CARRY-PROPAGATE FREE COMBINATIONAL MULTIPLIER

By

JONATHAN HOLDEN

Master of Science in Electrical Engineering

Oklahoma State University

Stillwater, OK

2009

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2009

CARRY-PROPAGATE FREE COMBINATIONAL

MULTIPLIER

Thesis Approved:

James Stine

Sohum Sohoni

Chris Hutchens

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

This work would not have been possible without the encouragement and support of my advisor Dr. James Stine, under whose supervision I chose this topic and was greatly assisted by his vast knowledge and resources in computer arithmetic, in particular for giving me access to much of Miloš Ercegovac and Tomàs Lang's work, much of which my research is based upon.

I would also like to thank the members of my thesis committee, Dr. Chris Hutchens, Dr. Sohum Sohoni, and Dr. James Stine, for their assistance in working with me to complete my thesis in a timely manner after my hospitalization.

Lastly I want to thank my friends and family, whose constant help and encouragement has assisted me throughout my time here at Oklahoma State University. I am grateful particularly to my grandfather, Anthony Maggio, who always assisted me with any problem I have had related to school or otherwise.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

I. INTRODUCTION

In digital arithmetic there are two classes of multipliers; parallel and sequential. Parallel architectures are faster, but require a larger die area, which in turn, relates to higher fabrication cost. Sequential multipliers have a smaller die area and are, thus, cheaper; however, sequential multipliers are slower and require a clock signal. For these reasons, parallel architectures of multipliers are usually chosen, if speed is important.

Multiplication is an important part of computer arithmetic, in particularly because it is important in business and scientific computations [1]. Multipliers found within general-purpose and applications-specific architectures are usually composed of digital signals that represent both *0* and *1 [2]*. Although many multipliers use binary to represented decimal numbers, the use of signed digit or redundant notation can be used to increase the speed of multiplication significantly. Unfortunately, using a representation that can allow multiple representations of the same number can have negative effects on the size of most multipliers. Consequently using complicated representations of numbers should be tempered with the advantages that are produced with speed.

Regular carry save array multipliers have three main parts: the first is partial product generation, the second is partial product reduction, and the third is the final carry propagate addition. Unfortunately the carry propagate adder causes a large delay as the multiplier increases its operand size. For example a ripple carry adder (RCA) can be analyzed using a linear time analysis. Therefore, assume *1* gate takes 1Δ delay and occupy *1* unit of area, a n-bit ripple-carry adder takes

$$\text{Delay} = 5 + (\text{N} - 1) * 2 + 3 = 2n + 4\,\Delta \qquad (1.1)$$

$$\text{Area} = N * 9 \qquad (1.2)$$

For a four bit multiplier, the delay is only *12* Δ, however, when the multiplier is scaled up to *64* or *128* bits the delay becomes *132* or *260* Δ, respectively. Alternate adder designs could be used, but the speed gains compared to the area are usually equal or even disproportionate. In [4], a *16* bit carry-lookahead adder is considered having a delay of *14* Δ and an area of *214* gates, as opposed to an RCA, with delay of *36* Δ and area of *144* gates, which nearly double the area to half the delay.

This research discusses an implementation of a multiplier that uses a redundant notation, yet uses a small amount of additional hardware to reduce the overall worst-case delay [3]. By modifying the representation of the numbers within a multiplier, additional hardware can compensate for the removal of the final carry-propagate addition and, hopefully, significantly improve the design. Although similar approaches have been previously introduced, this research differs from previous work in that it focuses mostly on the implementation of previous designs [3]. Moreover, a signed-digit adder is added to the circuit to allow the time to complete a computation within a multiplier to be significantly reduced.

Previous Work on Carry Free Multiplication

As mentioned in [1], one of the methods to increase the overall speed of a multiplier is to use signed digit notation within the multiplier. Although previous designs have

incorporated signed digit notation within the multiplier [5]-[7], the design presented in [3] uses the redundant notation of the input operands to remove the final carry-propagate adder . It should also be noted that although [8] and [9] use a similar redundant notation as [3], however, they fail to obtain carry-free multiplication.

Carry Propagate Free Multiplier

The multiplier design described in [3] is unique in several different areas: first, the design utilizes signed digit notation and using a radix *4* notation reducing the number of addition operations in the multiplier by half. Secondly, the multiplier design does not use a carry propagate adder, thus, reducing the delay after the partial product recurrence. Additionally, the design is slightly abstract and lacks detail especially on the recoding of one of the multiplicands, but fails in particular at describing how to implement the algorithms in hardware creating a unique challenge.

This thesis will explain in detail the architecture of a *4* bit implementation for three reasons. First the architecture we will be presenting is based on two bit blocks, increasing the number of bits simply increases the number of blocks. Secondly, *4*-bit multiplier arithmetic can be easily checked and understood for all solutions are between *0* and *255*. Finally, a smaller design allows synthesis engines to actively explore symmetry for the final design. Because of the large amount of time required to develop hardware implementations from an algorithm there may be significant work remaining to acquire sufficient data and it may be necessary to extrapolate data to come to a conclusion. This thesis will be considering area, power, and delay for each of the multipliers that are synthesized.

Paper Organization

The rest of this thesis is organized as follows Chapter II will focus on background information relating to multipliers and specific techniques used in our design such as SD adders and on the fly conversion. Chapter III will describe the details of the multiplier particularly how the algorithms work in hardware and where the design strengths and weaknesses may be. Chapter IV will discuss the design and synthesis process as well as the results from the synthesis and Chapter V will summarize the major findings of this paper and any future work that should be completed

II. BACKGROUND

It is important to understand that in computer arithmetic there is no intuitive operations and that everything must be done in a logical operation. When a computer multiplies X by Y it cannot intuitively know the answer no matter how simple the problem. Rather, it must add X by Y times to obtain the solution simply put every multiplication is a series of additions. This process is most easily observed in a carry save array multiplier, as shown in Figure 1. The array multiplier forms a partial product via an AND gate for the X multiplicand bit and Y multiplicand bit. In general, n-bit carry-save array multipliers have n^2 AND gates, (n-1) half adders, (n-1)(n-1) full adders, and a (n-1) final carry propagate adder. The critical or worst-case path of the data is the variable that will determine the speed of the design. In Figure 1, the critical path begins at the top left of where a3 and b0 are ANDed together and then descends diagonally to the bottom right, and finally moves horizontally to the bottom left hand side where z[7] or the most-significant bit (MSB) is computed.

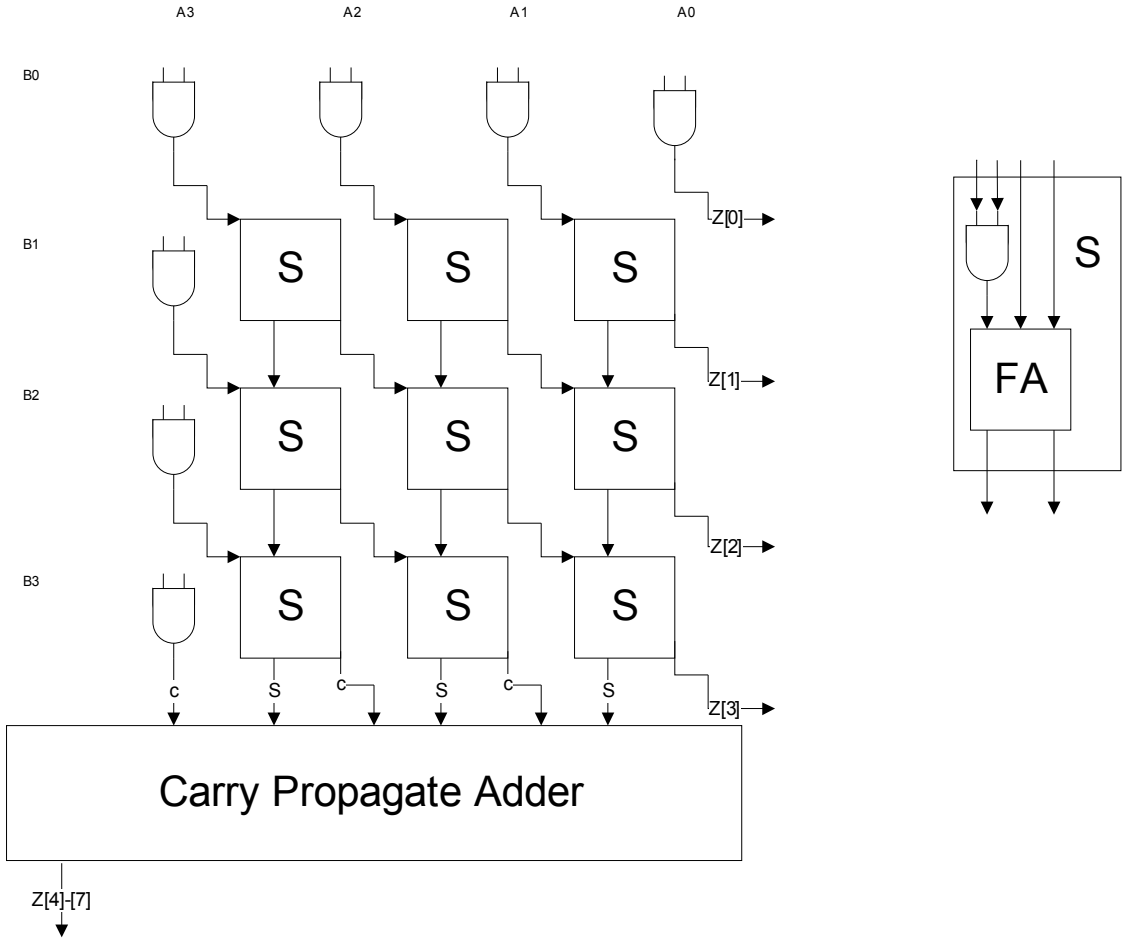

Figure 1 Simplified Multiplier Design

It is of importance to note that there are a large amount of steps required for any carry-save array multiplier and that the area and delay grow nearly by a power of two for each doubling of the bit size.

Redundant Notation

One way in which to increase performance is to reduce the number of steps involved in the multiplication operation. This can be accomplished by using redundant notation or signed digit notation for both input operands. Numbers are considered

redundant if each digit can be represented by more than one value. For this work, it is presumed two wires are used for the number system, such that it represents {-1, 0, 1}, as opposed to regular binary notation {0,1}. This method of representing or encoding the bits is often used in redundant designs and is commonly referred to as one hot encoding [10]. More importantly, using one-hot encoding allows negative numbers to be represented easier. For example, the redundant notation or *391* can be computed as follows:

$$4\bar{1}1 = 4\times10^{2} + -1\times10^{1} + 1\times10^{0} = 391 \qquad (2.1)$$

Redundant or signed digit (SD) notation allows multiplication units to use shifting instead of addition, which is typically faster in hardware. This technique is also utilized in similar designs that employ Booth encoding [11]. Number system are commonly defined according to the number of bits that they operate with or the radix. As in [3], the radix for this work is set to *4* and is in redundant notation so the number system is {-3,-2,-1,0,1,2,3}. By using a larger radix, tasks can be completed using a smaller number of steps and, subsequently, reduce the overall latency of the operation. Radix 4 can also be achieved easily by using two bit slices which works out well considering that this design uses two bit slices. Radix *4* redundant notation and radix *2* redundant notation are equivalent in hardware, as shown for the following example: $2\bar{3}10 = 10\bar{1}\,\bar{1}0100$.

There are several disadvantages associated with redundant notation. The first is that at the end of an operation the result must be converted back to standard notation for use in other parts of the digital system. Normally, in order to accomplish this conversion, a carry-propagate adder must be used, which is typically slow and cumbersome for large input operands. Another disadvantage is that the hardware design becomes increasingly

complex and can drastically increase the area of the individual blocks. Finally, the complexity of wiring blocks can also increase exponentially leading to unequal delay paths, which cause many problems within digital systems.

Signed Digit Adder

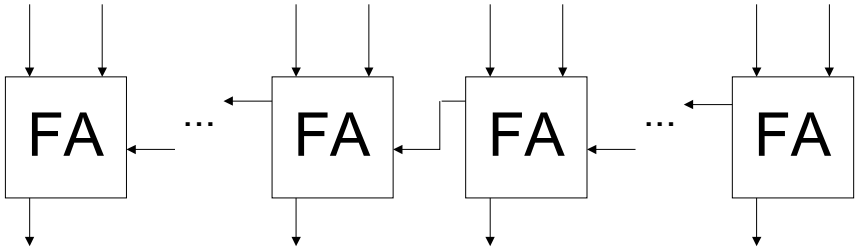

Figure 2 Ripple Carry Adder

When using redundant notation it is not possible to use regular adders, since there are two wires instead of one for each bit. This introduces an opportunity for using different forms of adders. Regular adders have a carry that propagates through each bit, as seen in Figure 2. This results in a large delay due to the carry chain which would be beneficial to eliminate or reduce [12]. Due to these two issues, it is advisable to use what is called a signed digit adder. A SD adder uses two wires and adds both the negative and positive parts together. In [13]-[18] different SD adder designs are discussed using a two step addition process and avoiding any carry propagate addition, as shown in Figure 3. The weakness of SD adders is that the input operands must be recoded into redundant notation, so that no carries are present in the addition operation. This normally requires a special recoding step to be added to the process as well as a decoding step that must be added to convert the solution into a non-redundant solution.

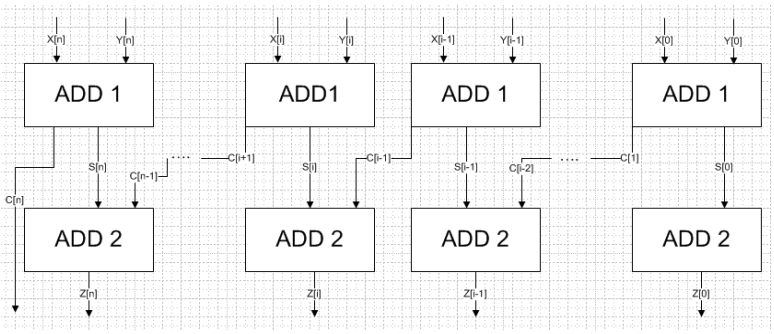

Figure 3 Signed Digit Adder

The advantage of the SD adder is in the speed in which a sum can be calculated. An *8*-bit Ripple Carry Adder (RCA) can be built from seven Full Adders (FA) and a Half Adder (HA). Using a linear area/delay analysis, each FA has an area of *9* gates and a carry-in to carry-out (worst case) delay of *2* delta where one delta represents the time required for a digital signal to propagate through one gate. Each HA has an area of *5* gates and a delay of *1* delta. This gives our eight bit adder an area of sixty-eight gates and a worst case delay of fifteen delta. For each bit that is added to the adder, the area increases by *9* gates and the delay is increased by *2* delta. An SD adder design has an add1 and add2 block for each bit slice each bit slice has a total area of *17* gates and a delay of *4* delta an *8*-bit SD adder would have an area of one hundred thirty-six gates and a delay of *4* delta. Adding bits to an SD adder increases the area of the adder, but not the delay. The major drawback of SD adders is that there must be a recoding stage prior to the adder that reconfigures one or both of the addends into redundant notation in such a way that a carry result does not occur.

Recoding Scheme

A recoding scheme converts a standard binary number into a redundant number that is designed to prevent carries throughout the SD adder blocks. The problem with the recoding scheme is that there is no standard algorithm to create the recoding hardware which results in a hit or miss design that must be verified through testing. Our process will involve designing a recoding scheme, testing it for functionality, determining where the recoding fails, and redesigning the scheme to avoid the error sequence. An example of standard addition with carries and a recoded addition without carries is seen in Table 1. Only one operator needs to be recoded and the only limitation on the recoding scheme is that the recoded number must equal the original number.

Table 1 Standard Carry Addition Compared to Recoded Addition

	1	1	1	1
	1	1	1	1
1	1	1	1	0

Standard Addition

					$\bar{1}$
1	0	0	0	$\bar{1}$	
	1	1	1	1	
1	1	1	1	0	

Recoded Addition

On The Fly Conversion

One of the drawbacks of using redundant notation is that it must converted back to non-redundant or standard notation at the end of the operation. Unfortunately, all redundant notation must use a carry propagate adder to recode it back to its original form. However, this causes a large delay and should be avoided. One solution to this process can be the use of online algorithms. An online algorithm is an algorithm in which parts of the most significant digit are computed before the least significant bits. As in [17] and

[19], a simplified conversion formula can be derived. To illustrate the process of on-the-fly-conversion, a number should define a digit vector with k significant digits as Q[k] such that

$$Q[k] = \sum_{i=0}^{k} q_i \, r^{-i} \qquad (2.3)$$

Therefore, during each iteration of Q, an equation can be formed based on the iteration and radix, r:

$$Q[k+1] = Q[k] + q_{k+1} r^{-(k+1)} \qquad (2.4)$$

Since q_i from Equation 2.3 can be negative, it is important to use a second set of equations to solve for Q. Since each iteration of the conversion allows the the digits to be converted back to normal representation, its easy to allow this conversion to be extended to numbers that are either greater than or less than Q. For example, Q can be derived, such that, it produces a value QM that is one unit in the last place less than Q. And, since the digits are produced from the most-significant bit towards the least-significant bit, two equations that allow the conversion to update the converted result is given by the following two equations:

$$Q[k+1] = \begin{cases} Q[k] + q_{k+1} r^{-(k+1)} & if\ q_{k+1} \geq 0 \\ QM[k] + (r - |q_{k+1}|) r^{-(k+1)} & if\ q_{k+1} < 0 \end{cases} \qquad (2.5)$$

$$\text{QM[k+1]} = \begin{cases} Q[k] + (q_{k+1} r^{-(k+1)}) r^{-(k+1)} & if\ q_{k+1} > 0 \\ QM[k] + ((r-1) - |q_{k+1}|) r^{-(k+1)} & if\ q_{k+1} \leq 0 \end{cases} \qquad (2.6)$$

By using both these equations there is no carry/borrow additions and only concatenations resulting in a quick conversion from redundant notation to a standard binary representation. For example, a radix 2 example of on-the-fly-conversion is shown in Figure 4 for $1\,1\,0\,1\bar{1}\,0\,0\,\bar{1}\,1\,0\,1\,0$.

k	q_k	Q[k]	QM[k]
0		0	0
1	1	0.1	0.0
2	1	0.11	0.10
3	0	0.110	0.101
4	1	0.1101	0.1100
5	-1	0.11001	0.11000
6	0	0.110010	0.110001
7	0	0.1100100	0.1100010
8	-1	0.11000111	0.11000110
9	1	0.110001111	0.110001110
10	0	0.1100011110	0.1100011101
11	1	0.11000111101	0.11000111100
12	0	0.110001111010	0.110001111011

Figure 4 On The Fly Conversion - General Theory

From the example there are two observations that should be noted first that Q[k] is always greater than QM[k] by one. Secondly that this implementation requires two registers on to store Q[k] and the other to store QM[k] a block diagram of the hardware implementation, as shown in Figure 5.

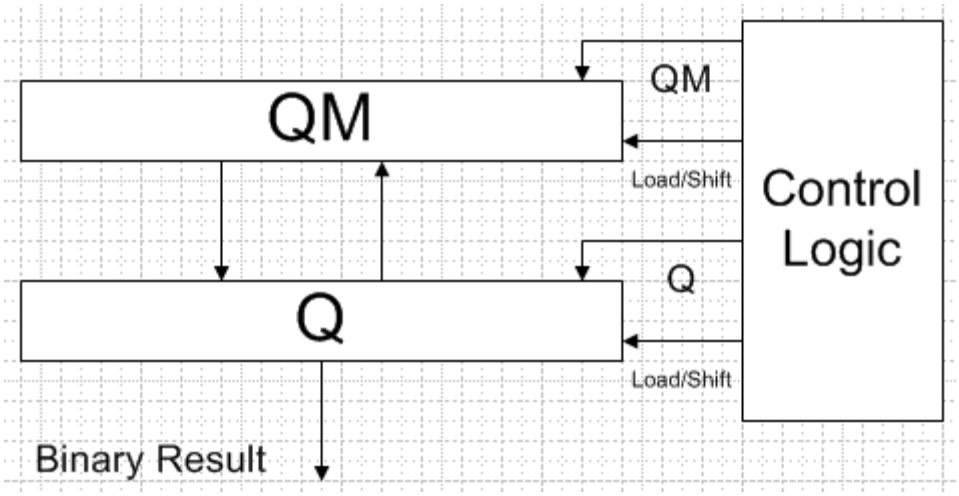

Figure 5 Block Diagram of OTF Conversion - General Theory

III. METHODOLOGY

Given that a standard carry save array multiplier has a large number of steps, which in turn results in large area, and that delay increases when the multiplier is increases, traditional carry-save array multipliers can incur a significant amount of delay for large input operands. Our design intends to create a multiplier that scales at a much better rate, but utilizes a number of techniques already given in [3] yet refined for hardware. The most significant part of the design is the use of signed digit radix 4 notation, which causes the design presented in this thesis to contain significantly smaller number of steps to complete. However, each step is more complicated as each multiplicand has four inputs instead of one. The more complicated design requires us to use SD adders which results in a larger area but the SD adder arrays do not utilize carry propagate adders resulting in a smaller delay time. The proposed design is also bit sliced in two bit slices which aides in the on the fly conversion returning the SD number into standard binary notation that can then be used by designs that use large amounts of hierarchy.

The propagate-free multiplier design differs from a traditional carry save multiplier in several areas. First, as mentioned before a regular carry save multiplier has two main parts the first is partial product recurrence and the second is the carry propagate adder our multiplier has three main parts; a Recoding block a Signed Addition block and an on-the-fly conversion block, as shown in Figure 6.

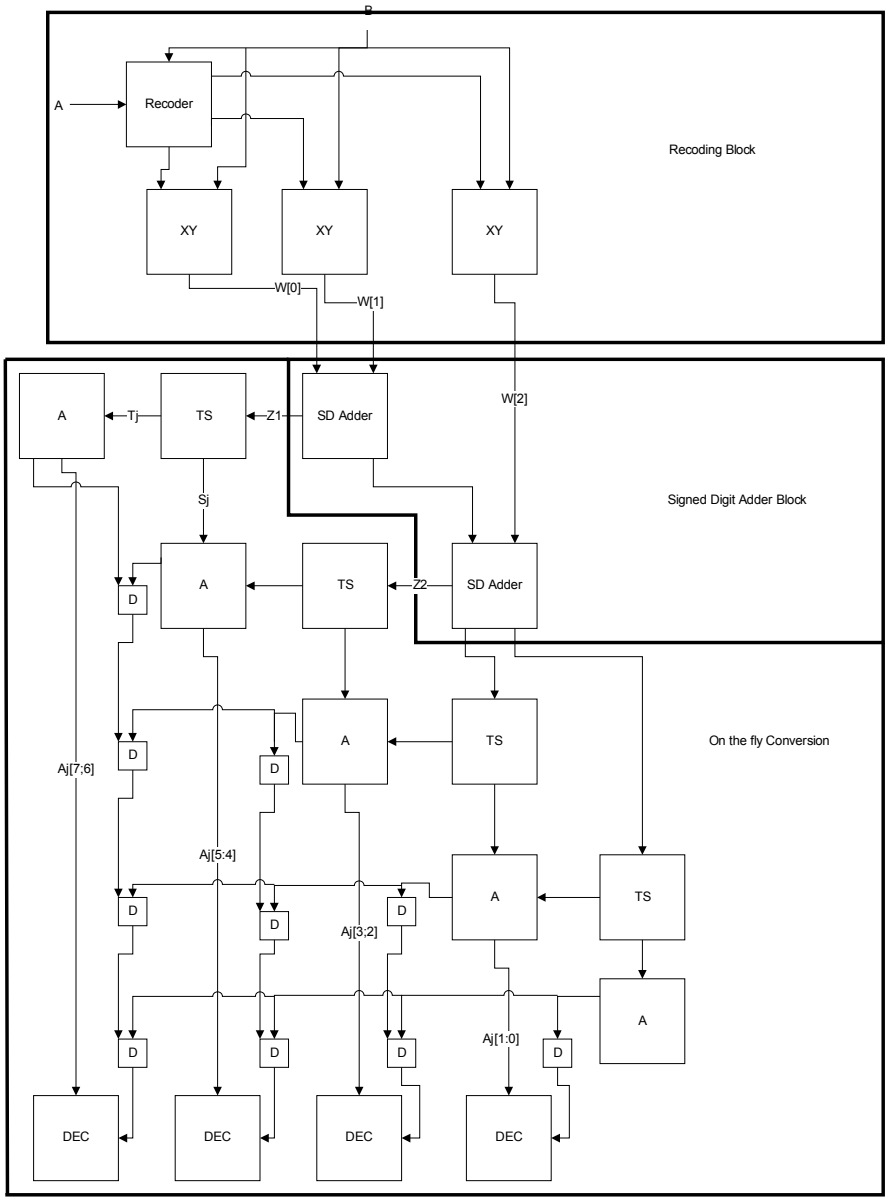

Figure 6 Block Diagram of Multiplier Design

In the carry-propagate free design, each multiplicand is divided into two bit numbers. During the signed digit addition, each two bit number is usually between -3 and 3, however, in some circumstances the result from the signed digit addition can be between -5 and 5 requiring a third or carry bit, as shown in Table 2.

Table 2 Negative Limit Example

15x5 = 75 A = 5 = 0x11 converts to 0x1$\overline{23}$ B = 15 = 0x33							
W		Z_j	T_{j-1}	S_j	P_{j-1}	$A_{[j-1]}$	$D_{[j-1]}$
1 2	$\overline{3}$ $\overline{3}$ $\overline{2}$ $\overline{1}$ 3 $\overline{2}$ 2 0 $\overline{2}$	2	0	2	0	0	U
3	$\overline{2}$ $\overline{3}$ $\overline{1}$ $\overline{2}$ $\overline{5}$ $\overline{1}$	$\overline{2}$	0	$\overline{2}$	2	02	nu
		$\overline{5}$	$\overline{1}$	$\overline{1}$	$\overline{3}$	021	ndu
		$\overline{1}$	0	$\overline{1}$	$\overline{1}$	0213	nddu
			0	0	$\overline{1}$	02133	ndddu
						01023	= 75

In [3] the algorithm does not convert the lower half of the answer into non-redundant form, however, in this implementation all bits are convert into non-redundant form to allow for more accurate rounding after the multiplication process. It should also be noted that adding these bits to the on the fly conversion does not significantly increase the size or delay of the multiplier. The design also needs to optimize these blocks, so that they can be eliminated in some cases without loss of accuracy, depending upon the rounding scheme used after the multiplier. The speed increase would be equal to the delay of block D multiplied by N/2 where N is the number of bits in the multiplier.

Recoding Block

The recoding block has two parts; signed recoding and XY Combination. The signed recoding block converts one multiplicand into a sign notation for the purposes of this paper the multiplicand denoted as “A” will always be the multiplicand that is recoded. In [3] the importance of the signed recoding is highlighted properly. A simple recoding scheme such as X = 16 -15 + A. (3.1) where X is the recoded number will break down in certain circumstances such as when multiplying 0x22 by 0xFF. Instead, a signed recoding algorithm must be chosen that focuses on the two bit sections of the multiplicand thus reducing the chances of a carry during the signed addition block it is important to note that a carry here is ok and is expected in a couple of situations.

The recoding scheme that has been chosen uses a four layer design as shown in Figure 7. the first three layers have a delay of four gates total that increases in size N but the delay remains constant the fourth layer is dependent upon the delay of a N bit signed digit adder where N is the number of bits of the multiplier. This is significant because as N increases then the delay of the recoding block will increase based upon the delay of the n bit signed digit adder and could result in a speed problem requiring an optimized signed digit adder or a different recoding scheme.

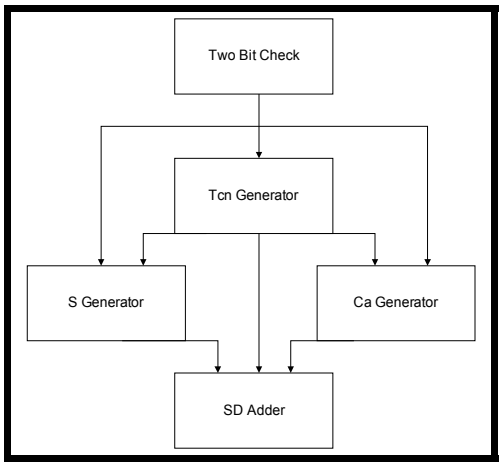

Figure 7 Internal Block of The Recoder

At higher bit implementations the radix digit system can be increased to handle numbers between -7 and 7 by adding only a few gates to the design allowing for a simpler recoding scheme. The XY Combination block takes each two bit number from X and multiplies it by B an example can be seen in Table 3. Each XY block is essentially an optimized combinational multipliers capable of multiply by -3 to 3 this process allows us to use radix 4 from this point and thus reduces the number of SD adders.

Table 3 Truth Table for XY Block

X is a two bit number, B is four bits all numbers in decimal			
Example #	X	B	xY Output
1	0	10	0
2	1	10	10
3	2	10	20
4	3	10	30
5	-1	10	-10
6	-2	10	-20
7	-3	10	-30

It should be noted that the XY block of the algorithm is a possible weakness in the design particularly in small designs as each multiplier has essentially many two bit regular multipliers that make up a part of the design if speed is desired then this block should be redesigned as there are a couple carries contained in this block that should be eliminated if possible.

Signed Digit Addition Block

The Signed Digit Addition block is straight forward, each w[n] and w[n+1] is added together. The two most significant bits are carried out to the on the fly conversion block while the lower four bits are carried into the next adder to be added to w[n+2]. After the last adder in our implementation the lower four bits are also carried into the on the fly conversion block. In [3] they are left in redundant form . There are several approaches to developing a signed digit adder while the adders proposed in [13] and [14] work as intended they are overly complicated for the task at hand each design used what was essentially three levels causing larger delays and area usage then we intended to use. The adder proposed in [15] was developed in our hardware implementation but we were unable to complete a fully function hardware implementation as each revision contain anomalous errors in it and was ultimately abandoned for a more simple design as described in [16] and [17]

Each SD Adder has a simplification block that is implemented at the end of the process in which every two bit slice is simplified into a non-convoluted form. This is so that each two bit slice follows the algorithm cleanly, the functionality of the block can be seen in Table 4.

Table 4 Simplification Block Outputs

Two bit Input	Two bit Output
0 0	0 0
1 0	1 0
0 1	0 1
1 1	1 1
$\bar{1}$0	$\bar{1}$0
0 $\bar{1}$	0 $\bar{1}$
1 $\bar{1}$	0 1
$\bar{1}$1	0 $\bar{1}$
$\bar{1}\bar{1}$	$\bar{1}\bar{1}$

On The Fly Conversion Block

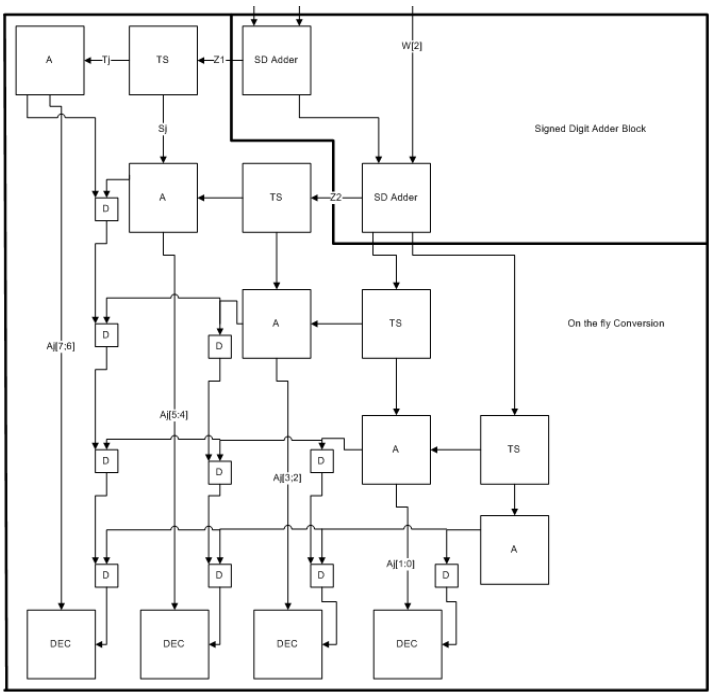

Figure 8 On The Fly Conversion Block

The third and final block converts our signed digit notation (-5 to 5) into standard binary (0, 1) we will be using the On the Fly Conversion as described in [3], [17], [20], and [21] there are four parts to OTF Conversion a Ts block that separates the two bit input, the A block that converts and sums s_{j-1} and t_j the D block array which keeps track of whether to decrement the two bits and the DEC block that decrements the two bits if needed an example of how this is implemented can be seen in Figure 8.

The Ts block takes the two bit output from the SD adder and converts it into a number represented by t_j and s_j the conversion chart is shown in Figure 9.

Zj	-5	-4	-3	-2	-1	0	1	2	3	4	5
tj-1	-1	-1	-1	0	0	0	0	0	1	1	1
sj	-1	0	1	-2	-1	0	1	2	-1	0	1

Figure 9 Ts Block Recoding Scheme

Table 5 D Block Truth Table

Inherited Value	New Value	Output
U	U	U
U	D	D
D	X	D
N	X	N

Block A adds t_j and s_{j-1} and outputs a_j and a sin bit, written as;

$A_j = t_j + s_{j-1}$ (3.2)

Sin = 1 if $A_j < 0$ else Sin = 0 (3.3)

the D block maintains a record of whether or not to decrement the two bit number Table 5. shows a truth table for the D block where U,D,N, and X stands for undecided, decrement, no change and don't care.

The DEC block decrements a_j if the output from the D block line is a one (D) where the two bit number is decremented, as shown in Table 6. The data is in binary all other outputs from the D block are considered don't cares and can be ignored. The output from the DEC block is the final multiplication result.

Table 6 Truth Table For DEC Block

Inpu	D Block Output	Output
11	00	11
11	01	01
11	10	11
11	11	11

Multiplication Example

Now that we have discussed the basics of how our carry-propagate adder will work, Table 7. shows an example our multiplication procedure note that during the recoding stage the numbers are converted to signed digit notation from -3 to 3 and that during signed addition the numbers range from -5 to 5. It is worth noting that only a small amount of logic is located before and after the signed digit adders and that most of the logic is executed in parallel with the signed addition which should result in a lower overall delay. The columns from Z_j to $D_{[i-1]}$ are steps of the on the fly conversion.

Table 7 Example Arithmetic

7x9=63 A=7=0x12 converts to 0x12$\bar{1}$ B = 9 = 0x21							
W		Z_j	T_{j-1}	S_j	P_{j-1}	$A_{[j-1]}$	$D_{[j-1]}$
1 2	2 1 $\bar{1}$ 0 $\bar{2}$ 1 1 $\bar{2}$	1	0	1	0	0	u
3	$\bar{2}$ $\bar{1}$ 1 $\bar{4}$ $\bar{1}$	1	0	1	1	01	nu
		$\bar{4}$	$\bar{1}$	0	0	010	nuu
		$\bar{1}$	0	$\bar{1}$	0	0100	nuuu
			0	0	$\bar{1}$	0100 3	ndddu
						0033 3	

IV. RESULTS

Due to the nature of the project our primary effort was to design hardware for the multiplier algorithms described in [3]. Each algorithm was divided into functional blocks then each block was designed at the gate level. Next each block was developed using structural Verilog Hardware Descriptive Language (HDL) then each block was tested in the Xilinx ISE. Each block was tested exhaustively, when an error was found the block was checked and redesigned on paper and then rewritten in Verilog HDL. After each of the individual blocks were designed all the blocks were integrated to build a four bit multiplier design. The design was then tested and the recoding block was modified to create a better recoding scheme. After the four bit multiplier design was successfully tested an eight bit design was built, the SD adders and recoding blocks were modified to handle eight bits. All other blocks were built as two bit slices and multiples of these blocks were added to handle eight bits. All carry save multiplier designs are built using a predefined script.

Synthesis for our purposes uses actual logic blocks for the layout of the design which gives a good estimate of area and power consumption it is useful as it is the first step in preparing a design for fabrication. All multipliers are synthesized using cadence at the AMI C5N 0.5 micron feature size. The power and size are taken from a synthesized simulation using a standard cell library. Standard cells are layout designs that have been

previously created and stored in a library for use in future projects. Standard cells tend not to be optimized for power, area, or delay.

Encounter places the synthesized cells and routes wires to each of the cells this is the one of the last design steps before sending a design off for fabrication the delay was recorded after Encounter had routed wires to the cells so that wire delay is included in the delay time. In Table 8. you can see the results from the standard multiplier design.

Table 8 Regular Carry Save Multiplier Results

	4 bit Multiplier	8 bit multiplier	16 bit multiplier
Dynamic Power [mW]	0.69	5.38	34.24
Leakage Power [mW]	0.01	0.03	0.02
Number of Cells	78.00	377.00	1,638.00
Total Cell Size [um^2]	19,278.00	97,029.00	421,479.00
Delay [ns]	4.13	9.46	19.28

Below in Table 9. you can see the results from the propagate free multiplier design

Table 9 Carry Propagate-Free Multiplier Results

	4 bit multiplier	8 bit multiplier
Dynamic Power [mW]	4.24	13.67
Leakage Power [mW]	0.03	0.08
Number of Cells	430.00	1,217.00
Total Cell Size [um^2]	97,713.00	273,528.00
Delay [ns]	11.01	17.86

By observation it can be seen that the standard multiplier is smaller and faster at four and eight bit implementations however if we plot the data on a logarithmic scale we should be able to determine how well our design will scale. The delay increase of the two adders is plotted in the Figure 10. revealing that by extrapolating the two measured data points the propagate-free adder does scale better. We can justify extrapolating with just two points

because as noted multiple times the propagate-free design uses two bit block slices and thus should increase in area, power, and delay at the same rate per new bit.

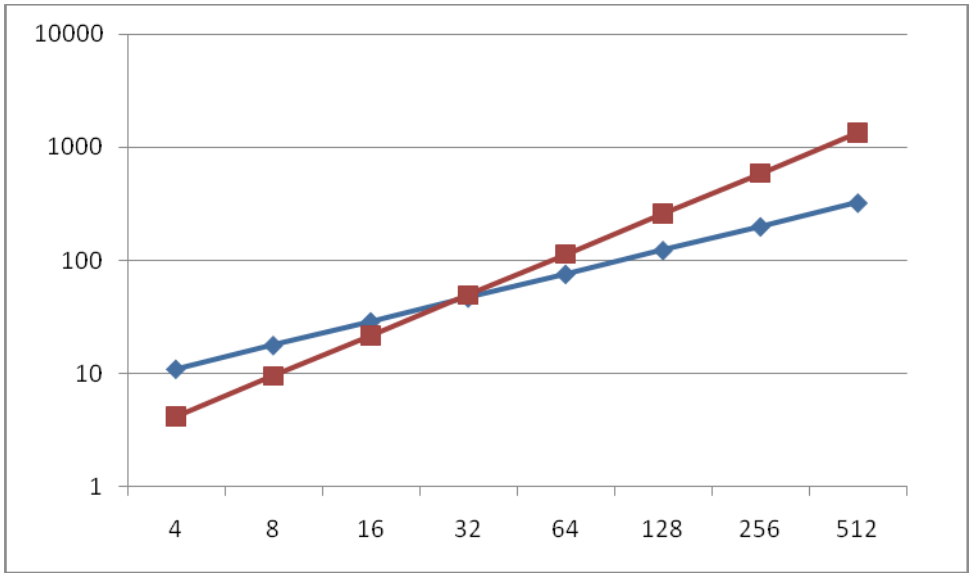

Figure 10 Multiplier Delay in ns with Respect to Bit Size (Extrapolated)

We can see that that the propagate-free adder starts out slower but the delay increases at a slower rate giving it a speed advantage at 64 bits and above. However, this observation must be scrutinized and further researched since it is based on a two point extrapolation. Because the regular multiplier is easily created with the use of scripts we can synthesize the regular multiplier up to 128 bits to help confirm our extrapolation. The results are shown in Table 10. the synthesis results are better than our extrapolated data and the delay is plotted in Figure 11. along with both of our extrapolated data lines.

Table 10 Regular Carry Save Results For Large Bit Implementations

	32 bit multiplier	64 bit multiplier	128 bit multiplier
Dynamic Power [mW]	232.09	1,339.70	3,300.80
Leakage Power [mW]	0.76	3.61	14.11
Number of Cells	8,456.00	39,701.00	123,560.00
Total Cell Size [um^2]	2,423,538.00	11,636,496.00	43,826,913.00
Delay [ns]	26.49	50.82	139.70

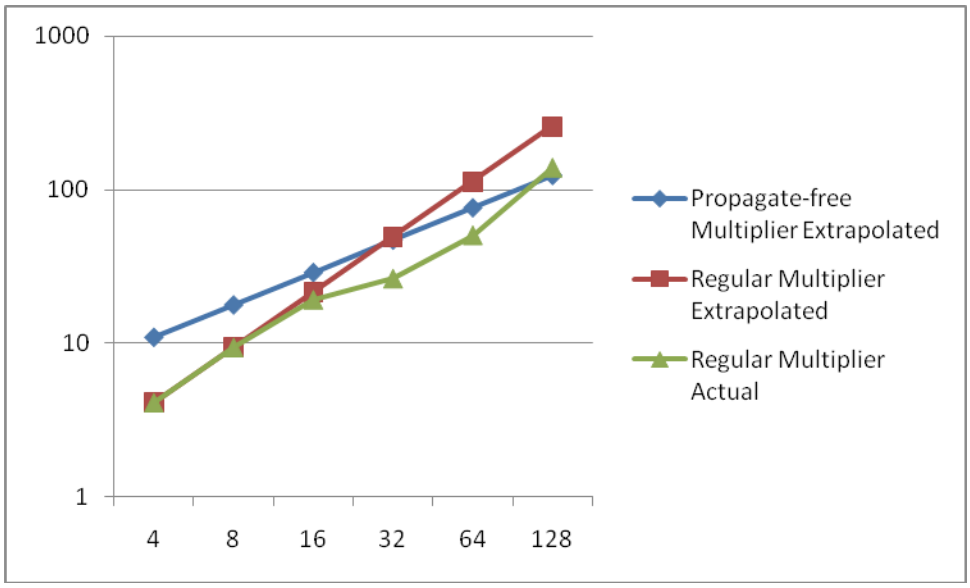

Figure 11 Multiplier Delay in ns with Respect to Bit Size (Actual VS Extrapolated)

After examining Figure 11 it is observed that the actual regular multiplier, out performs not only its own extrapolated results but also those of the propagate-free multiplier. However an unexpected variable was present in the cadence software that modified the results resulting in a non-uniform data line for the synthesized regular multiplier as the complexity of the design increases cadence compensates and increases its optimization aggressiveness this can be seen clearly by the much smaller than expected delay between the *16* and *32* bit multipliers which is then followed by a much

more expected jump between the *32* and *64* bit multipliers. This increase in optimization effort can also be seen in the synthesis time of different multipliers for example the *16* bit requires less than fifteen seconds to synthesize but the *128* bit multiplier requires nearly eight hours to synthesize. (Time does not include place and route time). This increase in the aggressiveness of the optimizations should also yield better results for the synthesized propagate-free multiplier giving it an added advantage in larger multiplier designs. The same principle applies to power consumption that the extrapolated data for the propagate-free multiplier consumes less power at the *32* bit implementation and gains a considerable advantage as the multiplier size increases as shown in Figure 12.

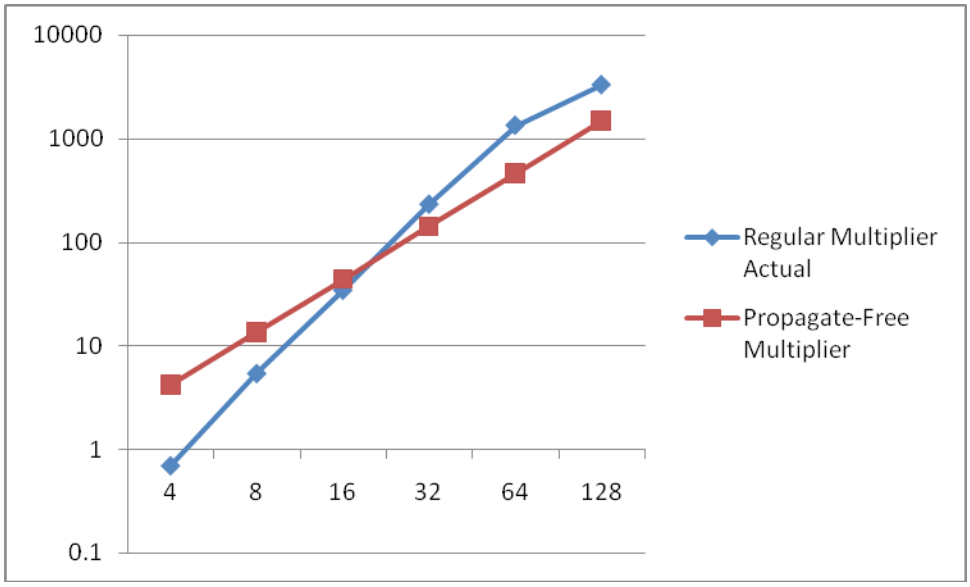

Figure 12 Multiplier Power Consumption in mW with Respect to Bit Size

When comparing number of cells the propagate-free multiplier advantage is diminished as the extrapolated data fails to gain an advantage over the standard multiplier until the bit size exceeds 64 bits as shown below in Figure 13.

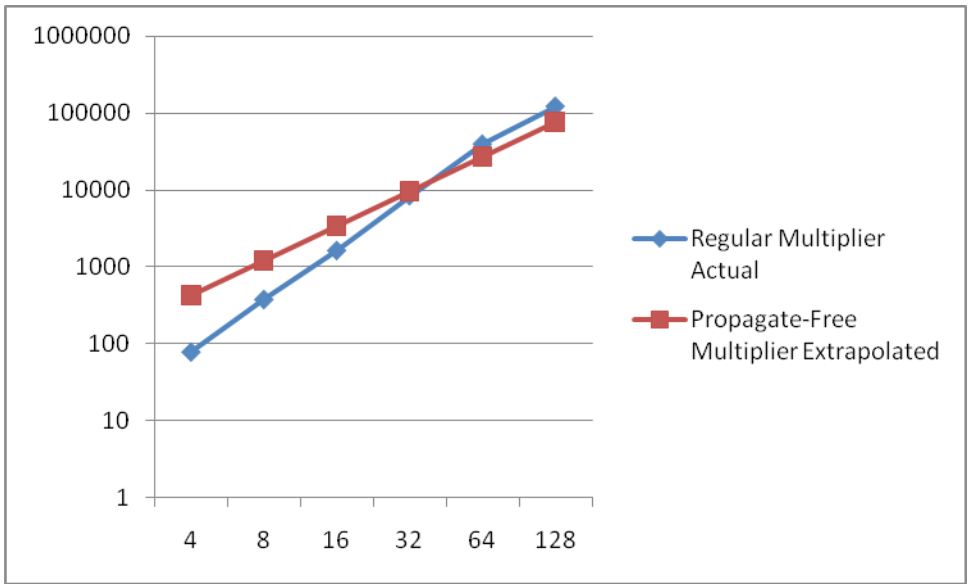

Figure 13 Multiplier Number of Cells with Respect to Bit Size

V. CONCLUSION

In this thesis we confirmed that the algorithms in [3] are in fact capable of being implemented in hardware and that the algorithm is correct. We also confirmed that the recode block is perhaps the most important part of the multiplier and requires careful planning regarding its implementation.

The implemented results were somewhat disappointing revealing that the Propagate free design is larger and slower than a regular carry save multiplier at any size smaller than 32 bits. The extrapolated propagate-free results outperform the results of the regular multiplier at 128 bits therefore it is likely that a propagate-free design is competitive or superior at 64 bits. The design presented in this thesis focused on a functional design and not an optimized design and it is possible the decrease the size and delay of several blocks that would increase multiplier speed specifically the XY combination block. The signed digit adders are used throughout the design and eliminating a single gate delay from it would increase performance greatly. Within the XY combination block there are several full adders with a small carry chain that could be replaced with a more complex combinational logic. This would add a modified carry-free adder decreasing delay considerably.

Summary

Our multiplier design was successfully implemented in a synthesized hardware environment and was tested to be fully functional as a multiplier. Our propagate free design is not superior to a regular carry save multiplier at four or eight bit implementations. However, the scaled difference between the two implementations project that our propagate free design should become smaller and faster in larger implementations. . Shows which design has the superior characteristics in area, delay, and power consumption the standard Carry Save Array Multiplier is represented as CSAM whereas our Carry-Propagate Free design is represented by CPFM. It can be seen that the standard design maintains an advantage at 16 bits or less our design begins to gains a power advantage at 32 bits and an area advantage at 64 bits and at 128 bits finally having a lower delay.

Table 11 Performance Comparison

Bit Size	Multiplier	Area (cells)	Delay (ns)	Power (mW)
4	CSAM	78	4.13	0.69
	CPFM	430	11.01	4.24
8	CSAM	377	9.46	5.38
	CPFM	1,217	17.86	13.67
16	CSAM	1,638	19.28	34.24
	CPFM	3,444	28.95	44.09
32	CSAM	8,456	26.49	232.09
	CPFM	9,748	46.93	142.24
64	CSAM	39,701	50.82	1,339.70
	CPFM	27,590	76.09	458.87
128	CSAM	123,560	139.70	3,300.80
	CPFM	78,087	123.36	1,480.33

The main weakness in our design is the recoding scheme used to recode the multiplicands for the signed digit adders this recoding scheme does not follow a standard algorithm and must be arbitrarily determined resulting in a mandatory exhaustive test of the multiplier do determine full functionality of the multiplier.

Future Work

Additional work on this project includes synthesizing 16, 32, 63, and 128 bit versions of the propagate-free design and comparing the multiplier design against more optimized designs, including booth multipliers. Much work should go into developing an algorithm to guide the recoding scheme since right now recoding is a hit or miss design that must be verified at each new implementation.

REFERENCES

[1] M. A. Erle, E. M. Scharz, M. J. Schulte, “Decimal Multiplication with Efficient Partial Product Generation,” *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, 2005

[2] G. W. Gerrity, “Computer Representation of Real Numbers,” *IEEE Transactions on Computers*. Vol c-31, pp. 709-714, August 1982.

[3] M. D. Ercegovac and T. Lang “Fast multiplication without carry-propagate addition,” *IEEE Transactions on Computers*., vol. 39, pp. 1385-1390, November 1990.

[4] “High Speed Computer Arithmetic,” class notes for ECEN 4013-J. E. Stine, Department of Electrical and Computer Engineering, Oklahoma State University, spring 2007

[5] M. Sudhakar, R.V. Kamala, M.B. Srinivas, “A Bit-Sliced, Scalable and Unified Montgomery Multiplier Architecture for RSA and ECC,” *IFIP International Conference on Very Large Scale Integration*, pp. 252-257, 2007

[6] J. U. Ahmed and A. A. S. Awwal, “Multiplier Design Using RBSD Number System,” Wright State University , 1993

[7] P. Sarkar, B. K. Roy, P. P. Choudhury, " VLSI implementation of modulo multiplication using carry free addition," *10th International Conference on VLSI Design*, January 1997.

[8] W. Rulling, "A Remark on Carry-Free Binary Multiplication," *IEEE Journal of Solid-State Circuits*, Vol 38, January 2003.

[9] M. D. Ercegovac and T. Lang “Comments on A Carry-free 54 b x 54 b Multiplier Using Equivalent Bit Conversion Algorithm,” *IEEE Journal of Solid-State Circuits*, Vol 38, pp. 160-161, January 2003.

[10] I. Cyliax, “One-Hot Encoding,” 2003 [online]. Available: http://archive.chipcenter.com/circuitcellar/november00/c1100lr5.htm [Accessed May 1, 2009]

[11] G. Knagge, “Booth Recoding,” May 2008 [online]. Available: http://www.geoffknagge.com/fyp/booth.shtml [Accessed May 1, 2009]

[12] I. W. Selesnic, C. S. Burrus, Extending Winograd’s Small Convolution Algorithm to Longer Lengths: Rice University, pp. 449-452

[13] E. Savas, “A carry-free Architecture for montgomery inversion,” *IEEE Transactions on Computers*. Vol 54, December 2005.

[14] B. Jose and D. Radhakrishnan, Delay Optimized Redundant Binary Adders: State University of New York, 2006

[15] G.C. Cardarilli, M. Ottavi, S. Pontarelli, m.Re, A. Salsano, "A signed digit adder with error correction and graceful degradation capabilities," *10th IEEE International On-line Testing Symposium*. 2004

[16] M. D. Ercegovac and T. Lang, "Fast Two-Operand Addition," 2003, http://www.cs.ucla.edu/digital_arithmetic/files/ch2.pdf

[17] M. D. Ercegovac and T. Lang, "Digital Arithmetic," Draft, January 2002, pp 43-113

[18] K. K. Parhi, "VLSI Digital Signal Processing Systems Design and Implementation," 1999, pp. 529-557.

[20] J. E. Stine, "Digital Computer Arithmetic Datapath Design using Verilog HDL," 2004, pp. 108-111.

[19] M. D. Ercegovac and T. Lang, “Division and Square Root Digit-Recurrence Algorithms and Implementations,” 1994, pp.121-124.

[21] M. D. Ercegovac and T. Lang, “On-the-fly conversion of redundant into conventional representations,” *IEEE Trans. Comput.,* vol. C-36 no. 7, pp. 895-897, July 1987.

VITA

JONATHAN DWAYNE HOLDEN

Candidate for the Degree of

Master of Science

Thesis: CARRY-PROPAGATE FREE COMBINATIONAL MULTIPLIER

Major Field: Electrical Engineering

Biographical:

Personal Data: Jonathan Holden is 23 years old and is the son of Randall and Mary Holden he was born in Stillwater, Oklahoma and raised in Fort Smith, Arkansas Jonathan currently resides in Stillwater, Oklahoma where he attended Oklahoma State University earning his Bachelors in Electrical Engineering in December of 2007 and his Masters of Electrical Engineering in may 2009 Jonathan has recently accepted a position with Raytheon Net Centric Systems in Mckinney, TX.

Education:

Expected to graduate with a Masters of Science in Electrical Engineering in May 2009 from Oklahoma State University – Stillwater, Oklahoma

Graduated with a Bachelor of Science in Electrical Engineering in December 2007 from Oklahoma State University – Stillwater, Oklahoma

Experience:

Defense Information Systems Agency
Information Systems Engineer Trainee (Intern) Jun. 2006 – Present

Oklahoma State University
Graduate Teaching Assistant Jan. 2008 – Present

Professional Memberships

I.E.E.E. O.S.U. Student Chapter 2003-2008

Name: Jonathan Holden Date of Degree: May 2009

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: CARRY-PROPAGATE FREE COMBINATIONAL MULTIPLIER

Pages in Study: 34 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study:

Findings and Conclusions:

Multipliers are the heart of most digital systems, however, they are quite complex devices. Standard multiplier designs in digital systems use three basic parts to compute a product, which mainly involve creating and adding partial products. Unfortunately, a significant amount of the worst-case delay attributed to larger multipliers stem from carry-propagate adders to compute the final product. This research involves modifying basic parallel multipliers, so that it can compute the final product using a redundant number notation. Using multipliers that use redundant numbers can increase the complexity of multiplication units, however, it can present designs that avoid the final carry-propagate addition. In this thesis, a design is presented that utilizes the Signed Digit notation, which is used to allow redundancy within the numbers, and subsequently avoid the final carry-propagate adder. Results using silicon standard-cell libraries indicate that for multipliers larger than *32* bits, a significant savings using the proposed architecture is shown. Comparisons versus traditional multipliers are presented and compared for analysis.

ADVISER'S APPROVAL: James Stine