The Dissertation Committee for Ronald S. Waters Certifies that this is the approved version of the following dissertation:

Total Delay Optimization for Column Reduction Multipliers Considering Non-Uniform Arrival Times to the Final Adder

Committee:

Earl E. Swartzlander, Jr. Supervisor

Lizy K. John

Mike Schulte

Arjang Hassibi

Adnan Aziz

Total Delay Optimization for Column Reduction Multipliers Considering Non-Uniform Arrival Times to the Final Adder

by

Ronald S. Waters, B.S.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2014

Abstract

Total Delay Optimization for Column Reduction Multipliers Considering Non-Uniform Arrival Times to the Final Adder

Ronald S. Waters, Ph.D.

The University of Texas at Austin, 2014

Supervisor: Earl E. Swartzlander, Jr.

Column Reduction Multiplier techniques provide the fastest multiplier designs and involve three steps. First, a partial product array of terms is formed by logically ANDing each bit of the multiplier with each bit of the multiplicand. Second, adders or counters are used to reduce the number of terms in each bit column to a final two. This activity is commonly described as column reduction and occurs in multiple stages. Finally, some form of carry propagate adder (CPA) is applied to the final two terms in order to sum them to produce the final product of the multiplication. Since forming the partial products, in the first step, is simply forming an array of the logical AND's of two bits, there is little opportunity for delay improvement for the first step. There has been much work done in optimizing the reduction stages for column multipliers in the second reduction step. All of the reduction approaches of the second step result in non-uniform arrival times to the input of the final carry propagate adder in the final step. The designs for carry propagate adders have been done assuming that the input bits all have the same arrival time. It is not evident that the non-uniform arrival times from the columns impacts the performance of the multiplier. A thorough analysis of the several column reduction methods and the impact of carry propagate adder designs, along with the column

reduction design step, to provide the fastest possible final results, for an array of multiplier widths has not been undertaken. This dissertation investigates the design impact of three carry propagate adders, with different performance attributes, on the final delay results for four column reduction multipliers and suggests general ways to optimize the total delay for the multipliers.

Table of Contents

Abstract iv

List of Tables ix

List of Figures x

Chapter 1. Introduction 1

Chapter 2. Previous Work 5

Townsend, *et al.* 5

Oklobdzija, *et al.* 7

Stelling and Oklobdzija 9

Oklobdzija and Villeger 12

Baran, *et al.* 13

Chapter 3. Column Reduction Methods 15

Wallace Multiplier 15

Dadda Multiplier 16

Reduced Area Multiplier 17

Modified Wallace 19

Chapter 4: Logical Effort (LE) 22

Introduction to Logical Effort 22

Determining values of Logical Effort, *g* 24

Determining value of Parasitic Delay, *p* 26

Limitations of Logical Effort 27

Use of Logical Effort in this Research 28

Chapter 5: Gate Delays versus Logical Effort Estimations 29

Chapter 6: Carry Propagate Adders 34

Ripple Carry 34

Carry Select 34
Kogge-Stone (Carry Look Ahead/Parallel Prefix) 35

Chapter 7: Scope of Work 36
Column Reduction Multipliers 37
Adders 38
Carry Propagate Adders (CPA) 41
Integer Multiplier Sizes 46
Design Constraints 46
Delay Profiler Development 49

Chapter 8: Column Reduction Delay Results 51

Chapter 9: Results Based upon Multiplier Type 58
Wallace Results 58
Dadda Results 63
Reduced Area Results 67
Modified Wallace Results 71

Chapter 10: Results Based upon Carry Propagate Adder 75
Ripple Carry Adder 75
Carry Select Adder 79
Kogge-Stone Parallel Prefix Adder 82

Chapter 11: Overall Delay Results 86
8-Bit Multipliers 86
12-Bit Multipliers 87
16-Bit Multipliers 88
24-Bit Multipliers 88
32-Bit Multipliers 89
53-Bit Multiplier 90

Chapter 12: Comparing Dynamic Power 93
8-Bit Dadda Multiplier Power Estimation 94

53-Bit Dadda Multiplier Power Estimation 96

Chapter 13: Conclusions 99

Column Reduction Method Selection 99

Minimizing Column delay by Term Selection in Dadda 99

Carry Propagate Adder Selection....101

Reducing LSB Side Delay 105

Adder Selection....106

Simplifying the Column Reduction Logical Delay Estimation 107

Dynamic Power Estimation 110

Summary 110

References....111

List of Tables

Table 1: Full Adder and Half Adder Delays for Townsend Comparison 6

Table 2: 9-Bit by 9-Bit Multiplier Comparison 21

Table 3: 53-Bit by 53-Bit Multiplier Comparison 21

Table 3: Logical Effort, *g*, values for various gates 24

Table 4: Gate Delay Counts for Eleven Gate Full Adder 29

Table 5: Gate Delay Counts for Nine gate NAND Full Adder 30

Table 6: Table of Delay Models Developed 37

Table 7: Full Adder Gate Delay Counts 39

Table 8: Half Adder Delay Summary 41

Table 9: Full Adder Delay Summary 41

Table 10: Summary of Worst Case Delays for Wallace Multipliers 63

Table 11: Summary of Worst Case Delays for Dadda Multipliers 67

Table 12: Summary of Worst Case Delays for Reduced Area Multipliers 71

Table 13: Summary of Worst Case Delays for Modified Wallace Multipliers 74

Table 14: Summary of Logical Effort Delays for all Multipliers 92

Table 15: Design Data for 8-Bit Dadda Multipliers 95

Table 16: Design Data for 53-Bit Dadda Multipliers 97

Table 17: Power Estimate Summary for 8-Bit and 53-Bit Multipliers 98

Table 18: First Reduction Stage Terms and Dadda Column Delay Improvement 100

Table 19: Table of LE/Gate delay ratio for 16-Bit Column Reductions 109

List of Figures

Figure 1: Diagram of Procedure for Column Multiplier Reduction 2

Figure 2: Dot Diagram for 8-bit by 8-bit Wallace Multiplier 3

Figure 3: Arrival Gate Delays for an 8-bit by 8-bit Wallace Multiplier 4

Figure 4: Townsend Ripple Carry Adder Delay for an 8-Bit by 8-Bit Wallace Multiplier (after [8]) 6

Figure 5: Townsend Ripple Carry Adder Delay for an 8-Bit by 8-Bit Dadda Multiplier (after [8]) 7

Figure 6: Oklobdzija Delay Profile for 16-bit Multiplier Using (4,2) Counters and a Carry Select Adder (after [9]) 8

Figure 7: Hybrid Adder using a Ripple Carry Adder, 1-level Carry Skip Adder and one Carry Select Adder (after [10]) 10

Figure 8: Regions of Delay in Column Multiplier Delay Profile (after [11]) 11

Figure 9: Delays Using Optimal Hybrid Adder for 32-Bit Multiply-Accumulate (after [13]) 13

Figure 10: Conventional Wallace 9-bit by 9-bit Reduction 16

Figure 11: Dadda 9-bit by 9-bit Reduction 17

Figure 12: Reduced Area 9-Bit by 9-Bit Reduction 18

Figure 13: Modified Wallace 9-Bit by 9-Bit Reduction 20

Figure 14: Tau values for various CMOS technology Nodes (after [24][25][26][27]) . 22

Figure 15: CMOS Inverter schematic (after [24]) 25

Figure 16: Two Input NAND Gate (after [24]) 25

Figure 17: Two Input NOR Gate (after [24]) 26

Figure 18: Full Adder Implemented with Eleven Gates 29

Figure 19: Full Adder Implemented with Nine NAND Gates 30

Figure 20: Gate Delay count comparison for 32-Bit by 32-Bit Dadda Multiplier using 9-gate and 11-gate Full Adders 31

Figure 21: Logical Effort delay comparison for 32-Bit by 32-Bit Dadda Multiplier using 9-gate NAND and 11-gate CMOS Full Adders 32

Figure 22: Logical Effort delay for 32-Bit by 32-Bit Dadda Multiplier including removal of impact of Branching Effort 32

Figure 23: Eleven Gate Full Adder 38

Figure 24: Nine Gate CMOS Full Adder 38

Figure 25: Nine Gate NAND Full Adder 38

Figure 26: Three 32-bit Dadda Multipliers using different full adders 39

Figure 27: Schematic for Five Gate, 3-2 Delay Half Adder 40

Figure 28: Schematic for Eleven Gate, 7-6-5-4 Delay Full Adder 41

Figure 29: Schematic diagram of Ripple Carry Adder 43

Figure 30: Schematic diagram of Carry Select Adder 44

Figure 31: Basic Building Block Cells for Kogge-Stone Adder 45

Figure 32: Schematic Diagram of Kogge-Stone Adder 45

Figure 33: Delay of 24-Bit by 24-Bit Dadda with Selective term grouping 48

Figure 34: 8-Bit by 8-Bit Column Reduction Multiplier Delays 51

Figure 35: 12-Bit by 12-Bit Colum Reduction Multiplier Delays 52

Figure 36: 16-Bit by 16-Bit Column Reduction Multiplier Delays 53
Figure 37: 24-Bit by 24-Bit Column Reduction Multiplier Delays 54
Figure 38: 32-Bit by 32-Bit Column Reduction Multiplier Delays 55
Figure 39: 53-Bit by 53-Bit Column Reduction Multiplier Delays 56
Figure 40: 8-Bit by 8-Bit Wallace Multiplier Results 58
Figure 41: 12-Bit by 12-Bit Wallace Multiplier Results 59
Figure 42: 12-Bit Wallace Multiplier with K-S Branching ignored 60
Figure 43: 16-Bit by 16-Bit Wallace Multiplier Results 61
Figure 44: 24-Bit by 24-Bit Wallace Multiplier Results 61
Figure 45: 32-Bit by 32-Bit Wallace Multiplier Results 62
Figure 46: 53-Bit by 53-Bit Wallace Multiplier Results 62
Figure 47: 8-Bit by 8-Bit Dadda Multiplier Results 64
Figure 48: 12-Bit by 12-Bit Dadda Multiplier Results 64
Figure 49: 16-Bit by 16-Bit Dadda Multiplier Results 65
Figure 50: 24-Bit by 24-Bit Dadda Multiplier Results 65
Figure 51: 32-Bit by 32-Bit Dadda Multiplier Results 66
Figure 52: 53-Bit by 53-Bit Dadda Multiplier Results 66
Figure 53: 8-Bit by 8-Bit Reduced Area Multiplier Results 68
Figure 54: 12-Bit by 12-Bit Reduced Area Multiplier Results 68
Figure 55: 16-Bit by 16-Bit Reduced Area Multiplier Results 69
Figure 56: 24-Bit by 24-Bit Reduced Area Multiplier Results 69
Figure 57: 32-Bit by 32-Bit Reduced Area Multiplier Results 70

Figure 58: 53-Bit by 53-Bit Reduced Area Multiplier Results 70
Figure 59: 8-Bit by 8-Bit Modified Wallace Multiplier Results 71
Figure 60: 12-Bit by 12-Bit Modified Wallace Multiplier Results 72
Figure 61: 16-Bit by 16-Bit Modified Wallace Multiplier Results 72
Figure 62: 24-Bit by 24-Bit Modified Wallace Multiplier Results 73
Figure 63: 32-Bit by 32-Bit Modified Wallace Multiplier Results 73
Figure 64: 53-Bit by 53-Bit Modified Wallace Multiplier Results 74
Figure 65: 8-Bit by 8-Bit Multiplier with Ripple Carry Final Adder 75
Figure 66: 12-Bit by 12-Bit Multiplier with Ripple Carry Final Adder 76
Figure 67: 16-Bit by 16-Bit Multiplier with Ripple Carry Final Adder 76
Figure 68: 24-Bit by 24-Bit Multiplier with Ripple Carry Final Adder 77
Figure 69: 32-Bit by 32-Bit Multiplier with Ripple Carry Final Adder 77
Figure 70: 53-Bit by 53-Bit Multiplier with Ripple Carry Final Adder 78
Figure 71: 8-Bit by 8-Bit Multiplier with Carry Select Final Adder 79
Figure 72: 12-Bit by 12-Bit Multiplier with Carry Select Final Adder 79
Figure 73: 16-Bit by 16-Bit Multiplier with Carry Select Final Adder 80
Figure 74: 24-Bit by 24-Bit Multiplier with Carry Select Final Adder 80
Figure 75: 32-Bit by 32-Bit Multiplier with Carry Select Final Adder 81
Figure 76: 53-Bit by 53-Bit Multiplier with Carry Select Final Adder 81
Figure 77: 8-Bit by 8-Bit Multiplier with Kogge-Stone Final Adder 82
Figure 78: 12-Bit by 12-Bit Multiplier with Kogge-Stone Final Adder 83
Figure 79: 16-Bit by 16-Bit Multiplier with Kogge-Stone Final Adder 83

Figure 80: 24-Bit by 24-Bit Multiplier with Kogge-Stone Final Adder 84

Figure 81: 32-Bit by 32-Bit Multiplier with Kogge-Stone Final Adder 84

Figure 82: 53-Bit by 53-Bit Multiplier with Kogge-Stone Final Adder 85

Figure 83: Logical Effort Delay for Twelve 8-Bit by 8-Bit Multipliers 86

Figure 84: Logical Effort Delay for Twelve 12-Bit by 12-Bit Multipliers 87

Figure 85: Logical Effort Delay for Twelve 16-Bit by 16-Bit Multipliers 88

Figure 86: Logical Effort Delay for Twelve 24-Bit by 24-Bit Multipliers 89

Figure 87: Logical Effort Delay for Twelve 32-Bit by 32-Bit Multipliers 90

Figure 88: Logical Effort Delay for Twelve 53-Bit by 53-Bit Multipliers 91

Figure 89: Relative Power for 8-Bit Dadda Multipliers 94

Figure 90: Relative Power for each CPA used in 8-Bit Dadda Multiplier 95

Figure 91: Relative Power for the 53-Bit Dadda Multipliers 96

Figure 92: Relative Power for each CPA used in the 53-Bit Dadda Multiplier 97

Figure 93: 12-Bit Dadda Multiplier Heat Maps 101

Figure 94: Regions in 32-Bit by 32-Bit Wallace Multiplier 103

Figure 95: 32-bit Dadda Multipliers Comparison with Kogge-Stone and Hybrid 104

Figure 96: Modified Wallace Multiplier Illustrating Reduction Issue 106

Figure 97: 32-Bit Dadda Multiplier using K-S CPA for three Different Full Adders . 107

Figure 98: Ratio of Logical Effort Delay/Gate Delay for 16-Bit Multiplier 108

Chapter 1. Introduction

Column Reduction Multipliers (CRM) are some of the faster multiplication circuits available; therefore, they are commonly used in Fast Fourier Transforms, Digital Signal Processor algorithms for convolution and filtering, graphics applications and communications applications, such as Viterbi decoders. Since multipliers tend to be in the critical path for an algorithm, their speed is important. The reason column reduction multipliers are preferred over array and other multipliers is that their delay is proportional to the logarithm of the multiplier width [1] as opposed to an array multiplier's delay which is proportional to N, the width of the multiplier.

Column reduction multipliers have three main sections. For an N-by-N multiplier, first an N^2 array of partial products is generated by performing the logical AND of each bit value of the multiplicand with each bit value of the multiplier. Second, these partial products are reduced by combining, with counters, compressors or adders, through multiple stages, until there are only two remaining rows in the final stage of the reduction. Finally, a carry propagate adder (CPA) is used to add the final two rows, producing the sum of the two rows, which results in the product of the multiplicand and the multiplier that were used to generate the N-by-N array of partial product terms [1]. Figure 1 illustrates the reduction flow for column reduction multipliers.

Column reduction multipliers have a set of rules applied during the partial product reduction stage. Four types of multipliers are explored, including: Wallace [2], Dadda [3, 4], Reduced Area [5, 6] and Modified Wallace [7].

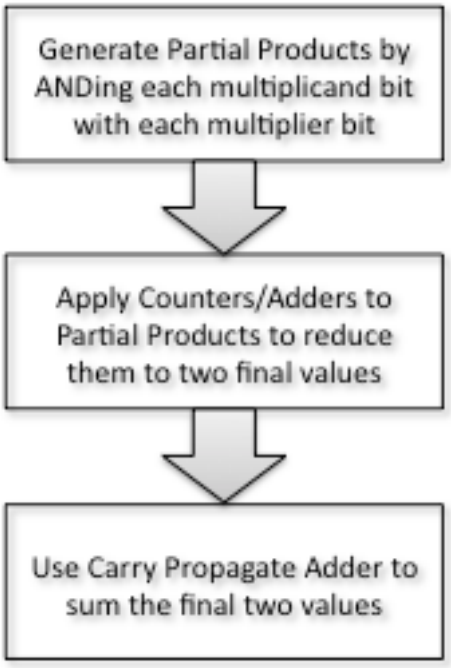

Figure 1: Diagram of Procedure for Column Multiplier Reduction

Dadda [3] popularized the "dot diagram" notation for drawing column reduction multiplier designs. The conventions that he used are still used today and are defined as:

- A Partial Product term (AND gate output)
- The outputs of a Half Adder
- The outputs of a Full Adder

A "dot diagram" for an 8-bit by 8-bit multiplier using one (Wallace) of the four reduction methods reviewed in this research is shown in Figure 2.

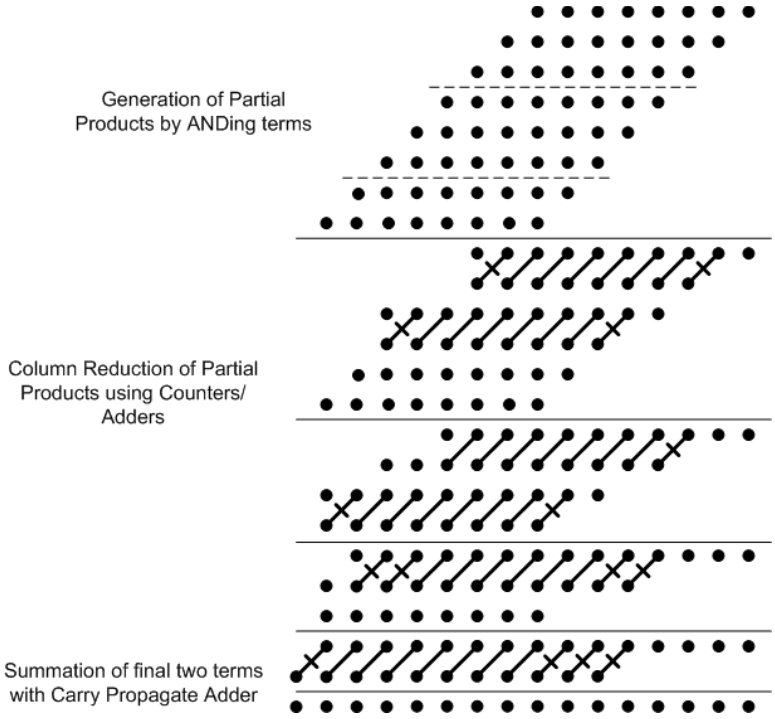

Figure 2: Dot Diagram for 8-bit by 8-bit Wallace Multiplier

Due to the nature of column reduction multipliers, the arrival times for the final two rows of bits to be summed by the final carry propagate adder (CPA) vary depending upon bit significance. For an N-by-N multiplier, the Least Significant Bits (LSB) and the Most Significant Bits (MSB) arrive well before the bits in the region of the N^{th} or $N^{th}+1$ significant bit columns. Figure 3 illustrates this non-uniform arrival profile for an 8-bit by 8-bit Wallace multiplier. The bits at the least and most significant positions arrive sooner than the bits in the center of the column reduction structure. Throughout this dissertation, the convention will be the LSB being on the right side and the MSB being on the left side of the diagrams.

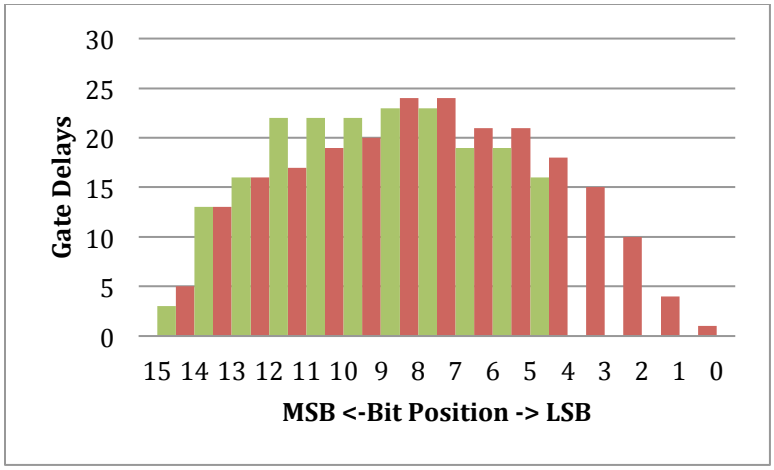

Figure 3: Arrival Gate Delays for an 8-bit by 8-bit Wallace Multiplier

Most carry propagate adder designs assume that the bits all arrive at the same time. Since that is not the case with the arrival times from column reduction, the best design to minimize the overall delay for the multiplier, of the carry propagate adder to sum the final two rows in the reduction process is not easily determined. This dissertation investigates overall multiplier delay for three designs of the carry propagate adder (ripple carry, carry select and Kogge-Stone parallel prefix) along with four column reduction methods (Wallace, Dadda, reduced area and modified Wallace) and multiplication widths (8-bit, 12-bit, 16-bit, 24-bit, 32-bit and 53-bit). Simply counting gate delays, as was done in Figure 3, will prove to be insufficient to accurately model overall multiplier delays. Therefore, rigorous use of a design modeling technique known as logical effort is used.

Chapter 2. Previous Work

Past work has dealt with multipliers of specific widths without extensive analysis of the impact of various column reduction methods. The analysis of the delay, introduced by the carry propagate adder in the final stage, typically looked at one type of adder or another or, for a single size of multiplier and reduction method, investigated the potential impact of hybridizing the carry propagate adder with a mixture of different types of adders. No analysis has been performed for a significant range of multiplier sizes and reduction methods in order to investigate the impact of multiplier width and reduction method on carry propagate adder selection.

TOWNSEND, *ET AL.*

Townsend, *et al*. analyzed both Wallace and Dadda multipliers of 4-bit and 8-bit width [8]. The carry propagate adders used were a ripple carry adder and a 4-bit carry look ahead adder. In the analysis of the reduction stages, terms were grouped in order to minimize the delay through the reductions stages and were not necessarily grouped by adjacency. This grouping technique is different than the approach taken here. In this research, adjacent terms are grouped. However, comparative analysis shows that the results from different grouping strategies do not matter much.

First, a ripple carry adder was applied to the final two reduction terms to generate the final product results. The adder characteristics used were for a nine gate full adder and a four gate half adder, with the following number of input to output gate delays:

Table 1: Full Adder and Half Adder Delays for Townsend Comparison

Path	Gate Delays
Full adder In to Sum	6
Full adder In to Cout	5
Full adder Cin to Sum	3
Full adder Cin to Cout	2
Half adder In to Sum	3
Half adder In to Cout	1

The following Figures 4 and 5 compare Townsend's results for an 8-bit by 8-bit Wallace and Dadda multiplier with a ripple carry adder. The bars represent the total delay through the column reduction stages for each bit in the multiplier. The solid line represents the final delay including the partial product generation, the reduction stage delay plus the delay through the final CPA, using a ripple carry adder.

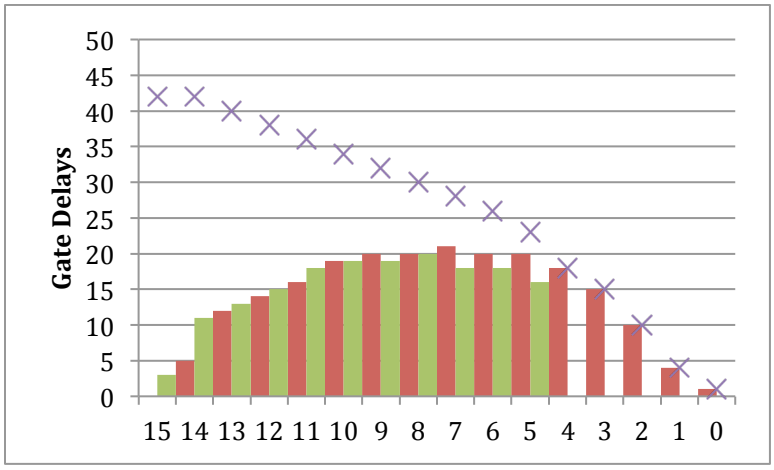

Figure 4: Townsend Ripple Carry Adder Delay for an 8-Bit by 8-Bit Wallace Multiplier (after [8])

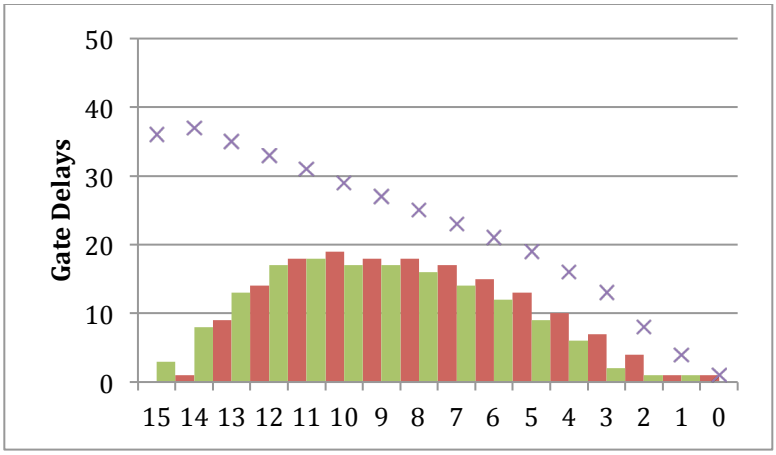

Figure 5: Townsend Ripple Carry Adder Delay for an 8-Bit by 8-Bit Dadda Multiplier (after [8])

Townsend's primary finding was that for 8-bit by 8-bit multipliers, though Dadda requires a wider final CPA, the overall delay through the multiplier, using a ripple carry adder or a carry look ahead is shorter for Dadda than for Wallace's column reduction method. This is due to the overall smaller delay through the reduction stages resulting from the Dadda reduction (19 delays maximum) versus the Wallace reduction (21 delays maximum) and the fact that the LSB to MSB slope of the delay on the LSB side of the reduction profile has a more shallow slope (delay/bit) than Wallace and as a result, a Ripple Carry Adder is just as effective as any other CPA approach for summing the final two terms in the reduction stage, and "delay build" on the LSB side still allows for a faster multiplier, even given the need for a wider CPA. Delay build is defined as the increase in delay from bit to bit as results pass from carry out of one stage into carry in of the next stage.

OKLOBDZIJA, *ET AL.*

Oklobdzija has probably been the most prolific researcher in optimizing the delay of multipliers by including analysis of the CPA. He and his research teams spent much

time in the mid to late 1990's looking at ways to reduce column reduction delays in multipliers as well as investigating ways to optimize the carry propagate adder in order to consider the non-uniform bit arrival times and minimize overall multiplier delay [9, 10, 11, 12, 13].

In [9], Oklobdzija and Villeger approached multiplier performance improvement by use of (4,2) and (9,2) counters for the column reduction stage, but more importantly, investigated a CPA scheme using a conditional sum adder and carry select adder for the design of the CPA. Using "dynamic programming optimization techniques" they determined that the optimum carry select adder based CPA for minimum delay was a carry select adder configuration of 1-2-3-1-3-4-2-3-4-8-1 for the 16-bit multiplier considered. Their analysis also capitalized upon the improved column reduction afforded by the use of (4,2) counters. Figure 6 illustrates their results.

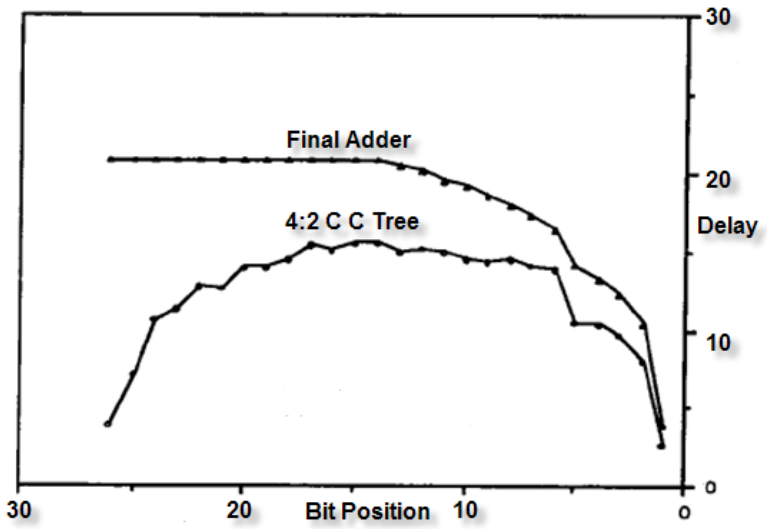

Figure 6: Oklobdzija Delay Profile for 16-bit Multiplier Using (4,2) Counters and a Carry Select Adder (after [9])

The figure has been mirrored so that the LSB is on the right as is the case for all of the analysis in this dissertation. Delays are expressed in equivalent XORs. One observation made is that the delay difference between conditional sum adder and carry select adder diminishes as the delay profile of the inputs to the CPA multiplier becomes less uniform, as is the case with the column reduction section outputs into the CPA. Further, the authors suggest that the carry select adder is slower than the conditional sum adder, but that the difference is so slight as to be offset by the relative ease of design of the carry select adder as compared to a conditional sum adder.

STELLING AND OKLOBDZIJA

Stelling and Oklobdzija [10] focused on optimizing a 32-bit by 32-bit multiplier using ripple carry adder, carry skip adder and carry look ahead blocks to form the CPA. By using hybrid adders comprised of blocks of these adders, they were able to achieve significant performance improvement. Using a hybrid ripple carry adder/1-level carry skip adder/one carry select adder based CPA; the delay profile for the 32-bit by 32-bit multiplier was developed and is shown in Figure 7. The figure is mirrored from the original diagram such that the LSB is on the right side of the figure; delays are not expressed in unit delays but in terms of delays of equivalent XORs.

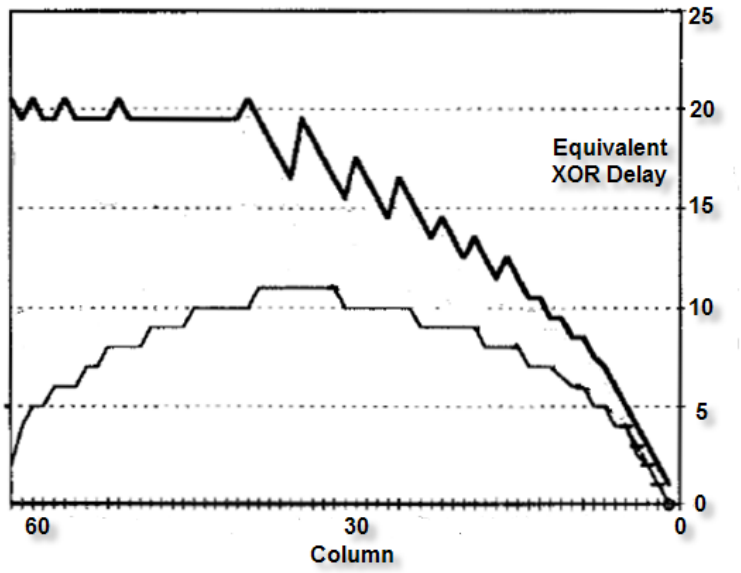

Figure 7: Hybrid Adder using a Ripple Carry Adder, 1-level Carry Skip Adder and one Carry Select Adder (after [10])

The delays presented at the bottom of the column reduction section of column multipliers, are non-uniform in arrival. Oklobdzija [11] suggests that analysis of the delay profile from column multipliers has three regions. The first region is described as region one on the LSB side of the delay profile. Region two is the central region of the delay profile where the differences between column delays are relatively small and region three is the region toward the MSB side. Figure 8 illustrates these three regions.

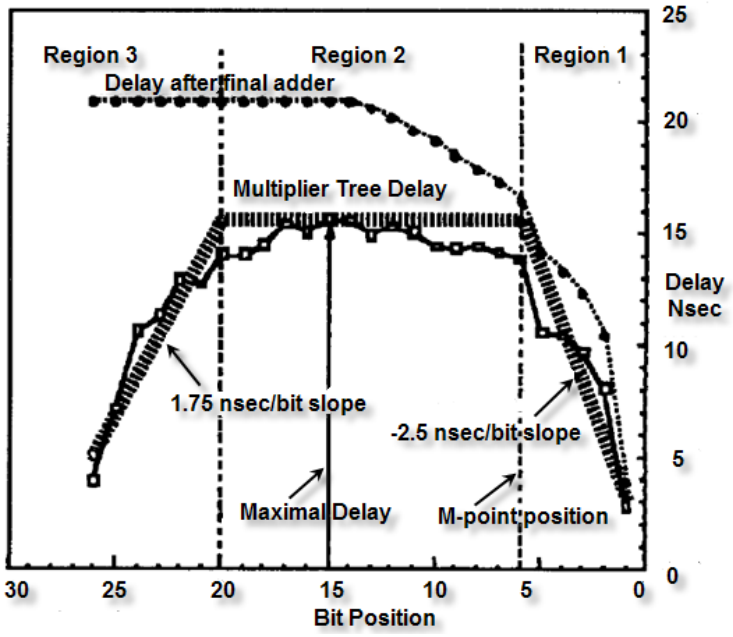

Figure 8: Regions of Delay in Column Multiplier Delay Profile (after [11])

This region nomenclature will be used in this research. Oklobdzija's primary work in [11] was based upon a 13-bit multiplier. Analysis of the CPA used ripple carry adders, conditional sum adders, carry look ahead adders, carry select adders and variable block adders. The results suggest that if the delay/bit in region one is greater than the delay through an XOR gate (Cin to Cout for a ripple carry adder), then a ripple carry adder is the most appropriate adder to use for region one. For the negative slope side (LSB), it was assumed that the delay/bit slope was less than an XOR delay and that a CPA design, other than ripple carry adder was needed. The analysis suggested that variable block adder using carry skip adders of various heuristically determined sizes would give the optimum delay results. CPAs using variable block size adders for carry select adders were also analyzed. In all analysis, the CPA was not hybridized but used either carry skip adders or carry select adders, but used variable block sizes in order to generate the minimal overall delay through the multiplier.

OKLOBDZIJA AND VILLEGER

In Oklobdzija and Villeger [12], a twelve-bit multiplier is analyzed with the conclusion that a combination of a ripple carry adder and a carry select adder would provide optimal delay. This assumes that the bit delays in region one are larger than the delay through a ripple carry adder stage, which is reasonable. The paper suggests that the inflection points between the three regions of the chart determine the length of the respective adders and that determining the lengths of the adders is an iterative process.

The analysis and design practices for generating CPAs in multipliers, is extended in Stelling and Oklobdzija [13] to Multiply-Accumulate. A 32-bit multiply-accumulate design is explored as compared to the 32-bit multiplier, only. The overall optimal delay is achieved by using a combination of a ripple carry adder, a conditional sum adder and a carry select adder for the three regions. The first region is B_0 containing bits 0 through 32 on the LSB side, B_1 covers the flat area in the center of the multiplier where the reduction delay is maximum, bits 33 through 40. Finally, B_2 is the region of the remaining bits 41 through 63. A ripple carry adder is used for B_0 while a "symmetric" conditional sum adder is used for B_1. Since the bit delays in B_2 are decreasing with bit significance, a carry select adder is used for that section. Figure 9 illustrates the results of using this hybrid CPA.

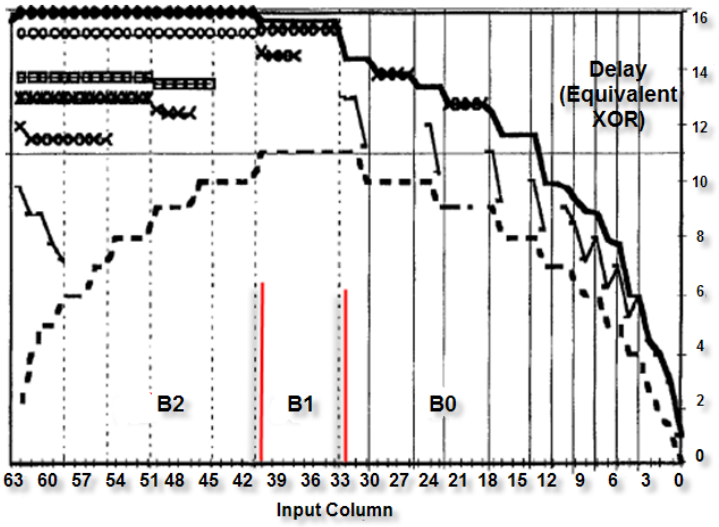

Figure 9: Delays Using Optimal Hybrid Adder for 32-Bit Multiply-Accumulate (after [13])

BARAN, *ET AL*.

In Baran, *et al*. [14], a multiplier for low power applications using Deep-CMOS is analyzed. The 16-bit by 16-bit multiplier used a hybrid carry propagate adder (CPA) with a 4-bit ripple carry adder on the LSB side, followed by a 24-bit Ling adder with a sparse-2 carry tree, ending with a 4-bit ripple carry adder.

Over time there has been much analysis performed on how to optimize multiplier performance by considering the delay through the final stage, the carry propagate adder. There has been no comprehensive analysis of the various column reduction methods along with considerations for the non-uniform arrival times for the partial product reduction stage for delays that are presented to the final CPA. This research will endeavor to develop some fundamental understandings regarding the interaction between the column reduction method and the final CPA in order to design optimum carry propagate adders in order to optimize the delay times through various multiplier designs and sizes.

Previous work has looked at individual multiplier widths, considering a particular column reduction method. Delays have either been counted in gate delays or in equivalent XOR gate delays. There has not been a comprehensive study done of multiple column reduction multiplier techniques, using logical effort as the analysis method. This research studies the delay performance of four column reduction methods, Wallace, Dadda, reduced area and modified Wallace. Further, multiplier widths of 8-bit, 12-bit, 16-bit, 24-bit, 32-bit and 53-bit are analyzed, considering a slow carry propagate adder, ripple carry adder, a moderately fast carry propagate adder, carry select adder, and a fast carry propagate adder, Kogge-Stone parallel prefix adder.

Chapter 3. Column Reduction Methods

This section reviews the strategy and approach for the design of four column reduction multipliers: Wallace, Dadda, reduced area and modified Wallace. Examples for the design of each are reviewed for subject completeness.

WALLACE MULTIPLIER

For the conventional Wallace reduction method [2], once the partial product array (of N^2 bits) is formed, adjacent rows are collected into non-overlapping groups of three. Each group of three rows is reduced by:

(1) Applying a full adder to each column that contains 3-bits or a triplet

(2) Applying a half adder to each column that contains 2-bits or a duple and

(3) Passing any single bit in a column to the next stage without processing

This reduction method is applied to each successive stage until only two rows remain. The final two rows are summed with a carry propagate adder. This process is illustrated by the 9-bit by 9-bit Wallace multiplier shown in Figure 10. Light lines show the three row groupings. The reduction is performed in four stages (each with the delay of one full adder) with a total of 50 full adders and 21 half adders being used for the reduction. The third phase will require a 13-bit wide carry propagate adder.

The use of a 9-bit by 9-bit multiplier is necessary in order to demonstrate the need for half adders in several of the other reduction methods.

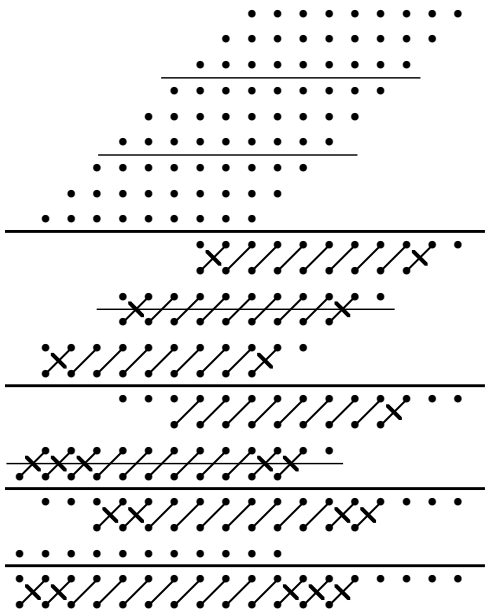

Figure 10: Conventional Wallace 9-bit by 9-bit Reduction

DADDA MULTIPLIER

In contrast to the Wallace reduction, the Dadda method [3, 4] does the least reduction necessary at each stage. To determine how many reduction stages are required, the maximum height of each stage is calculated by working back from the final stage. The final stage has a height of 2 rows. Each preceding stage height can be no larger than $\lfloor 3 \bullet \textit{successor height}/2 \rfloor$ where $\lfloor x \rfloor$ denotes the integer portion of x. This gives 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, etc. as the maximum heights for the various previous stages. The Dadda reduction then uses just enough full and half adders to achieve the limits for the stage reduction height. A 9-bit by 9-bit Dadda multiplier is shown in Figure 11. The reduction is performed in four stages (the same as with the Wallace reduction) with a total of 48 full adders and 8 half adders being used. The third phase will require a 16-bit wide carry propagate adder.

The Dadda multiplier uses 2 fewer full adders and 13 fewer half adders in the second phase reduction than the Wallace multiplier, but requires a larger carry propagate adder in the third phase as a result.

Habibi has suggested that the Dadda multiplier reduction method offers the optimum reduction in that it uses the least number of full adders [15].

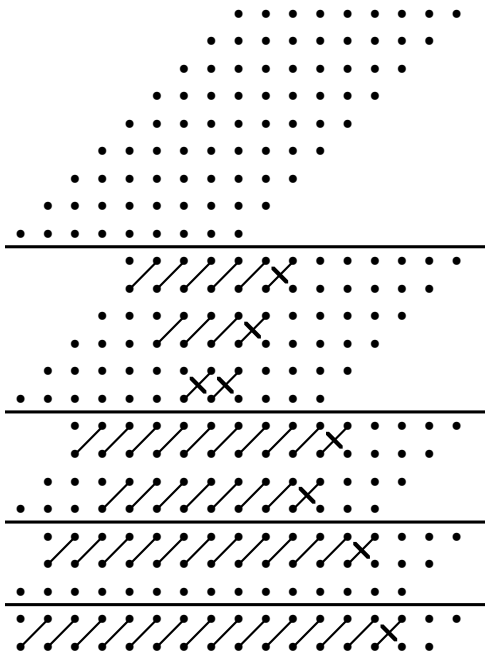

Figure 11: Dadda 9-bit by 9-bit Reduction

REDUCED AREA MULTIPLIER

In reduced area multipliers [5, 6], the objective is a multiplier design that minimizes the number of lines crossing from one reduction stage to another in order to minimize the number of latches required if the multiplier is pipelined. Also, (2,2) counters (also known as half adders) are used to move least significant partial products to the left, at each reduction stage, in order to minimize the size of the final carry propagate

adder. Additionally, (2,2) counters are used to ensure that the number of reduction stages matches that of Dadda so that the overall multiplier delay is not impacted by additional stage delays and provides equivalent multiplier delay as Wallace. Figure 12 illustrates the design of a reduced area 9-bit by 9-Bit multiplier. As can be seen, there is a (2,2) counter on the least significant bit side of each reduction stage. Also, there is a single (2,2) counter used in the first and second reduction stages in order to ensure that the total number of delay stages is not greater than Wallace or Dadda, thus ensuring an equivalent number of reduction stages and delay through the reduction section of the multiplier.

The reduced area multiplier uses 51 full adders and 12 half adders and a final phase carry propagate adder of 13-bits.

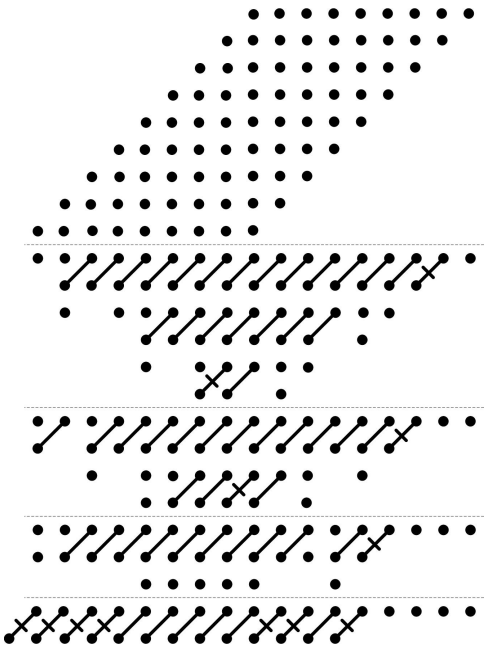

Figure 12: Reduced Area 9-Bit by 9-Bit Reduction

MODIFIED WALLACE

The modified Wallace multiplier [7] was developed with the intention of reducing the number of half adders. In column reduction, full adders (3,2) and half adders (2,2) are used to reduce the partial product terms to two single inputs that are then applied to a carry propagate adder. Full adders result in reducing terms (three inputs resulting in two outputs), while half adders do not reduce the number of partial products, but only migrate terms to more significant bits. It could be said that half adders do not do any work in reducing the complexity of the multiplier. The modified Wallace multiplier desires to minimize the use of half adders in order to improve the reduction efficiency for column reduction multipliers. The approach is different from the reduced area approach in that: 1. Since half adders do not reduce the number of partial products, use only full adders. 2. Use half adders only where they are required to keep the number of reduction stages to the number specified by Dadda for the given multiplier width. The reduction, using the modified Wallace multiplier is shown in Figure 13. The modified Wallace approach uses 52 full adders and four half adders and a final carry propagate adder size of 16-bits.

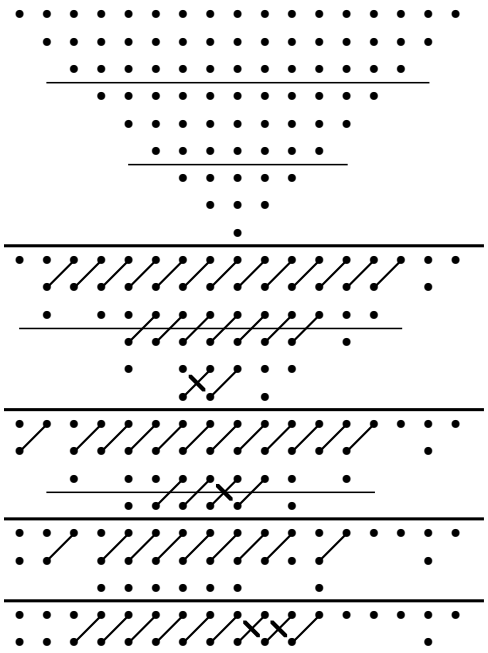

Figure 13: Modified Wallace 9-Bit by 9-Bit Reduction

As can be seen, the final CPA is the same width as Dadda, due to the second most LSB having two terms to deal with. This is the primary difference in results as compared to the reduced area approach.

Table 2 summarizes the complexity for each of the four column reduction approaches. The second column shows the number of full adders (3,2) to implement the four multipliers, the third column shows the number of half adders (2,2), the fourth column shows the number of reduction stages and the last column shows the carry propagate carry width needed for the final adder. An interesting note is that the Dadda and modified Wallace reduction methods always have the same number of adders, though different combinations of full adders and half adders.

Table 2: 9-Bit by 9-Bit Multiplier Comparison

Type (9-bit by 9-bit)	# (3,2)	# (2,2)	# Stages	CPA Length
Wallace	50	20	4	13
Dadda	48	8	4	16
Reduced Area	52	12	4	13
Modified Wallace	52	4	4	16

All four reduction methods require four reduction stages in order to reduce the partial products to two terms that are summed by the final carry propagate adder. For the Wallace and the reduced area multipliers, the final carry propagate adders are of 13 bit length while for the Dadda and the modified Wallace multipliers, the final adder is 16 bits wide. Though it should be noted that the first five bits of the carry propagate adder for modified Wallace may be implemented with half adders. The Wallace multiplier uses the most half adders even for this relatively small example.

This research extends to the 53-bit multiplier. The following table shows the complexity for each of the column reduction methods for the 53-bit multiplier case.

Table 3: 53-Bit by 53-Bit Multiplier Comparison

Type (53-bit by 53-bit)	# (3,2)	# (2,2)	# Stages	CPA Length
Wallace	2606	301	9	96
Dadda	2600	52	9	105
Reduced Area	2610	48	9	96
Modified Wallace	2610	42	9	105

For the 53-bit multipliers, each of the column reduction methods use essentially the same number of full adders and take the same number of reduction stages (9). In the case of the Wallace multipliers, it uses significantly more half adders. Since the multipliers each have the same number of stages and essentially the same number of full adders, the area required to lay out the multipliers will approximately be the same. The Wallace multiplier may take up slightly more area due to the higher use of half adders.

Chapter 4: Logical Effort (LE)

Introduction to Logical Effort

Logical effort is a relatively straightforward and simple method to calculate delays through CMOS gates and circuits. It is a reasonably simple technique that is more exact than simply gate delay counting, but not as exact as doing a design layout, back annotating parasitic values and performing a detailed SPICE circuit simulation.

The first step in logical effort is to determine the, time based, unit delay **τ** which is, for a given process, the delay through an inverter driving an identically sized inverter and is approximately 3RC. For various CMOS processes, **τ** is shown on the following figure [24][25][26][27].

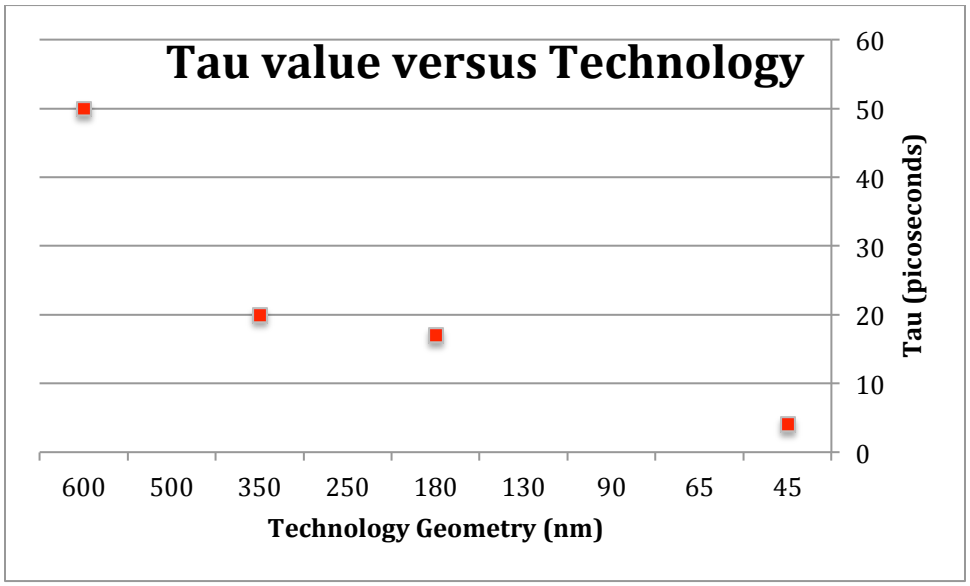

Figure 14: Tau values for various CMOS technology Nodes (after [24][25][26][27])

The absolute delay, d_{abs} through a gate or circuit, using logical effort, is the logical effort delay, d, times **τ.**

$$d_{abs} = d\boldsymbol{\tau}$$

Logical effort models delay in a logic gate with two elements, parasitic delay, p, and effort delay, f. The parasitic delay is dependent upon the gates structure as compared to an inverter. The effort delay, or stage effort, is a function of the load on the gate's output. The overall delay of a gate is the sum of the stage effort and the parasitic delay and is expressed as:

$$d = f + p$$

The effort delay can be further decomposed into logical effort, g, electrical effort, h, and branching effort, b. The effort delay then may be expressed as the product of the various effort terms as:

$$f = gbh$$

The logical effort, g, represents the gate topology to produce current as compared to an inverter. The electrical effort, h, is simply an assessment of the gate's electrical environment and is stated as the ratio of output capacitance (input capacitance to the subsequent node) to the input capacitance for a circuit being analyzed.

$$h = C_{out}/C_{in}$$

Obviously, if the input capacitance and the output capacitance (input capacitance that the final output stage of the circuit is driving) are the same, then the electrical effort, h, for the analysis is 1.

The final component in logical effort analysis is the branching effort, b. The branching effort looks at a given node's "in path" effort and the "off path" effort. The calculation for b for a given node is:

$$b = \frac{C_{in\,path} + C_{off\,path}}{C_{in\,path}}$$

Analysis typically looks at the total device width that is being driven, both in path and off path, divided by the in path device width.

If there is no fan-out or off path gates in the analyzed circuit path, then the branching effort, b, is 1.

DETERMINING VALUES OF LOGICAL EFFORT, *G*

Assuming pull-up transistor width versus pull-down transistor width and device gains produce an n-channel with twice the strength of a p-channel, the following tables illustrates the various values of logical effort, g.

Table 3: Logical Effort, g, values for various gates

	Number of Inputs		
Gate Type	**1**	**2**	**n**
Inverter	1		
NAND		4/3	(n+2)/3
NOR		5/3	(2n+1)/3
Multiplexer		2	2
XOR (Parity)		4	

The following figure shows the construction of an inverter using one p-channel and one n-channel. The numbers represent the device widths for each transistor. The ratio of 2:1 is usually the case due to the mobility or gain of the respective devices and gives equivalent pull-up and pull-down delays.

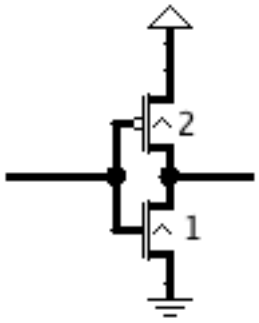

Figure 15: CMOS Inverter schematic (after [24])

Logical effort for gates are normalized to the inverter, so the logical effort, *g*, for this inverter is 3/3, or 1.

A two input NAND gate is comprised of four transistors configured as in the following figure.

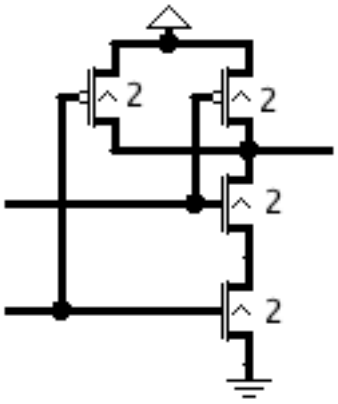

Figure 16: Two Input NAND Gate (after [24])

Note that the stack of n-channels gives an effective drive strength for the pull-down devices of 1. The logical effort, *g*, for the two input NAND gate is the total device widths seen by an input, divided by 3 to normalize to that of an inverter or 4/3.

A two input NOR gate is comprised of four transistors configured as in the following figure.

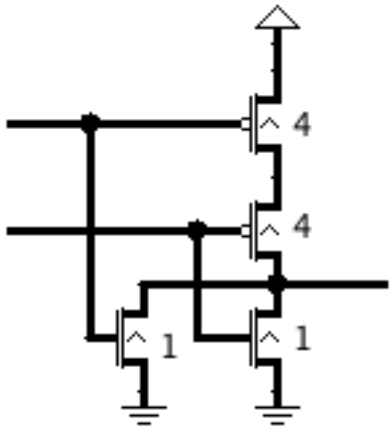

Figure 17: Two Input NOR Gate (after [24])

Note that the stack of p-channels gives an effective drive strength of the pull-up devices of 2. The logical effort, g, for the two input NOR gate is the total device widths, seen by an input, divided by 3 to normalize to that of an inverter or 5/3.

DETERMINING VALUE OF PARASITIC DELAY, *P*

Assuming simple layout styles, the parasitic delay of an inverter, p, is defined as 1. The parasitic delay is a model of the overhead delay due to the source and drain region capacitances of the transistors of the gate that drive the gate's output. This model of parasitic capacitance does not consider the capacitances of nodes that are between devices in series such as the pull-downs of a NAND gate or the pull-ups of a NOR gate. For NAND and NOR gates, the model for the parasitic capacitance of a gate is the parasitic capacitance of an inverter, p_{inv}, times the number of inputs, n, for the NAND or NOR gate. So an n input NAND or NOR gate has a parasitic delay of np_{inv}.

The final value that is required in order to perform a logical effort delay calculation is simply the count of the number of gates in an analyzed circuits path, N.

On a macro level, the overall values for a circuit path for logical effort, G, electrical effort, H branching effort, B, and parasitic delay, P, are expressed as either the product or sums of the respective individual efforts for the gates that make up the path being analyzed.

$$G = \Pi g_i$$

$$H = \Pi h_i$$

$$B = \Pi b_i$$

$$P = \Sigma p_i$$

Given these equations, the path effort, F, is defined as

$$F = GHB$$

The path overall delay, $\widehat{D}$, is defined as

$$\widehat{D} = NF^{1/N} + P$$

This delay is in gate delay units and is multiplied by the technology value, τ, to give the delay in time value.

LIMITATIONS OF LOGICAL EFFORT

The main limitation of logical effort is that it is difficult to model the impact of interconnect in a design. Usually, with logical effort, the design has not even been laid out. Interconnect, if an attempt is made to model it, impacts the branching effort as it adds more capacitance "off path" that must be considered by the model. Burgess in [28] suggests that the impact of capacitance loading of interconnect for a full adder is approximately the same as the input capacitance of a simple CMOS inverter.

Since logical effort is a simple RC based model, it does not consider the impact of rise or fall times. In well designed circuits, rise and fall time are relatively equal through out the design as are the effort delays.

Logical effort is typically used to design a path to minimize the delay through the path, but it does not lend any ability to minimize area or power with a set delay.

For complex circuits with complicated branch structures with different parasitic delays or gate delays in each branch, iterative analysis must be performed. Fortunately for multiplier designs, the branching considerations are within the full adders and can be modeled there. Connections between adders in the column reduction stages are point to point and do not involve branches. Therefore, branching effort need only be considered within each of the adders and not as the outputs transcend column reduction levels.

USE OF LOGICAL EFFORT IN THIS RESEARCH

Using the logical effort equations, spreadsheet based models were developed for each of the multiplier sizes and column reduction methods. Individual sheet tabs were created, based upon the multiplier design for each of the effort values, g, b, p and number of the gate count through each path in the multiplier structure. The electrical effort, h, was set to 1 since it is assumed that the multiplier fan-out from input to the partial product NAND gates to the input driven by the final carry propagate adder output is 1. Logical effort models were created for each of the carry propagate adders that were analyzed and applied to the outputs of the column reduction multiplier models.

Chapter 5: Gate Delays versus Logical Effort Estimations

Much of the prior work has looked at the gate delay count through the multiplier in order to determine relative figures of merit for various designs. Counting gate delays does not necessarily provide insight into the fastest multiplier design.

Two full adder designs are considered and the design of 32-bit by 32-bit Dadda multipliers are developed to compare the gate delay count versus the delay that a logical effort model for the column reduction multipliers would suggest. One full adder uses eleven total gates while the second, implemented with NAND gates, uses only nine gates. The eleven gate implementation is illustrated in Figure 18.

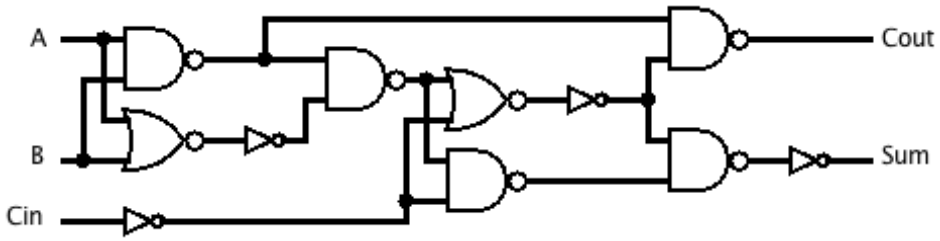

Figure 18: Full Adder Implemented with Eleven Gates

The eleven gate full adder has the following worst case gate delay counts through its various paths:

Table 4: Gate Delay Counts for Eleven Gate Full Adder

Input A/B to Cout	6
Input A/B to Sum	7
Cin to Cout	4
Cin to Sum	5

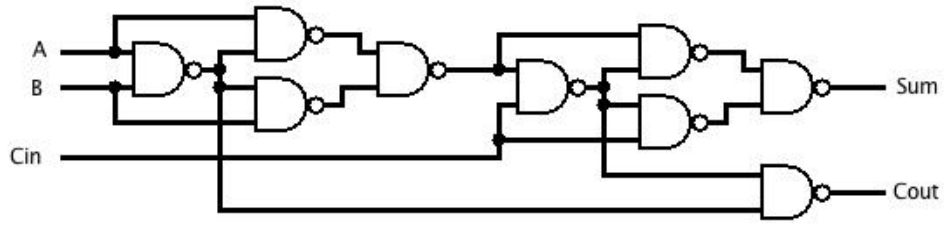

Figure 19: Full Adder Implemented with Nine NAND Gates

The nine gate full adder has the following worst case gate delay counts through its various paths:

Table 5: Gate Delay Counts for Nine gate NAND Full Adder

Input A/B to Cout	5
Input A/B to Sum	6
Cin to Cout	2
Cin to Sum	3

For the 9-gate NAND full adder, note that four of the six nodes from input to Sum and four of five nodes from input to carry out have branching effort considerations, of which two have three-way branches. This will have implications when multiplier delay is analyzed.

Applying the gate delay counts for these two full adder designs yields the following maximum delay profile through the Dadda multiplier column reduction for a 32-bit by 32-bit Dadda multiplier. The analysis suggests that the 9-gate NAND full adder implementation would be the fastest of the two designs.

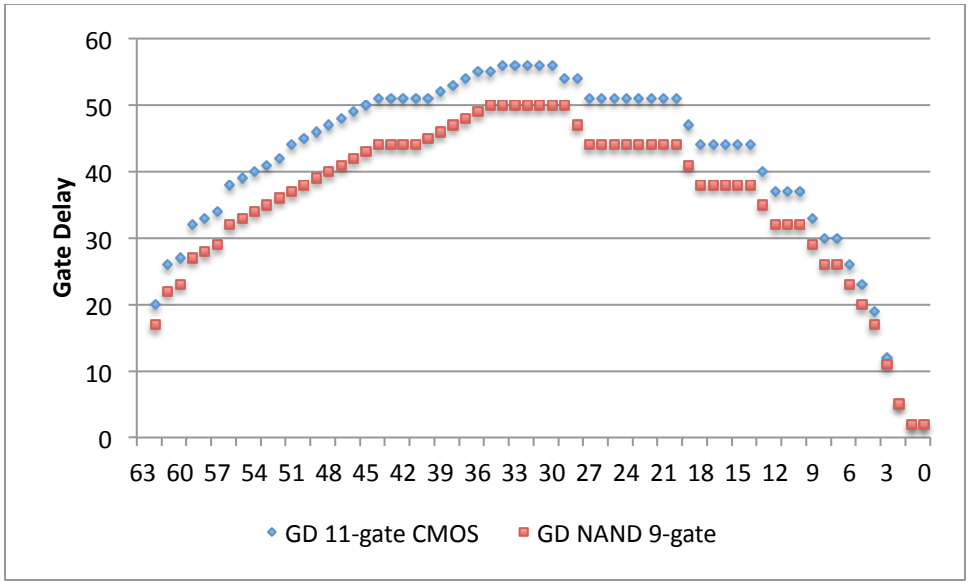

Figure 20: Gate Delay count comparison for 32-Bit by 32-Bit Dadda Multiplier using 9-gate and 11-gate Full Adders

Applying the principles of logical effort to the design yields entirely different results as is seen in the following figure. The 11-gate full adder implementation is faster. This is due to the extensive fan-outs of the NAND based design that contribute significant circuit performance impact in the logical effort analysis due to the branching effort effects. Figure 22 illustrates what the delay would be after removing the branching effort component in the logical effort calculations of the 9-gate full adder.

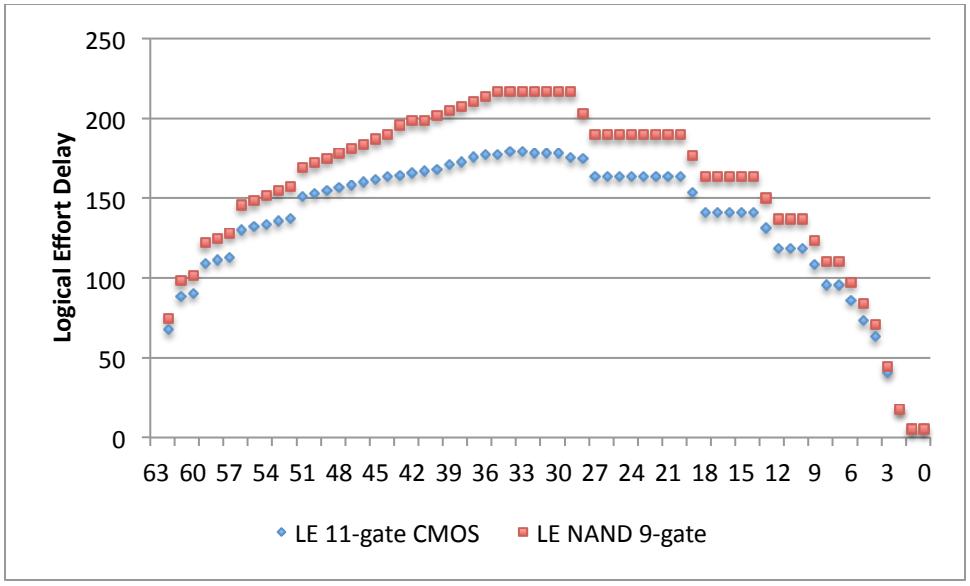

Figure 21: Logical Effort delay comparison for 32-Bit by 32-Bit Dadda Multiplier using 9-gate NAND and 11-gate CMOS Full Adders

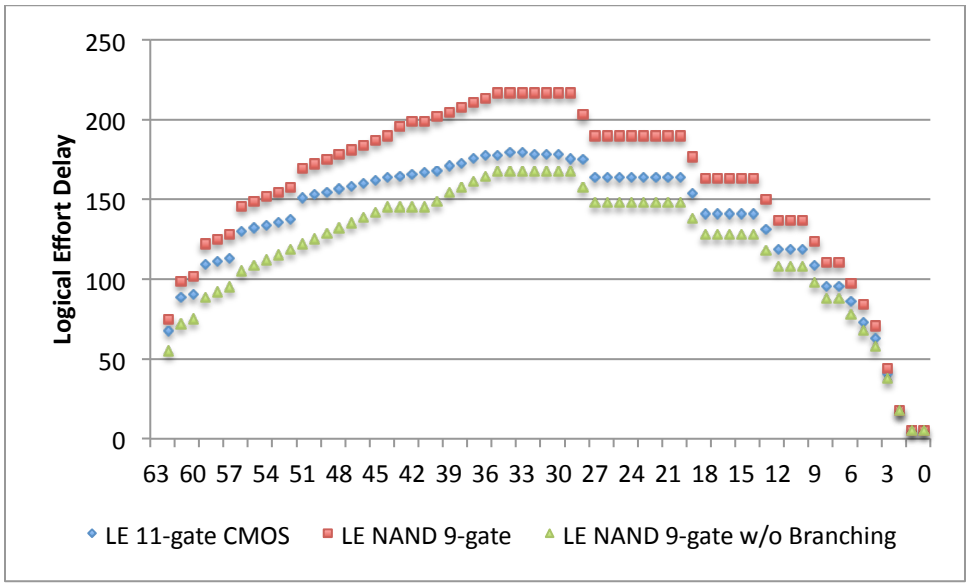

Figure 22: Logical Effort delay for 32-Bit by 32-Bit Dadda Multiplier including removal of impact of Branching Effort

Careful consideration to more than just gate count delays from input to output is required in the selection of the adder primitives to be used in a column reduction multiplier. The adder with the least gate delays, because of the impact of branching effort or fanout, may introduce more actual delay in the circuit path.

Chapter 6: Carry Propagate Adders

In order to set a reasonable boundary for this research, a finite number of carry propagate adders were selected for the analysis. Representative slow, medium and fast carry propagate adders were selected and modeled. Those adders are: ripple carry [29], carry select [16] and Kogge-Stone [20] parallel prefix adders. Other carry propagate adders such as carry look ahead [17], carry skip [18], and conditional sum [19] are not studied as the range of performance between ripple carry and parallel prefix covers the performance domain for these other carry propagate adders.

RIPPLE CARRY

The simplest carry propagate adder is the ripple carry adder [29]. It is simply a cascade of full adders where the carry out of a bit is fed into the carry in of the next most significant bit. Circuit wise, it is the simplest design, but performance wise, it is not the fastest. However, since column multipliers have an arrival time profile that increases from the less significant bit to near the center column, ripple carry adders may be sufficient for the less significant bit side of the multiplier. If the delay through the ripple carry is faster than the delay down the columns of the column multiplier, then a ripple carry adder is the best choice for that portion of the carry propagate adder. That is, indeed the case for the multipliers designed, as will be illustrated. The design of a ripple carry adder is discussed later in this work.

CARRY SELECT

Carry select adders [16] break the carry propagate adder into blocks of bit width and calculates the expected output of each block considering if a carry in occurs or not. This doubles the amount of hardware required as well as adds N+1 2:1 multiplexors to

the design for each block, where N is the width of a block. The specific design of a carry select adder used in this research is discussed in Chapter 7.

KOGGE-STONE (CARRY LOOK AHEAD/PARALLEL PREFIX)

There are several implementations of the parallel prefix carry propagate adder. One of the fastest adders is the Kogge-Stone adder [30]. Parallel prefix adders generate the carry propagate and generate values for each bit position in parallel.

Chapter 7: Scope of Work

There has been limited broad analysis performed to understand the column multiplier and CPA interaction, given that the column multiplier presents non-uniform arrival times to the input of the carry propagate adder. This research performed an extensive analysis of four types of column multipliers and the overall delay performance achievable using a multitude of carry propagate adders.

Column reduction designs were done using Wallace, Dadda, reduced area and modified Wallace reduction methods for 8-bit, 12-bit, 16-bit, 24-Bit, 32-bit and 53-bit multipliers. These reduction delay profiles were analyzed with three carry propagate adders.

The following table illustrates the number of delay models developed, and supporting work, considering the six multiplier sizes, the three carry propagate adders and the four types of column reduction methods.

Table 6: Table of Delay Models Developed

TASKS						
Design arrays	**8by8**	**12by12**	**16by16**	**24by24**	**32by32**	**53by53**
Wallace array	✔	✔	✔	✔	✔	✔
Modified Wallace array	✔	✔	✔	✔	✔	✔
Dadda array	✔	✔	✔	✔	✔	✔
Reduced Area array	✔	✔	✔	✔	✔	✔
Populate LE values	**8by8**	**12by12**	**16by16**	**24by24**	**32by32**	**53by53**
Wallace array	✔	✔	✔	✔	✔	✔
Modified Wallace array	✔	✔	✔	✔	✔	✔
Dadda array	✔	✔	✔	✔	✔	✔
Reduced Area array	✔	✔	✔	✔	✔	✔
Misc tasks	**8by8**	**12by12**	**16by16**	**24by24**	**32by32**	**53by53**
Dadda Optimization (r1)	✔	✔	✔	✔	✔	✔
Add LE values to 9-gate	✔	✔	✔	✔	✔	✔
Fix 11 gate LE B values	✔	✔	✔	✔	✔	✔
Ripple Carry	**8by8**	**12by12**	**16by16**	**24by24**	**32by32**	**53by53**
Wallace	✔	✔	✔	✔	✔	✔
Modified Wallace	✔	✔	✔	✔	✔	✔
Dadda	✔	✔	✔	✔	✔	✔
Reduced Area	✔	✔	✔	✔	✔	✔
Carry Select (by4) Type 2	**8by8**	**12by12**	**16by16**	**24by24**	**32by32**	**53by53**
Wallace	✔	✔	✔	✔	✔	✔
Modified Wallace	✔	✔	✔	✔	✔	✔
Dadda	✔	✔	✔	✔	✔	✔
Reduced Area	✔	✔	✔	✔	✔	✔
Carry Select (by4)	**8by8**	**12by12**	**16by16**	**24by24**	**32by32**	**53by53**
Wallace	✔	✔	✔	✔	✔	✔
Modified Wallace	✔	✔	✔	✔	✔	✔
Dadda	✔	✔	✔	✔	✔	✔
Reduced Area	✔	✔	✔	✔	✔	✔
Parallel Prefix (Kogge Stone)	**8by8**	**12by12**	**16by16**	**24by24**	**32by32**	**53by53**
Wallace	✔	✔	✔	✔	✔	✔
Modified Wallace	✔	✔	✔	✔	✔	✔
Dadda	✔	✔	✔	✔	✔	✔
Reduced Area	✔	✔	✔	✔	✔	✔

COLUMN REDUCTION MULTIPLIERS

Four types of column reduction multipliers with different reduction strategies were explored, including:

1. Wallace [2]
2. Dadda [3,4]
3. Reduced Area [5, 6]
4. Modified Wallace [7]

ADDERS

Many different full adder designs could be considered. A Dadda 32-bit by 32-bit multiplier was implemented with three full adder designs. These full adders were one using eleven gates and two with nine gate implementations.

The three designs for each of the full adders are shown in the following three figures.

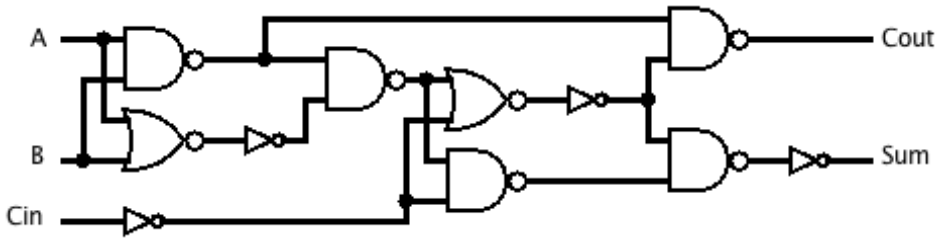

Figure 23: Eleven Gate Full Adder

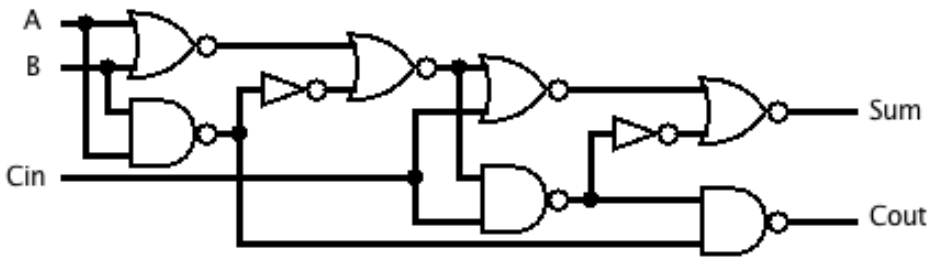

Figure 24: Nine Gate CMOS Full Adder

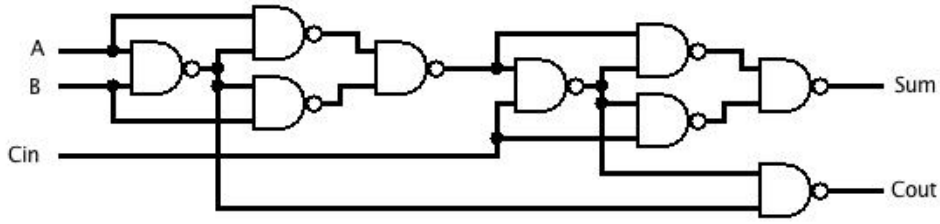

Figure 25: Nine Gate NAND Full Adder

The gate delay counts for the three full adders, from inputs to outputs, are shown in the following Table.

Table 7: Full Adder Gate Delay Counts

	11-Gate	9-Gate CMOS	9-Gate NAND	
FAAS_GD	7.0	6.0	6.0	Full Adder Gate delays A to Sum
FAAC_GD	6.0	5.0	5.0	Full Adder Gates delays A to Cout
FACS_GD	5.0	3.0	3.0	Full Adder Gate Delays Cin to Sum
FACC_GD	4.0	2.0	2.0	Full Adder Gate Delays Cin to Cout

As can be seen, both of the nine gate designs have the same number of gate delays from inputs to outputs. The eleven gate CMOS full adder has one additional gate delay through each data path. However, the logical effort delays do not align with the gate delays as will be seen. The maximum logical effort delay profiles for the three adders are compared in the following figure for the 32-bit by 32-bit Dadda column reduction multiplier.

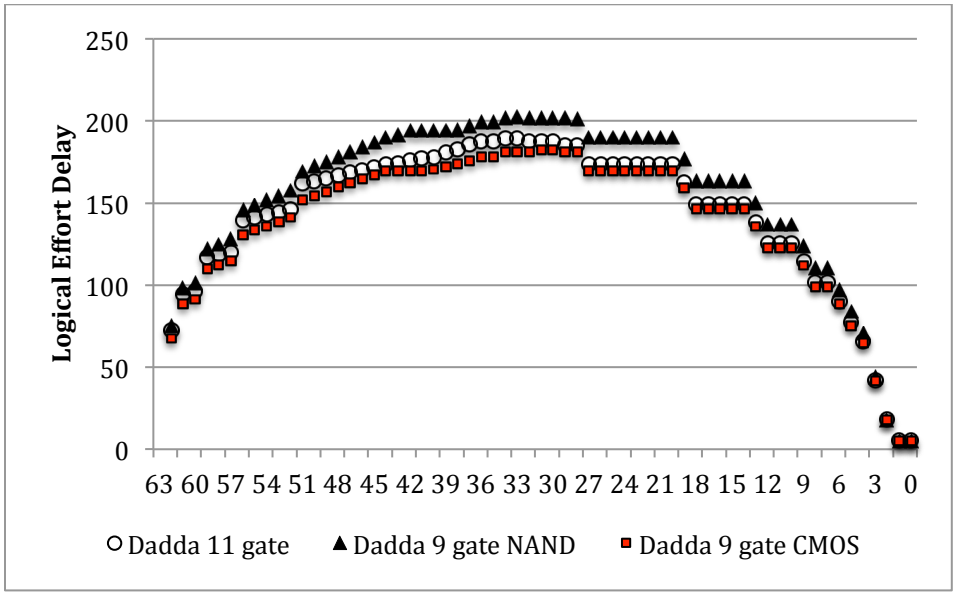

Figure 26: Three 32-bit Dadda Multipliers using different full adders

The eleven gate full adder is used in the multiplier analysis done in this work. One of the two nine gate implementations is faster then the eleven gate design, while the other full adder that uses nine NAND gates, is slower than the implementation using the eleven gate full adder. Clearly, results for logical effort modeling are dependent upon the adder designs used.

The objective for this dissertation is to analyze bit arrival times of various column reduction multipliers and suggest the best carry propagate adder designs to provide the best overall multiplier delay performance. To limit the scope of the designs, the five gate half adder and an eleven gate full adder are implemented; these designs use only inverters and two input CMOS gates. Alternative designs are, of course, possible using gates with three or more inputs or fewer gates such as the previously discussed nine gate full adders, or more compact circuit techniques such as merged gates.

Adders are defined by their number of gate delays and their gate count. For the half adder, the first delay is input (A, B) to sum (S) and the second delay is input to carry out (Cout). For the full adder, there are four numbers. The first number is the delay from input to sum, the second is for input to carry out (Cout), the third is for carry in (Cin) to sum and the fourth number is for carry in to carry out. The half adder used is a five gate implementation, in CMOS, using 2-input NAND gates and an inverter.

3-2 Five Gate Half Adder

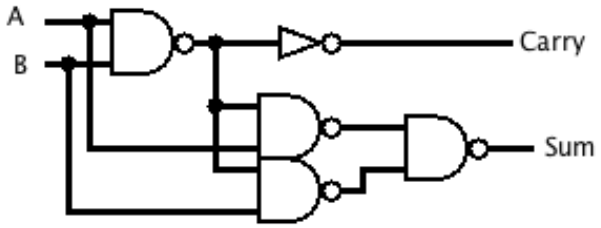

Figure 27: Schematic for Five Gate, 3-2 Delay Half Adder

For the full adder, the eleven gate CMOS design is used which is comprised of five 2-input NAND gates, two 2-input NOR gates and four inverters.

7-6-5-4 Eleven Gate Full Adder

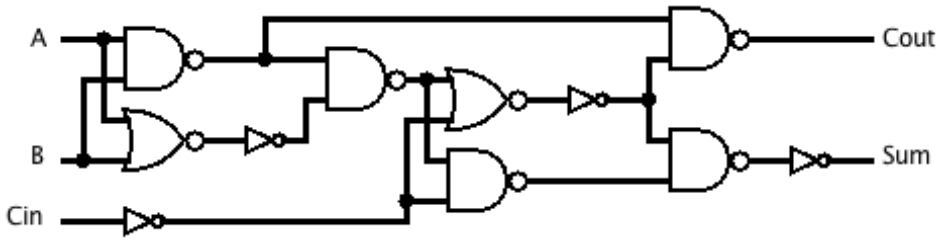

Figure 28: Schematic for Eleven Gate, 7-6-5-4 Delay Full Adder

Summary of Gate Delay Performance

The following tables summarize the gate delay counts for the half adder (HA) and the full adders (FA) being considered.

Table 8: Half Adder Delay Summary

HA Delays	5-Gate HA
In to Sum	3
In to Cout	2

Table 9: Full Adder Delay Summary

FA Delays	11-Gate FA
In to Sum	7
In to Cout	6
Cin to Sum	5
Cin to Cout	4

CARRY PROPAGATE ADDERS (CPA)

For review, for column reduction multipliers, there are three stages in the design. First the array of N^2 partial products is developed by the logical bit AND of each of the

terms of the multiplicand with each of the terms of the multiplier. Second, the desired column reduction technique is used to reduce the number of rows to the final two. Ultimately, the last two rows from the bottom of the column reduction process, representing the sum and carry terms from the column reduction, are added together using some type of carry propagate adder.

The designs of the final carry propagate adder will be performed using various types of adders, each with different delay characteristics. Hybrid adders using various lengths of different adders may be used to explore how to optimize the overall delay through the multiplier from the generation of the partial products through the outputs of the carry propagate adder. There are many designs for carry propagate adders (CPA). For this research a slow CPA, moderately fast CPA and a fast CPA are chosen for the analysis. Other carry propagate adders could have been selected, however, the three chosen for this research give a meaningful range of performance from slowest to fastest. The CPA will be designed using the following adder types:

Ripple Carry Adder [29]

The ripple carry adder is simply a chain of full adders that output a sum and a carry out into the carry in of the next most significant full adder. The adder on the LSB side is a half adder as there is no carry in for the LSB. The following figure illustrates the design of a ripple carry adder. Ripple carry is the slowest adder design. The critical path delay is from A and B inputs of the LSB to the carry out of the MSB of the CPA adder width.

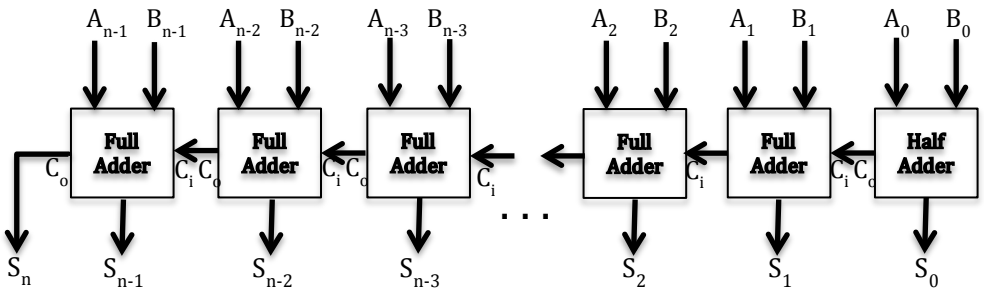

Figure 29: Schematic diagram of Ripple Carry Adder

Carry Select Adder [16]

Carry select adders are designed using two blocks of ripple carry bit adders that are each driven by a carry-in of one and zero. That way, both results are generated for the bits handled by the block before the propagate term arrives from the carry output of the previous block. A four bit carry select adder is illustrated in Figure 30. Both A and B adder inputs drive to full adders, each whose carry-in is either a one or a zero. The carry-in of the previous block controls which full adder output is multiplexed to the final adder sum out and also control which carry out state is propagated to the next carry select block.

Carry select adders are implemented in many ways. The optimum delay is achieved, for fixed block size and uniform data arrival times, when the carry select blocks are $\sqrt{N}$ where N is the width of the adder being designed [31]. Other implementations are possible as well, such as using variable width blocks. This research limits the block width for the carry select adder to four bit blocks. One reason for this is that for larger blocks, as the number of multiplexers that the carry out control from the previous block

increases, the branching effort or fanout increases to do the selection and impacts performance.

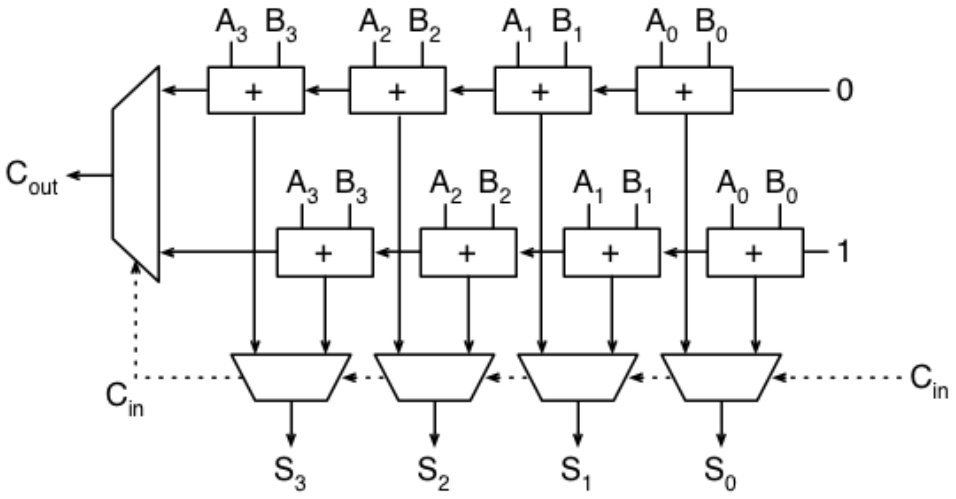

Figure 30: Schematic diagram of Carry Select Adder

Parallel Prefix Adder [20]

Kogge-Stone [20] carry lookahead or parallel prefix adders are the fastest adders and are used extensively in high-performance 32-bit and 64-Bit adders [30]. Kogge-Stone adders are built with blocks of logic that have been described as black cells, gray cells and buffers. The body of the adder is comprised of these cells and buffer building blocks which are various group propagate and generate cells shown logically as:

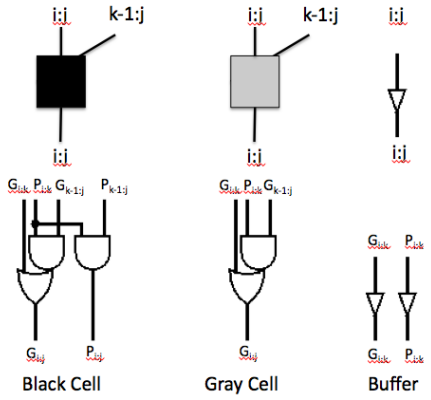

Figure 31: Basic Building Block Cells for Kogge-Stone Adder

The propagate terms are derived from the exclusive OR of the two inputs to be summed, while the generate terms are derived by an AND of the two inputs being summed. The resulting propagate and generate terms drive the black and gray cells of the Kogge-Stone architecture.

The final sum output is created by Exclusive ORing the propagate bit of bit position N with the generate term of bit position N-1.

The connections for a Kogge-Stone 16-bit adder are shown in Figure 32.

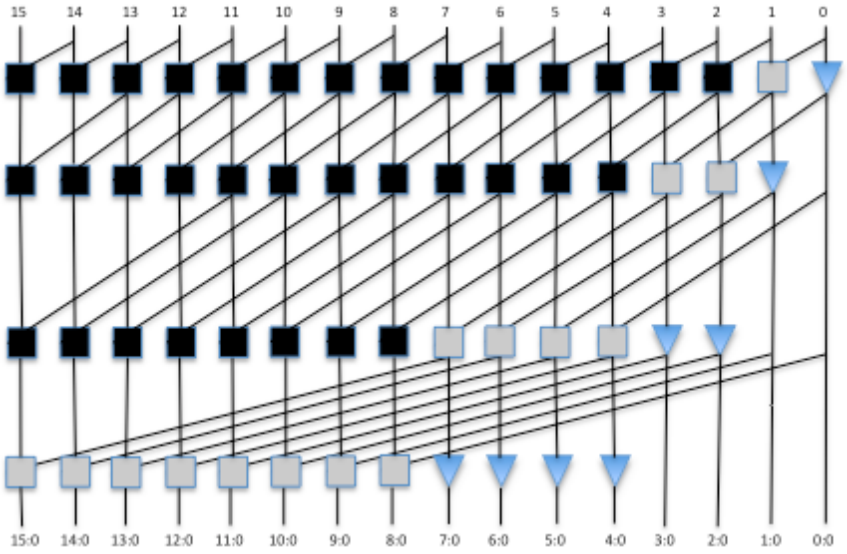

Figure 32: Schematic Diagram of Kogge-Stone Adder

INTEGER MULTIPLIER SIZES

Finally, it was expected that the design considerations may be different for various sizes of integer multipliers. Designs, using logical effort, were performed with various combinations of column reduction multipliers, final carry propagating adders (CPA) and sizes of multipliers including: 8-bit x 8-bit, 12-bit x 12-bit, 16-bit x 16-bit, 24-bit x 24-bit, 32-bit x 32-bit and 53-bit x 53-bit.

DESIGN CONSTRAINTS

There are many variables that could be considered in this research project; several design constraints have been imposed.

Static CMOS Design and Device Mobility

Static CMOS design techniques and topologies are used in the design of both the column multipliers as well as the final summing carry propagate adders (CPA).

The mobility relationship between PMOS and NMOS transistors is set at 2. This assumption drives the logical effort values throughout the analysis.

Reduction Stage Components

Various reduction techniques, using more complex compressors or counters, have been proposed beyond the reductions first proffered by Wallace and Dadda. This research will be limited to using classic full adders, which reduce three inputs of the same significance to one output of the same significance and one output of the next most significance (3,2) and half adders, which reduce two inputs of the same significance to one output of the same significance and one output of the next most significance (2,2). The others, including Oklobdzija [9] and Santoro [21], have used higher order (4,2) and (9,2) counters in order to reduce the delay through the column reduction section of the multiplier. Robinson [22] used a (4,3) counter for selected multiplier sizes that enabled

the removal of one stage of reduction delay in the column reduction section of the multiplier. The Robinson approach, however, is limited to multipliers of bit size 5, 14, 20, 29, 43, etc.

Integer Multiplication

This research focuses upon unsigned integer multiplication. The principles and practices learned are applicable beyond this limitation. By selectively inverting terms and adding a one in the top left position and bottom left position of the partial product array, a two's complement multiplier can be developed [23].

Adjacent Row Grouping in Reduction Stages

For the column reduction stages, two or three terms are grouped and applied to either a half adder or full adder, respectively, for reduction. All of the designs in this project, with the exception of Dadda, will use adjacent row terms in a reduction stage to generate the duples, for inputs to the half adders, or triplets, for inputs to the full adders, that are applied for column reduction. This is consistent with Wallace techniques as described and, for routing purposes, would result in the least complexity of routing, since grouped terms are in physical proximity. The initial work done, comparing overall multiplier results using adjacent grouping as compared to grouping to minimize column delay reduction [8], suggests that adjacent grouping is nearly equivalent. For Dadda, columns are grouped in order to minimize the sum and carry delays from each of the adder outputs. Because Dadda minimizes the use of adders in each reduction stage, there are many terms in each stage that do not increase from reduction step to the next. That is not the case with all other reduction methods, therefore, for Dadda, there are opportunities to selectively group terms in order to pair terms such that delays are minimized further than if adjacent rows were grouped. The following figure illustrates

the incremental delay improvement derived from selective grouping of reduction terms, versus adjacent term grouping, for a 24-bit by 24-bit Dadda column reduction multiplier.

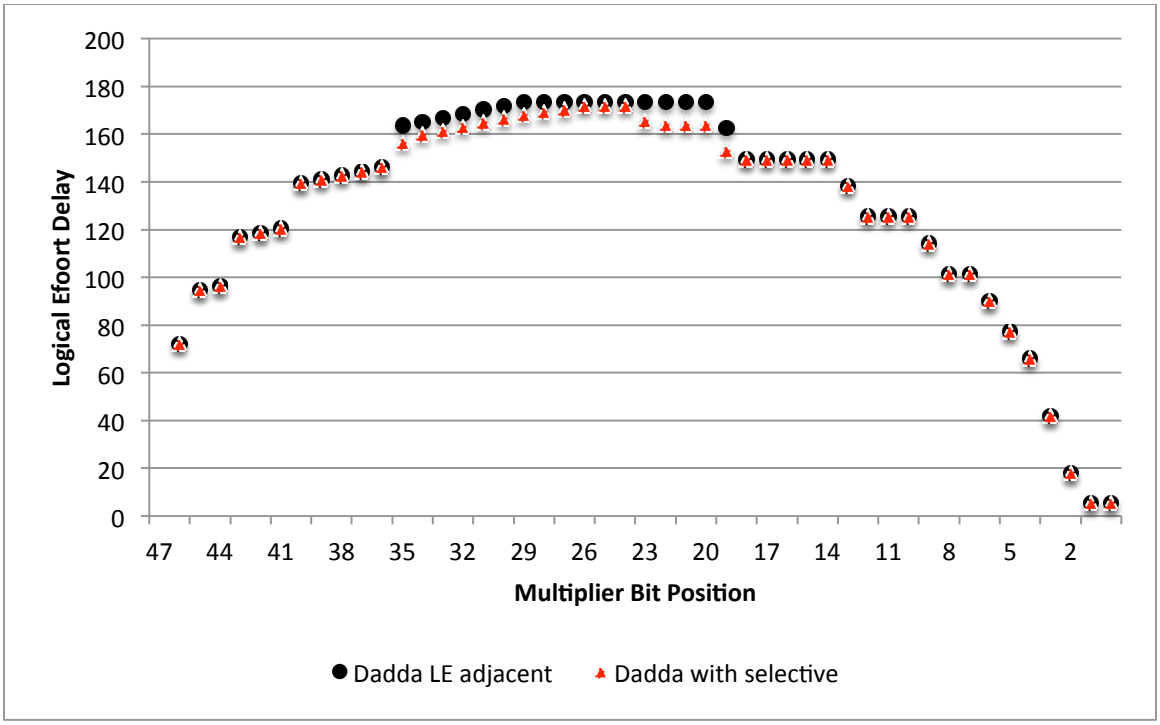

Figure 33: Delay of 24-Bit by 24-Bit Dadda with Selective term grouping

The speed improvement by selectively grouping terms in the Dadda multiplier is between 5% to 11% for the multipliers analyzed with the exception of the 24-bit multiplier and 53-bit multiplier, which only benefited by 1%. It should be noted that since Dadda reduction does not use grouping of adjacent terms, but is implemented with selective term grouping to minimize the delay through a reduction stage, that there might be increased metal routing lines which would increase the load capacitance and impact the performance. This potential increase in delay, which would contribute to increased "branching effort," is not considered in this analysis since the actual impact is dependent

upon actual design layout. If selective grouping of terms was not done, then it will be seen that the Dadda performance is the same as for the other three reduction methods.

Multiplier Configuration

For the work performed in this research, the number of bits of the multiplier and the number of bits of the multiplicand are the same.

Delay Considerations

Complete logical effort principles are used to develop the delay models for each of the multipliers. All designs were done considering CMOS elements for implementation. For the H term in the logical effort model, it is assumed that the input of the multiplier and the output that the multiplier drives into have the same size and capacitance. Therefore, the H value for calculations is of value 1.

Wire delays were not factored into the analysis. Burgess in [28] determined that the impact of wire interconnect crossing one bit position in an adder is approximately the same as a simple inverter input capacitance. Since the Kogge-Stone architecture has many lines traversing many bits, the performance impact on Kogge-Stone adders is believed to be higher than for similar width carry select adders. Consequently, the additional capacitance of wire would reduce the estimated performance advantage of the Kogge-Stone adders.

DELAY PROFILER DEVELOPMENT

Using Matlab and some manual designs, column multiplier designs were developed for the four column reduction methods (Wallace, Dadda, reduced area and modified Wallace). Using Excel, carry propagate adder (CPA) delay models were also designed to model the delays of various configurations of CPA with inputs from the various column multipliers. Combinations of column reduction methods with various

CPAs and hybridized CPAs were modeled for the multiplier widths and reduction methods of interest. From this analysis, fundamental design considerations for minimizing the delay through the various multiplier reduction methods and sizes were explored.

Chapter 8: Column Reduction Delay Results

Column reduction models were developed for 8-bit, 12-bit, 16-bit, 24-bit, 32-bit and 53-bit multipliers. Four different column reduction methods, Wallace, Dadda, reduced area and modified Wallace were used to reduce the terms to two final terms to apply to the carry propagate adder. The following figures illustrate the results, for logical effort delay, through each of the multipliers. The Dadda model results reflect selective grouping in order to have minimum delay through the columns. It is of note that if Dadda terms were group by adjacency as with the other reduction methods, then the delay through the Dadda multiplier would match that of the other three methods.

The following figures contain discrete data, however, lines are drawn connecting the data points to aid in seeing the data point groupings by multiplier reduction type.

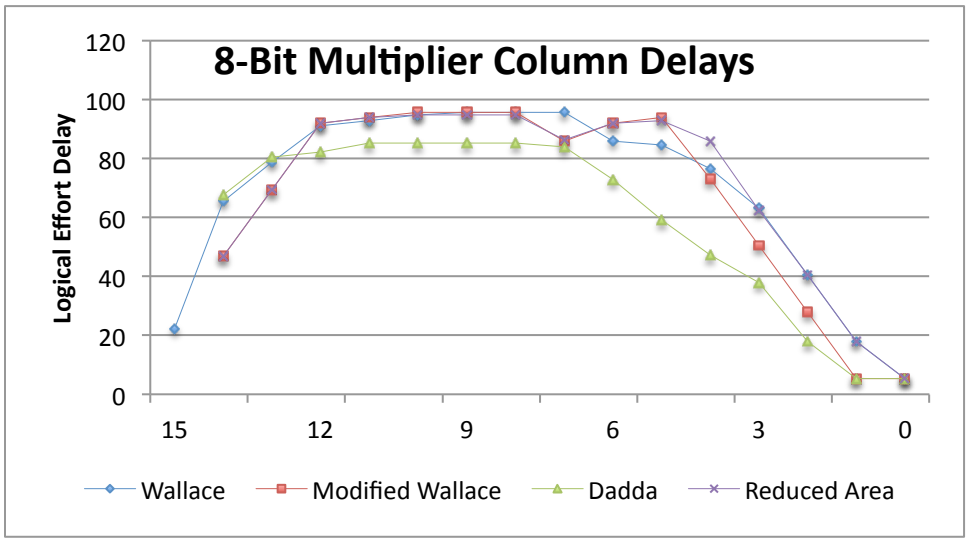

Figure 34: 8-Bit by 8-Bit Column Reduction Multiplier Delays

For the 8-bit Dadda column reduction, selective grouping of terms in the second reduction stage results in 11% improvement in delay performance through the column reduction multiplier.

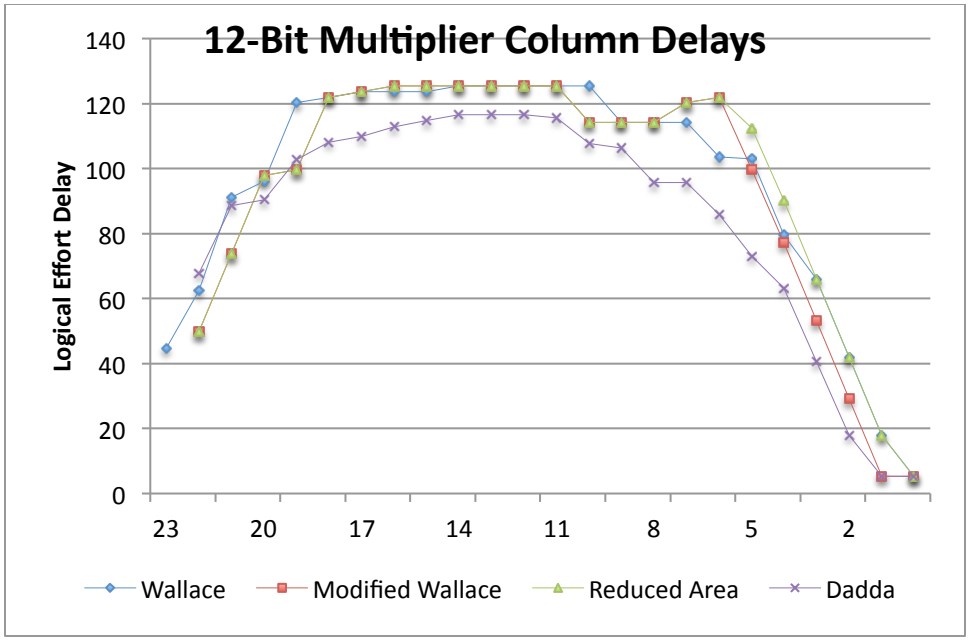

Figure 35: 12-Bit by 12-Bit Colum Reduction Multiplier Delays

For the 12-bit multiplier, selective grouping of reduction terms in the second stage of the Dadda column reduction results in a 7% improvement in delay through the column reduction stages of the multiplier.

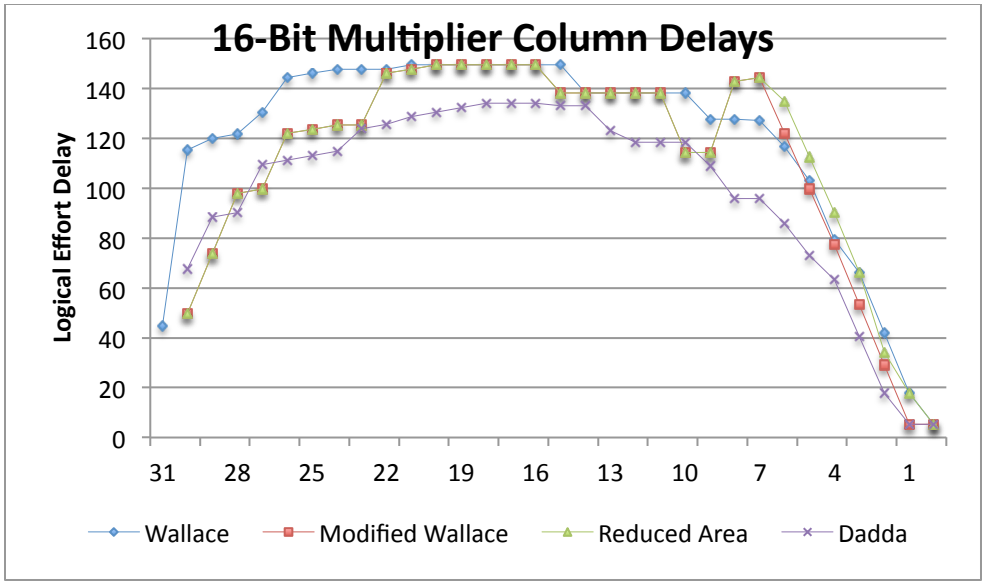

Figure 36: 16-Bit by 16-Bit Column Reduction Multiplier Delays

For the 16-bit Dadda multiplier, selective term grouping in the second stage of column reduction results in a 10.3% reduction in delay through the column reduction multiplier section of the multiplier.

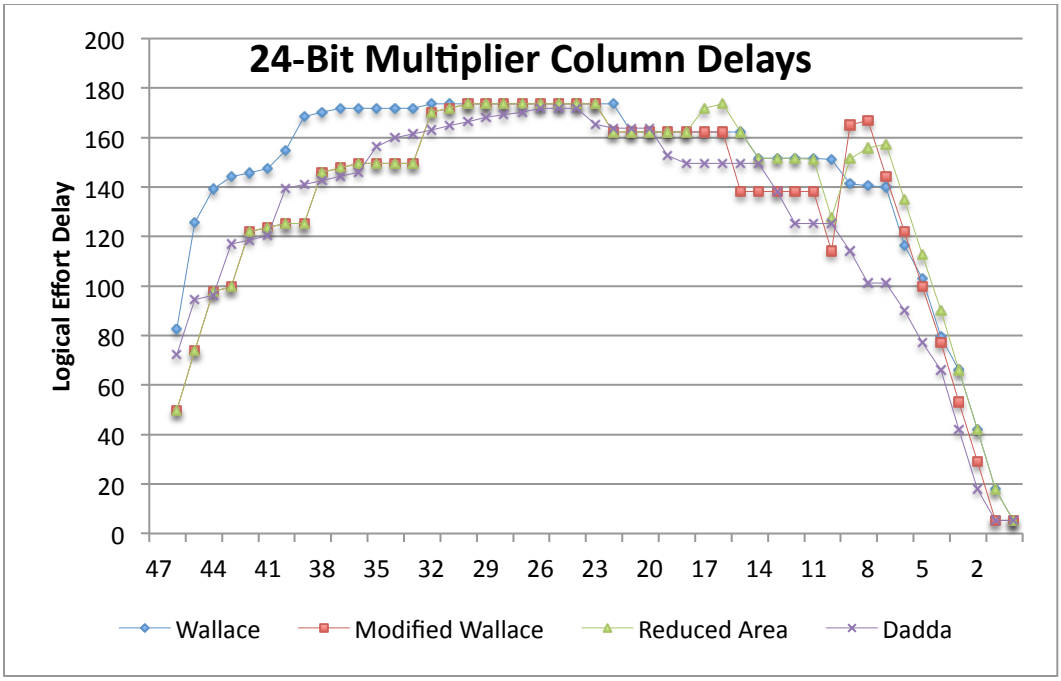

Figure 37: 24-Bit by 24-Bit Column Reduction Multiplier Delays

The 24-bit Dadda multiplier benefits next to least from selective term grouping. Grouping the reduction terms from the first stage in the second stage of reduction results in only 1.1% improvement in delay through the column reduction portion of the multiplier.

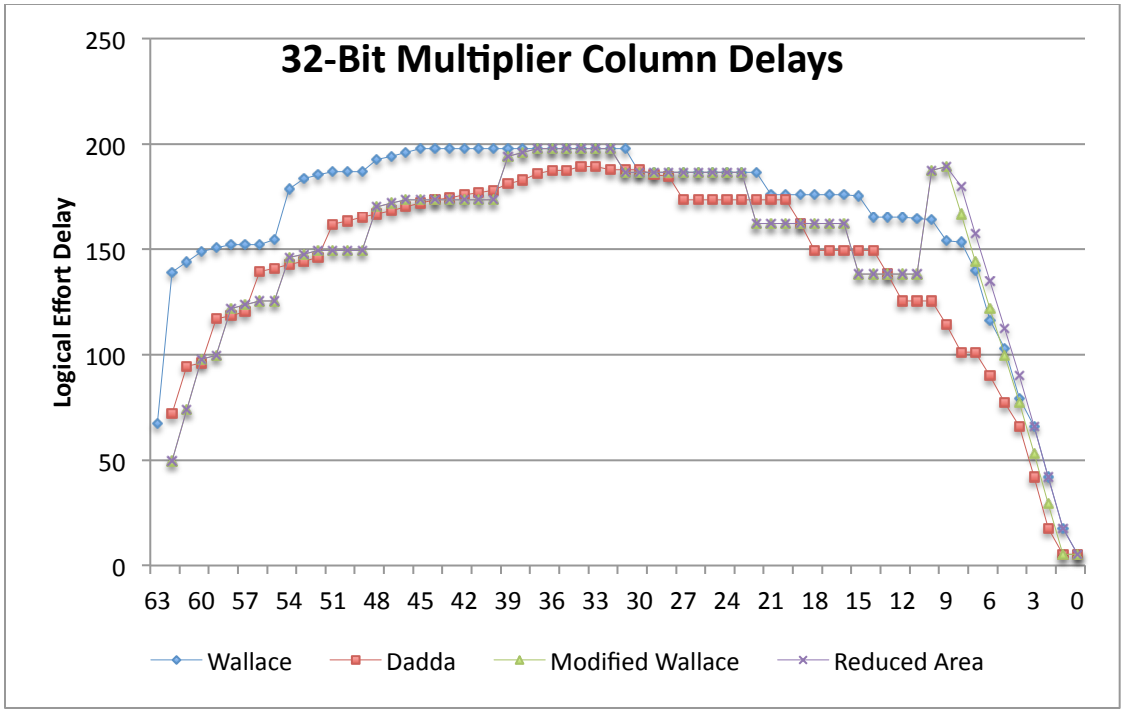

Figure 38: 32-Bit by 32-Bit Column Reduction Multiplier Delays

For the 32-bit Dadda multiplier, selective term grouping in the second stage of column reduction results in a 4.2% reduction in delay through the column reduction section of the Dadda multiplier.

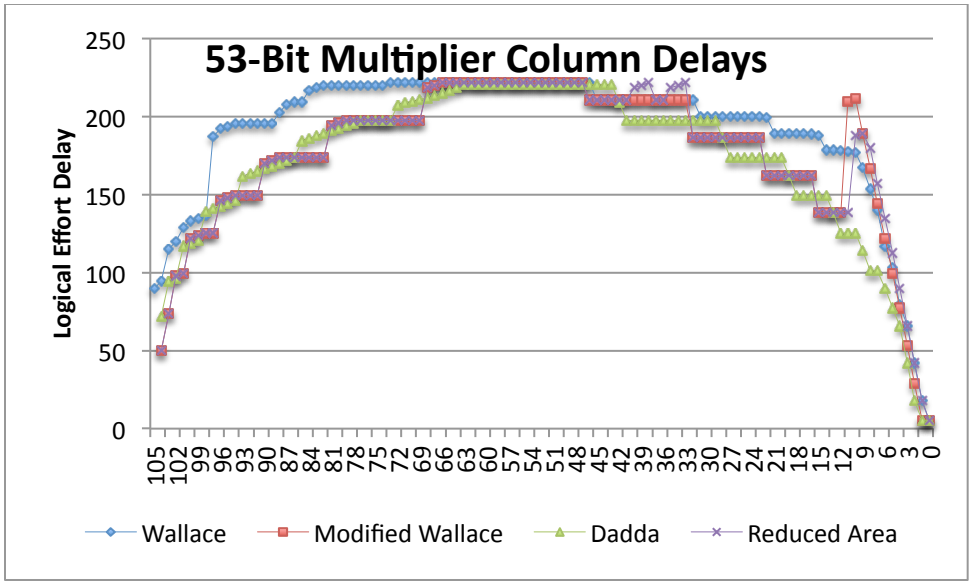

Figure 39: 53-Bit by 53-Bit Column Reduction Multiplier Delays

For the 53-bit Dadda multiplier, selective term grouping in the second stage of column reduction results in a less than 1% reduction in delay through the column reduction multiplier section of the multiplier.

In all cases, but with varying improvement, the selective grouping of terms in the second reduction stage of the Dadda multiplier reduction results in a shorter delay through the column reduction portion of the Dadda multiplier. Without the selective grouping, then Dadda reductions match the delay through the other three reduction methods. Therefore, for all multiplier sizes analyzed, the Dadda column reduction method resulted in less delay through the column reduction section of the multiplier. The minimal use of reduction adders in the reduction stages causes small delay values from previous stages to be passed down into the next multiplier reduction stage. Since the full adder has two inputs (A and B) with relatively long delay and one input (Cin) with shorter delay, if there are twice the number of delays in a column that are relatively short

as compared to adder outputs from the sum of the adder of the previous reduction stage, then there is opportunity to selectively group terms and, with Dadda, develop a multiplier design with less overall delay than the other three reduction methods.

Chapter 9: Results Based upon Multiplier Type

Wallace Results

Wallace reduction based designs have been analyzed for multipliers of sizes 8, 12, 16, 24, 32, and 53 bits. The 3-2 four gate half adder and the 7-6-5-4 eleven gate full adders were used. Overall multiplier delays were profiled for a standard ripple carry adder, carry select adder and a Kogge-Stone parallel prefix adder. The following figures show the logical effort delay profiles for each of the multiplier sizes.

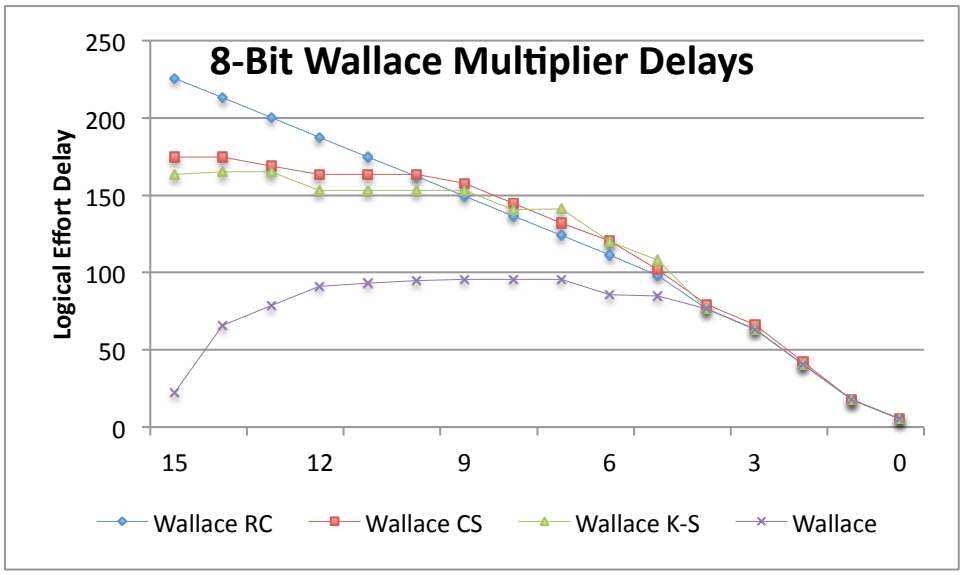

Figure 40: 8-Bit by 8-Bit Wallace Multiplier Results

Because the column delays in bits 0 through 4 are so much larger than the delays compiled through the CPA stages, all CPA designs have the same delays for the first five least significant bits. Thereafter, carry select and Kogge-Stone adders have similar delays with Kogge-Stone improving slightly more from product bit 10 onward.

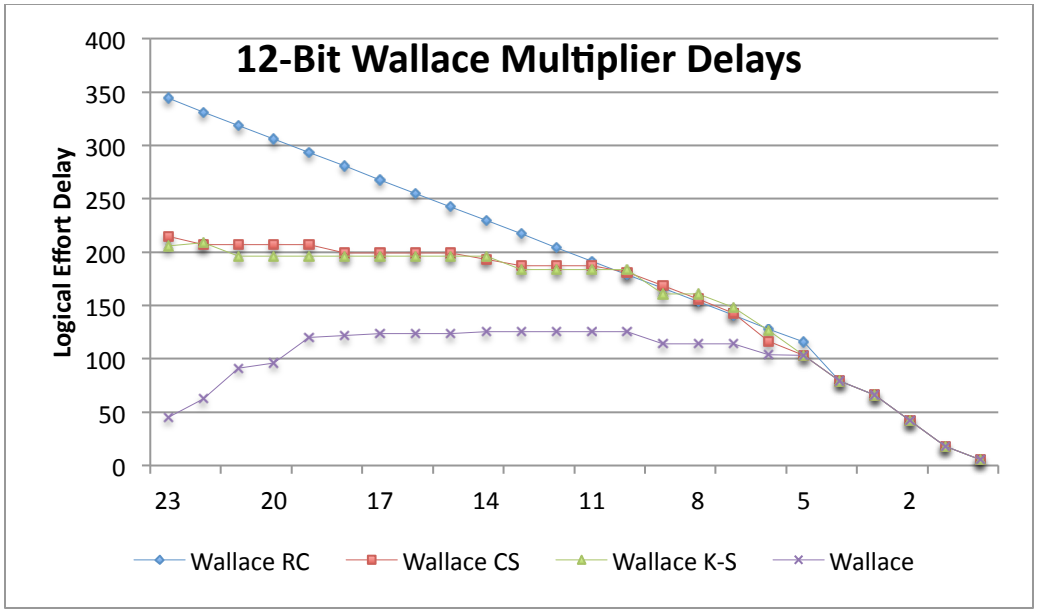

Figure 41: 12-Bit by 12-Bit Wallace Multiplier Results

In Figure 41, it appears that the Wallace multiplier using the carry select and Kogge-Stone adders have nearly the same performance. In order to evaluate this for validity, a Kogge-Stone model was generated with the effects of branching (fan-out) ignored by setting the branch effort to 1. The following figure illustrates the branching effort impact on Kogge-Stone multiplier performance. The models suggest that the benefit of parallel prefix adders, such as Kogge-Stone, are diminished due to their high fan-out or branching effort as compared to a conventional carry select carry propagate adder.

For the 12-Bit multiplier using a Kogge-Stone adder, the effect of branching in the parallel prefix carry propagate adder was removed and performance compared to a carry select and a Kogge-Stone adder with branching. Comparing carry select with Kogge-Stone adders, there is minimal delay difference. From the following figure, it is

clear that the impact of branching in the Kogge-Stone adder has significant impact on delay.

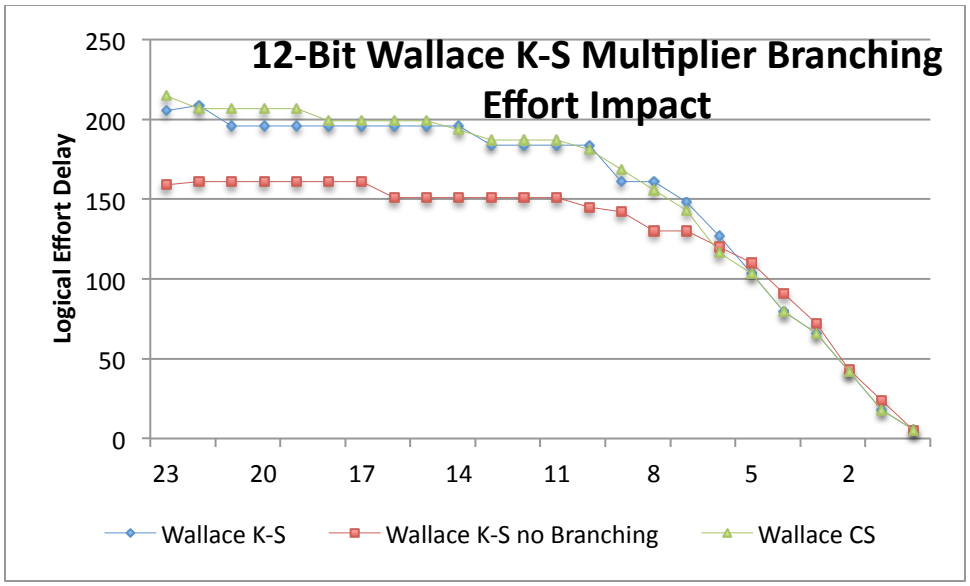

Figure 42: 12-Bit Wallace Multiplier with K-S Branching ignored

The following four figures compare ripple carry, carry select and Kogge-Stone adder's performance for 16-bit, 24-bit, 32-bit and 53-bit Wallace multipliers.

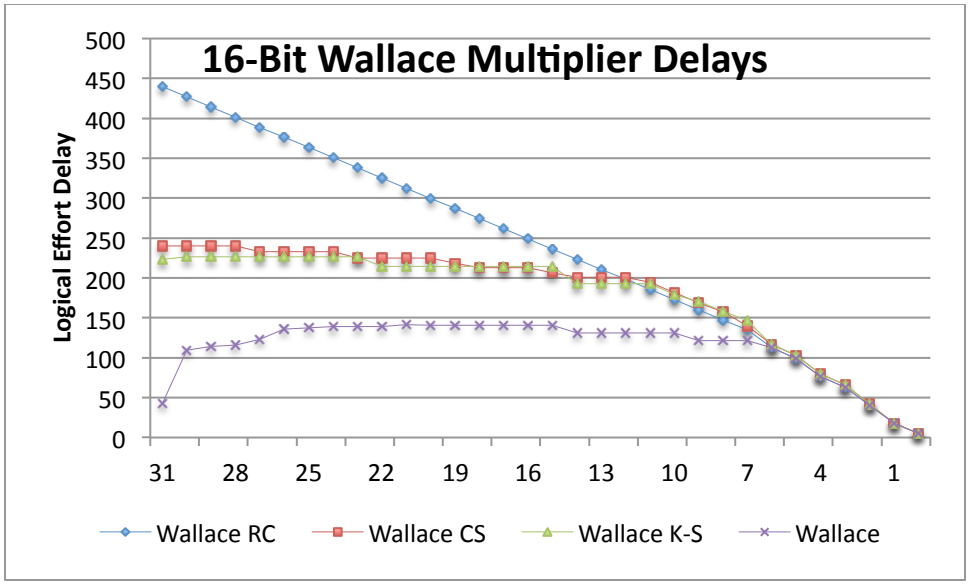

Figure 43: 16-Bit by 16-Bit Wallace Multiplier Results

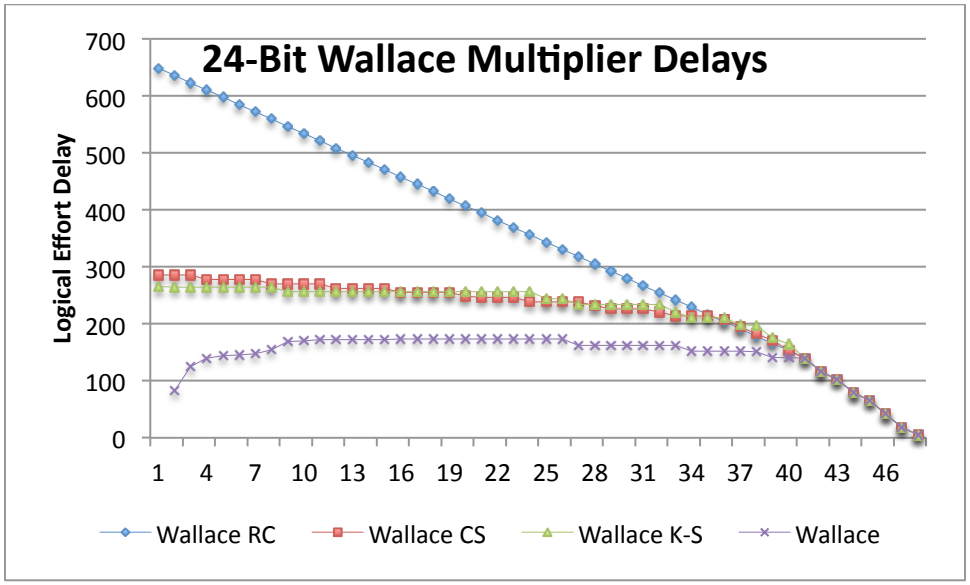

Figure 44: 24-Bit by 24-Bit Wallace Multiplier Results

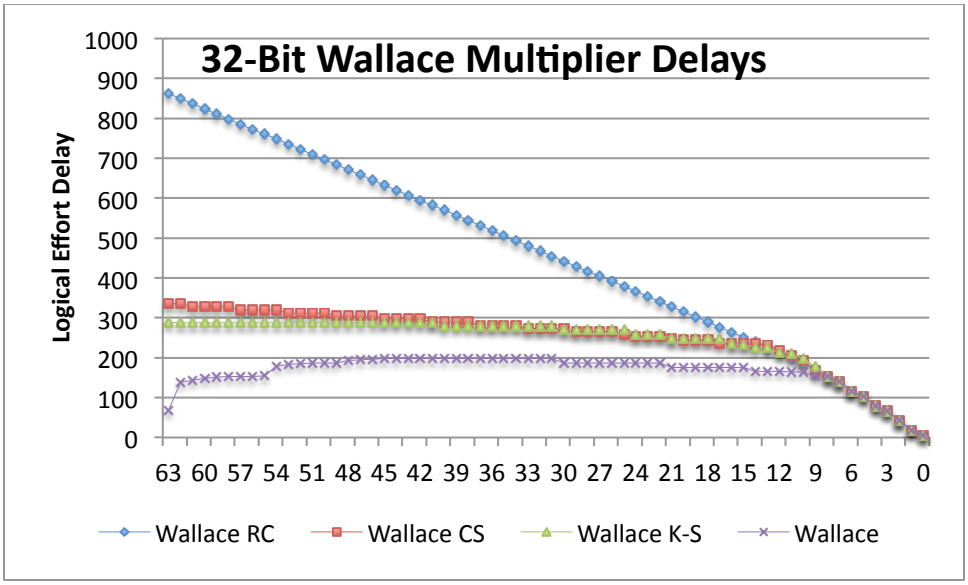

Figure 45: 32-Bit by 32-Bit Wallace Multiplier Results

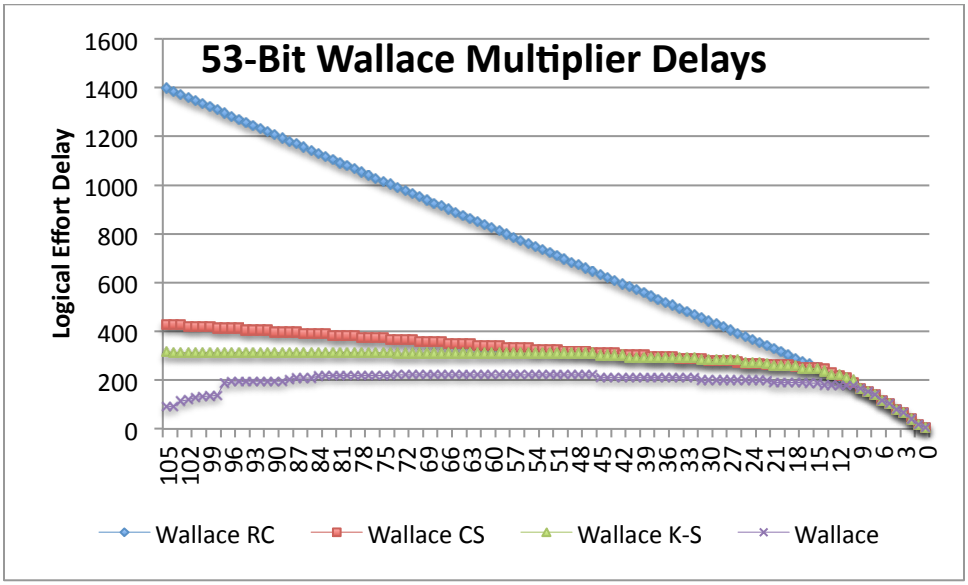

Figure 46: 53-Bit by 53-Bit Wallace Multiplier Results

For all of the Wallace multipliers, the Kogge-Stone adder has the best delay performance of the three carry propagate adders analyzed. However, the differences seen for smaller multipliers are not as evident as for the differences seen for the larger widths.

The following table summarizes the worst case logical effort delays through an 8-bit, 12-bit, 16-bit, 24-bit, 32-bit and 53-bit Wallace multiplier for the three carry propagate adders analyzed, ripple carry, carry select and Kogge-Stone final adders.

Table 10: Summary of Worst Case Delays for Wallace Multipliers

Wallace	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	226	344	444	649	862	1398
Carry Select	175	215	240	286	336	430
Kogge-Stone	165	209	227	266	290	315

For the 8-bit Dadda multiplier, the Kogge-Stone final adder results in only 6% better delay performance than a carry select final adder; for the 53-bit multiplier, Kogge-Stone is 27% faster than a multiplier using a carry select final adder.

DADDA RESULTS

Dadda reduction based designs have been analyzed for multipliers of sizes 8, 12, 16, 24, 32, and 53 bits. The 3-2 four gate half adder and the 7-6-5-4 eleven gate full adders were used. Overall multiplier delays were profiled for a standard ripple carry adder, a carry select adder and a Kogge-Stone parallel prefix adder. The following figures show the logical effort delay profiles for each of the multiplier sizes.

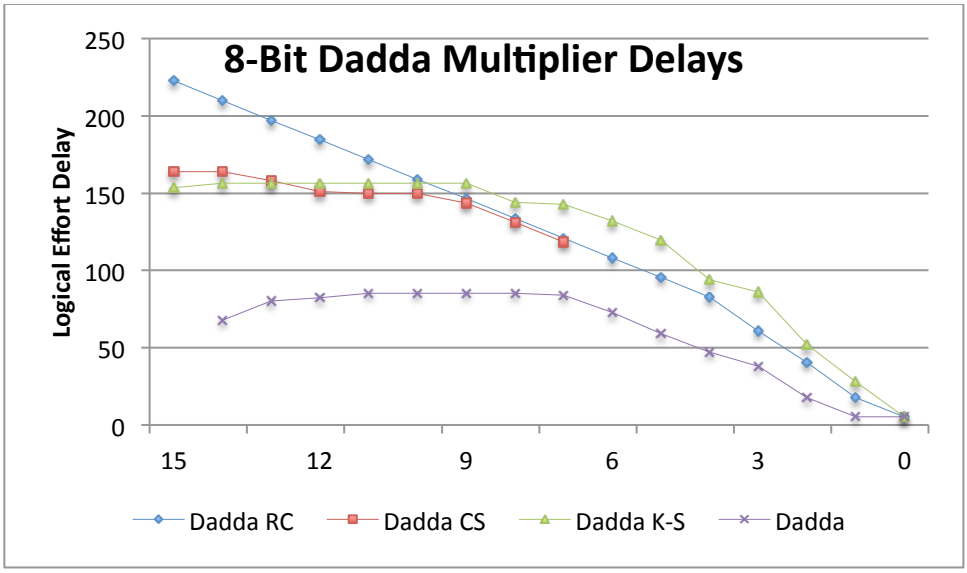

Figure 47: 8-Bit by 8-Bit Dadda Multiplier Results

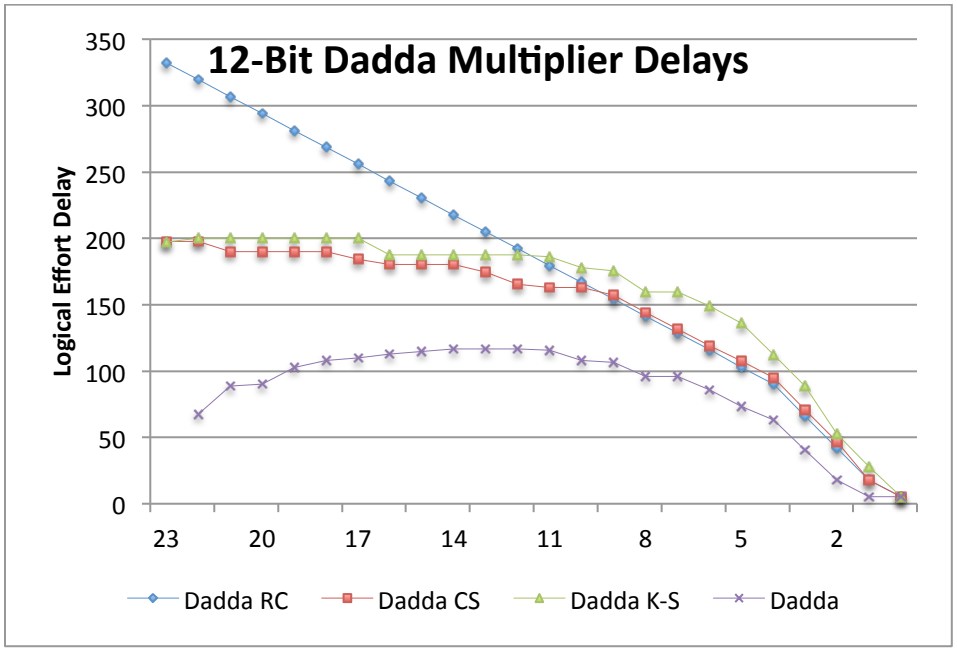

Figure 48: 12-Bit by 12-Bit Dadda Multiplier Results

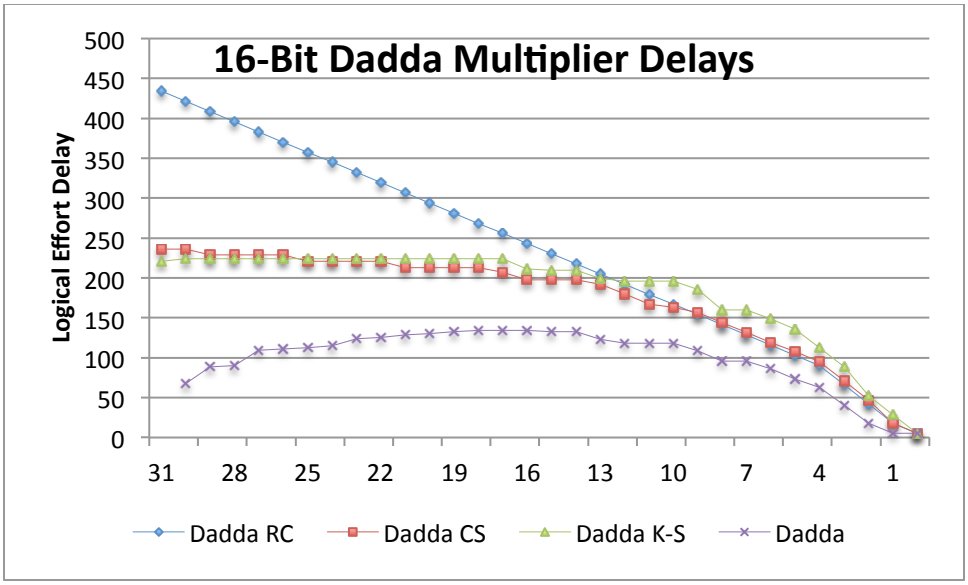

Figure 49: 16-Bit by 16-Bit Dadda Multiplier Results

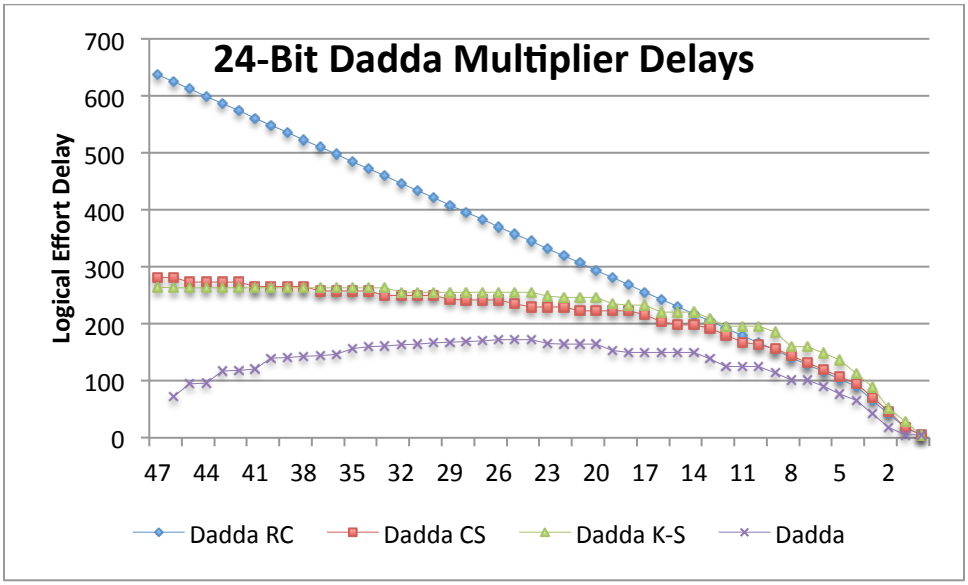

Figure 50: 24-Bit by 24-Bit Dadda Multiplier Results

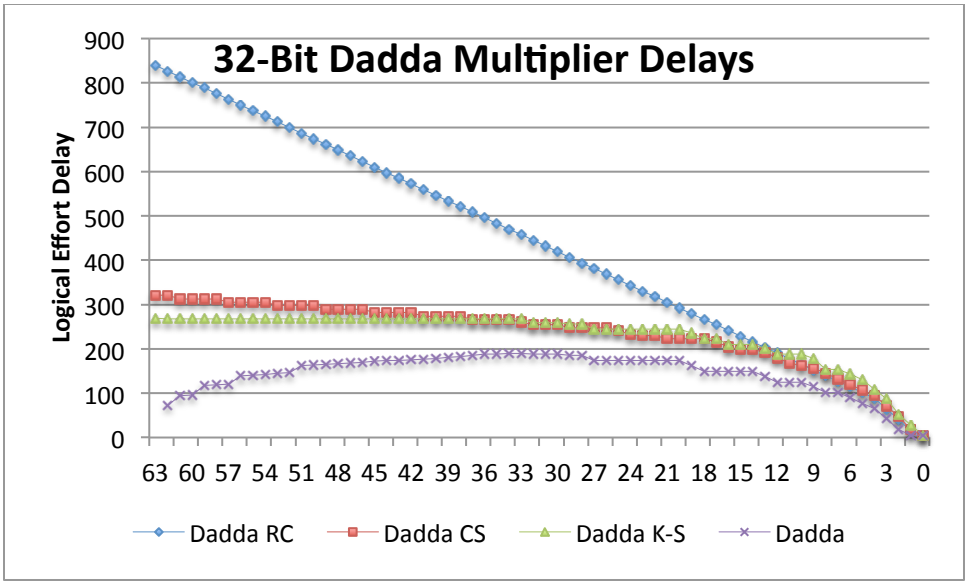

Figure 51: 32-Bit by 32-Bit Dadda Multiplier Results

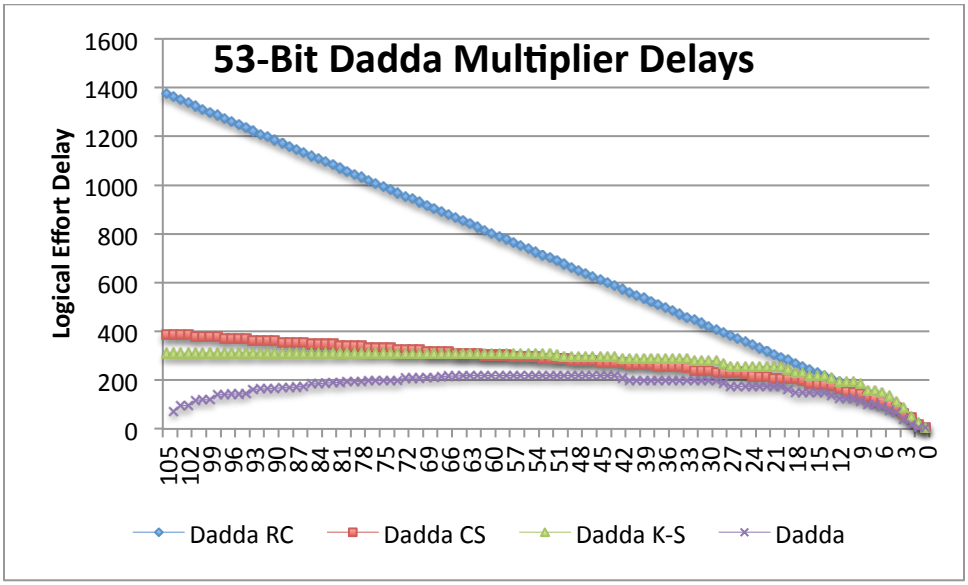

Figure 52: 53-Bit by 53-Bit Dadda Multiplier Results

For Dadda multipliers, the Kogge-Stone adder is the fastest carry propagate adder for the multipliers analyzed. However, the Dadda carry propagate adder begins at the

second bit on the LSB side and the Kogge-Stone adders is slower than either ripple carry or carry select adders for the first ten product bits, then surpasses both ripple carry and carry select adders in worst case delay performance.

Table 11: Summary of Worst Case Delays for Dadda Multipliers

Dadda	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	223	332	434	638	839	1376
Carry Select	164	198	236	281	319	409
Kogge-Stone	157	200	224	264	280	310

REDUCED AREA RESULTS

Reduced area multiplier reduction based designs have been analyzed for multipliers of sizes 8, 12, 16, 24, 32, and 53 bits. The 3-2 four gate half adder and the 7-6-5-4 eleven gate full adders were used. Overall multiplier delays were profiled for a standard ripple carry adder, a carry select adder and a Kogge-Stone parallel prefix adder. The following figures show the logical effort delay profiles for each of the multiplier sizes.

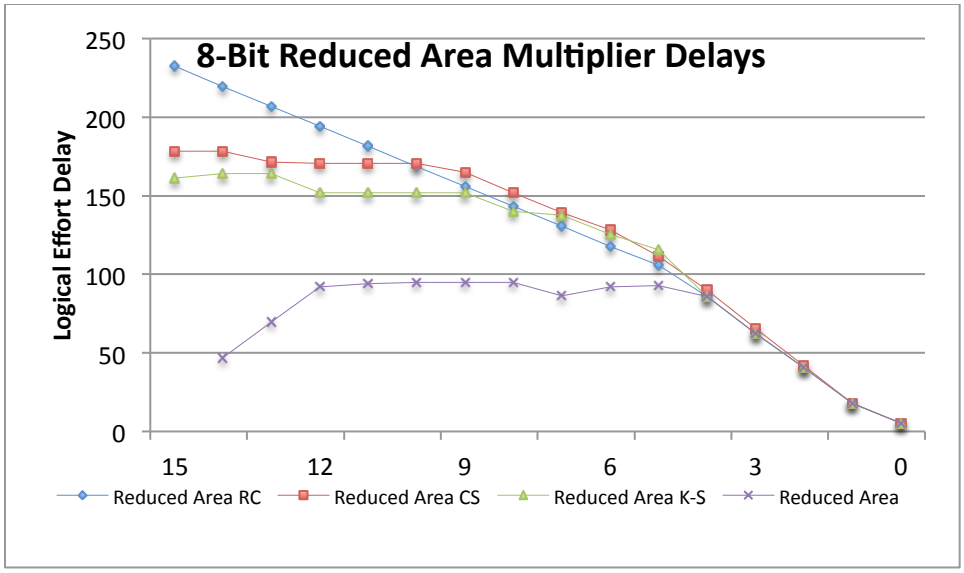

Figure 53: 8-Bit by 8-Bit Reduced Area Multiplier Results

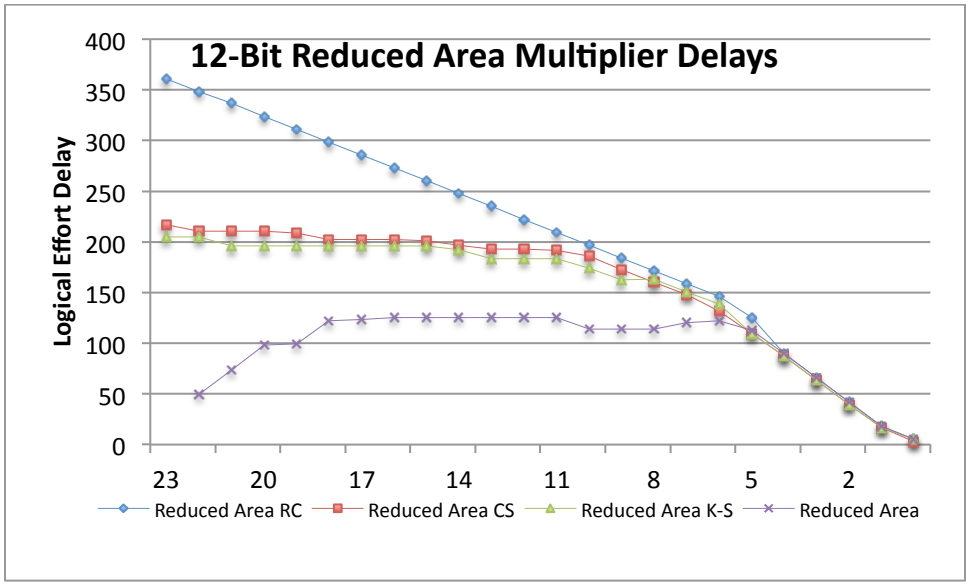

Figure 54: 12-Bit by 12-Bit Reduced Area Multiplier Results

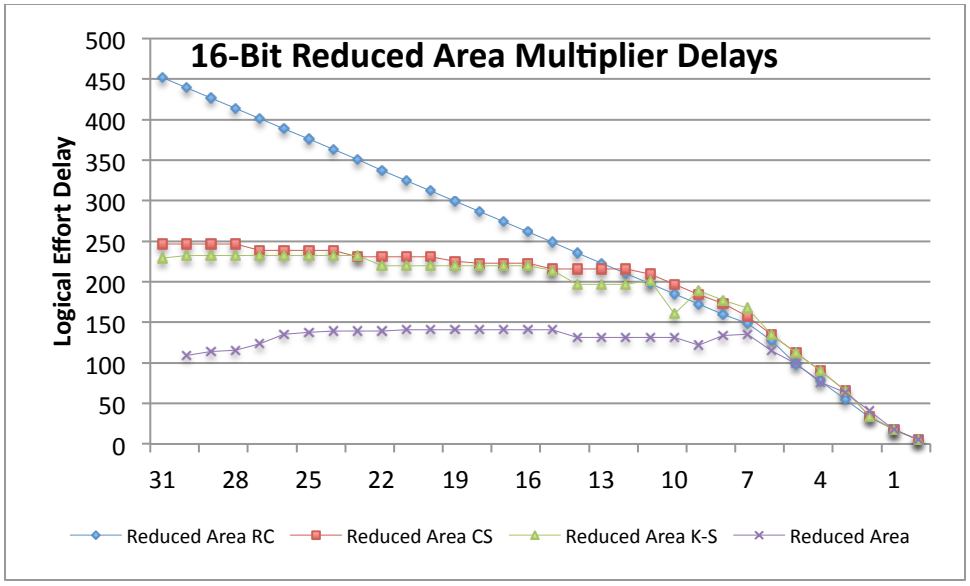

Figure 55: 16-Bit by 16-Bit Reduced Area Multiplier Results

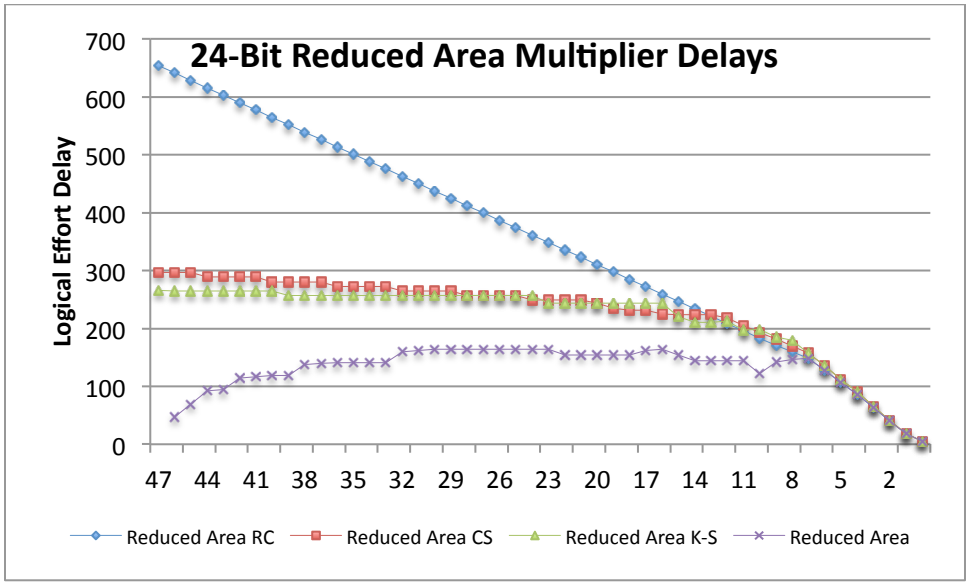

Figure 56: 24-Bit by 24-Bit Reduced Area Multiplier Results

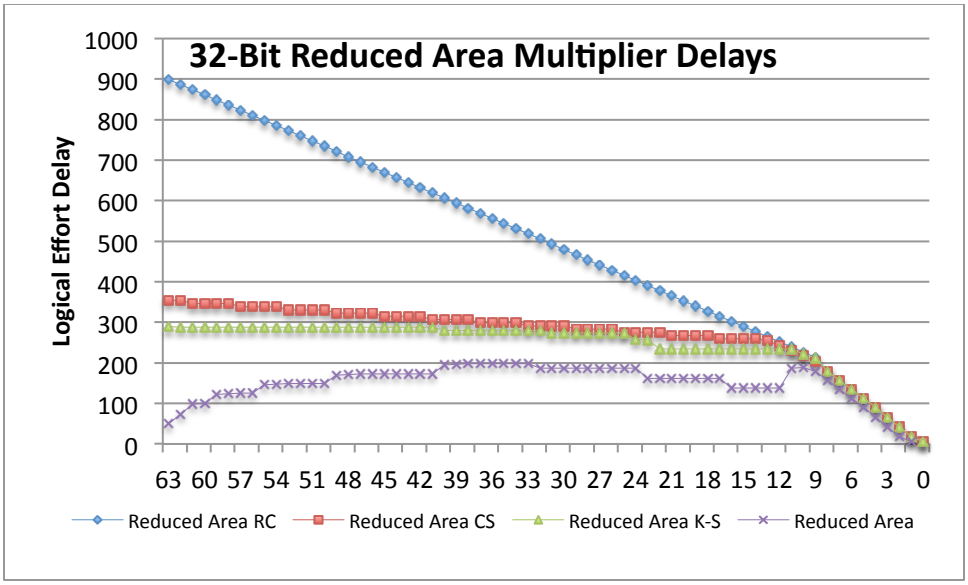

Figure 57: 32-Bit by 32-Bit Reduced Area Multiplier Results

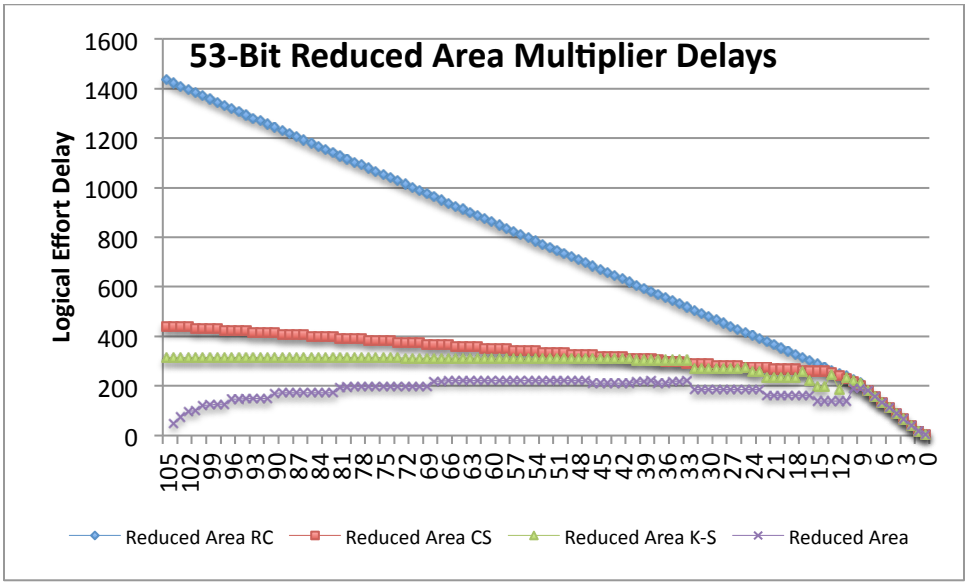

Figure 58: 53-Bit by 53-Bit Reduced Area Multiplier Results

For the reduced area multiplier, a carry propagate adder implemented with a Kogge-Stone adder has the fastest delay performance for all multiplier sizes analyzed.

This is especially true for the 53-bit multiplier where the Kogge-Stone adder is 29% faster than the carry select adder.

Table 12: Summary of Worst Case Delays for Reduced Area Multipliers

Reduced Area	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	232	361	461	662	900	1434
Carry Select	178	217	247	296	355	439
Kogge-Stone	164	205	232	266	290	314

MODIFIED WALLACE RESULTS

Modified Wallace reduction based designs have been analyzed for multipliers of sizes 8, 12, 16, 24, 32, and 53 bits. The 3-2 four gate half adder and the 7-6-5-4 eleven gate full adders were used. Overall multiplier delays were profiled for a standard ripple carry adder, a carry select adder and a Kogge-Stone parallel prefix adder. The following figures show the logical effort delay profiles for each of the multiplier sizes.

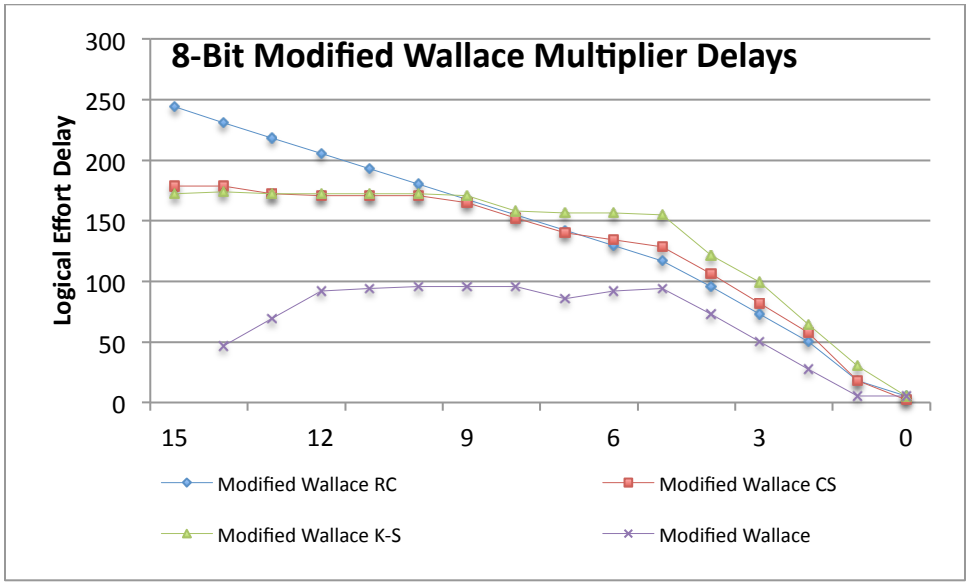

Figure 59: 8-Bit by 8-Bit Modified Wallace Multiplier Results

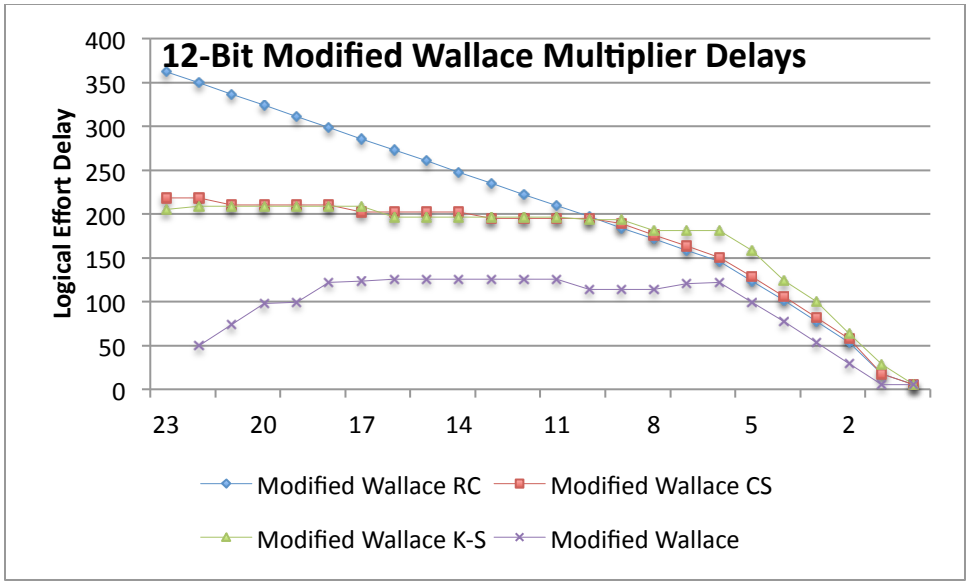

Figure 60: 12-Bit by 12-Bit Modified Wallace Multiplier Results

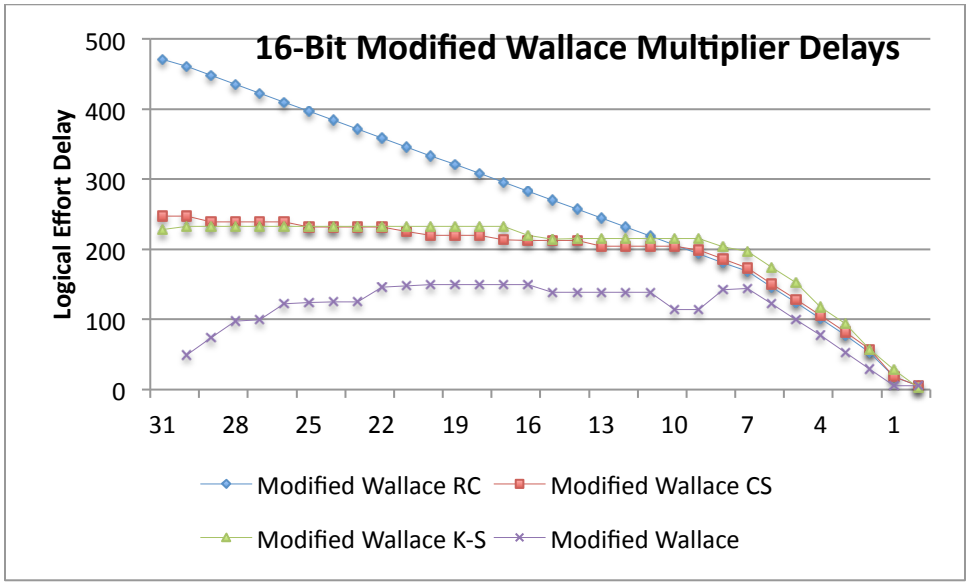

Figure 61: 16-Bit by 16-Bit Modified Wallace Multiplier Results

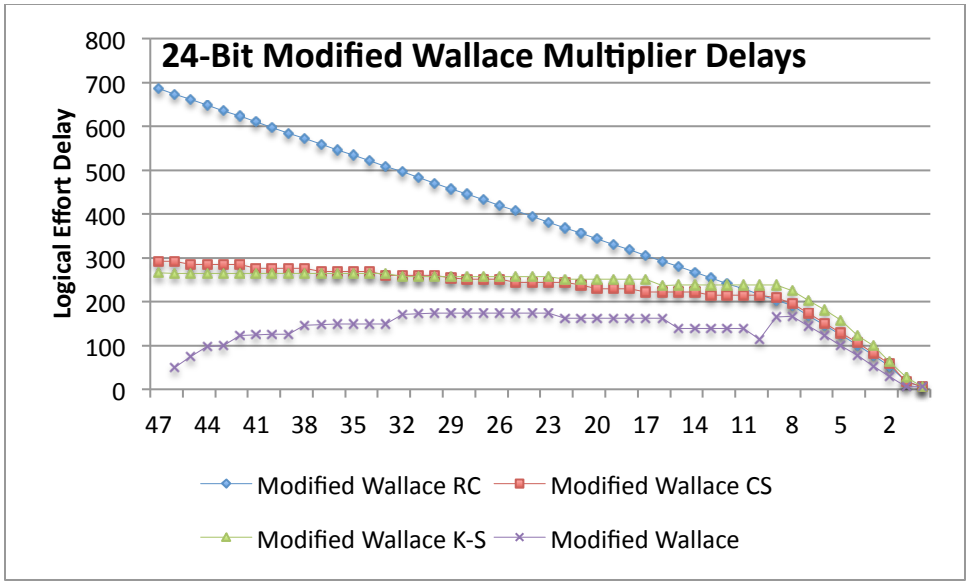

Figure 62: 24-Bit by 24-Bit Modified Wallace Multiplier Results

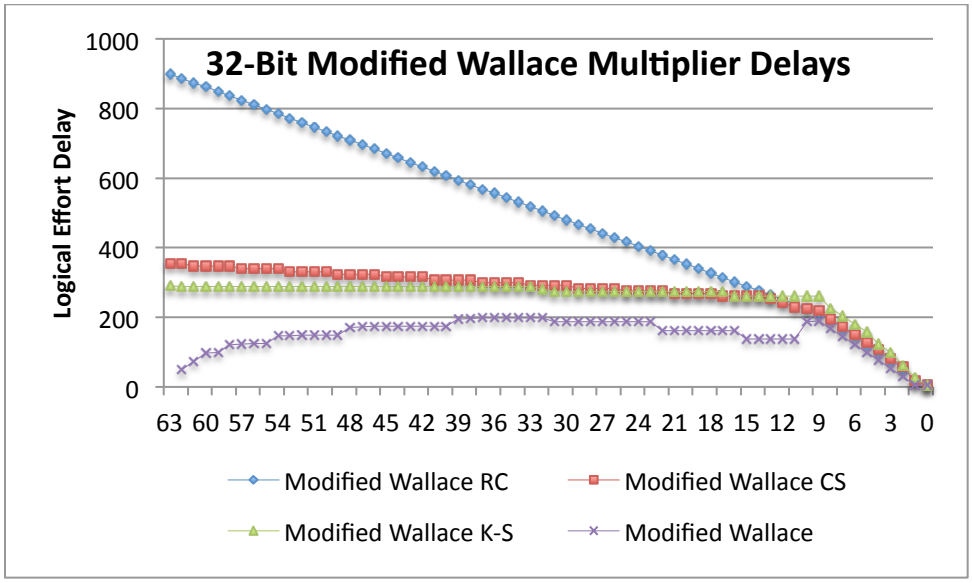

Figure 63: 32-Bit by 32-Bit Modified Wallace Multiplier Results

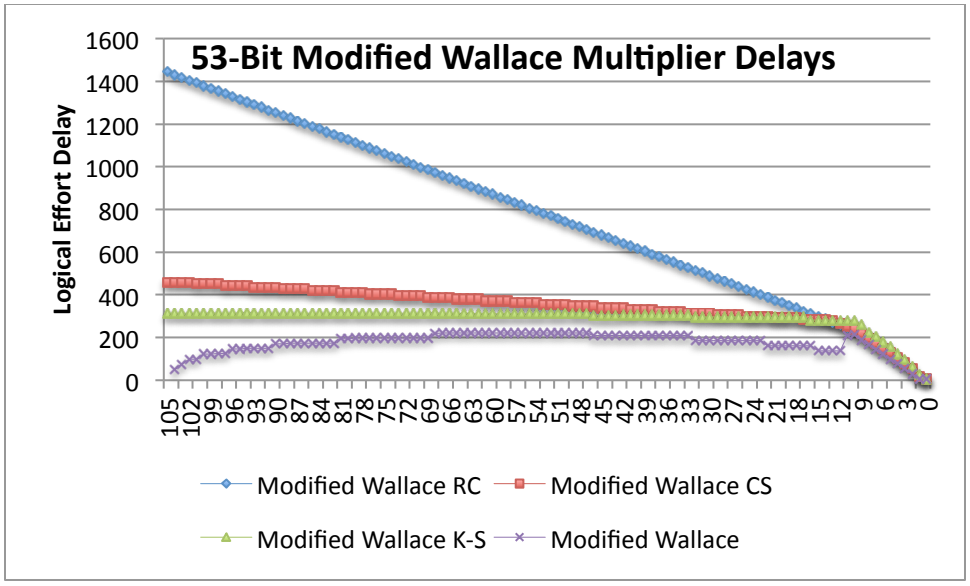

Figure 64: 53-Bit by 53-Bit Modified Wallace Multiplier Results

As with the Dadda multiplier, the carry propagate adder the modified Wallace multiplier begins at the second LSB position. Consequently, the Kogge-Stone adder initially has slower performance for the first few LSB's of the adder. However, its overall performance is superior to ripple carry or carry select adders. The advantage for the Kogge-Stone adder based 53-bit multiplier is 32% faster than a carry select adder for the modified Wallace multiplier.

Table 13: Summary of Worst Case Delays for Modified Wallace Multipliers

Modified Wallace	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	244	362	471	687	900	1444
Carry Select	178	218	247	292	355	459
Kogge-Stone	165	209	232	266	290	314

Chapter 10:Results Based upon Carry Propagate Adder

RIPPLE CARRY ADDER

For all ripple carry adder designs, the Dadda multiplier based design is the fastest.

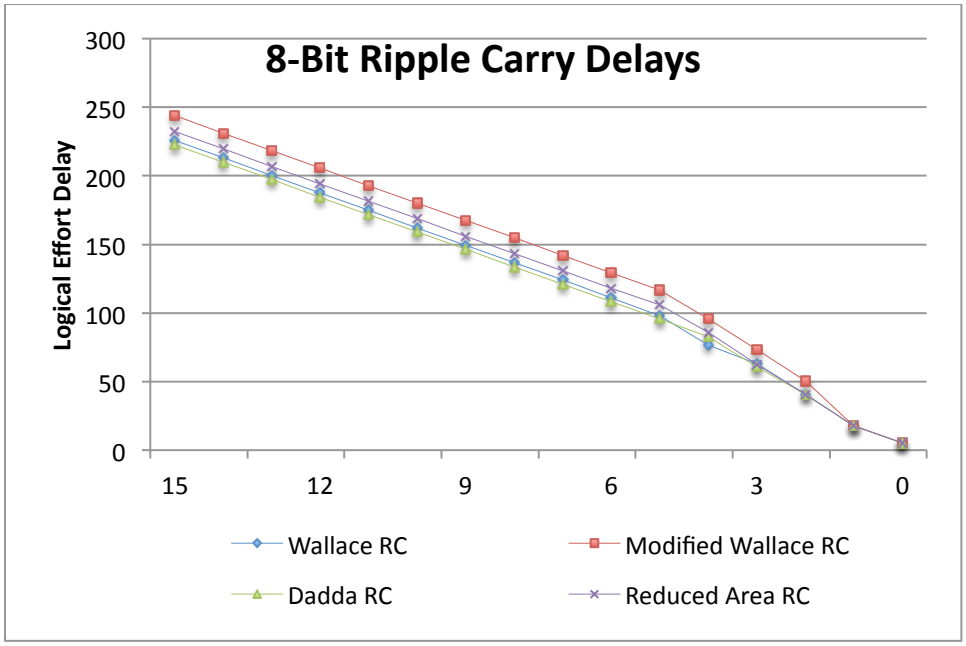

Figure 65: 8-Bit by 8-Bit Multiplier with Ripple Carry Final Adder

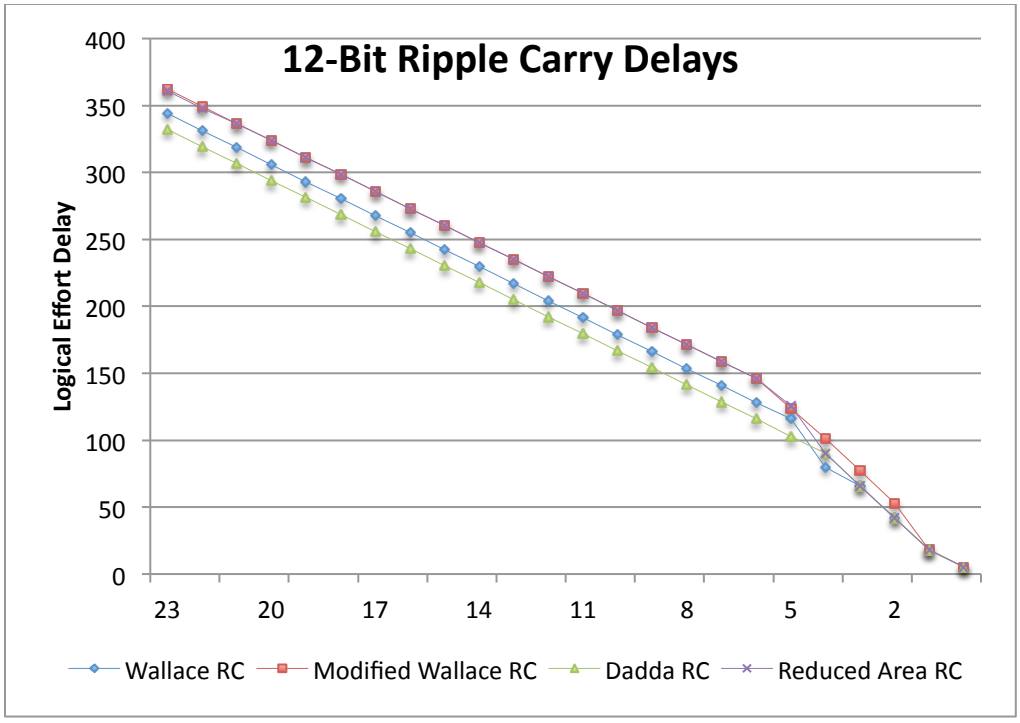

Figure 66: 12-Bit by 12-Bit Multiplier with Ripple Carry Final Adder

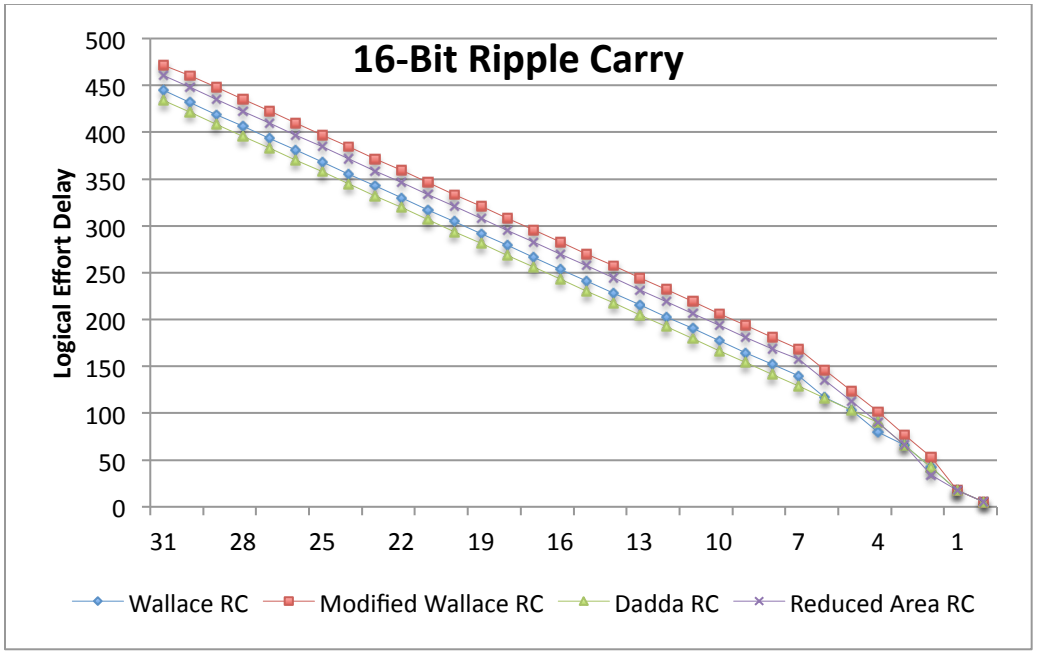

Figure 67: 16-Bit by 16-Bit Multiplier with Ripple Carry Final Adder

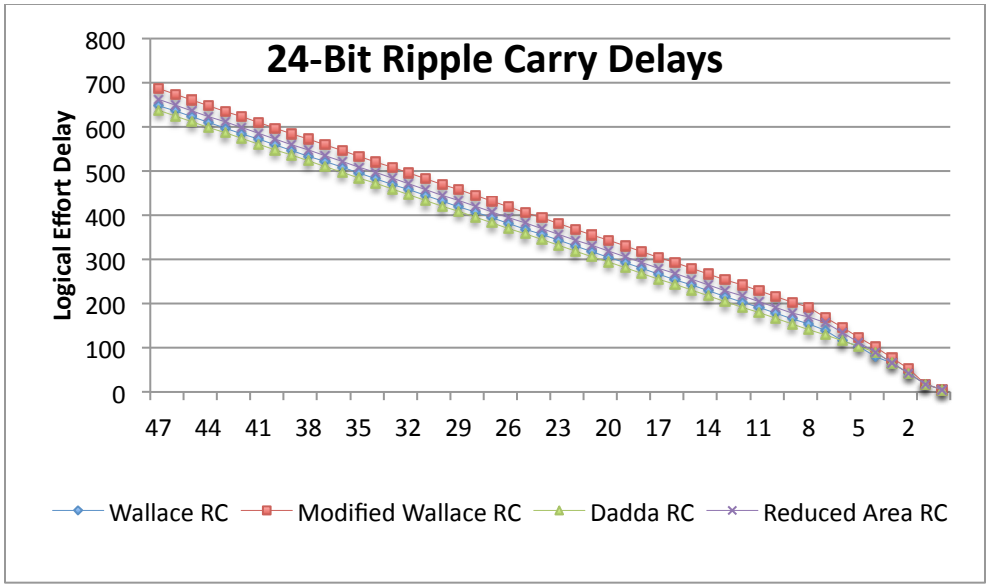

Figure 68: 24-Bit by 24-Bit Multiplier with Ripple Carry Final Adder

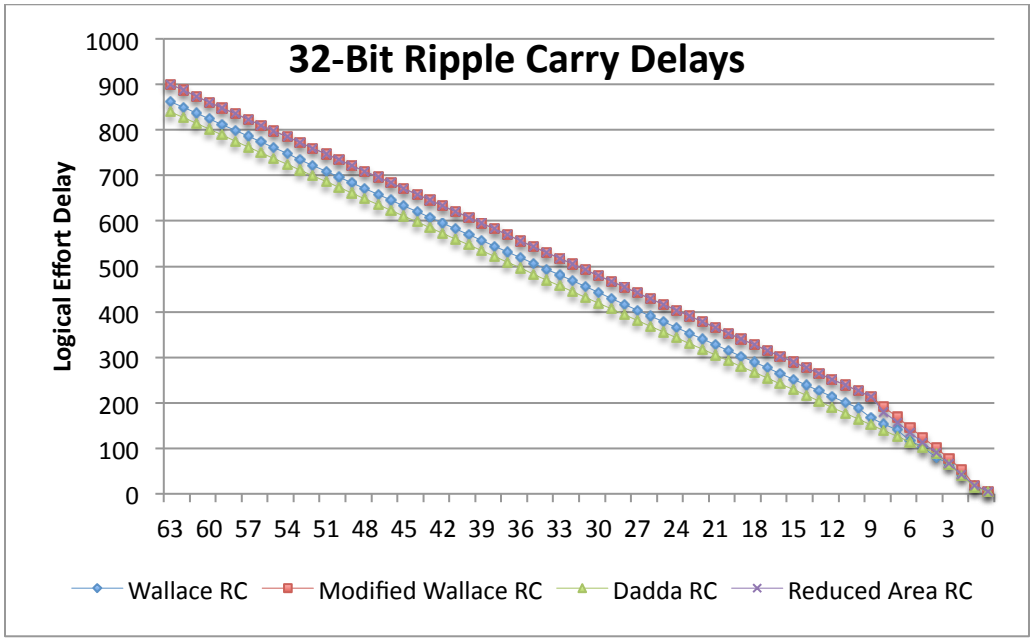

Figure 69: 32-Bit by 32-Bit Multiplier with Ripple Carry Final Adder

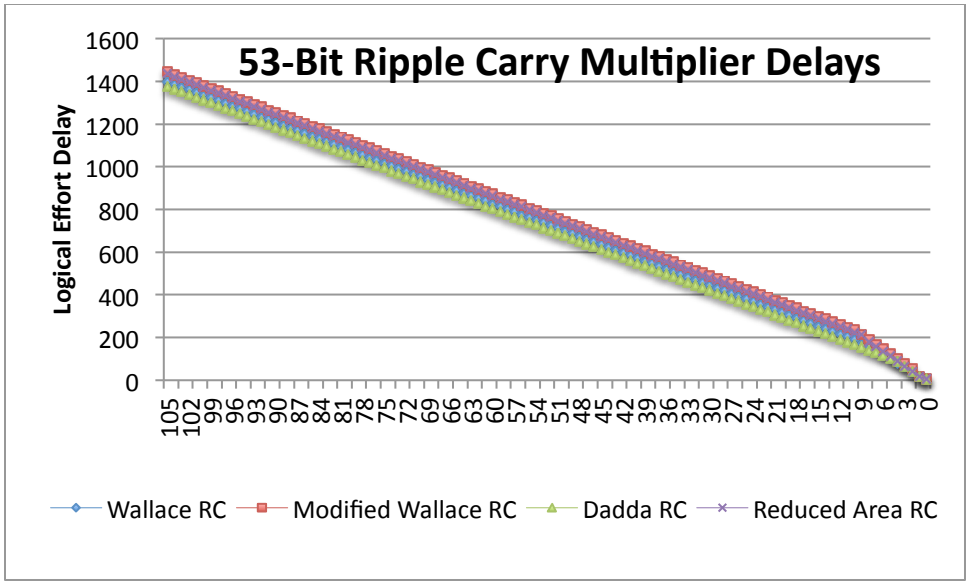

Figure 70: 53-Bit by 53-Bit Multiplier with Ripple Carry Final Adder

For multipliers implemented with a ripple carry adder for the carry propagate adder, the Dadda multiplier provided the best performance followed by the Wallace multiplier. Both the reduced area multiplier and the modified Wallace multiplier had slower overall performance than either the Dadda or Wallace multipliers.

CARRY SELECT ADDER

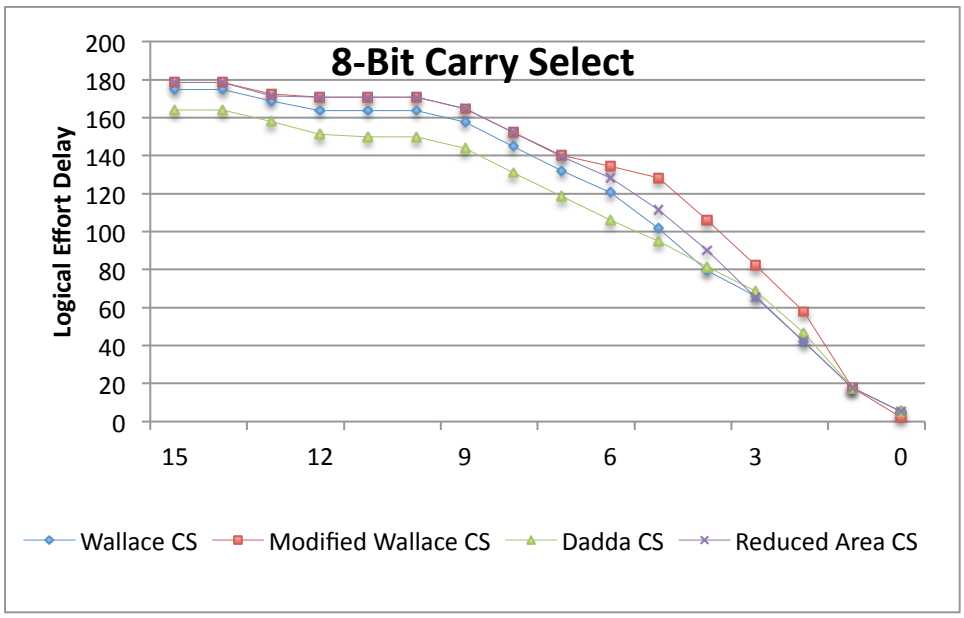

Figure 71: 8-Bit by 8-Bit Multiplier with Carry Select Final Adder

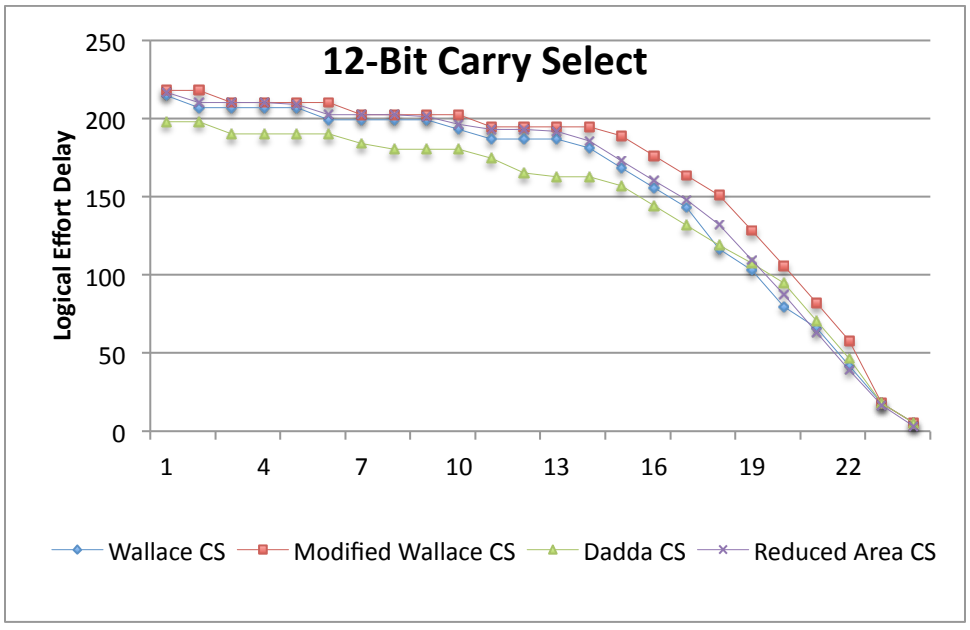

Figure 72: 12-Bit by 12-Bit Multiplier with Carry Select Final Adder

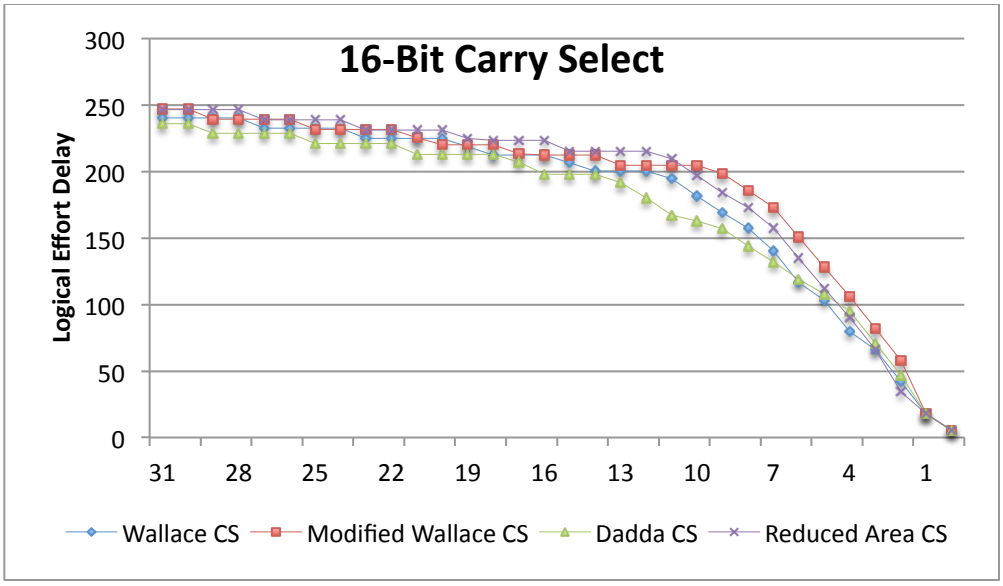

Figure 73: 16-Bit by 16-Bit Multiplier with Carry Select Final Adder

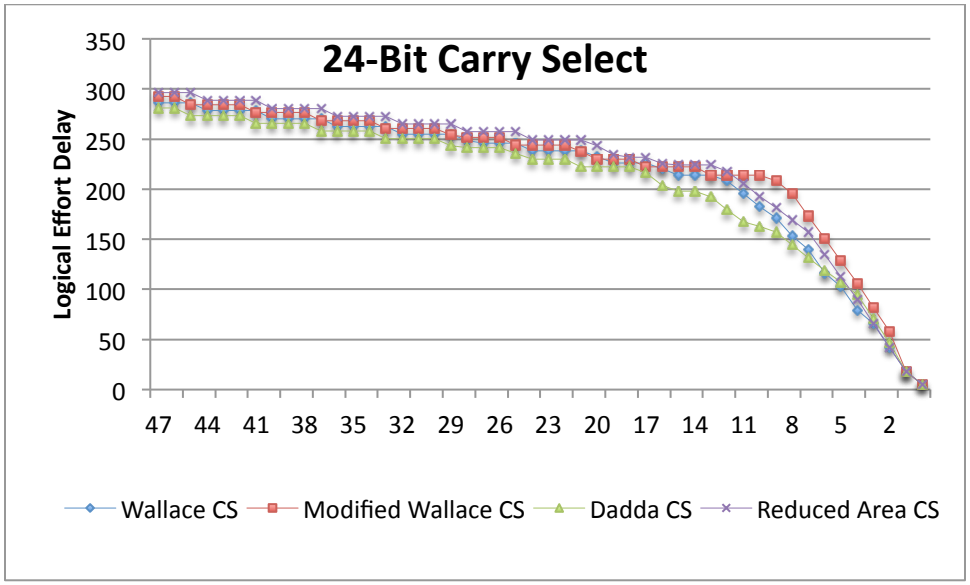

Figure 74: 24-Bit by 24-Bit Multiplier with Carry Select Final Adder

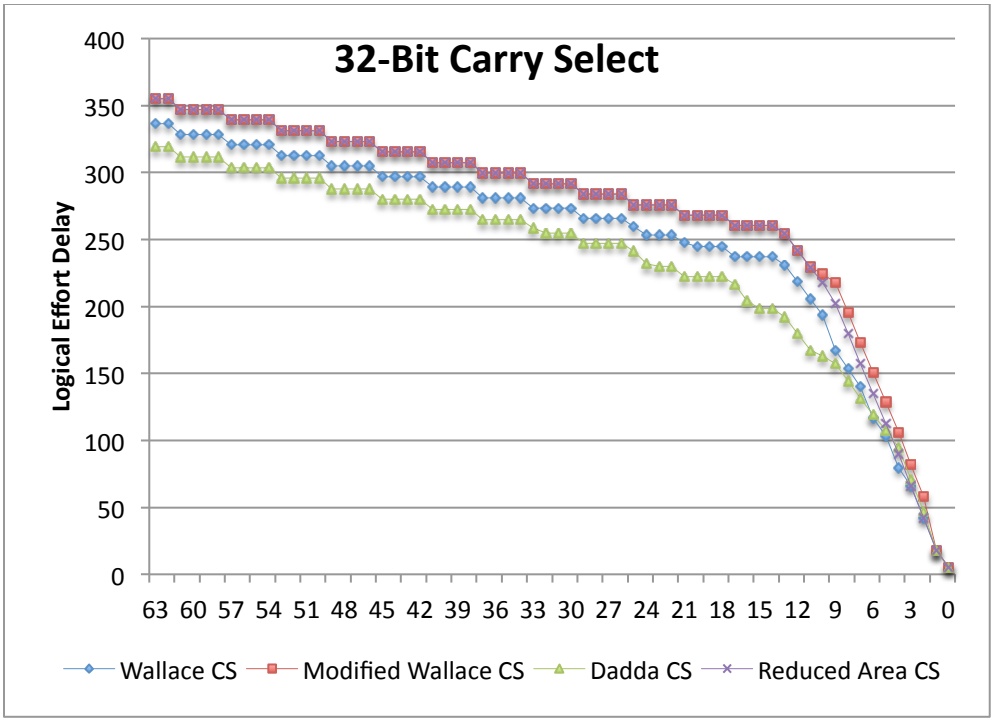

Figure 75: 32-Bit by 32-Bit Multiplier with Carry Select Final Adder

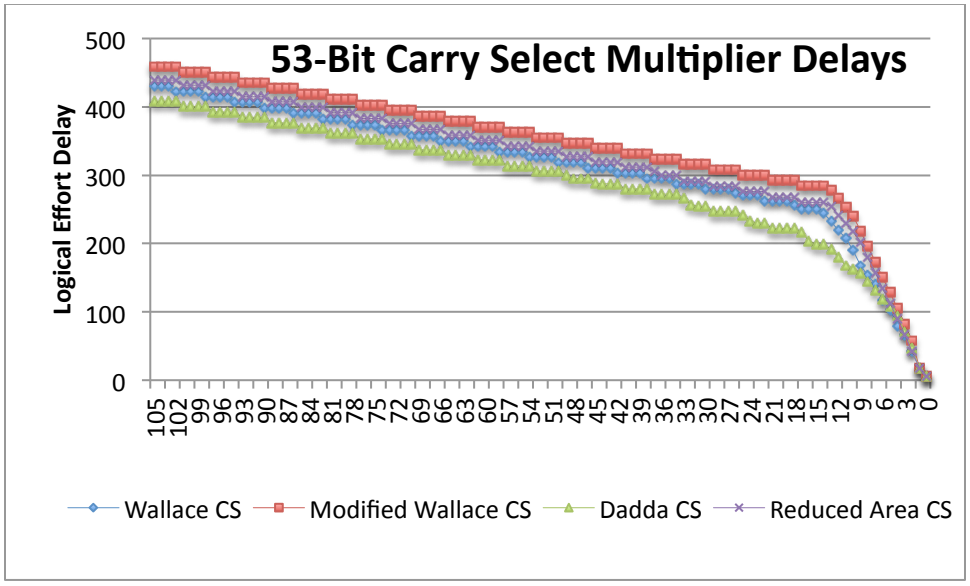

Figure 76: 53-Bit by 53-Bit Multiplier with Carry Select Final Adder

As with the multipliers that use ripple carry final adders, multipliers with carry select final adders based upon the Dadda reduction method provide better delay performance than Wallace, reduced area or modified Wallace multipliers. Wallace multiplier designs out performed reduced area and modified Wallace multipliers. In some cases, reduced area was faster than modified Wallace.

KOGGE-STONE PARALLEL PREFIX ADDER

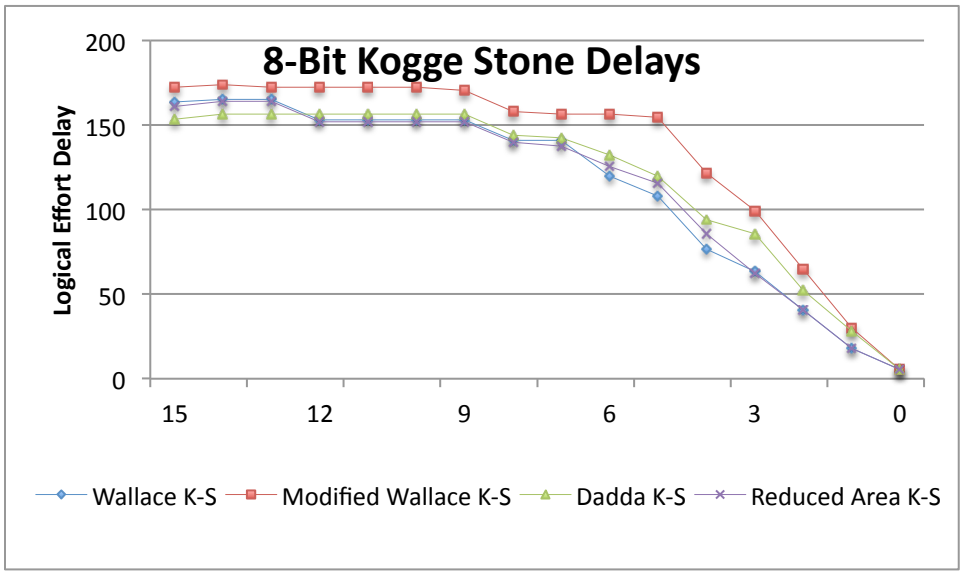

Figure 77: 8-Bit by 8-Bit Multiplier with Kogge-Stone Final Adder

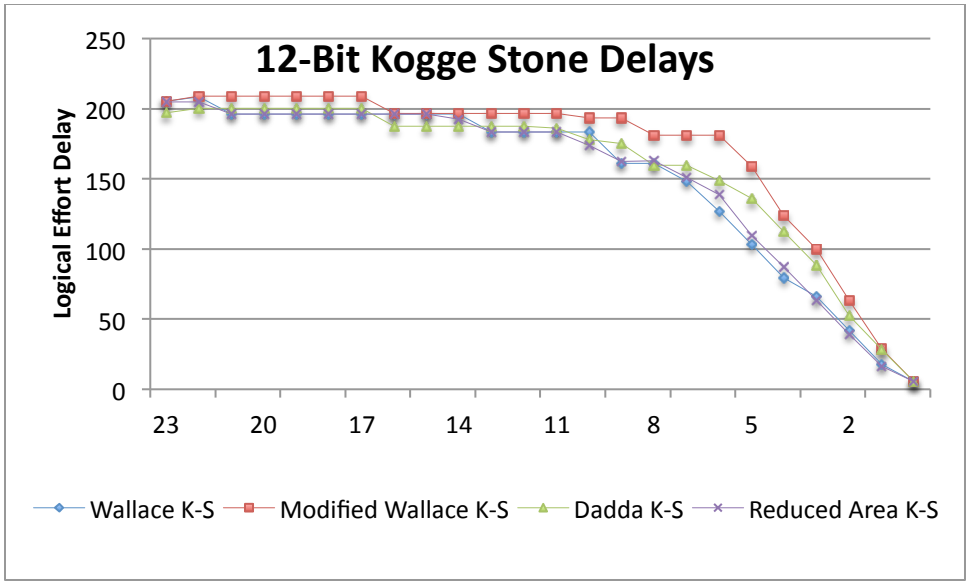

Figure 78: 12-Bit by 12-Bit Multiplier with Kogge-Stone Final Adder

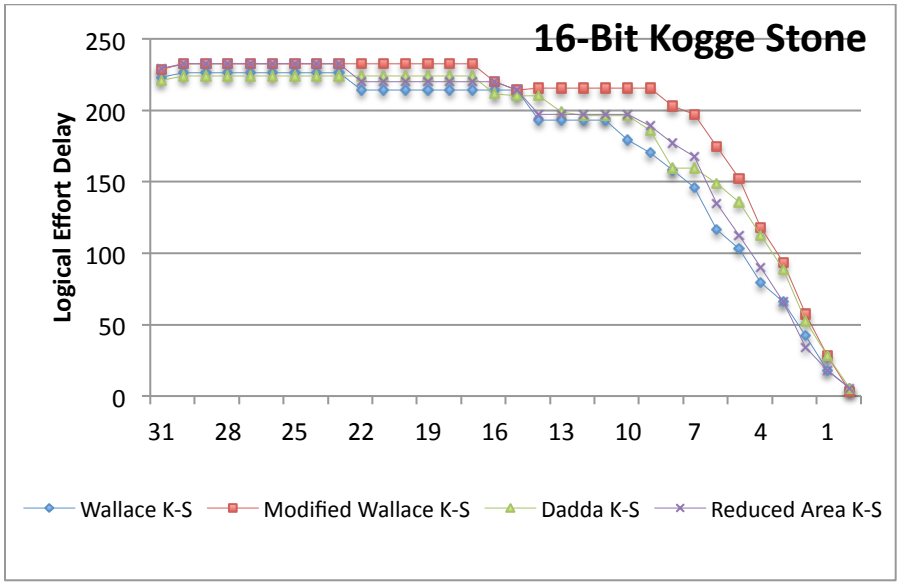

Figure 79: 16-Bit by 16-Bit Multiplier with Kogge-Stone Final Adder

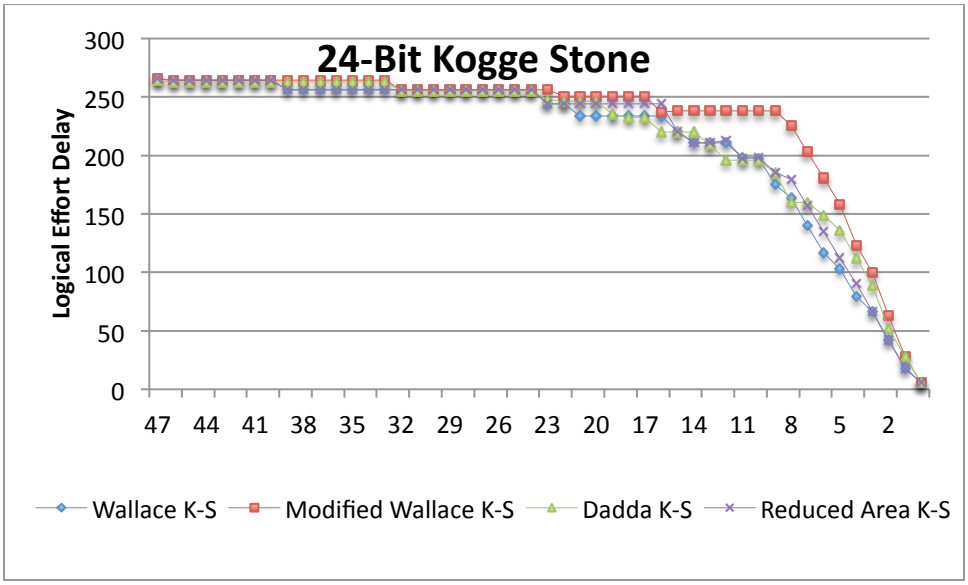

Figure 80: 24-Bit by 24-Bit Multiplier with Kogge-Stone Final Adder

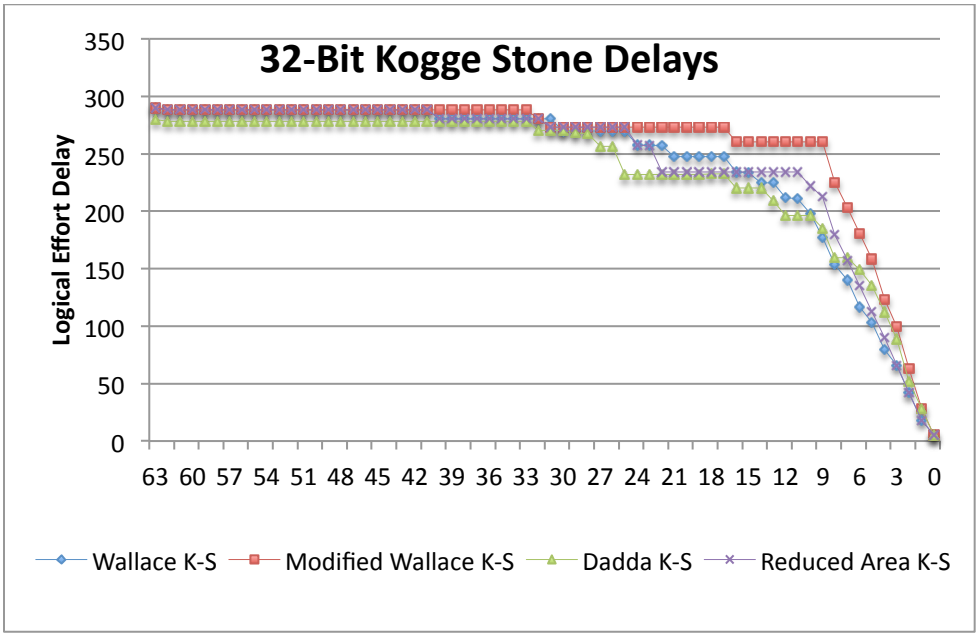

Figure 81: 32-Bit by 32-Bit Multiplier with Kogge-Stone Final Adder

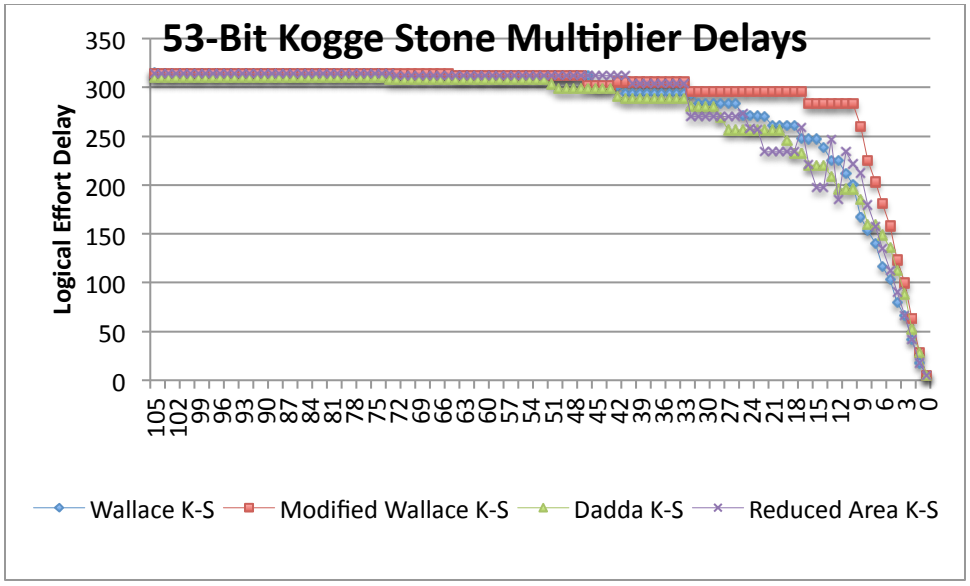

Figure 82: 53-Bit by 53-Bit Multiplier with Kogge-Stone Final Adder

As with the multipliers that use ripple carry final adders and carry select adders, multipliers with Kogge-Stone final adders based upon the Dadda reduction method provide better delay performance than Wallace, reduced area or modified Wallace multipliers. Wallace multiplier designs out performed reduced area and modified Wallace multipliers. In some cases, reduced area was faster than modified Wallace.

Chapter 11: Overall Delay Results

8-Bit Multipliers

Twelve 8-bit multipliers consisting of four column reduction methods (Wallace, Dadda, reduced area, and modified Wallace) were combined with three carry propagate adders (ripple carry, carry select and Kogge-Stone) and were analyzed. The following figure shows the maximum logical effort delay through each of these multiplier designs. Multiplying these numbers by the tau value for a given technology provides an estimate of the circuit's, time based performance.

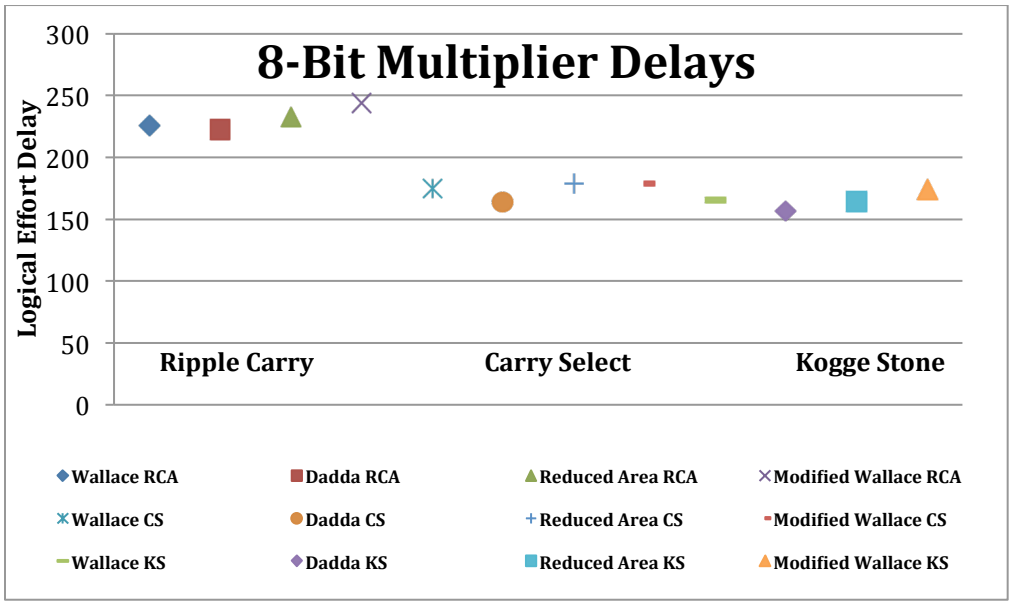

Figure 83: Logical Effort Delay for Twelve 8-Bit by 8-Bit Multipliers

For all three of the final adders, Dadda multipliers provide the best performance. Using the Dadda column reduction method, for the three carry propagate adders analyzed, always has the least delay through the multiplier and final adder. The Kogge-Stone final adder provides slightly better performance than the carry select final adder.

For the Kogge-Stone final adder, the logical effort delay is 157 units. Therefore, for a 0.35um technology, which has a tau value of approximately 20 psec, the time delay through the multiplier is calculated to be approximately 3.1 nsec. Considering 45nm technology, which as a tau value of 4.1 psec, the time delay through the multiplier is calculated to be approximately 0.64 nsec.

12-Bit Multipliers

For 12-bit multipliers, the Dadda multiplier reduction provided the highest performance for each of the carry propagate adders used. The carry select and Kogge-Stone carry propagate adders had the same performance. The following figure compares the relative logical effort delay for the twelve 12-bit multipliers analyzed.

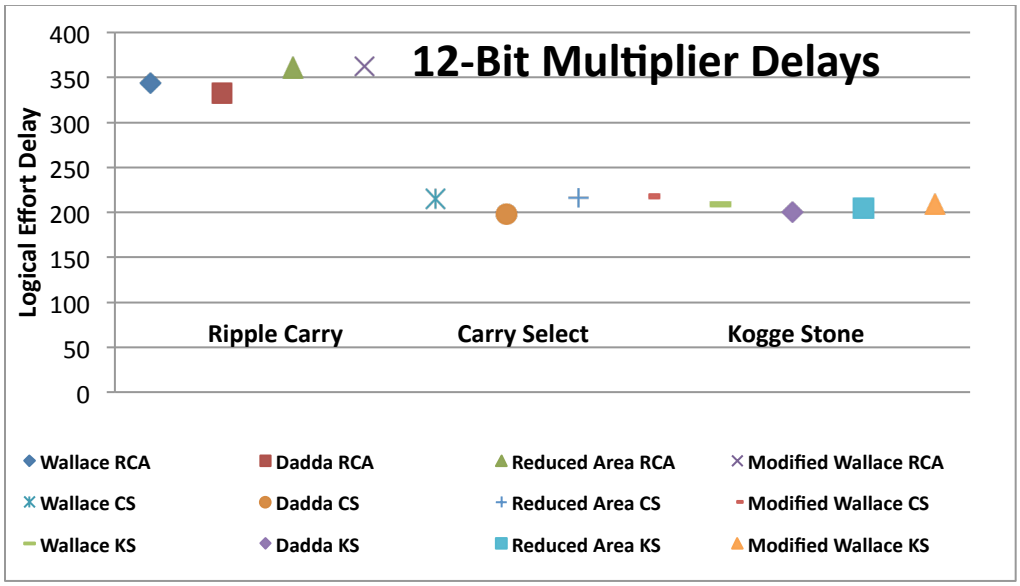

Figure 84: Logical Effort Delay for Twelve 12-Bit by 12-Bit Multipliers

For smaller width multipliers, there is little performance difference between using the carry select or Kogge-Stone final adder.

16-Bit Multipliers

The following figure shows the performance of twelve 16-bit multipliers. The left four points are for the multipliers with ripple carry final adders, the center four points show the performance of the four multipliers with carry select final adders and the right four points show the performance of the Kogge-Stone based implementations.

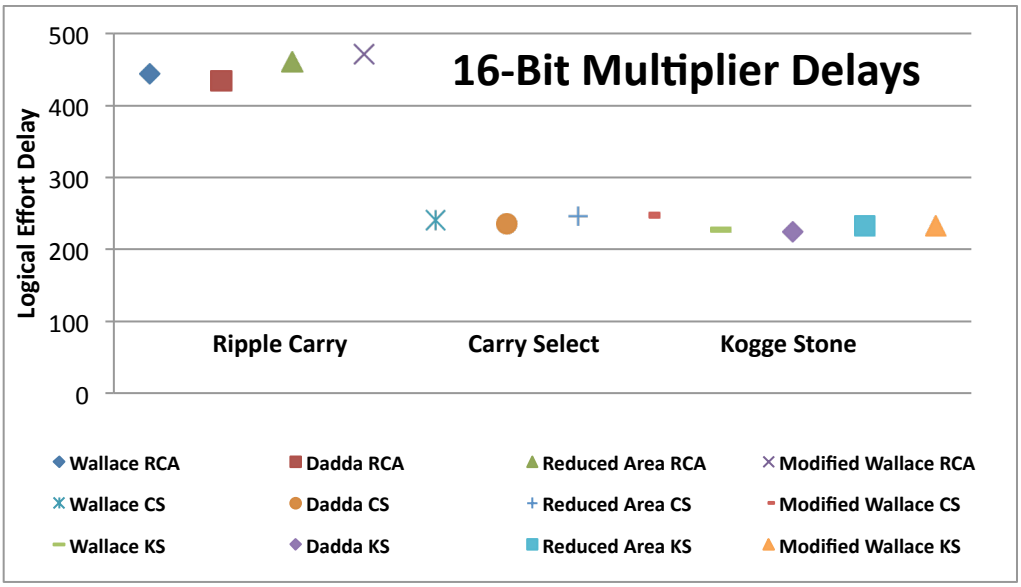

Figure 85: Logical Effort Delay for Twelve 16-Bit by 16-Bit Multipliers

For 16-bit multipliers, the Dadda column reduction multiplier using a Kogge-Stone carry propagate adder is always the fastest, though only marginally faster than a carry select final adder implementation.

24-Bit Multipliers

The following figure shows the performance of twelve 24-bit multipliers. The left four points are for the multipliers with ripple carry final adders, the center four points

show the performance of the four multipliers with carry select final adders and the right four points show the performance of the Kogge-Stone based implementations.

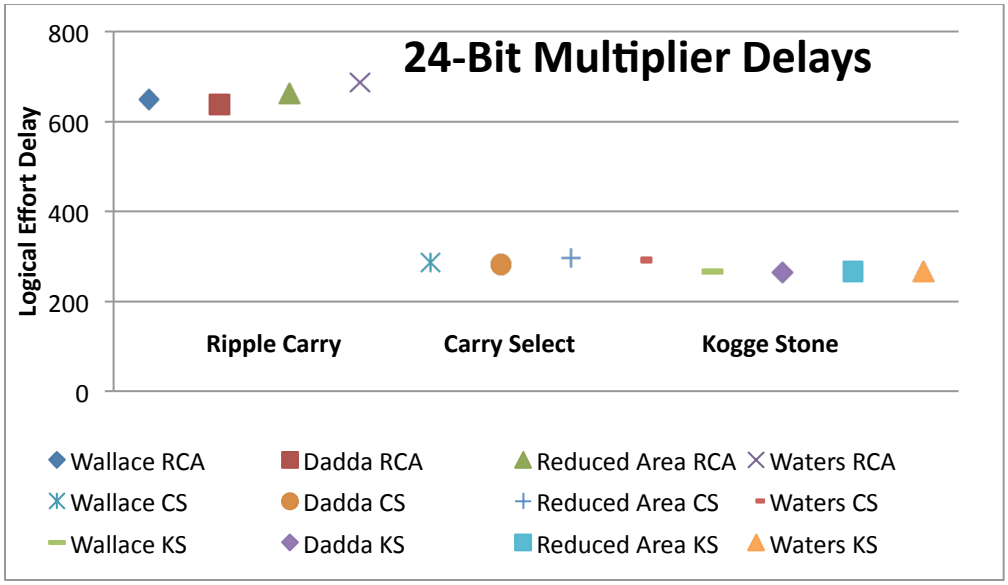

Figure 86: Logical Effort Delay for Twelve 24-Bit by 24-Bit Multipliers

Using the Dadda column reduction method along with Kogge-Stone CPA provides slightly better performance than the other column reduction methods. Dadda, for ripple carry and carry select is superior.

32-Bit Multipliers

The following figure shows the performance of twelve 32-bit multipliers. The left four points are for the multipliers with ripple carry final adders, the center four points show the performance of the four multipliers with carry select final adders and the right four points show the performance of the Kogge-Stone based implementations.

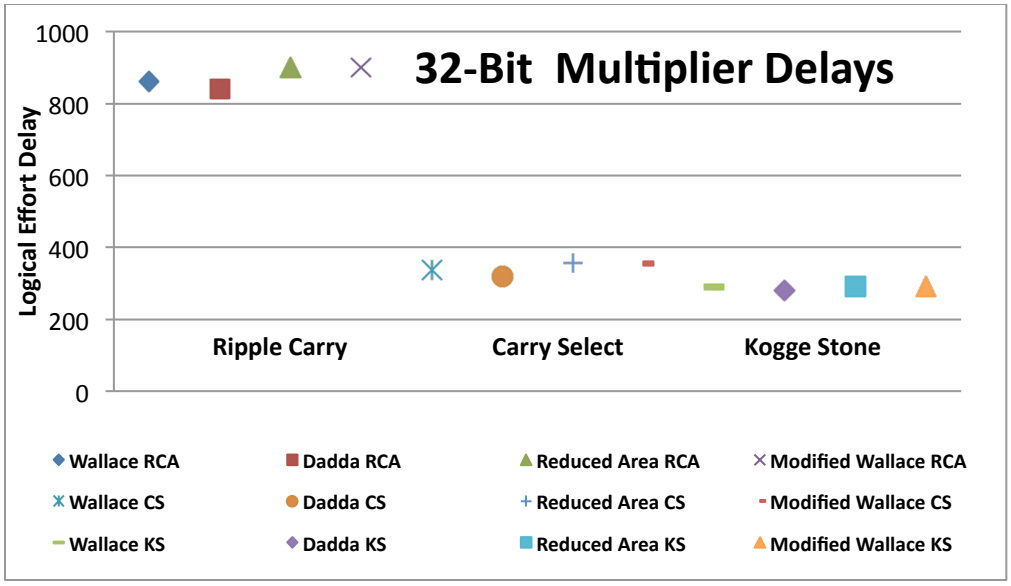

Figure 87: Logical Effort Delay for Twelve 32-Bit by 32-Bit Multipliers

As with smaller multipliers, Dadda column reduction provides faster performance. With wider multipliers, the difference between carry select and Kogge-Stone widens. The Dadda based multiplier with Kogge-Stone CPA is the faster 32-bit multiplier.

53-Bit Multiplier

The following figure shows the performance of twelve 53-bit multipliers. The left four points are for the ripple carry adders, the center four points show the performance of the four carry select based multipliers and the right four points show the performance of the Kogge-Stone based implementations.

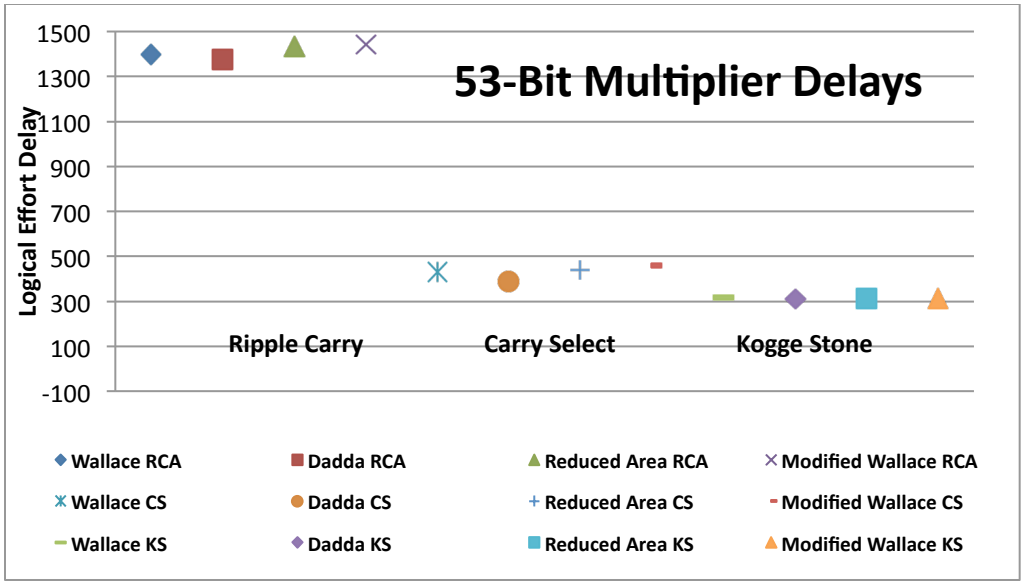

Figure 88: Logical Effort Delay for Twelve 53-Bit by 53-Bit Multipliers

Since there is minimal speed difference, down the column reduction section, between Dadda and the other three column reduction implementations, Kogge-Stone based CPA is the fastest implementation, but, the performance for any of the four reduction methods was essentially the same.

The following table lists the logical effort delays for all multiplier widths (8-bit, 12-bit, 16-bit, 24-bit, 32-bit and 53-bit), column reduction methods (Wallace, Dadda, reduced area and modified Wallace) and carry propagate adders analyzed (ripple carry, carry select and Kogge-Stone parallel prefix adders).

Table 14: Summary of Logical Effort Delays for all Multipliers

Wallace	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	226	344	444	649	862	1398
Carry Select	175	215	240	286	336	430
Kogge-Stone	165	209	227	266	290	315

Modified Wallace	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	244	362	471	687	900	1444
Carry Select	178	218	247	292	355	459
Kogge-Stone	165	209	232	266	290	314

Dadda	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	223	332	434	638	839	1376
Carry Select	164	198	236	281	319	409
Kogge-Stone	157	200	224	264	280	310

Reduced Area	8-bit	12-bit	16-bit	24-bit	32-bit	53-bit
Ripple Carry	232	361	461	662	900	1434
Carry Select	178	217	247	296	355	439
Kogge-Stone	164	205	232	266	290	314

With one exception, the 12-bit Dadda multiplier, the Kogge-Stone carry propagate adder provides the fastest worst case delay. For the 12-bit Dadda multiplier, the carry select carry propagate adder is 1% faster.

Chapter 12: Comparing Dynamic Power

Power dissipation, both static and dynamic is a function of the technology being used and the transistor sizes that are used in the circuit implementation. However, it is possible to make some comparisons for dynamic power dissipation for the multipliers reviewed in this research. Dynamic power is the result of outputs driving subsequent input capacitances and driving output capacitances. In Logical Effort, the electrical effort, h, the logical effort, g, and the parasitic delay, p, are all referenced to an inverter and are parameters that relate to some capacitance load. In [32], Kabanni has developed a logical effort based power model where the normalized switching power of a gate is:

$$P_{nm} = \alpha_{nm} Z(gh + p)$$

Where P_{nm} is the normalized switching power of a gate, α_{nm} is the normalized gate activity factor and Z is a constant that represents the size of a gate as compared to its template. Since the transistor width ratios are not known until a given technology is chosen, Z cannot be determined, but for a given technology, the normalized power of a gate is proportional to its activity, and the three logical effort variables, g, h and p. In this research, h has been set to 1 as the input and output capacitances are set to the same. Therefore, for a given activity factor, the normalized power of a gate is proportional to g and p.

$$P_{nm} \propto (g + p)$$

Therefore, by summing the g and p values for the circuits in a given design, the relative normalized power dissipation, for the same activity factor, can be determined for the multiplier designs in this research.

Both 8-bit and 53-bit Dadda column reduction multipliers using the three carry propagate adders were analyzed with this relative power estimation. First the total

multiplier power was estimated, separating the carry propagate power from the column reduction power. The activity factor for the multipliers are not considered as it is very dependent upon terms being multiplied. For example, if the multiplier were changed from 0 x 0 to 0 x 0 (no change from cycle to cycle), then the activity factor would be zero. However, if it were changed from 0 x 0 to 1 x max, where max represents the largest value expressed by a multiplier (all one's), then the activity factor would be very high (near one).

8-Bit Dadda Multiplier Power Estimation

The following figure shows the column reduction power for an 8-bit Dadda multiplier column reduction along with the power contributed by the carry propagate adders. Note that the Kogge-Stone carry propagate adder with the most power consumption, only uses about 15% of the total power for the multiplier.

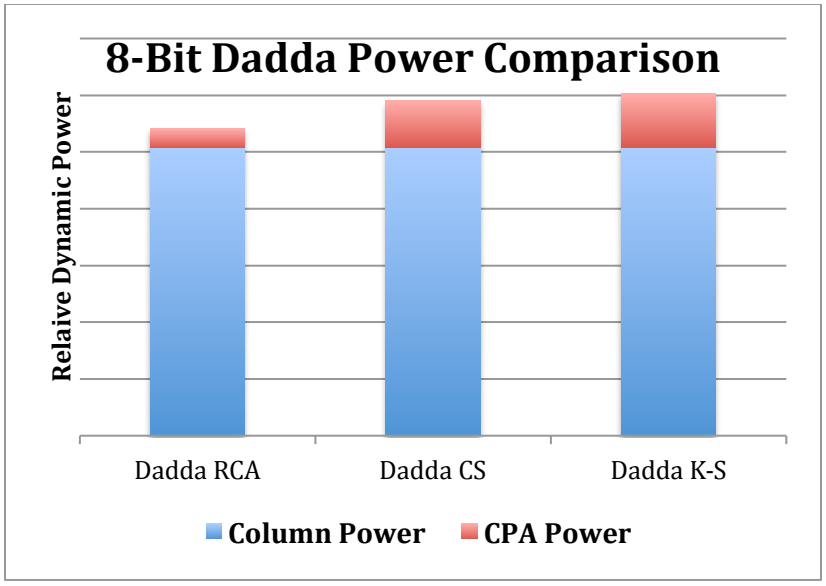

Figure 89: Relative Power for 8-Bit Dadda Multipliers

Since the column reduction power will be the same for each design, the following figure shows only the relative power dissipation for each of the carry propagate adders.

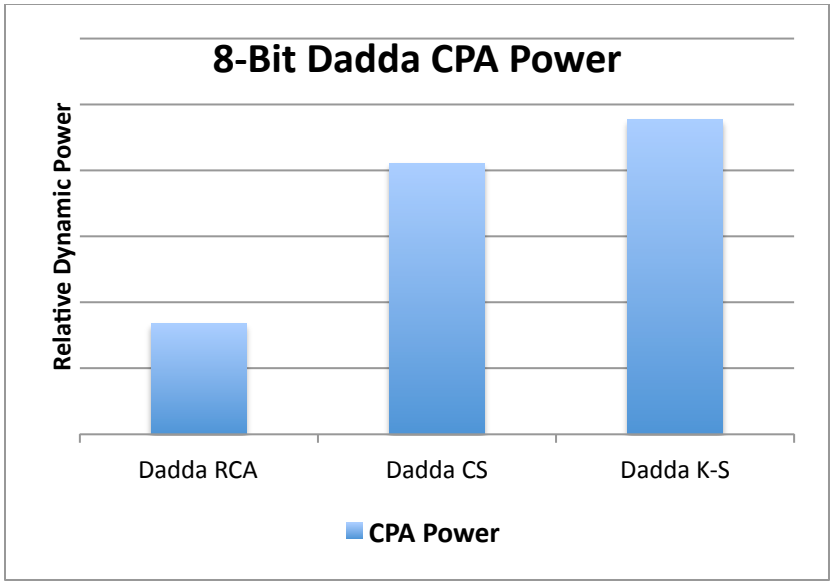

Figure 90: Relative Power for each CPA used in 8-Bit Dadda Multiplier

The Kogge-Stone carry propagate adder uses only about 20% more power than the carry select final adder, but about three times as much as the ripple carry final adder.

Table 15: Design Data for 8-Bit Dadda Multipliers

8-Bit Multiplier	Gates	Transistors	Sum G	Sum P	Power Est
Dadda RCA	144	470	135	202	107%
Dadda CS	383	1236	359	462	116%
Dadda K-S	231	747	510	446	119%
Dadda Column	548	1770	1878	3202	100%

The power estimate for the 8-bit Dadda multiplier is normalized to the power estimated for the column reduction section only (100%). The power is estimated by summing the g and p logical effort values for the column reduction section and each of

the carry propagate adders. The ripple carry final adder adds 7% more power than the power estimate for the 8-bit Dadda column reduction. The carry select final adder adds 16% more power. The Kogge-Stone final adder represents about 19% estimated power above the dynamic power of the 8-bit Dadda column reduction stages.

53-Bit Dadda Multiplier Power Estimation

For the 53-bit multiplier, almost all of the dynamic power is dissipated in the column reduction. The power dissipation of the carry propagate adder is small by comparison. The following figure illustrates the relative power dissipation for each of the three designs, breaking out the column reduction power from the carry propagate adder component.

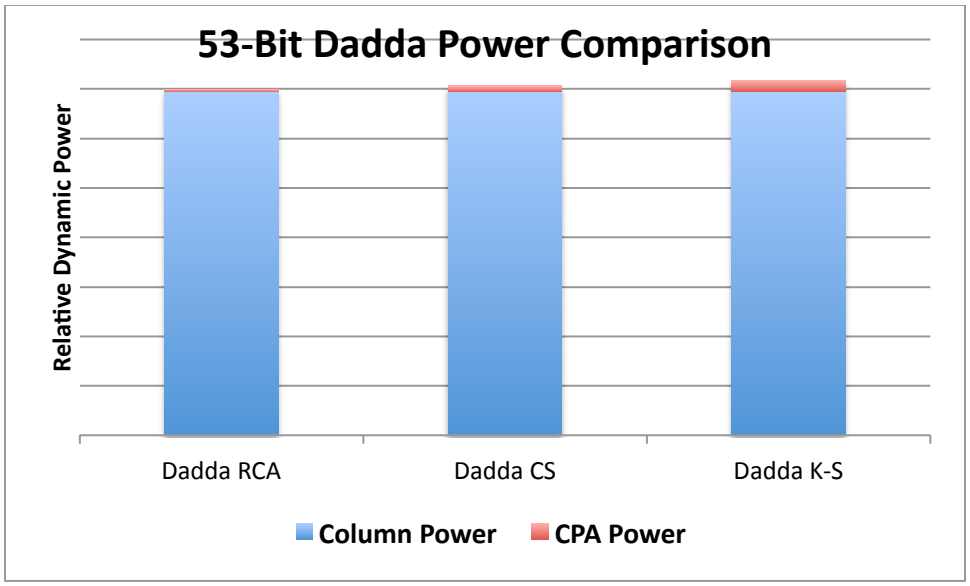

Figure 91: Relative Power for the 53-Bit Dadda Multipliers

The following figure compares only the carry propagate adder power for the three adder designs.

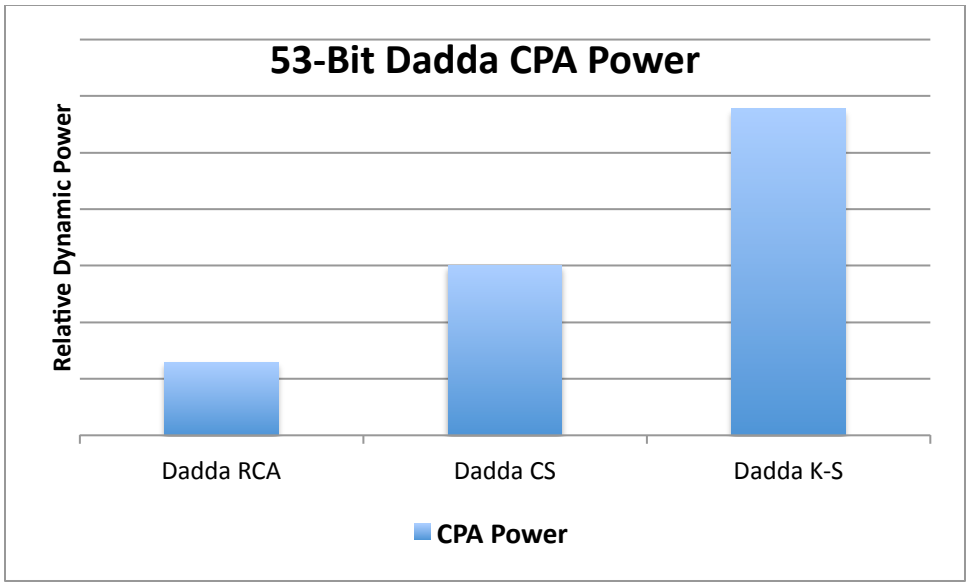

Figure 92: Relative Power for each CPA used in the 53-Bit Dadda Multiplier

For a 53-bit carry propagate adder, Kogge-Stone uses about six times the power of ripple carry or twice the carry select power.

The following table lists the gate counts, number of transistors and summations of the logical effort term, g and p, for the 53-bit multipliers.

Table 16: Design Data for 53-Bit Dadda Multipliers

53-Bit Multiplier	Gates	Transistors	Sum *g*	Sum *p*	Power Est
Dadda RCA	1138	3726	1035	1552	101%
Dadda CS	2808	9048	2608	3387	102%
Dadda K-S	2688	9175	6173	5374	103%
Dadda Column	34478	111390	127038	220327	100%

From this analysis, the column reduction portion of the 53-bit multiplier uses most of the power. The power estimation is normalized to the column reduction portion of the multiplier (100%). The ripple carry final adder only adds about 1% dynamic power

to the design. The Kogge-Stone carry propagate adder is faster than the carry select adder, and has about 60% higher power dissipation as compared to the carry select adder, but both are negligible (at most 3%) in comparison to the power consumed by the column reduction section.

Table 17: Power Estimate Summary for 8-Bit and 53-Bit Multipliers

Dadda Multiplier	8-bit	53-bit
Column Reduction Power Est.	100%	100%
Multiplier with Ripple Carry	107%	101%
Multiplier with Carry Select	116%	102%
Multiplier with Kogge-Stone	119%	103%

By adding all of the *g* and *p* logical effort terms for a given multiplier design, it is possible to estimate the relative dynamic power dissipated by each design. The activity factor for the four column reduction methods is assumed to be the same for a given multiplication. For smaller multipliers, the final carry propagate adder has a significant dynamic power contribution, for the 8-bit analysis it ranged from 7% to 19%. For larger multiplier widths, however, the dynamic power dissipation in the final carry propagate adder, regardless of type, was negligible at less than 3%.

Chapter 13: Conclusions

Much work has been done in analyzing the impact of the final carry propagate adder on multiplier performance. It has, however, been limited to individual multiplier widths without a comprehensive analysis of various widths. Also, much of the analysis has been measured in gate delay counts or in equivalent XOR delays. Little comparative research has been done in the analysis of column reduction methods and their impact on multiplier performance.

COLUMN REDUCTION METHOD SELECTION

Column reduction techniques, such as Dadda, that minimize the delays in the column of reduction stages are preferred as they present smaller delays to the CPA for final summation. All of the current column reduction techniques have non-uniform arrival times with the longer times in the central bits of the column reduction stages. Minimizing the delays through the center of the multiplier will have significant positive performance impact. In this research, Dadda was the fastest column reduction method.

MINIMIZING COLUMN DELAY BY TERM SELECTION IN DADDA

Because, mostly, all terms are used in subsequent reduction stages in all reduction methods except for Dadda, the term delays in a reduction stage track each other. That is, they have been input into a similar number of adders and have accumulated similar delays through the same number of reduction stages. That is not the case for Dadda multipliers which do the minimum amount of reduction possible from stage to stage, only to ensure that the same number of reduction stages as Wallace are met. Consequently, since the delays from the inputs to carry and sum on an adder are different, there are opportunities to selectively group terms in order to minimize the delays through the column reduction portion of Dadda multipliers. More importantly, the inputs to sum

outputs of full adders have significantly more delay than the inputs to outputs of half adders or the inputs to carry outs of full adders. For the Dadda multiplier, the opportunity to group terms is in the second reduction stage. The second stage groups terms from the previous, first, reduction stage and reduces the number of rows by $\lfloor 3 \bullet successor\ height/2 \rfloor$ where $\lfloor x \rfloor$ denotes the integer portion of x. For the multipliers analyzed the improvement by grouping terms to minimize the delay varies from 1% to 11%, depending upon the multiplier width. The important consideration is the number of full adder terms in the first reduction stage as compared to the number of terms that have been passed through from the partial product array or the outputs of half adders. The following table lists the number of terms in the first reduction stage for the multipliers that are outputs of full adders, in the center column, versus terms that are passed down from the partial product array or are outputs of half adders.

Table 18: First Reduction Stage Terms and Dadda Column Delay Improvement

First Stage	8-Bit	12-Bit	16-Bit	24-Bit	32-Bit	53-Bit
Full adder outputs	2	3	4	8	6	20
Half adder or partial product outputs	4	6	9	11	22	22
Ratio full adders/HA or PP outputs	2.0	2.0	2.3	1.4	3.7	1.1
Percent improvement in Dadda	11%	7%	10%	1%	4%	1%

In all cases analyzed, the Dadda multiplier with term selection yielded the fastest multiplier. For Dadda column reduction where term grouping was not performed, that is the terms were applied to adders based upon adjacency as with the other column reduction methods, the speed through the Dadda column reduction was the same as for the other three reduction methods.

The following figure illustrates a delay heat map for a 12-bit Dadda multiplier with selective term grouping (left side) and without selective term grouping (right side).

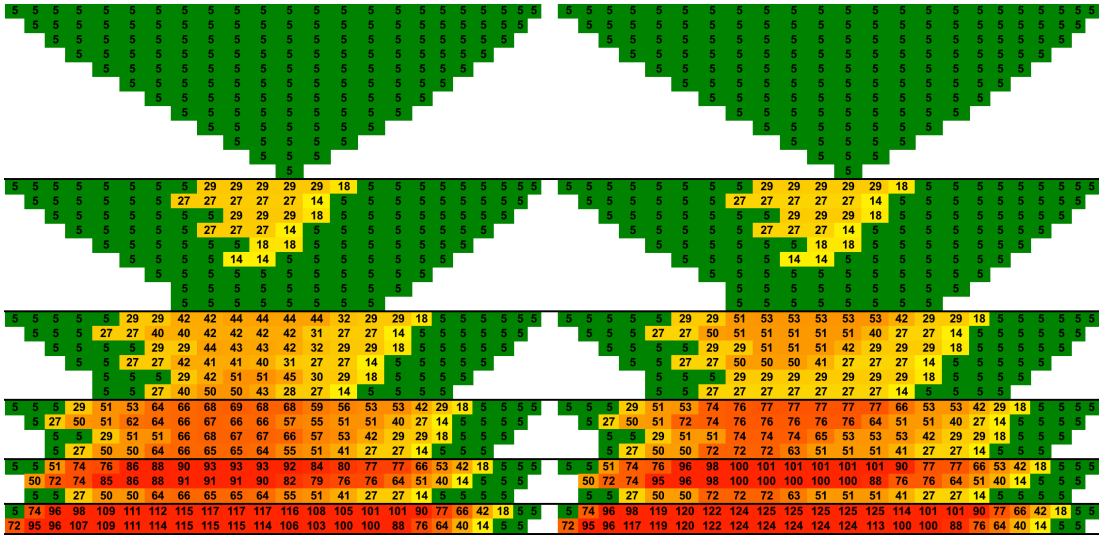

Figure 93: 12-Bit Dadda Multiplier Heat Maps

The least delay values are green, starting with the partial product AND array at the top and progress through yellow, orange and red as the delay values increase down the columns. The selective term grouping in the second reduction stage may be seen in the values for the delays as well as the heat map shading differential.

CARRY PROPAGATE ADDER SELECTION

Column multipliers can be considered as having three Regions. Region 1 is the rising delay on the LSB side of the reduction column. Region 2 is the center portion of the column reduction where the delay from column to column is relative flat. Region 3 is the MSB side of the column reduction where the column delays are tailing off. Those three regions are illustrated in the following figure showing the column reduction profile for a 32-bit by 32-bit column reduction multiplier using the Wallace reduction approach.

This delay profile is due to there not being many partial product terms on the LSB and MSB sides of the multiplier. Consequently, there are few reduction stages, which are delays through half or full adders that increase the overall delay through the multiplier paths for the LSB and MSB columns.

Since full adders use three input terms and since the sum of the full adder has a longer delay than the carry out, there is the opportunity to use a long delay term (from the sum of a previous full adder) to drive the carry input of the full adder on the subsequent reduction stage. This results in a shorter delay through the subsequent full adder than if the long delay term was applied to input A or B. Therefore, by taking one long delay term and applying it to the carry in and two shorter delay terms and applying them to the A and B inputs of the subsequent adder, the delay is minimized. However, this requires that there be two short delay terms for every long delay term. As can be seen in the previous table, when the ratio of short delay terms (partial product or half adder outputs) is 2 or more, the potential to improve delay through the Dadda column reduction is higher. In the case of 24-bit and 53-bit multipliers, there are insufficient low delay values in order to pair them with the outputs of full adders in order to minimize the delay through the second reduction stage.

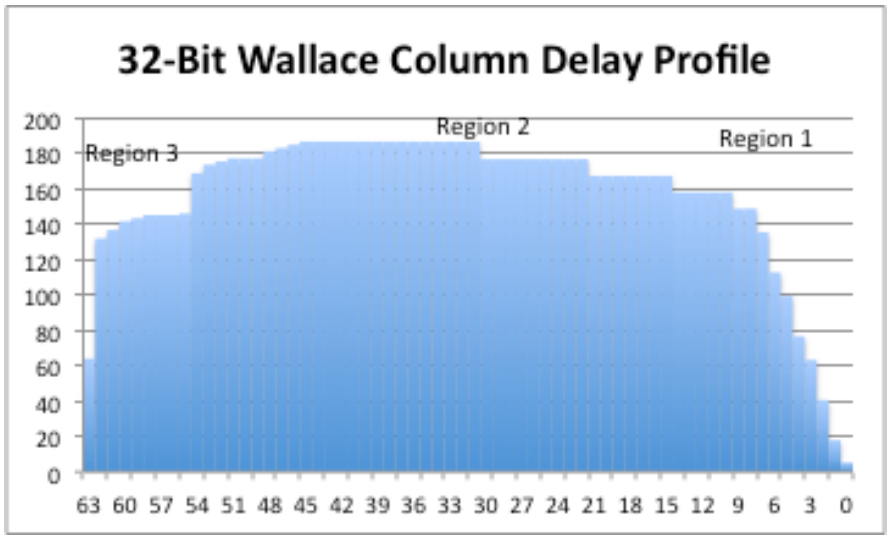

Figure 94: Regions in 32-Bit by 32-Bit Wallace Multiplier

For all the design implementations used in this research, the delay build from least significant bit toward most significant in region 1 is steeper than the delay build through the ripple carry adder which is the slowest carry propagate adder, Therefore, improving carry propagate performance will not improve performance as the circuit will be waiting for the delay times down the least significant columns. Therefore, a slow carry propagate adder, such as a ripple carry adder is sufficient for region 1.

Wider designs have a much broader region 2 and significant focus on minimizing the carry propagation through these bits in region 2 is essential for optimal design. Carry propagate adders such as carry select adders and Kogge-Stone adders are applicable in region 2.

For region 3, where the delays from the column reduction are “tailing off”, extending the carry propagate from Region 2 is advisable since each bit significance arrives sooner than the previous bit in Region 3 and is ready to be processed much earlier

than the arrivals of the results through the carry propagate adder. As a result, in region 3, performance is dependent upon quickly adding through the final carry propagate adder.

It is likely that optimal multiplier performance is achieved through the use of a hybrid carry propagate adder with different adder designs for the several regions of the multiplier. The use of ripple carry adders for region 1 and using fast carry propagate adders such as carry select or Kogge-Stone adders for region 2, and extending across region 3 will provide the best delay performance. To perform a preliminary assessment of this hypothesis, a 32-bit Dadda multiplier was designed with the ten least significant bits being added with a ripple carry adder and the remaining bits of the carry propagate adder being added using the Kogge-Stone carry propagate adder. The comparison of a 32-bit Dadda fully using a Kogge-Stone carry propagate adder and this 10-bit ripple carry adder followed by a 53-bit Kogge-Stone carry propagate adder is illustrated in Figure 95.

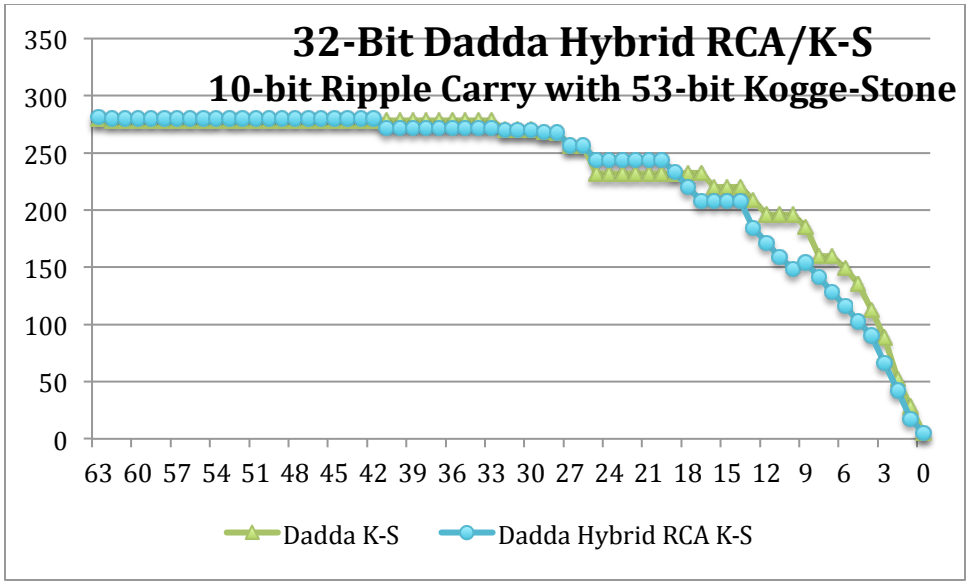

Figure 95: 32-bit Dadda Multipliers Comparison with Kogge-Stone and Hybrid

While the use of a hybrid carry propagate adder improved delay in the least significant bits for the 32-Bit Dadda multiplier, the delay efficiency of Kogge-Stone in later bits results in the overall worst case delays being the same. Therefore, a hybrid carry propagate adder could be used in order to reduce the final adder complexity, but, there is not improvement in worst case delay performance.

As multipliers become larger, the percent of delay contributed by the CPA becomes much larger and there is a wider region 2 where faster carry circuitry is critical.

REDUCING LSB SIDE DELAY

For the modified Wallace multiplier, the second most LSB partial product does not get initially reduced as with Wallace or reduced area. Figure 95 highlights that, as a result, the final carry propagate adder must be longer than for Wallace or for reduced area. While the carry propagate adder for modified Wallace reduction is as long as for an equivalent Dadda multiplier, the modified Wallace reduction method does not have the opportunity to use selective term grouping as with Dadda. Consequently, the modified Wallace reduction method will always be the slowest multiplier for a given size and carry propagate adder.

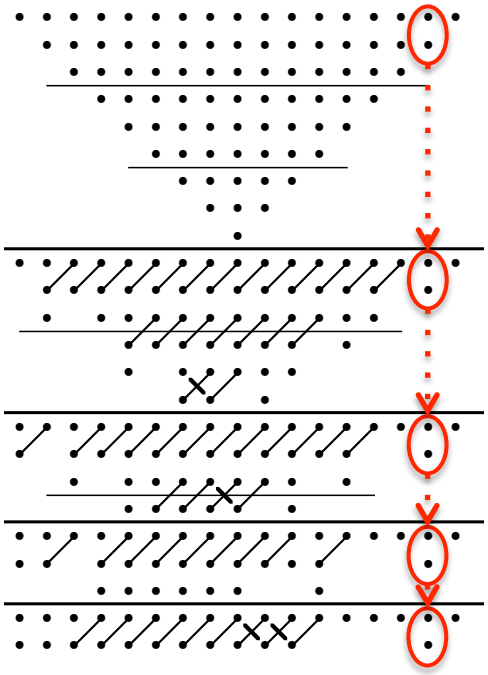

Figure 96: Modified Wallace Multiplier Illustrating Reduction Issue

ADDER SELECTION

Adder selection will obviously impact multiplier performance as well. Selection of the adder based upon the input to output gate delays will not necessarily yield the fastest multiplier. Using the three full adders reviewed in an earlier chapter, a 32-bit by 32-bit Dadda multiplier with a Kogge-Stone carry propagate adder was designed utilizing each of the full adders, the following figure illustrates the performance of the three designs.

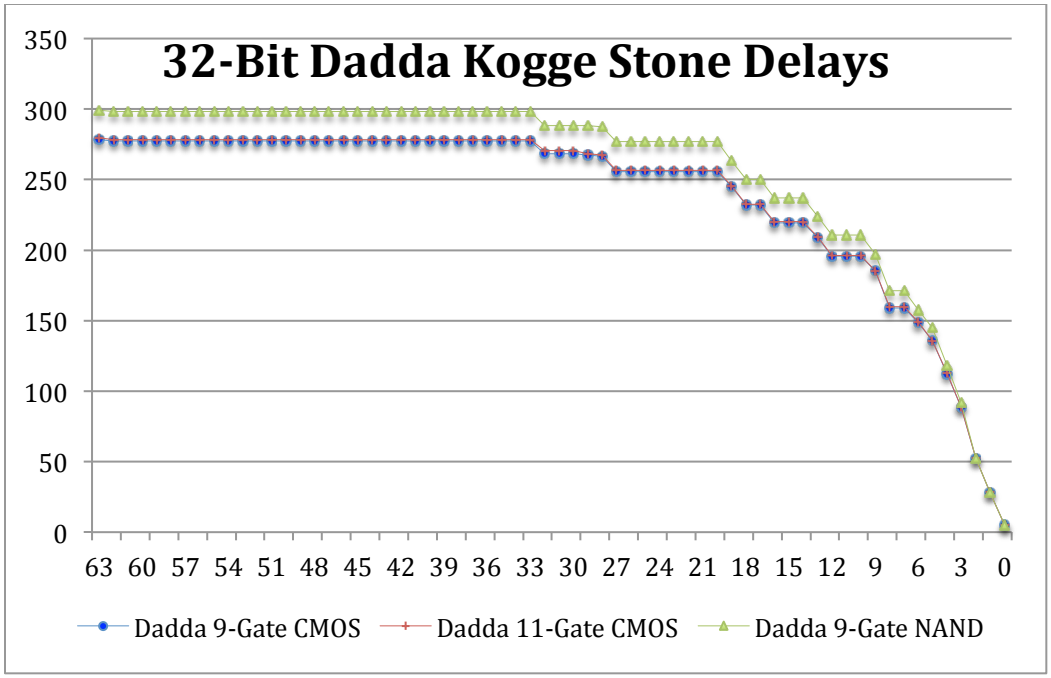

Figure 97: 32-Bit Dadda Multiplier using K-S CPA for three Different Full Adders

The 9-Gate CMOS and 9-Gate NAND full adders have the same gate delay counts from inputs to outputs. However, their performance differs due to the fan-out or branching effort of the 9-Gate NAND gate implementation. For the fastest multiplier analyzed, Dadda reduction with Kogge-Stone carry propagate adder, the 11-gate and 9-gate CMOS adders had nearly the same performance, but have different gate delay counts.

SIMPLIFYING THE COLUMN REDUCTION LOGICAL DELAY ESTIMATION

During the analysis of the column reduction multipliers for the eleven gate CMOS full adder, it was observed that the ratio of the logical effort delay divided by the gate delay in the center reduction columns was between 3.4 and 3.5. Therefore, it is feasible to estimate the gate delay count in a circuit path in the column reduction portion of the multiplier and simply multiply it by 3.5 in order to estimate the logical effort delay down

the columns. This simplified analysis is only applicable, however, for the column reduction section of the multiplier design. It does not apply to delay estimations for the final carry propagate adder. The following figure shows the ratio of logical effort delay divided by gate count in a column path, for a 16-bit by 16-bit multiplier, for the two terms that drive the final carry propagate adder, for the four column reduction methods explored.

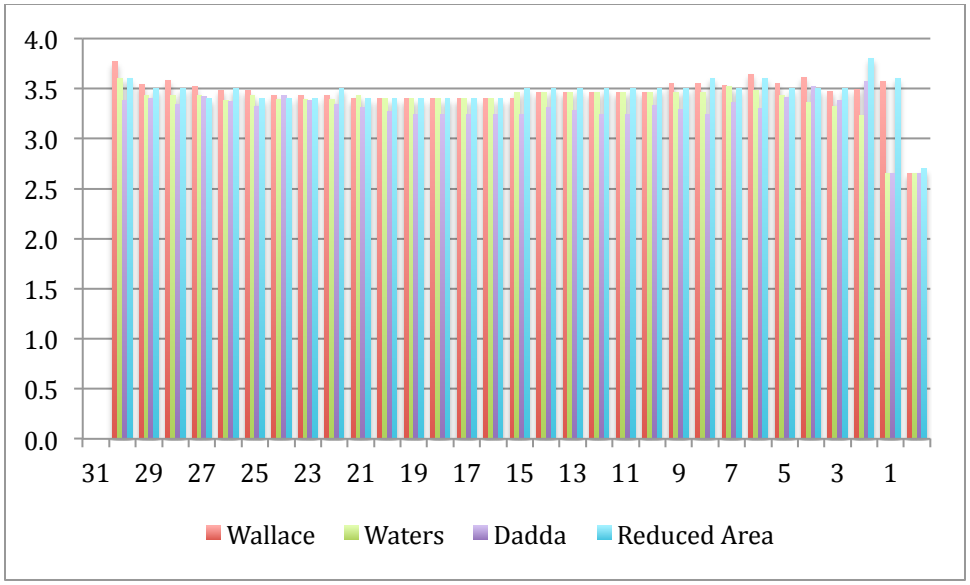

Figure 98: Ratio of Logical Effort Delay/Gate Delay for 16-Bit Multiplier

In Figure 98, the ratio of the logical effort divided by the gate delay count for each of the columns is shown for all four column reduction methods. Multiplying the gate delay count by a factor of 3.5 would provide a reasonably accurate estimation of the logical effort delay through the column reduction section of the multipliers. This 3.5 ratio occurs for all of the multipliers analyzed using the eleven gate full adder. Since the center

of the multiplier reduction is the slowest, the higher ratio factors on the MSB and LSB sides may be ignored as the center columns are the slowest paths through the multiplier.

Table 19: Table of LE/Gate delay ratio for 16-Bit Column Reductions

LE Delay/Gate Delay Ratio	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Wallace		3.8	3.5	3.6	3.5	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.6	3.6	3.6	3.5	3.5	3.6	2.7
Modified Wallace		3.6	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.4	3.4	3.3	3.2	2.7	2.7
Dadda		3.4	3.4	3.3	3.4	3.4	3.3	3.4	3.4	3.3	3.3	3.3	3.2	3.2	3.2	3.2	3.2	3.3	3.3	3.2	3.2	3.3	3.3	3.2	3.4	3.3	3.4	3.5	3.4	3.6	2.7	2.7
Reduced Area		3.6	3.5	3.5	3.4	3.5	3.4	3.4	3.4	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.6	3.5	3.6	3.5	3.5	3.5	3.8	3.6	2.7

The data used for Figure 97 is shown in Table 19. The MSB position, bit 31 has no data as the Dadda column reduction does not force a carry into the MSB column.

For the nine gate CMOS full adder, an analysis was done for a 32-bit Dadda multiplier. There was also a near constant ratio, however, it was higher at 3.9. Using the nine-gate NAND based full adder, a 32-bit Dadda multiplier was analyzed and the logical effort to gate count delay ration was an average of 4.3. While the two nine gate full adder implementations have the same number of gate delay counts, the ratios for logical effort to gate delay count are different. This is due to the higher branching efforts in the nine NAND gate implementation.

For a given full adder design, there appears to be a common logical effort to gate delay count ratio that may be determined.

The delay of a column reduction multiplier can be improved by doing three things. First, select a column reduction method that minimizes the delay through the center columns. This is achieved by selective term grouping using a Dadda reduction method. Second, move LSB terms toward the MSB side of the reduction columns by using the reduction strategy of the Reduced Area multiplier. This will reduce the size of the final carry propagate adder. There may be opportunities to merge the Dadda and Reduced Area column reduction methods to derive a column reduction method that yields a faster multiplier than Dadda alone. Finally, use a simple ripple carry adder for the LSB

terms in region 1 of the multiplier, then use a carry lookahead or parallel prefix adder for the other two regions of the multiplier to provide a fast multiplier with minimal circuitry in the carry propagate adder.

DYNAMIC POWER ESTIMATION

Using logical effort for design analysis allows a quick relative assessment of relative dynamic power as well as was being able to identify where the power is being consumed in the design.

SUMMARY

This research has extended the analysis to consider the use of logical effort in analyzing delays. Since logical effort is independent of technology, the results may be used to estimate multiplier performance for various CMOS technologies. In the course of this research, over 72 multipliers were designed using four different multiplier column reduction methods, six different multiplier widths and three different carry propagate adders. The results of the research suggest the best possible multiplier column reduction method, the importance of full adder selection on delay, the potential to use a slow and simple carry propagate adder for a certain region of the least significant bits, the potential to use the results of the logical effort analysis to estimate dynamic power for relative power analysis and the potential to quickly estimate the column reduction section's delay by multiplying column gate delay by a constant that is dependent upon the full adder selected.

References

[1] K. C. Bickerstaff, E. E. Swartzlander, Jr, and M. J Schulte, "Analysis of Column Compression Multipliers," *15th IEEE Symposium on Computer Arithmetic*, 2001, pp. 33-39.

[2] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, pp. 14-17, 1964.

[3] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza,* vol. 34, pp. 349-356, 1965.

[4] L. Dadda, "On Parallel Digital Multipliers," *Alta Frequenza*, vol. 45, pp. 574-580, 1976.

[5] K. C. Bickerstaff, M. Schulte and E. E. Swartzlander, Jr., "Reduced Area Multipliers," *International Conference on Application-Specific Array Processors,* 1993, pp. 478-489.

[6] K. C. Bickerstaff, M. J. Schulte and E. E. Swartzlander, Jr., "Parallel Reduced Area Multipliers," *Journal of VLSI Signal Processing*, vol. 9, no. 3, pp. 181-191, April 1995.

[7] R. S. Waters and E. E. Swartzlander, Jr., "A Reduced Complexity Wallace Multiplier Reduction," *IEEE Transactions on Computers,* vol. 59, pp. 1134-1137, August 2010.

[8] W. J. Townsend, E. E. Swartzlander, Jr. and J. A. Abraham, "A Comparison of Dadda and Wallace Multiplier Delays," *SPIE, Advanced Signal Processing Algorithms, Architectures, and Implementations, XIII*, vol. 5205, 2003, pp. 552-560.

[9] V. G. Oklobdzija and D. Villeger, "Multiplier Design Utilizing Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," *1993 International Symposium on VLSI Technology, Systems and Applications, 1993*, pp. 209-212.

[10] P. F. Stelling and V. Oklobdzija, "Design Strategies for the Final Adder in a Parallel Multiplier," *Twenty-Ninth Asilomar Conference on Signals, Systems and Computers, 1995*, vol. 1, pp. 591-595.

[11] V. G. Oklobdzija, "Design and Analysis of Fast Carry-Propagate Adder Under Non-Equal Input Signal Arrival Profile," *Twenty-Eighth Asilomar Conference on Signals, Systems and Computers*, 1994, vol. 2, pp. 1398-1401, Oct. 31-Nov. 2, 1994.

[12] V. G. Oklobdzija and D. Villeger, “Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology,” *IEEE Transactions on VLSI Systems*, vol. 3, pp. 292-301, 1995.

[13] P. F. Stelling and V. G. Oklobdzija, “Implementing Multiply-Accumulate Operation in Multiplication Time,” *Proceedings of 13th IEEE Symposium on Computer Arithmetic*, pp. 99-106, 1997.

[14] D. Baran, M. Aktan and V. G. Oklobdzija, “Multiplier Structures for Low Power Applications in Deep-CMOS,” *IEEE International Symposium on Circuits and Systems,* pp. 1061-1064, 2011.

[15] A. Habibi and P. A. Wintz, “Fast Multipliers,” *IEEE Transactions on Computers*, vol. C-19, pp. 153-157, 1970.

[16] O. J. Bedrij, “Carry-Select Adder,” *IRE Transactions on Electronic Computers*, EC-9, pp. 226-231, 1960.

[17] A. Weinberger and J. L. Smith, “A Logic for High-Speed Addition,” *National Bureau of Standards, Circ. 591*, pp. 3-12, 1958.

[18] T. Kilburn, D. B. G. Edwards, and D Aspinall, “Parallel Addition in Digital Computers: A New Fast “Carry” Circuit,” *Proceedings of IEE*, vol. 106, part B, pp. 464-466, Sept 1959.

[19] J. Sklansky, “Conditional-Sum Addition Logic,” *IRE Transactions on Electronic Computers*, vol. EC-9, pp. 226-231, 1960.

[20] P. M. Kogge and H. S. Stone, “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations,” *IEEE Transactions on Computers,* vol. C-22, pp. 786-793, 1973.

[21] M. R. Santoro, *Design and Clocking of VLSI Multipliers*, Stanford University, *PhD Dissertation*, Technical Report no. CSL-TR-89-397, 1989.

[22] M. E. Robinson and E. E. Swartzlander, Jr., “A Reduction Scheme to Optimize the Wallace Multiplier,” *International Conference on Computer Design*, pp. 122-127, 1998.

[23] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, New York: Oxford University Press, p. 179, 2000.

[24] I. Sutherland, B Sproull, D. Harris, *Logical Effort: Designing Fast CMOS Circuits*, San Diego: Academic Press, 1999.

[25] Mauro Olivieri, "Overview on a formal model of architecture/circuit trade-offs for the implementation of fast processors," *Computer Physics Communications* vol. 139.1, 2001, pp. 144-150.

[26] Jo Ebergen, Jonathan Gainsley, and Paul Cunningham, "Transistor sizing: How to control the speed and energy consumption of a circuit," *10th International Symposium on Asynchronous Circuits and Systems*, 2004, pp. 51-56.

[27] Dursun Baran, Mustafa Aktan, and Vojin G. Oklobdzija. "Multiplier structures for low power applications in deep-CMOS," *IEEE International Symposium on Circuits and Systems (ISCAS),* 2011, pp. 1061-1064.

[28] Neil Burgess. "Fast ripple-carry adders in standard-cell CMOS VLSI," *20th IEEE Symposium on Computer Arithmetic,* IEEE, 2011, pp. 103-111.

[29] Richard Kohler Richards, *Arithmetic Operations in Digital Computers,* Princeton: D. Van Nostrand Co., Inc. 1955.

[30] Neil Weste and David Harris, *CMOS VLSI Design, A Circuits and Systems Perspective,* Boston: Pearson Addison Wesley, 2005.

[31] Vojin G. Oklobdzija, ed., *The Computer Engineering Handbook*, Boca Raton, FL.: CRC Press, 2001.

[32] Adnan Kabbani, "Logical effort based dynamic power estimation and optimization of static CMOS circuits," *Integration, the VLSI Journal*, vol. 43.3 (2010), pp. 279-288.