DESIGN AND IMPLEMENTATION OF A 32 -BIT MAC UNIT WITH PIPELINED VARIABLE STAGE CARRY SELECT ADDER

Thesis submitted in the partial fulfillment of requirement for the award of degree of

Master of Technology

in

VLSI Design & CAD

Submitted by:

Chetan Gupta

Roll No : 601061006

Under the guidance of:

Mrs. Manu Bansal

Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

June 2012

DECLARATION

I, **CHETAN GUPTA**, hereby certify that the work which is being presented in this thesis entitled "**DESIGN AND IMPLEMENTATION OF A 32–BIT MAC UNIT WITH PIPELINED VARIABLE STAGE CARRY SELECT ADDER**" by me in partial fulfillment of the requirements for the award of degree of Master of Technology in VLSI Design from Thapar University (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of **Mrs.Manu Bansal**.

The matter presented in this thesis has not been submitted in any other University / Institute for the award of any other degree.

Date: 4/7/12

Chetan Gupta

Roll No.601061006

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Mrs. Manu Bansal

Assistant Professor

Thapar University

Patiala–147004,(Punjab)

Date: 4/7/12

Countersigned by:

(Dr. Rajesh Khanna)

Professor and Head ECED

Thapar University, Patiala

Date:

(Dr. S.K. Mohapatra)

Dean of Academic Affairs

Thapar University, Patiala

Date:

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to **Mrs. Manu Bansal** Assistant Professor**,** Electronics and Communication Engineering Department, Thapar University, Patiala for her patient guidance and support throughout this thesis. I am truly very fortunate to have the opportunity to work with her.

I am also thankful to our **Head of the Department, Dr. Rajesh Khanna** as well as **PG Coordinator, Dr. Kulbir Singh, Associate Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department and then friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would also like to thank my parents for their years of unyielding love and encourage. They have always wanted the best for me and I admire their determination and sacrifice.

Chetan Gupta

ABSTRACT

The addition and multiplication of two binary numbers is the fundamental and most often used arithmetic operation in microprocessors, digital signal processors, and data-processing application-specific integrated circuits. At the heart of data-path and addressing units in turn are arithmetic units, such as comparators, adders, and multipliers. Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier.

This MAC(Multiplier and Accumulator Unit) with 1 multiplier and 1 accumulator.The multiplier is composed of Booth encoder block, wallace tree, 64 bit carry select adder block. In 32bit × 32 bit multiplication, we have used modified Radix-4 Booth's algorithm where outputs of encoding block are partial products and additional 1bit signal were added to Wallace tree. By implementing Wallace tree with 4:2 CSA improves the regularity because c_out is independent to c_in. In fact, each stage acts at the same time. The final result is the sum of carry and the sum vector through 64bit carry select adder. The output of multiplier is accumulated by the accumulator and we used 64bit carry select adder in the accumulator.

Different adder architectures like Carry Chain Adder, Carry- look Ahead Adder, Carry Select Adder and Carry Skip Adder for different operand size like 16-bit,32-bit and 64-bit, are simulated and synthesized on FPGA using Xilinx ISE. On the basis of their synthesized results, one of the adder which have less delay and minimum area (Carry Select Adder) is chosen.

A 32-bit Pipelined Booth Wallace MAC Unit is designed in which the multiplication is done using the Modified Booth Wallace Multiplier and in the final stage addition of multiplier and in accumulator the Pipelined Variable Stage Carry Select Adder is used and the pipelining is done in the Booth Multiplier and Wallace Tree.

MAC unit is described in VHDL and synthesized the circuit using 90 nm standard cell library on FPGA. This MAC Unit (with Pipelined Variable Stage Carry select adder in the final stage of multiplication and accumulator)has higher speed than conventional or non pipelined Booth Wallace MAC and Pipelined Booth Wallace MAC (with non-pipelined Carry Select Adder).

TABLE OF CONTENTS

ABBREVIATIONS

ASIC	Application Specific Integrated Circuits
CLA	Carry-look ahead Adder
CLBs	Configurable logic blocks
CMOS	Complementary Metal Oxide Semiconductor.
Comb.	Combinational
CPA	Carry-propagate Adder
CSA	Carry-save Adder
CSKA	Carry-skip Adder
CSLA	Carry-select Adder
DC	Design Compiler
DCM	Digital Clock Manager
DSP	Digital Signal Processing
Dyn.	Dynamic
FA	Full Adder
FPGA	Field Programmable Gate Array
GRM	General Routing Matrix
HA	Half Adder
HDL	Hardware Description Language
IOBs	Input/output blocks
ISE	Integrated Software Environment
LCD	Liquid Crystal Display
Lkg.	Leakage
LUT	Look-up Tables
MAC	Multiply and Accumulator
MBA	Modified Booth's Algorithm
MBE	Modified Booth Encoding
NCD	Native Circuit Description

NGD	Native information and Generic Database
PAR	Place and Route
PIs	Programmable Interconnections
RAM	Random Access Memory
RCA	Ripple-carry Adder
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
UCF	User Constraints File
VHDL	Very High Speed Integrated Circuits HDL
VLSI	Very Large Scale Integration
PVCSLA	Pipelined Variable Stage Carry Select Adder

List of figures

List of tables

Chapter 1

Introduction

The addition and multiplication of two binary numbers is the fundamental and most often used arithmetic operation in microprocessors, digital signal processors, and data-processing application-specific integrated circuits. Therefore, binary adders and multipliers are crucial building blocks in VLSI circuits. This chapter introduces motivation, formally state the problem and present organization of Thesis.

1.1Motivation

The core of every microprocessor, DSP, and data-processing ASIC is its data path. Statistics showed that more than 70% of the instructions perform additions and multiplications in the data path of RISC machines [1]. At the heart of data-path and addressing units in turn are arithmetic units, such as comparators, adders, and multipliers. Digital multipliers are the most commonly used components in any digital circuit design. Multiplication based operations such as Multiply and Accumulate and inner product are among some of the frequently used Computation-Intensive Arithmetic Functions, currently implemented in many DSP applications such as convolution, fast Fourier transform, filtering and in microprocessors in its arithmetic and logic unit. Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier. Currently, multiplication time is still the dominant factor in determining the instruction cycle time of a DSP chip. The demand for high speed processing has been increasing as a result of expanding computer and signal processing applications. Higher throughput arithmetic operations are important to achieve the desired performance in many real-time signal and image processing applications. One of the key arithmetic operations in such applications is multiplication and the development of fast multiplier circuit has been a subject of interest over decades. Reducing the time delay and power consumption are very essential requirements for many applications.

Due to the importance of digital multipliers in DSP, it has always been an active area of research and moreover with the ever-increasing demand for portable electronic products an electronic component with low power consumption would survey lead the market trend.

Pipelined MAC based on Booth Wallace with Carry Select Adder is one of the fastest MAC Unit. Because modified Booth algorithm reduces the number of partial products to be generated and is known as the fastest multiplication algorithm and many researches on the multiplier architectures including array, parallel and pipelined multipliers have been pursued which shows that pipelining is the most widely used technique to reduce the propagation delays of digital circuits.

Finally, the basic operation found in MAC is the binary addition. Besides of the simple addition of two numbers, addition forms the basis for many processing operations, from counting to multiplication to filtering. But also simpler operations like incrimination and magnitude comparison based on binary addition. Therefore, binary addition is the most important arithmetic operation. It is also a very critical one if implemented in hardware because it involves an expensive carry-propagation step, the evaluation time of which is dependent on the operand word length. An extensive, almost endless, assortment of adder architectures serves different speed and area requirements. The efficient implementation of the addition operation in an integrated circuit is a key problem in VLSI design. Productivity in ASIC design is constantly improved by the use of cell-based design techniques such as standard cells, gate arrays, and field- programmable gate arrays (and by low-level and high-level hardware synthesis.

1.2 Problem Statement

The primary objective of this report is to study the binary adder and MAC architectures and to investigate and evaluate various existing adder architectures on the basis of speed, area and power. In this the various Adder architectures like Carry Chain Adder, Carry-look Ahead Adder, Carry Select Adder and Carry Skip Adder for different operand size like 4-bit, 8-bit,

16-bit, 32-bit and 64-bit are studied. We have noticed that in the ripple carry adder it takes the time to propagate the carry from one stage to the next stage , so this adder will introduce the overall delay in the circuit. To overcome this problem carry save adder came in to the role and had sort out this delay problem up to the some extent. The carry-look ahead adder [3] computes group generate signals as well as group propagate signals to avoid waiting for a ripple to determine if the first group generates a carry or not. From the point of view of carry propagation and the design of a carry network the actual operand digits are not important. What matters is whether in a given position a carry is generated, propagated or annihilated. The carry

skip adder [2][6][7] was invented for decimal arithmetic operations. The CSKA is an improvement over the ripple-carry adder. By grouping the ripple cells together into blocks, it makes the carry signal available to the blocks further down the carry chain, earlier. The primary carry c_i coming into a block can go out of it unchanged if and only if, x_i and y_i are exclusive-or of each other. This means that corresponding bits of both operands within a block should be dissimilar . The carry-select adder comes in the category of conditional sum adder [9]. Conditional sum adder works on some condition this scheme; blocks of bits are added in two ways: assuming an incoming carry of 0 or 1, with the correct outputs selected later as the block's true carry-in becomes known. And we have also discussed various multiplication algorithms in this report, and the most efficient and the most effective one is the modified booth algorithm. And lastly I have explained the basics of arithmetic pipelining, and had given the various examples which lead to prove that this pipelining could able to speed up the performance of the various arithmetic circuits such as adders , multipliers etc.

1.3 Thesis Organization

This report is organized into seven chapters.

Chapter 1 Introduces the motivation for thesis, outlines the problem addressed, and states the contributions of thesis work.

Chapter2 Starts with a literature review of the various adder architectures and the optimization techniques.

Chapter 3 Starts with a literature review of various Multiply schemes.

Chapter 4 Starts with a literature review of arithmetic pipelining.

Chapter 5 This chapter introduces the basics of binary multiplication, partial product generation, reduction and techniques to make the multiplication process faster.

Chapter 6 This chapter introduce about the FPGA concepts and FPGA Synthesis Flow.

Chapter 7 is mainly concerned with the Design Compiler. All Design and Timing attributes have been discussed in this chapter. The synthesis flow of Design Compiler is explained.

Chapter 8 This chapter introduces the simulation and synthesis results of Adder and MAC Unit synthesized by Xilinx ISE.

Chapter 9 This chapter conclude what have been done in thesis and what can be done in future.

CHAPTER 2

ADDER ARCHITECTURES

This chapter introduces the basic principles, architectures, optimal composition of speed-up schemes and circuit simplifications that exist for binary addition.

2.1 Carry Propagate Adder

A carry-propagate adder adds two k-bit operands, X = (xk-1, xk-2 . . , x0) and Y = (yk-1,yk-2…,y0) and an optional carry-in by performing carry propagation and the result is an irredundant (k+1)-bit number consisting of the k-bit sum S= (sk-1, sk-2, … , s0) and carry-out

2.1.1 Ripple Carry Adder

This is the simplest design in which the carry-out of one bit is simply connected as the carrying to the next [1]. It can be implemented as a combination circuit using n full-adder in series as shown in Figure 2.2 and is called ripple-carry adder.

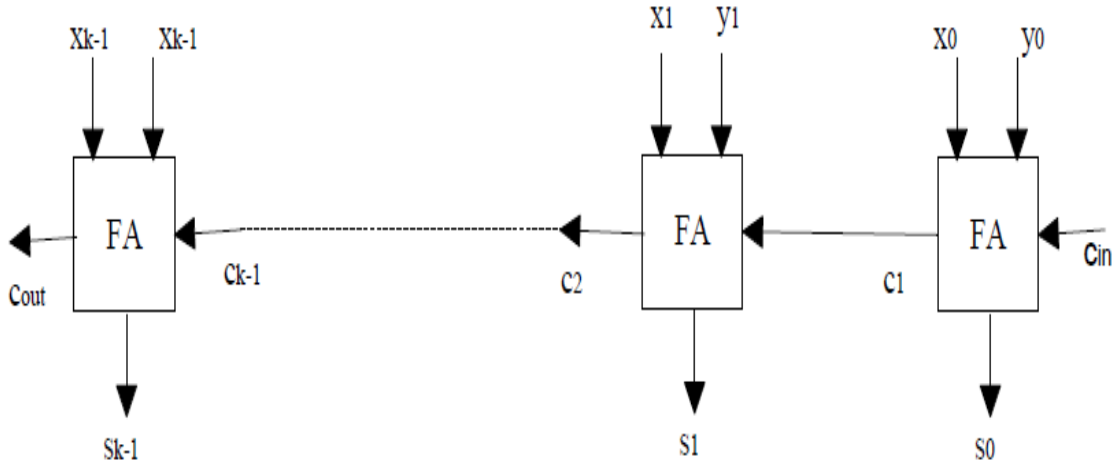
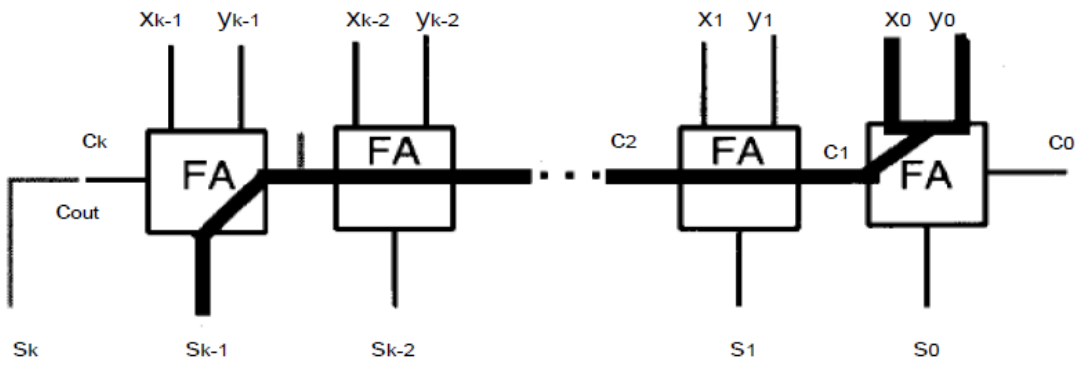


Figure 2.1: Ripple carry implementation of CPA[2]

The latency of k-bit ripple-carry adder can be derived by considering the worst-case signal propagation path. As shown in Figure 2.3 the critical path usually begins at the x_0 or y_0 input proceeds through the carry-propagation chain to the leftmost FA and terminates at the s_{k-1} output.



F

igure 2.2: Critical paths in a k-bit RCA [2]

The critical path might begin at c_0 and/or terminate at c_k. However given that the delay from carry-in to carry-out is more important than from x to c_{out} or from c_{in} to s, full-adder designs often minimize the delay from C_{in} to C_{out}, making the path as shown in Figure 2.3 with the largest delay.

2.1.2 Carry-look ahead Adder

The carry-look ahead adder [3] computes group generate signals as well as group propagate signals to avoid waiting for a ripple to determine if the first group generates a carry or not. From the point of view of carry propagation and the design of a carry network the actual operand digits are not important. What matters is whether in a given position a carry is generated, propagated or annihilated. In the case of binary addition the generate, propagate and annihilate signals [2][4][5] are characterized by the following logic equations:

$$g_i = x_i \,.\, y_i \quad (2.2)$$

$$p_i = x_i \oplus y_i \quad (2.3)$$

$$a_i = not(x_i + y_i) \quad (2.4)$$

$$t_i = x_i + y_i \quad (2.5)$$

Thus assuming that the above signals are produced and made available, the rest of the carry network design can be based on them and become completely independent of the operands or even the number representation radix. Using the preceding signals, the carry recurrence can be written as follows

$$c_{i+1} = g_i + c_i \,.\, p_i \quad (2.6)$$

The carry recurrence essentially states that a carry will enter stage i+1 if it is generated in stage i. The later version of carry recurrence leads to slightly faster adders because in binary addition, t_i is easier to produce than p_i. (OR instead of XOR).

$$c_{i+1} = g_i + c_i \,.\, t_i \quad (2.7)$$

The carry recurrence forms the basis of simple carry network known as Manchester carry chain. A Manchester adder is one that uses a Manchester carry chain [2] as its carry network. Each stage of a Manchester carry chain consists of three switches controlled by the signals p_i, g_i , and a_i, so that the switch closes (conducts electricity) when the corresponding control signal is 1.

As shown in Figure 2.4, the carry-out signal c_{i+1} is connected to 0 if $a_i = 1$, and to 1 if $g_i = 1$, and to c_i if $p_i = 1$, thus assuming the correct logical value from equation 2.6. Note that one, and only one, of the signals p_i, g_i, and a_i is 1.

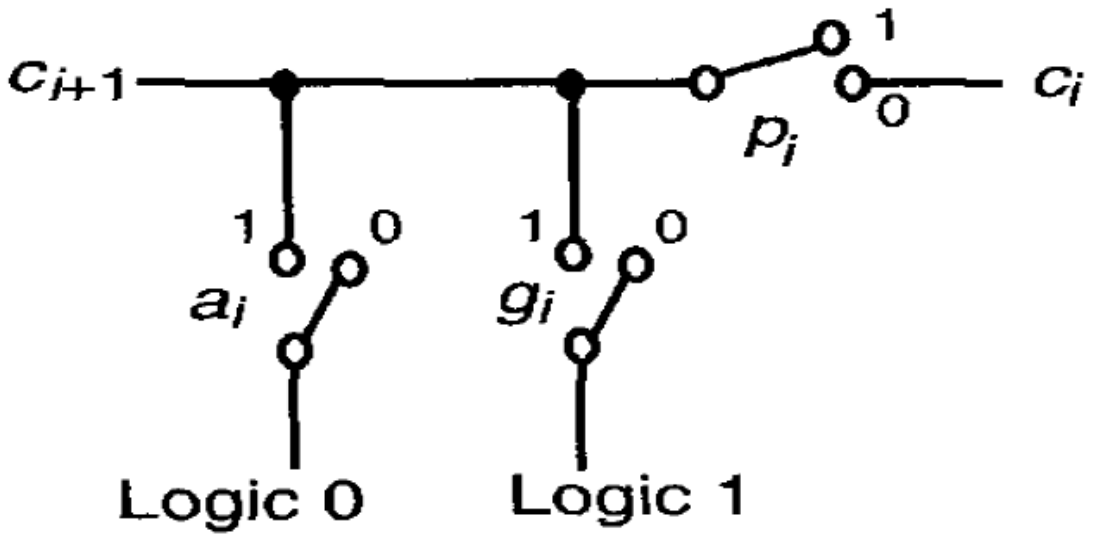


Figure 2.3: One stage Manchester carry chain [2]

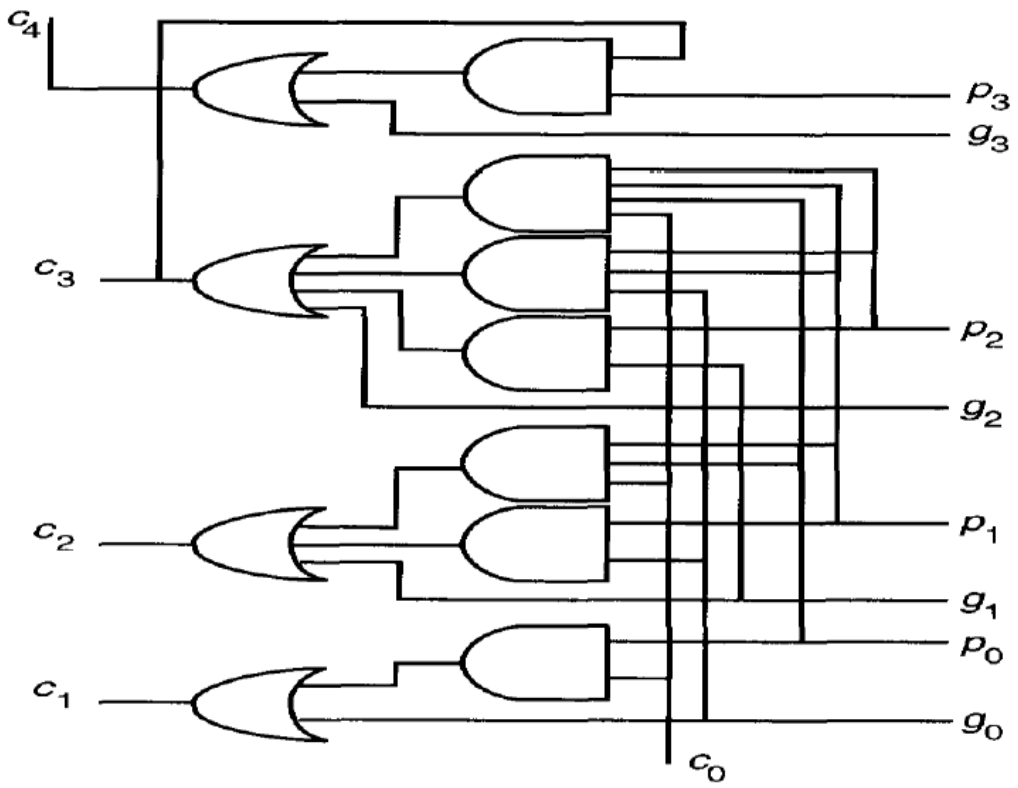


Figure 2.4: Carry-look ahead logic [2]

Carry-look ahead adder hardware may be designed as shown in Figure 2.5. The carry-look ahead logic consists of two logic levels, AND gates followed by an OR gate, for each C_i when the adder inputs are loaded in parallel, all g_i and p_i will be generated at the same time. The carry-look ahead logic allows carry for each bit to be computed independently.

Ideally, the carry signal C_i will be produced through two-stage logic at the same time, which means that the adder will have a constant time complexity. However, it is impractical to build a two stage full large-size carry-look ahead adder because of the practical limitations on fan-in and fan-out, irregular structure, and long wires delay [1]. In practice two approaches the block carry-look ahead adder and the complete carry-look ahead adder are used to implement the CLA[1]. In the first implementation, small (4-bit or 8-bit) carry-look ahead logic cells with sections generate and propagate functions are built, and then they are stacked to build larger carry-look ahead adders. In complete carry-look ahead logic, the adder is built for the given operand size but in a way that allow the use of parallel prefix circuits. One well-known adder of this type is the Brent-KungIn practice two approaches the block carry-look ahead adder and the complete carry-look ahead adder are used to implement the CLA[1]. In complete carry-look

ahead logic, the adder is built for the given operand size but in a way that allow the use of parallel prefix circuits. One well-known adder of this type is the Brent-Kung adder.

The total delay of the carry-look ahead adder is O (log k) which can be significantly less than the carry chain adder. There is a penalty paid for this gain in term increased area.

The carry- look ahead adders require O (k * log k) area. It seems that a carry-look ahead adder larger than 256 bits is not cost effective. Even by employing block carry-look ahead approach, a carry-look ahead adder with 1024 bits seems not feasible or cost effective [1].

2.1.3 Carry Skip Adder

The carry skip adder [2] was invented for decimal arithmetic operations. The CSKA is an improvement over the ripple-carry adder. By grouping the ripple cells together into blocks, it makes the carry signal available to the blocks further down the carry chain, earlier. The primary carry C_i coming into a block can go out of it unchanged if and only if, X_i and Y_i are exclusive-or of each other. This means that corresponding bits of both operands within a block should be dissimilar.

If $X_i = Y_i = 1$, then the block generates a carry without waiting for the incoming carry signal. And the generated carry will be used by blocks beyond this block in the carry chain. If $X_i = Y_i = 0$, then the block does not generate a carry and will absorb any carry coming into it by AND all p_i of a block.

The skip signal will be generated to select between the incoming carry and the generated carry using a 2:1 multiplexer as shown in Figure 2.6. A more simplified skip logic that requires less area [12] is shown in Figure 2.7. If the adder input is assumed to be loaded in parallel, then the skip signal of all blocks will be ready at about the same time.

The last FA stage of a block will generate a carry, if any, before arrival of the input carry c_i. When the input carry arrives, it needs to pass through two logic gates only so that the output carry c_{i+1} will stabilize.

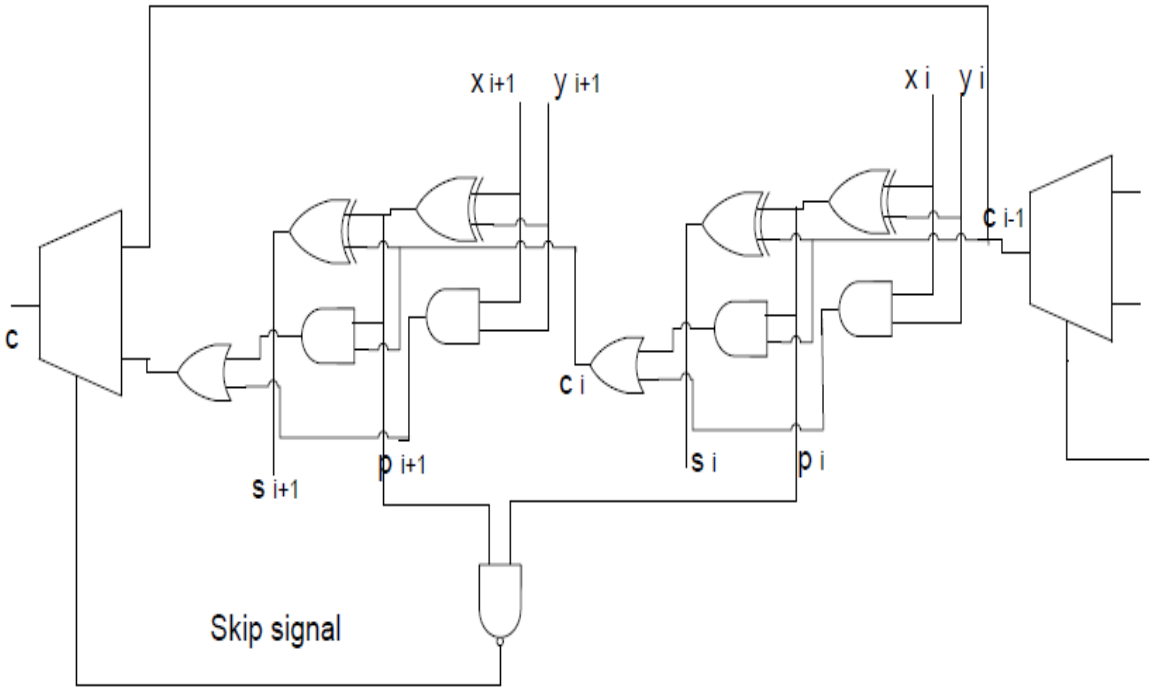


Figure 2.5: Carry-skip logic using multiplexers [7]

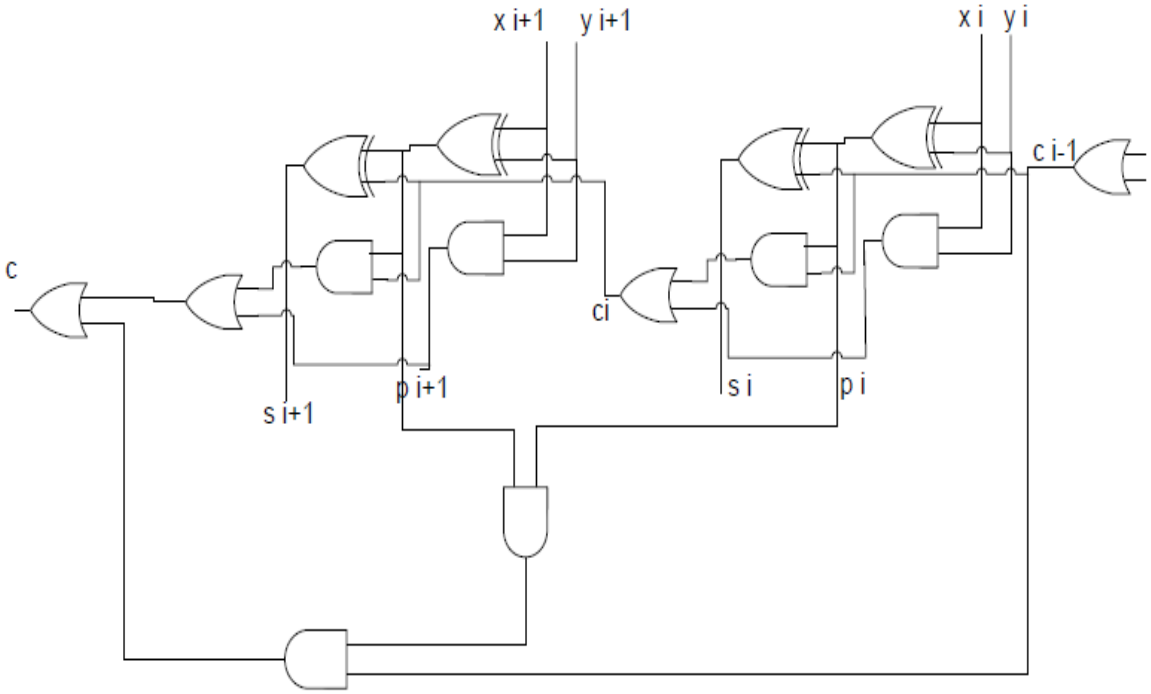


Figure 2.6: Carry-skip logic using gates [7]

Subsequent blocks can have larger size so that the carry will skip more bits and the adder speed will be increased. In this case, the adder is called one-level carry-skip adder with variable block

sizes. The adder speed can be improved even more by using a multilevel skip structure [6]. The skip logic determines whether a carry entering one block may skip the next group of blocks. However, the main design problem with the adder is working out how best to group the skips.

There exist many proposals for optimum design of carry-skip adders. Based on some assumptions and some input variables in addition to the desired size, the proposed algorithms [9] decide on the optimum size of each block and some times the number of skip levels.

2.1.4 Carry Select Adder

The carry-select adder comes in the category of conditional sum adder [6]. Conditional sum adder works on some condition this scheme; blocks of bits are added in two ways: assuming an incoming carry of 0 or 1, with the correct outputs selected later as the block's true carry-in becomes known. With each level of selection, the number of known output bits doubles, and leading to a logarithmic number of levels and thus logarithmic time addition.

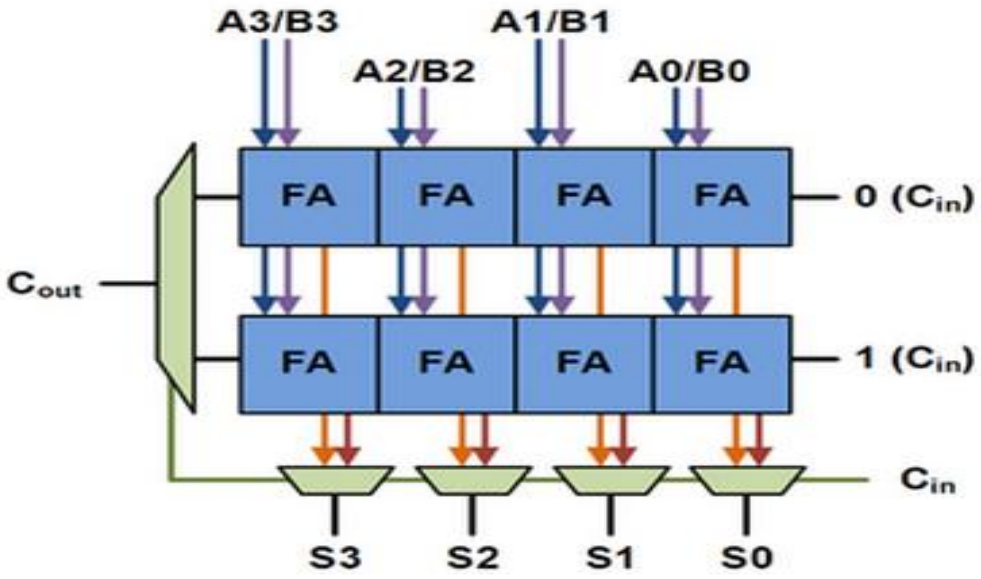


Figure 2.7: Basic Building Block of CSLA[18]

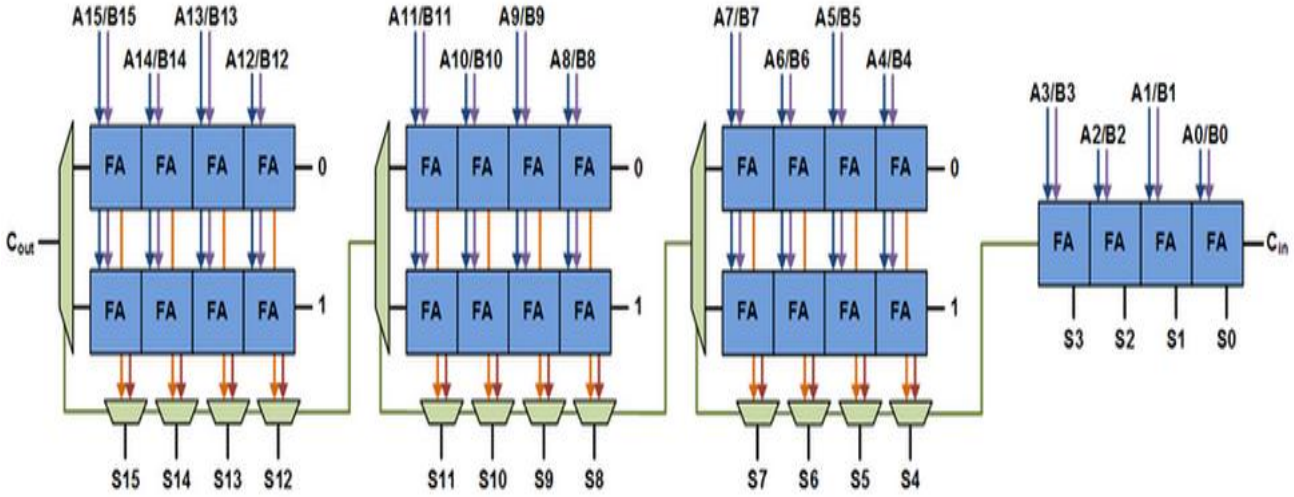


Figure 2.8: Uniform-Sized CSLA Adder[18]

A 16-bit carry-select adder with a uniform block size of 4 can be created with three of these blocks and a 4-bit ripple carry adder. Since carry-in is known at the beginning of computation, a carry select block is not needed for the first four bits. The delay of this adder [35] will be four full adder delays, plus three MUX delays.

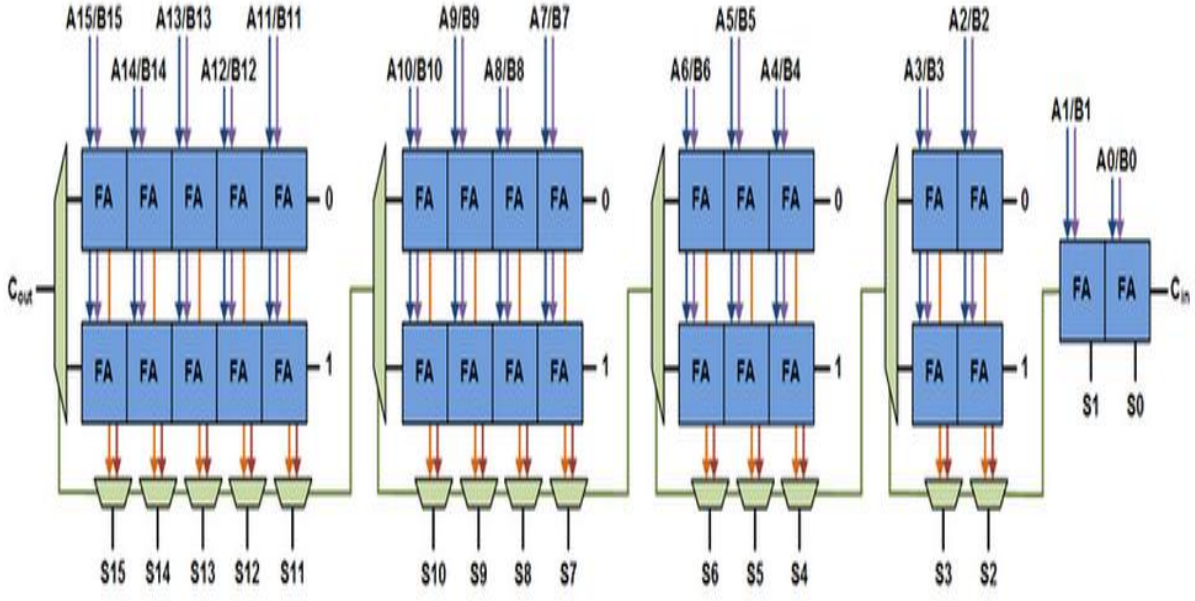


Figure 2.9: Variable-Sized CSLA Adder[18]

A 16-bit carry-select adder with variable size can be similarly created. Here we show an adder with block sizes of 2-2-3-4-5. This break-up is ideal when the full-adder delay is equal to the MUX delay, which is unlikely. The total delay is two full adder delays, and four mux delays [35].

A single-level carry-select adder [2] is one that combines three k l/2-bit adders of any design into a k-bit adder as shown in Figure 2.9. One k1/2-bit adder is used to compute the lower half of the k-bit sum directly. Two k1/2-bit adders are used to compute the upper k1/2-bits of the sum and the carry-out under two different scenarios: Ck/2 = 0 and Ck/2 = 1. The correct values for the adder's carry-out signal and the sum bits in positions k l/2 through k - 1 are selected when the value of Ck/2 becomes known. This technique of dividing adder in two stages increases the area utilization but addition operation fastens [9].

2.2 Multi-Operand Adder

Multi-operand adders are used for the summation of n k-bit operands (X0, X1… Xn-1) (n>2) yielding a result S in irredundant number representation with (k+ [log2 n]) bits [11]

$$S= \sum_{j=0}^{j-1} X$$

2.2.1 Carry Save Adder

The carry-save adder avoids carry propagation by treating the intermediate carries as outputs instead of advancing them to the next higher bit position, thus saving the carries for later propagation. A carry-save adder adds three k-bit numbers and produces the result without performing carry propagation by saving the carry.

Figure 2.10: Block diagram of CSA[2]

A carry-save adder design is shown in Figure 2.10. The result will be in an k-bit redundant format represented using two bit vectors, sum (S) and carry (C). The total delay of a carrysave adder equals to the delay of a single full-adder cell. In addition, the carry- save adder requires n

times the area of a full-adder cell. Thus, the carry-save adder takes O (1) time and O (n) space [12].

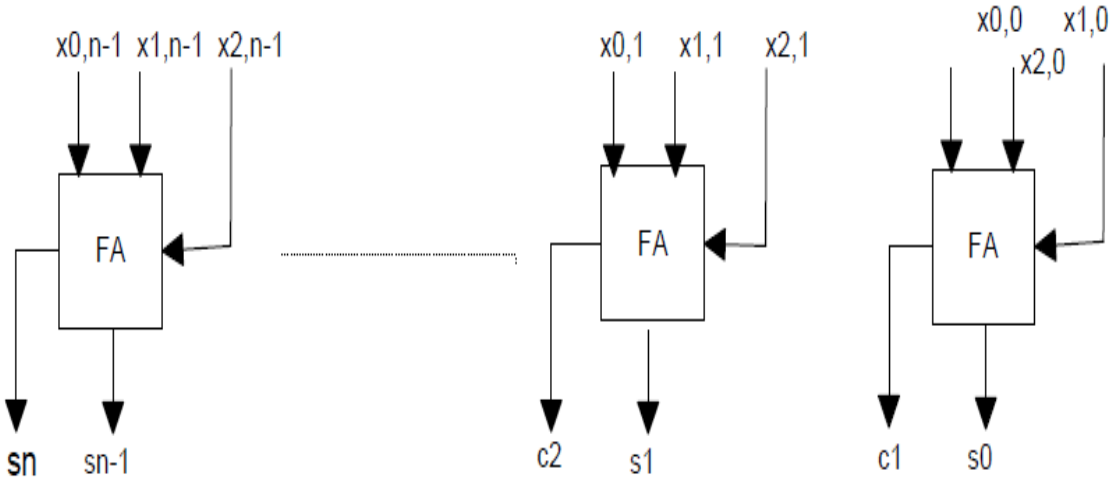


Figure 2.11: k-bit carry-save adders to add three k bit numbers[2]

A row of binary full-adders as a mechanism to reduce three numbers to two numbers rather than as one to reduce two numbers to their sum has been viewed. Figure 2.11 shows how n carry- save adders are arranged to add three k-bit numbers x, y and z. Figure 2.12 presents carry-save adder in dot notation. To specify more precisely how the various dots are related or obtained, enclose any three dots that form the inputs to a full-adder in a dashed box and to connect the sum and carry outputs of a full-adder by a diagonal line as shown in Figure 2.13. Occasionally, only two dots are combined to form a sum-bit and a carry-bit.

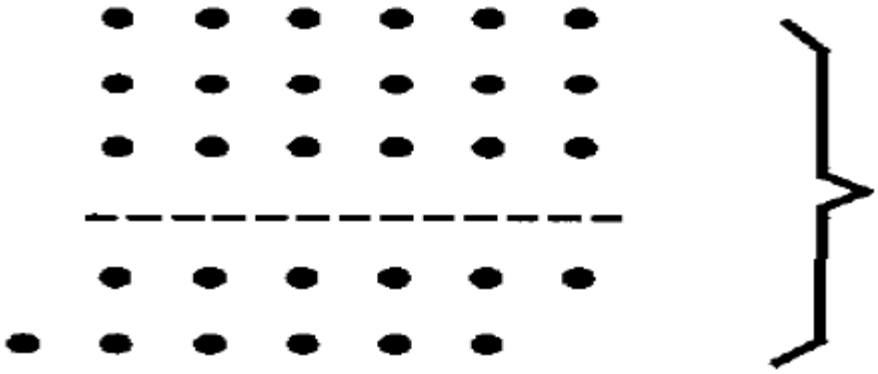


Figure 2.12: CSA function in dot notation [2]

Then the two dots are enclosed in a dashed box and the use of a half-adder is signified by a cross line on the diagonal line connecting its outputs shown in Figure 2.13.

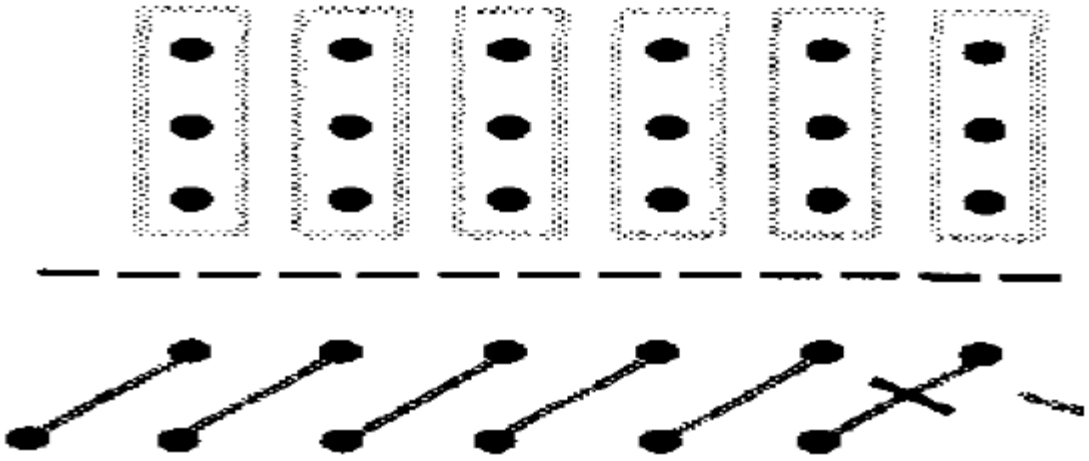


Figure 2.13: Specifying full-adder and half-adder blocks in dot notation [2]

The needed CSAs are of various widths, but generally the widths are close to k-bits; the CPA is of width at most (k +log$_2$ n).

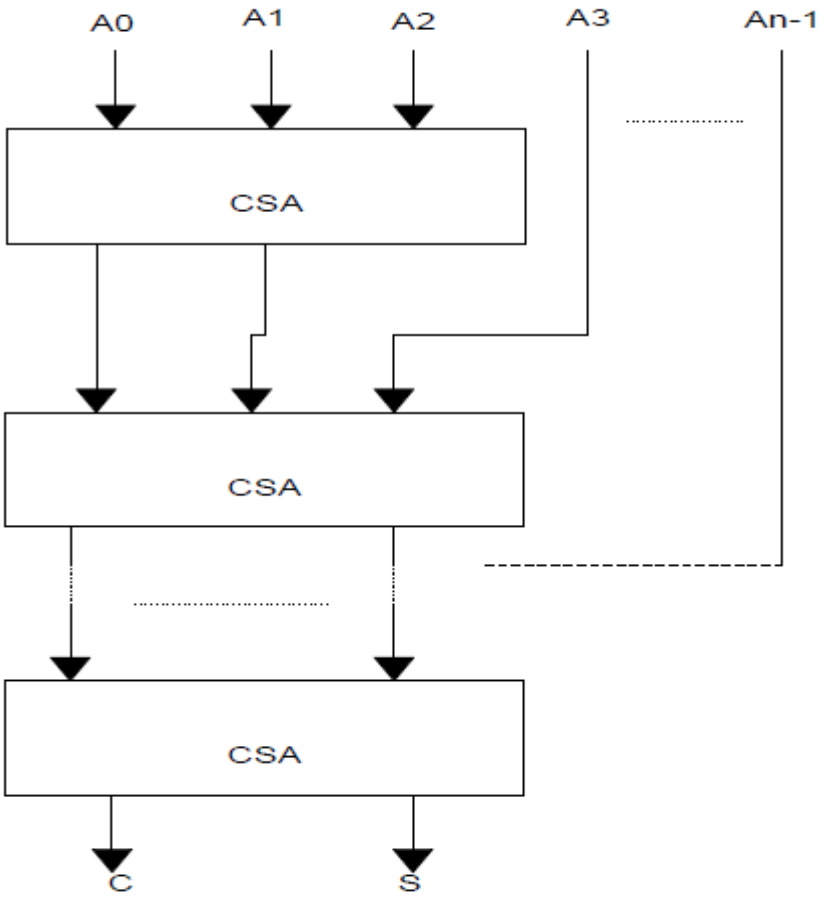


Figure 2.14: Adding n k-bit numbers using CSA[2]

There are basically two disadvantages of the carry-save adders [1].

- The carry-save adders do not add two numbers and produce a single output; instead, they add three inputs and produce two outputs such that the sum of the outputs equals to the sum of the inputs.
- The sign-detection is complex. Unless the addition on the outputs is performed in full length, the correct sign of the sum-carry pair may never be determined.

CHAPTER 3

MULTIPLIERS

Multiplication is one of the basic functions used in digital signal processing (DSP). It requires more hardware resources and processing time than addition and subtraction. In fact, 8.72% of all instructions in a typical processing unit is multiplier. In computers, a typical central processing unit devotes a considerable amount of processing time in implementing arithmetic operations, particularly multiplication operations. Most high performance digital signal processing systems rely on hardware multiplication to achieve high data throughput. Multiplication is an important fundamental arithmetic operation. Multiplication-based operations such as Multiply and Accumulate (MAC) are currently implemented in many Digital Signal Processing (DSP) applications such as convolution, Fast Fourier Transform (FFT), filtering and in microprocessors in its arithmetic and logic unit . Since multiplication dominate the execution time of most DSP algorithms, so there is a need of high speed multiplier. Currently, multiplication time is still that dominant factor in determining the instruction cycle time of a DSP chip. The multiplier is a fairly large block of a computing system.

The speed of the multipliers is greatly improved by properly deciding the number of pipeline stages and the positions for the pipeline registers to be inserted. Pipelines are widely used to improve the performance of digital circuits, since they provide a simple way of implementing parallelism from streams of sequential operations. In a pipelining system, the maximum operating frequency is limited by slowest stage which has the longest delay. A more stages are inserted in the pipeline, each stage becomes shorter, and ideally presents a smaller delay.

3.1Objective

The main objective of this thesis is design and implementation of a Booth Wallace multiplier. The programming objectives of Booth Wallace multiplier fall into following categories:

- Accuracy: The multiplier produces the correct result.
- Speed: The multiplier produces high speed.
- Area: The multiplier occupies less number of slices and LUTs.
- Power: The multiplier consumes less power.

Multiplication involves three main steps [18]:

- Partial product generation
- Partial product reduction
- Final addition

For the multiplication of an *n*-bit multiplicand with an *m*-bit multiplier, *m* partial products are generated and product formed is $n + m$ bits long.

The multiplier architectures can be generally classified into following categories:

- Serial multiplier
- Parallel multiplier
- Serial-parallel

3.2 Serial Multipliers

The simplest method to perform multiplication is to add series of partial products. The serial multipliers use a successive addition algorithm. They are simple in structure because both the operands are entered in a serial manner. Therefore, the physical circuit requires less hardware

and a minimum amount of chip area. However, the speed performance of the serial multiplier is due to the operands entered sequentially.

3.3 Parallel Multipliers

Most advanced digital systems incorporate a parallel multiplication unit to carry out high speed mathematical operations. A microprocessor requires multipliers in its arithmetic logic unit and a digital signal processing system requires multipliers to implement algorithms such as convolution and filtering. Some examples of parallel multipliers are array multipliers such as Braun multipliers, Booth multipliers and Baugh-Wooley multipliers, as well as the tree multipliers like Wallace multipliers. Array multipliers have a regular layout, although tree multipliers are generally faster. Parallel multipliers [10] present high-speed performance, but are expensive in terms of silicon area and power consumption because in parallel multipliers both the operands are input to the multiplier in parallel manner.

3.3.1 Array Multipliers

Array multiplier can be classified into following categories:

- Braun Multiplier
- Booth Multiplier
- Modified Booth Multiplier
- Baugh -Wooley Multiplier

3.3.1.1 Braun Array Multiplier

Braun Array multiplier is well known due to its regular structure. It is a simple parallel multiplier that is commonly known as carry save array multiplier. This multiplier is restricted to performing multiplication of two unsigned numbers. It consist of array of AND gates and adders arranged in iterative structure that does not require logic registers.

This is also known as the non-additive multiplier since it does not add an additional operand to result of multiplication [10]. A four-bit Braun multiplier is shown in Figure 2.1. To perform N-bit by N-bit multiplication, the N-bit multiplicand A is multiplied by N-bit Multiplier B to produce product. The unsigned binary numbers A and B can be expressed as:

$$A = \sum_{i=0}^{n-1} A_i \cdot 2^i \qquad \textbf{(3.1)}$$

$$B = \sum_{j=0}^{n-1} B_j \cdot 2^j \qquad \textbf{(3.2)}$$

The product of A and B is P and it can be written in the following form:

$$P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A_i \cdot B_j 2^{i+j} \qquad \textbf{(3.3)}$$

A bit multiplier requires adders and AND gates. In this array multiplier, counters are connected in a serial fashion for all bit slices of partial product.

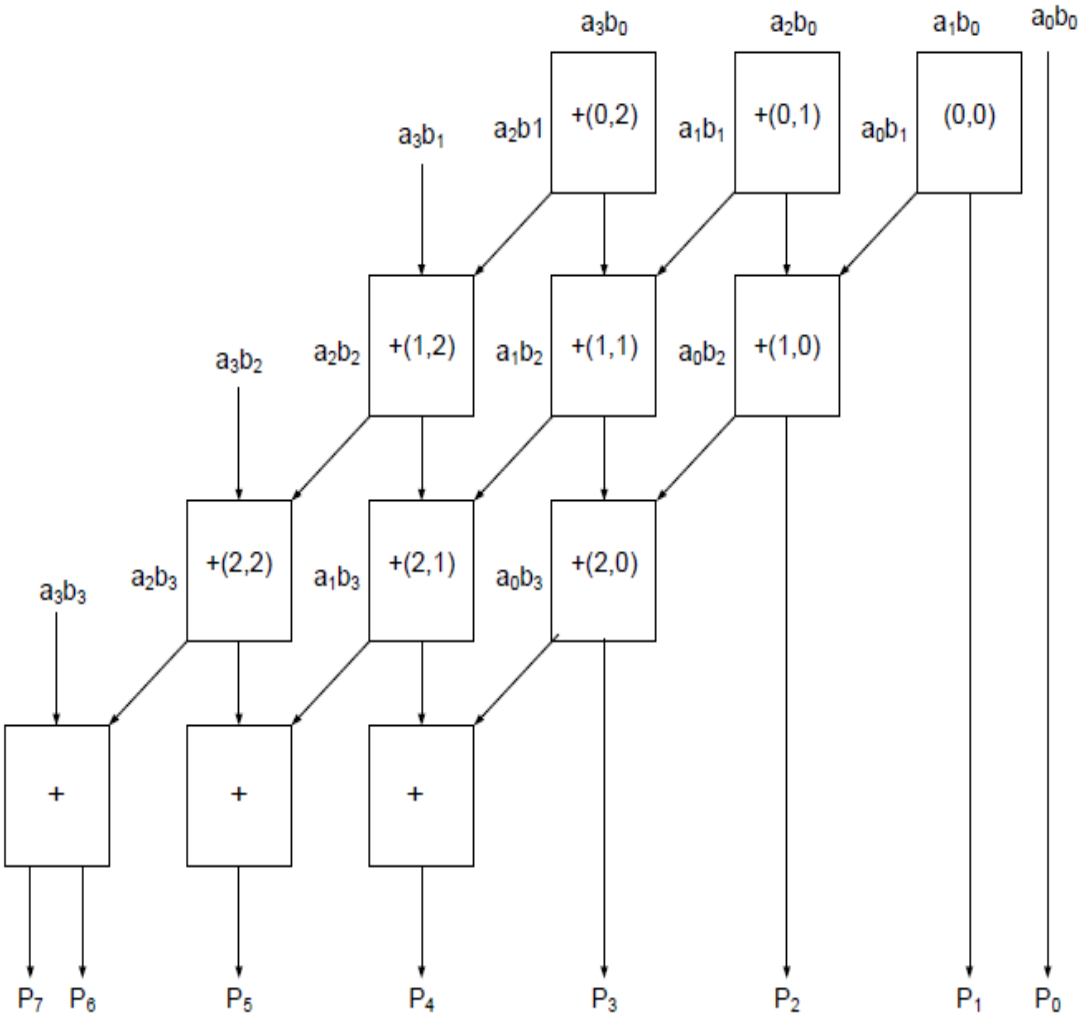


Figure 3.1: Braun multiplier **[8]**

Each block is of the form of the below structure:

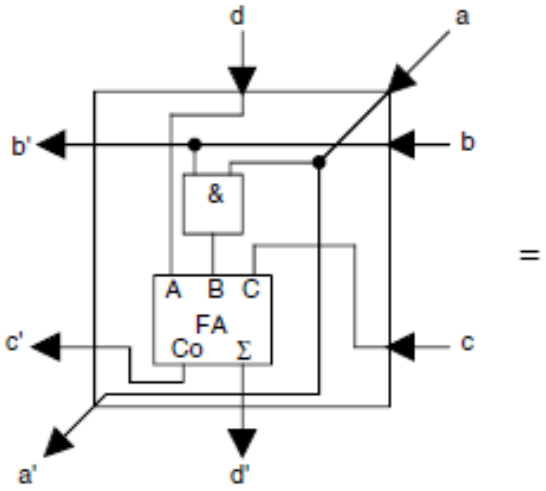


Figure 3.2: Basic Structure Of A Block.[8]

3.3.1.2 Advantages of Braun Array Multiplier

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. The design time of an array multiplier is much less than that of a tree multiplier. A second advantage of the array multiplier is its ease of design for a pipelined architecture.

3.3.1.3 Limitations of Braun Array Multiplier

Major limitation of Braun array multiplier is its size. As operand sizes increase, arrays grow in size at a rate equal to the square of the operand size. The large size of full arrays typically prohibits their use, except for small operand sizes, or on special purpose math chips where a major portion of the silicon area can be assigned to the multiplier array.

Another problem with array multipliers is that the hardware is underutilized [8].

3.4 Booth Multiplier

Booth's algorithm examines adjacent pairs of bits of the N-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit y_i, for i running from 0 to N-1, the bits y_i and y_{i-1} are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2^i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2^i is subtracted from P. The final value of P is the signed product. The representation of the

multiplicand and product are not specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of the steps is not determined. Typically, it proceeds from LSB to MSB, starting at $i = 0$; the multiplication by 2^i is then typically replaced by incremental shifting of the *P* accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest *N* bits of *P*. There are many variations and optimizations on these details.

The algorithm is often described as converting strings of 1's in the multiplier to a high-order +1 and a low-order –1 at the ends of the string. When a string runs through the MSB, there is no high-order +1, and the net effect is interpretation as a negative of the appropriate value.

3.4.1Implementation

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values *A* and *S* to a product *P*, then performing a rightward arithmetic shift on *P*. Let **m** and **r** be the multiplicand and multiplier, respectively; and let *x* and *y* represent the number of bits in **m** and **r [19]**.

1. Determine the values of *A* and *S*, and the initial value of *P*. All of these numbers should have a length equal to (*x* + *y* + 1).
    1. A: Fill the most significant (leftmost) bits with the value of **m**. Fill the remaining (*y* + 1) bits with zeros.
    2. S: Fill the most significant bits with the value of (−**m**) in two's complement notation. Fill the remaining (*y* + 1) bits with zeros.
    3. P: Fill the most significant *x* bits with zeros. To the right of this, append the value of **r**. Fill the least significant (rightmost) bit with a zero.
2. Determine the two least significant (rightmost) bits of *P*.
    1. If they are 01, find the value of *P* + *A*. Ignore any overflow.
    2. If they are 10, find the value of *P* + *S*. Ignore any overflow.
    3. If they are 00, do nothing. Use *P* directly in the next step.
    4. If they are 11, do nothing. Use *P* directly in the next step.
3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let *P* now equal this new value.

4. Repeat steps 2 and 3 until they have been done *y* times.

5. Drop the least significant (rightmost) bit from *P*. This is the product of **m** and **r**.

3.5 Modified Booth Algorithm

The modified Booth encoding (MBE), or modified Booth's algorithm (MBA), was proposed by O. L. Macsorley in 1961 [11]. The recoding method is widely used to generate the partial products for implementation of large parallel multipliers, which adopts the parallel encoding scheme. One of the solutions of realizing high-speed multipliers is to enhance parallelism, which helps to decrease the number of subsequent stages. The original version of Booth algorithm (Radix-2) had two drawbacks:

- The number of add subtract operations and the number of shift operation becomes variable and becomes inconvenient in designing parallel multipliers.
- The algorithm becomes inefficient when there are isolated 1"s.

These problems can be overcome by modified Booth algorithm. MBA process three bits at a time during recoding. Recoding the multiplier in higher radix is a powerful way to speed up standard Booth multiplication algorithm. In each cycle a greater number of bits can be inspected and eliminated therefore, total number of cycles required to obtain products get reduced. Number of bits inspected in radix r is given by $n = 1 + \log_2 r$. Algorithm for modified booth is given below [12]:

Consider two n-bit numbers X and Y to be multiplied where Y can be expressed as:

$$Y = -Y_{n-1}2^{n-1} + Y_{n-2}2^{n-2} + \cdots + Y_0 2^0 \qquad \textbf{(3.4)}$$

$$\begin{aligned} Y = {} & (-2Y_{n-1} + Y_{n-2} + Y_{n-3})2^{n-2} \\ & + (-2Y_{i-3} + Y_{i-4} + Y_{i-5})2^{i-4} \\ & + \cdots + (-2Y_1 + Y_0 + Y_{-1})2^0 \end{aligned} \qquad \textbf{(3.5)}$$

$$Y = \sum_{i=0}^{i=\frac{n}{2}-1} (-2Y_{2i+1} + Y_{2i} + Y_{2i-1}) . 2^{2i} = \sum_{i=0}^{i=\frac{n}{2}-1} Y_i . 2^i \quad \textbf{(3.6)}$$

$$X.Y = \left(-X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i . 2^i\right)\left(-Y_{i-1}2^{n-1} + \sum_{j=0}^{n-2} Y_j . 2^j\right) \quad \textbf{(3.7)}$$

$$X.Y = \left(-X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i . 2^i\right)\left(\sum_{j=0}^{\frac{n}{2}-1} Y_j . 2^{2j}\right) \quad \textbf{(3.8)}$$

In each cycle of radix-4 algorithm, 3 bits are inspected and two are eliminated. Procedure for implementing radix-4 algorithm is as follows

- Append a 0 to the right of LSB.
- Extend the sign bit 1 position if necessary to ensure that n is even.
- According to the value of each vector, find each partial product

Table 3.1

Modified Booth Algorithm [21]

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Recoded Digit	Operand Multiplication
0	**0**	**0**	**0**	0*Multiplicand
0	**0**	**1**	**+1**	+1*Multiplicand
0	**1**	**0**	**+1**	+1*Multiplicand
0	**1**	**1**	**+2**	+2*Multiplicand
1	**0**	**0**	**-2**	-2* Multiplicand
1	**0**	**1**	**-1**	-1*Multiplicand
1	**1**	**0**	**-1**	-1*Multiplicand
1	**1**	**1**	**0**	0*Multiplicand

3.5.1 Modified Booth Example

Assume two numbers to be multiplied are A= 34 and B = -42.

Multiplicand A= 34 = 00100010

Multiplicand B = -42 = 11010110(2"s Complement)

A X B = -1428

Table 3.2

Modified Booth Example

									0	0	1	0	0	0	1	0		34
									1	1	0	1	0	1	1	0		-42
						1	1	1	1	0	1	1	1	1	0	0		PP1
				1	1	0	0	1	0	0	0	1	0	0				PP2
		1	1	0	0	0	1	0	0	0	1	0						PP3
1	0	1	1	1	0	1	1	1	1	0								PP4
1	1	1	1	1	1	0	1	0	0	1	1	0	0					-1428

Table 2.4 shows Modified Booth example. PP1, PP2, PP3, PP4 are the partial products formed. The first partial product is determined by three digits LSB of multiplier with a appended zero. This 3 digit number is 100 which mean the multiplicand A has to multiply by -2. To multiply by -2, the process takes two's complement of the multiplicand value and then M shift left one bit of that product. Hence, the first partial product is 110111100. All of the partial products will have nine bits length. Next, the second partial product is determined by next three bits i.e. multiply by 2. Multiply by 2 means the multiplicand value has to shift left one bit. So, the second partial product is 001000100. Similarly the third partial product have to multiply by 1. So, the third partial product is the multiplicand value namely 000100010. The fourth partial product is determined by next three bits i.e. to multiply by -1. Multiply by -1 means the multiplicand has to convert to two's complement value. So, the forth partial product is 111011110.

LSB of each block gives information about sign bit of the pervious block, and there are never any negative products before the least significant block, so LSB of first block is always taken to be zero. Block diagram of an n x n – bit modified Booth multiplier is shown in Figure 2.2.

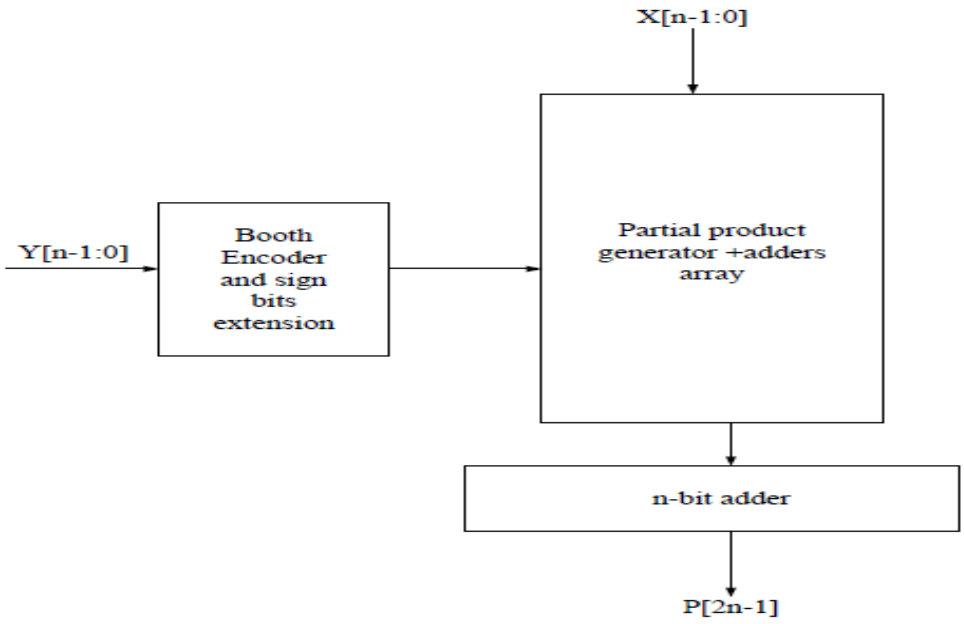


Figure 3.3: Block diagram of a n x n modified Booth multiplier[13]

It consists of the Booth encoder and the sign extension bits, the multiplier array, which comprises the partial product's generator and 1-bit adders, and the final stage adder, which executes the 2 -bit addition.

3.6 Wallace Tree Multiplier

The number of adders will be minimized by Wallace Tree.

3.6.1 4:2 Compressor:

It has 4 input lines X0, X1, X2, X3 that must be summed and has two output lines C and S which are so called result of compression. The additional lines are input and output carries as shown in Figure 3.4. A 4:2 compressor can be implemented with two stages of full adder (FA) connected in series as shown in Figure.3.5.

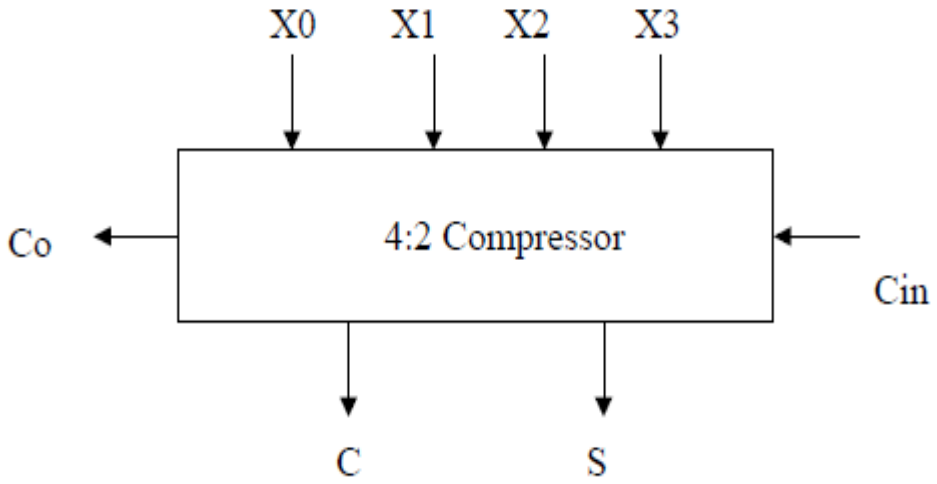


Figure 3.4: Block diagram of 4:2 Compressor[21]

Indeed a 4:2 structure is not a counter, since two output bits cannot represent five possible sums of four bits. Thus a carry out is necessary and subsequently carry in.

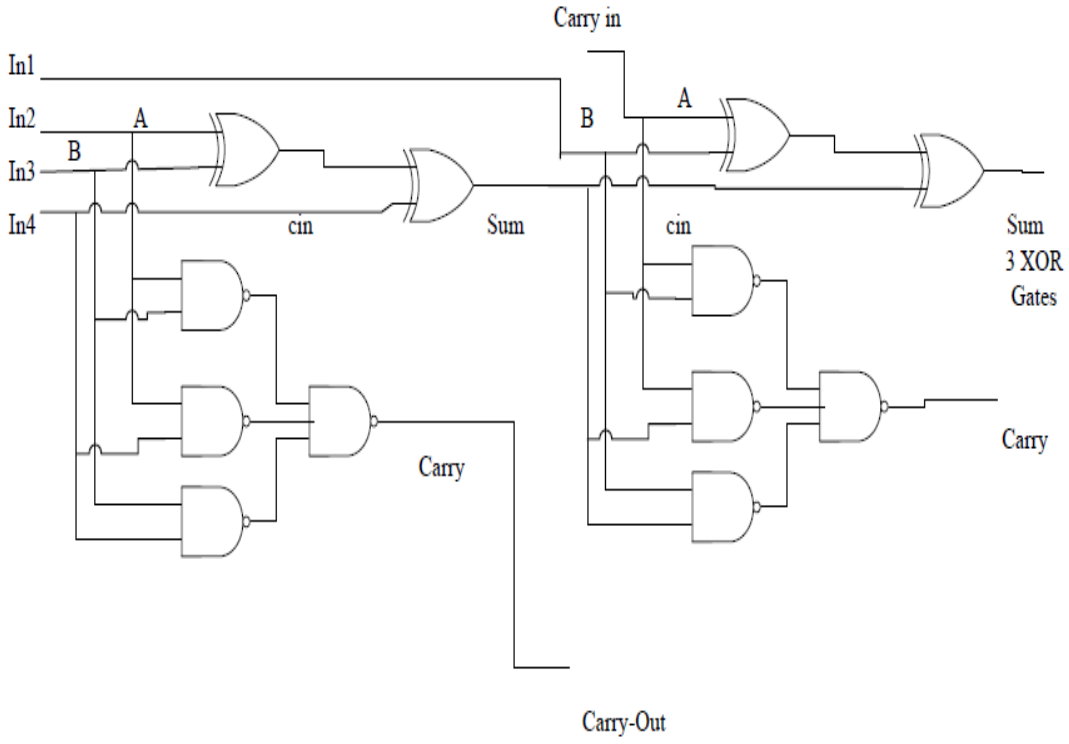


Figure 3.5: A 4:2 compressor logic diagram [21]

The 4:2 compressor structure actually compress five input bit into three output. The output of 4:2 compressor consists of one bit (sum) in position j and two bits in final carry and intermediate carry in position (j+1). However a carry out is independent on carry in as shown in figure.

3.6.2. Wallace Tree Implementation:

Wallace Tree is a reduction technique that uses the carry save adder to add the partial products. A block diagram for the data distribution among a tree architecture that employs 4:2 compressors is shown in Figure 3.6.

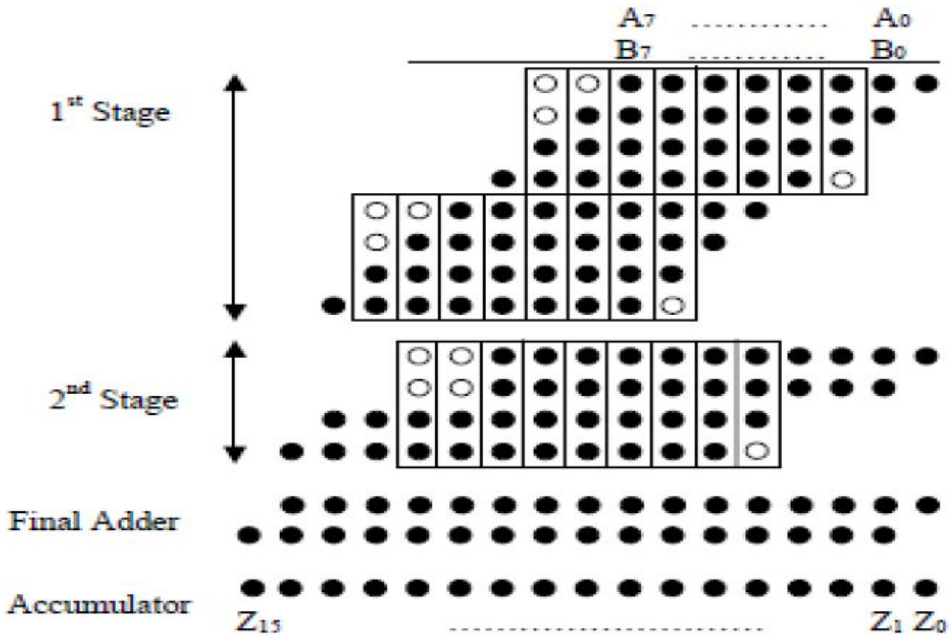


Figure 3.6: Data distribution among a tree architecture [20]

Each box contains the bits that are fed into a 4:2 compressor cell. Two stages of 4:2 compressors are used to reduce the number of partial products by a ratio of 2:1. This figure presents the reduction tree of 8 partial products to form two operands, which are then added together to form a final product by using a fast carry propagate adder. Using 4:2 compressor, there is a simple and more regular wiring of multiplier tree .

CHAPTER 4

INTRODUCTION TO PIPELINING

This chapter introduces the basics of binary multiplication, partial product generation, reduction and techniques to make the multiplication process faster.

4.1 Basic Architecture of MAC Unit

The multiplication and accumulates is the main computational kernel in Digital Signal Processing architectures. The MAC unit determines the speed of overall system as it is always lies in the critical path. Developing high speed MAC is crucial for real time DSP application. In order to improve the speed of the MAC unit, there are two major bottlenecks that need to be considered. The first one is the fast multiplication network and the second one is the accumulator. Both of these stages require addition of large operands that involve long paths for carry propagation.

The MAC unit basically do the multiplication of two numbers multiplier and multiplicand and add that product in result stored in the accumulator. The general construction of the MAC operation can be represented by this equation:

$$Z = A * B + Z \quad (4.1)$$

Where the multiplier A and multiplicand B are assumed to have n bits each and the addend Z has (2n+1) bits. A basic MAC unit can be divided into two main blocks [32].

- Multiplier
- Accumulator

4.1.1 Multiplier

A Fast Multiplication process consists of three steps [33]:

- Partial Product Generation.
- Partial Product Reduction.
- Final stage Carry Propagate Adder.

We have already discussed about this in the chapter -3.

4.1.2 Accumulator

Accumulator basically consists of register and adder. Register hold the output of previous clock from adder. Holding outputs in accumulation register can reduce additional add instruction. An accumulator should be fast in response so it can be implemented with one of fastest adder like Carry-look Ahead Adder or Carry Skip Adder or Carry Select Adder .

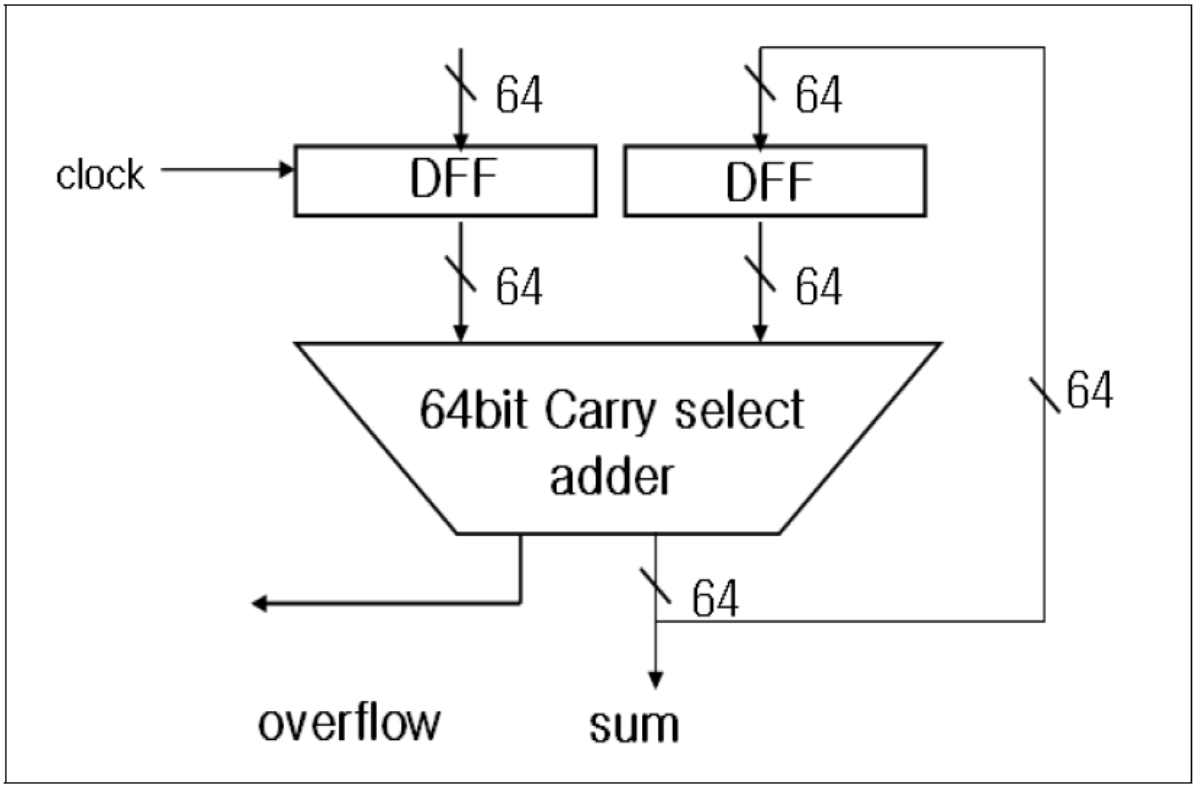


Figure. 4.1. Block diagram of an accumulator [22]

4.2 Pipelining

Pipelining is a technique of decomposing a sequential process into sub operations , with each sub process being executed in a special dedicated segment that operates concurrently with all other segments. A pipeline can be visualized as a collection of processing segments through which binary information flows . Each segment performs partial processing dictated by the way the task is partitioned . The result obtained from the computation in each segment is transferred to the next segment in the pipeline. Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational

circuit. The register holds the data and the combinational circuit performs the suboperation in the particular segment. The output of the combinational circuit in a given segment is applied to the input register of the next segment . A clock is applied to all registers after enough time has elapsed to perform all segment activity. Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

$$A_i * B_i + C_i \quad \text{for } i= 1 , 2 , 3 , 4 ....., 7$$

Each segment has one or two registers and a combinational circuit as shown in fig. 4-2

The suboperations performed in each segment of the pipeline are as follows:

R1 < ------- A_i ,	R2 < ------- B_i	Input A_i and B_i
R3 < --------- R1 * R2,	R4 < ---------C_i	Multiply and input C_i
R5 < --------- R3 + R4,		Add C_i to product

The five registers are loaded with new data every clock pulse . The effect of each clock is shown in table 4-1 .

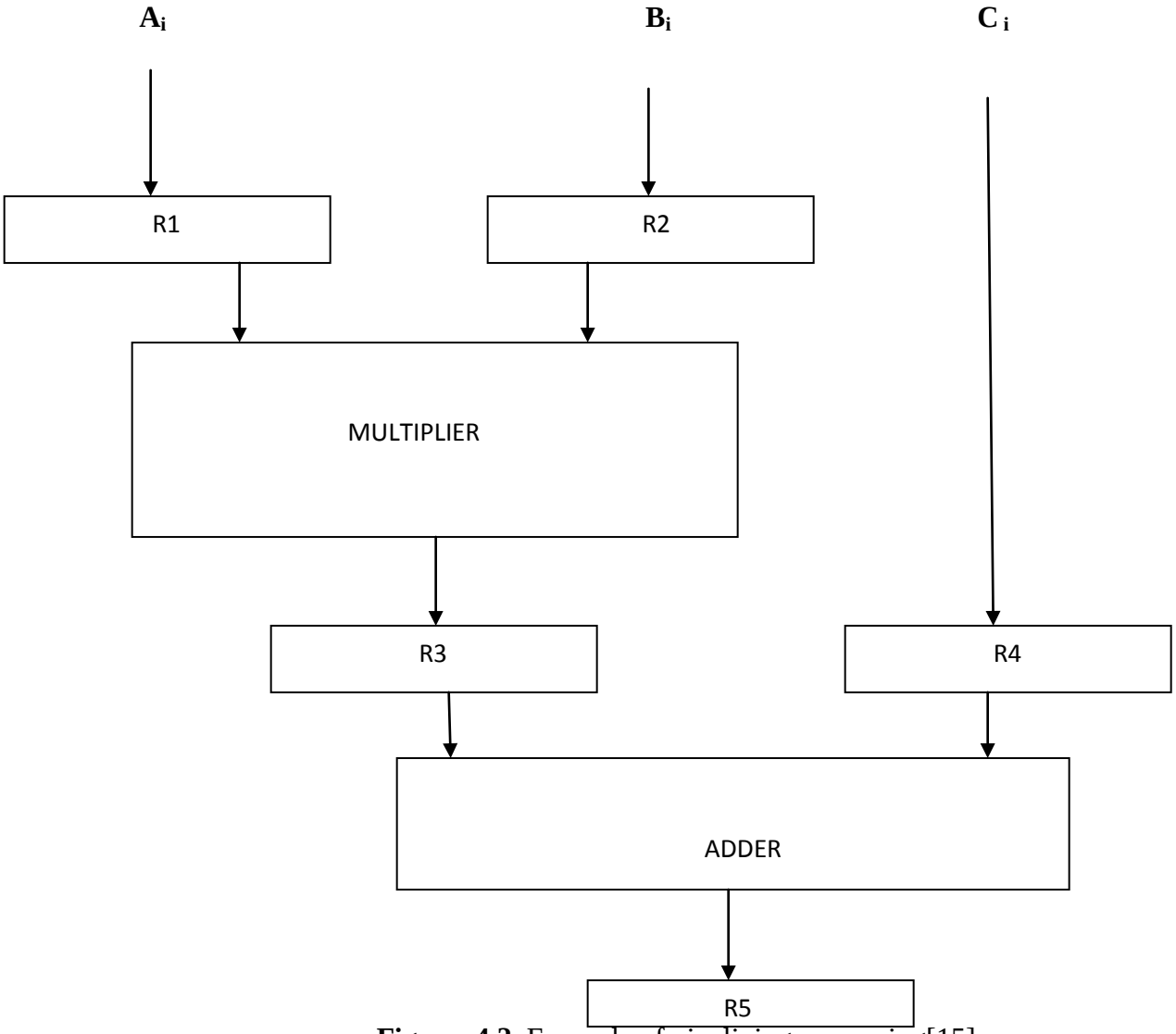


Figure. 4.2. Example of pipelining processing[15]

Table 4.1 Contents of registers in pipeline example[15]

CLOCK PULSE NUMBER	SEGMENT 1		SEGMENT 2		SEGMENT 3
	R1	R2	R3	R4	R5
1	A1	B1			
2	A2	B2	A1*B1	C1	
3	A3	B3	A2*B2	C2	A1 * B1 + C1
4	A4	B4	A3*B3	C3	A2 * B2 + C2
5	A5	B5	A4*B4	C4	A3 * B3 + C3
6	A6	B6	A5*B5	C5	A4 * B4 + C4
7	A7	B7	A6*B6	C6	A5 * B5 + C5
8			A7*B7	C7	A6 * B6 + C6

9			A7 * B7 + C7

4.2.1General Considerations:

Any operation that can be decompose into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor. The technique is efficient for those applications that need to repeat the same task many time with different sets of data. The general structure of a 4-segment pipeline is shown in fig 4.3 The diagram shows six tasks T1 through T6 executed in four segments. Initially, task T1 is handled by segment 1. After the first clock, segment 2 is busy with T1, while segment 1 is busy with task T2. Continuing in this manner, the first task T1 is completed after the fourth clock cycle. From then on, the pipe completes a task every clock cycle. No matter how many segments there are in the system, once the pipeline is full it takes only one clock period to obtain a output.

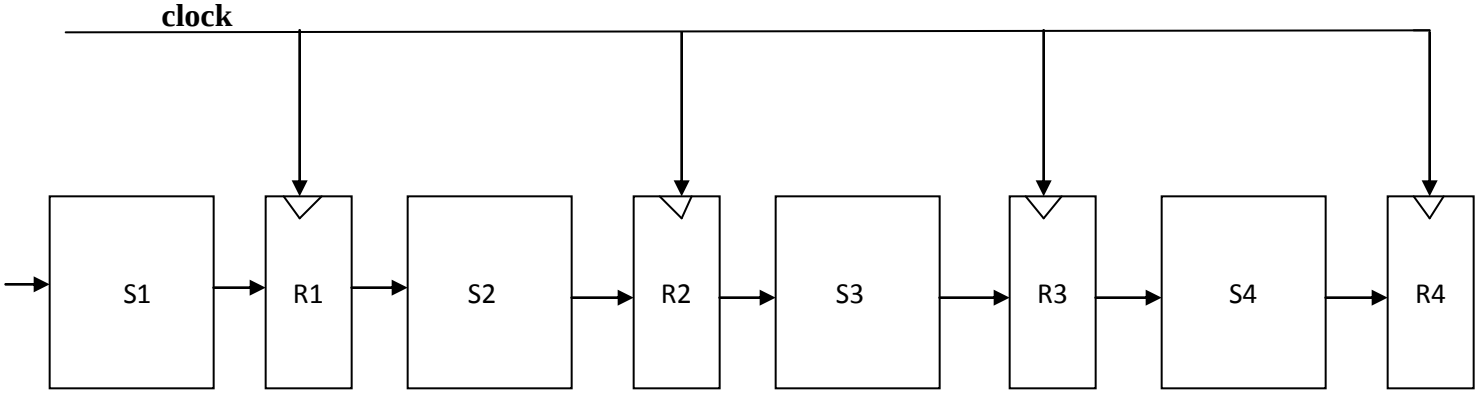


Figure. 4.3. four-segment pipeline[15].

Now consider the case where a k-segment pipeline with a clock cycle time t_p is used to execute n tasks. The first task T1 requires a time equal to k t_p to complete its operation since there are k segments in the pipe. The remaining n-1 tasks emerge from the pipe at the rate of one task per clock cycle. And they will be completed after the time equal to (n-1) t_p. Therefore, to complete n tasks using a k segment pipeline requires k+(n-1) clock cycles. For example the table shows four segments and six tasks. The time required to complete all the operation is 4+(6-1) = 9 clock cycles, as indicated in the table.

Next consider a non-pipeline unit that performs the same operation and takes a time equal to t_n to complete each task. The total time require for n tasks is n t_n.

The speed up of a pipeline processing over an equivalent non-pipe processing is defined by the ratio

$$S = \frac{ntn}{(k+n-1)tp}$$

Table 4.2 Space-time diagram for pipeline[15]

SEGMENT CLOCK CYCLES →

1	2	3	4	5	6	7	8	9
T1	T2	T3	T4	T5	T6			
	T1	T2	T3	T4	T5	T6		
		T1	T2	T3	T4	T5	T6	
			T1	T2	T3	T4	T5	T6

The principles used in instruction pipelining can be used in order to improve the performance of computers in performing arithmetic operations such as add, subtract, and multiply. In this case, these principles will be used to realize the arithmetic circuits inside the ALU. In this section, we will elaborate on the use of arithmetic pipeline as a means to speed up arithmetic operations.

4.2.2Fixed-Point Arithmetic Pipelines:

Addition of these two operands can be performed using a number of techniques . These techniques differ in basically two attributes: degree of complexity and achieved speed. These two attributes are somewhat contradictory; that is, a simple realization may lead to a slower circuit while a complex realization may lead to a faster circuit. Consider, for example, the carry ripple through (CRTA) and a carry look-ahead (CLAA) adder. The CRTA is simple, but slower, while the CLAA is complex, but fast. Addition of these two operands can be performed using a number of techniques . These techniques differ in basically two attributes

:degree of complexity and achieved speed . These two attributes are somewhat contradictory ; that is , a simple realization may lead to a slower circuit while a complex realization may lead to a faster circuit .Consider , for example , the carry ripple through (CRTA) and a carrylook-ahead (CLAA) adders. The CRTA is simple, but slower, while the CLAA is complex but faster.

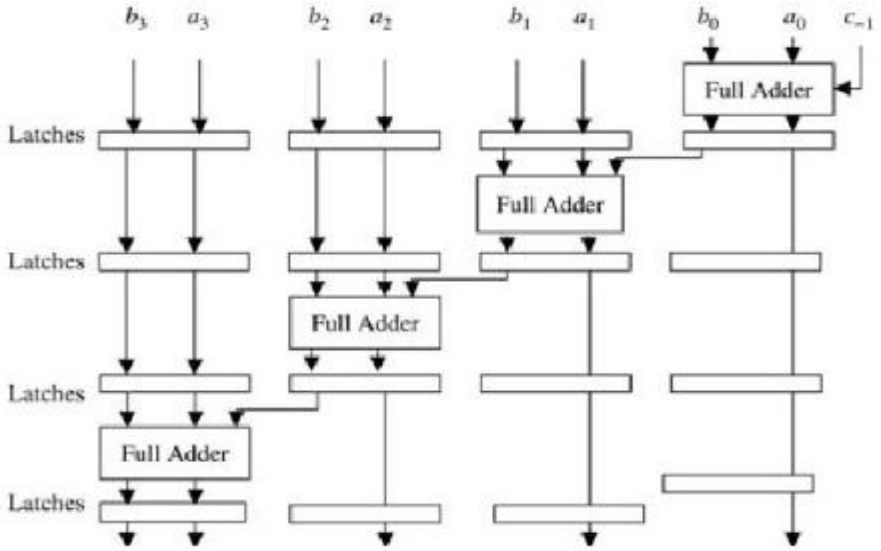


Figure. 4.4. A modified 4-bit CRTA [17]

It is possible to modify the CRTA in such a way that a number of pairs of operands upon, that is pipelined , inside the adder , thus improving the overall speed of addition in the CRTA . Figure 4.3 shows an example of modified 4- bit CRTA .In this case, the two operands A and B are presented to the CRTA through the use of synchronizing elements, such as clocked latches. These latches will guarantee that the movement of the partial carry values within the CRTA are synchronized at the input of the subsequent stages of the adder with the higher order operand bits. For example, the arrival of the first carry out (c0) and the second pair of bits (a1 and b1) is synchronized at the input of second full adder using a latch.

Although the operation of the modified CRTA remains in principle the same; that is the carry ripples through the adder, the provision of latches allows for the possibility of presenting multiple sets of pairs of operands to the adder at the same time.

4.2.3Pipelined Multiplication Using Carry – Save Addition

As indicated before, one of the main problems with addition is the fact that the carry has to ripple through from one stage to the next. Carry rippling through stages can be eliminated using

a method called carry-save addition .Consider the case of adding 44 , 28 , 32 , and 79. A possible way to add these without having the carry ripple through is illustrated in fig 9.4.The idea is to delay the addition of the carry resulting in the intermediate stages until the last step in the addition. Only at the last stage is a carry-ripple stage employed.

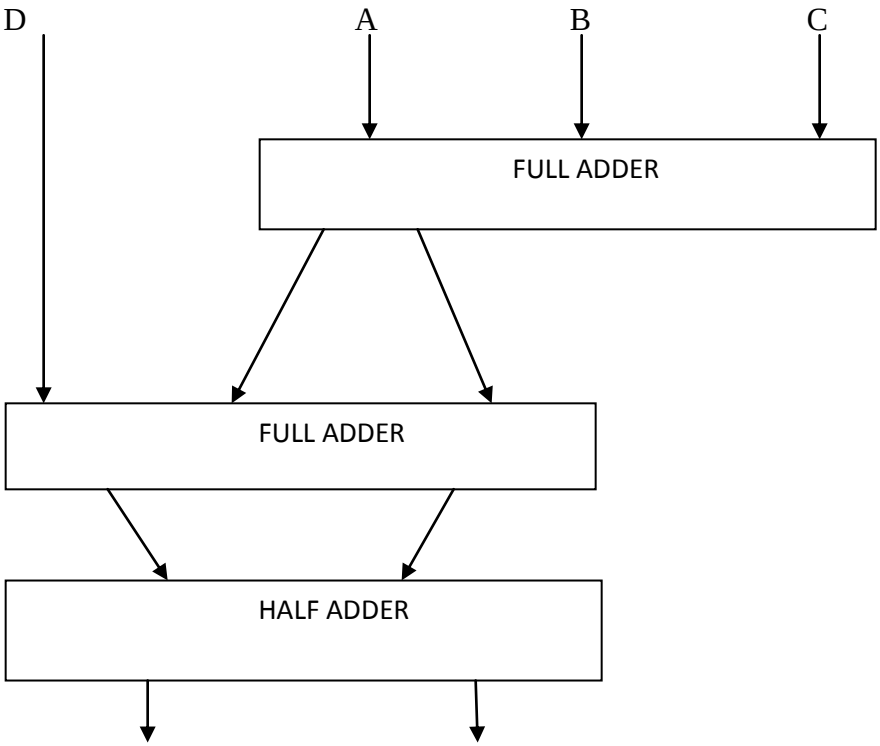


Figure. 4.5. Carry-save addition [17]

Carry-save addition can be used to realize a pipelined multiplication building block. Consider ,for example , the multiplication of two n-bit operands A and B. The multiplication operation can be transformed into an additional on as shown in fig.4.5.

$$P = A * B$$

$$= A * (B_7 * 2^7 + B_6 * 2^6 + B_5 * 2^5 + \ldots + B_0 * 2^0)$$

$$= \sum_{k=0}^{7} A * Bk * 2^k$$

$$= \sum_{k=1}^{7} (S_k)$$

$S_k = A * B_k * 2^k$ represents a 16-bit partial product

Figure. 4.6. A carry-saved based multiplication of two 8-bit operands M and Q [17]

The figure illustrates the case of multiplying two 8-bit operands A and B .A carry- save based multiplication scheme using the principle shown in figure 4.6 is shown in fig. 4.6.

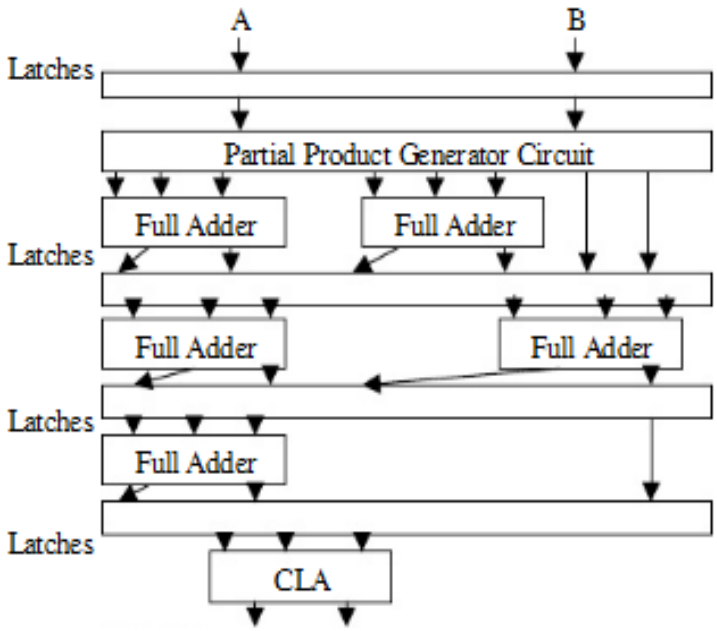


Figure. 4.7. A carry-saved addition-based multiplication scheme[17]

The scheme is based on the idea of producing the set of partial products needed and then adding them up using a carry- save addition scheme.

CHAPTER 5

MULTIPLY AND ACCUMULATE

This chapter introduces the basics of binary multiplication, partial product generation, reduction and techniques to make the multiplication process faster. It also describe the design of 32-bit Pipelined Multiply and Accumulate Unit using Booth Wallace Tree Algorithm with Pipelined Variable Stage Carry Select Adder in the final addition stage of Multiplication and in the Accumulator .

5.1 Basic Architecture of MAC Unit

In the majority of digital signal processing (DSP) applications the critical operations usually involve many multiplications and/or accumulations. For real-time signal processing, a high speed and high throughput Multiplier-Accumulator (MAC) is always a key to achieve a high performance digital signal processing system. In the last few years, the main consideration of MAC design is to enhance its speed. This is because, speed and throughput rate is always the concern of digital signal processing system. But for the epoch of personal communication, low power design also becomes another main design consideration. This is because, battery energy available for these portable products limits the power consumption of the system. Therefore, the main motivation of this work is to investigate various pipelined multiplier/accumulator architectures and circuit design techniques which are suitable for implementing high throughput signal processing algorithms and at the same time achieve low power consumption. A conventional MAC unit consists of (fast multiplier) multiplier and an accumulator that contains the sum of the previous consecutive products. The function of the MAC unit is given by the following equation:

$F = \Sigma A_i B_i$

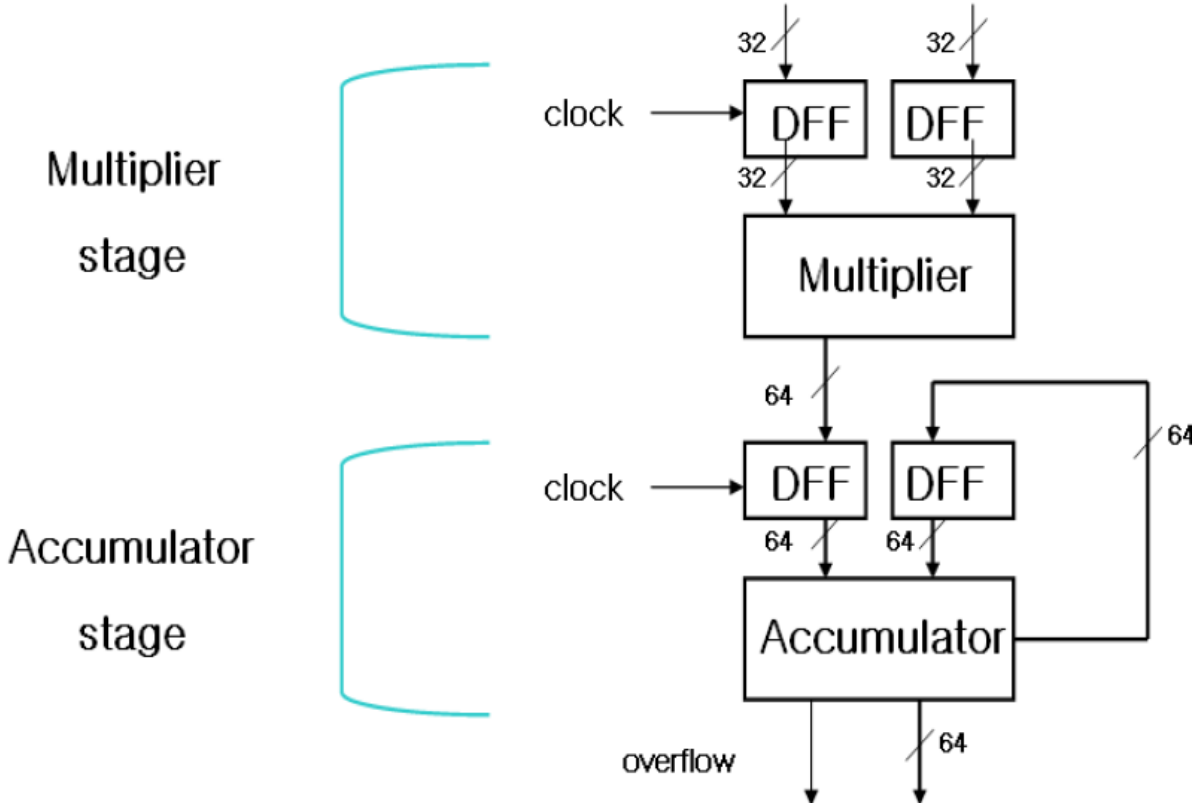


Figure. 5.1. Block diagram of MAC (Multiplier and Accumulator Unit)[22]

A basic MAC unit as shown in Figure 5.1 can be divided into two main blocks [16].

- Multiplier
- Accumulator

5.1.1 Multiplier

A Fast Multiplication process consists of three steps [14]:

- Partial Product Generation.
- Partial Product Reduction.
- Final stage Carry Propagate Adder.

a.Partial Product Generation

To generate the number of partial product Radix-4 Modified booth encoding techniques have been used [15]. The Modified Booth Encoding (MBE) or Modified Booth's Algorithm (MBA) was proposed by O. L. Macsorley in 1961 [16]. Booth's radix-4 algorithm is widely used to reduce the area of multiplier and to increase the speed. The booth encoding algorithm is a bit-pair encoding algorithm that generates partial products which are multiples

of the multiplicand. The booth algorithm shifts and/or complements the multiplicand (X operand) based on the bit patterns of the multiplier (Y operand). Essentially, three multiplier bits [$Y_{(i+1)}$,$Y_{(i)}$ and $Y_{(i-1)}$] are encoded into eight bits that are used to select multiples of the multiplicand [-2X,-X,0,+X,+2X]. The three multiplier bits consist of a new bit pair [$Y_{(i+1)}$ and $Y_{(i)}$] and the leftmost bit from the previously encoded bit pair [$Y_{(i-1)}$]. Grouping the three bits of multiplier with overlapping has half partial products which improve the system speed , as discussed in chapter 3.

b. Partial Product Compression

Multiplier require high amount of power and delay during the partial products addition. At this stage, most of the multipliers are designed with different kind of multi operands adders that are capable to add more than two input operands and results in two outputs, sum and carry. The number of adders will be minimized by Wallace Tree.

4:2 Compressor: 4-2 compressors are used as carry save adders. The 4-2 and 5-2 compressors have been widely employed in the high speed multipliers to lower the latency of the partial product accumulation stage. Owing to its regular interconnection, the 4-2 compressor is ideal for the partial products addition stage. The 4:2 compressor structure actually compresses five partial products bits into three. The architecture is connected in such a way that four of the inputs are coming from the same bit position of the weight j while one bit is fed from the neighboring position j-1(known as carry-in). The outputs of 4:2 compressor consists of one bit in the position j and two bits in the position j+1.This structure is called compressor since it compresses four partial products into two(while using one bit laterally connected between adjacent 4:2 compressors). Figure 5.2 shows the block diagram of 4-2 compressor. A 4-2 compressor can also be built using 3-2 compressors. It consists of two 3-2 compressors (full adders) in series and involves a critical path of 4 XOR delays as shown in Figure 5.3. The output Cout, being independent of the input Cin accelerates the carry save summation of the partial products.

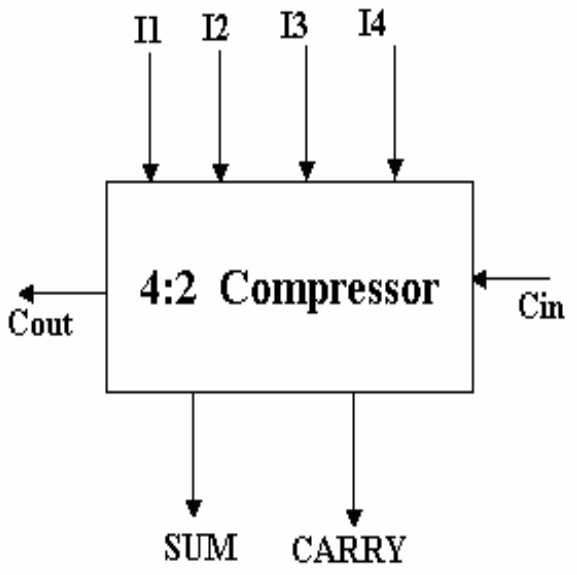


Figure 5.2: Block Diagram of 4:2 Compressor[16]

4:2 compressor is made from 2 full adders. The final carry is saved and hence is called carry save adder. The delay of 4:2 compressor is equal that of 4 xor gates.

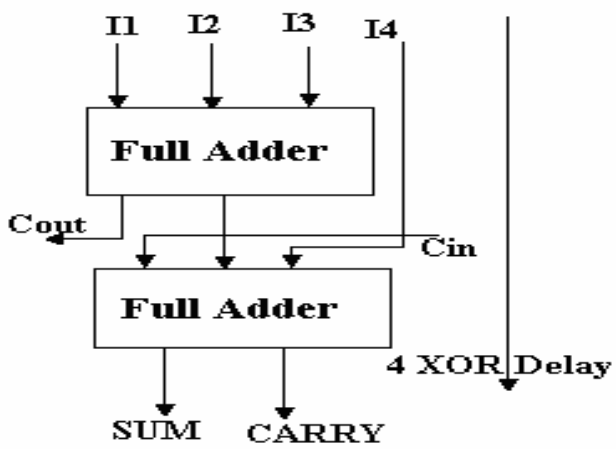


Figure 5.3: 4:2 Compressor Design using Full Adders[16]

ii.Wallace Tree: Wallace Tree is a reduction technique that uses the carry save adder to add the partial products. A block diagram for the data distribution among a tree architecture that employs 4:2 compressors is shown in Figure 5.4.

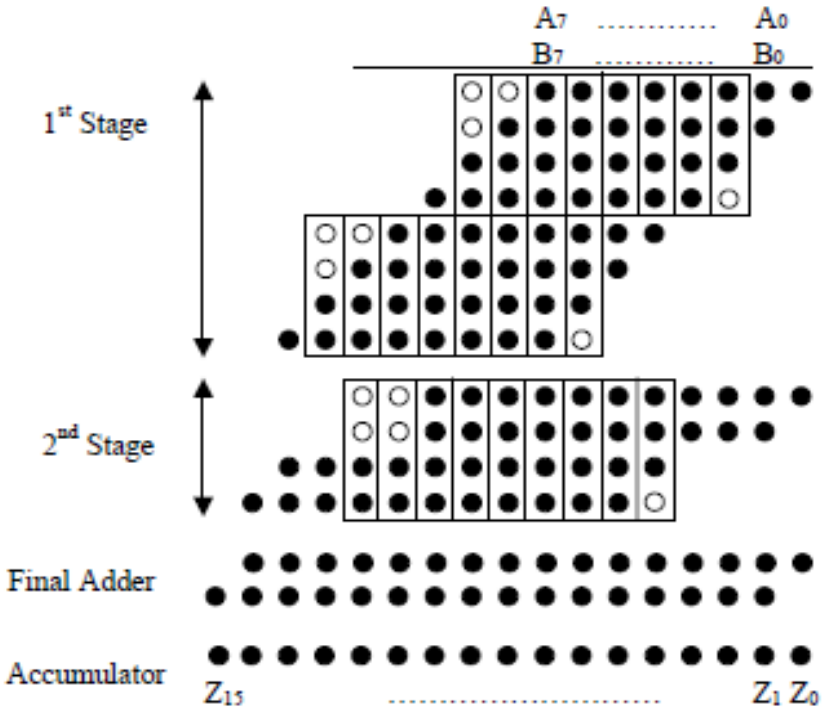


Figure 5.4: Data distribution among a tree architecture [14]

Each box contains the bits that are fed into a 4:2 compressor cell. Two stages of 4:2 compressors are used to reduce the number of partial products by a ratio of 2:1. This figure presents the reduction tree of 8 partial products to form two operands, which are then added together to form a final product by using a fast carry propagate adder.Using 4:2 compressor, there is a simple and more regular wiring of multiplier tree .

c. Final stage Carry Propagate Adder

This stage is also crucial for any multiplier because in this stage addition of large size operands is performed so in this stage fast carry propagate adders like Carry-look Ahead Adder or Carry Skip Adder or Carry Select Adder can be used as per our requirement.

5.1.2 Accumulator

Accumulator basically consists of register and adder. Register hold the output of previous clock from adder. Holding outputs in accumulation register can reduce additional add instruction. An accumulator should be fast in response so it can be implemented with one of fastest adder like Carry-look Ahead Adder or Carry Skip Adder or Carry Select Adder .

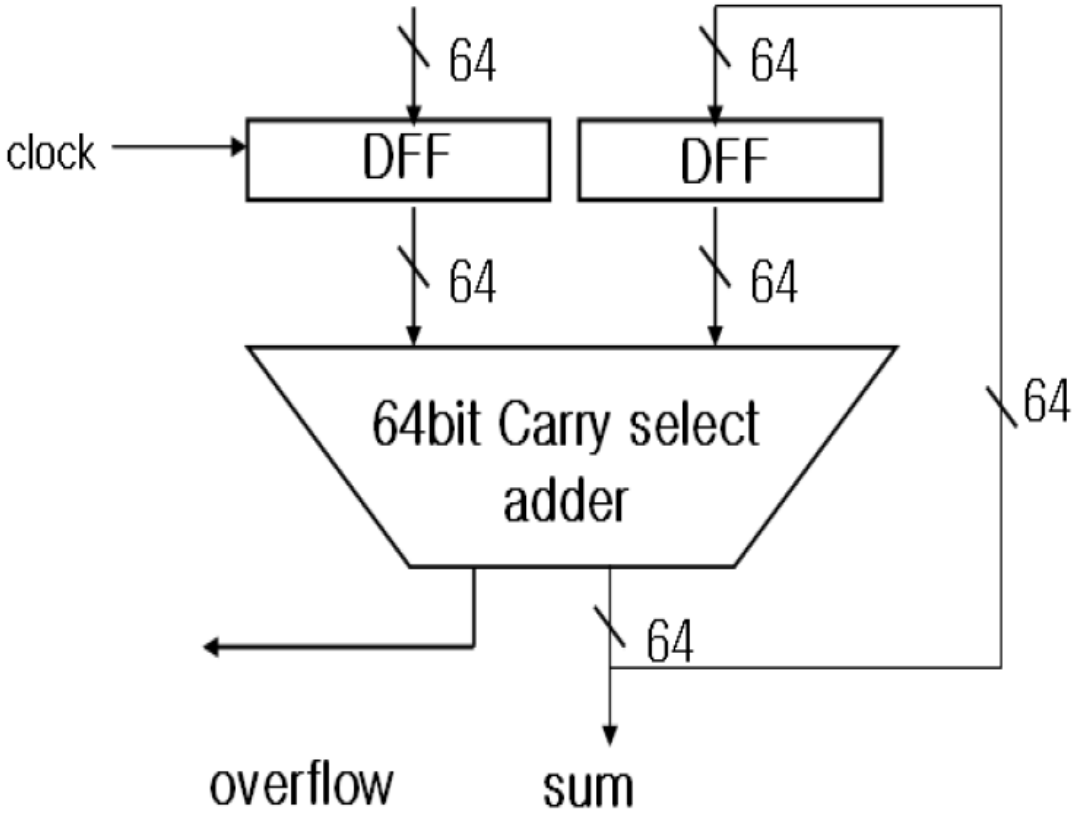


Figure. 5.5. Block diagram of accumulator[22]

5.2 32-bit Pipelined Booth Wallace MAC Unit

The pipeline technique is widely used to improve the performance of digital circuits. As the number of pipeline stages is increased, the path delays of each stage are decreased and the overall performance of the circuit is improved [18]. The various pipeline schemes have been investigated to find the optimum number of pipeline stages and the positions for the pipeline registers to be inserted in modified Booth multiplier in order to obtain the high-speed. At first, modified Booth multiplier is partitioned into three pipeline stages according to the functionality of the circuit as shown in Figure 5.6. The critical path of the pipelined Booth multiplier is in the Wallace tree because it requires the most intensive computation. It means that delays can be further reduced by adding more pipeline registers within the Wallace Tree as shown in Figure 5.7.

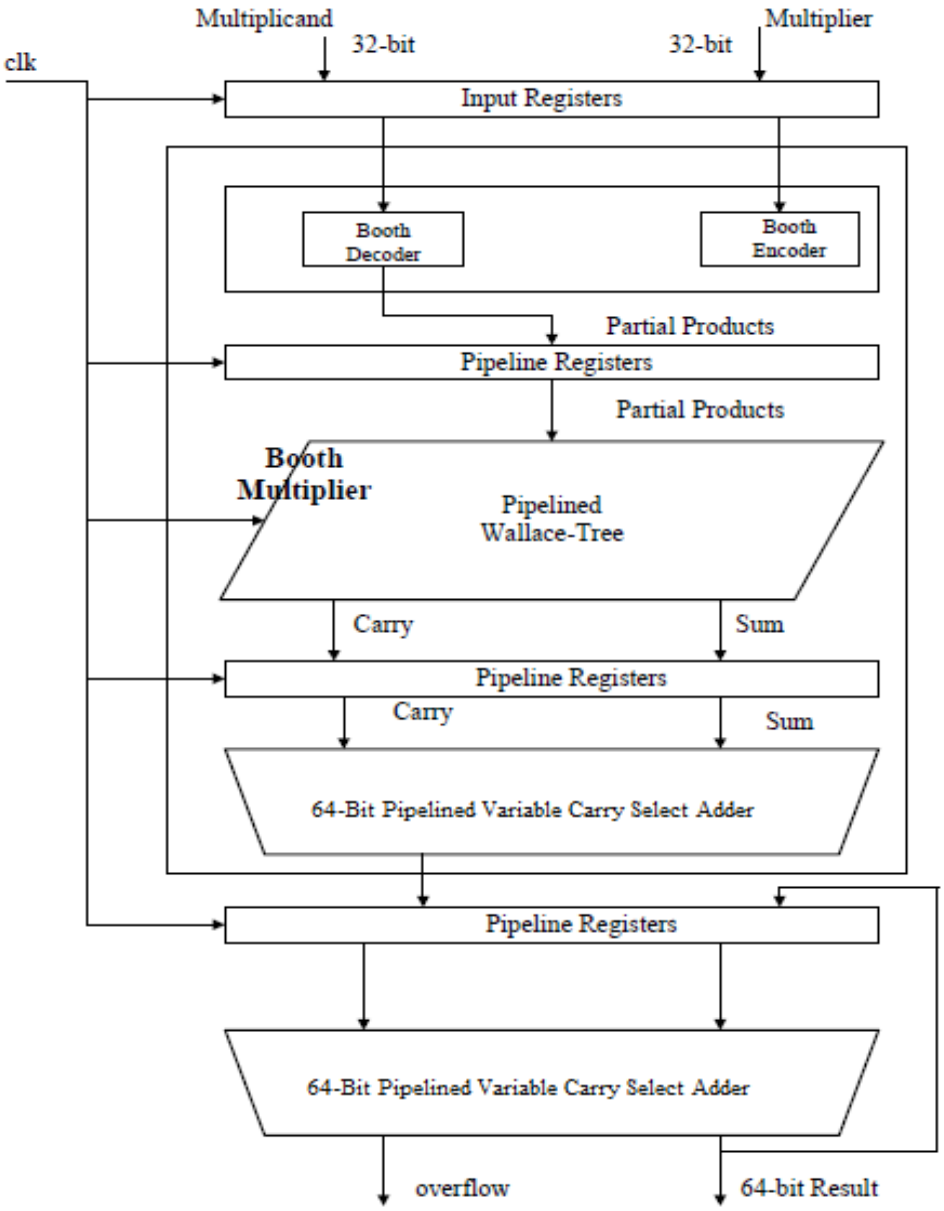


Figure 5.6: Block Diagram of 32-bit Pipelined Booth Wallace MAC Unit

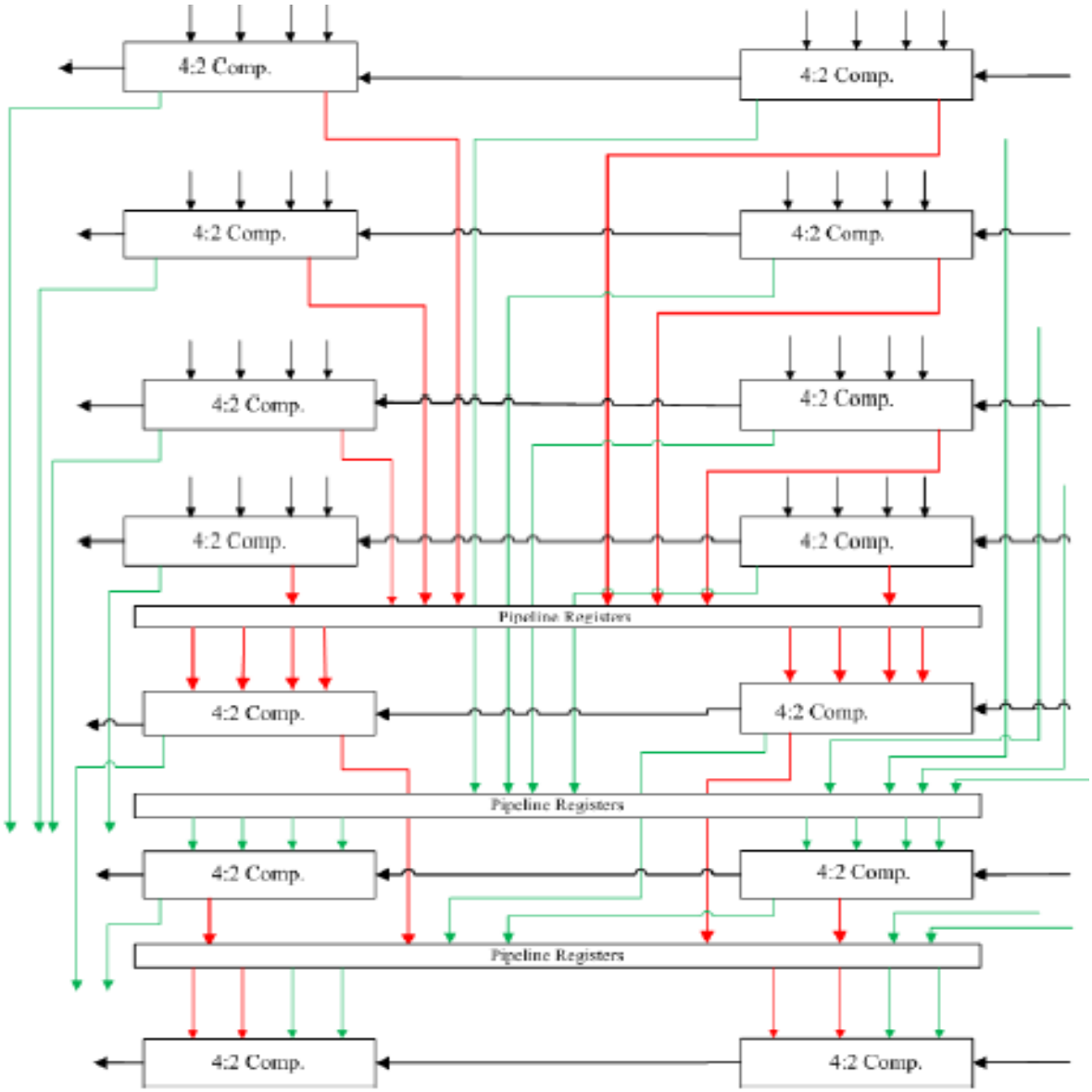


Figure 5.7: Pipelining in Wallace Tree

5.3 Carry Select Adder

The result from wallace tree is the sum and carry vector hence, it needs to be added to get final result. If we use ripple carry adder to add the sum and the carry vector, then the carry propagation delay has a bad effect on the performance of MAC. Carry Select Adder computes when both carry is 0 and 1. In spite of large implementation, it can increase the speed of execution . In this I have used Pipelined Variable Stage Carry Select Adder ,in which ripple carry adders of 4-bit and 6-bit have been used , as shown in figure 5.8.

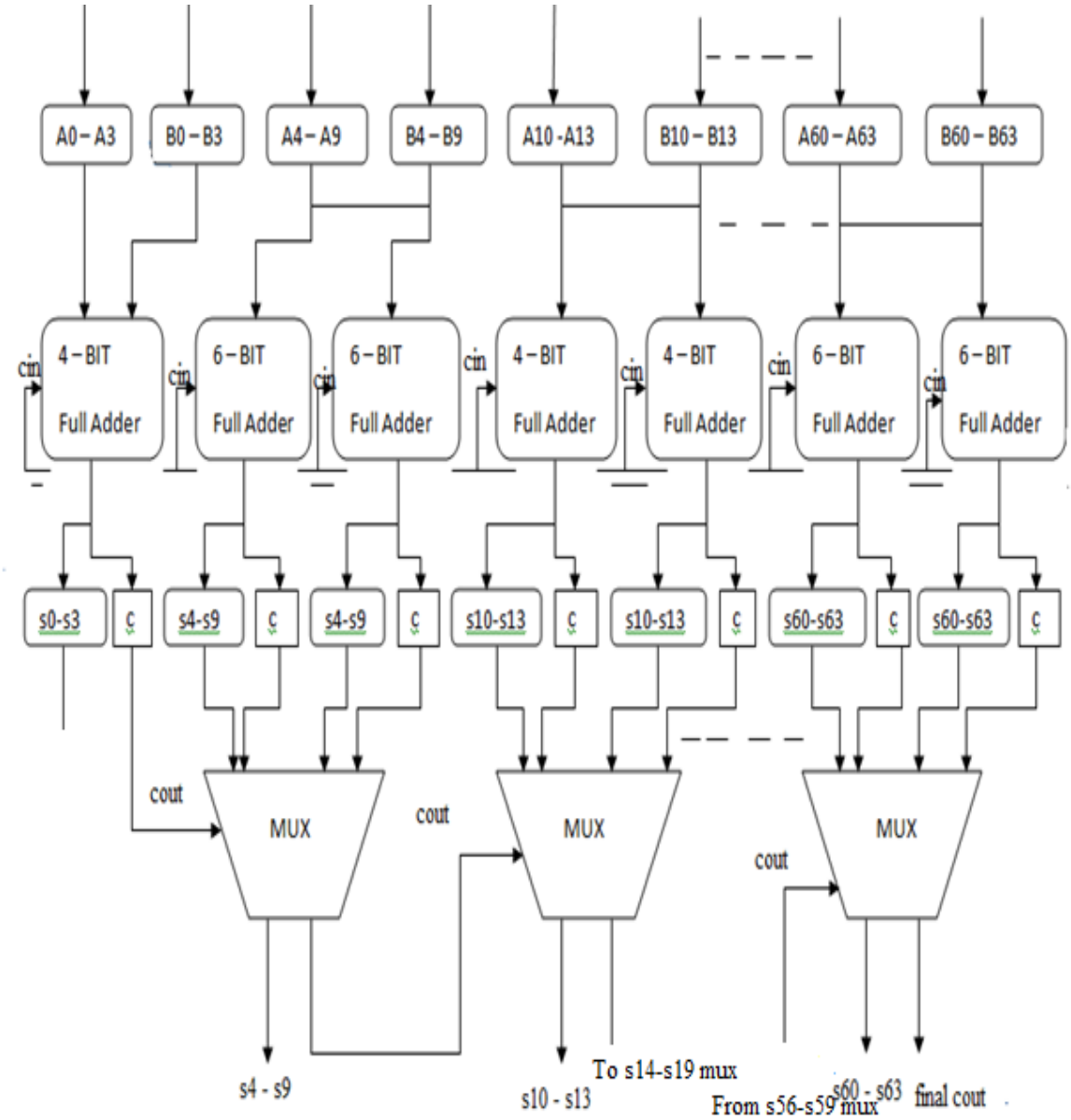


Figure 5.8: 64-bit Pipelined Variable Stage Carry-Select Adder

CHAPTER 6

FIELD PROGRAMMABLE GATE ARRAY

This chapter introduce about the FPGA concepts and FPGA Synthesis Flow. An FPGA is a device that consists of thousands or even millions of transistors connected to perform logic functions [25]. They perform functions from simple addition and subtraction to complex digital filtering and error detection and correction. Aircraft, automobiles, radar, missiles, and computers are just some of the systems that use FPGAs. The main benefit of using FPGAs is that design changes need not have an impact on the external hardware. Under certain circumstances, an FPGA design change can affect the external hardware.

6.1 Basic FPGA Concepts

The basic FPGA architecture consists of a two dimensional array of logic blocks and flip-flops with means for the user to configure

- The function of each logic blocks.
- The inputs/outputs.
- The interconnection between blocks.

Families of FPGAs differ from each other by the physical means for implementing user programmability, arrangement of interconnection wires, and basic functionality of the logic blocks.

6.2 Xilinx Specifics

Spartan III is the low-cost version of Virtex II. All Xilinx FPGAs contain the same basic resources like

- Configurable logic blocks (CLBs).
- Input/output blocks (IOBs).
- RAM blocks.
- Programmable Interconnections (PIs).
- Other resources like three-state buffers, clock buffers, boundary scan logic, and so on.

6.2.1 Configurable Logic Blocks

The basic building block of Xilinx CLBs is the slice. Spartan III hold four slices per CLB. Each slice contains two 4-input function generators (F/G), carry logic, and two elements. Each function generator output drives both the CLB output and the D-input of a flip-flop. Besides the four basic function generators, the Spartan III CLB contains logic that combines function generators to provide functions of five or six inputs. The look-up tables and storage elements of the CLB have the following characteristics:

a.Look-Up Tables

The way logic functions are implemented in a FPGA is another key feature. Logic blocks that carry out logical functions are look-up tables, implemented as memory, or multiplexer and memory. Figure 6.1 shows these alternatives, together with an example of memory contents for some basic operations. A 2n*1 ROM can implement any n-bit function. Typical sizes for n are 2, 3, 4, or 5. In Figure 6.1(a), an n-bit LUT is implemented as a 2n*1 memory; the input address selects one of 2n memory locations. The memory locations are normally loaded with values from the user's configuration bitstream. In Figure 6.1(b), the multiplexer control inputs are the LUT inputs. The result is a general-purpose "logic gate." An n-LUT can implement any n-bit function.

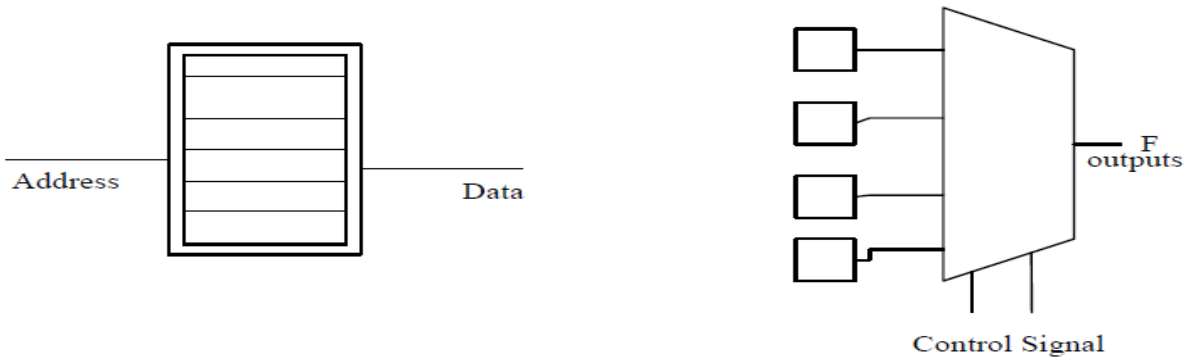


Figure 6.1: Look-up table implemented as (a) Memory (b) Multiplexers and Memory [25]

b. Storage Elements

The storage elements in a slice can be configured either a edge-triggered D-type flipflops or as level-sensitive latches. The D-input can be driven either by the function generators within the slice or directly from the slice inputs, bypassing the function generators. As well as clock and clock enable signals, each slice has also synchronous set and reset signals.

6.2.2 Input/output Blocks

The Xilinx IOB includes inputs and outputs that support a wide variety of I/O signalling standards. The IOB storage elements act either as D-type flip-flops or as latches. For each flip-flop, the set/reset (SR) signals can be independently configured as synchronous set, synchronous reset, asynchronous preset, or asynchronous clear. Pull-up and pull-down resistors and an optional weak-keeper circuit can be attached to each pad. IOBs are programmable and can be categorized as follows:

a. Input Path

A buffer in the IOB input path is routing the input signals either directly to internal logic or through an optional input flip-flop.

b. Output Path

The output path includes a 3-state output buffer that drives the output signal onto the pad. The output signal can be routed to the buffer directly from the internal logic or through an optional IOB output flip-flop. The 3-state control of the output can also be routed directly from the internal logic or through a flip-flop that provides synchronous enable and disable signals.

c. Bidirectional Block

This can be any combination of input and output configurations.

6.2.3 RAM Blocks

Xilinx FPGA incorporates several large RAM memories (block select RAM). These memory blocks are organized in columns along the chip. The number of blocks ranging from 8 up to more than 100, depending on the device size and family.

6.2.4 Programmable Routing

Adjacent to each CLB stands a general routing matrix (GRM). The GRM is a switch matrix through which resources are connected, the GRM is also the means by which the CLB gains access to the general-purpose routing. Horizontal and vertical routing resources for each row or column include:

a. Long Lines

Bidirectional wires that distribute signals across the device. Vertical and horizontal long lines span the full height and width of the device.

b. Hex Lines

Route signals to every third or sixth block away in all four directions.

c. Double Lines

Route signals to every first or second block away in all four directions.

d. Direct Lines

Route signals to neighbouring blocks vertically, horizontally & diagonally.

e. Fast Lines

Internal CLB local interconnections from LUT outputs to LUT inputs.

6.3 FPGA Generic Design Flow

The FPGA design flow has several points in common with the semicustom ASIC design flow. Figure 4.2 shows a simplified FPGA design flow. The successive process phases of Figure 6.2 are described as follows:

6.3.1 Design Entry

The basic architecture of the system is designed in this step which is coded in a Hardware Description Language like Verilog or VHDL.

6.3.2 Simulation

Simulation is the process of applying stimulus or inputs that mimic actual data to the design and observing the output.

a. Functional Simulation

After the design phase, the code is verified to check its functionality using a simulation software for different inputs and if it verifies then proceed further otherwise necessary corrections and modification will be done in HDL code. The functional or behavioural simulation does not take into account component or interconnection delays. It just check the functionality of design.

b. Timing simulation

It uses back-annotated delay information extracted from the circuit. Other reports are generated to verify other implementation results, such as maximum frequency and delay and resource utilization. The timing simulation takes into account component or interconnection delays.

6.3.3 Design Synthesis

A process that starts from a high level of logic abstraction (typically Verilog or VHDL) and automatically creates a lower level of logic abstraction using a library of primitives. During synthesis, the Xilinx ISE tool does the following operations:

a. HDL Compilation

The tool compiles all the sub-modules in the main module if any and then checks the syntax of the code written for the design.

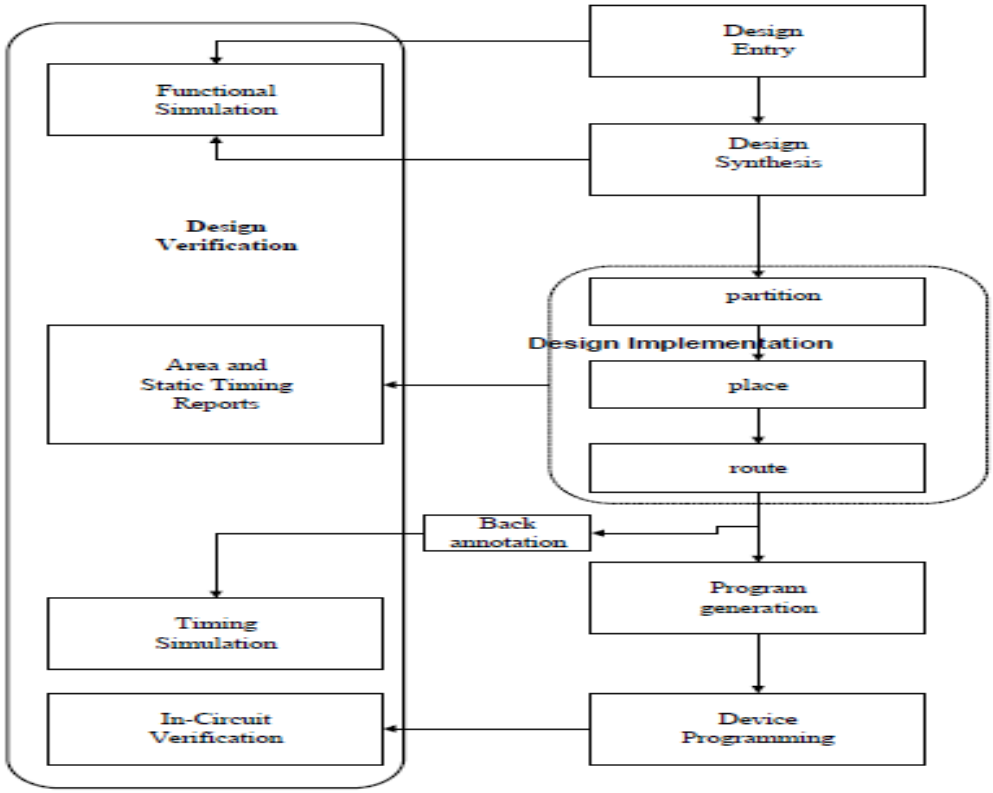


Figure 6.2: FPGA Design Flow [21]

b. Design Hierarchy Analysis

Analysis the hierarchy of the design.

c. HDL Synthesis

The process which translates VHDL or Verilog code into a device net list format. i.e. a complete circuit with logical elements such as Multiplexer, Adder, Subtractors, Counters, Registers, Flip flops, Latches, Comparators, XORs, Tristate Buffers, Decoders, etc.

d. Advanced HDL Synthesis

The blocks synthesized in the HDL synthesis and the Advanced HDL synthesis are further defined in terms of the low level blocks such as buffers, lookup tables. It also optimizes the logic entities in the design by eliminating the redundant logic, if any. The tool then generates a, netlist file (NGC file) and then optimizes it.

6.3.4 Design Implementation

Implementation is the process that maps the synthesized netlist to the specific or target FPGA's resources and interconnects them to the FPGA's internal logic and I/O resources. During this process, the physical design layout is determined. This is the final development process that manipulates the design before it is programmed into a device.

a. Translate

The translate process merges all of the input netlist and design constraint outputs a Xilinx NGD file. The .ngd file describes the logical design reduced to the Xilinx device primitive cells. The output in the floor planner tool supplied with Xilinx ISE software. Here, defining constraints is nothing but, assigning the ports in the design to the physical elements (ex. pins, switches, buttons etc) of the targeted device and specifying time requirements of the design. This information is stored in a file extension named UCF.

b. Map

The map process is run after the translate process is complete. Mapping maps the logical design described in the NGD file to the components/primitives (Slices/CLBs) present on the NCD file created by the Map process to place and route the design on the target FPGA design. In this process the whole circuit is divided into sub blocks so that they can fit into FPGA logic blocks. The logic defined in the input NGD file is mapped into targeted FPGA elements i.e. CLBs and IOBs and an output NCD file is generated which depicts the design mapped into the FPGA.

c. Place and Route

After map the design is placed and routed. Place and Route program is used for this process. The place and route process places the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks. Example, if a sub block is placed in a logic block which is very near to I/O pin, then it may save the time but it may affect some other constraint. So trade off between all the constraints is taken account by the place and route process .The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. Output NCD file consist the routing information. During the place process

the sub blocks are placed according to the logic but these blocks do not have physical routing among them and with I/O pads also but there is only a logical connection between them which can be clearly seen using the FPGA Editor just after the place process ends. These logical connections are shown by nets in FPGA Editor. Then the Route process is run which makes physical connections between the sub blocks placed on FPGA. The connections are made using the switch matrices.

6.3.5 Device Programming

A programming file is generated by running the Generate Programming File process. This process can be run after the FPGA design has been completely routed. The Generate Programming File process runs bitgen, the Xilinx bit stream generation program, to produce a bitstream (.BIT) or (.ISC) file for Xilinx device configuration. The FPGA device is then configured with the .bit file using the JTAG boundary scan method. After the Spartan device is configured for the intended design, then its working is verified by applying different inputs.

6.3.6 Static Timing Analysis

a. Post-fit Static Timing Analysis

The Analyse Post-Fit Static timing process opens the timing Analyzer window, which lets the interactively select timing paths in design for tracing the timing results.

b. Post-Map Static Timing Analysis

It can be analyse the timing results of the Map process. Post-Map timing reports can be very useful in evaluating timing performance (logic delay + route delay).

c. Post Place and Route Static Timing Analysis

Post-PAR timing reports incorporate all delays to provide a comprehensive timing summary. If a placed and routed design has met all of timing constraints, then it can proceed by creating configuration data and downloading a device.

6.3.7 LCD Interfacing

There are two possible interfaces to the LCD controller, one 8 bits wide and another 4 bits wide. The designers of the Spartan-3E chose to use the four bit interface and share it with the onboard Intel Strata Flash storage device to minimize pin count. This will slightly complicate the procedure of initializing and writing to the display. There are three main steps in using the display, the first being the initialization of the four bit interface itself, the second being the commands to set the display options and the third being the writing of character data.

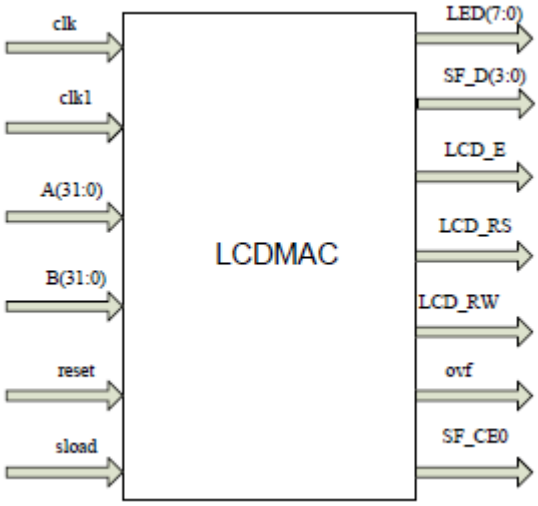


Figure 6.3: Block diagram of LCD interfacing for 32-bit MAC Unit

a. Initialization

1. Wait 15 ms or longer, the 15 ms interval is 750,000 clock cycles at 50 MHz.
2. Write SF_D<3:0> = 0x3, pulse LCD_E High for 12 clock cycles.
3. Wait 4.1 ms or longer, which is 205,000 clock cycles at 50 MHz.
4. Write SF_D<3:0> = 0x3, pulse LCD_E High for 12 clock cycles.
5. Wait 100 µs or longer, which is 5,000 clock cycles at 50 MHz.
6. Write SF_D<3:0> = 0x3, pulse LCD_E High for 12 clock cycles.
7. Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.
8. Write SF_D<3:0> = 0x2, pulse LCD_E High for 12 clock cycles.
9. Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.

b. Configuration

This step involves the configuration and actual writing to the LCD RAM.

1. Issue a Function Set command, 0x28, to configure the display for operation on the Spartan-3E Starter Kit board.
2. Issue an Entry Mode Set command, 0x06, to set the display to automatically increment the address pointer.
3. Issue a Display On/Off command, 0x0C, to turn the display on and disables the cursor and blinking.

4. Finally, issue a Clear Display command. Allow at least 1.64 ms (82,000 clock cycles) after issuing this command.

c. Display

This step involves the actual process of writing data to the DD-RAM.

1. Specify the start address with a Set DD-RAM Address command.

2. Display a character with a Write Data command.

6.3.8 Keyboard Interfacing

The Spartan 3E boards support the use of either a mouse or keyboard using a PS/2 connector on the board. This provides only the basic electrical connections from the PS/2 cable. It is necessary to design a hardware interface using communicate with a keyboard or a mouse. Serial-to-parallel conversion using a shift register is required.

a. PS/2 Port Connections

The PS/2 port consists of 6 pins including ground, power (VDD), keyboard data, and a keyboard clock line. The UP 1 board supplies the power to the mouse or keyboard. Two lines are not used. The keyboard data line is pin 31 on the FLEX chip, and the keyboard clock line is pin 30. Pins must be specified in one of the design files. Both the clock and data lines are open collector and bidirectional. The clock line is normally controlled by the keyboard, but it can also be driven by the computer system or in this case the FLEX chip, when it wants to stop data transmissions from the keyboard. Both the keyboard and the system can drive the data line. The data line is the sole source for the data transfer between the computer and keyboard.

b. Keyboard Scan Codes

Keyboards are normally encoded by placing the key switches in a matrix of rows and columns. All rows and columns are periodically checked by the keyboard encoder or "scanned" at a high rate to find any key state changes. Key data is passed serially to the computer from the keyboard using what is known as a scan code. Each keyboard key has a unique scan code based on the key switch matrix row and column address to identify the key pressed.

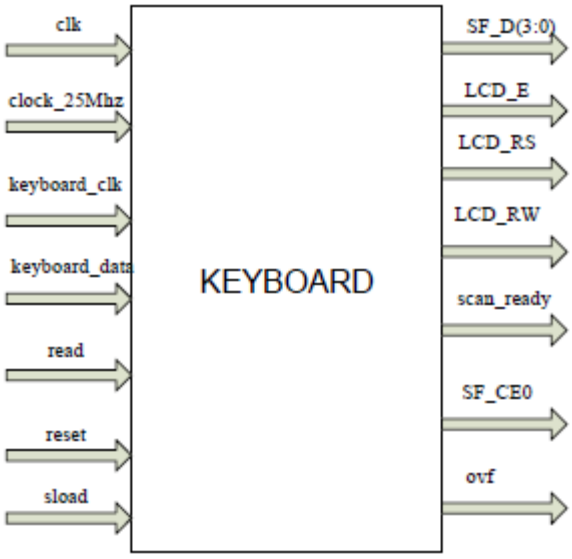


Figure 6.4: Block Diagram of 32-bit MAC unit with Keyboard and LCD Interfacing

c. Make and Break Codes

The keyboard scan codes consist of Make and Break codes. One make code is sent every time a key is pressed. When a key is released, a break code is sent. For most keys, the break code is a data stream of F0 followed by the make code for the key. Be aware that when typing, it is common to hit the next key(s) before releasing the first key hit. Using this configuration, the system can tell whether or not the key has been pressed, and if more than one key is being held down, it can also distinguish which key has been released.

Chapter 7

SYNOPSYS DESIGN COMPILER

The Design Compiler is a synthesis tool from Synopsys Inc.. In simple terms, the synthesis tool takes a Register Transfer Logic hardware description (design written in either Verilog/VHDL), and standard cell library as input and the resulting output would be a technology dependent gatelevel-netlist. The gatelevel-netlist is nothing but structural representation of only standard cells based on the cells in the standard cell library.

7.1 Basic Design and Timing Attributes

These attributes set the environment in which a design is synthesized. These attributes specify the process parameters, I/O port attributes, and statistical wire-load model and timing.

1. Design: It corresponds to the circuit description that performs some logical function. The design may be stand-alone or may include other sub-designs. Although sub-design may be part of the design, it is treated as another design by the Synopsys.

2. Cell: It is the instantiated component of the sub-design in the design.

3. Reference: This is the definition of the original design to which the cell or instance refers. For example, a leaf cell in the netlist must be referenced from the link library, which contains the functional description of the cell. Similarly an instantiated sub-design must be referenced in the design, which contains functional description of the instantiated sub design.

4. Ports: These are the primary inputs, outputs or I/O's of the design.

5. Pin: It corresponds to the inputs, outputs or I/O's of the cells in the design.

6.Net: These are the signal names, i.e., the wires that hook up the design together by connecting ports to pins and/or pins to each other.

7. Clock: The port or pin that is identified as a clock source. The identification may be internal to the library or it may be done using dc_shell commands.

8. Library: Corresponds to the collection of technology specific cells that for the design is targeting for synthesis, or linking for reference.

9. Load: Each output can specify the drive capability that determines how many loads can be driven within a particular time. Each input can have a load value specified that determines how much it will slow a particular driver. The load attribute specifies how much capacitive load

exists on a particular output signal. The load value is specified in the units of the technology library in terms of picofarads or standard loads, etc.

10. Drive: The drive specifies the drive strength at the input port. It is specified as a resistance value. This value controls how much current a particular driver can source. The larger a driver is, i.e 0 resistance, the faster a particular path will be, but a large driver will take more area, so the designer needs to trade off speed and area for the best performance.

11. Clock Latency: Clock latency means delay between the clock source and the clock pin. This is called as source latency of the clock. Normally it specifies the skew between the clock generation point and the clock pin.

12. Rise Time: It is defined as the time it takes for a waveform to rise from 10% to 90% of its steady state value.

13. Fall time: It is defined as the time it takes for a waveform to rise from 90% to 10% of its steady state value.

14. Clock-Q Delay: It is the delay from rising edge of the clock to when Q (output) becomes available. It depends on input clock transition and Output Load Capacitance.

15. Clock Skew: It is defined as the time difference between the clock path reference and the data path reference. The clock path reference is the delay from the main clock to the clock pin and data path reference is the delay from the main clock to the data pin of the same block. (Another way of putting it is the delay between the longest insertion delay and the smallest insertion delay.)

16. Metastability: It is a condition caused when the logic level of a signal is in an indeterminate state.

17. Critical Path: The clock speed is normally determined by the slowest path in the design. This is often called as 'Critical Path'.

18. Clock jitter: It is the variation in clock edge timing between clock cycles. It is usually caused by noise.

19. Set-up Time: It is defined as the time the data/signal has to stable before the clock edge.

20. Hold Time: It is defined as the time the data/signal has to be stable after the clock edge.

21. Interconnect Delay: This is delay caused by wires. Interconnect introduces three types of parasitic effects – capacitive, resistive, and inductive – all of which influence signal integrity and degrade the performance of the circuit.

22. Negative Setup Time: In certain cases, due to the excessive delay (caused by lot of inverters in the clock path) on the clock signal, the clock signal actually arrives later than the data signal. At the actual clock edge the data to latch arrives later than the data signal. This is called negative set up time.

23. Negative Hold Time: It basically allows the data that was supposed to change in the next cycle, change before the present clock edge.

24. Slack: The amount of time by which the timing constraint is met is called the slack of the timing check. If the signal arrives earlier than necessary, then the slack is positive. If the signal arrives later than required time, then slack is negative. If the signal arrives exactly at the required time then slack is zero.

7.2 Basic DC Synthesis Flow

The DC synthesis flow has several points in common with the semicustom ASIC design flow. The flow chart of synthesis process is shown in Figure 7.1.

7.2.1 Reading of Design and Library

Design Compiler reads the RTL hardware description written in either Verilog/VHDL. Design Compiler reads in technology libraries, Design Ware libraries, and symbol libraries to implement synthesis. During the synthesis process, Design Compiler translates the RTL description to components extracted from the technology library and Design Ware library. The technology library consists of basic logic gates and flip-flops. The Design Ware library contains more complex cells for example adders and comparators which can be used for arithmetic building blocks. DC can automatically determine when to use Design Ware components and it can then efficiently synthesize these components into gate-level implementations.

7.2.2 Design Environment

The design environment is a set of attributes and constraints that model the environment surrounding the design being synthesized. Design Compiler uses these attributes and constraints to compute the effect of the design environment on the circuit performance. The design environment includes the following items:

a. Operating Conditions

The operating conditions include the operating temperature, supply voltage, and manufacturing process.

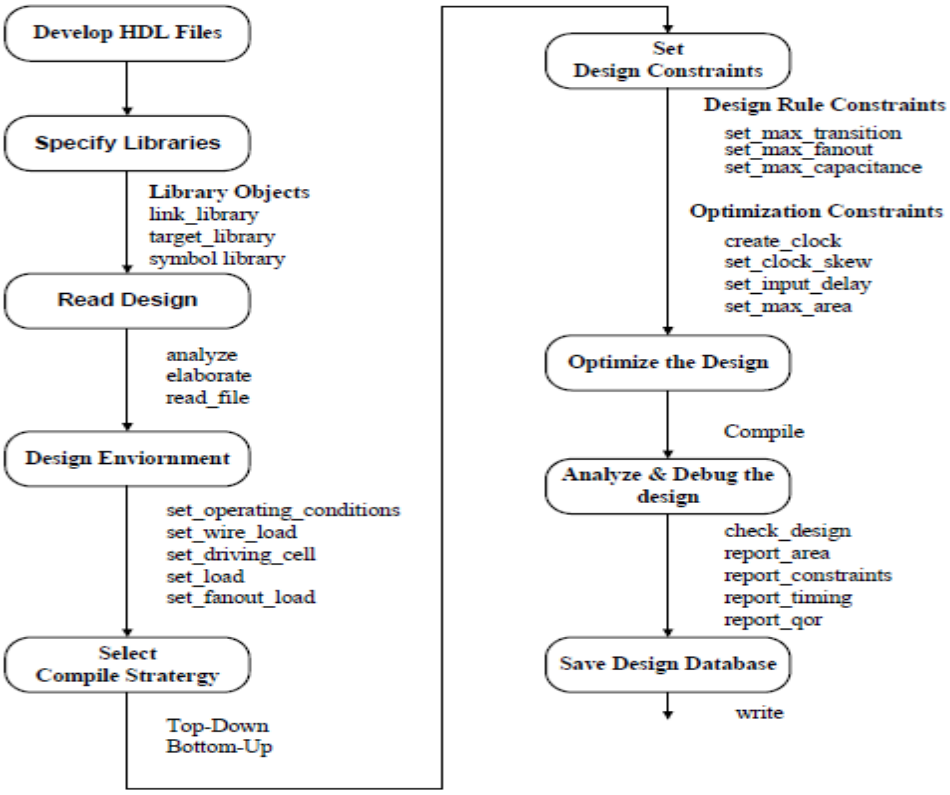


Figure 7.1: Basic Design Compiler Synthesis Flow [31]

b. Wire Load Models

Wire load models estimates the effect of wire length and fan-out on resistance, capacitance, and area to estimate wire delays.

c. System Interface

The system interface includes the cells driving the design and the loads driven by the design.

7.2.3 Compile Strategy

There are two methods used in synthesis compilation Top-Down and Bottom-Up. The different strategy for different designs can be used, hierarchy levels, and components in the design.

a. Top-Down Compilation

In this the designer need only be concerned with top-level design constraints. The design constraints of sub modules in lower-level hierarchy need not to be considered.Submodule timing information is handled by Design Compiler from top-level hierarchy. Top-Down Compilation is not advisable for large designs. The compilation time can be much longer in this by using this compilation method, if the design is large and also Design Compiler might crash due to insufficient memory.

b. Bottom-Up Compilation

In this compilation method, compilation begins at the submodule level and moves towards the top level. In this designer needs to know the timing information for the inputs and outputs for each submodule so the designer needs to perform time budgeting on the submodules. It's usually much faster than the Top-Down Compilation.

7.2.4 Design Constraints

A designer, in order to achieve optimum results, has to methodically constrain the design, by describing the design environment, target objectives and design rules. The constraints contain timing and/or area information, usually derived from the design specifications. The synthesis tool uses these constraints to perform synthesis and tries to optimize the design with the aim of meeting target objectives.

a. Design Rule Constraints

Design rule constraints have two sources, attributes specified in the technology library and attributes applied explicitly. If a technology library defines these attributes, then Design Compiler implicitly applies them to any design using that library when it compiles the design or creates a constraint report. The design rule attributes defined in the technology library cannot be removed because they are requirements for the technology, but these can be made more restrictive to suit the design.

The most commonly specified design rule constraints are:

- Transition time
- Fan-out load
- Capacitance

i. Transition Time Constraints: The transition time of a net is the time required for its driving pin to change logic values. Design Compiler calculates the transition time for each net by

multiplying the drive resistance of the driving pin by the sum of the capacitive loads connected to the driving pin. The maximum transition time restriction specified in a technology library by using the set_max_transition command. This command sets a maximum transition time for the nets attached to the identified ports or to all the nets in a design by setting the max_transition attribute on the named objects.

ii. **Fan-out Load Constraints:** The maximum fan-out load for a net is the maximum number of loads the net can drive. The fan-out load value does not represent capacitance; it represents the weighted numerical contribution to the total fan-out load. Design Compiler calculates the fan-out of a driving pin by adding the fan-out load values of all inputs driven by that pin. To determine if the pin meets the maximum fan-out load restriction, Design Compiler compares the calculated fan-out load value with the pin's max_fanout value. The command set_max_fanout sets the maximum number of fan-out allowed on input port of design. The command to set the fan-out load on an output port of a design is set_fan-out_load.

iii. Capacitance Constraints: The maximum capacitance constraint is used to control capacitance directly. Design Compiler calculates the capacitance on a net by adding the wire capacitance to the capacitance of the pins attached to the net. To determine if the net meets the capacitance restrictions, Design Compiler compares the calculated capacitance value with the pin's max_capacitance value. To change or add to the maximum capacitance restriction specified in a technology library the set_max_capacitance command is used. The set_max_capacitance command sets a maximum capacitance for the nets attached to the named ports or to all the nets in a design by setting the max_capacitance attribute on the specified objects.

b. Optimization Constraints

These Constraints are used to optimize the design. Timing constraints specify the required performance of the design.

To set the timing constraints clock, clock latency, I/O delay etc have to be defined.

i. Defining a Clock: For synchronous designs, the clock period is the most important constraint, because it constrains all register-to-register paths in the design. This command create_clock is used to define a clock object with a particular period and waveform. The –period option defines the clock period, while the –waveform option controls the duty cycle and the starting edge of the clock. This command is applied to a pin or port, object types.

ii. Specifying Clock Network Delay: By default, Design Compiler assumes that clock networks have no delay (ideal clocks). The set_clock_latency command to specify timing information about the clock network delay.

iii. Input Delay: It specifies the input arrival time of a signal in relation to the clock. It is used at the input ports; to specify the time it takes for the data to be stable after the clock edge. The set_input_delay command defines the arrival times for input ports.

iv. Output Delay: It specifies the time it taken by the data to be available before the clock edge. The set_output_delay command defines the required output arrival time.

c. Area Constraints

The set_max_area command specifies the maximum area for the current design by placing a max_area attribute on the current design. Specify the area in the same units used for area in the technology library. Design area consists of the areas of each component and net. The Unknown components, components with unknown areas and technology-independent generic cells are ignored when Design Compiler calculates design area.

7.2.5 Optimization of design

After selecting compile strategy and constraining the design, based on the selected strategy, design can be optimized. There are many ways in which a designer can tweak his/her design to obtain optimum performance results. Several of the more general often used tweaks for performance optimization are as follows [32]:

a. Compilation with Map Effort High Option

In general, a rule of thumb to remember is that moving from map effort medium to map effort high compilation can improve design performance by about 10%.

b. Weight Factor

For designs that still cannot meet timing requirements even with a map effort high compilation option, the designer can use the group path command to group timing critical paths and set a weight factor on these critical paths. The larger the value of the weight, the more effort will Design Compiler use to try to optimize that path. This command allows a designer to prioritize critical paths for optimization.

c. Logical Flattening of a Design

Logical flattening of a design can be used to break the hierarchy of a design. All logic gates for that particular design will be at the same level of hierarchy. This would allow Design Compiler

to try to optimize those logic gates to gain better performance and area utilization. Design Compiler during optimization must maintain the integrity of block interface/ports and therefore is not able to optimize across the hierarchical boundary. This option of logical flattening is used for hierarchical designs. However, this option is not suitable for usage if the hierarchical design is large. Too huge a design will take up considerable computing resources (for example, a long time to compile), thus preventing Design Compiler from performing a good optimization.

d. Register Balancing

Register balancing is a very useful command when it comes to optimizing designs that are made up of pipelines. The concept here is to allow Design Compiler to move logic from one stage of the pipeline to another. This would allow Design Compiler the flexibility to move logic away from pipeline stages that are overly constrained to pipeline stages that have additional timing.

7.2.6 Analyzing and Debugging the Design

The reports generated by Design Compiler are used to analyze and debug the design. The reports both before and after compiling the design can be generated. The reports generated before compiling is used to check the attributes, constraints, and design rules are set properly. The reports generated reports after compiling is used to analyze the results and debugging the design.

7.2.7 Saving the Design

Design Compiler can save a database in different formats. A common format in which to save a synthesized database is the Synopsys DB format.

CHAPTER 8

SIMULATION AND SYNTHESIS RESULTS

This chapter introduces the simulation and synthesis results of Adder and MAC Unit synthesized by Xilinx ISE.

8.1 Adder

8.1.1 Synthesis Results on FPGA and Synopsis

a. Carry Select Adder

The table 8.1 shows the various synthesis results for different size Carry Select Adder.

Table 8.1: Synthesis Results of Carry Select Adder(Without Pipelining)

Size	4-bits	8-bits	16-bits	32-bits	64-bits
No. of Slices	4	9	19	39	78
4 input LUTs	8	17	35	71	143
Bonded IOs	14	26	50	98	194
Level of Logic	6	9	15	27	66
Logic Delay (%)	74.6	69.4	65.2	62.1	61.8
Route Delay (%)	25.4	30.6	34.8	37.9	38.2
Max.Comb. Delay (ns)	9.794	13.612	21.072	35.992	70.909

b. Carry Skip Adder

The table 8.2 shows the various synthesis results for different size Carry Skip Adder.

Table 8.2: Synthesis Results of Carry Skip Adder

Size	4-bits	8-bits	16-bits	32-bits	64-bits
No. of Slices	8	15	30	61	120
4 input LUTs	14	28	55	112	223
Bonded IOs	14	26	50	98	194
Level of Logic	7	13	24	44	90
Logic Delay (%)	75.2	70.6	66.3	63.5	63.2
Route Delay (%)	24.8	29.4	33.7	36.5	36.8
Max.Comb. Delay (ns)	9.629	14.626	24.368	43.412	80.923

c. Analysis

For addition of larger operands CSLA should be used. However CSLA has more LUTs than other adder architectures but it has very less combinational path delay. So CSLA is the best adder architecture where the operands size is large.

8.2 Multiply and Accumulator

The MAC unit using different adder architectures like carry chain adder, carry look ahead adder, carry select adder and carry skip adder is synthesized for different size like 16-bit and 32-bit on FPGA and Design Compiler.

8.2.1 32-bit Multiply and Accumulator

a. Synthesis Results on FPGA

The table 8.3 shows the various synthesis results of 32-bit MAC Unit for different final stage and accumulator adder.

Table 8.3: FPGA Synthesis Results of 32-bit MAC Unit

Adder Type	CSLA	CSKA
No. of Slices	1871	2114
Slice Flip-Flops	237	229
4 input LUTs	3646	4084
LUT Utilization %	39	43
Bonded IOs	131	131
Level of Logic	47	32
Min.Clock Period (ns)	24.127	28.165
Max. Frequency (MHz)	41.447	35.505
Offset In before clock (ns)	2.057	2.057
Offset Out after clock (ns)	24.557	28.972
Total Power (mW)	279.26	290.67

b.Analysis

The FPGA Results shows that the LUTs used in 32-bit MAC with CSLA is lesser than other MAC architectures and has higher frequency of operations. So 32-bit MAC with CSLA is best in term of LUTs and frequency of operations but it requires more power supply than other MAC architectures.

The FPGA synthesis results shows that 32-bit Conventional Booth Wallace MAC Unit has maximum frequency of 41.447 MHz and 3646 LUTs and 279.26 mW power supply.

8.3 32-bit Conventional Booth Wallace MAC with Carry Select Adder

8.3.1 Synthesis Results on FPGA and Design Compiler

The table 8.4 and 8.5 shows the synthesis result of 32-bit Conventional Booth Wallace MAC unit on FPGA and Design Compiler.

Table 8.4: FPGA Synthesis Results of 32-bit Conventional Booth Wallace MAC Unit

No. of Slices	1871
Slice Flip-Flops	237
4 input LUTs	3646
LUT Utilization %	39
Bonded IOs	131
Level of Logic	47
Min.Clock Period (ns)	24.127
Max. Frequency (MHz)	41.447
Offset In before clock (ns)	2.057
Offset Out after clock (ns)	24.557
Total Power (mW)	279.26

Table 8.5: DC Synthesis Results of 32-bit Conventional Booth Wallace MAC Unit

CLK Period(ns)	3.97
Levels of Logic	51
Critical Path Length	3.78
Critical Path Slack	0.00
Total Negative Slack	0.00
No. of Violating Paths	0
Leaf Cell Count	13613
Comb. Area(C)	99313.200
Non Comb.Area (S)	2686.768
C/S Ratio	36.96
Cell or Design Area	101999.968
Total Number of Nets	13681
Total Dynamic Power	10.085 mW
Cell Leakage Power	74.829 uW
Frequency (MHz)	251.88

8.3.2 Analysis

The FPGA synthesis results shows that 32-bit Conventional Booth Wallace MAC Unit has maximum frequency of 41.447 MHz and 3646 LUTs and 279.26 mW power supply. The

design compiler results shows that the MAC met all the constraints at the clock period 3.97 ns or at the frequency 251.88 MHz

8.4 32-bit Pipelined Booth Wallace MAC Unit with Carry Select Adder

8.4.1 Simulation Results

a : Input Data 32-bit

b : Input Data 32-bit

clk :Input Clock

z: : Output Data 64-bit

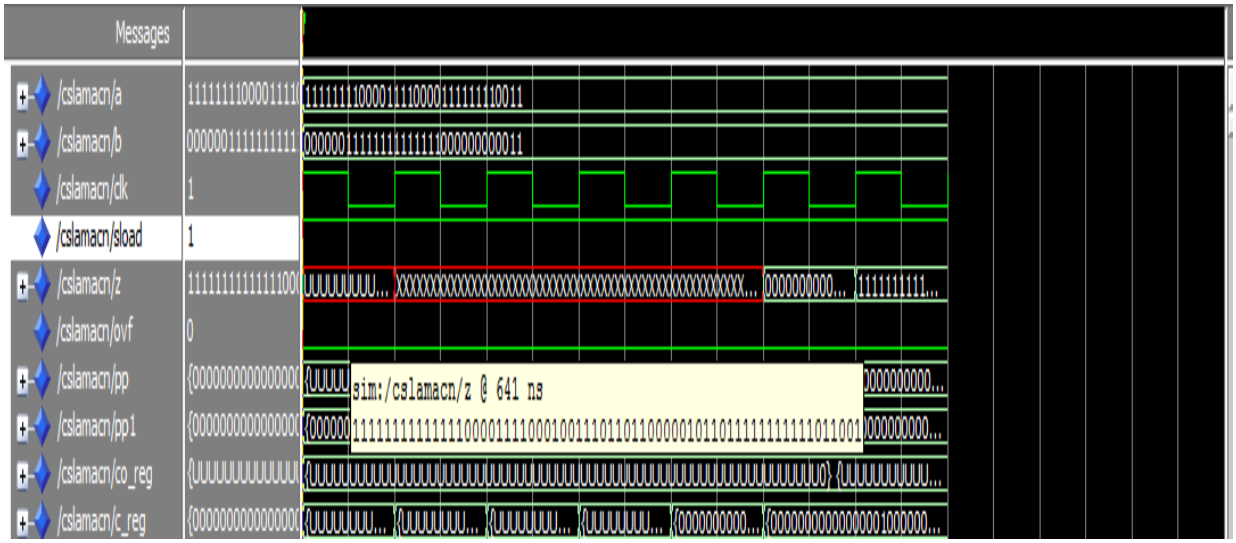


Figure 8.1: Simulated Waveform of 32-bit Pipelined Booth Wallace MAC Unit

There is latency of 5 clock cycle means we get the final output of MAC in 6th clock cycle.

8.4.2 Synthesis Results on FPGA and Design Compiler

The table 8.6 and 8.7 shows the synthesis result of 32-bit Pipelined Booth Wallace MAC Unit with carry select adder in final stage of multiplier as well as in accumulator on FPGA and Design Compiler.

Table 8.6: FPGA Synthesis Results of 32-bit Pipelined Booth Wallace MAC Unit

No. of Slices	1971
Slice Flip-Flops	1399
4 input LUTs	3298
LUT Utilization %	35
Bonded IOs	131
Level of Logic	8
Min Clock Period (ns)	10.5
Max. Frequency (MHz)	95.23
Offset In before clock (ns)	1.946
Offset Out after clock (ns)	16.073
Total Power (mW)	531.42

Table 8.7: DC Synthesis Results of 32-bit Pipelined Booth Wallace MAC Unit

CLK Period(ns)	2.43
Levels of Logic	22
Critical Path Length	2.23
Critical Path Slack	0.00
Total Negative Slack	0.00
No. of Violating Paths	0
Comb. Area(C)	56189.280
Non Comb.Area (S)	23781.072
C/S Ratio	2.36
Cell or Design Area	79970.352
Total Number of Nets	11215
Frequency (MHz)	411.52
Total Dynamic Power (mW)	17.3468
Cell Leakage Power (uW)	37.5806

8.4.3 Analysis

It is clear from the table that in Pipelined MAC the Max. Frequency of operation reach at 95.23 MHz which is more than double from the frequency of non pipelined MAC unit but with the pipelining the latency increases and slice flip-flops also increased as they are used as register. Power supply also increased to a large extend in case of pipelining. The design compiler results shows that the MAC met all the constraints at the clock period 2.43 ns or at the frequency 411.52 MHz

8.4.4 RTL View

The RTL view of 32-bit Pipelined Booth Wallace MAC on FPGA is shown in Figure 8.2.

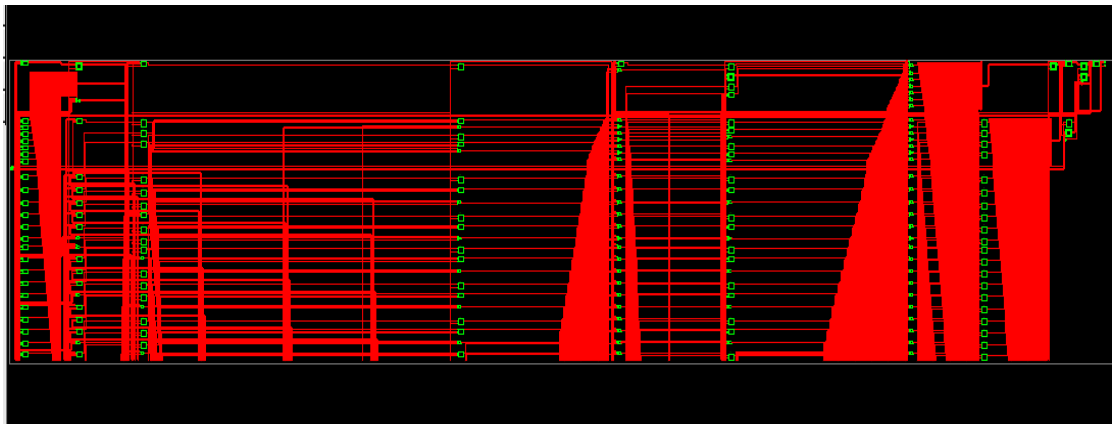


Figure 8.2: RTL View of 32-bit Pipelined Booth Wallace MAC Unit

8.5 32-bit Pipelined Booth Wallace MAC Unit with Pipelined Variable Stage Carry Select Adder

8.5.1 Simulation Results

a : Input Data 32-bit

b : Input Data 32-bit

clk :Input Clock

z: : Output Data 64-bit

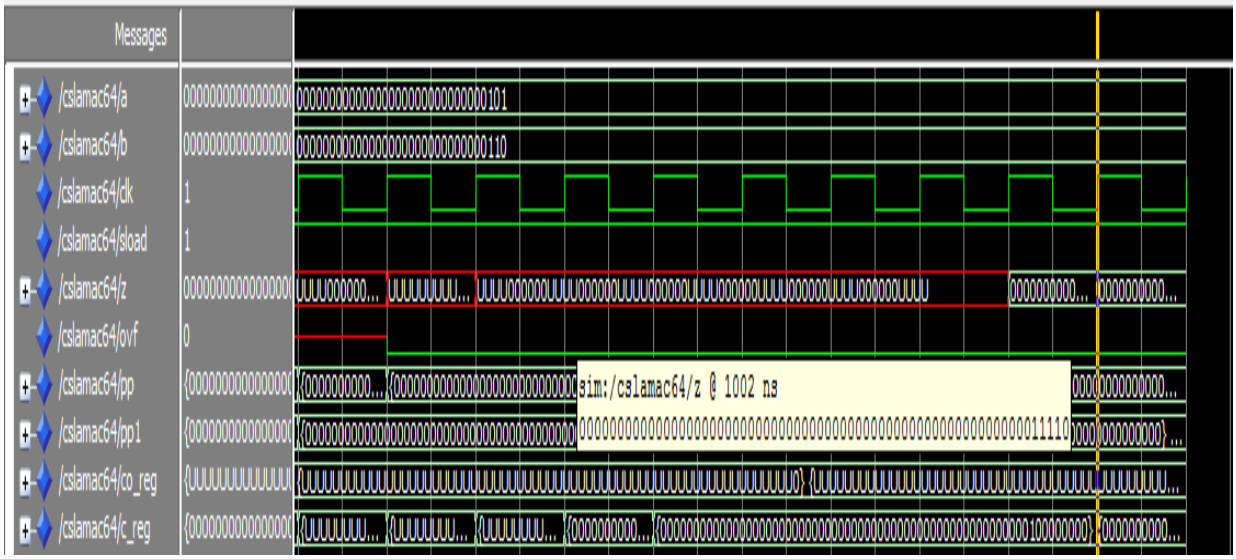


Figure 8.3: Simulated Waveform of 32-bit Pipelined Booth Wallace MAC Unit With Pipelined Variable Stage Carry Select Adder (PVCSLA)

8.5.2 Synthesis Results on FPGA and Design Compiler

The table 8.8 and 8.9 shows the synthesis result of 32-bit Pipelined Booth Wallace MAC Unit with Pipelined variable carry select adder (ripple carry adder of sizes 4-bit and 6-bit are used) in final stage of multiplier as well as in accumulator on FPGA and Design Compiler.

Table 8.8: FPGA Synthesis Results of 32-bit Pipelined Booth Wallace MAC Unit With Pipelined Variable Stage Carry Select Adder

No. of Slices	1811
Slice Flip-Flops	1851
4 input LUTs	3361
LUT Utilization %	35
Bonded IOs	131
Level of Logic	8
Min.Clock Period (ns)	6.328
Max. Frequency (MHz)	158.035
Offset In before clock (ns)	1.810
Offset Out after clock (ns)	4.990
Total Power (mW)	630.12

Table 8.9: DC Synthesis Results of 32-bit Pipelined Booth Wallace MAC Unit with Pipelined Variable Stage Carry Select Adder

CLK Period(ns)	1.88
Levels of Logic	38
Critical Path Length	1.77
Critical Path Slack	0.00
Total Negative Slack	0.00
No. of Violating Paths	0

Comb. Area(C)	214190.10
Non Comb.Area (S)	117176.14
Cell or Design Area	331353.78
Total Number of Nets	12458
Frequency (MHz)	531.91
Total Dynamic Power (mW)	181.2742
Cell Leakage Power (uW)	39.7328

8.5.3 RTL View

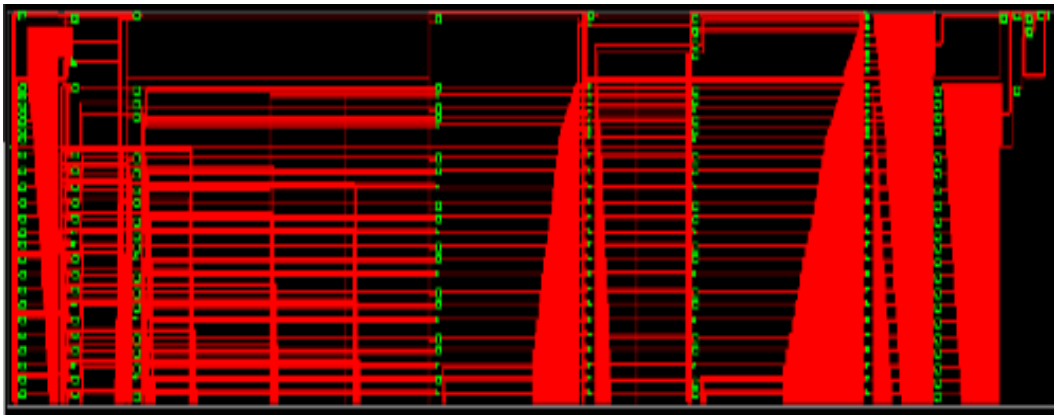


Figure 8.4: RTL View of 32-bit Pipelined Booth Wallace MAC Unit(With Pipelined Variable Stage Carry Select Adder)

8.5.4 Analysis

It is clear from the table that in this Pipelined MAC the Max. Frequency of operation reach at 158.035 MHz which is much more than frequency of pipelined MAC unit ,but in this MAC unit the latency further increases and slice flip-flops also increased as they are used as register. Power supply also increased to a large extend in case of pipelining. The design compiler results shows that the MAC met all the constraints at the clock period 1.88 ns or at the frequency 531.91MHz.

CHAPTER 9

CONCLUSION & FUTURE SCOPE

This chapter conclude what have been done in thesis and what can be done in future.

9.1 Conclusion

By analyzing the results on Spartan XC3S500-4FG320 device, for large size operands like for 16, 32 or 64 bits operands CSLA architecture has minimum combinational path delay as compared to other adder architectures , but for small size operands up to 16 bits CLA architecture has less combinational path delay as compared to other adder architectures. Table 9.1 and 9.2 shows the synthesis result of Carry select adder and Carry skip Adder architectures .

Table 9.1: Synthesis Results of Carry Select Adder

Size	4-bits	8-bits	16-bits	32-bits	64-bits
No. Of Slices	6	14	29	56	113
4 input LUTs	11	26	54	104	210
Bonded IOs	14	26	50	98	194
Level of Logic	6	7	11	19	35
Logic Delay (%)	73.7	70.1	65.1	61.9	57.4
Route Delay (%)	26.3	29.9	34.9	38.1	42.6
Max.Comb. Delay (ns)	9.393	11.434	16.626	26.608	48.31

Table 9.2: Synthesis Results of Carry Skip Adder

Size	4-bits	8-bits	16-bits	32-bits	64-bits
No. of Slices	8	15	30	61	120
4 input LUTs	14	28	55	112	223
Bonded IOs	14	26	50	98	194
Level of Logic	7	13	24	44	90
Logic Delay (%)	75.2	70.6	66.3	63.5	63.2
Route Delay (%)	24.8	29.4	33.7	36.5	36.8
Max.Comb. Delay (ns)	9.629	14.626	24.368	43.412	80.923

After that simple 32 bits Booth Wallace MAC unit(without pipelining) are designed using two types of adder architecture (Carry Skip Adder & Carry Select Adder) and have been implemented on Spartan XC3S500-4FG320 device and the synthesis results on FPGA verifies that MAC with CSLA is much faster as compared to MAC with Carry Skip Adder.. Table 9.3 shows that MAC with CSLA is much faster for 16 bits and 32 bits.

Table 9.3: FPGA Synthesis Results of 32-bit MAC Unit.

Adder Type	CSLA	CSKA
No. of Slices	1871	2114
Slice Flip-Flops	237	229
4 input LUTs	3646	4084
LUT Utilization %	39	43
Bonded IOs	131	131
Level of Logic	47	32
Min.Clock Period (ns)	24.127	28.165
Max. Frequency (MHz)	41.447	35.505
Offset In before clock (ns)	2.057	2.057
Offset Out after clock (ns)	24.557	28.972
Total Power (mW)	279.26	290.67

After that a 32-bit Pipelined Booth Wallace MAC with CSLA in final stage of multiplier and accumulator and has been implemented on Spartan XC3S500-4FG320 device and its results compared with the conventional MAC .

Table 9.4: Synthesis Results of Conventional and Pipelined 32-bit Booth Wallace MAC Unit

MAC	Conventional MAC	Pipelined MAC
Size	32 bits	32 bits
Slice Flip-Flops	237	1399
LUTs	3646	3298
Delay(ns)	24.1	10.5
Frequency (MHz)	41.5	95.23 (x2.29 improvement)
Power (mW)	279.2	531.42

Table 9.4 shows that Pipelined Booth Wallace MAC has double speed than a conventional Booth Wallace MAC but the power requirement in pipelined increased as compared to conventional MAC . Now we have used the Pipelined Variable Stage Carry Select Adder (used 4-bits and 6-bits ripple carry adder) for final addition of product terms and for accumulator in the above pipelined Booth Wallace MAC .And its results are compared with

the conventional as well as with Pipelined Booth Wallace MAC unit . Table shows the comparison .

Table 9.5: Synthesis Results of Conventional, Pipelined and with Pipelined Variable Stage CSLA 32-bit Booth Wallace MAC Unit

MAC	Conventional MAC	Pipelined MAC	Pipelined MAC with Pipelined Variable Stage Carry Select Adder
Size	32 bits	32 bits	32 bits
Slice Flip-Flops	237	1399	1851
LUTs	3646	3298	3361
Delay(ns)	24.1	10.5	6.328
Frequency (MHz)	41.5	95.23 (x2.29 improvement)	158.035 (x3.8080723 improvement)
Power (mW)	279.2	531.42	630.12

The table 9.5 shows that this final MAC unit (with Pipelined Variable Stage Carry Select Adder) is 3.8080723 times better than conventional MAC and 1.65 times better than Pipelined Booth Wallace MAC unit

9.2Future Scope

In this thesis work all design and synthesizing is done by considering the Speed as a critical parameter.

- The speed can be further increased if more and more different stages (different bits carry ripple adder) of variable carry select adder are used.
- In the future, if we want to optimize this MAC unit for low power, we must use carry skip adder.

REFERENCES

[1] Koc, C.K., "RSA Hardware Implementation", RSA Laboratories, RSA Data Security, Inc. ,1996.

[2] B. Parhami, "Computer Arithmetic, Algorithm and Hardware Design", Oxford University Press, New York, pp.73-137, 2000.

[3] A. Weinberger; J.L. Smith, "A Logic for High-Speed Addition", National Bureau of Standards, Circulation 591, pp. 3-12, 1958.

[4] Neil. H. E. Weste , "Principle of CMOS VLSI Design", Adison-Wesley, 1998.

[5] Raahemifar, K. and Ahmadi, M., "Fast carry look ahead adder", IEEE Canadian Conference on Electrical and Computer Engineering, 1999.

[6]Kantabutra,V., "Designing Optimum One-Level Carry-Skip Adders", IEEE Transactions on Computers, pp.759-764, June, 1993.

[7] Chan, P. K., Schlag, M. D. F., Thomborson, C. D. and Oklobdzija, V. G.,"Delay Optimization of Carry-skip Adders and Block Carry-lookahead Adders", proceedings, IEEE Symposium on Computer Arithmetic, 1991.

[8] M. C. Wen, S. J. Wang, Y.N. Lin,"Low-Power parallel multiplier with column bypassing " , ELECTRONICS LETTERS, vol. 41, no. 10, 12th May, 2005.

[9]P. Devi and A. Girdher, "Improved Carry Select Adder with Reduced Area and Low Power Consumption", International Journal of Computer Applications (0975– 8887) vol. 3 – No.4, June, 2010.

[10] A.D. Booth, "A signed binary multiplication technique", Quart. J. Mech. Appl. Marh, vol. 4, pp. 236-240, 1951.

[11] O. L. MacSorley, "High speed arithmetic in binary computers", Proc.IRE,vol.49,pp. 67-91, 1961.

[12] A. A. A., Gutub and H. A., Tahhan, "Efficent Adders To Speedup Modular Multiplication for Cryptography", Computer Engineering Department, KFUPM, Dhahran, SAUDI ARABIA, 2001.

[13] Razaidi Hussin, Ali Yeon Md. Shakaff, Norina Idris, Zaliman Sauli, Rizalafande Che IIsmail, and Afzan Kamaraudin, "An efficient modified Booth multiplier architecture", International Conference on Electronic Design, 978-1-4244-2315- 6/08, 2008 IEEE.

[14] Parhami, Behrooz, "Computer Arithmetic: Algorithms and Hardware Designs", Oxford University Press 2000.

[15] Computer System Architecture, 3rd edition By....M.MORRIS MANO

[16] Gong Renxi, Zhang Hainan, Meng Xiaobi,Gong Wenying,"Hardware Implementation of a High Speed Floating Point Multiplier Based on FPGA",2009 4th International Conference on Computer Science & Education.

[17] Computer Organization And Architecture 5th edition By...WILLIAM STALLINGS

[18] http://en.wikipedia.org/wiki/Carry-select_adder

[19] http://en.wikipedia.org/wiki/Booth's_multiplication_algorithm

[20] Fayed, Ayman A., Bayoumi, Magdy A., "A Merged Multiplier-Accumulator for high speed signal processing applications", IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp 3212 -3215, 2002.

[21] Oklobdzija, V.G.; Villeger, D.; Liu, S.S.; "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach" IEEE Transactions on Computers, vol.45, pp 294 – 306, 1996.

[22] Fayed, Ayman A., Bayoumi, Magdy A., "A Merged Multiplier-Accumulator for high speed signal processing applications", IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp 3212 -3215, 2002.

[23] DESIGN AND IMPLEMENTATION OF 32-BIT HIGH SPEED BOOTH WALLACE MAC UNIT

[24] Abdelgawad, A., Bayoumi, M., "High Speed and Area-Efficient Multiply Accumulate (MAC) Unit for Digital Signal Processing Applications", IEEE International Symposium on Circuits and Systems, pp . 3199 – 3202, 2007.

[25] Deschamps, J.P.; Bioul, G.J.A; Sutter,G.D.; "Synthesis of Arithmetic Circuits"; FPGA, ASIC and Embedded Systems, John Wily & Sons Inc., Publication, 2006.

[26] Gina R. Smith, "FPGAs 101: Everything you need to know to get started", Elsevier , 2010.

[27] "Spartan-3E FPGA Starter Kit Board User Guide", UG230 (v1.1) June 20, 2008.

[28] Deming Chen, Jason Cong, and Peichan Pan, "FPGA Design Automation: A Survey", Foundations and Trends in Electronic Design Automation, vol. 1, Issue 3, November, 2006.

[29] S.Shah, A.J.Kbalili,D.Al-Khabili, "Comparison of 32-bit multipliers for various performance measures", The 12th International Conference on Microelectronics Tehran, Oct. 31- Nov. 2, 2000.

[30] A. Dandapat, S. Ghosal, P.Sarkar, D.Mukhopadhyay, "A 1.2ns 16X16-bit binary multiplier using high speed compressors", International Journal of Electrical and Electronics Engineering 4:3, 2010.

[31] Design Compiler User Guide v1999.10.

[32] Weng Fook Lee, "VHDL Coding and Logic Synthesis with Synopsys" Academic Press pp.147-227, 2000.