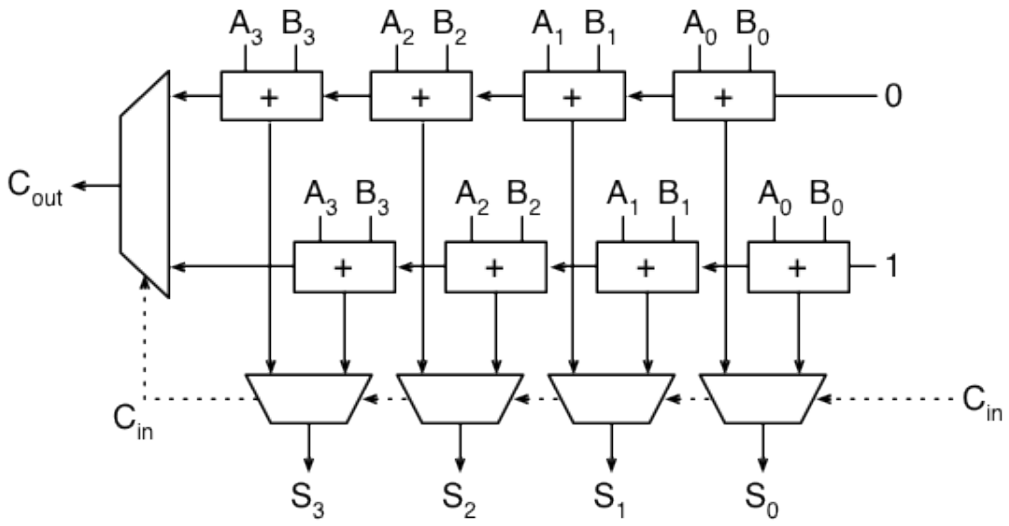

Figure 3.12: Carry Select Adder

3.11, the hardware overhead of the carry-select adder is quite significant with an additional carry path and a multiplexer.

Chapter 4

Overview on Multipliers

Multiplications are expensive and slow operations. The performance of many computational problems often is dominated by the speed at which a multiplication operation can be executed. Many multiplier designs focus on the performance perspective of a multiplier, and may not fit a low-power environment. Hence, the fastest multiplier in the literature comes with the expense of Area, Power or complexity in routing. This chapter tries to discuss some basic multiplier designs from the literature [ref] which are similar to the decimal multiplication using paper and pencil and fastest multiplier with significant overhead in area.

4.1 The Multiplier

Consider two unsigned binary numbers X and Y that are M and N bits wide, respectively. To introduce the multiplication operation, it is useful to express X and Y in the binary representation

$$X = \sum_{i=0}^{M-1} X_i 2^i \quad Y = \sum_{j=0}^{N-1} Y_j 2^j \tag{4.1}$$

with $X_i \in 0, 1$. The multiplication operation is then defined as follows:

$$Z = X \times Y = \sum_{k=0}^{M+N-1} Z_k 2^k = (\sum_{i=0}^{M-1} X_i 2^i)(\sum_{j=0}^{N-1} Y_j 2^j) = \sum_{i=0}^{M-1} (\sum_{j=0}^{N-1} X_i Y_j 2^{i+j}) \tag{4.2}$$

The simplest way to perform a multiplication is to use a single two-input adder [9]. For inputs that are M and N bits wide, the multiplication takes M cycles, using an N bit adder.

This shift and-addd algorithm for multiplication adds together M partial products. Each partial product is generated by multiplying the multiplicand with a bit of the multiplier - which, essentially, is an AND operation - and by shifting the result on the basis of the multiplier bit's position.

A faster way to implement multiplication is to resort to an approach similar to manually computing a multiplication. All the partial products are generated at the same time and organized in the array. A multioperand addition is applied to compute the final product. The approach illustrated in fig 4.1.

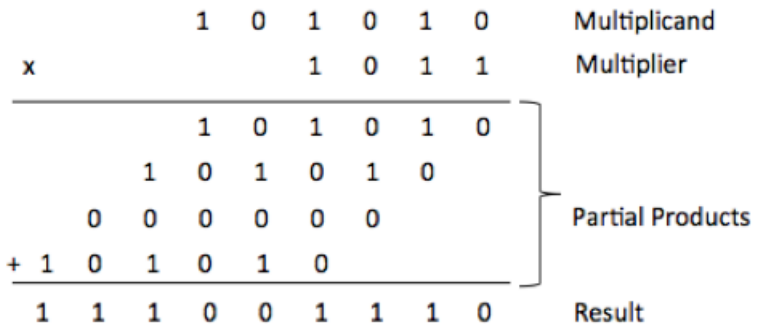

Figure 4.1: Binary Multiplication - an example

This set of operations can be mapped directly into hardware. The resulting structure is called an array multiplier and combines the following three functions:

i) Partial-Product Generation

ii) Partila-Product Accumulation and

iii) Final Addition

4.2 Partial Product Generation

Partial products result from the logical AND of multiplicand X with a multiplier bit Y_i. Each row in the partial-product array is either a copy of the multiplicand or a row of zeros.

It is obvious that this stage of multiplication is done quickly and the maximum delay of multiplier is contributed by the partial-product accumulation and the final addition which have a fixed and a variable delay respectively as discussed in the following subsections.

4.3 Partial Product Accumulation

After the partial products are generated, they must be summed. This accumulation is essentially a multioperand addition. A straightforward way to accumulate partial products is by using a number of adders that will form an array - hence, the name, array multiplier.

4.3.1 Array Multiplier

The most direct approach to multiplication implementation is the Array Multiplier. The composition of an array multiplier is shown in fig. There is a one-to-one topological correspondence between this hardware structure and the manual multiplication shown in fig 4.1. The generation of N partial products requires NxM two-bit AND gates (in the style of fig). Most of the area of the multiplier is devoted to the adding of the N partial products, which require N-1 M-bit adders. The shifting of the partial products for their proper alignment is performed by simple routing and does not require any logic. The overall structure can be easily compacted into a rectangle, resulting in a very efficient layout.

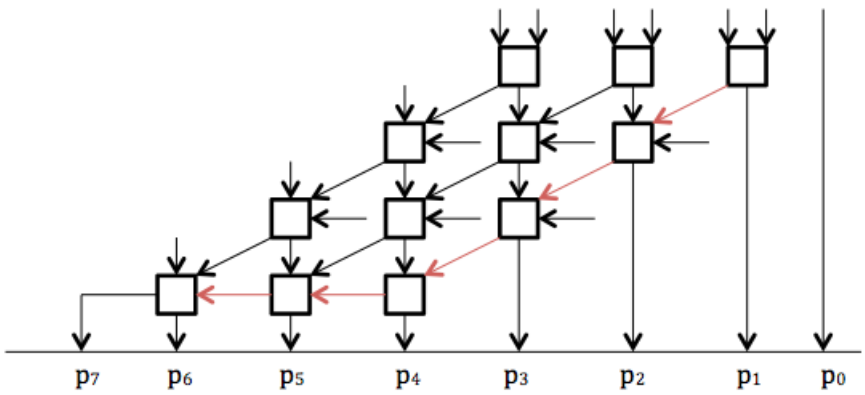

Figure 4.2: 4x4 Array Multiplier

Due to array organization, determining the propagation delay of this circuit is not straightforward. Consider the implementation of fig. The partial sum adders are implemented as ripple-carry structures. Performance optimization requires that the critical timing path be identified first. This turns out to be nontivial. In fact, a large number of paths of almost identical length can be identified. One such path which is the critical path is shown in the fig 4.2. A closer look at the critical path yields an approximate expression for the propagation delay (derived here for critical path), we can represent this as

$$t_{mult} = [(M-1) + (N-1)]t_{carry} + (N-1)t_{sum} + t_{and} \tag{4.3}$$

where t_{carry} is the propagation delay between input and output carry, t_{sum} is the delay between carry and the sum bit of the full adder (as calculated from capter 3 section), and t_{and} is the delay of the AND gate. From equation 4.3 we can say that the timing paths in a Arrray Multiplier are balanced which is illustrated in the Delay Distribution of an Array Multiplier obtained for 50,000 vectors.

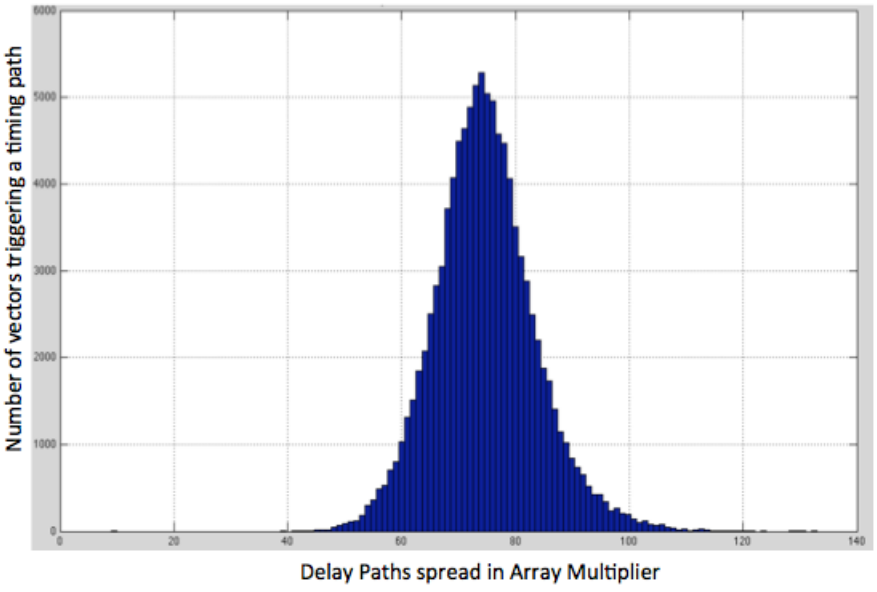

Figure 4.3: Delay Distribution of an Array Multiplier for 50,000 vectors

Since all critical paths have the same length, speeding up just one of them-for instance, by replacing one adder by a faster one such as a carry-select adder- does not make much sense from a design standpoint. All critical paths have to be attacked at the same time. From equation 4.3, it can be deduced that the minimization of t_{mult} requires the minimization of both t_{carry} and t_{sum}. In this case, it could be beneficial for t_{carry} to equal t_{sum}. This contrasts with the requirements for adder cells discusssed before, where a minimal t_{carry} was of prime importance.

4.3.2 Carry-Save Multiplier

Due to large number of almost identical critical paths, increasing the performance of the Array multiplier through transistor sizing yields marginal benefits. A more efficient realization can be obtained by noticing that the multiplication result does not change when the output carry bits are passed diagonally downwards instead of only to the right, as shown in fig

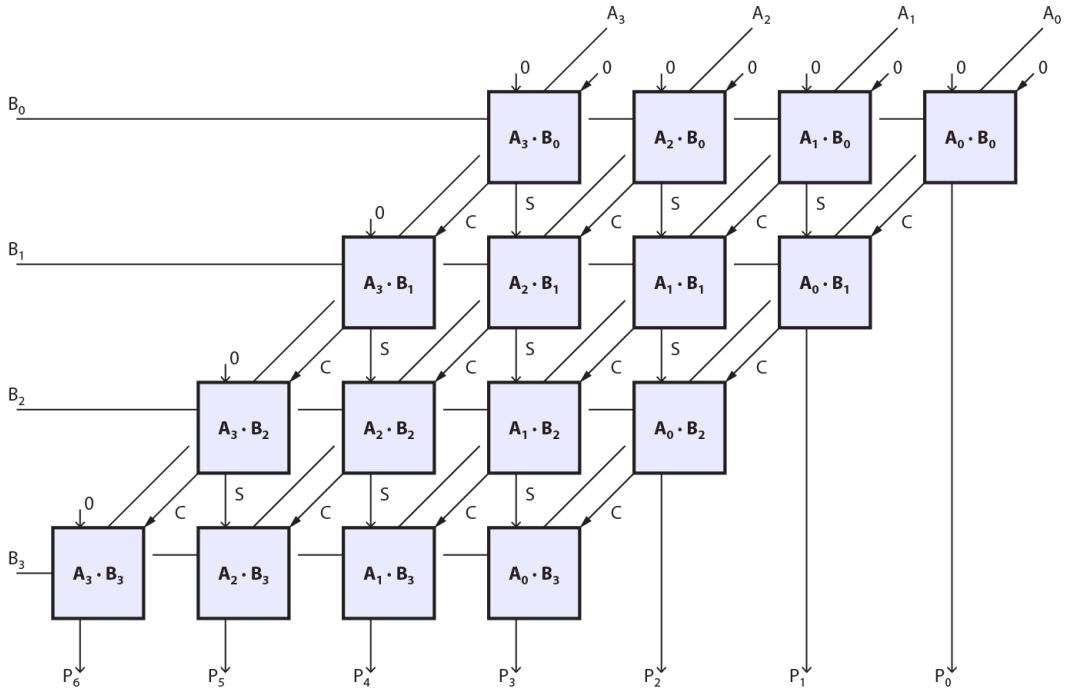

Figure 4.4: A 4x4 carry-save multiplier

An extra adder is included to merge the final vectors to generate the final result. The resulting multiplier is called a *carry-save multiplier* or a, because the carry bits are not immediately added, but rather are "saved" for the next adder stage. In the final stage, carries and sums are merged in a fast carry-propagate (e.g, carry-lookahead or carry-skip) adder stage (the major contribution of this thesis is to attain comparable performance to a Carry Save multiplier design with one of the above mentioned adders as final vector merger without any area overhead). While this structure has a slightly increased area cost (one extra adder), it has the advantage that its worst case critical path is shorter and uniquely defined, and is expressed as

$$t_{mult} = t_{and} + (N-1)t_{carry} + t_{merge} \tag{4.4}$$

4.3.3 Wallace Multiplier

The partial-sum adders can also be rearranged in a treelike fashion, reducing both the critical path and the number of adder cells needed. Consider the simpler example of four partial products each of which is four bits wide, as shown in fig. The number of full adders needed for this operation can be reduced by observing that only column 3 in the array has to add four bits. All other columns are somewhat less complex. This is illustrated in fig, where the original matrix of partial products is reorganized into a tree shape to visually illustrate its varying depth. The challenge is to realize the complete matrix with a minimum depth and a minimum number of adder elements. The first type of operator that can be used to cover the array is a full adder, which takes three inputs and produces two outputs: the sum, located in the same column and the carry, located in the next one. For this reason, the FA is called a *3-2 compressor*. It is denoted by a circle covering three bits. The other operator is the half-adder, which takes two input bits in a colum and produces two outputs. The HA is denoted by a circle covering two bits.

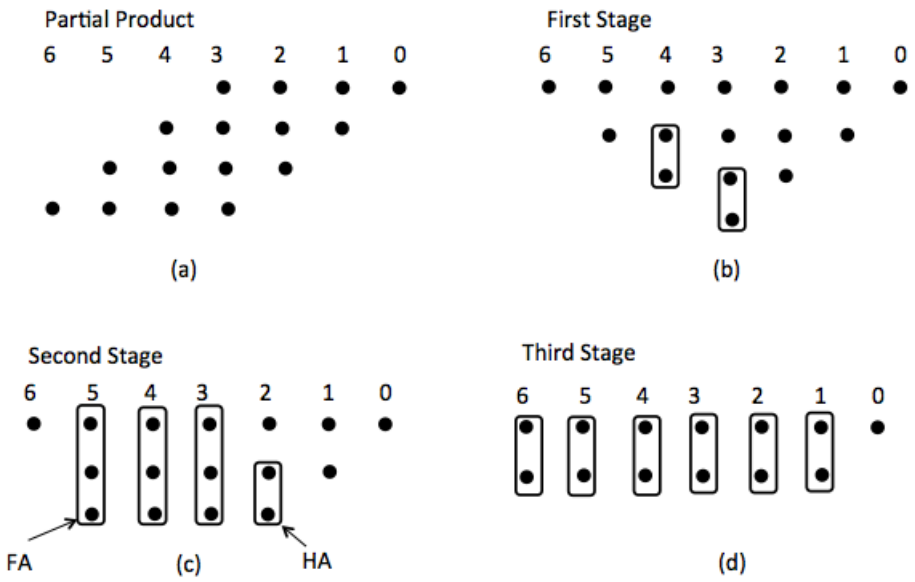

Figure 4.5: Transforming a Partial-Product tree (a) into a Wallace tree (b,c,d), using an iterative covering process.

To arrive at the minimal implementation, we iteratively cover the tree with FAs and HAs, starting from its densest part. In a first step, we introduce HAs in columns 4 and 3 fig b. The reduced tree is shown in fig c. A second round of reductions creates a tree of depth 2 d. Only three FAs and three HAs are used for the reduction process, compared with six FAs and six HAs in the carry-save multiplier of fig. The final stage consists of a simple two-input adder, for which any type of adder can be used (as discussed in the next section "Final Addition")

The presented structure is called the *Wallace tree multiplier*, and its implementation is shown in fig along with the illustration of an 8x8 multiplier with the three different stages of a multiplier. The tree muliplier realizes substantial hardware savings for larger multipliers. The propagarion delay is reduced as well. In fact, it can be shown that the propagation delay through the tree is equal to $O(\log_{3/2}(N))$. While substantially faster than carry-save structure for large multiplier word lengths, the Wallace multiplier has the disadvantage of being very irregular, which complicates the task of coming up with an efficient layout.

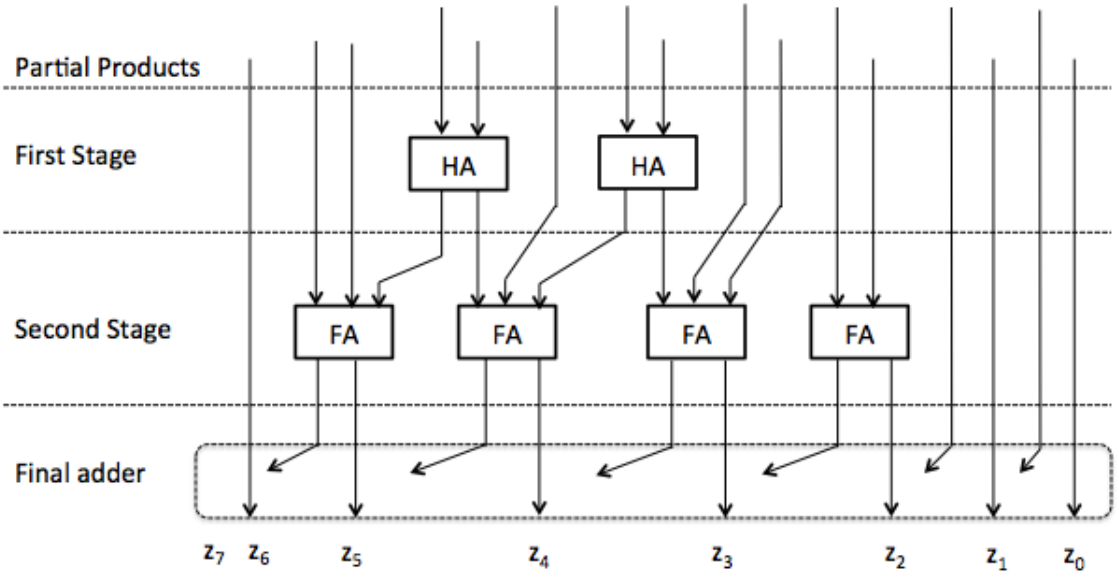

Figure 4.6: Wallace Multiplier

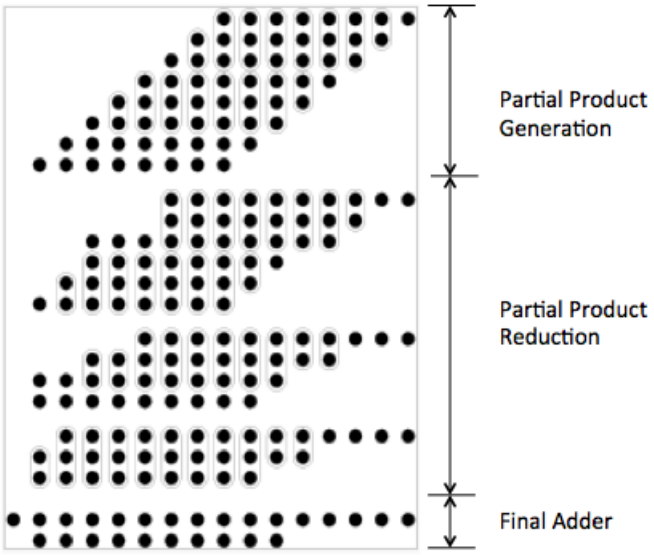

Figure 4.7: Illustration of Wallace Multiplier

There are numerous other ways to accumulate the partial-product tree. A number of compression circuits has been proposed in the literature [ref], They are all based on the concept that when full adders are used as 3:2 compressors, the number of partial products is reduced by two-thirds per multiplier stage. Hence, we can go further and device a 4:2 compressor shown in fig 4.8 or even higher order compressors.

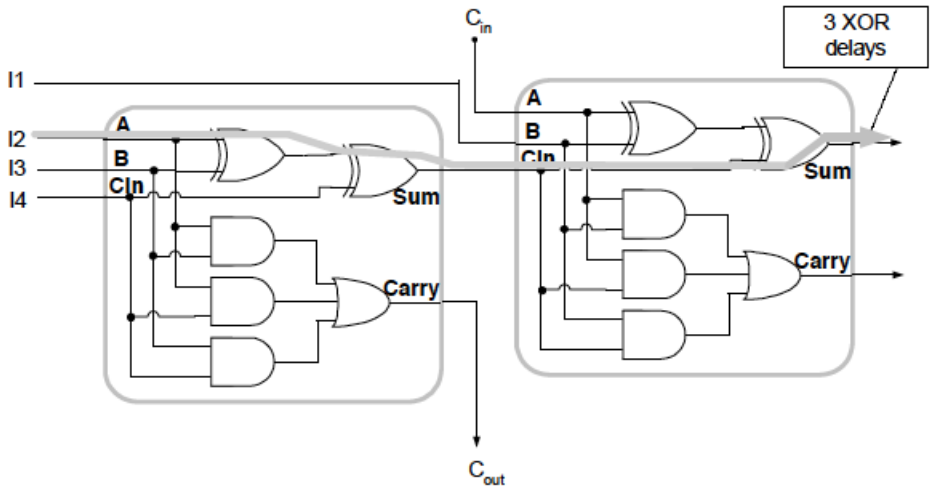

Figure 4.8: 4:2 Compressor

By using a higher order compressor like the one shown in fig 4.8 the carry save tree or the partial product stage delay reduces by one *XOR* gate delay for every two stages depicted by the highlighted path in the fig 4.8. The higher level picture is shown in fig 4.9 which is similar to fig 4.7 but with the use of 4:2 compressors besides a Half Adder and a Full Adder.

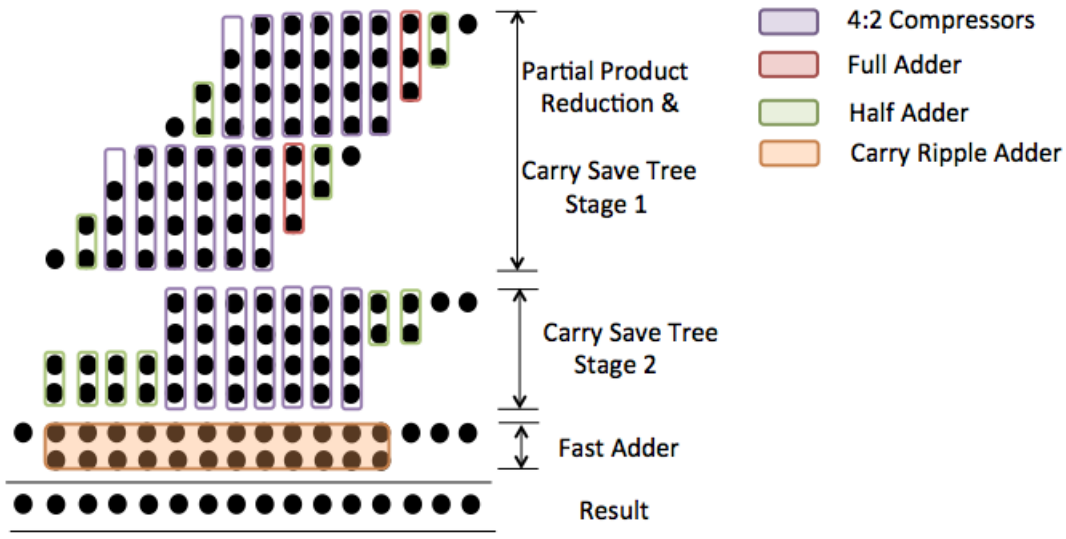

Figure 4.9: Illustration of Wallace Multiplier with 4:2 compressors

4.4 Final Addition

The final step for completing the multiplication is to combine the result in the final adder. Performance of this "vector-merging" operation is of key importance. The choice of the adder style depends on the structure of the accumulation array. A carry look-ahead adder is preferable option if all input bits to the adder arrive at the same time, as it yields the smallest possible delay. This is the case if a pipeline stage is placed right before the final addition. Pipelining is a technique frequently used in high-performance multipliers. In nonpipelined multipliers, the arrival time profile of the inputs to the final adder is quite uneven due to the varying logic depts of the multiplier tree and hence, we in this thesis use a Ripple Carry Adder with a skewed delay distribution as shown in fig 4.10 with an efficient Error Detection and Correction design. It should be intuitive that the skewed distribution of the RCA fits perfectly for Better than Worst case design (BTWC) making the multipliers performance comparable to that of the fastest Multipliers with a Carry Select adder or other hybrid adders [ref]

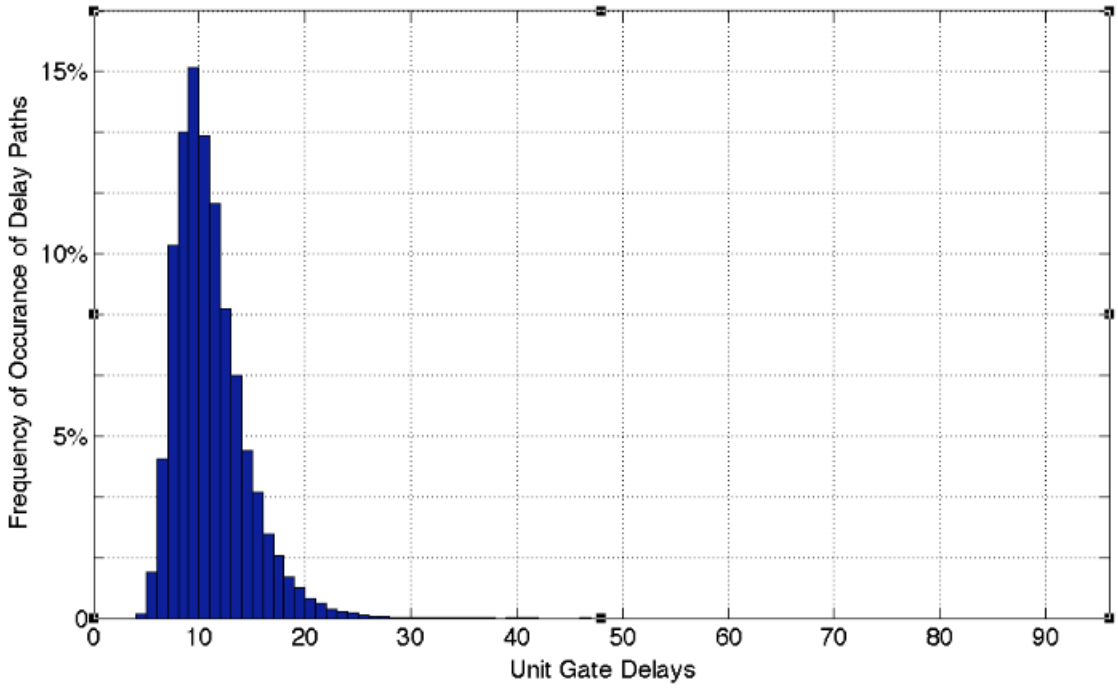

Figure 4.10: Delay Distribution of a Ripple Carry Adder

From the above discussion we can conclude that, even though the partial product reduction stage has a variable delay it is limited by the number of stages in the carry save tree and hence the delay of the carry save tree can be treated as constant delay and the final adder delay as variable delay as illustrated in the fig 4.10. So, having a Ripple Carry Adder with a variable delay in a Wallace multiplier would result in a delay distribution which is displayed by the carry save trees fixed delay. The unit delay simulation results for a Wallace Multiplier to obtain the delay distribution is illustrated in the fig 4.11 along with the RCA's distribution.

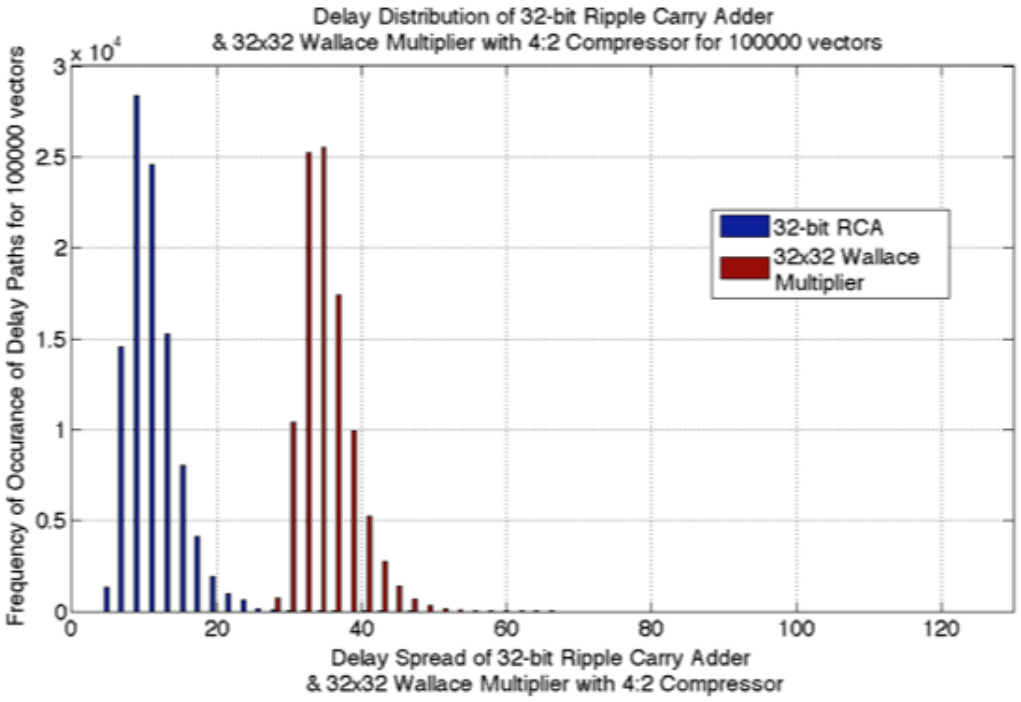

Figure 4.11: Delay Distribution of 32-bit Wallace Multiplier with final RCA distribution for 100000 vectors

Chapter 5

Proposed BTWC Design

In Chapter 2 a brief overview on Better than Worst Case designs was given and in chapter 3 and 4 several Adder and Multiplier designs were discussed. In this chapter the key features of how the design is implemented to leverage the unique characteristics of a Ripple Carry Adder are described(shown in the delay distribution in fig) besides the primary requirement of a Better than Worst Case design i.e. efficient error detection and providing multiple clock cycles during an occasional error are illustrated.

5.1 The Internal signal Hold until Completion Confirmed Scheme

The schematic in Fig illustrates pipelined operation of proposed Wallace multiplier design at a high level, where new inputs arrive and the current cycle results are captured and made available on each edge. The multiplier itself is broken up into two parts, the front-end logic that generates the partial products along with the carry save tree (these mostly display a fixed delay relatively independent of inputs), and the carry ripple adder with highly variable delay. For purposes of this initial discussion we assume that there exists a register of latches between these two parts that can hold the output signals of the carry save tree (the inputs to the ripple carry adder) for a desired duration. (In practice, as we shall see later, his capability can be more cost effectively implemented using tri-state gates that create dynamic latches.) This (low active) hold signal is activated on each active clock edge, but released (making the latches transparent again) shortly thereafter if no error is detected. *Observe that if the hold activated at the start of a clock cycle is released within a period less than the propagation delay of the first (carry save tree) part of the logic, i.e. before new logic values due to the updated inputs propagate to the latch register, the activation of the hold in each*

cycle does not in any way restrict the propagation of logic signals. Therefore it has no impact on the overall multiplication delay, beyond the extra logic and loading delays introduced by the insertion of the latches.

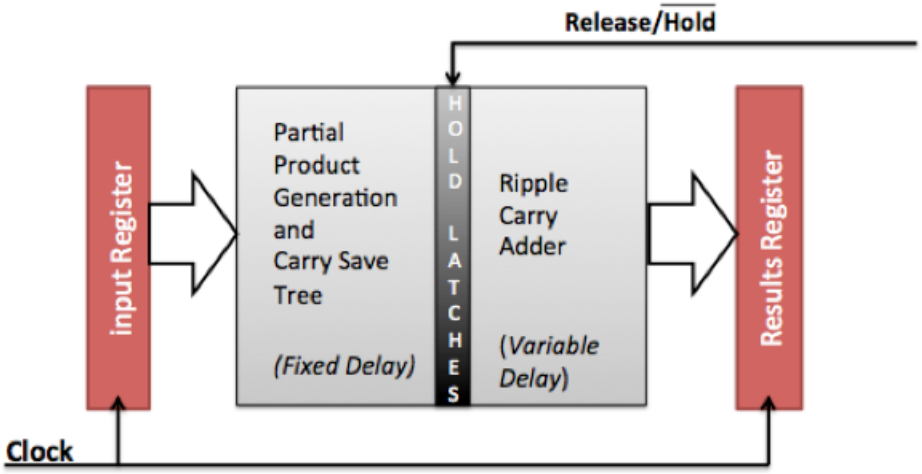

Figure 5.1: Pipelined Schemtic for Wallace Multiplier Operation

As was observed in previous chapter, because of the highly skewed completion delay distribution of the carry ripple adder, even with the system running at a clock 2-3X faster than the worst case delay, errors in the result of the multiplication due to timing violations on long paths can be expected to be extremely rare. Assume for the moment that a capability for the fast detection of such errors exists, i.e. it is possible to know quickly after the clock edge that captures a result in the results register whether or not that result is correct. This information can be used to control the hold/release signals for the latches, as shown in the timing waveforms in Figure 5.2. In case an error is detected, the hold on the latches is not released for the remainder of the cycle, and into the next cycle, until an error free result in the next cycle is confirmed. Observe that since the hold is activated on the same clock edge that updates the input registers, changes from the updated inputs are unable to immediately propagate through the carry save tree to influence the values in the latches. Thus in the hold mode the latches always hold signal values from the previous clock cycle. If not released in the event of an error, the hold latches continue to hold the values from the previous cycle at the inputs of the ripple carry adder, giving that computation an additional cycle

to complete. The correct result is then available one cycle after the incorrect result for the same multiplication was captured in the results register. This can even extend to a third (or fourth) cycle for worst-case carry propagation until an error free result is confirmed. Meanwhile, the error flag can also be used as a signal to supervisory system for appropriate handling of the input and output values in each clock cycle. Erroneous cycles must be marked as stalls in pipelined operations, with the correct result following in the next cycle. Additionally, the flow of new inputs to the multiplier must be halted corresponding to each stall.

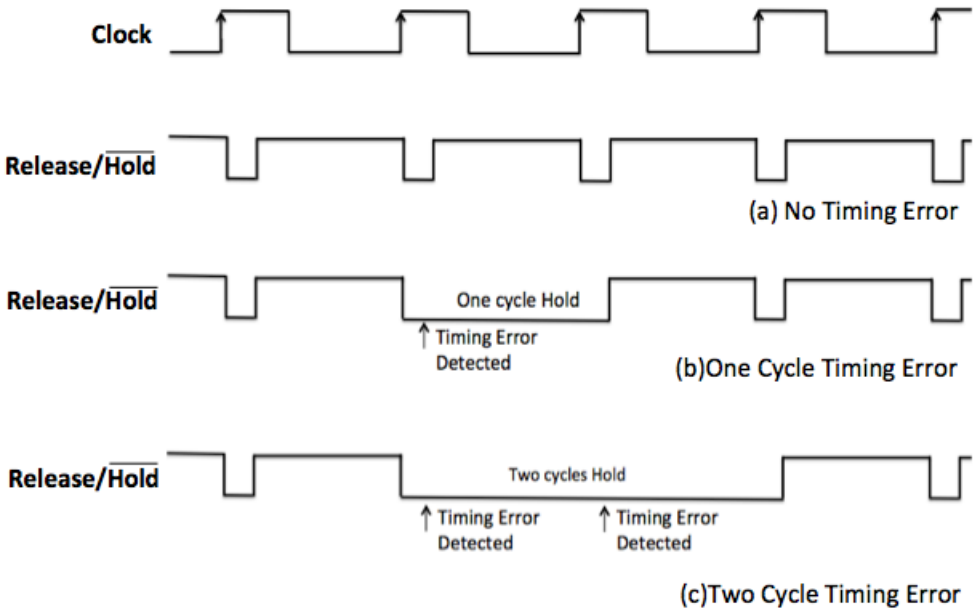

Figure 5.2: Timing Model Waveforms for Hold Until Completion

5.2 Timing Error Detection

In the discussion in the previous Section we have assumed that a timing error, caused by a longer carry propagation path in the ripple carry adder than was allowed for in setting the clock period, can be quickly detected following the capture clock edge. The key to this capability lies in an important characteristic associated with the propagating carries. Observe from Fig 5.2 that the latches go into the hold mode and retain the signal values

at the input of the ripple carry adder on the clock edge that captures a new multiplication result. This allows the ripple carry adder to continue working on the previous (just captured) result until the hold is released. In most cases, the outputs of the ripple carry adder stabilize before the clock edge, and the correct multiplication result is captured in the results register. Here the outputs of the ripple carry adder (inputs to the results register) do not change even as the adder is given additional time to operate into the next clock cycle. However, if the outputs of the ripple carry adder have not stabilized at clock time because of a long carry propagation path, then they will continue to change after the clock edge. This condition can be detected by comparing each of the adder output bits with the corresponding bit captured in the results register at the clock edge a mismatch indicates a timing error. A timing error detection circuit can therefore be constructed as shown in Figure 5.3.

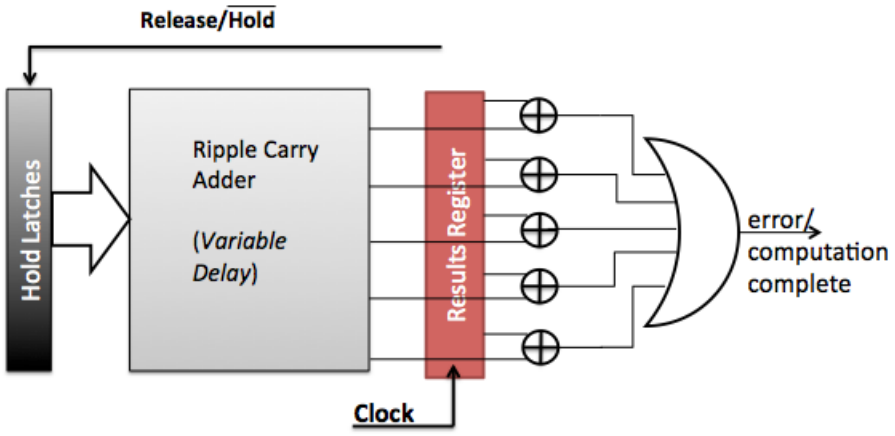

Figure 5.3: Timing Error Detection Circuitry

Recall that a propagating carry in the carry ripple adder causes a change in the sum output of every full adder stage it propagates through. This is because the sum is is formed using EXORs on three inputs, two of which are stable as the carry propagates. *Thus if final stable outputs are not captured on a clock edge, at least on ripple carry adder output can be expected to change within a full adder delay of the clock edge.* (It is impossible for the adder outputs to show no change over any full adder delay window, but then change with a larger delay.) This allows the error/computation complete indication in Figure 5.3 to be evaluated

within a few gate delays of the clock edge, achieving the important requirement of quick completion confirmation that can allow the hold in the latches to be released without any performance penalty.

5.3 Fast Comparator to monitor output completion

As mentioned in the previous section in fig 5.3 the key design feature which helps in detecting the output completion, is the ability to detect any carry which is still propagating in the RCA one cycle after the inputs are fed into the multiplier design. The fast detection is primarily due to the full adder logic which reflects any change in its input carry to the output sum. Hence, by comparing the sum bit of each Full Adder and the data captured at the end of previous clock cycle, we can flag an error saying that a carry is still alive in the RCA and that we need to continue holding the input data to the RCA beyond the previous clock cycle.

One major overhead is as the number of full adders increase the number of comparators increases and the delay to generate an error signal increases thus, impeding the current clock cycles computational data at the input of the RCA. In the proposed design the final vector merger (RCA) has 59 full adders. Besides the*XOR* gates which flags an error we need to have all the generated signals from the *XOR* gate *OR*'ed Since, the standard cell library does not support one monolithic *OR* gate (we use an *OR* gate to flag an error in case of any carry propagating in the RCA, since a change in any of the sum output is sufficient condition to flag an error) the synthesis tool might place several stages or *OR* logic (5 stages of 3-input *OR* gates in the case of our design) thus, delaying the error signal generation by the number of 3-input*OR* gates deduced by the synthesis tool. The following is the verilog code implemented as one monolithic *OR* gate.

```
\\Verilog Code for Comparator to detect Error


module monolithic_OR(a,b);


  input [58:0] a;
  output b;


  assign \#6 b = |a;
endmodule
```

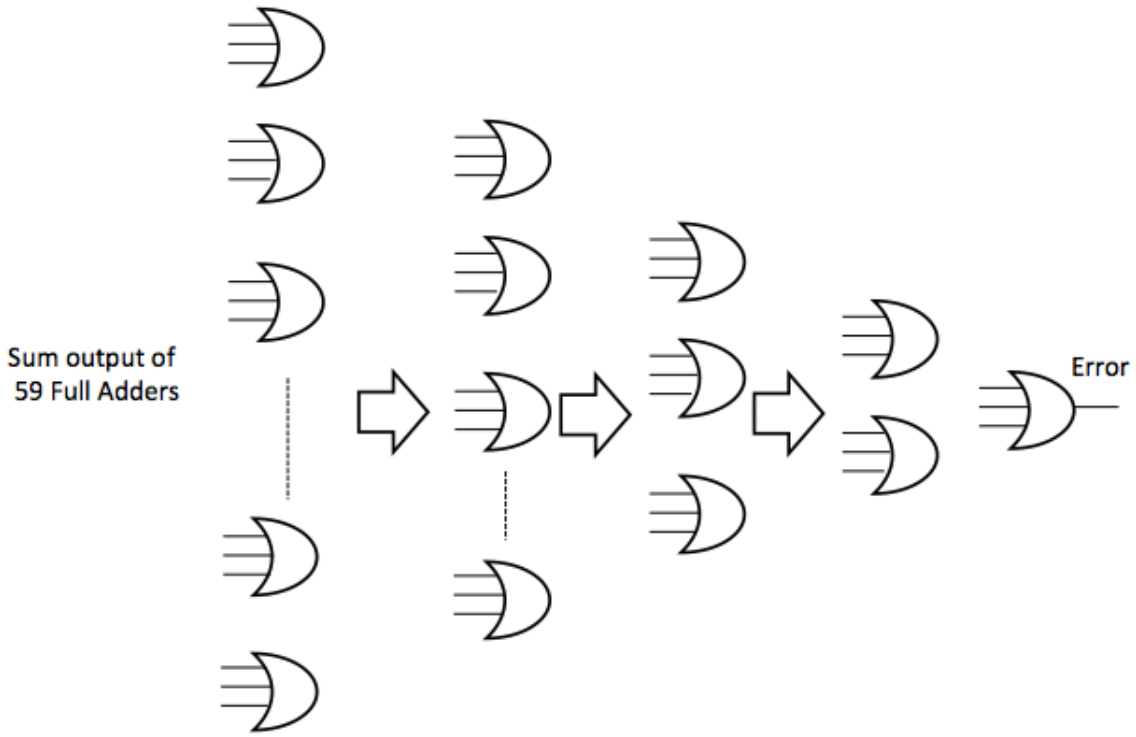

Figure 5.4: Implementation of the above verilog code by Mentor Graphics Synthesis tool

Fig 5.4 shows the comparator implementation of the monolithic *OR* gate coded in verliog. As can be seen from the figure 5.4 if each 3-input OR gate is considered to have a delay of 2-units, the total delay of the entire comparator logic will be 10 units, implying that the current clock cycles computation is held by the hold signal until the Completion detection is done. Even thought the overhead is not significant, on an average case there would be some loss in performance. To address this issue we describe some fast Comparators present in the literature.

5.3.1 Equality Comparator

Most of the high performance processors use Dynamic logic as comparators which are robust when compared to the Static CMOS desing but are sensitive to noise and hence, proper care is to be taken while designing a Dynamic logic by having fixed noise margins. One such design is shown in fig 5.5

An example of Equality Comparator [15] CMOS 4-bit equality comparator is shown in Fig. 5.5. In Fig. 5.5, when the CLK is low, Node 1 is precharged to VDD. If A $< 0 >$ and B$< 0 >$ are both high, then N1 and N2 are on and P1 and P2 are off. Thus, no current path exists during the evaluation period, and then Node 1 will be kept high. If A$< 0 >$ is high and B$< 0 >$ is low, then N1 and P2 are on. Thus, a current path is formed between Node 1 and ground through P2 and N1 during the evaluation period. Node 1 will then be pulled down.

The operation for A $< 1 >$ and B$< 1 >$, A$< 2 >$ and B$< 2 >$, and A$< 3 >$ and B$< 3 >$ is the same. In short, when any pair of A$< i >$ and B$< i >$ is not equal, a current path will be formed and Node 1 will be low. By contrast, if A$< i >$ is equal to B$< i >$ for all i, Node 1 will keep high.

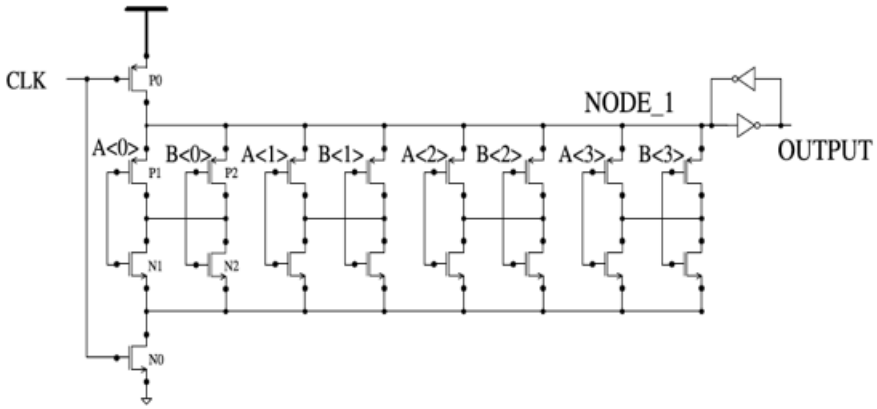

Figure 5.5: 4-bit Equality comparator

5.3.2 A Pass Logic Single Stage Comparator

The second dynamic comparator design [16], shown in Figure 5.6, avoids the use of dominostyle logic altogether.

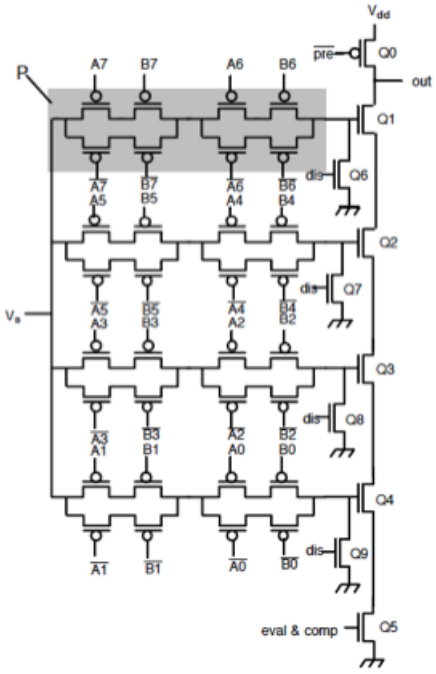

Figure 5.6: 4-bit Equality comparator

The pass transistor logic shown within the greyed box in Figure 5.6 passes a high logic level to the gate of the ntransistor Q1 when bits A7 and B7, as well as bits A6 and B6 of the comparands match. The series pulldown structure consisting of the devices Q1, Q2,

Q3 and Q4 thus conducts when all 8 bits of the comparands are equal. The output of this comparator, precharged to Vdd by Q0 is thus discharged when all bits of the comparands are equal and when the evaluate device, Q5, is on. The ntransistors Q6, Q7, Q8 and Q9 discharge any accumulated charges when partial matches occur

Hence, the OR tree structure shown in fig 5.4 for our design can be replaced with the above dynamic comparator which is both robust and less in hardware implementation, but as mentioned proper care is to be taken to avoid any noise due to coupling capacitance.

Chapter 6

Implementation of 32x32 Wallace Multiplier with the proposed BTWC Design

A 32x32 Wallace multiplier with statistical timing was implemented as described in the earlier Sections with a carry save tree built from 4:2 compressors, and a ripple carry adder for the final stage. A schematic of the design is shown in Figure 6.1.

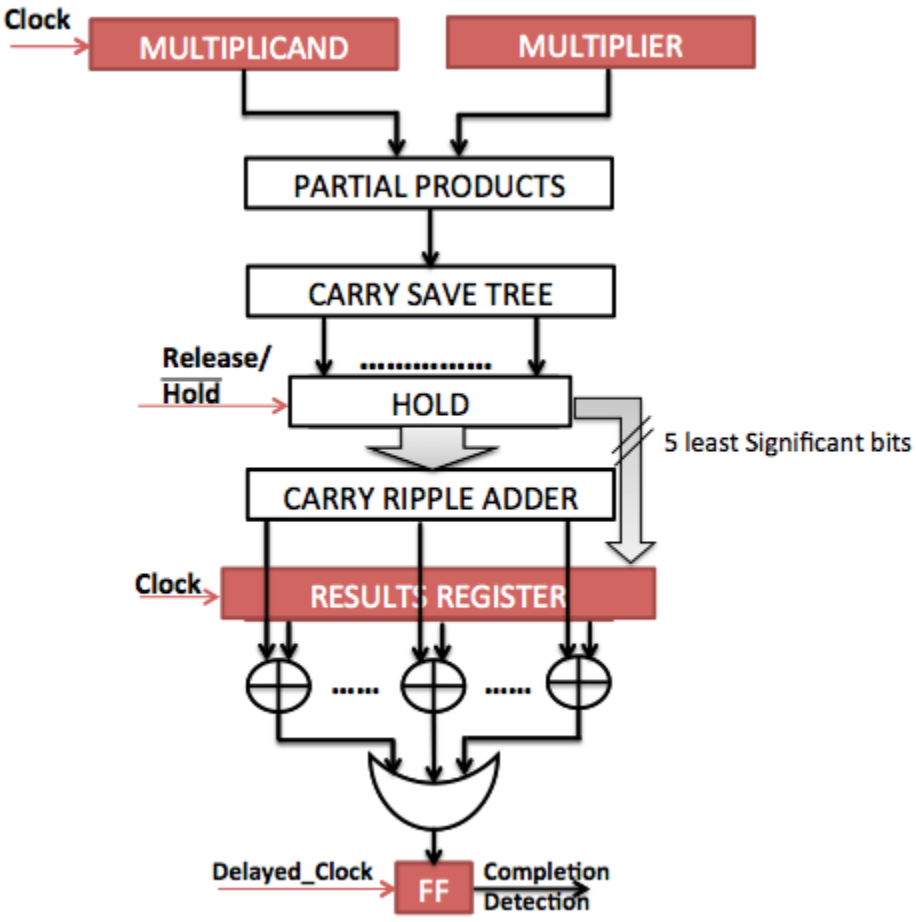

Figure 6.1: Illustration of Wallace Multiplier with the Proposed Design

The partial products are generated by performing the AND operation on the appropriate bits from the multiplicand and the multiplier, resulting in 32 partial products, each 32 bits

long, arranged as illustrated in Figure 2 for 8x8 multiplication. The carry save tree then compresses the partial products in 8 stages using 4:2 compressors to two numbers that are added by the ripple carry adder. As can be observed form the example in Figure 2, some of the least significant bits of the multiplication result are already available from the carry save compression and need not be added further. For the 32x32 multiplier, 5 bits are pre-generated in this manner, requiring a 59 bit ripple carry adder to add the remaining bits. (Note that the some of the 5 pre-generated bits least significant have very short logic paths through the carry save tree, and may need to be delayed through buffers to avoid a race condition at the hold latches.)

The hold block shown represent the latch and hold mechanism that is initiated on each clock edge, to continue to hold the inputs of the ripple carry adder until the error detection logic confirms that the correct multiplication result was captured at that clock edge. To minimize, performance and hardware overhead, this function is actually optimized in our design as a dynamic signal hold by tri-stating the driving output gates of the carry save tree.

As described in the previous section, detection whether or not the result captured on a clock edge is in error is achieved by comparing the captured result on a clock edge with the output of the ripple carry adder driving the results register. Since the inputs of the carry ripple adder are held stable past the active clock edge for all clock-cycles (until the hold is released), it can, if needed, continue working on the computation for which an initial result is captured on the clock edge. A subsequent mismatch between the ripple adder output and the captured result after the clock edge is an indication that the addition did not complete within the clock period. In our design an error signal is generated by latching this mismatch signal (from the EXOR-OR comparator circuit) using a clock appropriately delayed from the active clock edge. Since a propagating carry in the carry ripple adder causes a change in the sum output of every full adder stage it propagates through, this delayed clock need only incorporate the carry to sum output delay of a full adder, plus the comparator delay. In our design, this is about ten gate delays, allowing for the high fan in of the OR gate. Such fast

error evaluation/completion detection, allows the tri-state/hold condition at the outputs of the carry save tree to be removed well before results for the new set of partial products for the next computation are ready, thereby avoiding any performance penalty from the hold during an error free cycle.

A 32x32 Wallace multiplier with statistical timing incorporating the hold until completion detection logic described above was implemented at the RTL level and synthesized and optimized for timing using Mentor Graphics Spectrum in tsmc180nm technology. The entire design was then simulated for performance evaluation.

Chapter 7

Simulation Results and Conclusion

7.1 Simulation Results using TSMC 180nm technology

The timing simulations results shown in Table 1 were obtained using SPICE and tsmc 180nm technology files. The pessimistic worst-case delay of the multiplier without timing errors was found to be 9.5ns from the Mentor Graphics STA tool. Actual SPICE timing evaluation using know worst case inputs [5] yielded a delay of 9ns. This extremely rare worst case delay was not observed in the simulation of 10,000 random inputs in SPICE.

Table I shows the number of timing errors observed for the 10,000 random inputs when the allowed clock period for multiplication completion is reduced from the worst case 9ns to a range between 5ns and 3.5ns. It can be observed that as the clock period decreses the number of errors with a single clock period penalty increase very quickly. This is consistent with the delay distribution in Figure 4. For yet shorter clock periods, errors that require two additional clock cycle also begin to grow rapidly. Only 4 out of the 10,000 inputs triggered a timing error with the clock period set at 5ns, and understandably no case required more than two clock periods (10ns). 26 errors were triggered with the clock period set at 4ns, with one case requiring more than two cycles (greater than 8ns)to complete. Column 4 in the Table shows the effective average time required per multiplication at the corresponding single cycle clock period. This can be used to select an optimum clock period, which from the table is 3.75ns for our design. At this clock period the average multiplication time is 3.86ns. Finally, the last column shows the performance improvement over using a worst case clock of 9ns for all multiplications. This is obtained by dividing 9ns by the average multiplication time in column 3. At the optimum clock period of 3.75ns, the performance improvement is 2.36X.

Table 7.1: Spice Simulation Results with Reduced Clock Periods

Reduced Clock period ns	Error Count 10,000 random inputs		Average Clock Period ns	Performance Improvement (actual clk/avg clk)
	(one cycle penalty)	(two cycle penalty)		
5	4	0	5.002	1.79x
4.5	10	0	4.5045	1.99x
4	26	1	4.015	2.24x
3.85	113	3	3.91	2.3x
3.75	257	13	3.856	2.36x
3.5	1876	47	4.2	2.14x

Direct comparisons of the performance of this new design with other designs in the literature are difficult to make, even at the same feature sizes, because of differences in the cell libraries and technology files, particularly the threshold and supply voltages used in the reported simulations. Nevertheless, the fastest multiplier described in the literature in comparable 180nm technology reported a simulated multiplication time of 2.85ns. This design employed a Kogge-Stone fast adder which is extremely expensive in hardware, along with other enhancements. For the 60-bit final addition required by 32x32 bit multipliers, the chip area and gate count overhead of the fast adders can be 2-4X when compared to a ripple carry adder, and this hardware dominates the chip area. Our design as presented incurs a 35-40% overhead in gate counts, mostly in the comparator used for error detection. Fortunately comparators, being widely used in many applications, have been extensively improved and optimized. It should be possible to use fast comparators designs such as those in [4] to significantly reduce the overhead of the proposed approach to about 20%.

7.2 Simulation Results using TSMC 28nm technology

From the simulation results in table 7.1 it is observed that a Multiplier design with a Carry Save tree and a Ripple Carry Adder as final adder deployed with the proposed better-than-worst-case design implementation with Hold Until Completion can, on average, achieve a performance gain of 2.36X. Assuming a similar speed up for the better-than-worst-case (area optimized) design in 28nm technology, our new multiplier design can achieve a performance almost comparable to that of the design optimized for performance as shown in table 7.2. It can be observed that the multiplier design optimized for performance will dissipate 2.5X power and require almost 2.5X area compared to our area optimized better-than-worst-case design.

Table 7.2: Relative numbers for 28nm Multiplier Design (Performance and Area Optimized)

Multiplier Design	Timing (ps)	Area (units)	Power (Watts)
Optimized for Performance	1	1	1
Optimized for Area (Traditional Worst Case Design)	3.07	0.41	0.4
Optimized for Area (Better Than Worst Case Design)	1.3	0.41	0.4

7.3 Conclusion

In this thesis a novel design approach was proposed for exploiting the highly skewed statistical variation in completion delays observed in low cost Wallace multipliers implemented with ripple carry adders in the final stage. In the vast majority of cases, such multipliers complete the computation in well under half the worst case delay, making it possible to operate them with a 50-60% shorter clock period with fewer than one timing error every thousand multiplications. To support such operation, we have developed a novel internal signal hold until completion detection recovery scheme that automatically allows such an "overclocked" multiplier to steal one or more additional clock periods as needed to transparently complete the infrequent computation that needs additional time. Of course the overall system, utilizing such a multiplier, must be designed to support and work with this occasional recovery process. Fortunately, architectural level handing of exceptions through pipeline stalls and out of order instruction issue are now well understood and commonplace in processors to handle other forms of speculative execution; our design can be considered just another speculative implementation.

Achieving comparable deterministic multiplication speed requires very high speed, long word length, adders in the design which can require 3-4x more gates/area and consume significantly more power. By reducing the clock period to the point of encountering an acceptable number of timing errors, our design, like the Razor[3] approach, eliminates the wasted static (leakage) power during the frequent quiescent intervals observed in traditional designs that reliably allow for rare worst case signal propagation. This greatly reduces the average power consumed per multiplication, which is critical in battery powered mobile applications.

Bibliography

[1] Gadamsetti, B.; Singh, A.D.; , "Current Sensing Completion Detection for high speed and area efficient arithmetic," *Circuits and Systems (APCCAS), 2010 IEEE Asia Pacific Conference on* , vol., no., pp.240-243, 6-9 Dec. 2010.

[2] Borkar, S.; Karnik, T.; Vivek De; , "Design and reliability challenges in nanometer technologies," *Design Automation Conference, 2004. Proceedings. 41st* , vol., no., pp.75, 7-11 July 2004.

[3] Ernst, D.; Nam Sung Kim; Das, S.; Pant, S.; Rao, R.; Toan Pham; Ziesler, C.; Blaauw, D.; Austin, T.; Flautner, K.; Mudge, T.;"Razor: a low-power pipeline based on circuit-level timing speculation," *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, vol., no., pp. 7-18, 3-5 Dec. 2003.

[4] Blaauw, D.; Kalaiselvan, S.; Lai, K.; Wei-Hsiang Ma; Pant, S.; Tokunaga, C.; Das, S.; Bull, D.; , "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance," *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International* , vol., no., pp.400-622, 3-7 Feb. 2008.

[5] Fojtik, M.; Fick, D.; Yejoong Kim; Pinckney, N.; Harris, D.; Blaauw, D.; Sylvester, D.; "Bubble Razor: An architecture-independent approach to timing-error detection and correction, *"Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, vol., no., pp.488-490, 19-23 Feb. 2012.

[6] Ghosh, S.; Bhunia, S.; Roy, K.; , "CRISTA: A New Paradigm for Low-Power, Variation-Tolerant, and Adaptive Circuit Synthesis Using Critical Path Isolation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol.26, no.11, pp.1947-1956, Nov. 2007.

[7] Choudhury, M.; Chandra, V.; Mohanram, K.; Aitken, R.; , "TIMBER: Time borrowing and error relaying for online timing error resilience," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010* , vol., no., pp.1554-1559, 8-12 March 2010.

[8] Eriksson, H.; Larsson-Edefors, P.; Eckerbert, D.; , "Toward architecture-based test-vector generation for timing verification of fast parallel multipliers," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.14, no.4, pp.370-379, April 2006.

[9] J.M. Rabaey, *Digital Integrated Circuits.* Englewood Cliffs, NJ: Prentice-Hall, 1996.

[10] Wallace, C. S.; , *"A Suggestion for a Fast Multiplier," Electronic Computers, IEEE Transactions on* , vol.EC-13, no.1, pp.14-17, Feb. 1964.

[11] L. Dadda. Some Schemes for Parallel Multipliers. *Alta Freq.*, 34:349-356,1965.

[12] Vojin G. Oklobdzija High-Speed VLSI Arithmetic Units: Adders and Multipliers.

[13] Oklobdzija, V.G.; Villeger, D.; Liu, S.S.; , "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach ," *Computers, IEEE Transactions on* , vol.45, no.3, pp.294-306, Mar 1996.

[14] Vratonjic, M.; Zeydel, B.R.; Oklobdzija, V.G.; , "Low- and ultra low-power arithmetic units: design and comparison," *Computer Design: VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on* , vol., no., pp. 249- 252, 2-5 Oct. 2005.

[15] Ergin, O.; Ghose, K.; Kucuk, G.; Ponomarev, D.; , "A circuit-level implementation of fast, energy-efficient CMOS comparators for high-performance microprocessors," *Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on*, vol., no., pp. 118- 121, 2002.

[16] Chua-Chin Wang; Hsin-Long Wu; Chih-Feng Wu; , "A fast dynamic 64-bit comparator with small transistor count," *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva*. The 2000 IEEE International Symposium on , vol.5, no., pp.545-548 vol.5, 2000

[17] Saha, K. S., Modeling Process Variability in Scaled CMOS Technology, IEEE Design and Test of Computers, Vol. 27, Number 2, pp. 8-16, March/ April 2010.

[18] Victoria Wang, Kanak Agarwal, Sani R. Nassif, Kevin J. Nowka, Dejan Markovic , A Design Model for Random Process Variability Proceeding 2008 ISQED pp. 734-737

[19] J. Patel. CMOS process variations: A critical operation point hypothesis, 2008.

[20] R. Kumar. Stochastic processors. In NSF Workshop on Science of Power Management, 2009.

[21] S. Narayanan, G. Lyle, R. Kumar, and D. Jones. Testing the critical operating point (cop) hypothesis using fpga emulation of timing errors in over-scaled soft-processors. In In SELSE 5 Workshop - Silicon Errors in Logic - System Effects, 2009.

[22] J Sartori, R Kumar; ,"Characterizing the voltage scaling limitation s of Razor-based Designs"

[23] T. D. Burd, S. Member, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. IEEE Journal of Solid-State Circuits, 35:15711580, 2000.

[24] V. Gutnik and A. Chandrakasan, An efficient controller for variable supply-voltage low power processing. pages 158159, Jun 1996.

[25] S. Dhar, D. Maksimovic, and B. Kranzen. Closed-loop adaptive voltage scaling controller for standard-cell asics. In ISLPED '02: Proceedings of the 2002 international symposium on Low power electronics and design, pages 103107, New York, NY, USA, 2002. ACM.

[26] T. Kehl. Hardware self-tuning and circuit performance monitoring. pages 188192, Oct 1993.

[27] I. Corporation. Enhanced intel speed step technology for the intel pentium M processor, 2004.