12-4-2006

64 x 64 Bit Multiplier Using Pass Logic

Shibi Thankachan

64 × 64 BIT MULTIPLIER USING PASS LOGIC

by

SHIBI P.THANKACHAN

Under the Direction of A. P. Preethy

ABSTRACT

Due to the rapid progress in the field of VLSI, improvements in speed, power and area are quite evident. Research and development in this field are motivated by growing markets of portable mobile devices such as personal multimedia players, cellular phones, digital camcorders and digital cameras. Among the recently popular logic families, pass transistor logic is promising for low power applications as compared to conventional static CMOS because of lower transistor count. This thesis proposes four novel designs for Booth encoder and selector logic using pass logic principles. These new designs are implemented and used to build a 64 x 64-bit multiplier. The proposed Booth encoder and selector logic are competitive with the existing and shows substantial reduction in transistor count. It also shows improvements in delay when compared to two of the three published works.

INDEX WORDS: Algorithms, Multipliers, Booth encoder, Compressors, Wallace Tree, Adder

64 × 64 BIT MULTIPLIER USING PASS LOGIC

by

SHIBI P.THANKACHAN

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

In the College of Arts and Science

Georgia State University

2006

64 × 64 BIT MULTIPLIER USING PASS LOGIC

by

SHIBI P.THANKACHAN

Major Professor:	A. P. Preethy
Committee:	Michael Weeks
	Saeid Belkasim

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2006

Dedicated to everyone who was a part of this
for all the support

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. A. P. Preethy, for her encouragement, advice and guidance throughout my thesis work which made my graduate studies a wonderful experience of my life. I am thankful for her innovative ideas and interest in new technologies which motivated me to go forward in this prototype.

I would like to thank Dr. Saeid Belkasim and Dr. Michael Weeks for reviewing my manuscript and providing me fine pointers to meet the standards.

I would like to thank my Papa and Mommy for their prayers and advice. I would also like to thank my sisters and their families for their constant support.

I finally thank my loving husband for his valuable encouragement and support throughout the academic program. Without his co-operation it would be difficult for me to make this achievement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPA: Carry Propagation Adders

CSA: Carry Save Adders

FA: Full Adder

HA: Half Adder

LSB: Least Significant Bit

MSB: Most Significant Bit

PP: Partial Product

PPG: Partial Product Generator

MUX: Multiplexer

XCSA: XOR based Conditional Select Adder

BCGB: Block Carry Generation Block

CHAPTER 1. INTRODUCTION

VLSI designers have used static CMOS style over the past few decades to design safe and scalable circuits because of its simplicity. Classical logic design is based on a set of basic logic gates: AND, OR, NAND, NOR, NOT, etc. These design techniques, when applied to MOS designs prove to be very inefficient. CMOS circuits consist of two separate networks, one to pull up the output to logic one and the other to pull down the output to logic zero. The pull up network is connected between the output node and $V_{DD,}$ called as pMOS network (p-net). The pull down network is connected between the output node and V_{ss} and is called an nMOS network (n-net). One of the disadvantages of the CMOS logic is that, the logic is implemented twice. The n-net and the p-net both have all the information needed to implement the function. Hence, a substantial amount of area is wasted in the CMOS designs. Also, the switching capacitance of a static CMOS circuit is very large and hence is considered a drawback. Currently, there are four factors making it necessary to examine alternative design styles to static CMOS; shrinking feature sizes, increasing transistor counts, higher speed, and lower power. These factors gave rise to pass transistor-based logic families. A pass transistor is an nMOS (or pMOS) transistor with signal input fed to the drain (source) and the signal output taken from source (drain). The propagation of the signal through the transistor is controlled by a signal applied to its gate. In the case of an nMOS transistor, a logic one at the gate passes the input from source to drain circuit. A pMOS transistor exhibits similar behavior, except for a change in the control signal logic level. If signals *X* and *Y* are connected to the gate and drain of an nMOS transistor, respectively, then this is represented as *X(Y)* and read as '*X* passing *Y*'. When both nMOS and pMOS transistors are used to pass a signal *Y,* the circuit is referred to as a CMOS transmission gate.

1.1 Motivation:

Multiplication is the key in arithmetic operation and multiplier plays an important role in digital signal processing. Unfortunately, the major source of power dissipation in digital signal processors is multipliers. In the past decade, researchers developed multipliers with the help of CMOS logic, which has all the disadvantages as discussed earlier. Therefore, the design of multipliers for digital signal processing applications should be efficient while still being able to handle low-power applications. So, the proposed work is designed using pass logic principles, which shows improvements over CMOS designs. Pass logic principle based circuits are able to achieve better performance in area, power and speed when implemented in VLSI [1]. Several case studies show that pass logic principle based design implements most functions with fewer transistors which reduces the overall capacitance than static CMOS; thus, resulting in faster switching times and lower power. Pass logic principle based design is a promising alternative to static CMOS in deep sub-micron technology due to its better performance in power consumption, speed and area.

One third of the multiplier space is occupied by the Booth encoder and selector logic [1-3]. So a better design of Booth encoder and selector is vital. The main objective of this work is to design and implement new Booth encoders and selector logics which are hardware efficient and consequently power-aware. Various designs of these logic units are proposed in this work where the number of transistors needed are less when compared to previously designed units. The gate level implementations of these designs were tested for functionality using LoKon software (www.bmtmicro.com/BMTCatalog/win/LoKon.html). The pass logic implementation of all the gates (XNOR, XOR, NAND, NOR, AND, XOR-XNOR combination gate) and MUX used in these circuits were simulated and verified for functionality using TopSPICE

(www.penzar.com/TopSPICE.htm). Due to the limitation in the transistor count in the demo version of TopSPICE, it was not able to simulate the entire circuit in transistor level. Further, these designs were used to build 64 x 64 bit multiplier. The main reason for designing 64 x 64 bit multiplier is the need for higher word width for signal process applications. This design is scalable without any loss of merits. All the pass transistor circuits have been tested for fully restored voltage at the output. Hence, when these circuits are combined to form the entire multiplier, voltage drop will not cause a problem.

This thesis is structured as follows. After the introduction in Chapter 1, Chapter 2 explains the conventional architecture of the multiplier, the basic components and their functions. It also throws light on the radix-4 algorithm which is used for Booth encoding purpose. Chapter 3 discusses the various researcher's designs and also points out the area used in terms of number of transistors. Chapter 4 discusses about the proposed work which includes various Booth encoder and selector logic designs. Chapter 5 suggests the design of entire multiplier using these proposed works together with the compressor and carry propagation adder. Chapter 6 deals with the results which show hardware reduction in terms of transistor counts for Booth encoder and selector logic circuit. The final section deals with the conclusion and the future work.

CHAPTER 2. MULTIPLIER ARCHITECTURE

A multiplier has two stages. In the first stage, the partial products are generated by the Booth encoder and the partial product generator (PPG), and are summed by compressors. In the second stage, the two final products are added to form the final product through a final adder.

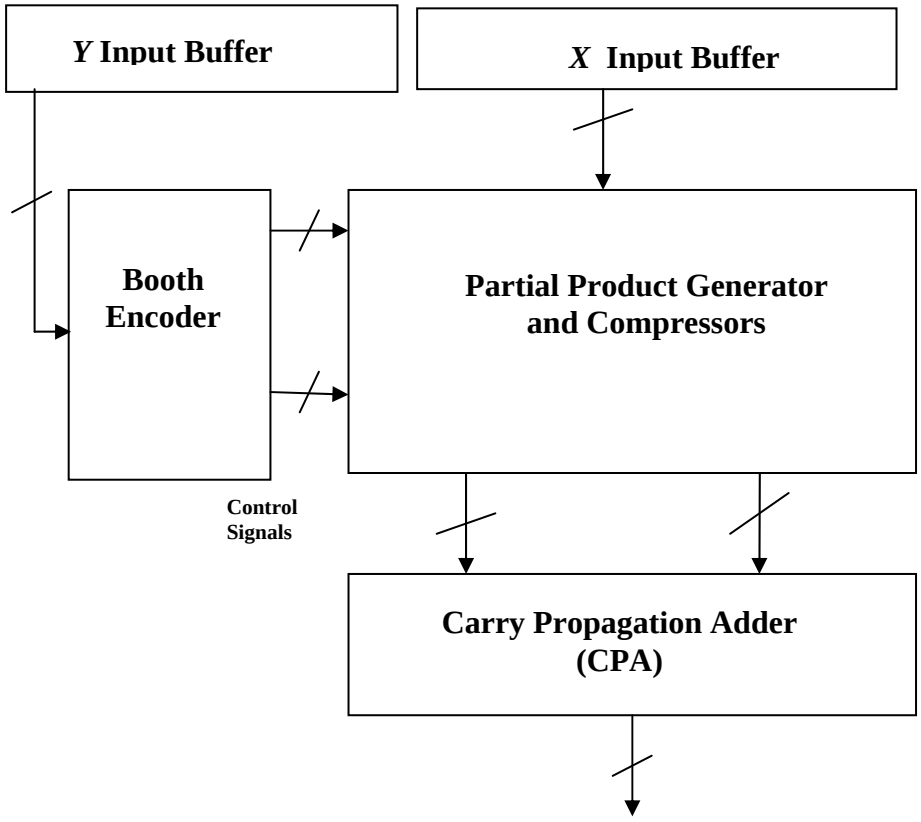

Figure 1. Block Diagram of Multiplier Architecture

The block diagram of traditional multiplier is depicted in Figure 1. It employs a booth encoder block, compression blocks, and an adder block. *X* and *Y* are the input buffers. *Y* is the multiplier which is recoded by the Booth encoder and *X* is the multiplicand. PPG module and compressor form the major part of the multiplier. Carry propagation adder (CPA) is the final

adder used to merge the sum and carry vector from the compressor module. Each block is further explained in this chapter in detail.

2.1 Booth Encoder and Partial Product Generator

Partial product generation is the very first step in binary multiplier. Partial product generators for a conventional multiplier consist of a series of logic AND gates as shown in Figure 2.

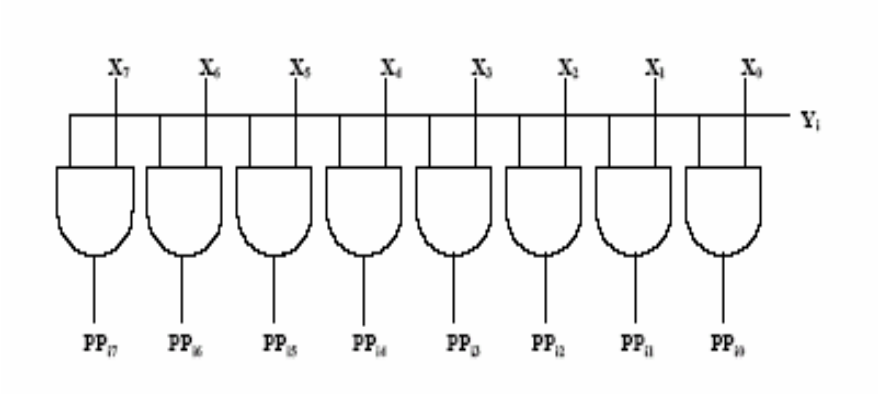

Figure 2. Partial Product generator using AND gates [4]

If the multiplier bit is '0', then partial product row is also zero, and if it is '1', then the multiplicand is copied as it is. From the second bit multiplication onwards, each partial product row is shifted one unit to the left. In signed multiplication, the sign bit is also extended to the left.

2.2 Booth's Algorithm:

A.D. Booth proposed Booth encoding technique for the reduction of the number of partial products [1]. This algorithm is also called as Radix-2 Booth's Recoding Algorithm. Here the multiplier bits are recoded as Z_i for every i^{th} bit Y_i with reference to Y_{i-1} .This is based on the fact that fewer partial products are generated for groups of consecutive zeros and ones. For a group of consecutive zeros in the multiplier there is no need to generate any new partial product. We only need to shift previously accumulated group partial product one bit position to the right for every 0 in the multiplier.

The radix-2 algorithms results in these observations [4]:

(a) Booth observed that whenever there was a large number of consecutive ones, the corresponding additions could be replaced by a single addition and a subtraction

$$2^j + 2^{j-1} + \ldots\ldots\ldots\ldots + 2^{i+1} + 2^i = 2^{j+1} - 2^i$$

(b) The longer the sequence of ones, the greater the savings.

(c) The effect of this translation is to change a binary number with digit set [0, 1] to a binary signed-digit number with digit set [-1, 1].

The Radix-2 Booth algorithm Table 1 is give below:

Table 1. Radix-2 Booth recoding [4]

$\mathbf{Y_i}$	$\mathbf{Y_{i-1}}$	$\mathbf{Z_i}$	**Explanation**
0	0	0	No string of 1s in sight
0	1	1	End of string of 1s in Y
1	0	1	Beginning of string of 1s in Y
1	1	0	Continuation of string of 1s in Y

In this algorithm the current bit is Y_i and the previous bit is Y_{i-1} of the multiplier Y_{n-1} Y_{n-2}..... Y_1 Y_0 are examined in order to generate the ith bit Z_i of the recoded multiplier Z_{n-1} Z_{n-2} **.....**Z_1 Z_2. The previous bit Y_{i-1} serves only as the reference bit. The recoding of the multiplier bits need not be done in any predetermined order and can be even done in parallel for all bit positions. The observations obtained from the radix-2 Booth recoding are listed below:

- It reduces the number of partial products which in turn reduces the hardware and delay required to sum the partial products. It adds delay into the formation of the partial products.

- It works well for serial multiplication that can tolerate variable latency operations by reducing the number of serial additions required for the multiplication.
- The number of serial additions depends on the data (multiplicand)
- Worst case 8-bit multiplicand requires 8 additions
- 01010101 $\Leftrightarrow$ 1 -1 1 -1 1 -1 1 -1
- Parallel systems generally are designed for worst case hardware and latency requirements. Booth-2 algorithm does not significantly reduce the worst case number of partial products.

Radix-2 Booth recoding is not directly applied in modern arithmetic circuits; however, it does help in understanding the higher radix versions of Booth's recoding. It doesn't have consecutive 1s or -1s. The disadvantages of the radix-2 Booth algorithm can be overcome by using Modified Booth algorithm.

2.3 Modified Booth Algorithm:

The radix-2 disadvantages can be eliminated by examining three bits of *Y* at a time rather than two. The modified Booth algorithm is performed with recoded multiplier which multiplies only $\pm a$ and $\pm 2a$ of the multiplicand, which can be obtained easily by shifting and/or complementation. The truth table for modified Booth recoding is shown below:

Table 2. Radix-4 Booth Recoding [2]

Y_{i+1}	Y_i	Y_{i-1}	Z_{i+1}	Z_i	$Z_{i/2}$	Explanation

0	0	0	0	0	0	No string of 1s in sight
0	0	1	0	1	1	End of strings of 1s
0	1	0	0	1	1	Isolated 1
0	1	1	1	0	2	End of string of 1s
1	0	0	-1	0	-2	Beginning of string of 1s
1	0	1	-1	1	-1	End a string, begin a new one
1	1	0	0	-1	-1	Beginning of string of 1s
1	1	1	0	0	0	Continuation of string of 1s

The main advantage of the modified Booth algorithm is that it reduces the partial products to *n/2*.

The following gives the algorithm for performing sign and unsigned multiplication operations by using radix-4 Booth recoding.

Algorithm: (for unsigned numbers)

- Pad the LSB with one zero
- Pad the MSB with two zeros if n is even and one zero if *n* is odd
- Divide the multiplier into overlapping groups of 3-bits
- Determine partial product scale factor from modified Booth-2 encoding table
- Compute the multiplicand multiplies
- Sum partial products

Algorithm: (for signed numbers)

- Pad the LSB with one zero

- If *n* is even don't pad the MSB (*n/2* PP's)
- Divide the multiplier into overlapping groups of 3-bits
- Determine partial product scale factor from modified Booth-2 encoding table
- Compute the multiplicand multiplies
- Sum partial products

Booth recoding is fully parallel and carry free. It can be applied to design a tree and array multiplier, where all the multiples are needed at once. Radix-4 Booth recoding system works perfectly for both signed and unsigned operations.

2.4 Compressors

A Carry-Save Adder (CSA) is a set of one-bit full adders, without any carry-chaining. Therefore, an *n*-bit CSA receives three n-bit operands, namely *a (n-1)..a (0), b (n-1)..b (0), and* c_{in} *(n-1)..*c_{in} *(0),* and generates two n-bit result values, *sum (n-1)..sum (0) and* c_{out} *(n-1)..* c_{out} *(0).*

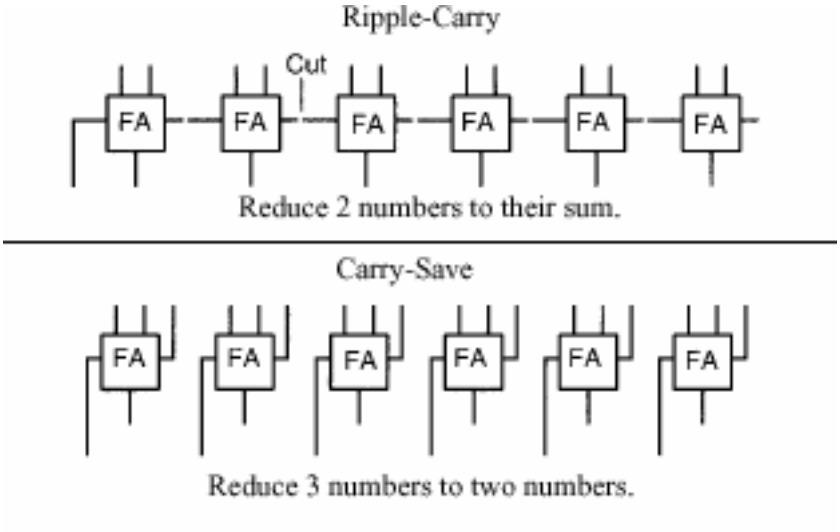

Figure 3. Carry Save Adders [4]

A carry save adder tree can reduce *n* binary numbers to two numbers having the same sum in *O (log n)* levels. Carry save adder is also called a compressor and a Wallace Tree is

constructed with CSAs. Wallace trees are CSAs in a tree structure used as a compressor. The most important application of a carry-save adder is to add the partial products in integer multiplication. From CSA separate sum and carry vector are obtained. In CSA, the output carry is not passed to the neighboring cell but is saved and passed to the cell one position down.

2.5 Carry Propagation Adder

The final step in completing the multiplication procedure is to add the final terms in the final adder. The Carry Propagation Adder, CPA, is a final adder used to add the final carry vector to the final sum vector partial products to give the final multiplication result. This is normally called a “Vector-merging” adder. The choice of the final adder depends on the structure of the accumulation array. Various fast adders can be used as CPA. Some of them are Carry look-ahead adder, Simple carry skip adder, Multi level carry skip adder, Carry- select adder, Conditional sum adder and Hybrid adder. A Carry look-ahead Adder is an adder used in digital logic. All the carry outputs are calculated at once by specialized look-ahead logic. But requires generate and propagate signals. Simple carry skip adders looks for the cases in which carry out of a set of bits are identical to carry in. Circuits for binary adders to efficiently skip a carry bit over two or more bit positions with two or more carry-skip paths is called multilevel carry skip adders. In the 4-bit carry select adder there are two 4-bit adders each of which takes a different preset carry-in bit. The two sums and carry-out bits that are produced are then selected by the carry-out from the previous stage. In conditional sum adder, sum and carry outputs at the first stage assume the previous carry to be zero and sum and carry outputs at the second stage assume the previous carry to be one. For CPA we can also combine any of these adders as a hybrid adder.

CHAPTER 3. RELATED WORK

Fast multipliers are imperative for high speed and low power signal processing systems and hence much thrust have been given to different design techniques. As explained in Chapter 2 multiplier consists of a Booth encoder, compressors, and carry propagation adders. The speed of the multiplier can be enhanced by reducing the number of partial products and thus the Booth algorithm plays a major role. In this chapter, we discuss about the related literature works for number of Booth encoder and the selector logic and the several design methods used to reduce the partial products.

Booth encoding is a technique that leads to smaller, faster multiplication circuits, by recoding the numbers that are multiplied. It is the standard technique used in chip design, and provides significant improvements over the "long multiplication" technique. The widely used Booth algorithm is the radix-4 based modified Booth algorithm proposed by McSorley where it reduces the partial products into half. As the number of partial products reduces the number of CSAs required for the compression module, the height of the Wallace tree is also reduced.

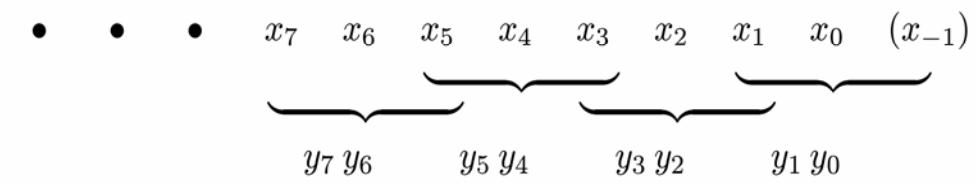

Figure 4. Modified Booth recoding pattern [4]

Modified Booth algorithm's basic idea is that the bits Y_i and Y_{i-1} are recoded into Z_i and Z_{i-1}, while, Y_{i-2} serves as reference bit. In a separate step, Y_{i-2} and Y_{i-3} recoded into Z_{i-2} and Z_{i-3} with, Y_{i-4} serving as reference bit. This signifies that the modified Booth's encoding partitions input Y into a group of 3-bits with 1-bit overlap and generates the following five signed digits, 2, 1, 0, -1 and -2. Encoding on the each group reduces the number of partial products by factor of 2.

Operations on the encoded digits performed with multiplier input *X* is illustrated in Table 3.

Table 3. Partial Product Selections and Operations [4]

Recoded digit	Booth's operation on *X*	Y_{2i-1} Y_{2i} Y_{2i+1}
0	Add 0 to PP	{0 0 0, 1 1 1}
+1	Add *X* to PP	{0 0 1, 0 1 0}
+2	Shift *X* left & add to PP	{0 1 1}
-1	Add 2's complementary *X* to PP	{ 1 0 1, 1 1 0}
-2	2's complementary *X* & shift-add	{1 0 0}

An example for radix-4 modified Booth algorithm is shown in Figure 5 [18]

A			01	00	01			17
X	×		11	01	11			−9
Y			$0\bar{1}$	10	$0\bar{1}$			recoded multiplier
			$-A$	$+2A$	$-A$			operation
Add $-A$	+		10	11	11			
2-bit Shift		1	11	10	11	11		
Add $2A$	+	0	10	00	10			
			01	11	01	11		
2-bit Shift			00	01	11	01	11	
Add $-A$	+		10	11	11			
			11	01	10	01	11	− 153

Figure 5. Example for a Modified Booth multiplication [4]

There are *n/2* = 3 steps in this multiplication and in each step two multiplier bits are considered. As a result, all shift operations are two bit positions shift and an additional bit for storing the correct sign is required to properly handle the addition of *2A*.

3.1 Booth Encoder and PPG proposed by Ohkubo

Ohkubo, *et al.*, developed a CMOS multiplier using pass transistor multiplexer.

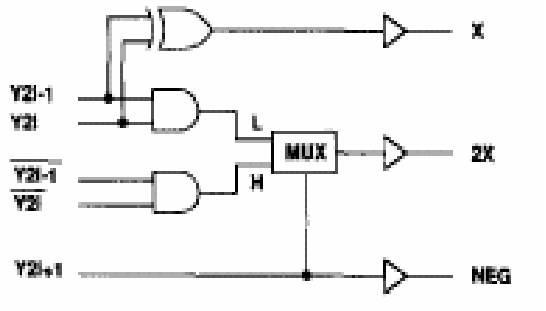

Figure 6. Booth encoder [2]

There were three control signals for complement, shifting and direction. The complement signal was generated by XOR function and the Shift by the AND and MUX operation. The partial products were obtained by the NAND and XOR operations.

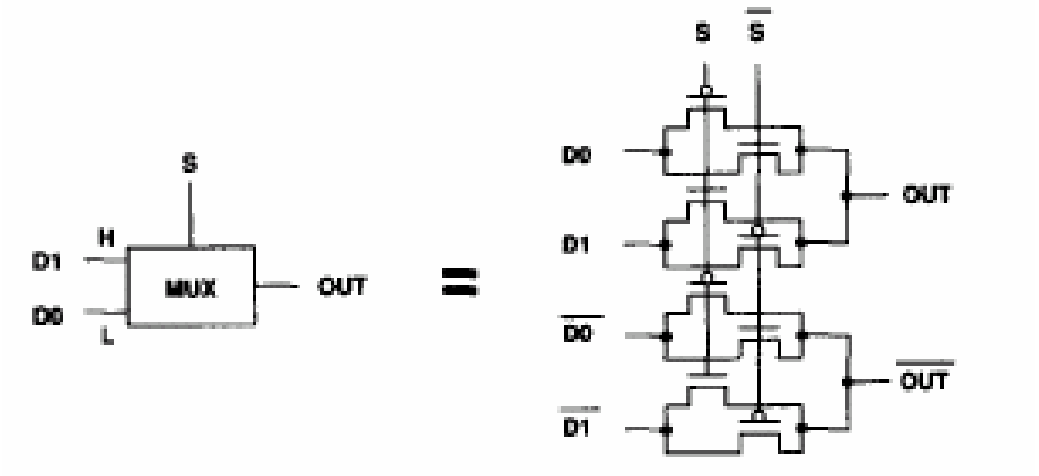

Figure 7. Pass- transistor multiplexer circuit [2]

The multiplexer used in Booth encoder itself used 8 transistors which used separate transistors to design nMOS and pMOS.

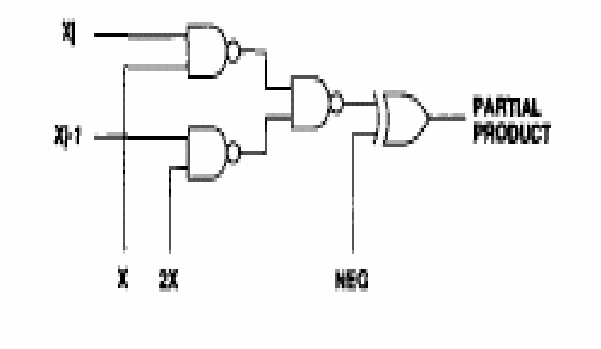

Figure 8. Partial Product Generator [2]

The PPG was implemented using NAND and XOR gates. Here the inputs were the control signals generated by the Booth encoder and these signals were used to output the data inputs X_i and X_{i-1}.

Ohkubo, *et al.*, work provided a speed advantage over conventional CMOS circuits because the critical path gate stages were minimized using pass transistor multiplexer. The drawback of Ohkubo's work was that it consisted of more transistors and it produced unnecessary glitches by the partial product generator. According to his design; any change in the value of the partial products also caused a change all along the multiplier array, and the final adder. This energy dissipation associated with the glitches in the modified Booth algorithm was an important portion of the total energy dissipation of the whole multiplier and the issue has been dealt by Fried [7] in his work. The total number of transistors for the encoder and selector logic added up to 48 transistors which occupied a large amount of space.

3.2 Booth Encoder and PPG proposed by Goto

Goto, *et al.,* was successful in reducing the number of transistors when compared with Ohkubo's work [3]. In Goto's work, there were two control signals used for generation of sign of the partial product: M_j (for negative) and PL_j (for positive). The modified Booth Selector required four multiplexers which consumed a large area. Booth encoder and PPG module constitute one third part of the entire multiplier design. In fact, Goto's work used the multiplicand as the select signals in the selector, which was very different from the conventional method which used the encoded signals as the select signals. However, encoded signals ran through the two multiplexers in series, thus incurred more delay than some other multipliers which were developed in later periods. Five gates were needed on the critical path. The truth table for the Booth encoding as per Goto's work is given in Table 4.

Table 4. Booth encoding [3]

Inputs				Usual			Sign select			
b_{j+1}	b_j	b_{j-1}	Func.	X_j	$2X_j$	M_j	X_j	$2X_j$	PL_j	M_j
0	0	0	0	0	0	0	0	1	0	0
0	0	1	+A	1	0	0	1	0	1	0
0	1	0	+A	1	0	0	1	0	1	0
0	1	1	+2A	0	1	0	0	1	1	0
1	0	0	-2A	0	1	1	0	1	0	1
1	0	1	-A	1	0	1	1	0	0	1
1	1	0	-A	1	0	1	1	0	0	1
1	1	1	0	0	0	1	0	1	0	0

z

$(j = 0,2,4,\ldots,n-4,n-2)$

$P = A \times B$

$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$

$B = -b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j$

PL_j : Positive partial product
M_j : Negative partial product
X_j : Not doubled
$2X_j$: Doubled

Here inputs are b_{j+1}, b_j and b_{j-1}. The Booth encoder had four outputs and the selector had two outputs. The design also used a number of inverters which resulted in power consumption. In Goto's work two signals had to be activated at the same time to perform a single operation. For example when $+2A$ was needed the PL_j and $2X_j$ signals were active, and the logical product of PL_j and $2X_j$ choose $+2A$ as the partial product. When $-A$ was needed, the logical product of M_j and X_j choose the correct partial product. So this caused complexity as well as making larger delay path.

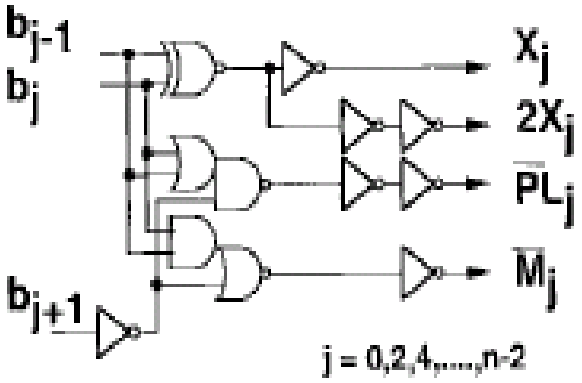

Figure 9. Booth encoder [3]

The Booth encoder consists of AND, XOR, NOR and NAND operations. Number of inverters was also used in this circuit. The outputs obtained are the control signals for complement and the shift. Two separate signals for positive and negative are also generated.

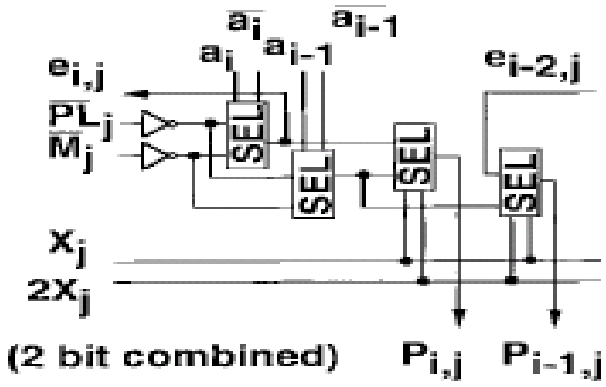

Figure 10. Selector logic [3]

The SEL component used here performed the multiplexer action. The main disadvantage of Goto's work was that encoded signals ran through the two multiplexers in series and it incurred more delay than some other multipliers which were developed in later periods. Five gates were needed on the critical path.

3.3 Booth Encoder and PPG proposed by Fried

The unnecessary glitches from Goto's design were eliminated by Fried's design of a new two-gate-delay implementation of the Booth encoder and partial product generator. He proposed two approaches to eliminate the unnecessary glitches in the Booth algorithm. One was to latch all the partial products and allow them to change only after steady-state was reached in the encoder and partial product generator. This was achieved by using a clock derivative from the global clock, whose duty cycle was defined according to the slowest path in the Booth implementation. However, this approach required large area and dissipates a lot of energy by itself. The second approach was to synchronize all the paths in Booth encoder and partial product generator.

Table 5. Truth table for race-free Booth algorithm [5]

Input Signals			Output Signals			
Y_{2i+1}	Y_{2i}	Y_{2i-1}	NEG	X1	X2P	ZP
0	0	0	0	0	1	1
0	0	1	0	1	0	1
0	1	0	0	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	1	1	0	0
1	1	0	1	1	0	1
1	1	1	1	0	1	1

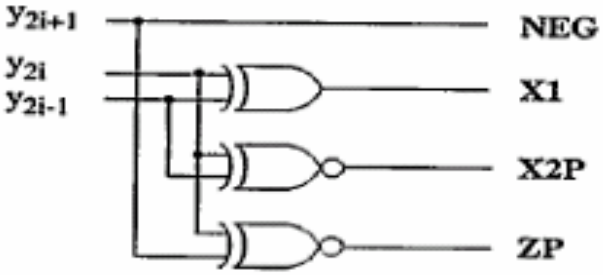

Figure 11. Booth encoder [5]

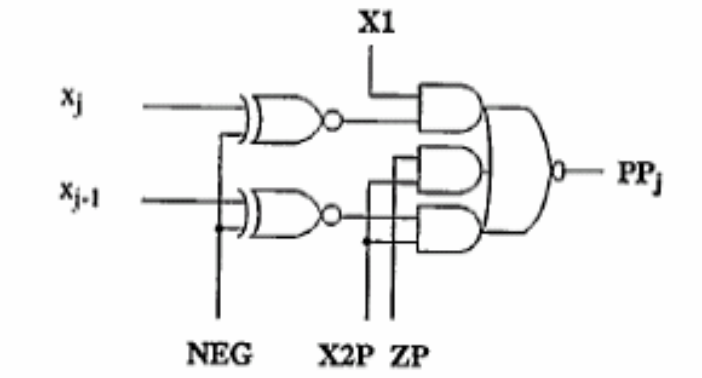

Figure 12. Partial product generator [5]

In the Booth encoder XOR-XNOR gates were used to generate the control signals for the PPG. Four control lines were used in the PPG for each row to get the required output. The load, for each column in each row, on *XI* and *X2P* was one gate, and *NEG* was loaded with two gates. The additional control line *ZP* was loaded with one gate. All the paths were equalized to have exactly same propagation delay by using only XOR-XNOR gates till the last stage. But the penalty for this fast and race-free implementation was the higher transistor count for the partial product generators. The full CMOS implementation of the partial product generator consisted of 24 transistors when compared to only 15 for the conventional implementation.

3.4 Booth Encoder and PPG proposed by Groβschädl

The partial product generator developed by Groβschädl was used for two different types of operands; integers and binary polynomials. For integer mode it was done by modified Booth recoding technique and for polynomial by a digital serial polynomial multiplier.

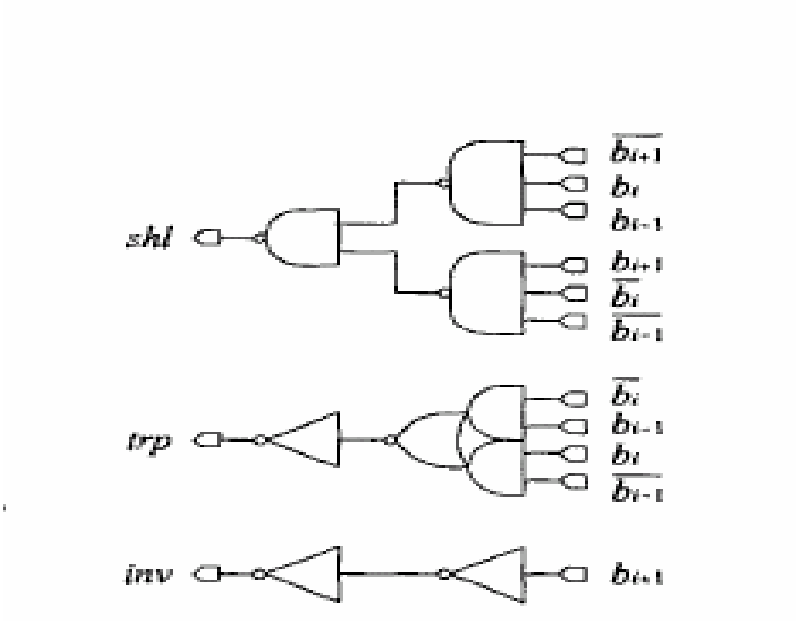

Figure 13. Booth encoder [6]

For encoding multiplier was partitioned into overlapping groups of three bits (b_{i+1}, b_i, b_{i-1}) with i= 0, 2, 4, 6, …. Each group had its own encoder circuit which produced the control signal *inv* (invert), *trp* (transport, and *shl* (shift left). When control signal *inv* = 1 then the PP is negative. When control signal *trp* =1 means the PP = ± *A* (no shift left). When *shl* = 1, a one bit left-shift was performed. The PP= 0 was generated by *trp* = *shl* = 0.

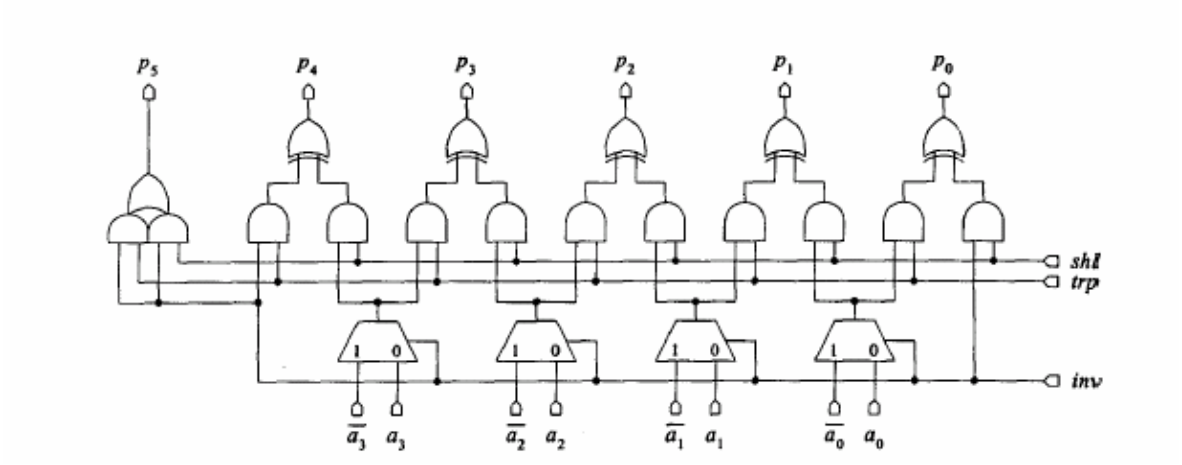

Figure 14. Partial Product Generator using radix -4 [6]

The PPG required *A* (the multiplicand) and *A'* as an input, and the multiplexers selected between *A* and *A'*. The ANDOR gates performed a left- shift if multiplication by -2 or 2 was desired.

But the Booth encoder and PPG circuit consisted of large number multiplexers, inverters, AND gates and XOR gates. As a result the circuit used more number of transistors when compared to some other designs.

3.5 Booth Encoder and PPG proposed by Cho

In 2003, Cho, *et al.,* developed a new Booth encoder and the selector with a fewer number of components. They developed a new encoder based on the modified Booth algorithm. Table 6. shows the truth table of the operations developed by Cho [1]:

Table 6. Truth Table for Booth encoding

Ym+1	Ym	Ym-1	Booth Op.	Dir.	Sht.	Add.
0	0	0	0x	0	0	0
0	0	1	1x	0	-	1
0	1	0	1x	0	-	1
0	1	1	2x	0	1	0
1	0	0	-2x	1	1	0
1	0	1	-1x	1	-	1
1	1	0	-1x	1	-	1
1	1	1	-0x	1	0	0

In their design they described Booth function as three basic operations, which they called *'direction', 'shift',* and '*addition'* operation.

Direction determined whether the multiplicand was positive or negative, *shift* explained whether the multiplication operation involved shifting or not and *addition* meant whether the multiplicand was added to partial products. The expressions for Booth encoding were stated below as [1]:

Direction, $D_m = Y_{m+1}$;

Shift, $S_m = Y_{m-1} \cdot (Y_{m+1} \oplus Y_m) + Y_{m-1}' \cdot (Y_{m+1} \oplus Y_m)$

$= Y_{m+1} \oplus Y_m$;

Addition, $A_m = Y_{m-1} \oplus Y_m$;

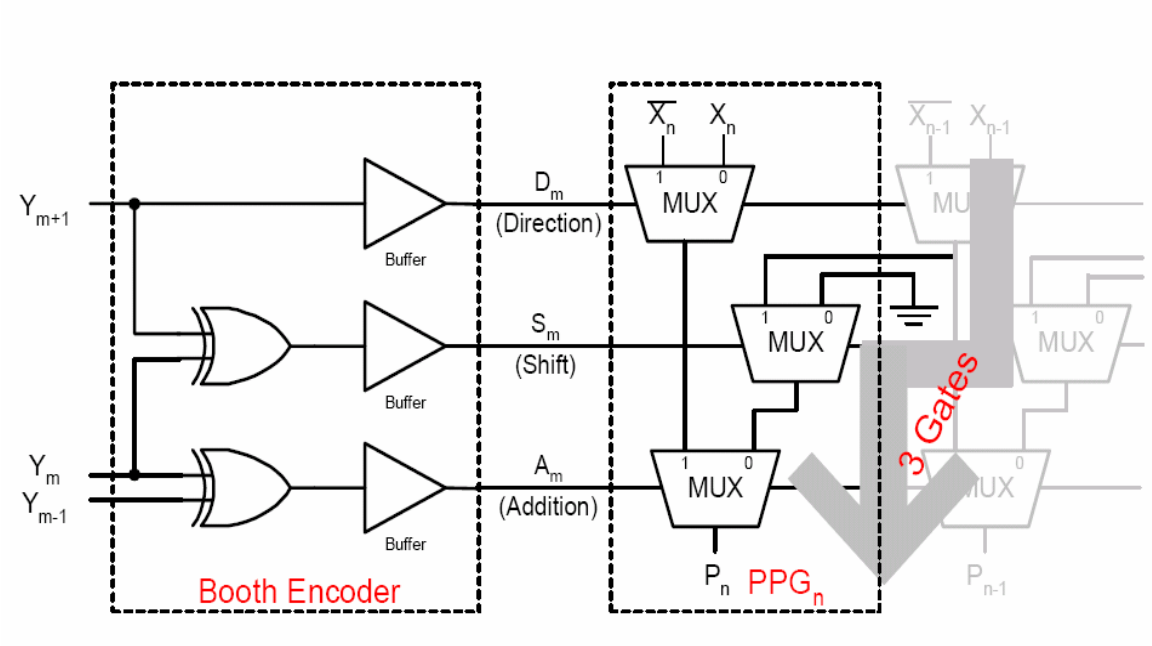

Figure 15. Booth encoder and PPG

The Booth encoder was implemented using two XOR gates and the selector using 3 MUXes and an inverter which counted to a total of 40 transistors.

Careful optimization of the partial-product generation can lead to some substantial delay and hardware reduction. Keeping this in mind, some designs are proposed in Chapter 4.

CHAPTER 4. PROPOSED WORK

4.1 Booth Encoder Module

For the design of a faster multiplier, we should either reduce the number of partial products or increases the summation of partial products. The Booth algorithm reduces the number of partial products. Based on the available literature, we propose a few designs of the Booth encoder and selector logic. The proposed designs are based on modified Booth recoding system using radix-4 multiplication where it reduces the number of partial products to half. The multiplicands are replicated and separate carry and sum vectors are obtained at the output of the compressor. Hence Booth recoding is fully parallel and carry free. Moreover, it can be applied to design a tree and array multiplier, where all the multiples are needed at once. Radix-4 Booth recoding system works perfectly for both signed and unsigned operations.

The Booth encoder constitutes one third part of the multiplier circuit, so it is significant to have an efficient design for the partial product generator. Modified Booth algorithm successfully proved to reduce the partial products by half. To further enhance the performance of the multiplier in terms of power, area and delay, pass logic principle can be incorporated. Novel Booth encoder designs using pass logic principle are proposed in this section which combines the benefits of low power consumption and reduced chip area when compared to other conventional designs.

In Cho's design [1], the Booth encoder consisted of two XOR gates and PPG consisted of three MUXes and one inverter which count to a total of 40 transistors. In order to compute the number of transistors sown in Figure 13, it has been redrawn using CMOS logic. CMOS circuit using a Booth encoder with the operational expressions mentioned above is shown in Figure 14

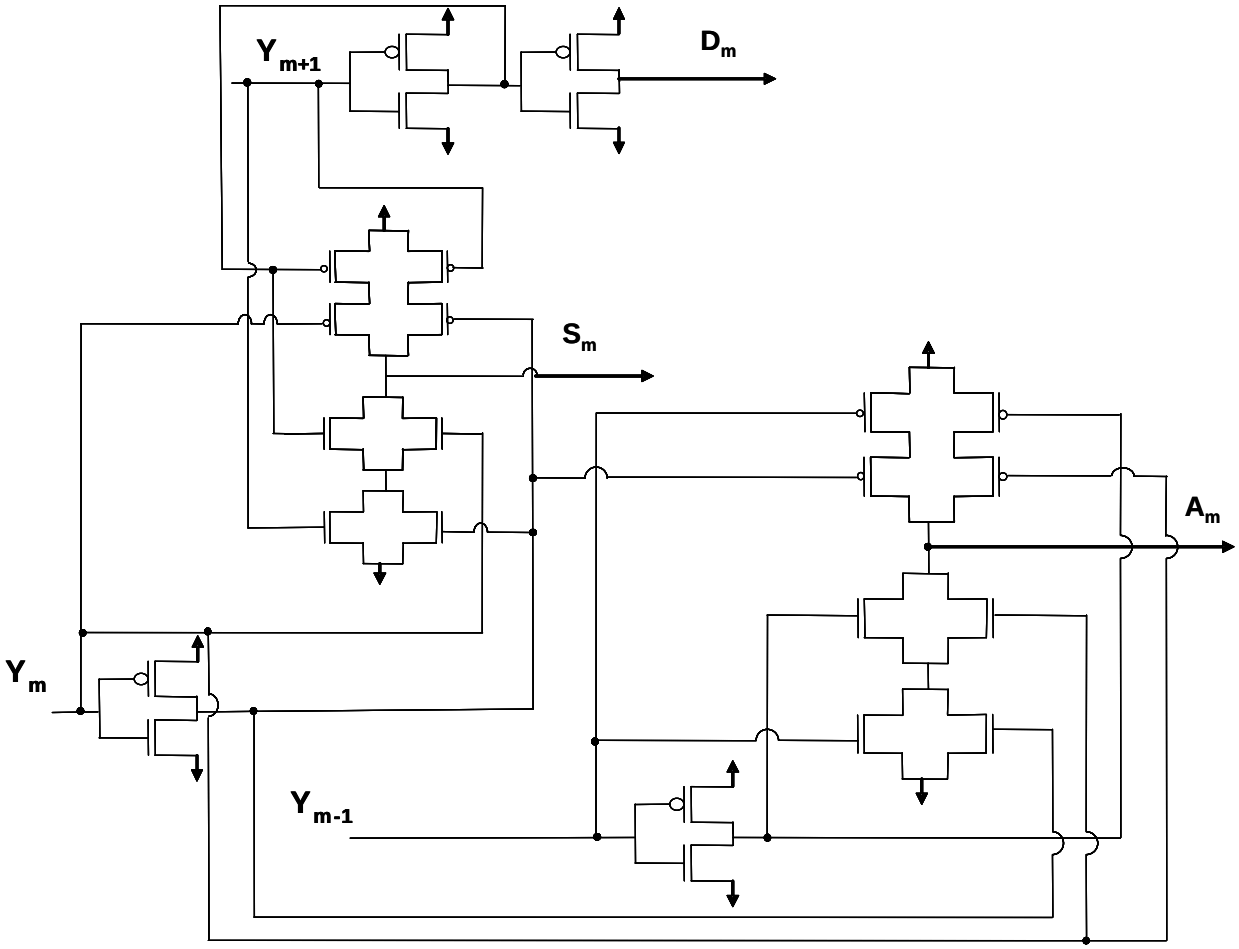

Figure 16. CMOS implementation of Booth Encoder

The Booth encoder itself shown above has a total of 26 transistors including three inverters. Conventional static CMOS is reliable, robust and noise tolerant, but, today's VLSI design trends are bringing requirements of increased speed and reduced power dissipation. Accordingly, many researchers have investigated the use of pass logic based principle designs to achieve the speed low power dissipation. In order to get the best design for Booth encoder and selector logic, we tried different techniques and successfully came up with four final designs.

The proposed designs are named – Two MUX- NAND Design, Three MUX- XOR Design, MUX- NAND Design and MUX- AND Design. The first part denotes the number of

MUXes in the selector logic and the second part denotes the logic gates used to select data inputs, X_n or X_{n-1}.

4.2 Two MUX- NAND Design:

In Two MUX- NAND design, the inputs are multiplier bits Y_{m+1}, Y_m and Y_{m-1}.

Table 7. Truth table of Two MUX- NAND Design

Inputs			Operation	**Outputs**			
Y_{m+1}	**Y_m**	**Y_{m-1}**		**Neg**	**SFT**	**U**	**ADD**
0	**0**	**0**	**0**	**0**	**1**	**1**	**0**
0	**0**	**1**	**x**	**0**	**0**	**1**	**1**
0	**1**	**0**	**x**	**0**	**0**	**0**	**1**
0	**1**	**1**	**2x**	**0**	**1**	**0**	**1**
1	**0**	**0**	**-2x**	**1**	**1**	**0**	**1**
1	**0**	**1**	**-x**	**1**	**0**	**0**	**1**
1	**1**	**0**	**-x**	**1**	**0**	**1**	**1**
1	**1**	**1**	**0**	**1**	**1**	**1**	**0**

Neg, SFT and *ADD* are the control signals. *U* is an intermediate signal added to get the desired operations using the input signals Y_{m+1}, Y_m and Y_{m-1}. The *Neg* signal is same as the input signal Y_{m+1} and it shows whether the partial product is positive or negative. *SFT* signal is the shift signal used to select between data inputs X_n and X_{n-1} where X_{n-1} is the shifted version of X_n. When SFT = 0 X_n is passed down the MUX and when *SFT = 1* X_{n-1} is passed down the MUX. In the truth table *SFT = 1* for *Y*= 000 and 111 even though no shifting operation is needed for these combinations. This is done to get the XNOR implementation of

SFT signal. But this will not hurt the encoding process since it is blocked at the second MUX level on the selector logic. *SFT* signal also determines the selection of *X* or *2X* operation. The *ADD* signal is active when the addition process takes place. The *ADD* signal can be configured to determine the other operations like 0, ±*X* and ±*2X*. When *ADD*=0, PP_n inhibits the addition. When *ADD*=1, ±*X* and ±*2X* are produced as PP_n. The schematic diagram of the design is in Figure 17.

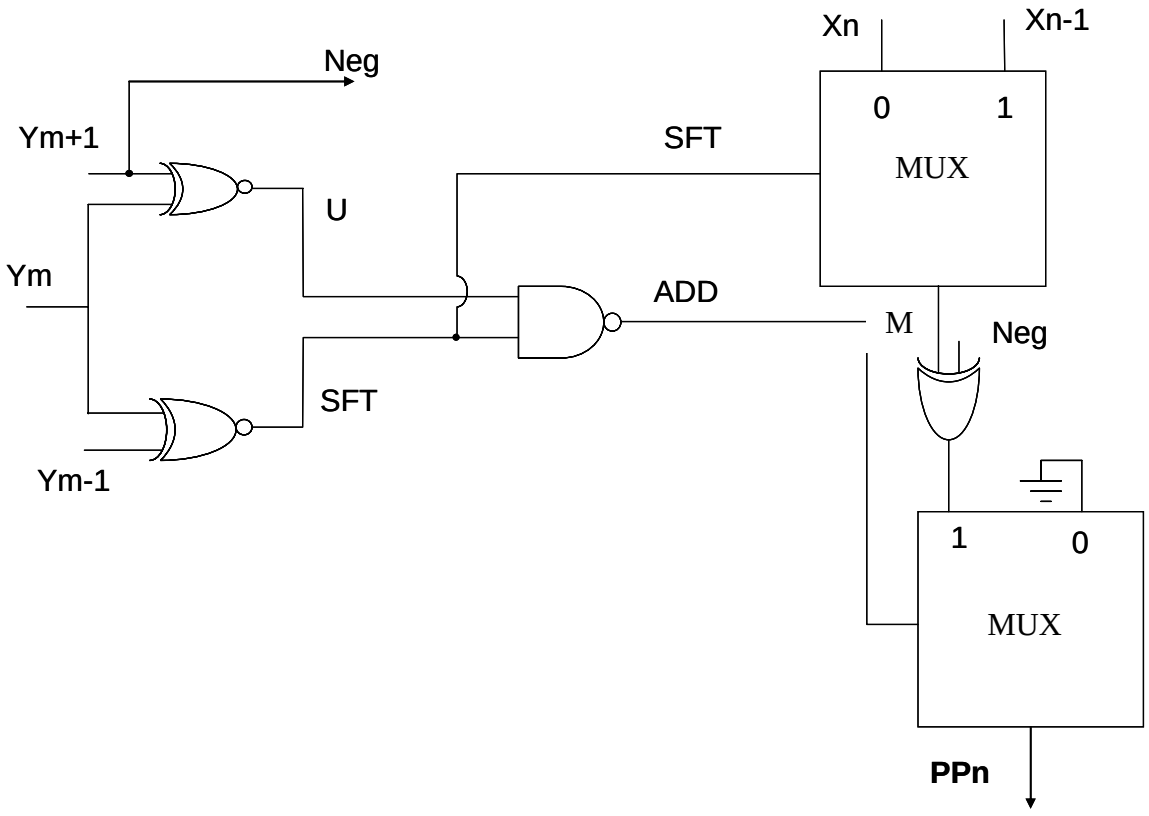

Figure 17. Block Diagram of Two MUX- NAND Design

The *SFT* signal is obtained by the XNOR operation of Y_m and Y_{m-1}. The U is an intermediate signal obtained by the XNOR operation of Y_m and Y_{m+1}. The *ADD* signal is easily generated by NAND operation of *U* and *SFT*. *ADD* signal selectively outputs the PP_n. For contiguous number of ones and zeros, the ADD signal will be zero thus outputting a zero as PP_n or one otherwise.

The XOR in the PPG is used to selectively complement the signals. The XOR in the PPG is used to complement the signals whenever necessary. *U* and *SFT* signals are obtained by the XOR-XNOR operation [8] of Y_m and Y_{m+1} and Y_m and Y_{m-1} respectively. The implementation of Two MUX- NAND Design in pass logic circuit is shown in Figure 18.

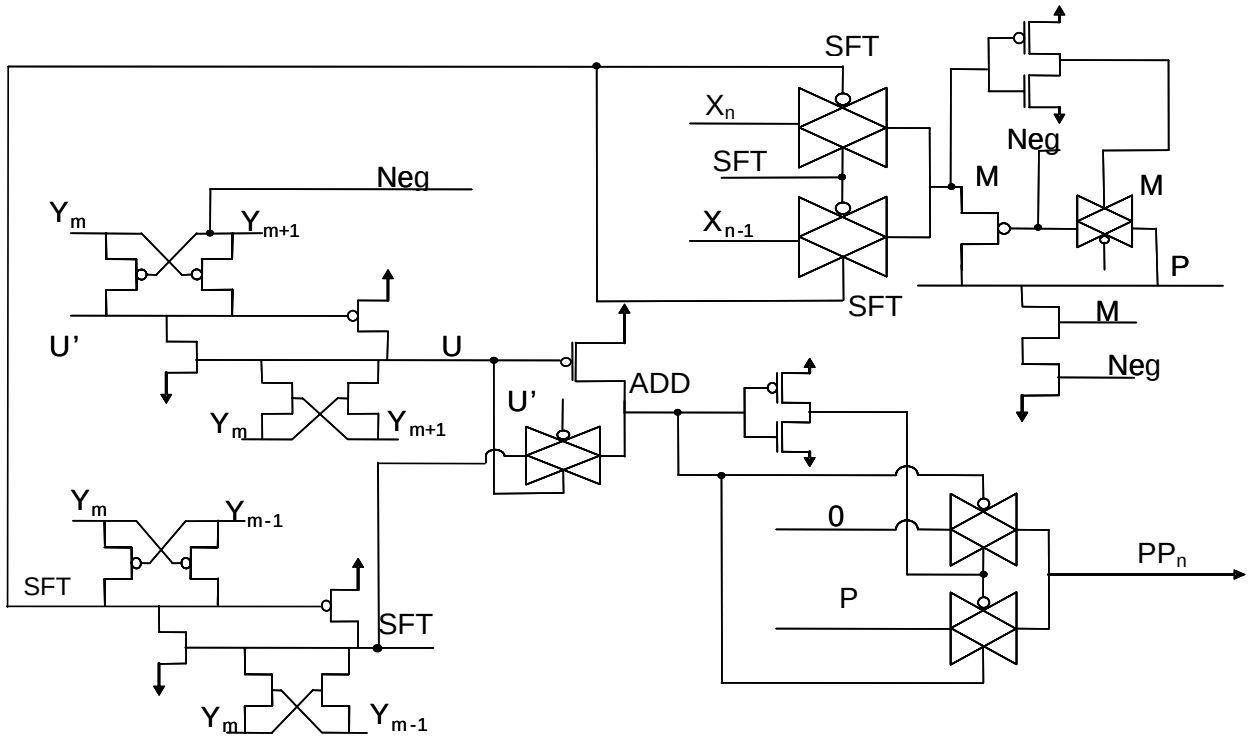

Figure 18. Two MUX- NAND Design using pass logic principles

The partial product generation is simplified using these encoded signals. For example, when *Y* = 010, *SFT* = 0 and it selects X_n data from MUX1 and it passes through the XOR gate. The XOR gate will complement the signal only if *Neg* = 1. At this instant *ADD* = 1 and X_n is obtained at the output. The output is obtained only when *ADD* signal is 1. *SFT, ADD* and *Neg* signals together determine whether *0, ±X or ±2X* should be produced at the output. In this implementation XOR-XNOR circuit is used to generate control signals. MUX, NAND and XOR are implemented using transmission gates. So a fully restored output is obtained. The transistor

count for the Booth encoder is 17 and the selector is 15. When compared with Cho's 20 transistors for selector part we saved 5 transistors for one bit. So for a 64 x 64 bit multiplier we saved 320.

4.3 Three MUX - XOR Design:

This design uses the input signals Y_{m+1}, Y_m and Y_{m-1} to generate three control signals which generates the partial products.

Table 8. Truth Table of Three MUX- XOR Design

Inputs			Operation	**Outputs**			
Y_{m+1}	Y_m	Y_{m-1}		**Neg**	**SFT**	V	**ADD**
0	**0**	**0**	**0**	**0**	**1**	0	**0**
0	**0**	**1**	**x**	**0**	**0**	**1**	**1**
0	**1**	**0**	**x**	**0**	**0**	1	**1**
0	**1**	**1**	**2x**	**0**	**1**	1	**1**
1	**0**	**0**	**-2x**	**1**	**1**	**0**	**1**
1	**0**	**1**	**-x**	**1**	**0**	**0**	**1**
1	**1**	**0**	**-x**	**1**	**0**	0	**1**
1	**1**	**1**	**0**	**1**	**1**	**1**	**0**

The *Neg* signal determines whether the partial product is negative or positive. The *SFT* signal is the shift signal used to determine the selection of *X* or *2X* operation. *V* is an intermediate signal generated for *ADD* signal. *ADD* signal is the signal which is active wherever the addition

takes place. It is also the final control signal which selects operations like 0, ±*X* and ±2*X*. The schematic diagram of the design is shown in Figure 19.

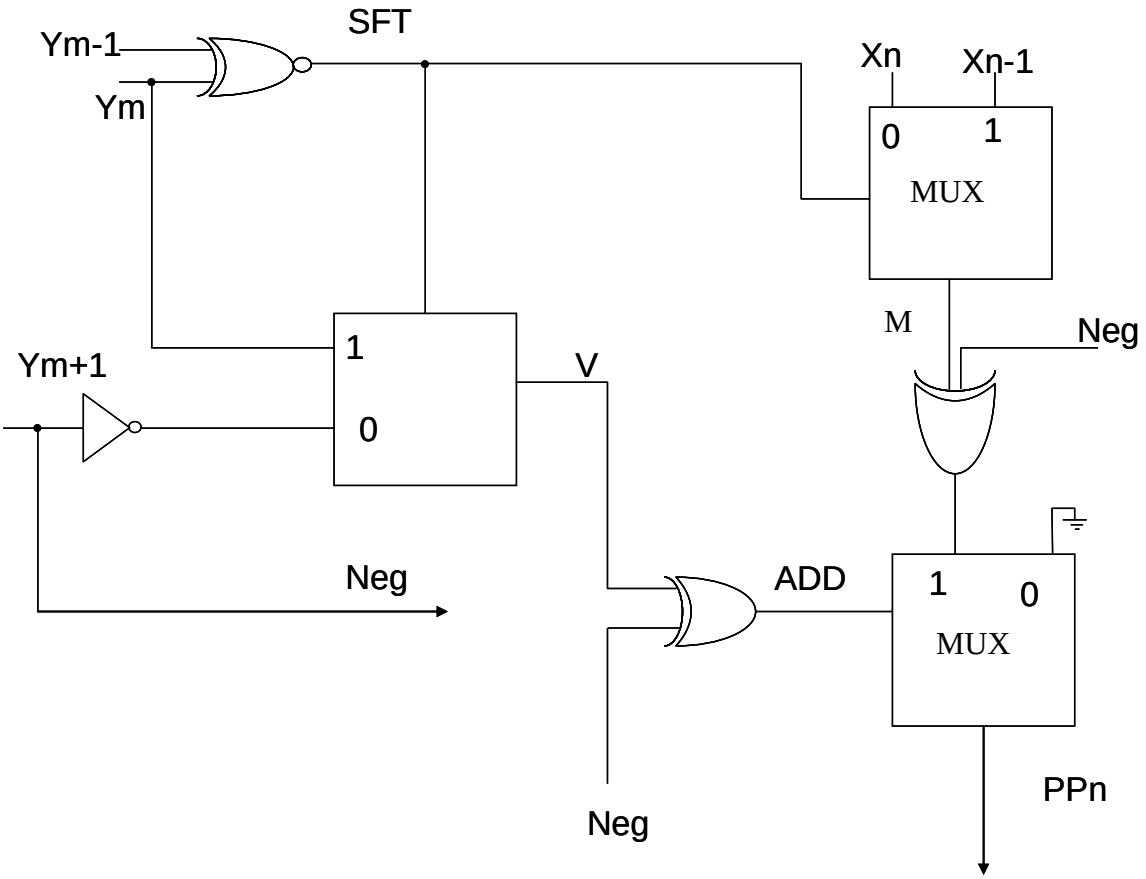

Figure 19. Block diagram of Three MUX- XOR Design

The *Neg* signal is same as $Y_{m+1.}$ The *SFT* signal is produced as an XNOR function of Y_m and $Y_{m-1.}$ The *V* signal is generated as the MUX output using inputs Y_m and Y_{m-1}'. It is then XORed with *Neg* to get *ADD* signal which selectively outputs the data. X_n and X_{n-1} are the data inputs. PPG consists of two MUX and a XOR. The XOR in the PPG is used to complement the signals whenever necessary. *M* is the output from the first MUX. In the design these components

are implemented using pass logic principles. *SFT* and XOR signals are implemented as feed back circuit [8].

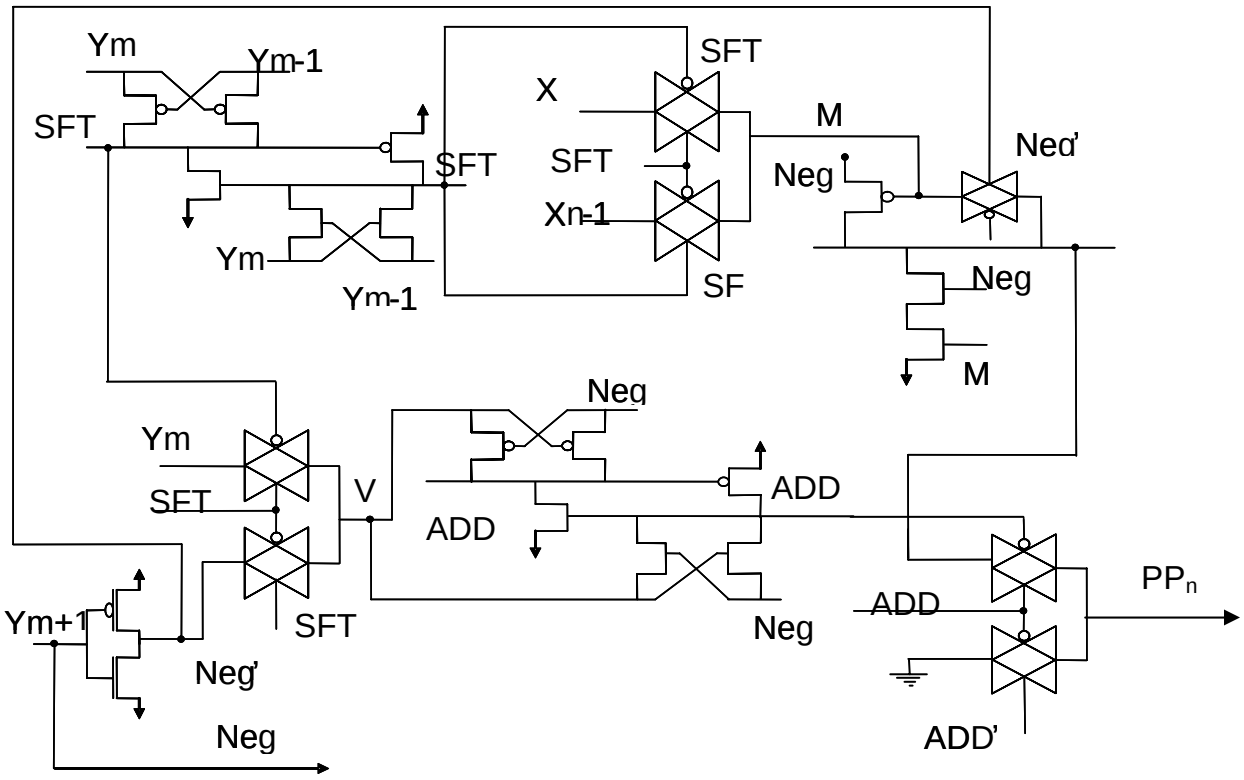

Figure 20. Pass logic implementation of Three MUX- XOR Design

Using these encoded signals, the partial products are simplified. For example, when *SFT* is 0 or 1 the signals are obtained at the output of the MUX1 and it is complemented with the *Neg* signal. But only when *ADD* signal is 1, then output we will get the *±X* or *±2X* according to whether the current Shift signal is 0 or 1, otherwise no operation is performed and zero will be the output. Here the MUX and XOR are implemented using transmission gates. So a fully restored output is obtained. The transistor count for the Booth encoder is 18 and the selector is 13. Since there will be 32 pairs of selector part this will reduce the hardware and power consumption to a large extend when compared with Cho's and other researchers work.

4.4 MUX- NAND Design:

In the MUX- NAND design, extra control signals, *U* and *ADD* are added to get the desired operations using the input signals Y_{m+1},Y_m and Y_{m-1}.

Table 9. Truth table of MUX – NAND Design

Inputs			Operation	**Outputs**			
Y_{m+1}	**Y_m**	**Y_{m-1}**		**Neg**	**SFT**	**U**	W
0	**0**	**0**	**0**	**0**	**1**	**1**	1
0	**0**	**1**	**x**	**0**	**0**	**1**	0
0	**1**	**0**	**x**	**0**	**0**	**0**	0
0	**1**	**1**	**2x**	**0**	**1**	**0**	0
1	**0**	**0**	**-2x**	**1**	**1**	**0**	0
1	**0**	**1**	**-x**	**1**	**0**	**0**	0
1	**1**	**0**	**-x**	**1**	**0**	**1**	0
1	**1**	**1**	**0**	**1**	**1**	**1**	1

The *SFT* signal is the shift signal used to determine the selection of *X* or *2X* operation. *U* and *SFT* signals are obtained by the XOR-XNOR operation [8] of Y_m and Y_{m+1} and Y_m and Y_{m-1} respectively. The *W* signal can be configured to determine the other operations like 0, ±*X* and ±*2X*. The schematic diagram of the design is shown below:

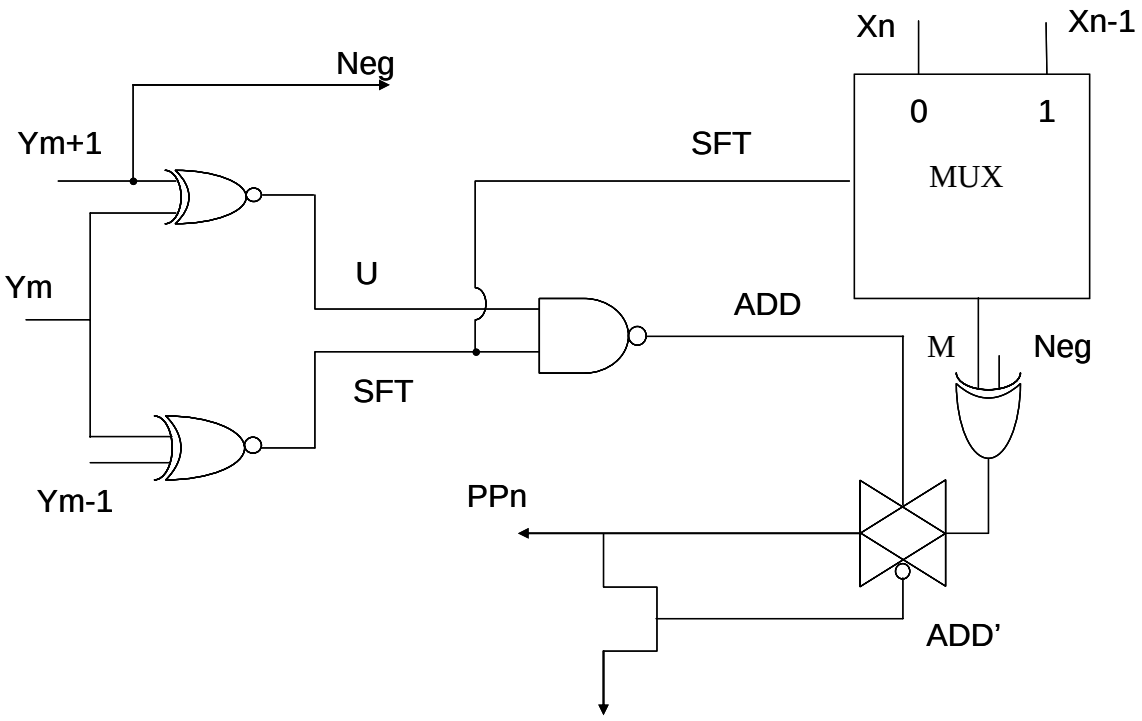

Figure 21. Block diagram of MUX – NAND Design

The *Neg* signal tells us whether the operation is positive or negative and it is same as $Y_{m+1.}$ The *ADD* signal is obtained by NAND operation of *U* and *SFT*. The *ADD* signal generates the PPG output. The XOR in the PPG is used to complement the signals whenever necessary.

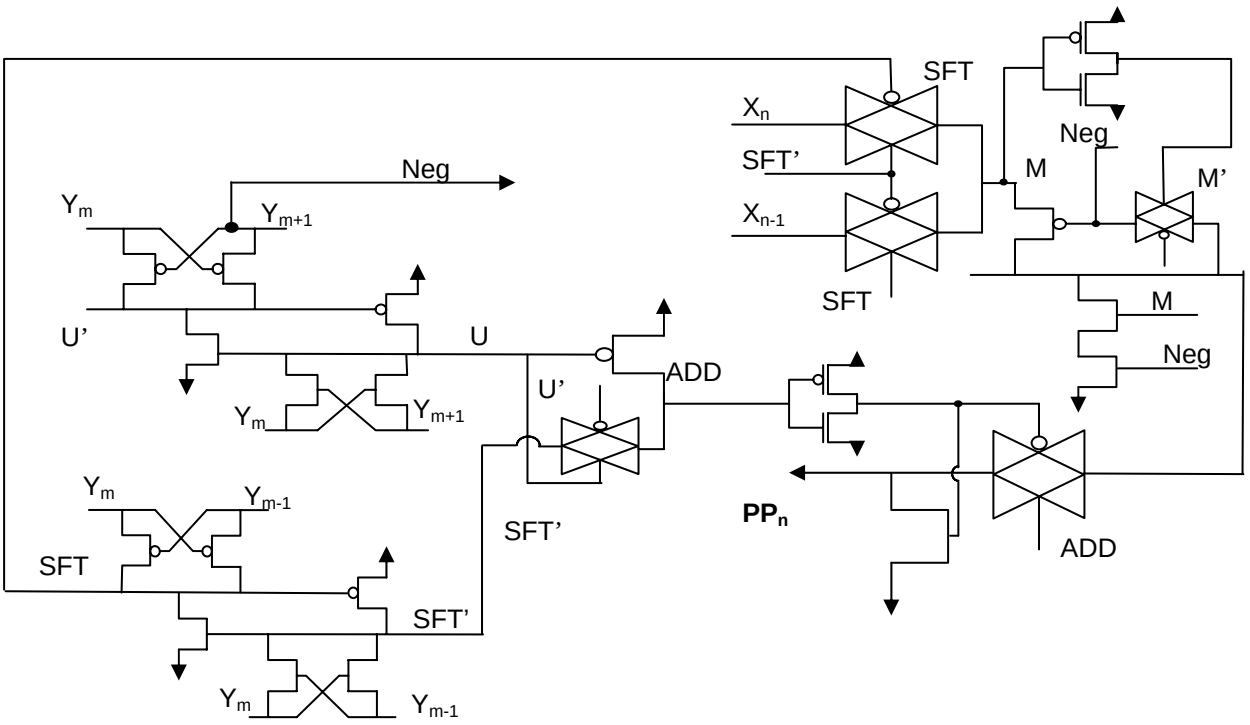

Figure 22. MUX- NAND Design using pass logic

The partial products are simplified using these encoded signals. For example, when *SFT* is 0 or 1 the signals are obtained at the output of the Mux1 and it is complemented with the *Neg* signal. But only when *ADD* signal is 1, the enable pin will be active and it passes the XOR output through it. When the enable pin is 0 then the *ADD'* signal will be active which triggers the nMOS and as a result a good 0 will pass as the output. Thus the various operation 0, $\pm X$ or $\pm 2X$ are obtained by enabling and disabling the enable pin. Here the last transmission gate and the N-type transistor form the enable pin. When ever the *ADD* signal is one the enable pin becomes active otherwise the *ADD'* will trigger the n-type transistor and it will pass a good zero to output. The Booth Encoder part will count to 17 and the selector part as 14 transistors. So the total will be 31 i.e. 9 transistors less than other researcher's work.

4.5 MUX- AND Design:

In this design, *U* and *W* are the intermediate signals, to get the desired operations using the input signals Y_{m+1}, Y_m and Y_{m-1}.

Table 10. Truth table of MUX– AND Design

Inputs			Operation	**Outputs**			
Y_{m+1}	Y_m	Y_{m-1}		**Neg**	**SFT**	**U**	W
0	**0**	**0**	**0**	**0**	**1**	**1**	1
0	**0**	**1**	**x**	**0**	**0**	**1**	0
0	**1**	**0**	**x**	**0**	**0**	**0**	0
0	**1**	**1**	**2x**	**0**	**1**	**0**	0
1	**0**	**0**	**-2x**	**1**	**1**	**0**	0
1	**0**	**1**	**-x**	**1**	**0**	**0**	0
1	**1**	**0**	**-x**	**1**	**0**	**1**	0
1	**1**	**1**	**0**	**1**	**1**	**1**	1

The *SFT* signal is the shift signal used to determine where there is any shifting in the multiplication process. *SFT* also selects the data input according to whether it is 0 or1. The *W* signal can be configured to determine the other operations like 0, $\pm X$ *and* $\pm 2X$.

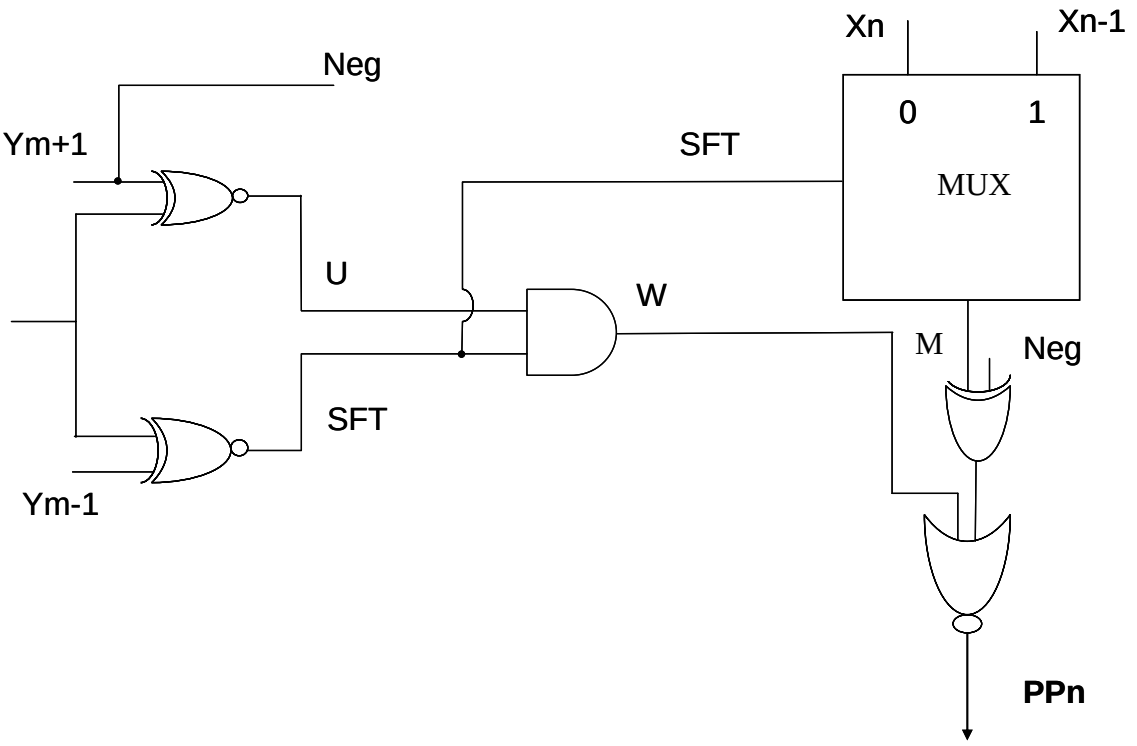

Figure 23. Block Diagram of MUX- AND Design

The *Neg* signal tells us whether the operation is positive or negative and it is same as $Y_{m+1.}$ The *W* signal is obtained by *AND* operation of *U* and *SFT*. The XOR in the PPG is used to complement the signals whenever necessary. X_n and X_{n-1} are the data inputs. *M* is the output signal from MUX. The partial product generations are simplified using these encoded signals. The *W* signal is fed to the NOR gate where the output $\pm X$ or $\pm 2X$ is available only when *W* signal is 0 which also depends on whether the current *SFT* signal is 0 or 1, otherwise no operation is performed and zero will be the output. The MUX- AND design in pass logic circuit is shown in Figure 24.

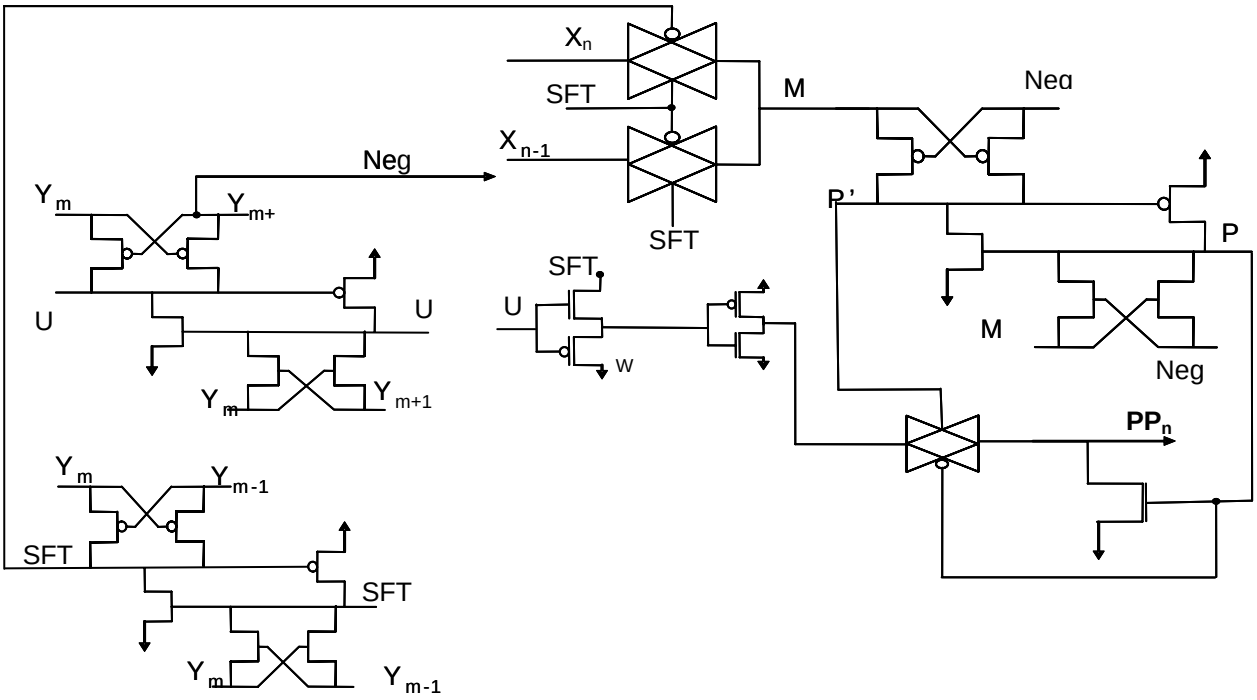

Figure 24. MUX- AND Design using pass logic implementation

Here the mainly the feed back circuit [8] of XOR-XNOR combination is used. MUX, AND and NOR are implemented by using transmission gates. So a fully restored output is obtained. The transistor count for the Booth encoder is 16 and the selector is 13. When compared with Cho's 20 transistors for selector part we saved 7 transistors.

This section discussed various Booth encoder and selector design and all these designs had total number of transistors count less than the published works.

CHAPTER 5. COMPRESSION MODULE

The next step in the multiplication process is the addition of the partial products. For this purpose carry save adders or generally called Wallace trees are used. The basic idea behind this process is as follows:

- Use only half adders in the first row (no partial product reduction)
- Reduce the partial product from eight to seven with the second row
- Reduce the partial products from seven to six with the third row
- Continue this reduction process until there are only two final partial products

Each reduction step (except the first non-reduction step) is performing by reducing the top three partial products to two partial products with an adder row. The rest of the partial products are left alone until the next reduction step.

5.1 Conventional 3:2 compressors:

The conventionally used compressors are 3:2 compressors where there are three inputs and two outputs.

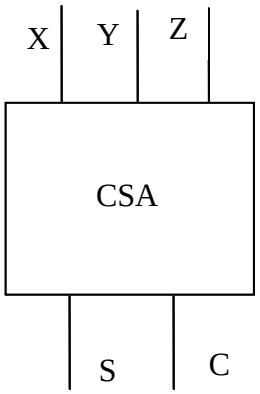

Figure 25. Block Diagram of CSA [4]

In the design of 64×64-bit multiplier, the 3:2 compressors output is shown as a tabular method in Table 11.

Table 11. 32 input Wallace Tree for 64 bit operands using 3:2 Compressors

					63	**62**	**….**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
					32	32	….	…….	32	32	32	32	32	32	32
				10	22	22	….	…….	22	22	22	22	22	22	12
			3	11	15	15	….	…….	15	15	15	15	15	12	4
		1	4	10	10	10	….	…….	10	10	10	10	9	5	2
		2	5	7	7	7	….	…….	7	7	7	7	4	3	2
		3	5	5	5	5	….	…….	5	5	5	4	3	1	2
	1	2	4	4	4	4	….	…….	4	4	4	3	1	1	2
	1	3	3	3	3	3	….	…….	3	3	3	1	1	1	2
	2	2	2	2	2	2	….	…….	2	2	1	1	1	1	2

In Wallace trees, we reduce the number of operands at the earliest opportunity, i.e., if there are *m* bits in a column, we immediately apply *m/3* full adders to that column. Since the number of bits to sum has been reduced by three fold at each level, the depth of the Wallace tree is *O (log N)*, where *N* is the initial number of bits. This tends to minimize the overall delay by making the final CPA as short as possible. Here the total number of full adders is 1910.

The delay of the fast adder is not a smoothly increasing function of the word width. In Dadda trees, we reduce the number of operands to the next lower number using the fewest number of full adders and half adders. The Table 12 below shows the maximum numbers of inputs for an *h*-level carry save adder tree.

Table 12. Comparing the delays of CSA using 3:2 and 4:2 compressors [4]

Number of Operands	Number of Levels using (3,2)	Number of Levels using (4:2)	Equivalent Delay
3	1	1	1.5
4	2	1	1.5
5-6	3	2	3
7-8	4	2	3
9	4	3	4.5
10-13	5	3	4.5
14-16	6	3	4.5
17-19	6	4	6
20-28	7	4	6
29-32	8	4	6
33-42	8	5	7.5

From the table shown above, we can see that 7, 8, or 9 operands require only 4 CSA levels. As a result, the cost of the carry save adders can be reduced and there will be an optimum view on the point of hardware.

The carry save adders redone by means of Dadda's strategy is given as Table 13..

Table 13. 32 input Dadda Tree for 64 bit operands using 3:2 Compressors

					6	**6**	**….**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
					3	3	….	……	3	3	3	3	3	3	3
				4	2	2	….	……	2	2	2	2	2	2	2
			1	1	1	1	….	……	1	1	1	1	1	1	8
			4	1	1	1	….	……	1	1	1	1	1	8	4
		1	5	9	9	9	….	……	9	9	9	9	7	5	2
		2	6	6	6	6	….	……	6	6	6	6	4	3	2
		4	4	4	4	4	….	……	4	4	3	4	3	1	2
	1	3	3	3	3	3	….	……	3	3	2	3	1	1	2
	2	2	2	2	2	2	….	……	2	1	1	1	1	1	2

By using Dadda tree the number of Full Adders (FA) is reduced to 1890. But height of the tree is 8. The height of the Wallace tree can be further reduced by using the 4:2 compressors. So in this work, 4:2 compressors are used to achieve hardware reduction.

5.2 4:2 Compressor:

To increase the speed of the partial product summation we must not only reduce the number of levels, but also assure that all the signals originated in the carry save adders of the lower positions (i = 1,..,N-1) do not contribute to the delay of the signal in the position N. Hence for this thesis the number of FAs is reduced by using 4:2 compressors.

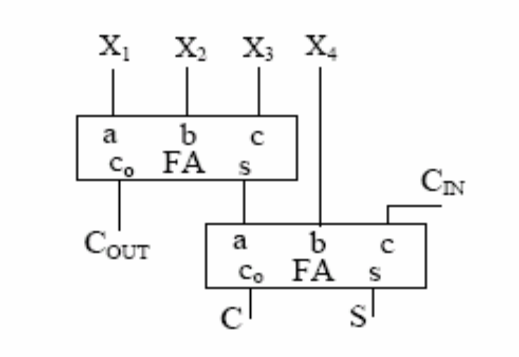

Figure 26. 4:2 Compressor [7]

A 4:2 compressor consists of five inputs and three outputs and can be implemented with two stages of full-adders connected in series as shown in Figure 24. Here we get separate sum and carry vectors as the output. The Wallace tree with 4:2 compressors is shown in the Table 14.

Table 14. 32 input Wallace Tree for 64 bit operands using 4:2 Compressors

					63	**6**	**….**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
					32	3	….	…	32	32	32	32	32	32	32
				8	1	1	….	…	16	16	16	16	1	16	8
			2	6	8	8	….	…	8	8	8	8	8	6	2
			4	4	4	4	….	…	4	4	4	4	4	2	2
		1	1	2	2	2	….	…	2	2	2	1	1	2	2

Here the number of levels is reduced to half. Here the number of full adders is reduced to 960. Moreover, an adder tree using 4:2 compressor will have a more regular structure and lower delay than a CSA using 3:2 compressor. So here the delay is only 1.5 times when compared with 3:2 compressors (Refer to Table 11). So the delay for 64×64-bit multiplier using 3:2

compressors is 8, whereas, the delay for 64×64-bit multiplier using 3:2 compressors is only 6. This Wallace tree principle can be further used to implement a 64×64-bit multiplier. 64×64-bit multiplier implementation using 4:2 compressor is shown in Figure 27.

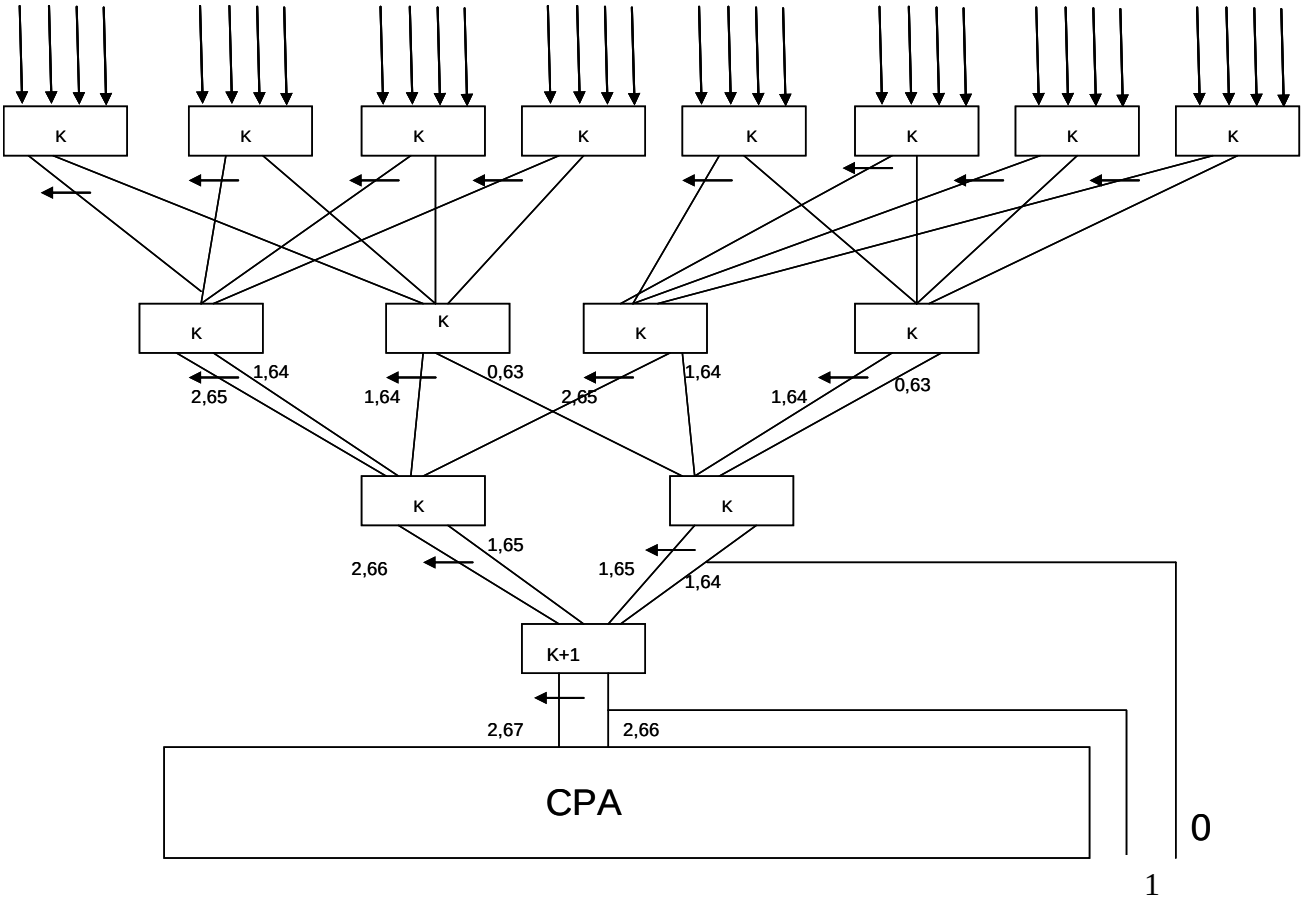

Figure 27. 4:2 CSA tree for the Wallace tree in Table 14.

The Wallace tree implemented using 4:2 compressors gave a regular structure since it's a multiple of four. Each block represents a *k* bit wide. The outputs coming from each block are the sum and the carry vectors. The left arrow used in the Figure 27 indicates the shifting of carry vector. The blocks are arranged in the order of weight. The signals can be extracted wherever possible. These vectors are then merged in the carry propagation adder.

The gate level implementation of 4:2 compressor in shown in Figure 28. The direct implementation of 4:2 compressor using CMOS logic design required seven transistors to implement each XOR gate [7]. Furthermore, the inverters used in the design increased the switching activity and hence the power consumption too.

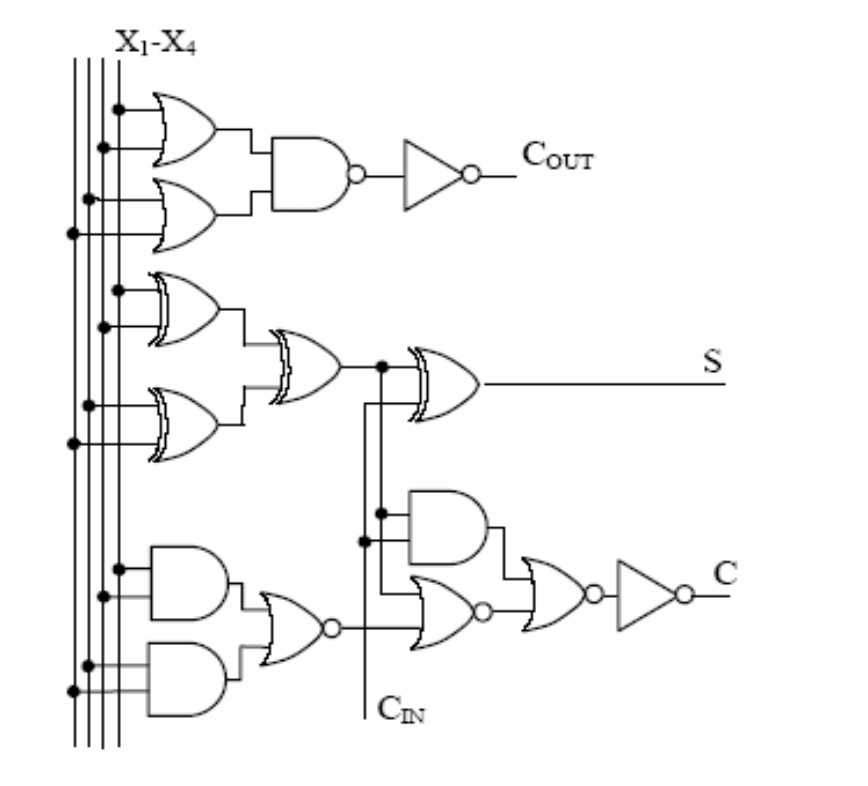

Figure 28. 4:2 Compressors using CMOS logic [8]

5.3 XOR-XNOR Implementation of 4:2 Compressors:

For achieving low power consumption and area, a 4:2 compressor developed using pass logic principles using XOR-XNOR combination[8] The sum and carry expressions are given by: $S=H \oplus C_{in}$ and $C_{out} = H'A+H\,C_{in}'$ where $H= A \oplus B$. The pass logic design equations for the sum and carry outputs are given as:

$S = H'(C_{in}) + H(C_{in}')$

$C = H'(A) + H(C_{in'})$

The block diagram of the 4:2 compressors using the pass logic principle is shown in Figure 29.

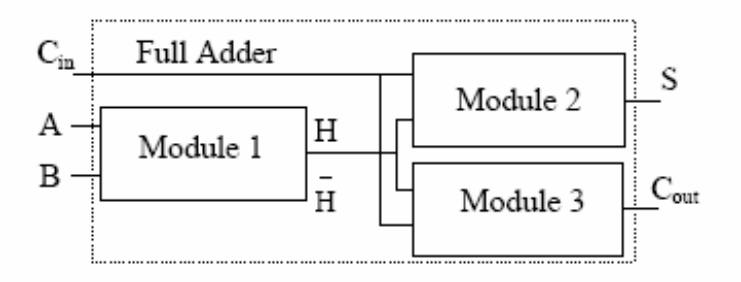

Figure 29. Block Diagram of 4:2 Compressor [7]

The equations for sum and carry are rewritten as

$S = H_3'(C_{in}) + C_{in}(H_3') + C_{in}'(H_3)$

$C = H_3'(X_4) + H_3(C_{in})$

where $H_3 = X_1 \oplus X_2 \oplus X_3 \oplus X_4$.

The diagram for the 4:2 compressors using XOR-XNOR cell is shown in Figure 30.

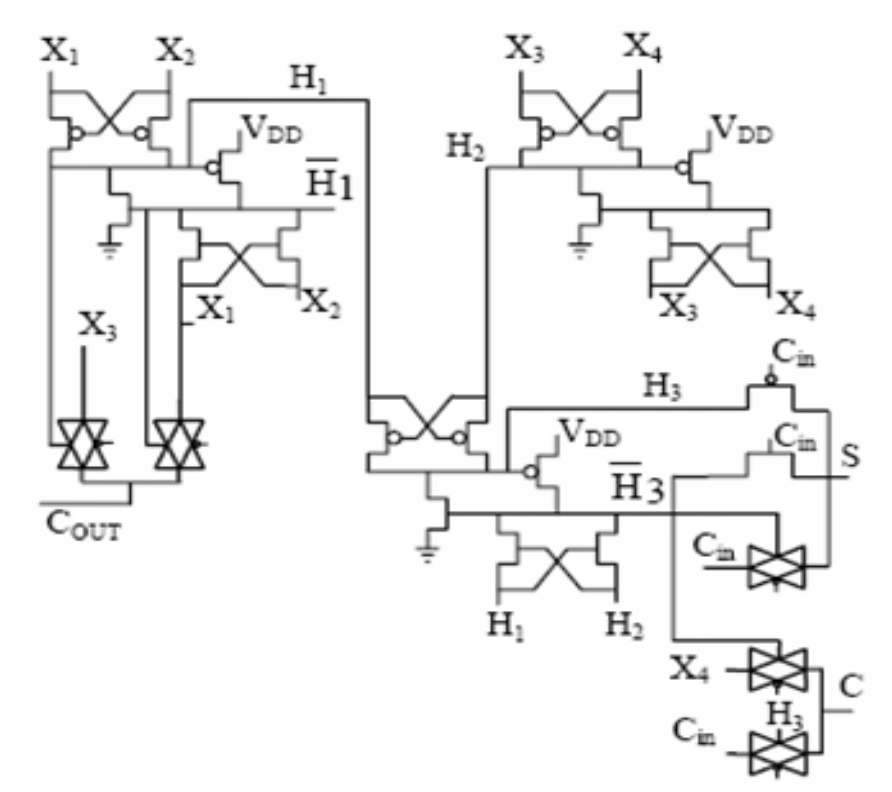

Figure 30. 4:2 compressors using XOR-XNOR cell [7]

The 4:2 compressor is constructed by coupling two circuits by feedback to generate both XOR and XNOR functions. This circuit saves two transistors when compared with its conventional design. In this circuit, due to the regenerative feedback introduced by the pull-down (nMOS) and the pull-up (pMOS) transistors, the threshold voltage drop is completely eliminated from both the outputs, thereby providing the full voltage swing at the outputs under all input conditions. But this feedback is going to adversely affect the maximum operating frequency of the circuit. Also for proper functioning of the circuit under various operating conditions the transistor sizes must be carefully chosen.

The main advantages of this circuit are listed below.

- There is no direct path from the power supply to the ground for any input combination, there by eliminating the short-circuit power component.

- The total number of capacitances generated for this cell is less than that of all the other adders.
- Reliable operation of the circuit is guaranteed when the supply voltage is scaled down.

This 4:2 compressor designed with low power pass logic based XOR-XNOR combination requires only 28 transistors while the conventional design takes up to 40 transistors. Moreover, due to the presence of both XOR and XNOR outputs, the carry generation multiplexers do not need any extra inverters and none of the inputs need any inverters. Furthermore, it provides full voltage swing at all nodes in the circuit.

5.4 Carry Propagation Adder:

The final step in completing the multiplication procedure is to add the final sum and carry vectors in the final adder. In this work, conditional select adders are used. The adder is an XOR based implementation which minimizes gate counts and critical path delay. The following expressions describe how to determine a sum and a carry using XOR function.

$Sum = A \oplus B \oplus C$

Sum = if ((A ⊕ B) = = 1) then Sum = C_{in}*';*

else if ((A ⊕ B) = = 0) then Sum = C_{in}

Carry = if ((A ⊕ B) = = 1) then $C_{out} = C_{in}$*;*

else if ((A ⊕ B) = = 0) then $C_{out} = A$*;*

The block diagram of the conditional select adder is given in Figure [31]

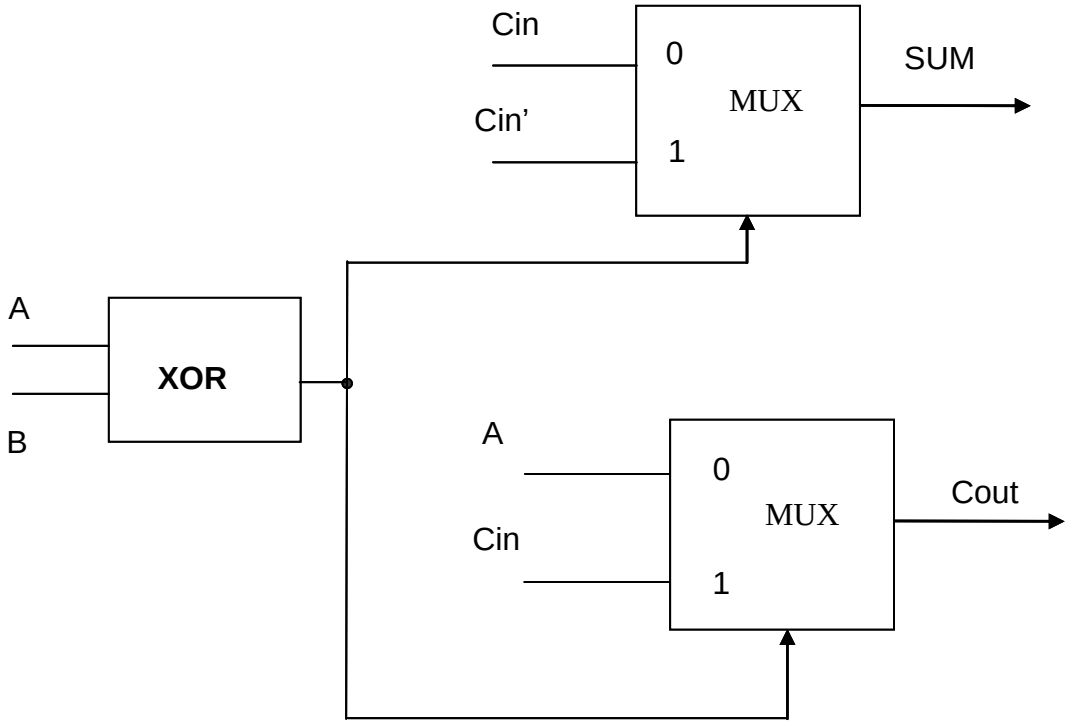

Figure 31. Conditional select adder

The adder consists of only one XOR gate and two Multipliers. The various carry propagation adders used in the existing designs are having more critical path delay than Cho's design. So in this work, we have adopted Cho's carry propagation adder for better results.

According to Cho's design, fourteen XOR based conditional select adder (XCSA) blocks and a separated carry generation block were combined to make the carry propagation adder. Each modularized XCSA consists of an 8-bit sum generator and a carry generator. The carries of each XCSA are transmitted to the block carry generation block (BCGB).

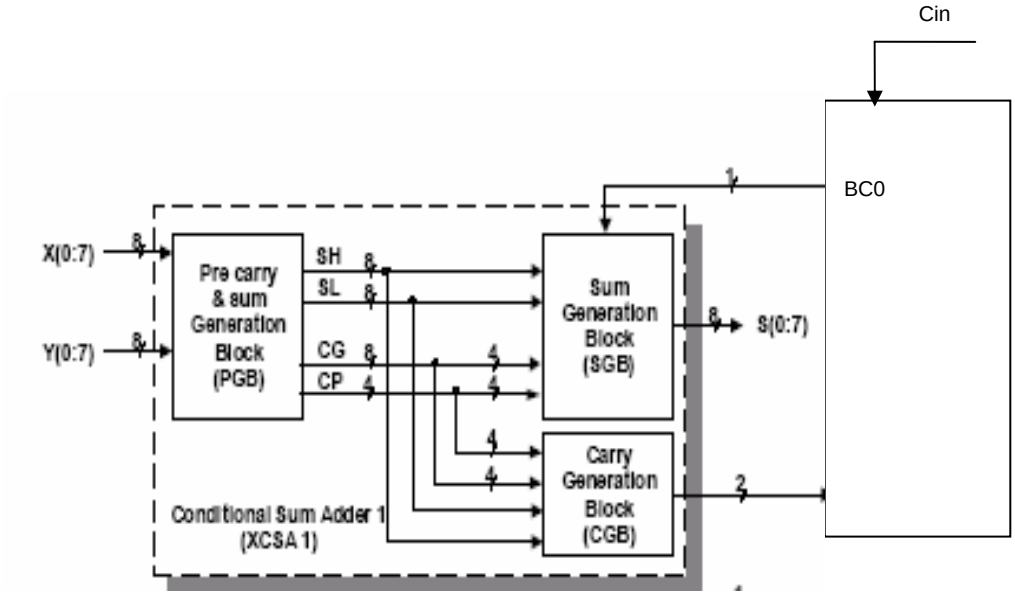

Figure 32. Conditional Select Adder Block [1]

The XCSA has only 10 gate delays when compared with other designs. Goto and Ohkubo's work explained earlier in the related works are having 12 and 13 respectively.

CHAPTER 6. RESULTS

6.1 Comparison of Booth Encoders and selectors

The comparison of the proposed designs of the Booth encoder and the selector logic with the existing designs is shown in Table 15. The novel designs of Booth encoder and the selector show substantial reduction in hardware.

Table 15. Comparison of Booth encoders and selectors

	Ohkubo, *et al.*, Work [2]	Goto, *et al.*, Design [3]	Cho, *et al.*, Design [1]	Proposed Two MUX-NAND Design	Proposed Three MUX-XOR Design	Proposed MUX-NAND Design	Proposed MUX – AND Design
Critical Path (gate)	6	5	3	4	4	4	4
Booth Encoder (transistor count for one bit pair)	30	36	20	17	18	17	16
Selector (transistor count for one bit)	18	32	20	15	13	14	13
Total	48	68	40	32	31	31	29

The proposed designs use only 13 transistors when compared to 18 to 32 in the existing designs for selector logic for one bit. Since encoder and selector part occupies one third of the entire multiplier architecture, considerable reduction of hardware can be achieved through these proposed designs. Comparing MUX – AND Design with Goto, *et al.*, Design, 20 transistors were saved for one bit pair. The proposed designs saved 2 to 30 transistors when compared with the published Booth encoders. Similarly, for the selector logic 5 to 19 transistors were saved for one bit when matching with the existing designs. When comparing with Cho's work for Booth encoder, for one bit pair MUX- AND Design saved 4 transistors. So for a 64 x 64 bit multiplier, there are 32 pairs of Booth encoder and hence a total of 128 transistors are saved. Similarly, with the selector logic, 7 transistors are saved for one bit pair. So for 64 x 64 bit multiplier, there are 64 selector logic parts and a total of 448 transistors are saved. So a total of 576 transistors are saved for one 64 x 64 bit multiplier.

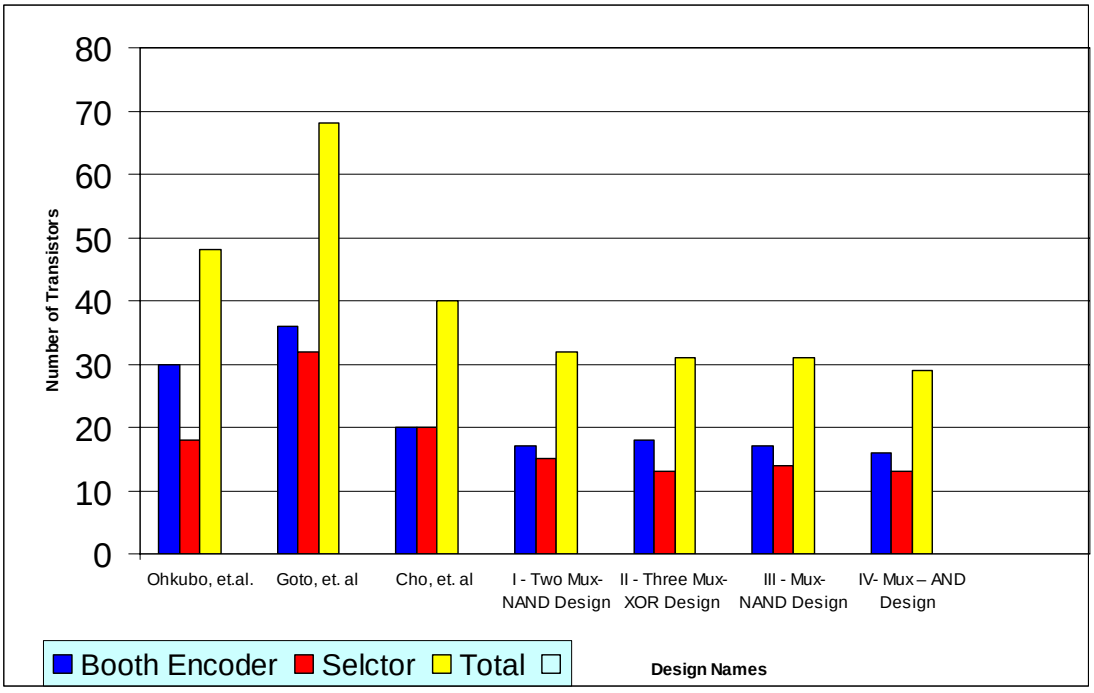

Figure 33. Comparison of Proposed Booth encoder and selector logic designs with existing designs

Figure 33 shows that the proposed designs give an improvement in the hardware reduction when compared with the existing designs.

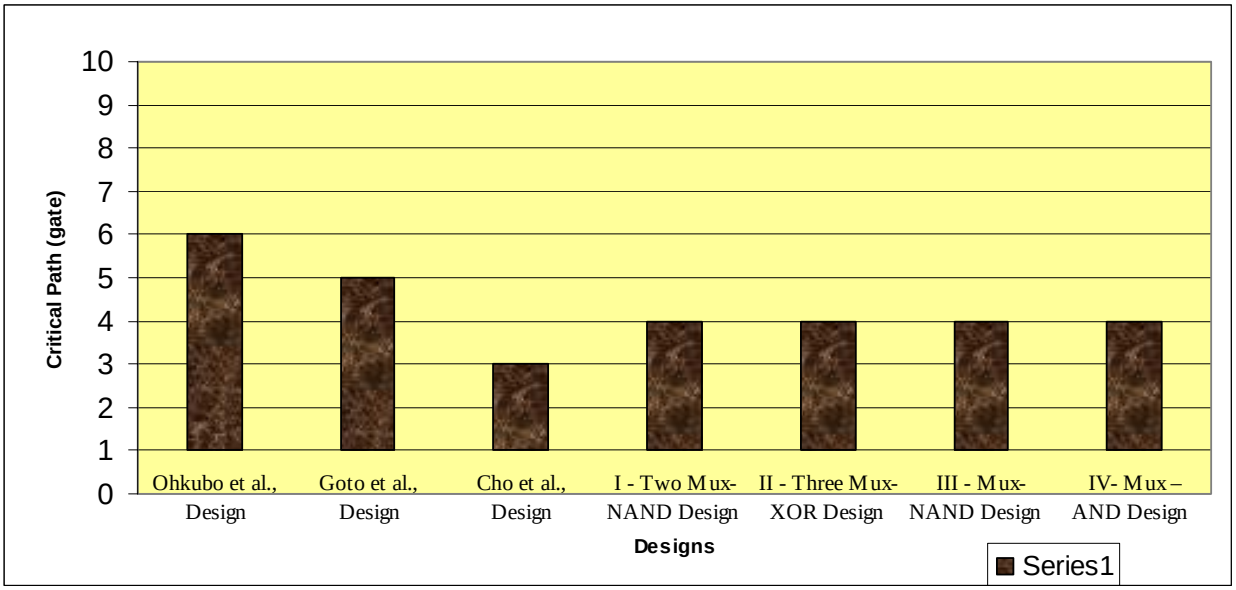

Figure 34. Comparison Chart for Delay

The chart shown in Figure 34 gives the comparison of proposed designs delay with the existing designs. It can be seen that the delay is uniform throughout the four proposed designs. The chart also shows reduction in gate delay by two and one units when compared with Ohkubo and Goto's designs.

CHAPTER 7. CONCLUSION

Multiplication is a frequently encountered operation, especially in signal processing applications. So the development of a multiplier is vital for applications in portable mobile devices such as personal multimedia players, cellular phones, digital cam coders and digital cameras. Many designs have been proposed for Booth encoder and selector logic using CMOS over the past decades. But those designs when implemented in CMOS resulted in higher transistor count. In our research, pass logic was found to be more efficient than CMOS logic. Booth encoder and selector logic occupies one third of the entire multiplier architecture. So careful optimization of these logic parts will result in a considerable reduction of hardware.

In this work, we proposed four new designs for Booth encoder and selector logic with less number of transistors than the published ones. The architecture was based on a modified Booth-encoding scheme, which reduced the number of partial-products by half compared to a traditional implementation. Using the pass logic based implementations; the number of transistors was reduced, resulting in hardware-reduced and consequently power-aware designs. The proposed Booth encoder and selector logic can be successfully used to build a 64 x 64 bit multiplier. Our new designs are fully scalable without the loss of merits.

The proposed designs saved up to 30 transistors when compared with the published Booth encoders. Similarly, for the selector logic 19 transistors were saved for one bit when matching with the existing designs. Critical path is uniform throughout the four proposed designs. The proposed designs gate delay was reduced by two and one units when compared with Ohkubo and Goto’s designs. The gate level implementations of these designs were tested for functionality using LoKon software. The pass logic implementation of all the gates (XNOR,

XOR, NAND, NOR, AND, XOR-XNOR combination gate) and MUX used in these circuits were simulated and verified for functionality using TopSPICE.

7.1 Future Work

The present work on the new multiplier architecture can be further extended in various directions.

- The design can be simulated to check the power consumption.
- Other methods can be incorporated with this to further improve the delay.
- In order to completely analyze the performance, the circuit can be extended to chip level where the delays due to wiring, interconnects and PAD are included.

CHAPTER 8. BIBLIOGRAPHY

1. Ki-seon Cho, Jong-on Park, Jin-seok Hong, Goang-seog Choi, “54x54-bit Radix-4 Multiplier based on Modified Booth Algorithm,” *ACM*, *Proceedings of the 13th ACM Great Lakes symposium on VLSI*, pp. 233-236, April 2003
2. N. Ohkubo, *et. al.*, "A 4.4ns CMOS 54x54-b Multiplier Using Pass-Transistor Multiplexer", *IEEE J. of Solid-State Circuits*, vol. 30, no. 3, pp. 251-257, Mar., 1995
3. G. Goto, *et. al.*, "A 4.1-ns Compact 54×54-b Multiplier Utilizing Sign-Select Booth Encoders", *IEEE J. of Solid-State Circuits*, vol. 32, no. 11, pp. 1676-1681, Nov. 1997
4. *Computer Arithmetic Algorithms* by Israel Koren, 2nd Edition, A K Peters, Natick, Massachusetts, 2002
5. Rafael Fried, “Minimizing Energy Dissipation in High-Speed Multipliers”, *ACM*, *International Symposium on Low Power Electronics and Design*, pp. 214-219, 1997
6. Johann Groβschadl, “A Unified radix-4 Partial Product Generator For Integers And Binary Polynomials”, *IEEE Symposium on Circuits and Systems*, vol. 3, pp. 567-570, May 2002
7. Damu RadhaKrishnan, “A New low Power CMOS Full Adder”, *IEE Electronics Letters* vol.35, No. 21, pp. 1792-1794, October 1999
8. D. RadhaKrishnan and A. P. Preethy, “Low Power CMOS Pass Logic 4-2 Compressor for High Speed Multiplication”, *IEEE Midwest Symposium on Circuits and Systems*, vol. 3, pp. 1296-1298, August 2000

APPENDIX

The proposed designs are constructed and functionally simulated in gate level using the software LoKon V2.4. For all proposed designs two snap shots for selected operations (*X*, *-X*, *2X*, *-2X*, 0) are shown. The red line indicates logic one and black line indicates logic zero.

Two MUX- NAND Design:

Output PP_n: *X*

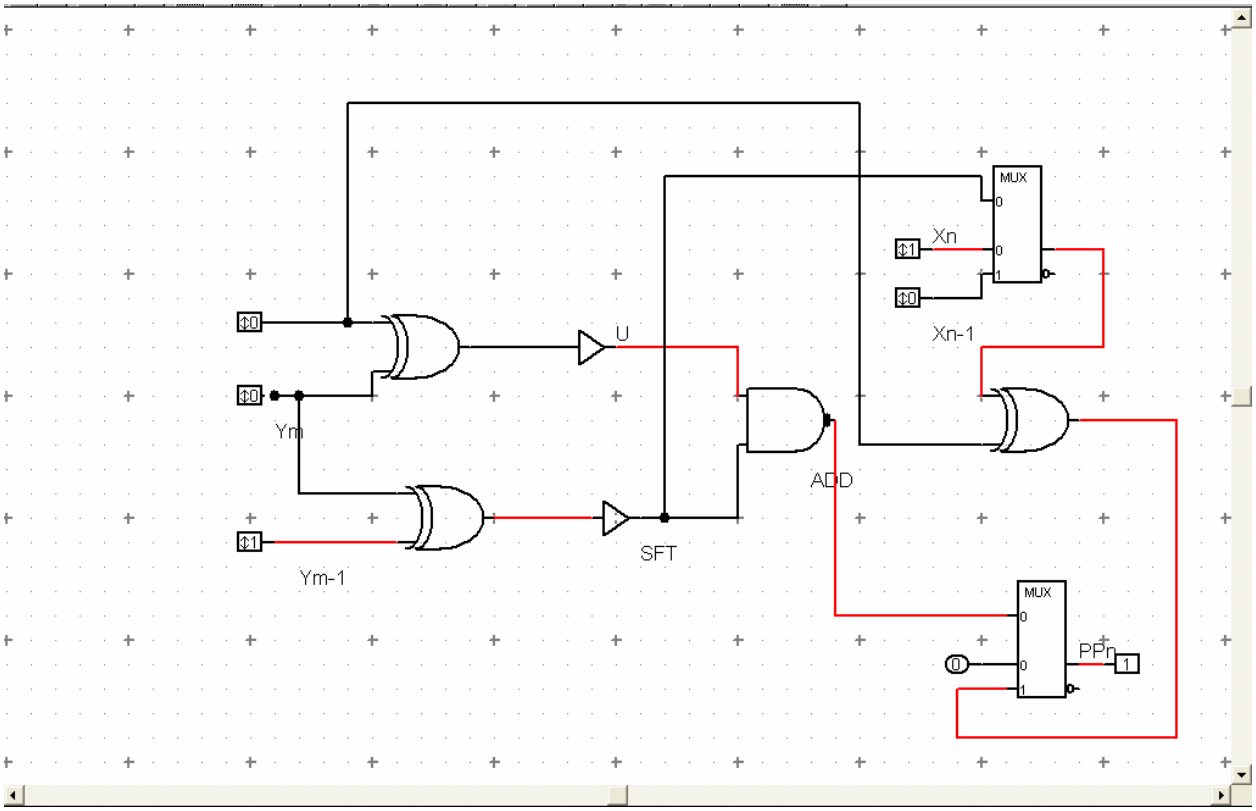

Figure A-1. Snap Shot Showing *X* operation

In Figure A-1 inputs Y_{m+1} = 0, Y_m = 0 and Y_{m-1} = 1. The inputs X_n and X_{n-1} are given one and zero respectively. SFT becomes 0 and it selects the data input X_n as one from the MUX on the top. Since Neg=0 , X_n is passed as one through the XOR gate uncomplemented. Since *ADD*

signal at this instant is one, the MUX at the bottom outputs PP_n as X_n in logic one state (Refer to Table 28).

Output PP_n: -*X*

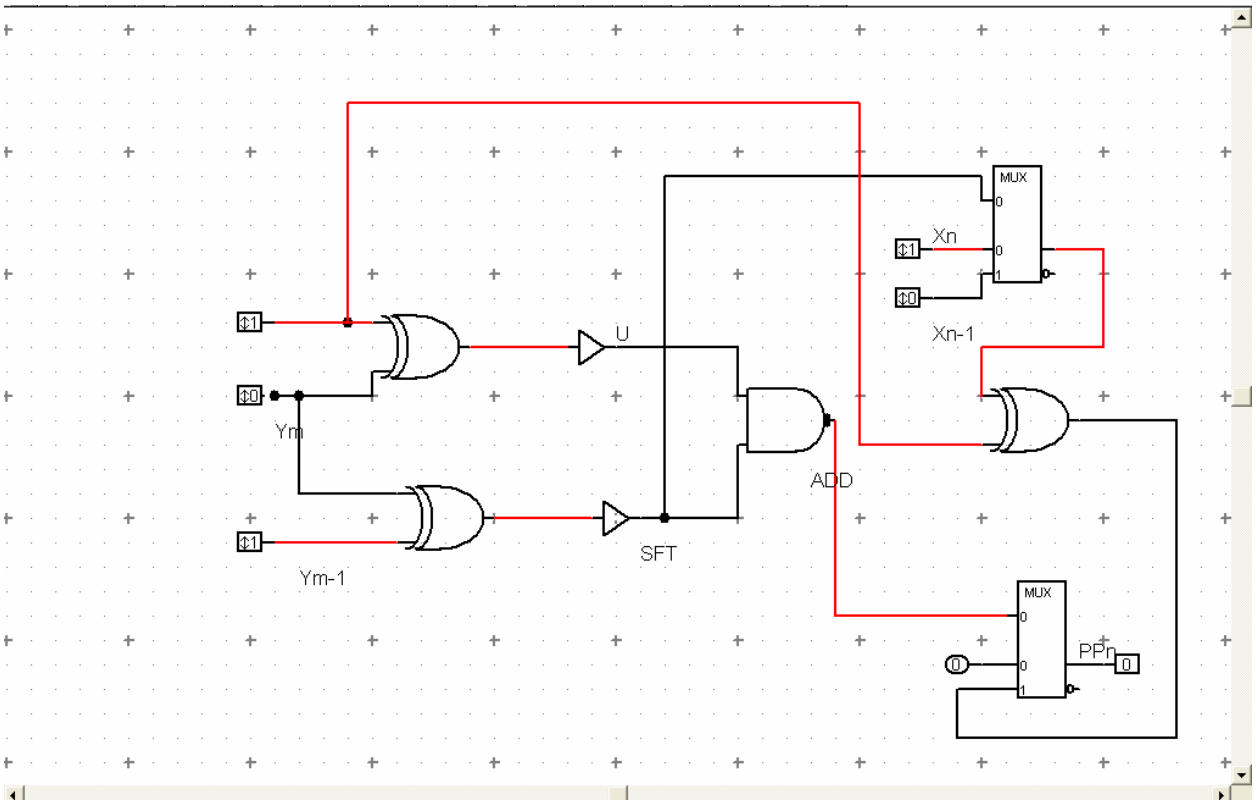

Figure A-2. Snap Shot Showing -*X* operation

The inputs Y_{m+1} = 1, Y_m = 0 and Y_{m-1} = 1 are shown in Figure A-2. The inputs X_n and X_{n-1} are given one and zero respectively. SFT becomes 1 and it selects the data input X_n as one from the MUX on the top. Since Neg= 1, X_n is complemented and passed as zero through the XOR gate. Since *ADD* signal at this instant is one, the MUX at the bottom outputs PP_n as X_n in logic zero state (Refer to Table 28).

Three MUX- XOR Design:

Output PP_n: *2X*

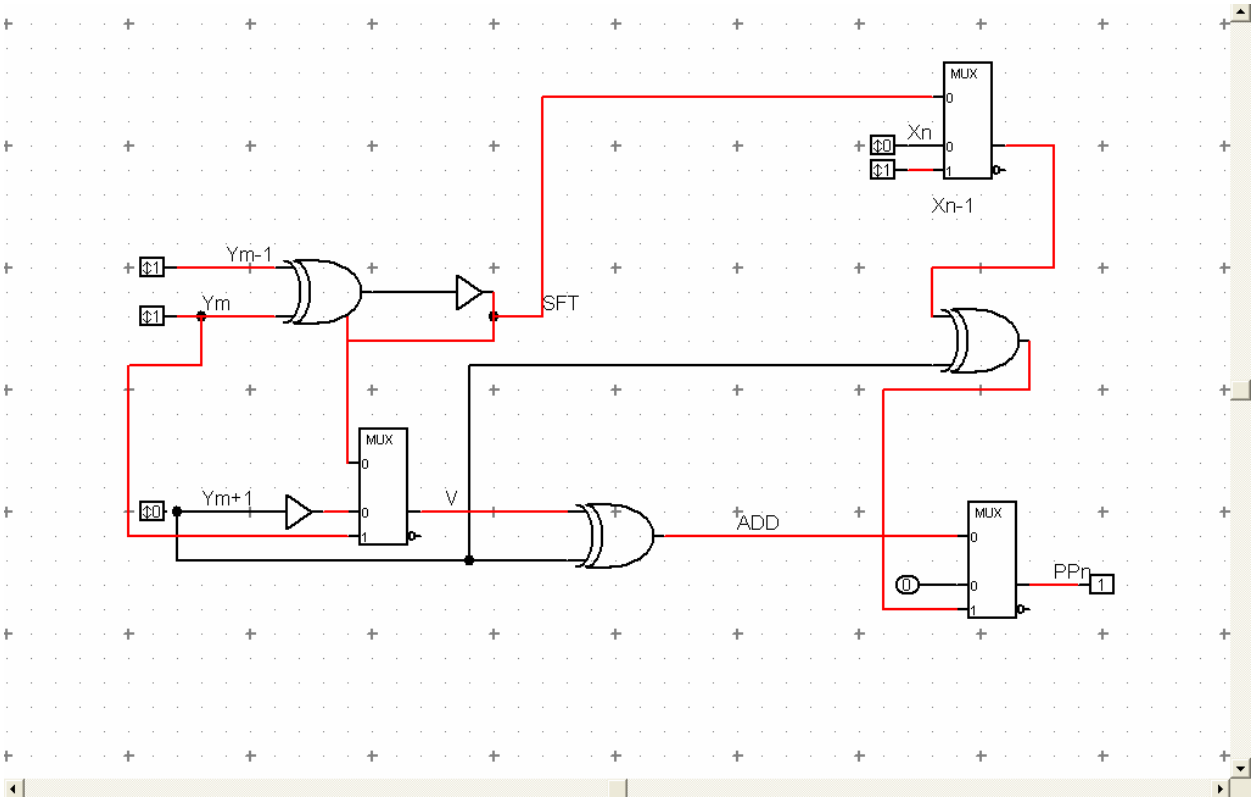

Figure A-3. Snap Shot Showing 2*X* operation

The inputs Y_{m+1} = 0, Y_m = 1and Y_{m-1} = 1 are shown in Figure A-3. The inputs X_n and X_{n-1} are given zero and one respectively. SFT becomes 1 and it selects the data input X_{n-1} as one from the MUX on the top. Since Neg= 0, X_{n-1} is uncomplemented and it passes as one through the XOR gate. Since *ADD* signal at this instant is one, the MUX at the bottom outputs PP_n as X_{n-1} in logic one state (Refer to Table 29).

Output PP_n: *-2X*

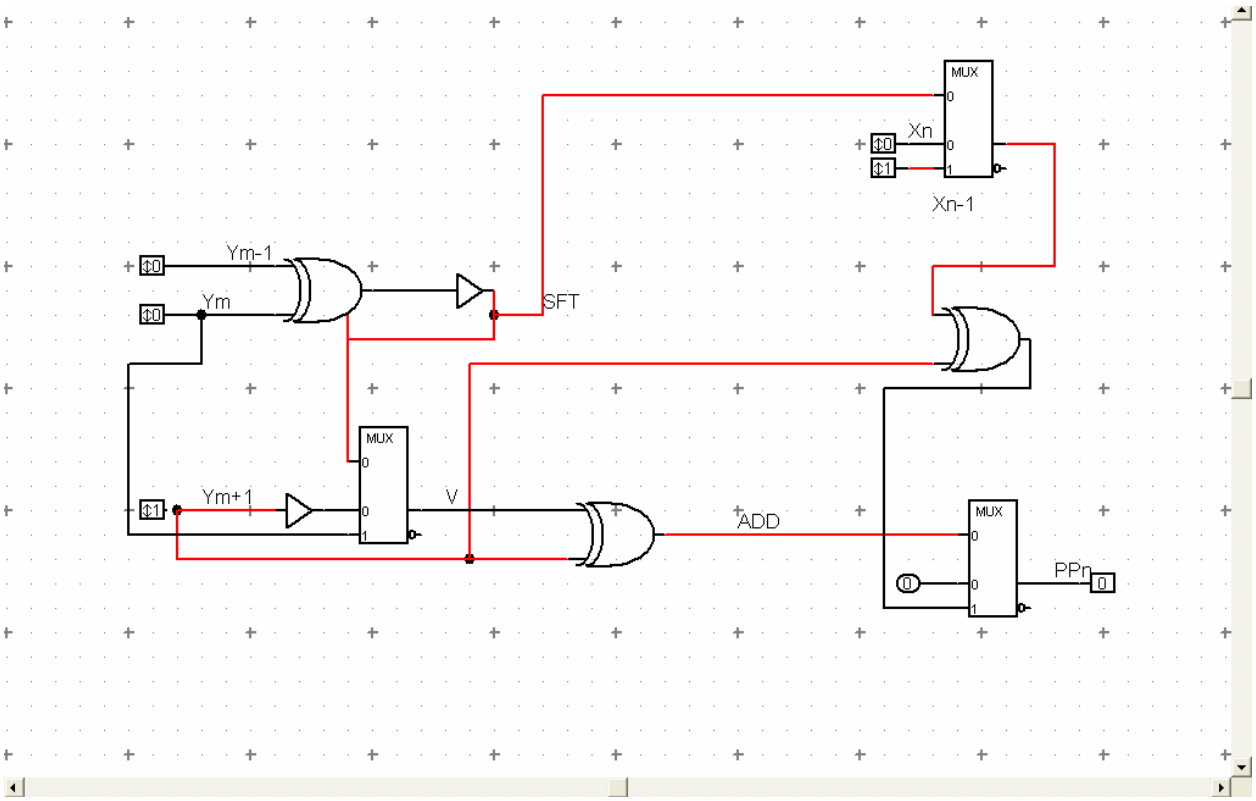

Figure A-4. Snap Shot Showing -2*X* operation

The inputs $Y_{m+1} = 1$, $Y_m = 0$ and $Y_{m-1} = 0$ are shown in the Figure A-4. The inputs X_n and X_{n-1} are given zero and one respectively. SFT becomes 1 and it selects the data input X_{n-1} as one from the MUX on the top. Since Neg= 1, X_{n-1} is complemented and passed as zero through the XOR gate. Since *ADD* signal at this instant is one, the MUX at the bottom outputs PP_n as X_{n-1} in logic zero state (Refer to Table 29).

MUX -AND Design

Output PP_n: Zero

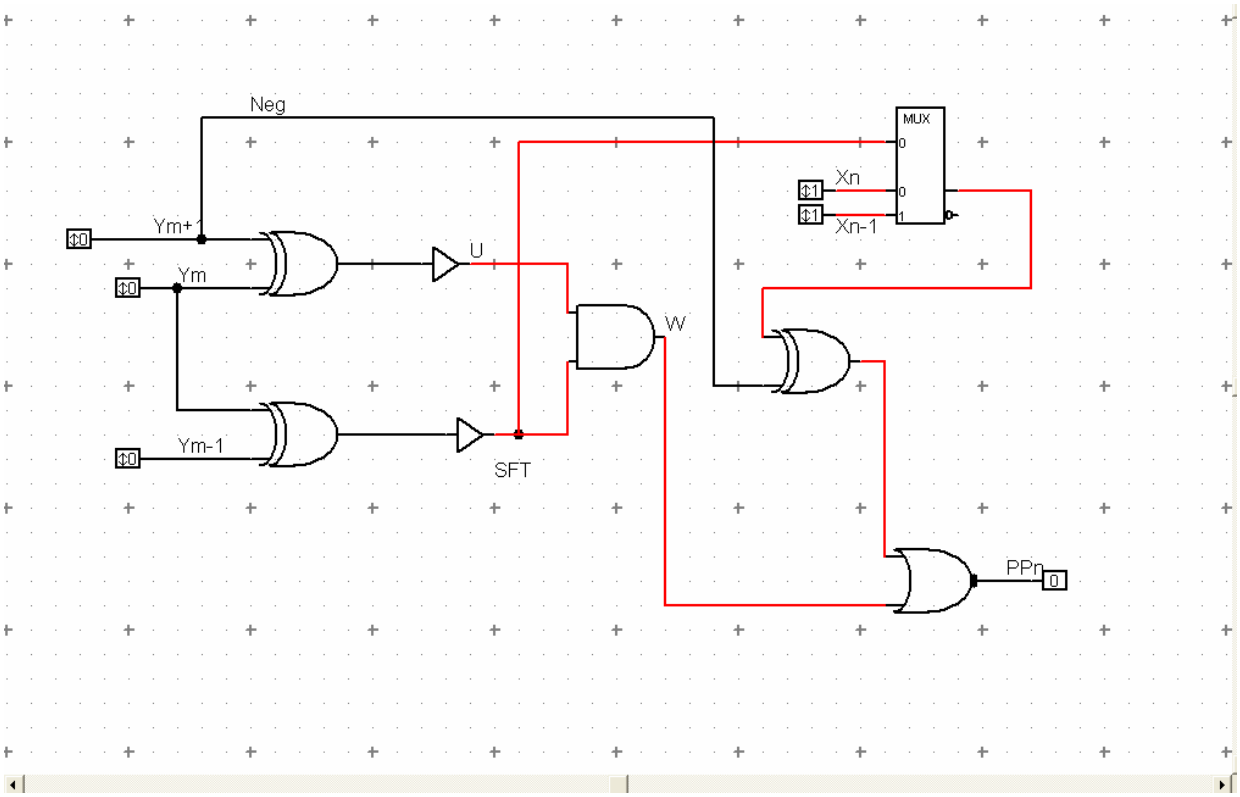

Figure A-5. Snap Shot Showing *Zero* operation

The inputs $Y_{m+1} = 0$, $Y_m = 0$ and $Y_{m-1} = 0$ are shown in Figure A-5. The inputs X_n and X_{n-1} are given one and one respectively. SFT becomes 1 and it selects the data input X_{n-1} as one from the MUX on the top. Since Neg= 0, X_{n-1} is uncomplemented and it passes as one through the XOR gate. Since *ADD* signal at this instant is zero, the MUX at the bottom outputs logic zero as PP_n (Refer to Table 30).

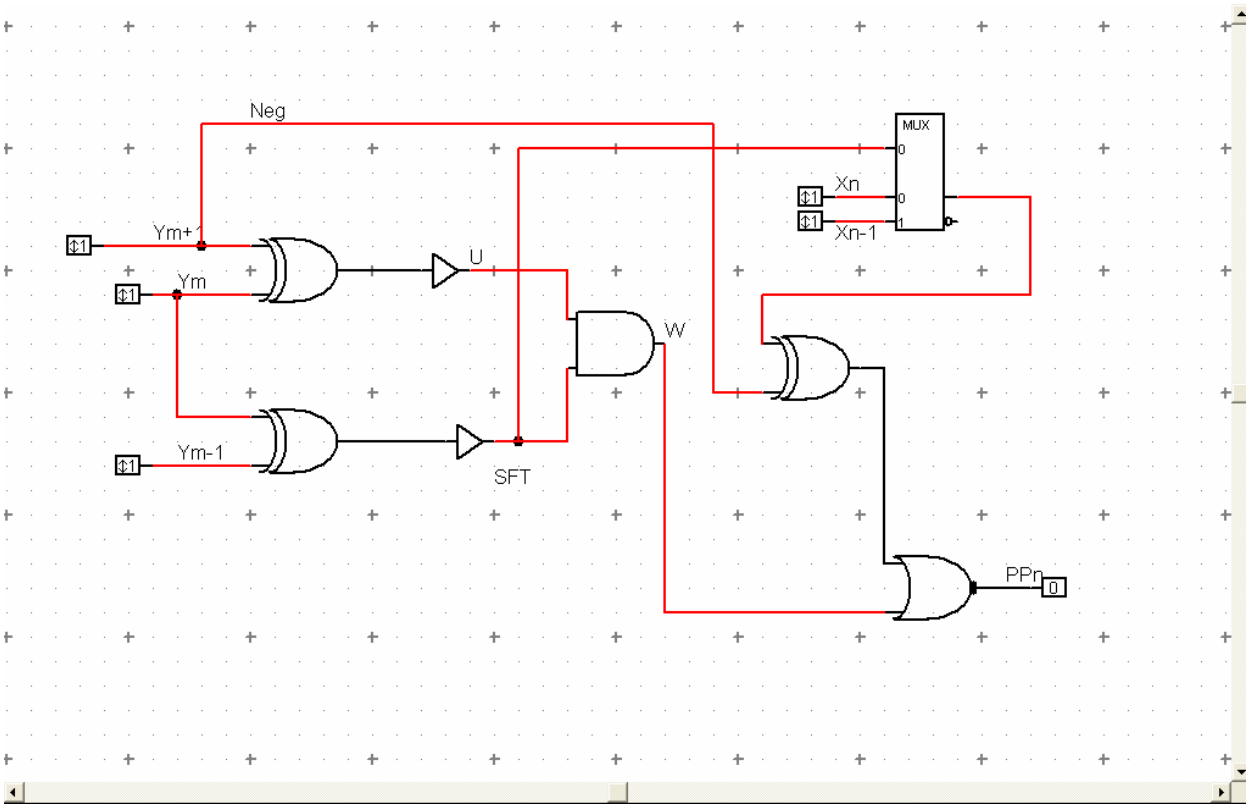

Figure A-6. Snap Shot Showing *Zero* operation

The inputs Y_{m+1} = 1, Y_m = 1 and Y_{m-1} = 1 are shown in Figure A-6. The inputs X_n and X_{n-1} are given one and one respectively. SFT becomes 1 and it selects the data input X_{n-1} as one from the MUX on the top. Since Neg= 1, X_{n-1} is complemented and it passes as zero through the XOR gate. Since *ADD* signal at this instant is zero, the MUX at the bottom outputs logic zero as PP_n (Refer to Table 30).

Due to the unavailability of transistors in LoKon software, the final MUX- NAND design was not able to simulate.

Pass Logic Implementation of the components using TopSPICE

The various components in the proposed designs were implemented in pass logic and were simulated using TopSPICE software. Due to the number of transistor limit in the demo version of the software, the entire circuit was not able to simulate. The snap shots of the various components simulated using software and its graphical output is shown.

AND Gate

In MUX-AND design we use AND gate to selectively output the data inputs. So AND gate implemented in pass logic was simulated in TopSPICE. The technology used was 0.25µ and the voltage was 3V. The pass logic equation for AND gate is given below.

$Y = A' (0) + A (B)$

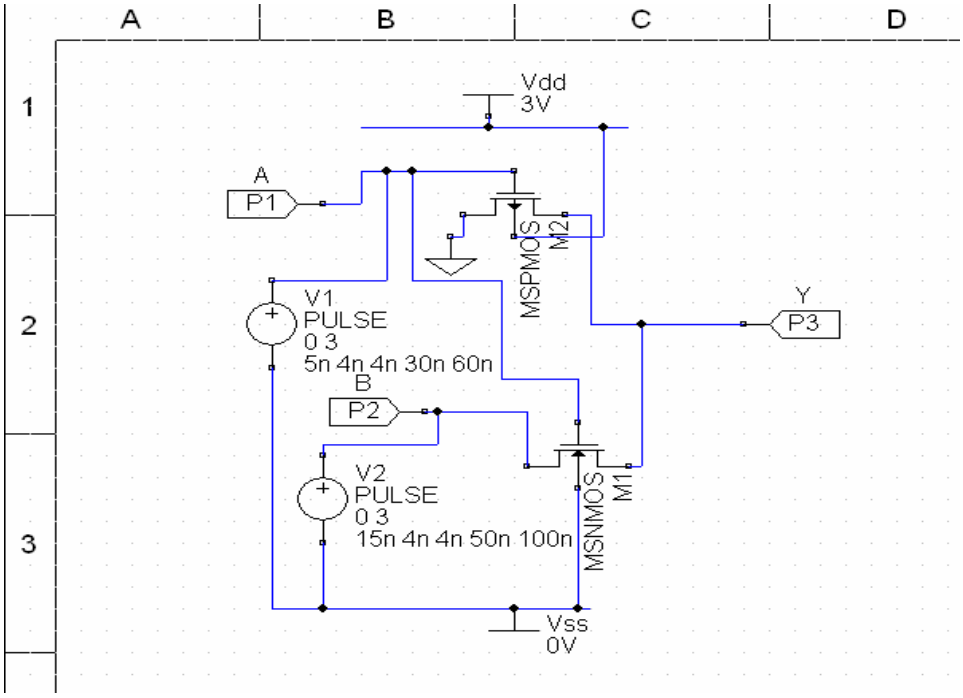

Figure A-7. Pass Logic Implementation of AND Gate

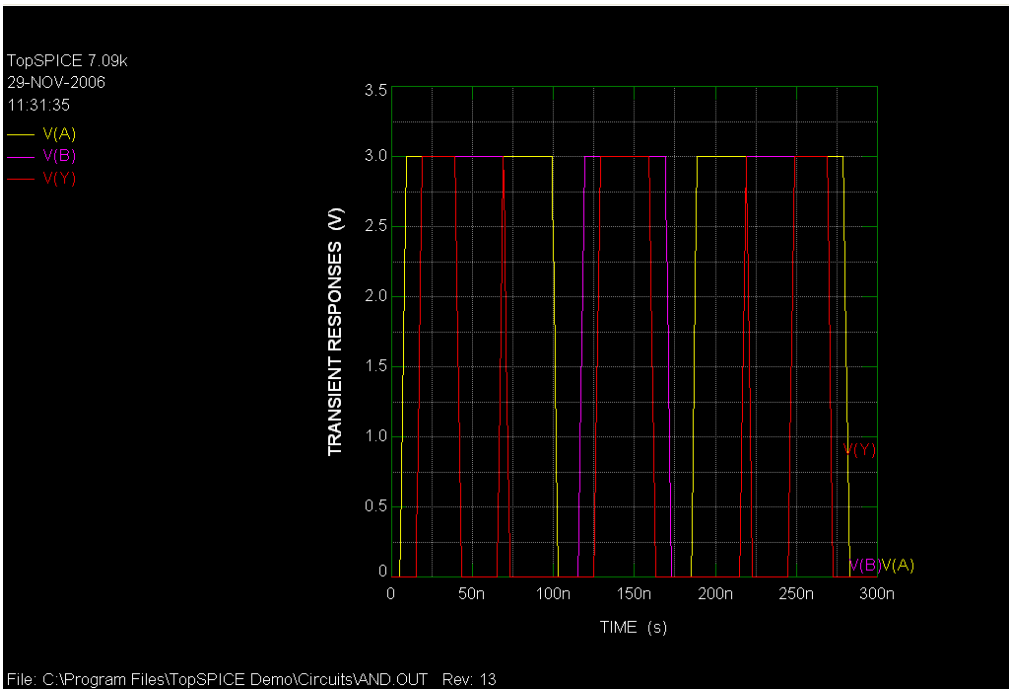

Figure A-8. Waveform of AND Gate from TopSPICE

NAND Gate

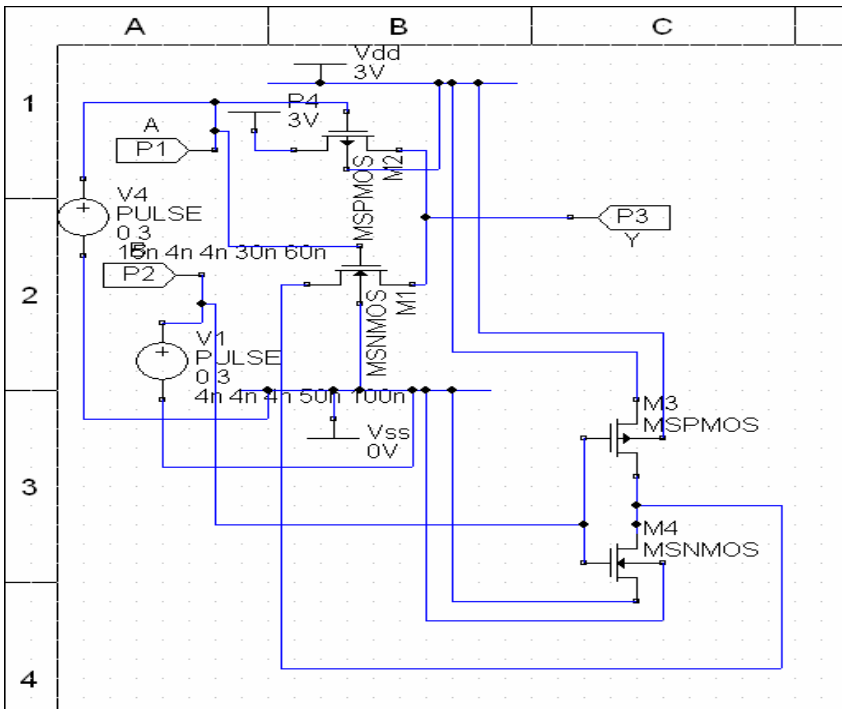

Figure A -9. Pass Logic Implementation of NAND Gate

In Two MUX-NAND and MUX- AND designs, we used NAND gate to selectively output the data inputs as the PP_n. The NAND gate simulated in TopSPICE is shown in Figure A -9. The technology used was 0.25µ and the voltage was 3V. The pass logic equation for NAND gate is given below.

Y = A' (1) + A (B')

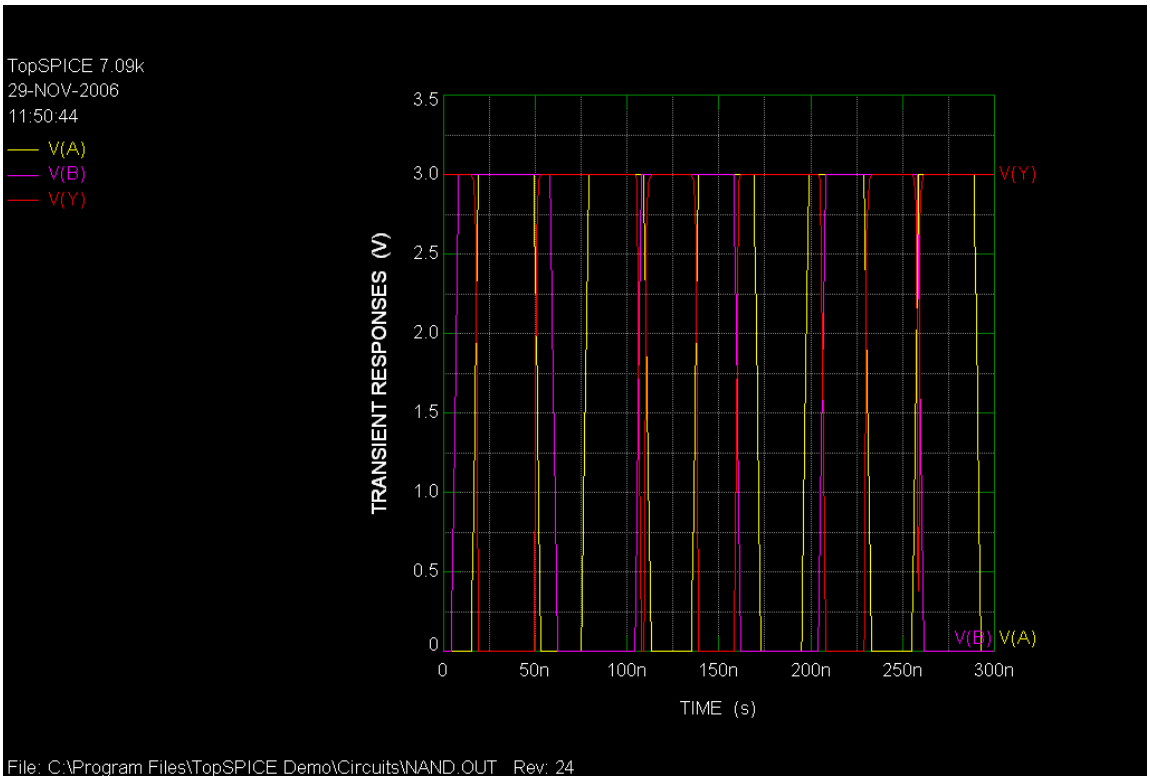

Figure A-10. Waveform of NAND gate from TopSPICE

NOR Gate

In MUX-AND design NOR gate was used to selectively output PP_n. So NOR gate implemented in pass logic was simulated in TopSPICE. The technology used was 0.25µ and the voltage was 3V. The pass logic equation for NOR gate is given below.

Y = A' (B') + A (0)

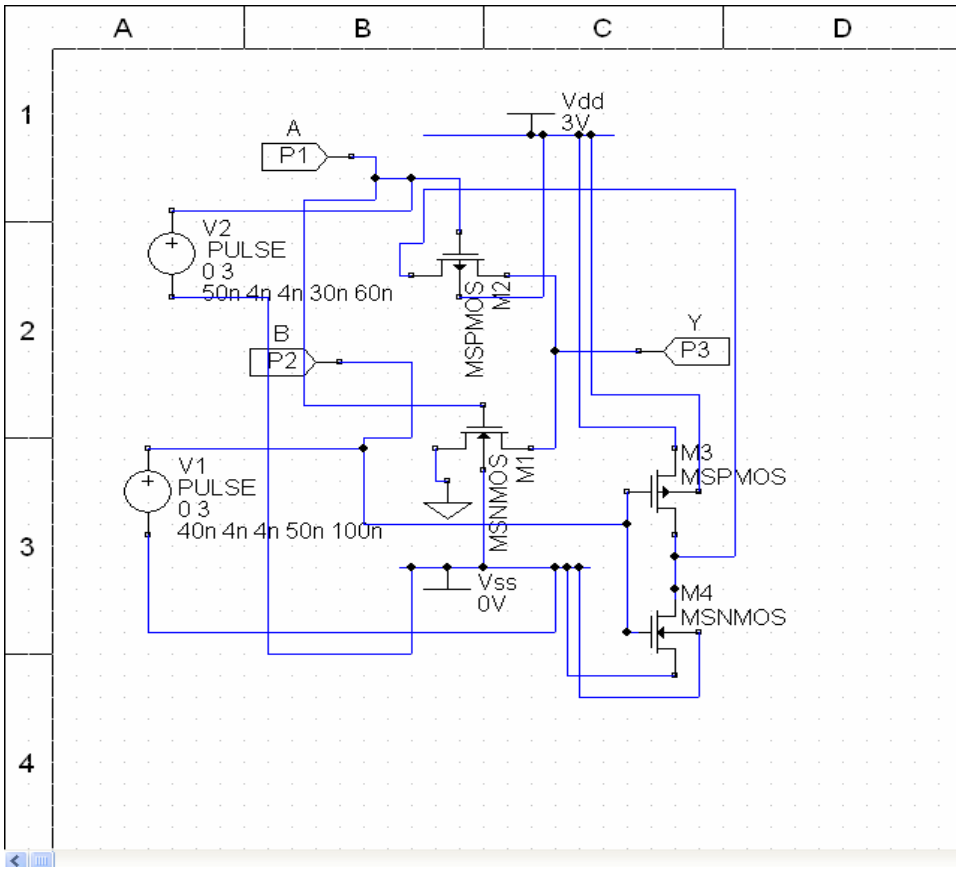

Figure A-11. Pass Logic Implementation of NOR Gate

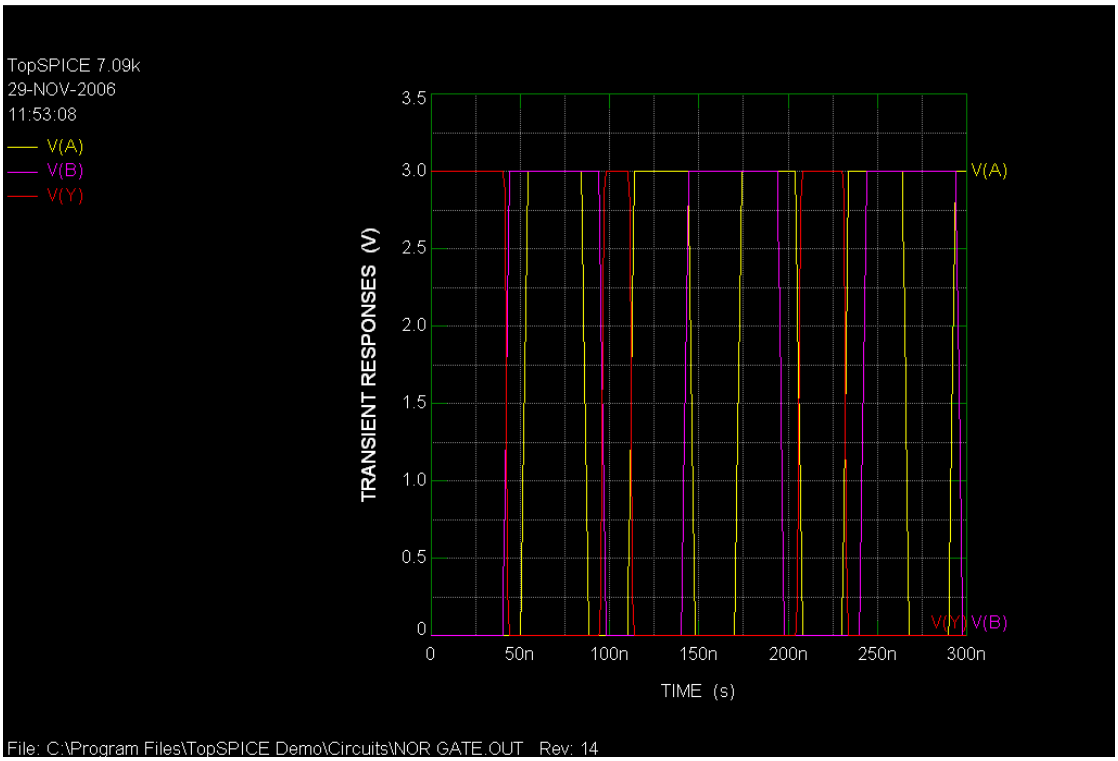

Figure A-12. Waveform of NOR gate from TopSPICE

XOR-XNOR Combination Gate

XOR-XNOR combination gate was the main component used in the designs to reduce the number of transistors. In this circuit, without using the transmission gate the fully restored output was obtained by the regenerative feedback circuit. In all the four proposed designs, this circuit was used. This was simulated and the graph with separate plots for XOR and XNOR gates are shown below. The technology used is 0.25 μ and the voltage supply was 3V. The pass logic expression for XOR-XNOR combination gate is given below.

XOR = A' (B) + B' (A) + AB (0)

XNOR = A (B) + B (A) +A' B' (1)

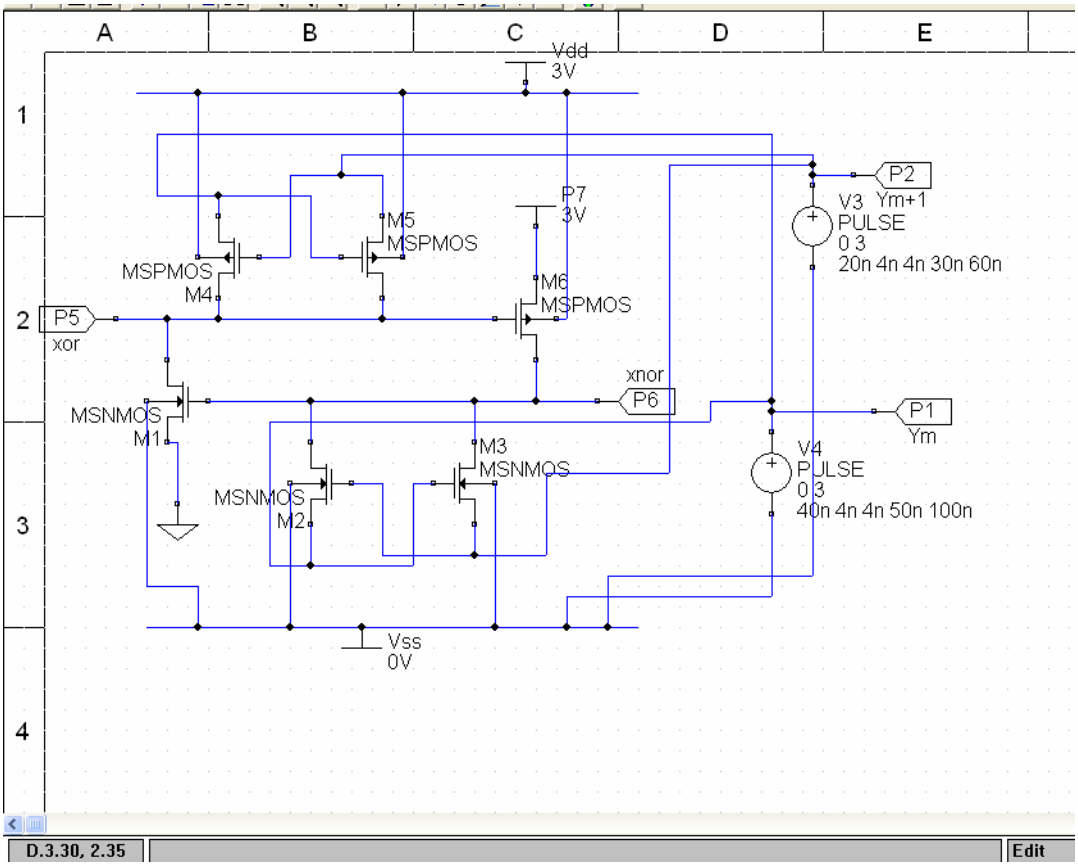

Figure A-13. Pass Logic Implementation of XOR-XNOR Combination Gate

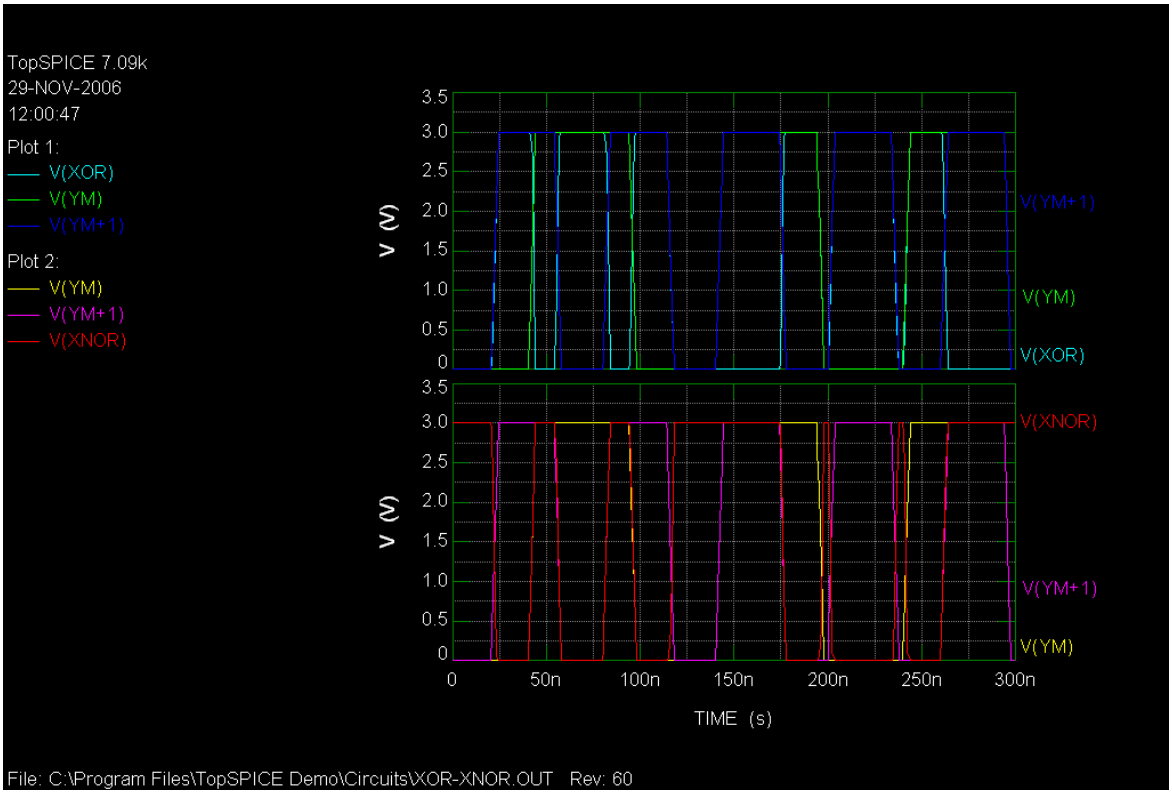

Figure A-14. Waveform of XOR-XNOR Combination gate from TopSPICE

MUX Gate

In all the four proposed designs Mux was used to used to selectively pass zero or (±X, ±2X). In order to obtain a fully restored output, transmission gate was used. Transmission gate was implemented by parallel connection of nMOS and pMOS transistor. The technology used is 0.25 μ and the voltage supply was 3V. The pass logic expression for XOR-XNOR combination gate is given below.

Y = S' (A) +S (B)

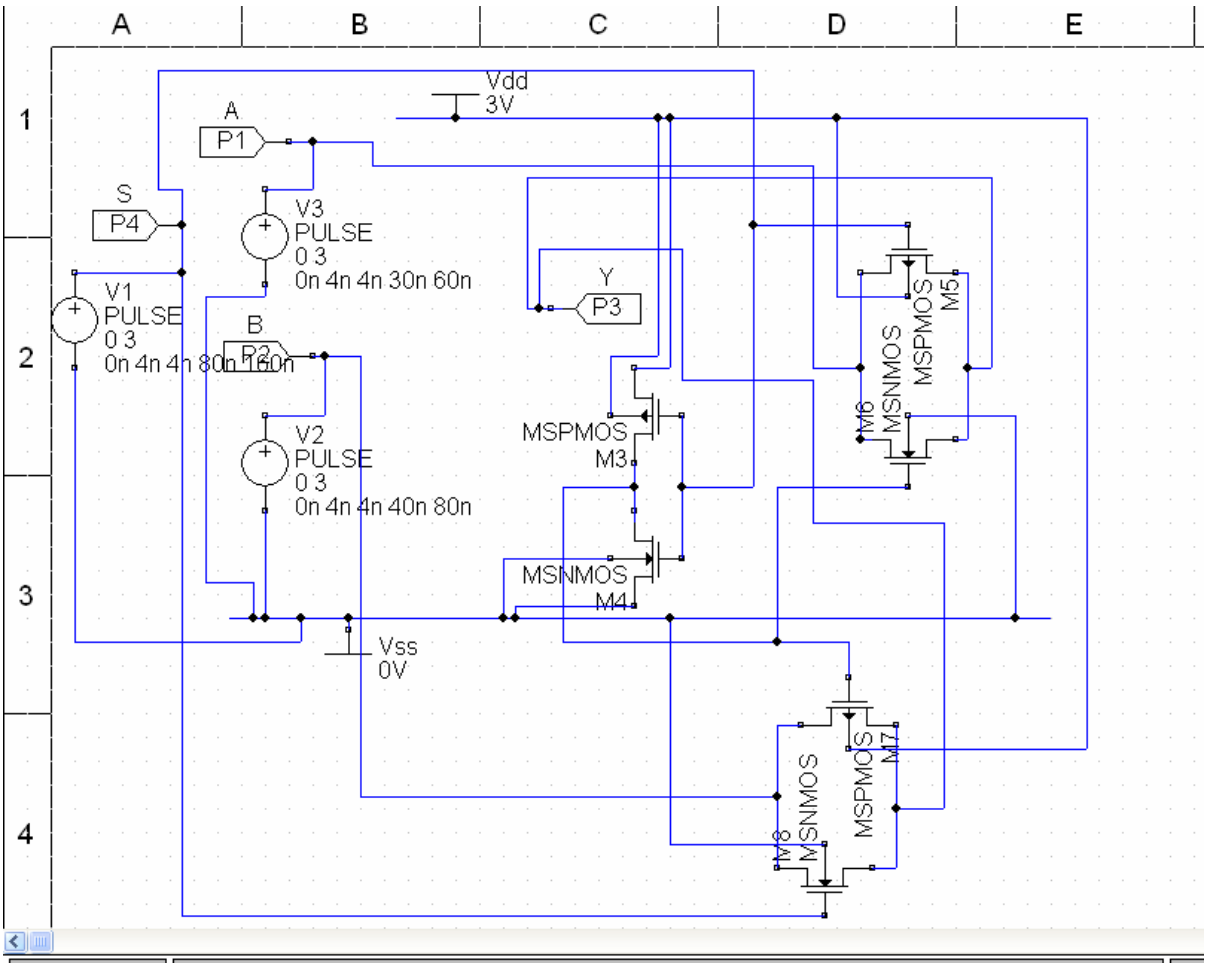

Figure A-15. Pass Logic Implementation of MUX

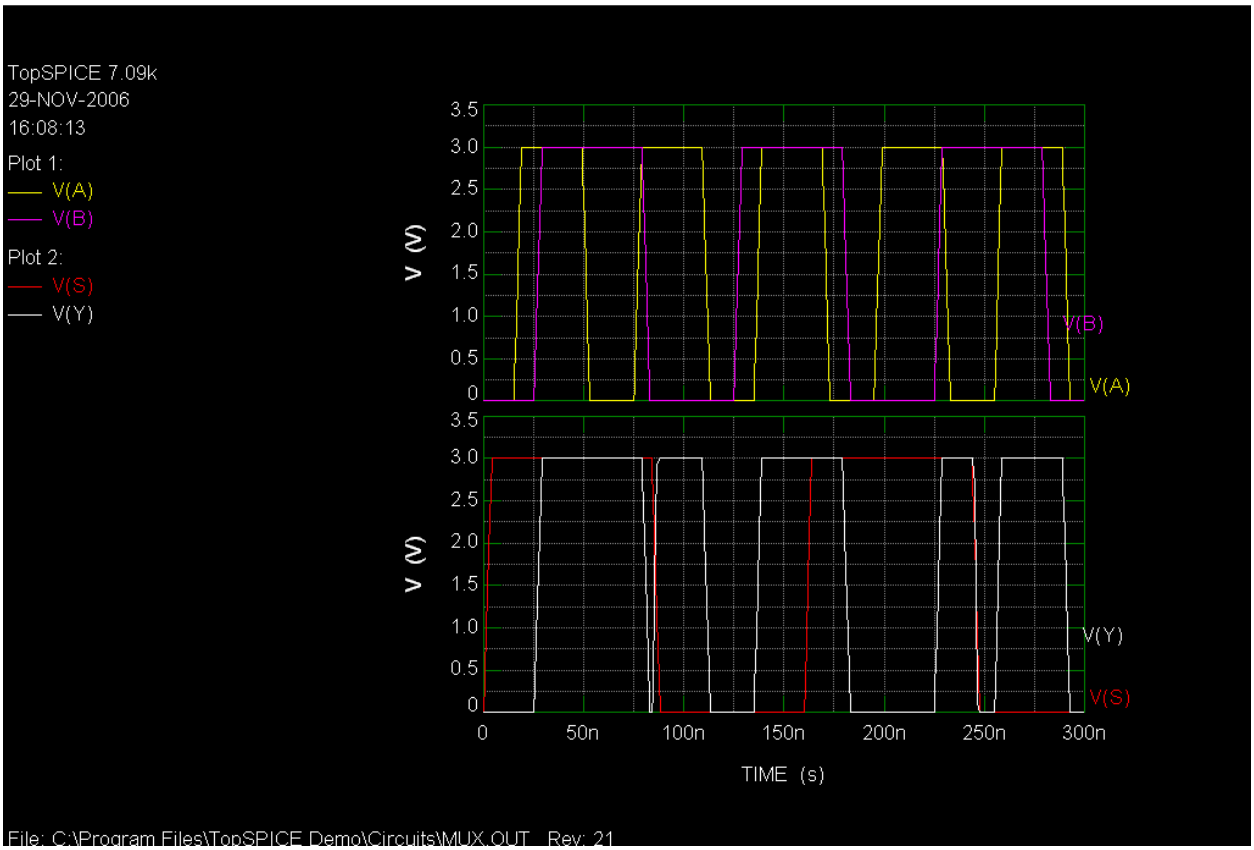

Figure A-16. Waveform of MUX from TopSPICE