POWER EFFICIENT MULTIPLIER ARCHITECTURE USING SPST TECHNIQUE

A PROJECT REPORT

Submitted by

RADHA.P	**Reg. No.:1110107069**
SINDHUJA.N	**Reg. No.:1110107095**
SOUNDARYA.M	**Reg. No.:1110107099**
AARTHI.P	**Reg. No.:1110107301**

In partial fulfilment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION

ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2015

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report titled **"POWER EFFICIENT MULTIPLIER ARCHITECTURE USING SPST TECHNIQUE"** is the bonafide work of "RADHA P, SINDHUJA N, SOUNDARYA M, AARTHI P**"** who carried out the project work under my supervision.

SIGNATURE

Ms.A.Kalaiselvi,
Assistant Professor/ECE
Kumaraguru College of Technology
Coimbatore.

SIGNATURE

Dr.Rajeswari Mariappan M.E., Ph.D.,
HEAD OF THE DEPARTMENT
Electronics & Communication Engineering
Kumaraguru College of Technology
Coimbatore.

The candidates with Register numbers 1110107069, 1110107095, 1110107099, 1110107301 are examined by us in the project viva-voce examination held on …….

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First we would like to express our praise and gratitude to the Lord, who has showered his grace and blessing enabling us to complete this project in an excellent manner. He has made all things in beautiful in his time.

We express our sincere thanks to our beloved Joint Correspondent, **Shri. Shankar Vanavarayar** for his kind support and for providing necessary facilities to carry out the project work.

We would like to express our sincere thanks to our beloved Principal **Dr. R. S. Kumar M.E.,Ph.D.,** who encouraged us with his valuable thoughts.

We would like to express our sincere thanks and deep sense of gratitude to our HOD, **Dr. Rajeswari Mariappan M.E., Ph. D.,** for her valuable suggestions and encouragement which paved way for the successful completion of the project.

We wish to thank and express our everlasting gratitude to the Supervisor and project coordinator **Ms. A. Kalaiselvi M. E. , (Ph. D)**, for her expert counselling in each and every steps of project work and we wish to convey our deep sense of gratitude to all teaching and non-teaching staff members of ECE Department for their help and cooperation.

Finally, we thank our parents and our family members for giving us the moral support in all of our activities and our dear friends who helped us to endure our difficult times with their unfailing support and warm wishes.

ABSTRACT

Today every circuit has to face the power consumption issue for both portable device aiming at large battery life and high end circuits avoiding cooling packages and reliability issues that are too complex. It is generally accepted that larger design will generally consume more power. In this project, we proposed a new architecture of multiplier which consumes low power. Multiplication occurs frequently in signal processing applications such as Finite Impulse Response Filters, Fast Fourier Transforms, Discrete Cosine Transforms, convolution, etc., Micro Processor and multimedia kernels. The objective of a good multiplier is to provide a physically compact, good speed and low power consuming chip. To save significant power consumption of a VLSI design, it is a good direction to reduce its dynamic power that is the major part of total power dissipation. For getting the low power the modifications made to the conventional architecture consist of the reduction in switching activities of the major blocks of the multiplier, which includes the reduction in switching activity of the adder.

Here, we proposed a low power multiplier by implementing the new SPST technique. To filter out the unwanted switching power, there are two approaches, i.e using registers and using AND gates, to assert the data signals of multipliers after data transition. This multiplier is designed by equipping the Spurious Power Suppression Technique (SPST) on a modified Booth encoder which is controlled by a detection unit using an AND gate. The simulation result shows that the SPST implementation with AND gates owns an extremely high flexibility on adjusting the data asserting time which not only facilitates the robustness of SPST but also leads to power reduction.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuits
CLA	Configurable Logic Array
CLB	Configurable Logic Block
CPLD	Configure Programmable Logic Device
DCT	Discrete Cosine Transform
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IOB	Input Output Block
Ios	Input Output
LSP	Least Significant Bit
LUT	Look up Table
MAC	Multiplier and Accumulator
MR	Multiplicand Register
MSP	Most Significant Bit
PCL	Programmable Control Logic
PPG	Partial Product Generator
PR	Product Register
RTL	Register Transfer Level
SPST	Spurious Power Suppressed Technique
TBW	Test Bench waveform
VHDL	Verilog Hardware Discription Language
VLSI	Very Large Scale Integration

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand; the number of times that it is added is the multiplier; and the result is the product. i.e The basic multiplication principle is two fold:

1. Evaluation of partial products and
2. Accumulation of the shifted partial products

In most computers, the operand usually contains the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of operands in order to preserve the information content.

For generating partial products, the booth algorithm is often used because it is less complex but for ‘n’ bit multiplier it produces ‘n’ partial products. Whereas by using the Radix_4 modified Booth algorithm number of partial products will be reduced by a factor of 2. I.e. For n bit multiplier we have only n/2 partial products. Thus dynamic power will be reduced. In order to avoid spurious addition, SPST adder is designed and used. The SPST uses a detection logic circuit to detect the effective data range of arithmetic units, namely adders or multipliers. When a portion of data does not affect the final computing results, the data controlling circuits of the SPST latch this portion to avoid useless data transitions occurring inside the arithmetic units.

The data are separated into the Most Significant Part (MSP) and the Least Significant Part (LSP). To know whether the MSP affects the computation results or not, the detection logic unit is used to detect the

effective ranges of the inputs. I.e. When the MSP is either zeros or ones then SPST adder is implemented to avoid switching power consumption otherwise normal adder is used. And for LSP, normal full adder is used for all cases. And the final product will be the combined output of MSP's and LSP's.

The dynamic power consumption for this Radix-4 modified booth multiplier using SPST is generated by X power analyzer in Xilinx 10.1 ISE (integrated software environment).The general flow of the project is expressed by the schematic below.

1.2 FLOW DIAGRAM

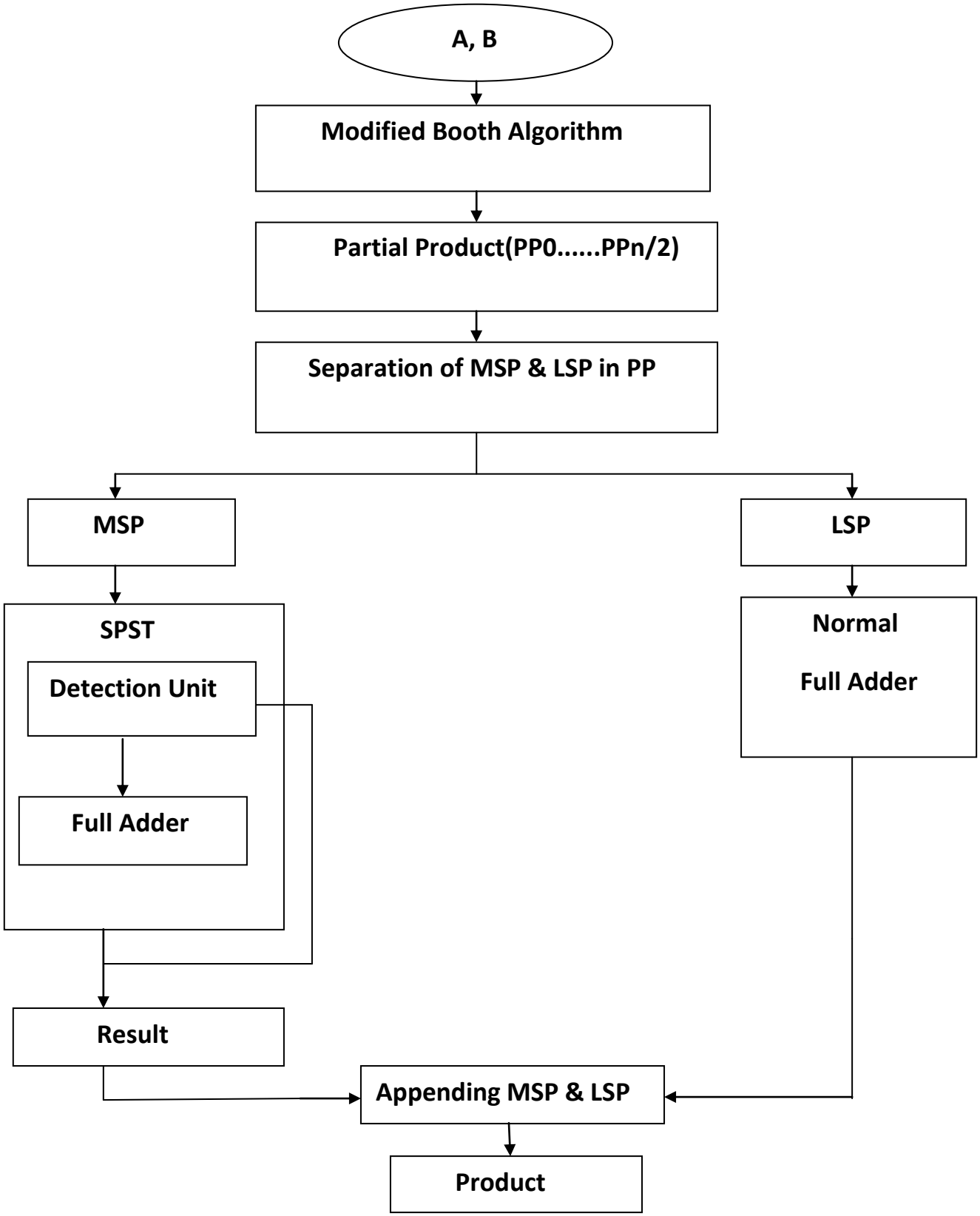


Fig.1.1 general flow of program

CHAPTER 2

MULTIPLIER

2.1 INTRODUCTION

Multiplication is an important part of real-time digital signal processing (DSP) applications ranging from digital filtering to image processing. Lowering down the power consumption and enhancing the processing performance of the circuit designs are undoubtedly the two important design challenges of wireless multimedia and DSP applications, in which multiplications are frequently used for key computations, such as FFT, DCT, quantization, and filtering. All multiplication methods share the same basic procedure - addition of a number of partial products. A number of different methods can be used to add the partial products. The simple methods are easy to implement, but the more complex methods are needed to obtain the fastest possible speed. The simplest method of adding a series of partial product is based upon an adder-accumulator, along with a partial product generator and a hard wired shifter. This is relatively slow, because adding N partial products requires N clock cycles. The easiest clocking scheme is to make use of the system clock, if the multiplier is embedded in a larger system. The system clock is normally much slower than the maximum speed at which the simple iterative multiplier can be clocked, so if the delay is to be minimized an expensive and tricky clock multiplier is needed, or the hardware must be self-clocking.

2.2 MULTIPLIER

Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand; the number of times that it is added is the multiplier; and the result is the product. Each step of addition generates a partial product. In most computers, the operand usually contains the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of operands in order to preserve the information content. This repeated addition method that is suggested by the arithmetic definition is slow that it is almost always replaced by an algorithm that makes use of positional representation. It is possible to decompose multipliers into two parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them. As for adders, it is possible to enhance the intrinsic performance of multipliers. Acting in the generation part, the booth algorithm is often used because it reduces the number of partial products.

Multipliers play an important role in today's digital signal processing and various other applications. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed, low power and compact VLSI implementation. The objective of a good multiplier is to provide a physically compact, good speed and low power consuming chip.

The basic multiplication principle is two fold:

1. Evaluation of partial products and
2. Accumulation of the shifted partial products

It is performed by the successive additions of the columns of the shifted partial product matrix. The 'multiplier' is successfully shifted and gates the

appropriate bit of the 'multiplicand'. The delayed, gated instance of the multiplicand must all be in the same column of the shifted partial product matrix. They are then added to form the product bit for the particular form. Multiplication is therefore a multi operand operation. To extend the multiplication to both signed and unsigned numbers, a convenient number system would be the representation of numbers in two's complement format. Some of the advantages of using two's complement number system include:

1. Representation of negative numbers in two's complement allowing for an easy subtraction function, since $-M = \overline{M} + 1$
2. Sufficient redundancy allowing for the annihilation of carry or borrows chains and also, in fact, propagation free addition and subtraction.
3. Booth's recoding technique, used for signed binary numbers enabling faster multiplications.

The common multiplication method is "add and shift" algorithm. In parallel multipliers number of partial products to be added is the main parameter that determines the performance of the multiplier. To reduce the number of partial products to be added, Modified Booth algorithm is one of the most popular algorithms. To achieve speed improvements Wallace Tree algorithm can be used to reduce the number of sequential adding stages. Further by combining both Modified Booth algorithm and Wallace Tree technique we can see advantage of both algorithms in one multiplier. However with increasing parallelism, the amount of shifts between the partial products and intermediate sums to be added will increase which may result in reduced speed, increase in silicon area due to irregularity of structure and also increased power consumption due to increase in interconnect resulting from complex routing. On the other hand "serial-parallel" multipliers compromise speed to achieve better performance for area and power consumption. The selection of a parallel or serial multiplier actually depends on the nature of application. In this lecture we introduce the multiplication

algorithms and architecture and compare them in terms of speed, area, power and combination of these metrics.

The multiplier plays a major role in DSP application. The present development in processor design aim at low power multiplier architecture using in their processor circuit. So the need for low power multiplier has increased, hence the designer concentrate more on low power efficient circuit design. Generally the computational performance of DSP processor is affected by its multiplier performance. The low power and high speed VLSI can be implemented with different logic style. The three important considerations for VLSI design are power, area and delay .there are many proposed logic low power consideration and high speed and each logic style has its own advantage in terms of speed and power. The objective of good multiplier is to provide a physically compact, high speed and low power consumption unit. Being a core part of arithmetic processing unit multipliers are in extremely high demand on its speed and low power consumption. To reduce significant power consumption of multiplier designs it in good direction to reduce the no. of operation thereby reducing a dynamic power which is a major part of total power dissipation. There are no. of techniques that to perform binary multiplication, low power multiplier using MAC unit ,modified booth multiplier, and low power multiplier using SPST are some of approaches to have hardware implementation of binary multiplier which are suitable for VLSI implementation at CMOS level.

2.3 MULTIPLICATION ALGORITHM

The multiplication algorithm for an N bit multiplicand by N bit multiplier is shown below:

Y= Yn-1 Yn-2 .....Y2 Y1 Y0 Multiplicand

X= Xn-1 Xn-2..... X2 X1 X0 Multiplier

Y= Yn-1 Yn-2 .....Y2 Y1 Y0

X= Xn-1 Xn-2..... X2 X1 X0

Yn-1X0 Yn-2X0 Yn-3X0 …… Y1X0 Y0X0

Yn-1X1 Yn-2X1 Yn-3X1 …… Y1X1 Y0X1

Yn-1X2 Yn-2X2 Yn-3X2 …… Y1X2 Y0X2

… … … … …. …. …. ….

Yn-1Xn-2 Yn-2X0 n-2 Yn-3X n-2 …… Y1Xn-2 Y0Xn-2

Yn-1Xn-1 Yn-2X0n-1 Yn-3Xn-1 …… Y1Xn-1 Y0Xn-1

P2n-1 P2n-2 P2n-3 P2 P1 P0

2.3.1 EXAMPLE

1101 4-bits

1101 4-bits

1101

0000

1101

1101

10101001

2.4 COMMON FEATURES OF MULTIPLIERS

(*i*) Counter flow Organization: A novel multiplier organization is introduced, in which the data bits flow in one direction, and the Booth commands are piggybacked on the acknowledgments flowing in the opposite direction.

(*ii*) Merged Arithmetic/Shifter Unit: An architectural optimization is introduced that merges the arithmetic operations and the shift operation into the same function unit, thereby obtaining significant improvement in area, energy and speed.

***(iii)* Overlapped Execution:** The entire design is pipelined at the bit-level, which allows overlapped execution of Proceedings of multiple iterations of the Booth algorithm, including across successive multiplications. As a result, both the cycle time per Booth iteration, as well as the overall cycle time per multiplication is significantly improved.

(*iv*) Modular Design: The design is quite modular, which allows the implementation to be scaled to arbitrary operand widths without the need for gate resizing, and without incurring any overhead on iteration time.

2.5 ANALYSIS OF DIFFERENT MULTIPLIERS

Generally multiplication consists of three steps: generation of partial products or PPs (PPG), reduction of partial products (PPR), and final carry-propagate addition (CPA). Different multiplication algorithms vary in the approaches of PPG, PPR, and CA. Multiplication is basically a shift add operation. There are, however, many variations on how to do it. Some are more suitable for FPGA use than others; some of them may be efficient for a system like CPU. This section explores various varieties and attracting

features of multiplication hardware. The multiplier area is quadratically related to the operand precision. Second, parallel multipliers have many logic levels that introduce spurious transitions or glitches. Third, the structure of parallel multipliers could be very complex in order to achieve high speed, which deteriorates the efficiency of layout and circuit level optimization. As a fundamental arithmetic operation, multiplication has many algorithm-level and bit-level computation features in which it differs from random logic. These features have not been considered well in low-level power optimization. It is also difficult to consider input data characteristics at low levels. Therefore, it is desirable to develop algorithm and architecture level power optimization techniques.

2.6 MULTIPLICATION TECHNIQUES

There are different algorithms used for multiplication. The design of those multipliers is discussed below.

2.6.1 SHIFT AND ADD MULTIPLIER

Shift-and-add multiplication is similar to the multiplication performed by paper and pencil. This method adds the multiplicand X to itself Y times, where Y denotes the multiplier. To multiply two numbers by paper and pencil, the algorithms is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results.

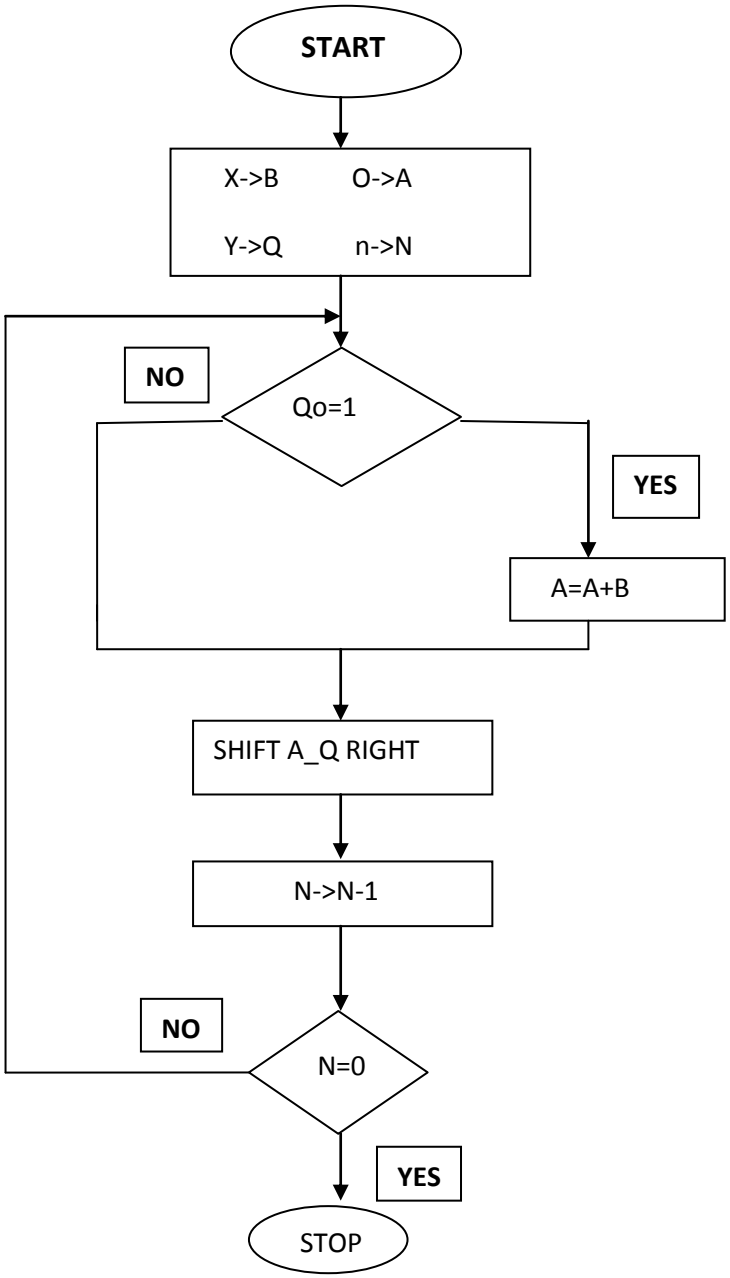
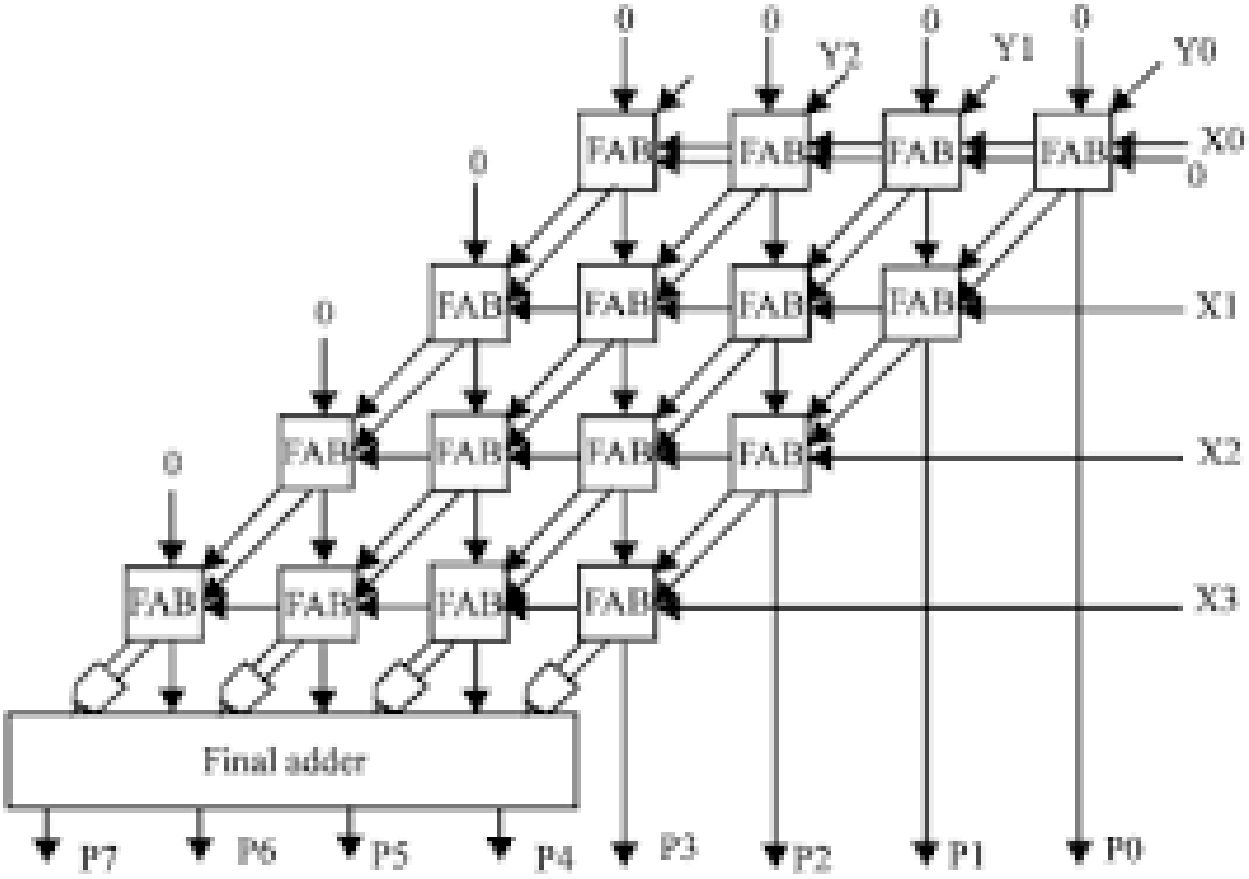


Fig.2.1 Flowchart for Shift & Add

2.6.2 Array Multiplier

Array multiplier is well known due to its regular structure. Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added. The addition can be performed with normal carry propagate adder. N-1 adders are required where N is the multiplier length.



2.2 Figure for array multiplier

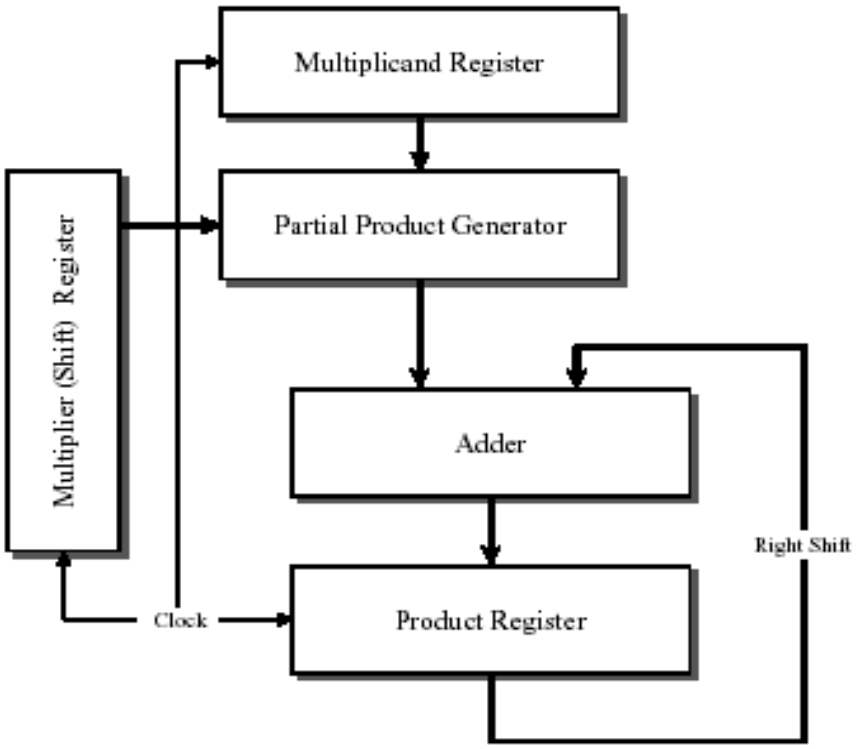


Fig 2.3 Block Diagram

The array consists of three rows of adder cells . Each row is responsible for adding a shifted form of the multiplicand to the partial product and passing the partial product to the row below. Each adder cell contains a 1-bit full adder with three inputs of equal binary value, labelled a, b, and c. Each cell has two outputs representing the sum (s) and carry (cry) that result from adding together the three inputs. The sum output represents the same binary weight as each of the three inputs. The carry output represents twice the binary weight of the sum output. As you can see from the figure, each cell combines a sum and a carry input from cells earlier in the array with a product bit, labelled P_{ij}.

The product bits P_{ij} are generated by combining multiplicand and multiplier bits using an and gate, P_{ij} D Q_i ^ R_j, where Q_i are bits of the

multiplier and R_j are bits of the multiplicand, again using the notation that bit j has weight 2j. The effect is that a row of cells adds 0 to the partial product if the corresponding bit of the multiplier is 0 and adds the multiplicand if it is 1. The structure of the array causes the multiplicand to shift to the left, corresponding to the weight of the multiplier bit. The multiplier Q and multiplicand R are both stored in registers operated at the same time as the partial-product register T. Of course, at the end of the first cycle, the multiplier must be shifted three bits to the right so that in the second cycle the high-order three bits of the multiplier are used to control the array. Neither the registers Q and R nor the shifting logic appears in the figure. Also not shown are registers to save the low-order three bits of the result of the first cycle (T−1, T−2, and T−3).

The design task is to make the long paths through the arrays of AND gates and adders operate quickly. In order to obtain results that are somewhat more general than the example, we'll represent the number of bits in the multiplicand by the symbol' m' and the number of rows in the multiplier array by 'n'. So the longest path through the array has an AND gate and n adder cells speed-space trade-off.

Advantages: First advantage of the array multiplier is that it has a regular structure. Since it is regular, it is easy to layout and has a small size. A second advantage of the array multiplier is its ease of design for a pipelined architecture.

Limitations: Major limitation of array multiplier is its size. As operand sizes increase, arrays grow in size at a rate equal to the square of the operand size.

CHAPTER 3

3.1 BOOTH MULTIPLIER

The Booth algorithm was invented by A. D. Booth forms the base of Signed number multiplication algorithms that are simple to implement at the hardware level, and that have the potential to speed up signed multiplication considerably. It is a powerful algorithm for signed-number multiplication, which treats both positive and negative numbers uniformly. For the standard add-shift operation, each multiplier bit generates one multiple of the multiplicand to be added to the partial product. If the multiplier is very large, then a large number of multiplicands have to be added.

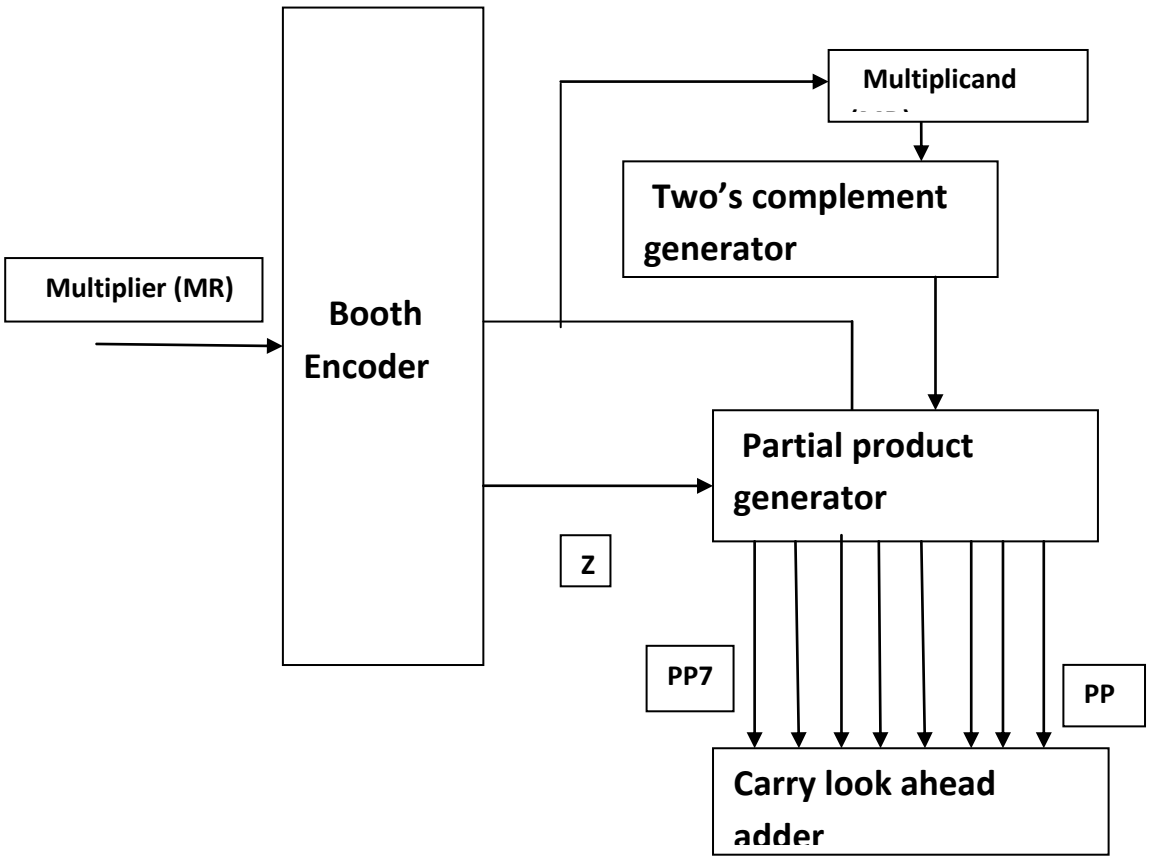


Fig 3.1 Architecture of Booth Multiplier

3.2 BOOTH RECODING TABLE FOR RADIX-2

Yi	Yi-1	Zi-1	Multiplier Value	Situation
0	0	0	0	String of 0s
0	1	1	+1	End of string of l s
1	0	1	-1	Begin string of 1s
1	1	0	0	String of 1s

3.3 BOOTH MULTIPLICATION ALGORITHM

Booth Multiplication Algorithm for radix 2

Booth algorithm gives a procedure for multiplying binary integers in signed –2's complement representation. Illustration of the booth algorithm is explained with the following example:

3.4 Step 1: Making the Booth table:

E.g.: 2x (- 4)

In 2's complement notation, 0010 * 1100

I. From the two numbers, pick the number with the smallest difference between series of consecutive numbers, and make it a multiplier.

i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change, and so there are two changes on this one

1100—from 1 to 1 no change, 1 to 0 one change , 0 to 0 no change, so there is only one change on this one.

Therefore, multiplication of 2 x (– 4), where 2(0010) is the multiplicand and (– 4) (1100) is the multiplier.

II. Let X = 1100 (multiplier)

Let Y = 0010 (multiplicand)

Take the 2's complement of Y and call it –Y

–Y = 1110

III. Load the X value in the table.

IV. Load 0 for X-1 value it should be the previous first least significant bit of X

V. Load 0 in U and V rows which will have the product of X and Y at the end of operation.

VI. Make four rows for each cycle; this is because we are multiplying four bits numbers.

U	V	X	(X-1)
0000	0000	1100	0

Step 2: Booth Algorithm:

Booth algorithm requires examination of the multiplier bits, and shifting of the partial product. Prior to the shifting, the multiplicand may be added to partial product, subtracted from the partial product, or left unchanged according to the following rules: Look at the first least significant bits of the multiplier "X", and the previous least

Significant bits of the multiplier "X - 1".

I. 0 0 Shift only

1 1 Shift only.

0 1 Add Y to U, and shift

1 0 Subtract Y from U, and shift or add (-Y) to U and shift

II. Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.

III. Shift X circular right shifts because this will prevent us from using two registers for the X value.

Shift only

U	V	X	(X-1)
0000	0000	1100	**0**
0000	0000	0110	0

Repeat the same step until the four cycles are completed

U	V	X	(X-1)
0000	0000	1100	0
0000	0000	0110	**0**
0000	0000	0011	0

In 4th row: Add-Y(0000+1110=1110)

Shift only

U	V	X	(X-1)
0000	0000	1100	0
0000	0000	0110	0
0000	0000	0011	**0**
1110 **1111**	**0000** **0000**	**0011** **1001**	**0** **1**

In 5th row: shift only

U	V	X	(X-1)
0000	0000	1100	0
0000	0000	0110	0
0000	0000	0011	0
1110 1111	0000 0000	0011 1001	0 1
1111	**1000**	**1100**	**1**

3.5 EXAMPLE

BR= 10111

QR= 10011

Qn	Qn+1	BR=10111 BR+1=01001	AC	QR	Qn+1
		Initial	**00000**	**10011**	**0**
1	**0**	**Subtract BR**	**01001**		
			01001		
		Shift Right	**00100**	**11001**	**1**
1	**1**	**Shift Right**	**00010**	**01100**	**1**
0	**1**	**Add BR**	**10111**		
			11001		
		Shift Right	**11100**	**10110**	**0**
0	**0**	**Shift Right**	**11110**	**01011**	**0**
1	**0**	**Subtract BR**	**01001**		
			00111		
		Shift Right	**00011**	**10101**	**1**

CHAPTER 4

4.1 MODIFIED BOOTH MULTIPLIER

In order to achieve high-speed multiplication, multiplication algorithms using parallel counters, such as the modified Booth algorithm has been proposed (Booth 1951) and some multipliers based on the algorithms have been implemented for practical use. This type of multiplier operates much faster than an array multiplier for longer operands because its computation time is proportional to the logarithm of the word length of operands. The modified Booth multiplier on the processor is a major source of energy consumption for Digital Signal Processing programs. Given a pair of values to be multiplied, power should be reduced if the value is put in with the lower recoding weight into the second input of the modified Booth multiplier. Booth multiplication is a technique that allows for smaller, faster multiplication circuits, by recoding the numbers that are multiplied. It is the standard technique used in chip design, and it provides significant improvements over the "long multiplication" technique. First, the "shift and add", or normal "long multiplication" multiplier design is briefly reviewed.

4.2 MODIFIED BOOTH MULTIPLICATION ALGORITHM FOR RADIX4

It is possible to reduce the number of partial products by half, by using the technique of radix 4 Booth recoding. The basic idea is that, instead of shifting and adding for every column of the multiplier term and multiplying by 1 or 0 for second column, multiply by ±1, ±2 or 0 to obtain the same results. One of the solutions of realizing high speed multipliers is to enhance parallelism which helps to decrease the number of subsequent calculation stages. The original version of the Booth algorithm (Radix-2) had two drawbacks. They are:

(i) The number of add/subtract operations and the number of shift operations becomes variable and becomes inconvenient in designing parallel multipliers.

(ii) The algorithm becomes inefficient when there are isolated 1's. These problems are overcome by using modified Radix4 Booth algorithm which scan strings of three bits with the algorithm given below:

1) Extend the sign bit 1 position if necessary to ensure that n is even.

2) Append a 0 to the right of the LSB of the multiplier.

3) According to the value of each vector , each Partial Product will he 0, +y , -y, +2y or -2y.

The negative values of y are made by taking the 2's complement and in this paper Carry-look-ahead (CLA) fast adders are used. The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, in designing a n-bit parallel multipliers, only n/2 partial products are generated.

4.2.1 ALGORITHM

Radix-4 Booth algorithm which scan strings of three bits with the algorithm given below:

(1) Extend the sign bit 1 position if necessary to ensure that n is even.

(2) Append a 0 to the right of the LSB of the multiplier.

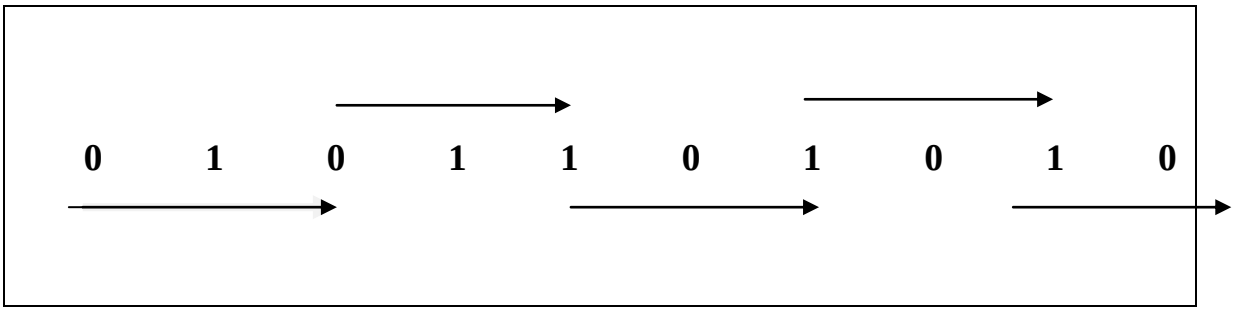
(3) According to the value of each vector, each Partial Product will be 0, $+y$, $-y$, $+2y$ or $-2y$.

The negative values of *y* are made by taking the 2's complement and Carry-look-ahead (CLA) fast adders are used. The multiplication of *y* is done by shifting *y* by one bit to the left. Thus, in any case, in designing n-bit parallel Multipliers, only *n*/2 partial products are generated [10, 11, and 12].

To Booth recode the multiplier term, we consider the bits in blocks of three, such that each block overlaps the previous block by one bit. Grouping starts from the LSB, and the first block only uses two bits of the multiplier.

Let us consider an example:
Multiplicand - (001011)2
Multiplier - (010011)2
First of all we will make group of three bits for Multiplier

4.2.2 Grouping of bits for multiplier



4.2.3 ENCODING FOR RADIX-4 BOOTH MULTIPLIER

X(i)	X(i-1)	X(i-2)	Y
0	0	0	+0
0	0	1	+Y
0	1	0	+Y
0	1	1	+2Y
1	0	0	-2Y
1	0	1	-Y
1	1	0	-Y
1	1	1	+0

From the table 1:

(010)2 - 1

(001)2 - 1

(110)2 - (-1)

Therefore Multiplicand is multiplied with the three

Encoded digit which is 1, 1 and (–1).

(i) –1 * (001011)2 = (001011)2 And 1111 added with the result because of negative

Sign . Thus final answer of multiplication of (-1) is

(1111001011)2 {Negative term Sign Extended}

(*ii*) 1 * (001011)2 = (001011)2

(*iii*) 1 * (001011)2 = (001011)2

(*iv*) (00001)2 are added with these three resultants as

An error correction for negation.

4.3 UNSIGNED NUMBERS

4.3.1 Algorithm

1. Pad the LSB with one zero
2. Pad the MSB with 2 zeros if n is even (n/2+1 PP'S) and 1 zero if n is odd (n/2+1 PP'S)
3. Divide the multiplier into overlapping groups of 3 bits
4. Determine the partial product scale factor
5. Compute the multiplicand multiples
6. Sum partial products

4.3.2 EXAMPLE

8*20→

8=00001000 20=00010100

By algorithm, Multiplier 20= 00000101000

00001000	8
00000101000	20
0000000000000000	0*Y
00000000001000	1*Y
000000001000	1*Y
0000000000	0*Y
00000000	0*Y
0000000010100000	

4.3.3 Grouping concept

Multiplier bits after padding

0 0 0 0 0 1 0 1 **0 0 0** → 0*Y

0 0 0 0 0 1 **0 1 0** 0 0 → 1*Y

0 0 0 0 **0 1 0** 1 0 0 0 → 1*Y

0 0 **0 0 0** 1 0 1 0 0 0 → 0*Y

0 0 0 0 0 1 0 1 0 0 0 → 0*Y

4.4 SIGNED NUMBERS

4.4.1 ALGORITHM

1. Pad the LSB with one zero
2. If n is even don't pad the MSB (n/2 PP'S) and if n is odd sign extend the MSB by 1 bit(n+1/2 PP'S)
3. Divide the multiplier into overlapping groups of 3 bits
4. Determine the partial product scale factor
5. Compute the multiplicand multiples
6. Sum partial products

4.4.2 EXAMPLE

-107*105→ -107=10010101 105=01101001

By algorithm, Multiplier 105= 011010010

10010101	-107
011010010	105
1111111110010101	1*Y
00000011010110	-2*Y
000001101011	-1*Y
0100101010	2*Y
1101010000011101	-11235

CHAPTER 5

5.1 SPST (SPURIOUS POWER SUPPRESSION TECHNIQUE)

The spurious-power suppression technique (SPST) can dramatically reduce the power dissipation of combinational VLSI designs for multimedia/DSP purposes. The proposed SPST separates the target designs into two parts, i.e., the most significant part and least significant part (MSP and LSP), and turns off the MSP when it does not affect the computation results to save power. Furthermore, glitch-diminishing technique is used to filter out useless switching power by asserting the data signals after the data transient period.

This is the technique which is used to suppress the spurious power dissipation existed in the data-paths for multimedia VLSI designs. The proposed technique adopts the design concept of separating the arithmetic units into Most Significant Part (MSP) and Least Significant Part (LSP), and then freezing the MSP whenever this part of circuits does not affect the computation result.

5.1.1 SPST AS PRECOMPUTATION LOGIC

Pre Computation logic is one of the efficient Low power VLSI techniques to reduce the useless power dissipation in the circuits. The influence of the spurious signal transitions illustrated as five cases of a 16-bit addition is explored as shown in Fig. The case1 illustrates a transient state in which the Spurious transitions of carry signals occur in the MSP though the fina1 result of the MSP are unchanged. The 2nd and 3rd cases describe the situations of one negative operand adding another positive operand without and with carry from LSP, respectively. Moreover, the 4th 5th cases respectively demonstrate the conditions of two negative operands addition

without and with carry-in from LSP. In those cases, the results of the MSP are predictable, therefore the computations in the MSP are useless and can be neglected. Eliminating those spurious computations will not only save the power consumed inside the SPST adder/ subtractor but also decrease the glitching noises which will affect the next arithmetic circuits.

5.1.2 CASES FOR PRECOMPUTATIONAL LOGIC

Case 1:

0000 0001 LSP Carry=0

0000 1110

0000 1111

Case 2:

0000 1001 LSP Carry=1

0000 1110

0001 0111

Case 3:

1111 0001 LSP Carry=0

1111 1110

1110 1111

Case 4:

1111 1001 LSP Carry=1

1111 1110

1111 0111

5.2 BASIC BLOCK DIAGRAM

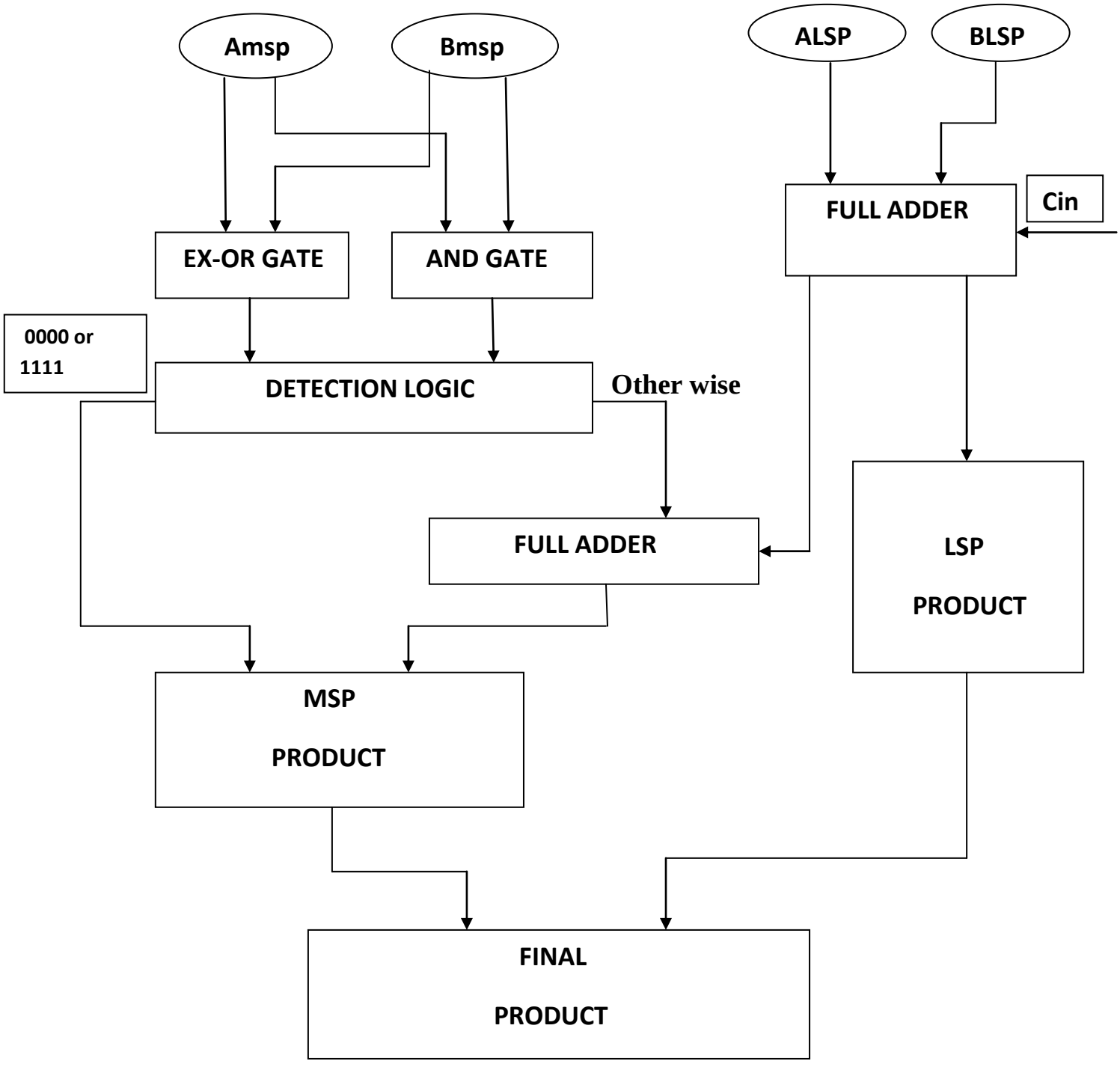


Fig 5.1 Block diagram of SPST Adder

5.3 DETECTION LOGIC

The Boolean logical equations from 3.1 to 3.6 given below express the behavioural principles of the detection logic unit in the MSP circuits of the SPST-based adder/subtract or:

AMSP = A [7:4]; B MSP = B [7:4]; (3.1)

X [3:0] = A [7:4] xor B[7:4]; (3.2)

i.e X [0] = A [4] xor B [4] ;

X [1] = A [3] xor B [3];

X [2] = A [2] xor B [2];

X [3] = A [1] xor B [1];

Y [3:0] = A [7:4] xor B [7:4]; (3.3)

I.e Y [0] = A [4] xor B [4];

Y [1] = A [3] xor B [3];

Y [2] = A [2] xor B [2];

Y [3] = A [1] xor B [1];

X [3:0] = A [7:4] and B [7:4]; (3.4)

Y [3:0] = A [7:4] and B [7:4]; (3.5)

Where ‘A[m]’ and ‘B[n]’ respectively denote the mth bit of the operands, A and the nth bit of the operand ‘B’ and ‘AMSP’ and ‘BMSP’ respectively denote the MSP parts, i.e. the 9th bit to the 16th bit, of the operands ‘A’ and ‘B’.

When the bits in ‘AMSP, and those in ‘BMSP’ are all same, the value of ‘Axor’ and that of ‘Bxor’ respectively become zero and ‘Aand’ , ‘Band’ will be either zeros or ones. If this and LSP carry is 0, then final result of MSP will be ‘0000’.

If LSP carry is 1 then ‘0001’.If ‘Aand’ , ‘Band’ bits in AMSP and/or those in BMSP are all zeros, the value of ‘Anor’ and/or that of ‘Bnor’ respectively turn into one.

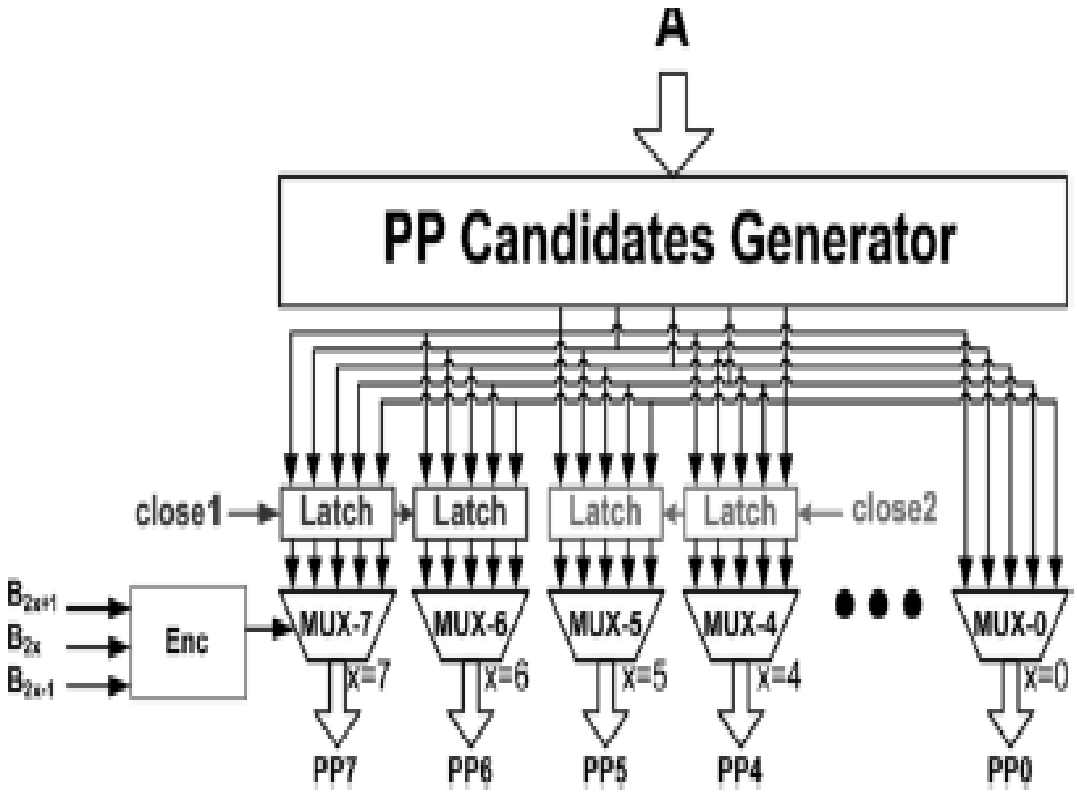
CHAPTER 6

6.1 APPLYING SPST ON MODIFIED BOOTH ENCODER

While multiplying two 16 bit numbers through booth algorithm eight partial products produced, but some of the partial product may contain all the bits as zero, so saving those computations can significantly reduce the power consumption caused by the transient signals. We propose the SPST-equipped modified-Booth encoder, which is controlled by a detection unit. The detection unit has one of the two operands as its input to decide whether the Booth encoder calculates redundant computations. As shown in Fig. 5, the latches can respectively, freeze the inputs of MUX-4 to MUX- 7 or only those of MUX-6 to MUX-7 when the PP4 to PP7 or the PP6 to PP7 are zero, to reduce the transition power dissipation. Such cases occur frequently in e.g., FFT/IFFT, DCT/IDCT, and Q/IQ which are adopted in encoding or decoding multimedia data.

6.1.1 Proposed Spurious Power Suppression Technique

It is a 16-bit adder/ subtractor design based on the proposed SPST. This example, the 16-bit adder/ subtractor, is divided into MSP and LSP at the place between the 8th bit and the 9th bit. Latches implemented by simple AND gates are used to control the input data of the MSP. When the MSP is required to produce the result, the input data of MSP remain the same as usual. When the MSP is not required to produce the result, the input data of the MSP become zeros to avoid switching power consumption. From the derived Boolean equations (3.1) to (3.5), the detection logic unit of the SPST which can determine whether the input data of MSP should be latched or not.



6.1 Fig: Diagram for pp candidates generator

Partial product generator is the combination circuit of the product generator. Product generator is designed to produce the product by multiplying the multiplicand X by 0, 1, -1, 2 or -2. For product generator, multiply by zero means the multiplicand is multiplied by "0". Multiply by "1" means the product still remains the same as the multiplicand value. Multiply by "-1" means that the product is the two‟s complement form of the number. Multiply by "-2" is to shift left one bit the two‟s complement of the multiplicand value and multiply by "2" means just shift left the multiplicand by one place.

The PP generator generates five candidates of the partial products, i.e., {-2A,-A, 0, A, 2A}. These are then selected according to the Booth

encoding results of the operand B. When the operand besides the Booth encoded one has a small absolute value, there are opportunities to reduce the spurious power dissipated in the compression tree.

For the partial product generation, we adopt Radix-4 Modified Booth algorithm to reduce the number of partial products for roughly one half. For multiplication of 2’s complement numbers, the two-bit encoding using this algorithm scans a triplet of bits. When the multiplier B is divided into groups of two bits, the algorithm is applied to this group of divided bits.

CHAPTER 7

7.1 BOOTH MULTIPLIER

7.1.1 OUTPUT

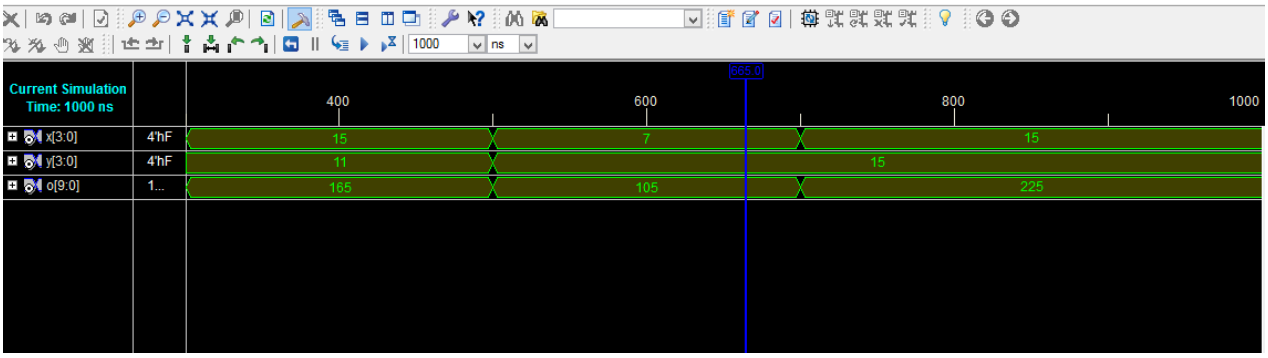


Fig 7.1 Booth Multipier output

7.1.2 RTL SCHEMATIC

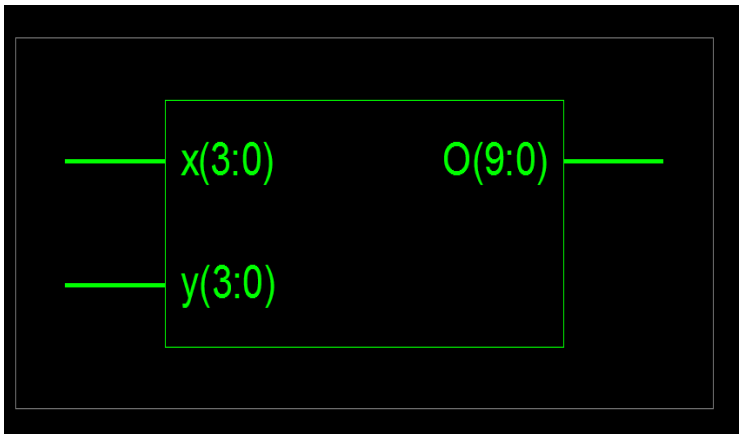
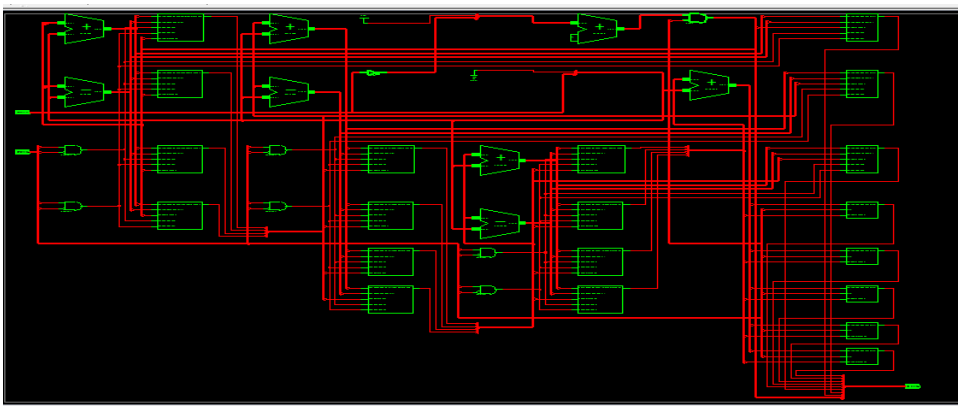


Fig 7.2 RTL schematic for Booth Multipier

7.1.3 TECHNOLOGY SCHEMATIC



7.2 MODIFIED BOOTH

7.2.1 OUTPUT

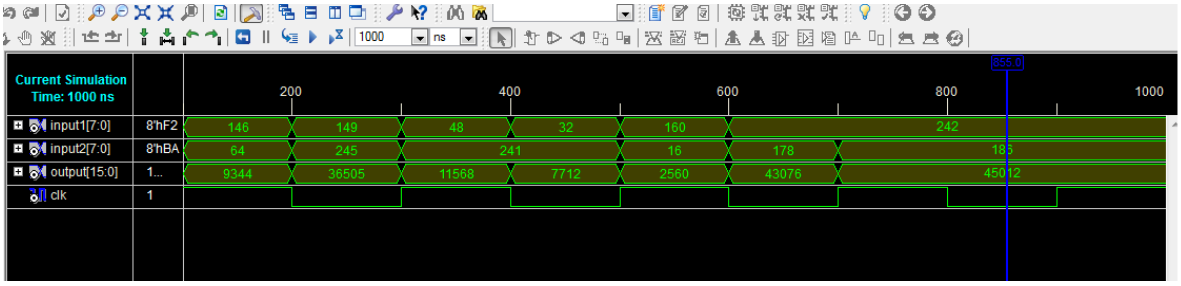


Fig 7.3 Modified Booth Multipier output

7.2.2 RTL SCHEMATIC

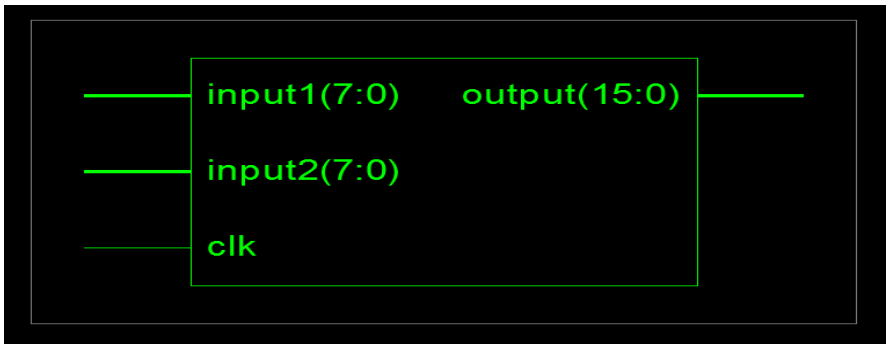
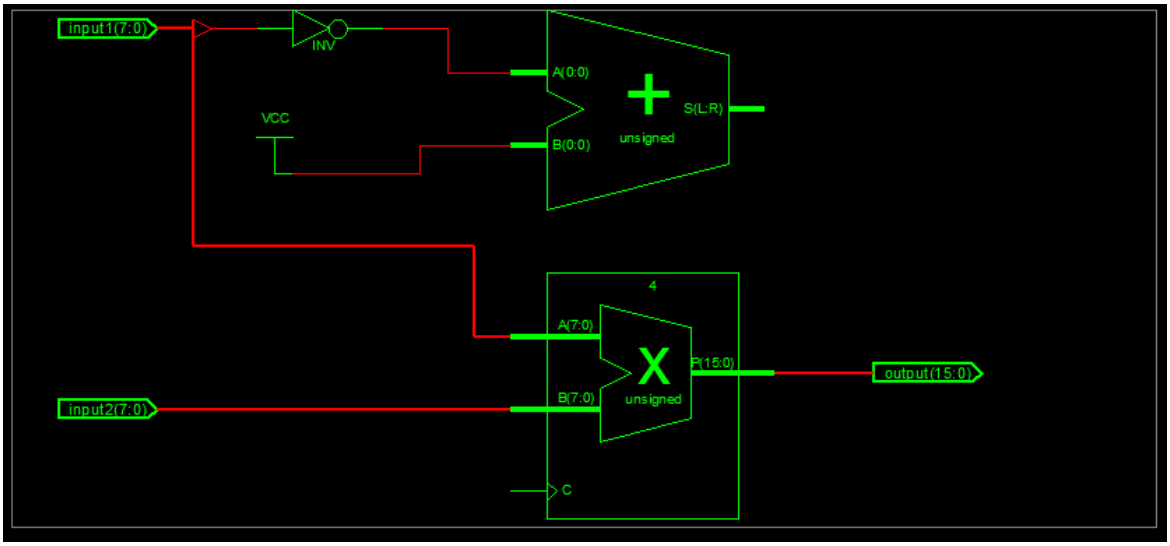


Fig 7.4 RTL Schematic for Modified Booth Multipier

7.2.3 TECHNOLOGY SCHEMATIC



7.3 SPST ADDER

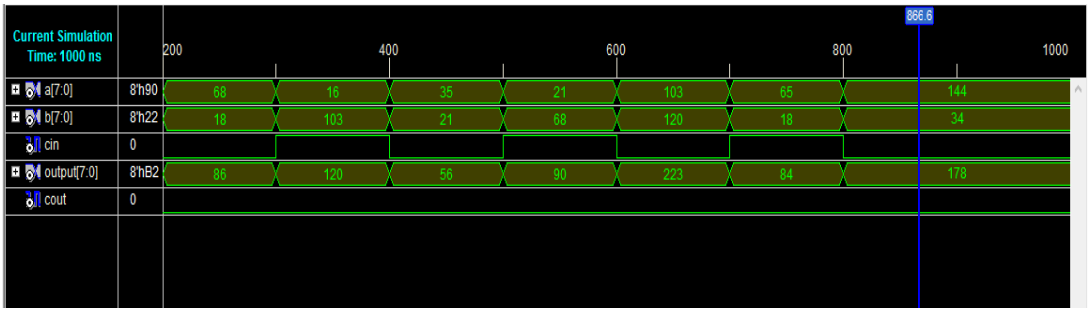


Fig 7.5 SPST Adder Output

7.4 MODIFIED BOOTH WITH SPST

7.4.1 OUTPUT

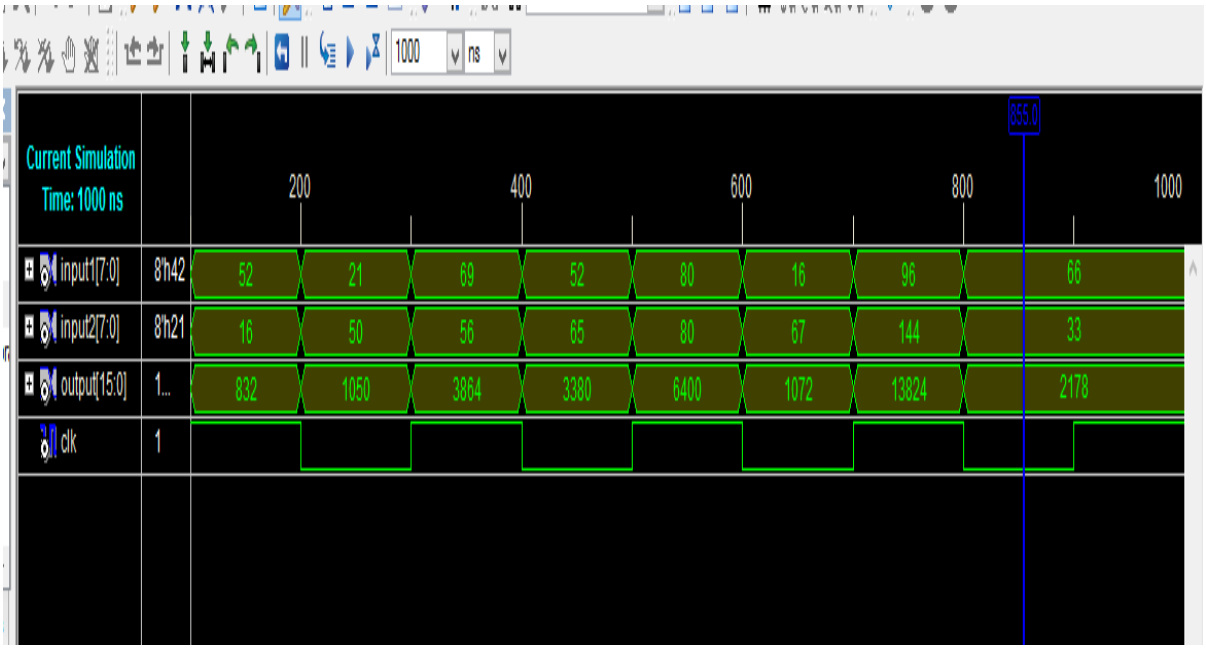


Fig 7.6 Modified Booth Multiplier Output

7.4.2 RTL SCHEMATIC

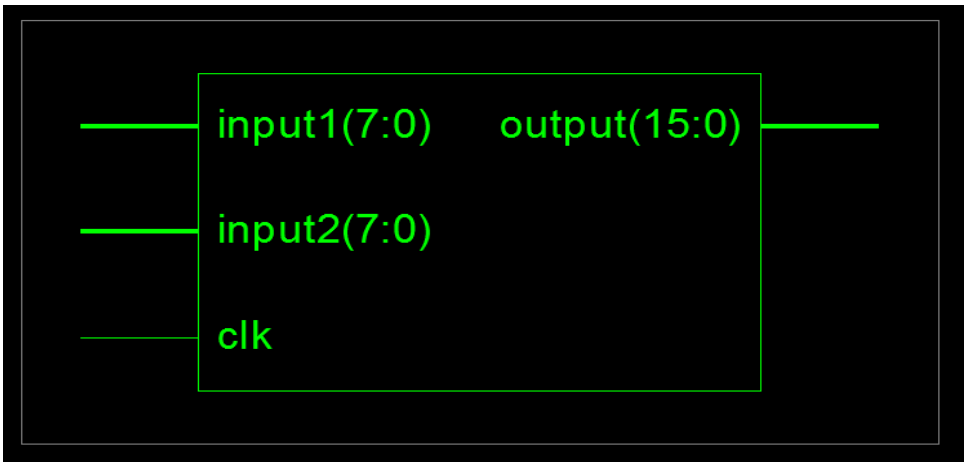
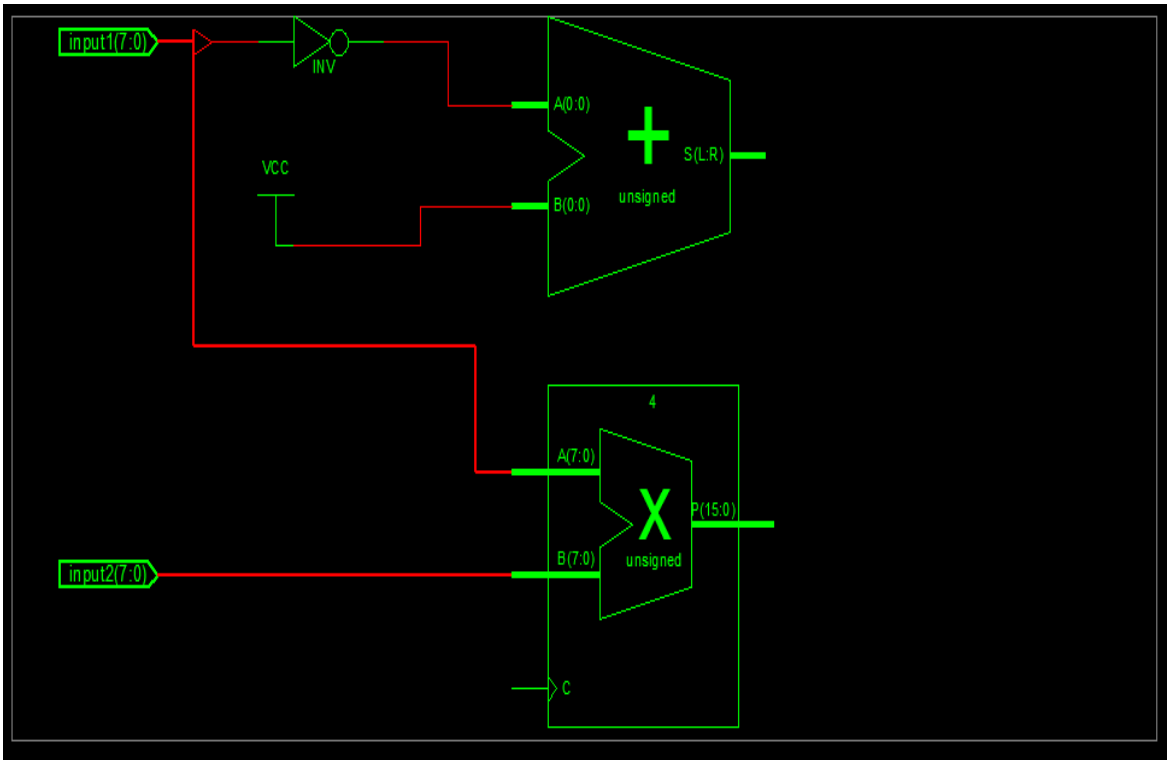


Fig 7.7 RTL Schematic for Modified Booth Multiplier with SPST

7.4.3 TECHNOLOGY SCHEMATIC



CHAPTER 8

8.1 DEVICE UTILISATION

8.1.1 DEVICE UTILISATION FOR BOOTH MULTIPLIER

PARAMETERs	USED	AVAILABLE	%
Number of Slices	42	4656	1
Number of 4 input LUT	80	9312	1
Number of IOs	864	864	-
Number of bonded IOB	18	190	9

8.1.2 DEVICE UTILISATION FOR MODIFIED BOOTH MULTIPLIER

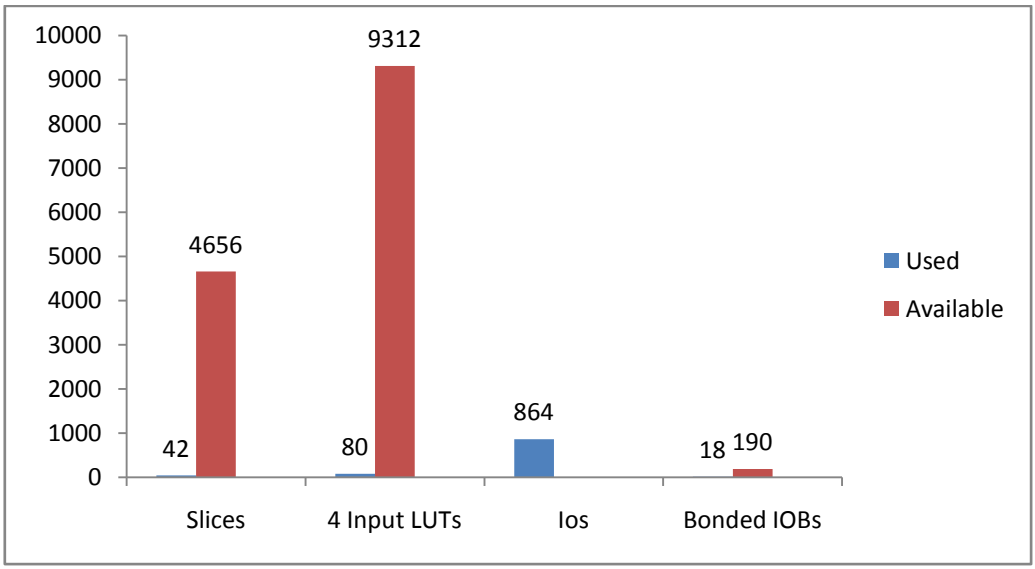
PARAMETERs	USED	AVAILABLE	%
Number of Slices	0	4656	0
Number of MULT18X18SIOs	1	20	5
Number of IOs	1536	1536	-
Number of bonded IOB	32	190	16

8.1.3DEVICE UTILISATION FOR MODIFIED BOOTH MULTIPLIER WITH SPST

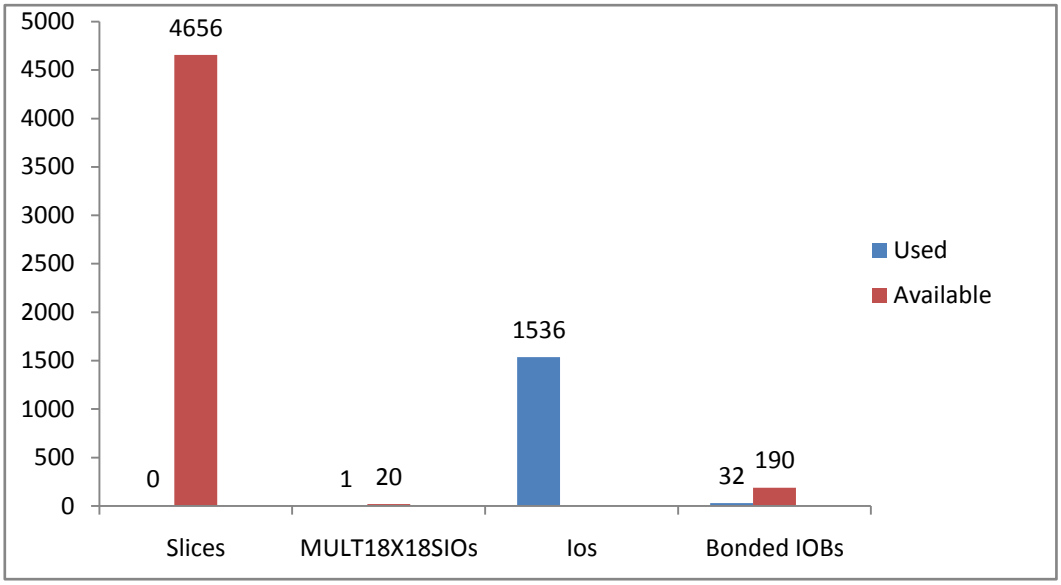
PARAMETERs	USED	AVAILABLE	%
Number of Slices	0	4656	0
Number of MULT18X18SIOs	1	20	5
Number of IOs	1536	1536	-
Number of bonded IOB	32	190	16

8.2 BAR CHARTS

8.2.1 BOOTH MULTIPLIER



8.2.2 MODIFIED BOOTH MULTIPLIER



8.3 COMPARISON OF MULTIPLIERS

S.No	Multiplier Type	Vendor	Device	Estimated Delay (ns)	Power Dissipation (mW)
1.	Booth Multiplier	Xilinx	Automotive Spartan 3E	14.610ns	92.07
2.	Modified Booth Multiplier	Xilinx	Automotive Spartan 3E	9.286ns	90.48
3.	Modified Booth Multiplier with SPST	Xilinx	Automotive Spartan 3E	9.286ns	90.47

CHAPTER 9

CONCLUSION

Today every circuit has to face power consumption issue to have long battery life. Multiplication occurs frequently in finite impulse response filters, fast Fourier transforms, discrete cosine transforms, convolution, and other important DSP, Micro processor applications. Many algorithms are used for multiplication. But power consumption is different for different algorithms. To have low power efficient Multiplier architecture, Radix-4 Modified Booth Algorithm adopting the SPST technique is designed.

The Multiplier is designed by equipping SPST on a modified Booth encoder which is controlled by a detection unit using AND gate and XOR gate. The modified Booth encoder will reduce the number of partial products generated by a factor two. The SPST adder will avoid the unwanted addition and thus minimize the switching power dissipation. This can attain significant speed improvement and power reduction when compared with other multipliers. While comparing all algorithms, power consumption and delay is very low for radix-4 modified booth algorithm with SPST technique. Modified Booth Multiplier has 5.32ns less delay and 1.59mW low power when comapared to booth multiplier. Thus based on the experimental values we conclude that Radix-4 modified booth algorithm with SPST is efficient and consumes low power for VLSI multiplier architecture.

REFERENCE

1. Sukhmeet Kaur, Suman and Manpreet Signh Manna(2013), "*Implementation of Modified Booth Algorithm (Radix 4) and its Comparison with Booth Algorithm (Radix-2)*", Research India Publications, ISSN 2231-1297, Volume 3, Number 6 , pp. 683-690.

2. G.Sasi, "*Design of Low power /High speed multiplier using Spurious power suppression Technique(SPST)*",International Journal of Computer Science and Mobile Computing, ISSN 2320-088X,vol-3,2014.

3. Sneha Manohar Ramteke, Alok Dubey, Yogeshwar Khandagare, "*VLSI Designing of Low Power Radix 4 Booths Multiplier*", International journal of Electrical,Electronics and Computer Systems(IJEECS)

4. Iffat Fatima, "*Analysis of Multipliers in VLSI"*, Journal of Global Research in Computer science (2012),ISSN 2229-371X ,vol 3,No.11.

5. S. JAGADEESH , S.VENKATA CHARY," *Design of Parallel Multiplier–Accumulator Based on Radix-4 Modified Booth Algorithm with SPST"*, International Journal Of Engineering Research And Applications (IJERA) 2012,ISSN: 2248-9622 Vol. 2, pp.425-431

6. Ken Chapman, *"Initial Design for Spartan-3E Starter Kit (LCD Display Control)"*, Xilinx Ltd 16th February 2006.

7. SPST based power optimized multiplier

8. *www.google.com*

9. *www.ieee.org*

10. *www.multiplier.org*

APPENDIX

XILINX (VHDL LANGUAGE)

The Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors. The Xilinx ISE is primarily used for circuit synthesis and design, while the Model Sim logic simulator is used for system-level testing .Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and Chip Scope.

The primary user interface of the ISE is the Project Navigator, which includes the design hierarchy (Sources), a source code editor (Workplace), an output console (Transcript), and a processes tree (Processes). The Design hierarchy consists of design files (modules), whose dependencies are interpreted by the ISE and displayed as a tree structure. For single-chip designs there may be one main module, with other modules included by the main module, similar to the main() subroutine in C++ programs. Design constraints are specified in modules, which include pin configuration and mapping. The Processes hierarchy describes the operations that the ISE will perform on the currently active module. The hierarchy includes compilation functions, their dependency functions, and other utilities. The window also denotes issues or errors that arise with each function. The Transcript window provides status of currently running operations, and informs engineers on design issues. Such issues may be filtered to show Warnings, Errors, or both. Simulation System-level testing may be performed with the Model Sim logic simulator, and such test programs must also be written in HDL languages. Test bench programs may include simulated input signal waveforms, or monitors which observe and verify the outputs of the device under test.

Model Sim may be used to perform the following types of simulations:

- Logical verification, to ensure the module produces expected results
- Behavioural verification, to verify logical and timing issues
- Post-place & route simulation, to verify behaviour 1 2 4 EXTERNAL LINKS after placement of the module within the reconfigurable logic of the FPGA.

Xilinx's patented algorithms for synthesis allow designs to run upto 30% faster than competing programs, and allow greater logic density which reduces project costs. Also, due to the increasing complexity of FPGA fabric, including memory blocks as and I/O blocks, more complex synthesis algorithms were developed that separate unrelated modules into slices, reducing post-placement errors. IP Cores are offered by Xilinx and other third-party vendors, to implement system-level functions such as digital signal processing (DSP), bus interfaces, networking protocols, image processing, embedded processors, and peripherals. Xilinx has been instrumental in shifting designs from ASIC-based implementation to FPGA based implementation.

VHDL

VHDL (VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming language.

VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that

interface to the design. This collection of simulation models is commonly called a *testbench*.

One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE, Altera Quartus, Synopsys Synplify or Mentor Graphics HDL Designer) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench. To generate an appropriate test bench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required.

In VHDL, a design consists at a minimum of an entity which describes the interface and an architecture which contains the actual implementation. In addition, most designs import library modules. Some designs also contain multiple architectures and configurations.

A simple AND gate program in VHDL,

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity ANDGATE is
Port(
  I1: in std_logic;
  I2: in std_logic;
  O: out std_logic);
end entity ANDGATE;
Architecture RTL of ANDGATE is
```

```
begin
  O<=I1 and I2;
end architecture RTL;
```

FPGA IMPLIMENTATION

FPGAs are programmable semiconductor devices that are based around a matrix of Configurable Logic Blocks (CLBs) connected through programmable interconnects. As opposed to Application Specific Integrated Circuits (ASICs), where the device is custom built for the particular design, FPGAs can be programmed to the desired application or functionality requirements. Although One-Time Programmable (OTP) FPGAs are available, the dominant type is SRAM-based which can be reprogrammed as the design evolves. FPGAs allow designers to change their designs very late in the design cycle– even after the end product has been manufactured and deployed in the field. In addition, Xilinx FPGAs allow for field upgrades to be completed remotely, eliminating the costs associated with re-designing or manually updating electronic systems

FPGAs contain an array of Programmable Logic Block, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic Block can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip flops or more complete blocks of memory.