# DESIGN AND IMPLEMENTATION OF
# 32-BIT HIGH SPEED BOOTH WALLACE MAC UNIT

A thesis submitted in partial fulfillment of the requirements for the award of degree of

**Master of Technology**

in

**VLSI Design & CAD**


Submitted By:

**Naveen Kumar**

**Roll No. 600961012**

Under the Guidance of

**Ms. Manu Bansal**

**Assistant Professor**



**Department of Electronics & Communication Engineering**

**Thapar University, Patiala**


**July 2011**

# CERTIFICATE

I hereby declare that the work which is being presented in the thesis entitled, **"Design and Implementation of 32-bit High Speed Booth Wallace MAC Unit"** in partial fulfillment of the requirement for the award of degree of M.Tech. (VLSI Design & CAD) at Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Manu Bansal, Assistant Professor, ECED.

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

Naveen Kumar

**Date:** July 15, 2011

**Naveen Kumar**
**Roll.No.600961012**

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

Manu Bansal

**Ms. Manu Bansal**
**Assistant Professor**
**ECED,Thapar University**

**Counter signed by:**

A.K.Chatterjee 20.7.11

**Dr. A. K. Chatterjee**
**Professor & Head**
**ECED, Thapar University**
**Patiala-147004**

**Dr. S. K. Mohapatra**
**Dean of Academic Affairs**
**Thapar University**
**Patiala-147004**

# ACKNOWLEDGEMENT

I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I acknowledge with gratitude and humility my indebtedness to **Ms. Manu Bansal, Assistant Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to **Head of the Department, Dr. A. K. Chatterjee** as well as **PG Coordinator, Dr. Alpana Agarwal, Assistant Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

I would also like to thank Ms. Sakshi Grover Bajaj (Assistant Professor), Mohit Bhasin , Anil Rawat and Simran Kaur who were always there at the need of the hour and provided me all the help, which I required for the completion of my thesis.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

**Naveen Kumar**

# 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.

Thesis starts with the study of different 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, simulated and synthesized on FPGA using Xilinx ISE and Synopsys Design Compiler. 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 Carry Select Adder is used and the pipelining is done in the Booth Multiplier and Wallace Tree. To check and verify its functionality on FPGA, LCD and Keyboard interfacing also has been done.

This 32-bit pipelined Booth Wallace MAC described in VHDL and synthesized the circuit using 90 nm standard cell library on FPGA and Design Compiler. This MAC has higher speed than conventional or non pipelined Booth Wallace MAC.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ABBREVIATIONS

| | |
|---|---|
| ASIC | Application Specific Integrated Circuits |
| BCCOM | Best Case Commercial |
| 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 |
| TCCOM | Typical Case Commercial |
| UCF | User Constraints File |
| VHDL | Very High Speed Integrated Circuits HDL |
| VLSI | Very Large Scale Integration |
| WCCOM | Worst Case Commercial |

# 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.

## 1.1 Motivation

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. Minimizing power consumption for digital systems involves optimization at all levels of the design. This optimization includes the technology used to implement the digital circuits, the circuit style and topology, the architecture for implementing the circuits and at the highest level the algorithms that are being implemented.

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. Therefore, it is needed to design a low-power high speed MAC unit. It is often the crucial circuit component if die area, power dissipation, and especially operation speed are of concern.

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). Furthermore, they should provide enough flexibility in order to accommodate custom timing and area constraints as well as to allow the implementation of customized adders.

### 1.2 Problem Statement

The primary objective of thesis 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, simulated and synthesized on FPGA using Xilinx ISE and Synopsys Design Compiler. On the basis of their synthesized results, one of the adder with less delay and minimum area (CSLA) is chosen.

The different MAC architectures are studied and designed in which the multiplication is done using the Booth Wallace Multiplier. But in the final stage addition of multiplier and in accumulator the different adder architectures like CCA, CLA, CSLA, CSKA are used for 16-bit and 32-bit MAC unit. All theses MAC units are synthesized on FPGA and Design Compiler.

After analyzing the results; a 32-bit Pipelined Booth Wallace MAC unit with Carry Select Adder in final stage addition of multiplier and accumulator has been designed and dumped on Spartan 3E kit. To check and verify its functionality on FPGA, LCD and Keyboard interfacing with FPGA has been done. After that static timing analysis with input constraints and Floor plan has been done for this MAC unit using Xilinx ISE12.4.

This MAC Unit also synthesized with 90 nm standard cell library on Synopsys Design Compiler and their results are compared with conventional or non-pipelined MAC and with some other MAC units. This design has been automated using TCL Script on Design Compiler. After that various optimization technique to make the MAC faster and effect of environment, design and timing constraints and have been analyzed.

### 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.

**Chapter 2** starts with a literature review of the various adder architectures and the optimization techniques.

**Chapter 3** starts with a literature review of Multiply and Accumulate Unit. The various technique to speed-up the multiplication process in MAC has been studied. A 32-bit pipelined Booth Wallace MAC architecture also designed in this chapter.

**Chapter 4** is mainly concerned with the FPGA. The synthesis flow of FPGA in detail like synthesis, translate, map, PAR, programming file generation is discussed in this chapter. Keyboard and LCD Interfacing with Spartan-3E kit is also discussed in this chapter.

**Chapter 5** 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 in this chapter. The design ware libraries, wire load models, compile strategy, timing constraints, design constraints, optimization constraints, timing issues etc. have been explained in this chapter.

**Chapter 6** represent the simulation, synthesis results and comparisons table of different adder architectures and different MAC units. Also the results of full design flow of 32-bit Pipelined Booth Wallace MAC Unit on FPGA and Design Compiler. Also static timing analysis have been discussed for 32-bit Pipelined Booth Wallace MAC Unit. Finally a conclusion is given in **Chapter7**.

# 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. Figure 2.1 shows the basic adder structures and their relationships.

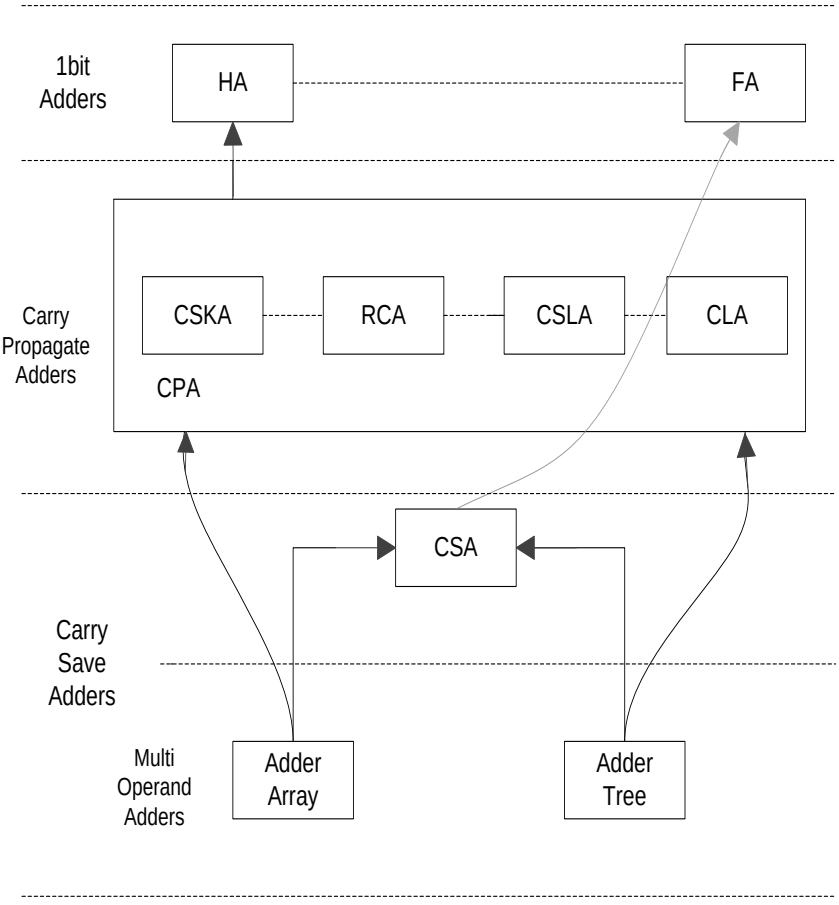



**Figure 2.1:** Adder Structures

## 2.1 Carry Propagate Adder

A carry-propagate adder adds two k-bit operands, $X = (x_{k-1}, x_{k-2} \ldots, x_0)$ and $Y = (y_{k-1}, y_{k-2} \ldots, y_0)$ 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 $S= (s_{k-1}, s_{k-2}, \ldots, s_0)$ 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 carry-in 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.

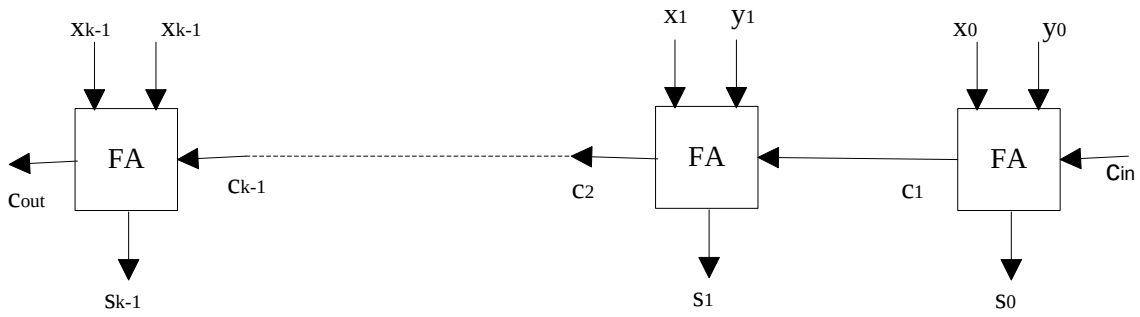



**Figure 2.2:** Ripple carry implementation of CPA

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.

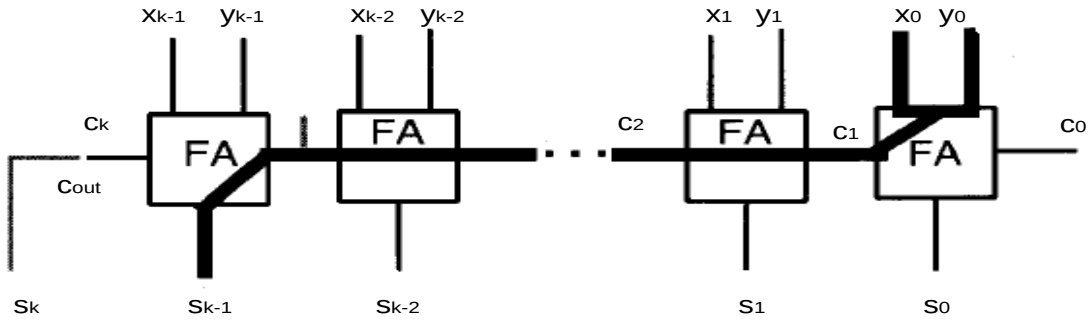



**Figure 2.3:** Critical paths in a k-bit RCA [2]

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. Thus the expression for latency of k-bit ripple-carry adder is:

$$T_{ripple\ -add} = T_{FA}(x, y \rightarrow c_{out}) + (k-2) * T_{FA}(c_{in} \rightarrow c_{out}) + T_{FA}(c_{in} \rightarrow s) \qquad (2.1)$$

Where $T_{FA}$ (input→output) represents the latency of a full-adder on path between its specified

input and output. As an approximation to the foregoing, it can be concluded that the latency of a ripple-carry adder is $kT_{FA}$[2]. It is seen that the latency grows linearly with k, making the ripple-carry design undesirable for large k or for high-performance arithmetic unit. However a carry completion detection adder takes advantage of the $\log_2 k$ average length of the longest carry chain to add two k-bit binary number in O (log k) time on the average.

### 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:

$$g_i = x_i . y_i \tag{2.2}$$

$$p_i = x_i \oplus y_i \tag{2.3}$$

$$a_i = not(x_i + y_i) \tag{2.4}$$

$$t_i = x_i + y_i \tag{2.5}$$

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

$$c_{i+1} = g_i + c_i . p_i \tag{2.6}$$

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).

$$c_{i+1} = g_i + c_i . t_i \tag{2.7}$$

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.

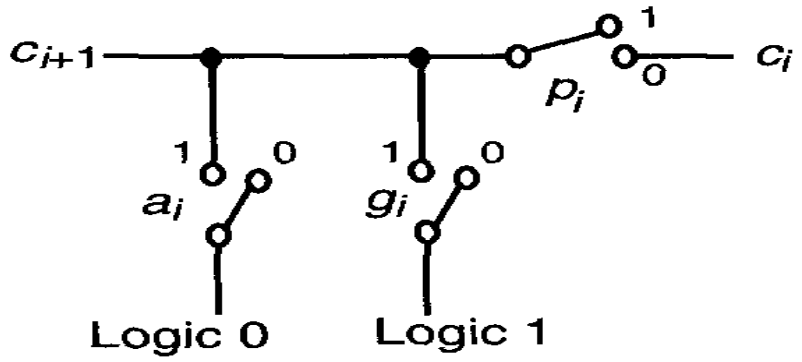



**Figure 2.4:** One stage Manchester carry chain [2]

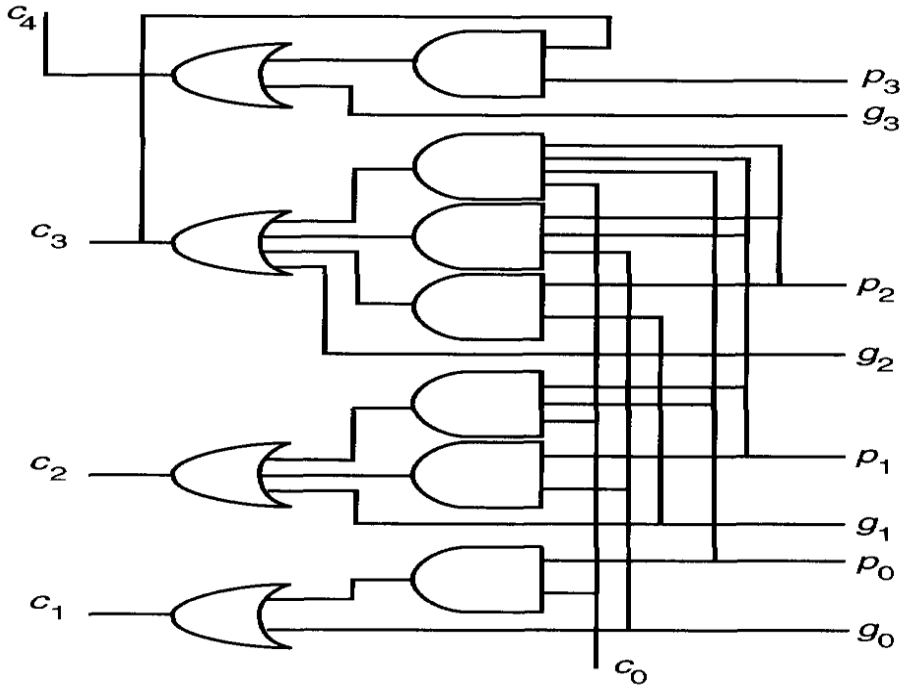



**Figure 2.5:** Carry-look ahead logic [2]

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-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][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.

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.

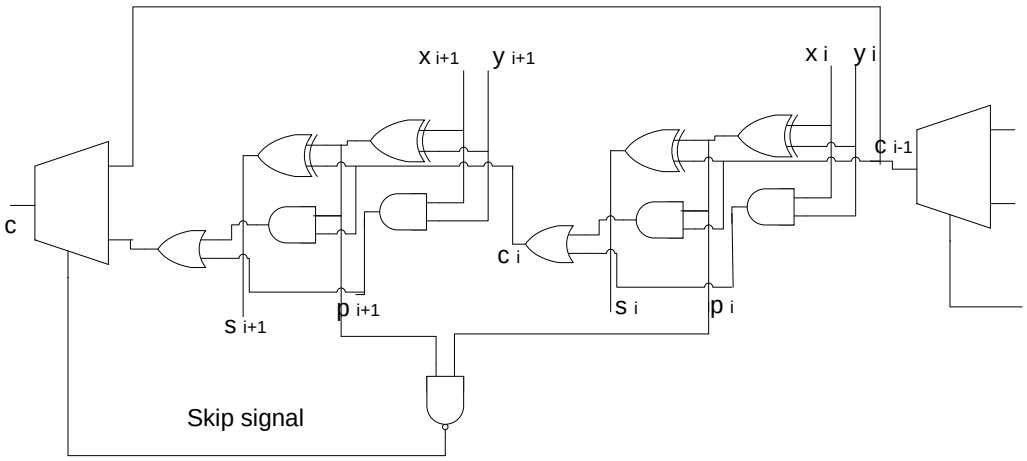



**Figure 2.6:** Carry-skip logic using multiplexers [12]

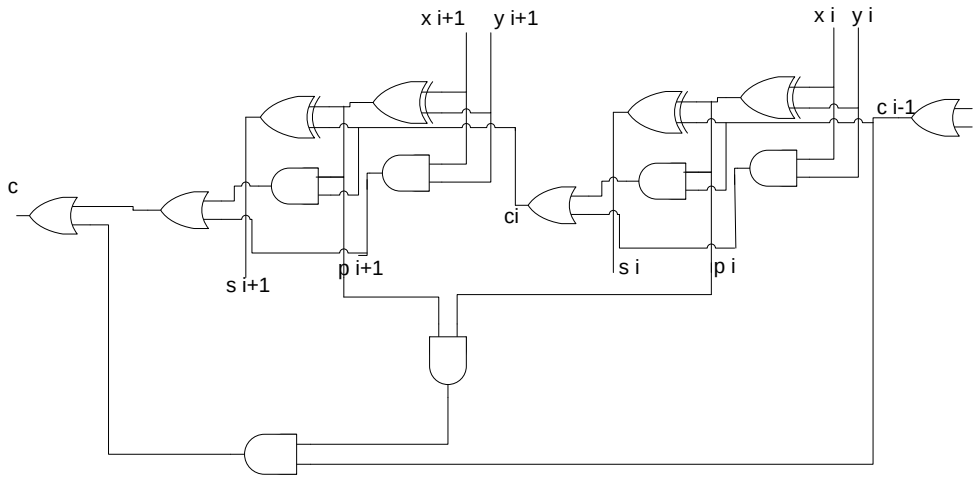



**Figure 2.7:** Carry-skip logic using gates [12]

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 [9]. 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.

In carry skip adder the skip and ripple delays were assumed to be equal and ripple delay was assumed to be linearly proportional to block width. But in practice it is not true. With CMOS implementation the ripple Delay in a Manchester carry chain grows as the square of block width.

The carry-skip adder has a simple and regular structure that requires an area in the order of O (k) which is hardly larger than the area required by the ripple-carry adder. The time complexity of the carry-skip adder is bounded between O (k) and O (log k). An equal block size one-level carry-skip adder will have a time complexity of O (k). Block width tremendously affects the latency of adder. Latency is directly proportional to block width [10]. More number of blocks means block width is less, hence more delay.

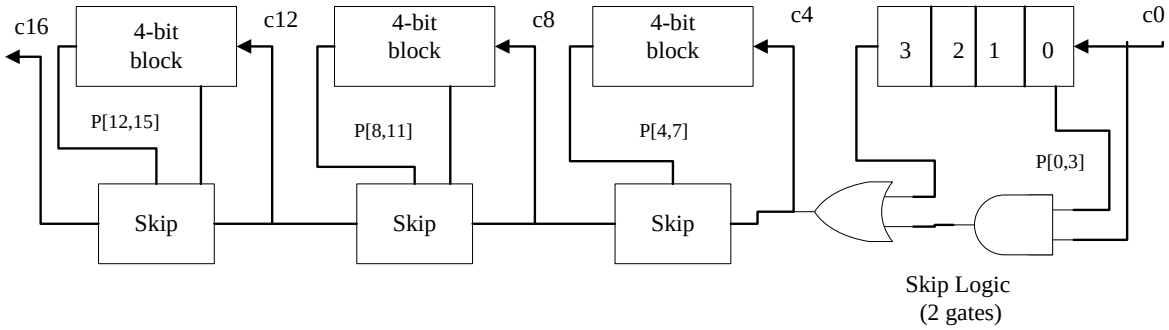



**Figure 2.8:** 16-bit carry-skip adder [9]

The idea behind Variable Block Adder [9] is to minimize the critical path delay in the carry chain of a carry-skip adder, while allowing the groups to take different sizes. In case of carry-skip adder, such condition will result in more number of skips between stages. Such adder design is called variable block design, which is tremendously used to fasten the speed of adder. In the variable block carry-skip adder design a 32-bit adder is divided in to 4 blocks or groups. The bit widths of groups are taken as: first block is of 4 bits, second is of 6 bits, third is 18 bits wide and the last group consist of most significant 4 bits [9].

### 2.1.4 Carry Select Adder

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. 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.

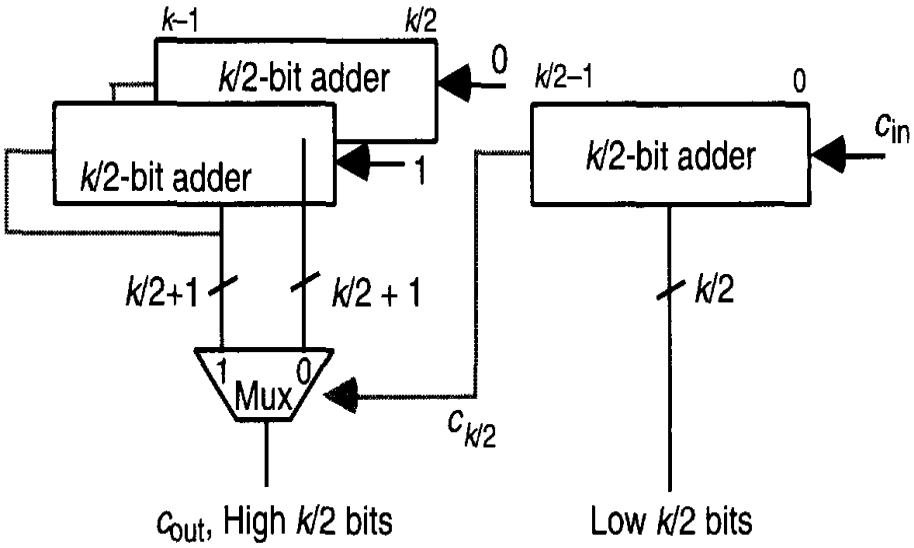



**Figure 2.9:** CSLA for k-bit numbers built from three $k_{1/2}$-bit adders [2]

A single-level carry-select adder [2] is one that combines three $k_{1/2}$-bit adders of any design into a k-bit adder as shown in Figure 2.9. One $k_{1/2}$-bit adder is used to compute the lower half of the k-bit sum directly. Two $k_{1/2}$-bit adders are used to compute the upper $k_{1/2}$-bits of the sum and the carry-out under two different scenarios: $C_{k/2} = 0$ and $C_{k/2} = 1$. The correct values for the adder's carry-out signal and the sum bits in positions $k_{1/2}$ through k - 1 are selected when the value of $C_{k/2}$ 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 ($X_0$, X1… $Xn_{-1}$) (n>2) yielding a result S in irredundant number representation with (k+ [$\log_2 n$]) bits [11]

$$S = \sum_{j=0}^{j-1} X^j \tag{2.8}$$

Multi-operand addition [2] is implicit in both multiplication and computation of vector inner products. In multiplying a multiplicand a by a k-digit multiplier x, the k partial products must be formed and then added. For inner-product computation, the component product terms obtained by multiplying the corresponding elements of the two operand vectors x and y, need to be added. Computing averages (e.g. in the design of a mean filter) is another application that requires multi-operand addition.

### 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.

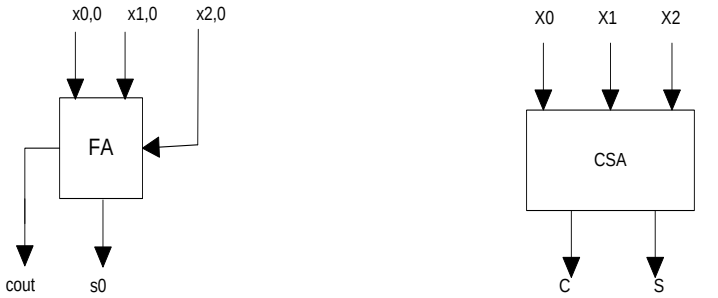



**Figure 2.10:** Block diagram of CSA

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 carry-save 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].

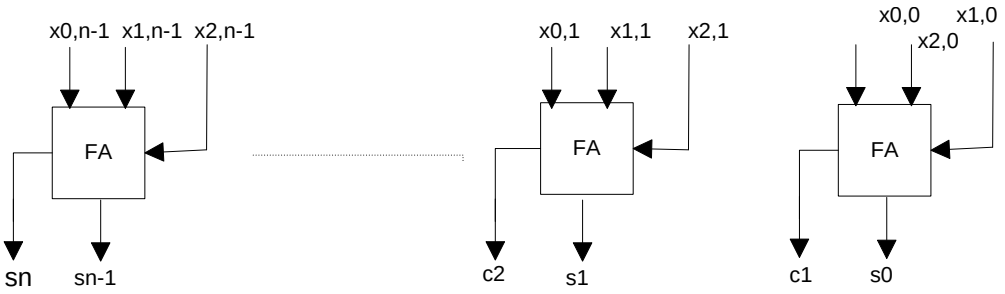



**Figure 2.11:** k-bit carry-save adders to add three k bit numbers

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.

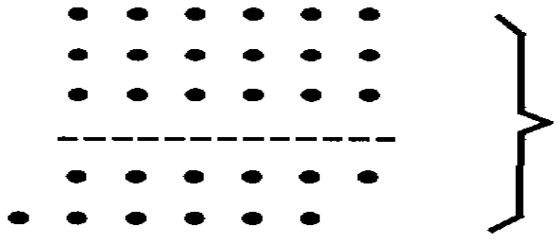


**Figure 2.12:** CSA function in dot notation [2]

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.

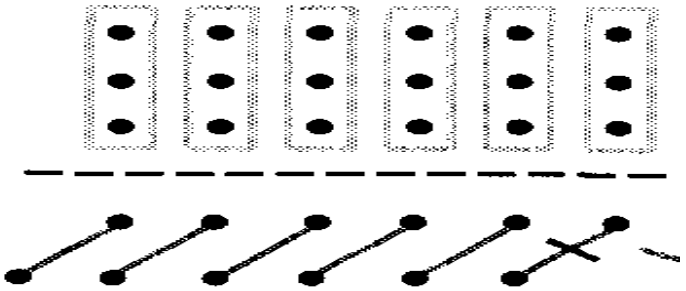


**Figure 2.13:** Specifying full-adder and half-adder blocks in dot notation [2]

Dot notation suggests another way to view the function of a carry-save adder, as converter of a radix-2 number with the digit set O [3][1] (three bits in one position) to one with the digit set O [2][1] (two bits in one position). A carry-save adder tree can reduce n binary numbers to two numbers having the same sum in O (1og n) levels. If a fast logarithmic time carry-

propagate adder is used to add the two resulting numbers, then the following results are shown for the cost and delay of n-operand addition

$$C_{\text{cany }-\text{save }-\text{multi }-\text{add}} = (n-1)C_{CSA} + C_{CPA} \quad (2.9)$$

$$T_{\text{cany }-\text{save }-\text{multi }-\text{add}} = O(\log n + \log k) \quad (2.10)$$

The needed CSAs are of various widths, but generally the widths are close to k-bits; the CPA is of width at most (k +$\log_2 n$).

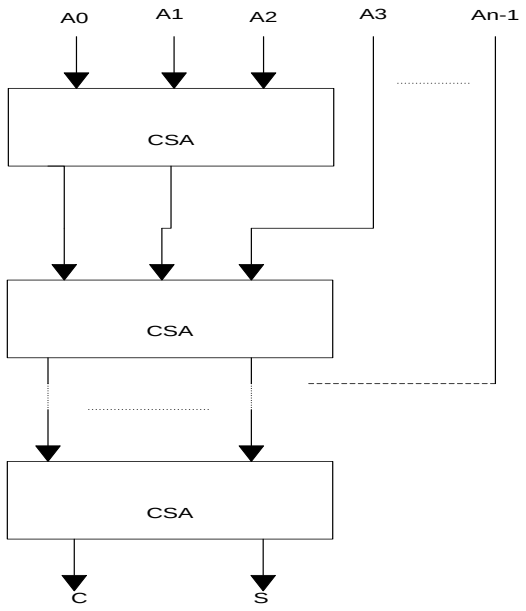



**Figure 2.14:** Adding n k-bit numbers using CSA

This is the preferred method when the operands arrive serially or must be read out from memory one by one. Note, however, that in this case both the CSA and final CPA will have to be wider. The depth of the tree is $\log_{3/2}$(n). Like before, the width of the numbers will increase as it gets deeper in the tree. At the end of the tree, the numbers will be O (k + log n) bits wide, and therefore the LCA will have a O (log (k + log n)) gate delay.

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
# MULTIPLY AND ACCUMULATOR

---

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 Carry Select Adder in the final addition stage of Multiplication and in the Accumulator .

## 3.1 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:

$$Z = A * B + Z \tag{3.1}$$

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 as shown in Figure 3.1 can be divided into two main blocks [13].

- Multiplier
- Accumulator

### 3.1.1 Multiplier

Multiplication is a mathematical operation that at its simplest is an abbreviated process of adding an integer to itself a specified number of times. A number (multiplicand) is added to itself a number of times as specified by another number (multiplier) to form a result. Figure 3.2 shows the flow for the basic multiplication technique. Each black dot represents a single digit.

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

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

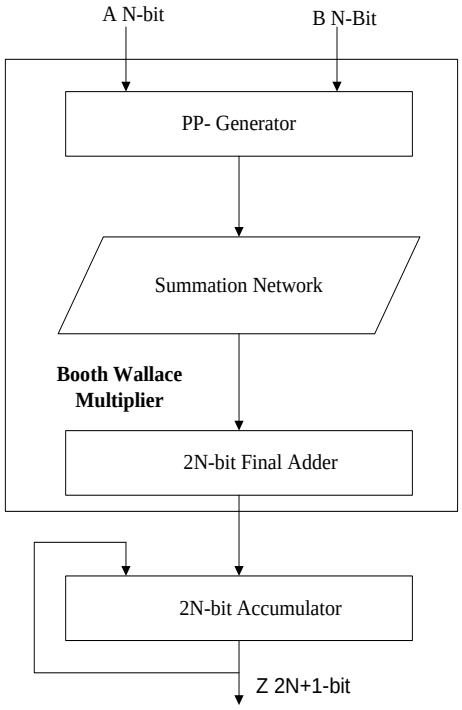



**Figure 3.1:** Basic Architecture of MAC Unit [14]

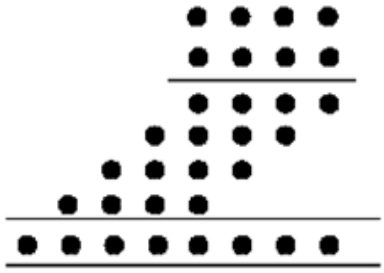


**Figure 3.2:** Basic Multiplication

**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.

Radix-4 Modified Booth's algorithm [17] [18] is:

- Y (i-1) = 0; Insert 0 on the right side of LSB of multiplier.
- Start grouping each three bits with overlapping from Y (i-1) as shown in Figure 3.3.
- If the number of multiplier bits is odd, add a extra 1 bit on left side of MSB and generate partial product from Table 3.1.
- When new partial product is generated, each partial product is added two bit left shifting in regular sequence.
- It is then sign extended.

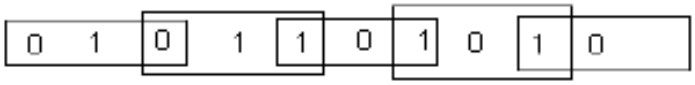



**Figure 3.3:** Three bit overlapping scheme of Radix-4 Booth Algorithm

**Table 3.1:** Radix-4 Modified Booth Algorithm [17][18].

| Yi+1 | Yi | Yi-1 | Recoded Digit | Operand Multiplication |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 x multiplicand |
| 0 | 0 | 1 | +1 | +1 x multiplicand |
| 0 | 1 | 0 | +1 | +1 x multiplicand |
| 0 | 1 | 1 | +2 | +2 x multiplicand |
| 1 | 0 | 0 | -2 | -2 x multiplicand |
| 1 | 0 | 1 | -1 | -1 x multiplicand |
| 1 | 1 | 0 | -1 | -1 x multiplicand |
| 1 | 1 | 1 | 0 | 0 x multiplicand |

**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.

**i. 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.

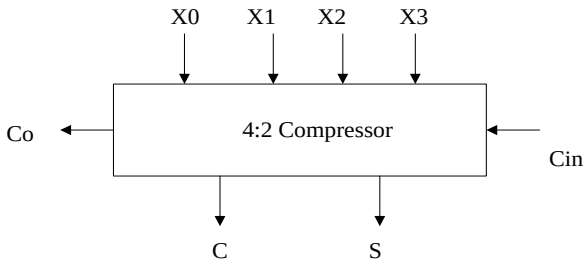



**Figure 3.4:** Block diagram of 4:2 Compressor

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.

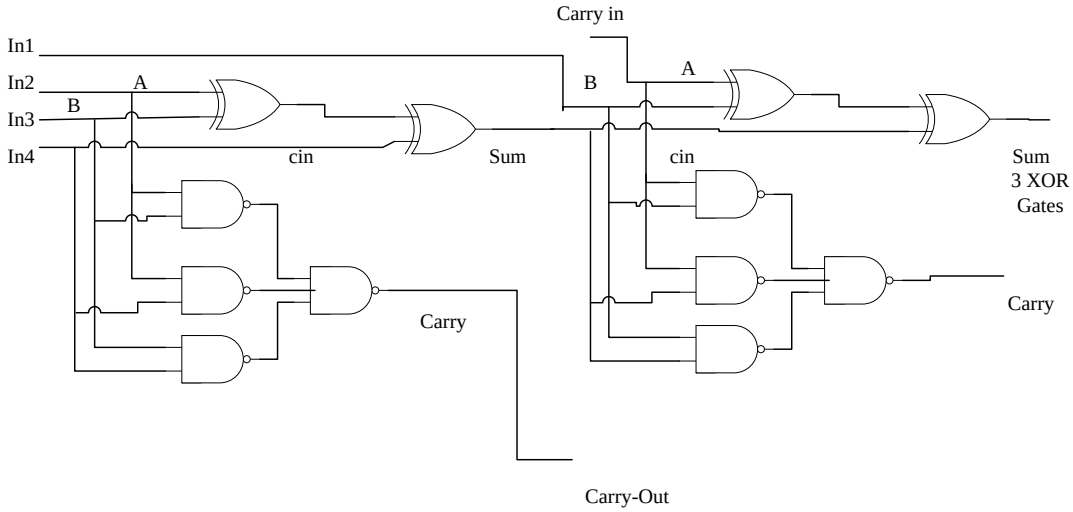



**Figure 3.5:** A 4:2 compressor logic diagram [19]

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.

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 3.6.

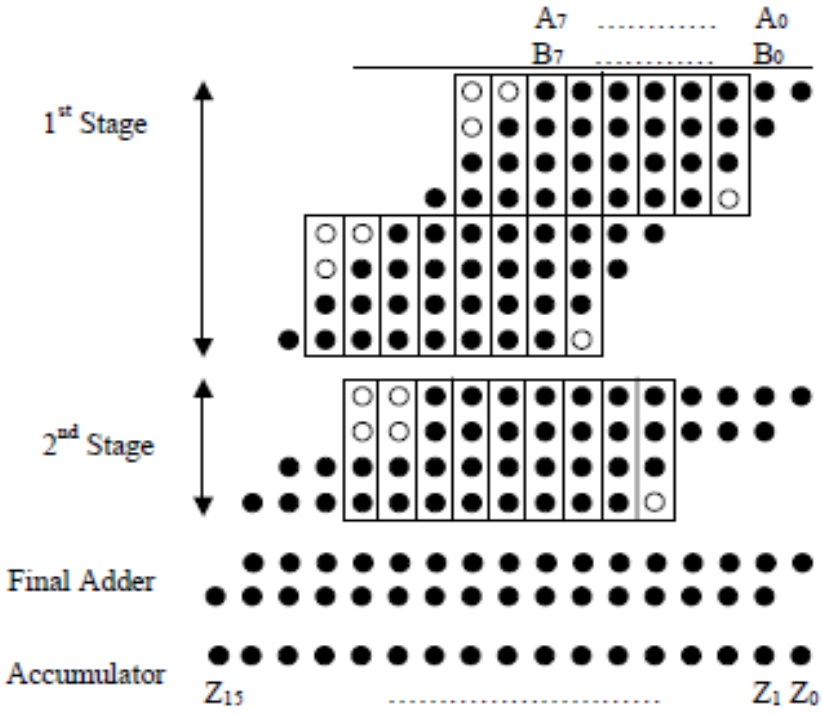



**Figure 3.6:** Data distribution among a tree architecture [14]

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.

### 3.1.2 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 .

## 3.2 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 3.7. 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 3.8.

### 3.2.1 Components

**a. Register**

Registers are used to store the intermediate data. These registers are also called the pipeline registers. All registers used are the rising edge triggered D-Flip Flop. In 32-bit Pipelined Booth Wallace MAC Unit seven registers of different sizes are used according to the intermediate data size.

**b. Booth Encoder**

Modified Radix-4 Booth Algorithm is used in booth encoder. Booth encoder takes three bits from the multiplier with '0' appended at the right end. Then according to the combination of that three bits; booth decoder decode the multiplicand according to the booth encoding Table 3.1 explained in section 3.1.1(a) and gives the output in 33 bits. For 32*32-bit multiplier it develops 16 partial products [17]. After that each de-coder output is shifted by two bits in left and then these partial products are sign extended and zero filled in the right end.

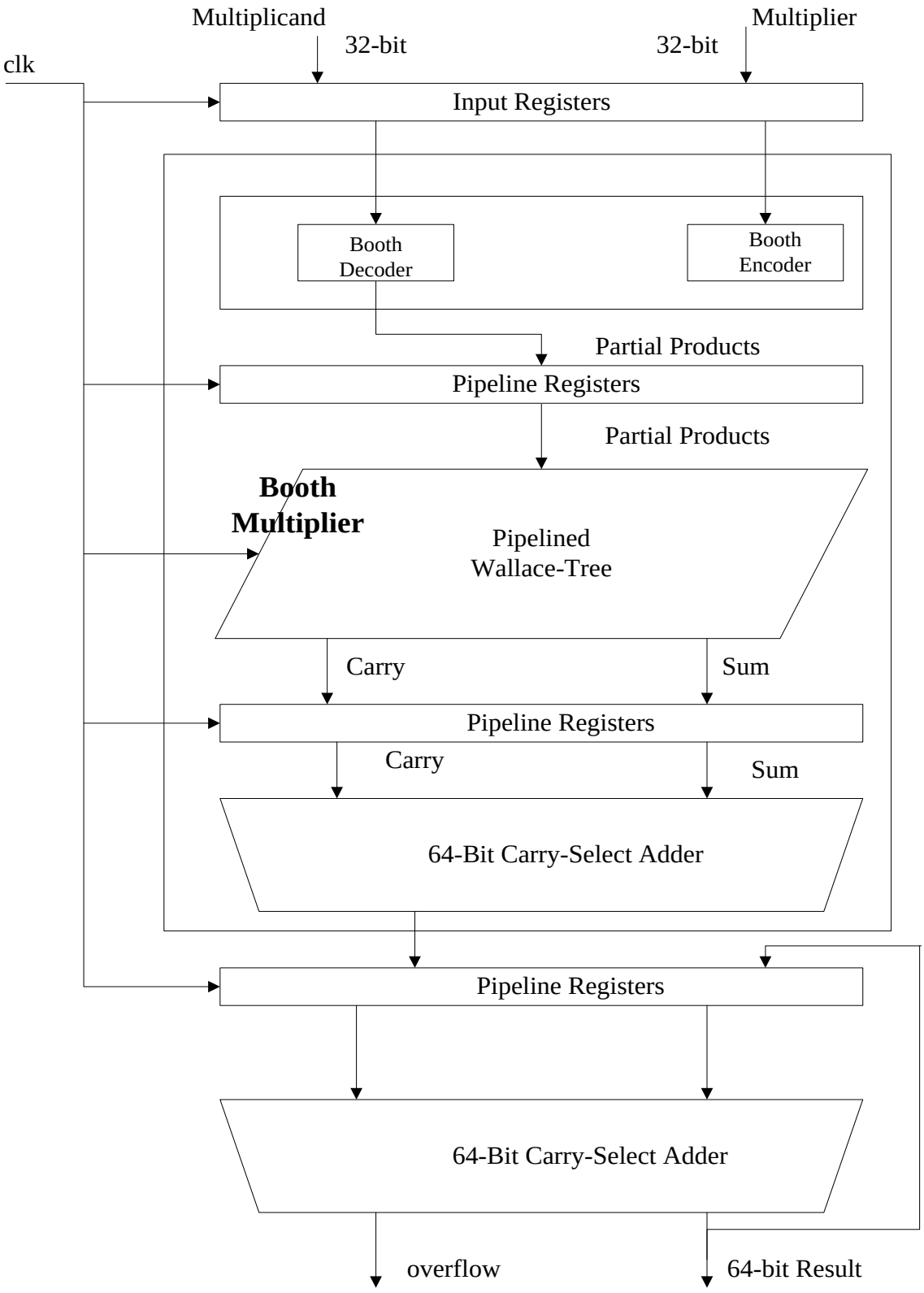



**Figure 3.7:** Block Diagram of 32-bit Pipelined Booth Wallace MAC Unit

**c. Pipelined Wallace Tree**

It is, however, still the most critical part of the multiplier because it is responsible for the largest amount of computation. Naturally the pipeline registers should be inserted within the Wallace tree to improve the performance as shown in Figure 3.8.

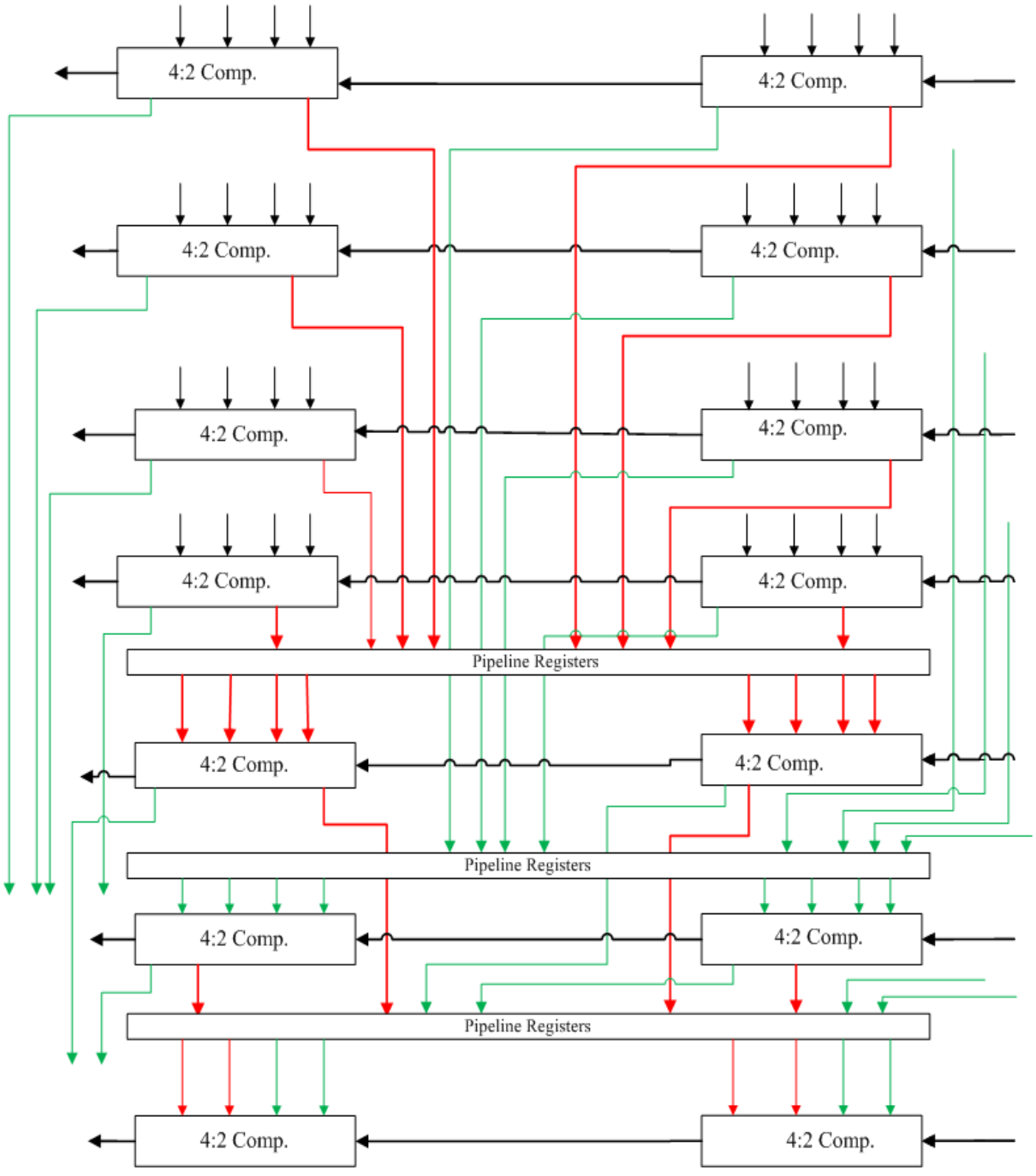



**Figure 3.8:** Pipelining in Wallace Tree

The number of partial products of the N-bit modified Booth multiplier is N/2. In case of the 32-bit modified Booth multiplier, the number of partial products is sixteen. In this three stage of pipelining is used, in 1st stage sum and carry of first four row of compressors are stored in two pipeline registers. In 2nd stage these sum and carry are fed into the fifth and sixth row of compressors and their otput is stored in pipeline registers. In 3rd stage the sum and carry of fifth and sixth row are added in seventh row of compressors. The Wallace tree adds four rows and generates two rows using 4:2 compressors. One row represents the carries and the other row represents the sums.

**d. 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 the ripple carry adder is used 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 Sum for carry is 0 and 1. In spite of large implementation, it can increase the speed of execution. Both vectors are added in 64-bit carry select adder block and the final result is generated.

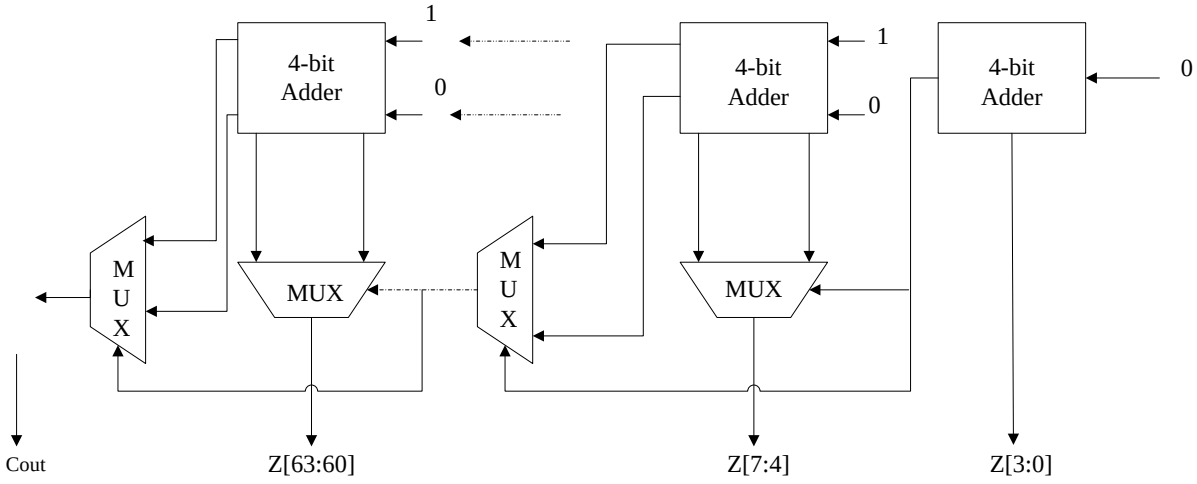



**Figure 3.9:** 64-bit Carry-Select Adder

Accumulator holds output of previous clock from accumulation register. Holding outputs in accumulation register can reduce additional "add" instruction. An accumulator was implemented with one 64-bit carry select adder.

# CHAPTER 4

# 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 [21]. 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.

## 4.1 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.

## 4.2 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.

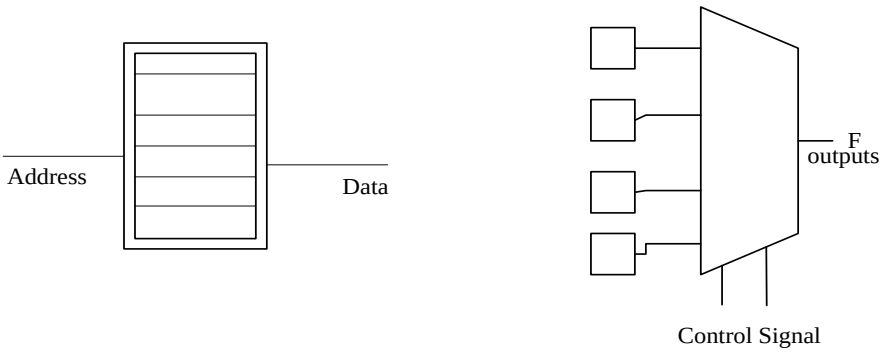
### 4.2.1 Configurable Logic Blocks

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 storage

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 4.1 shows these alternatives, together with an example of memory contents for some basic operations. A $2^n*1$ ROM can implement any n-bit function. Typical sizes for n are 2, 3, 4, or 5. In Figure 4.1(a), an n-bit LUT is implemented as a $2^n*1$ memory; the input address selects one of 2n memory locations. The memory locations are normally loaded with values from the user's configuration bit-stream. In Figure 4.1(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.




(a) (b)

**Figure 4.1:** Look-up table implemented as (a) Memory (b) Multiplexers and Memory [21]

**b. Storage Elements**

The storage elements in a slice can be configured either a edge-triggered D-type flip-flops 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.

### 4.2.2 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.

### 4.2.3 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.

### 4.2.4 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.

## 4.3 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 4.2 are described as follows:

### 4.3.1 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.

### 4.3.2 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.

### 4.3.3 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.

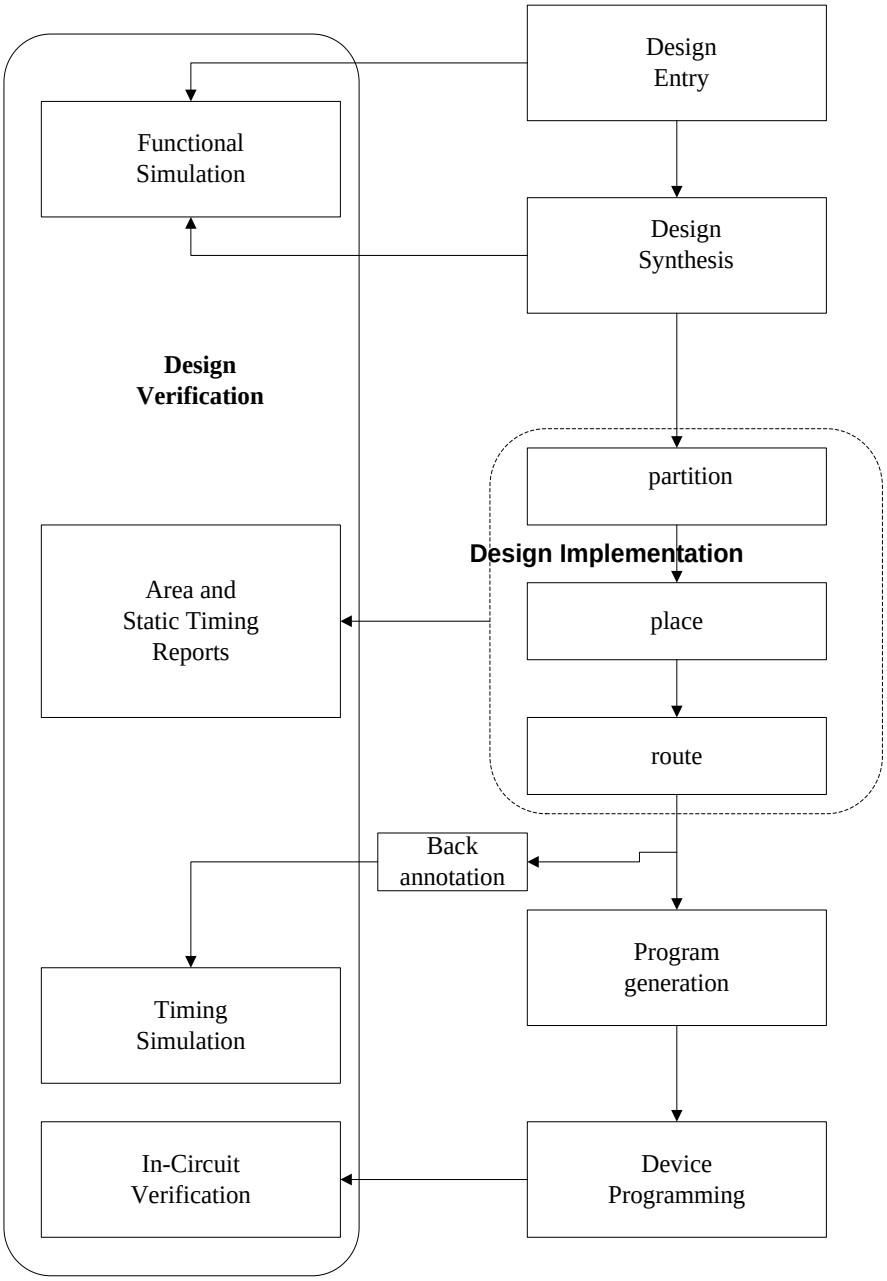



**Figure 4.2:** FPGA Design Flow [21]

**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.

### 4.3.4 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.

### 4.3.5 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.

### 4.3.6 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.

### 4.3.7 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.

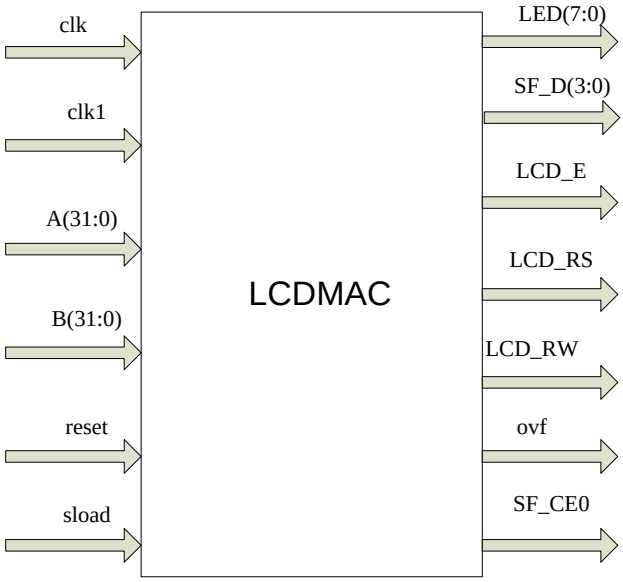



**Figure 4.3:** Block diagram of LCD interfacing for 32-bit MAC Unit

**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.

### 4.3.8 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.

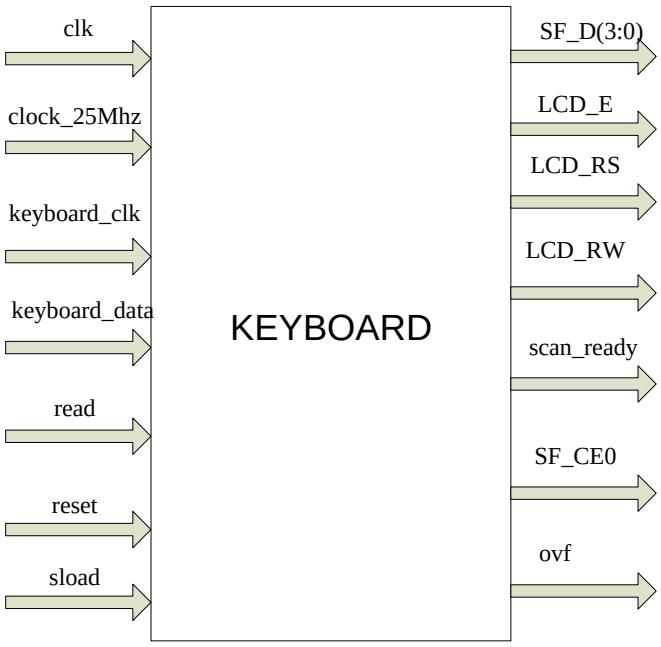



**Figure 4.4:** Block Diagram of 32-bit MAC unit with Keyboard and LCD Interfacing

**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 5

# 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.

### 5.1 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 larger 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.

## 5.2 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 5.1.

### 5.2.1 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.

### 5.2.2 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.

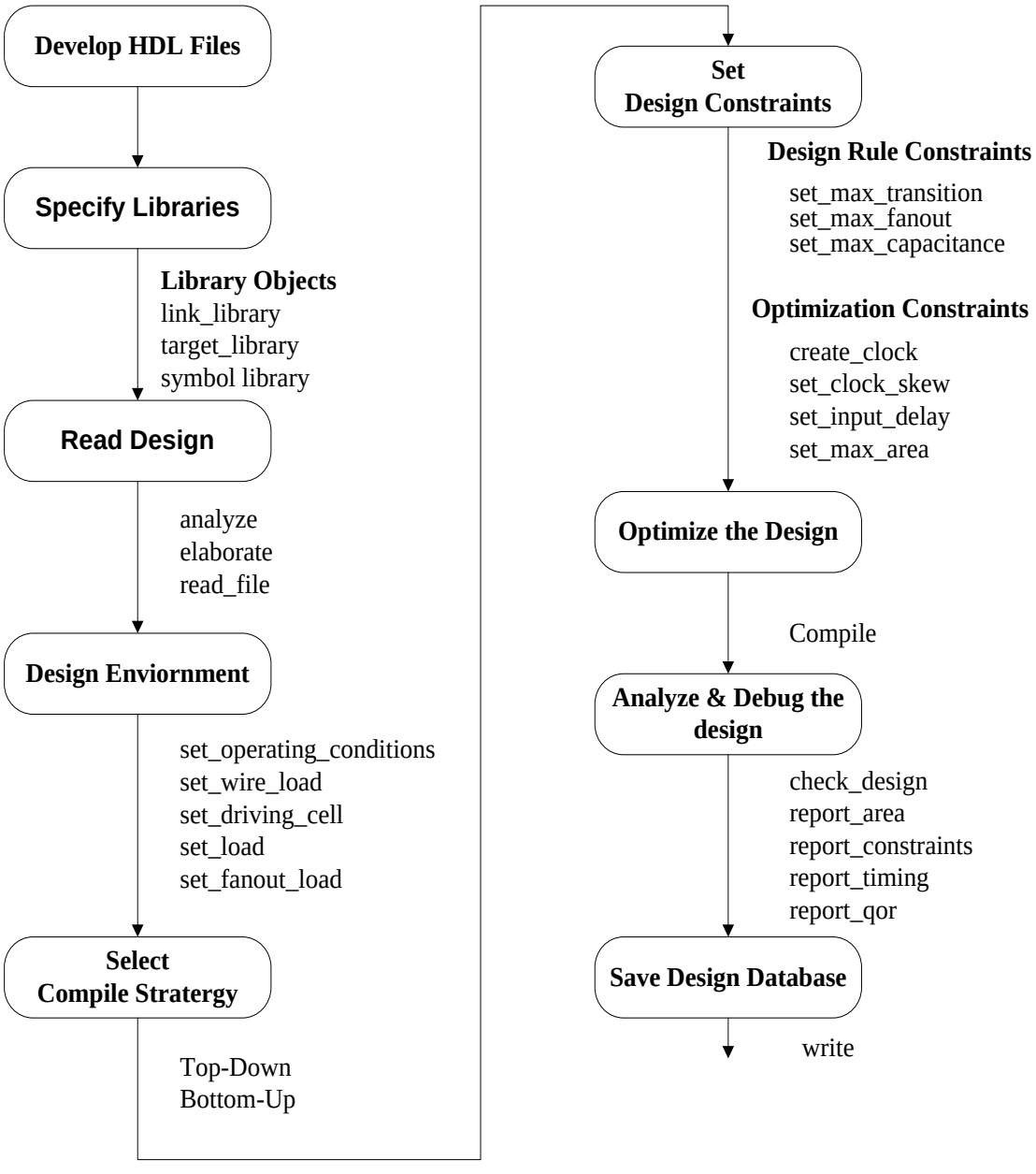



**Figure 5.1:** Basic Design Compiler Synthesis Flow [22]

**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.

### 5.2.3 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.

### 5.2.4 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.

### 5.2.5 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 [24]:

**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.

### 5.2.6 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.

### 5.2.7 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 6

# SIMULATION AND SYNTHESIS RESULTS

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

## 6.1 Adder

### 6.1.1 Synthesis Results on FPGA

The Xilinx ISE 12.4 is used for implementation of all the circuits. The Working Environment for the design is :

- Target Device : XC3S500E-4FG320
- Technology : 90 nm
- Tool Version : ISE 12.4
- FSM Style : LUT
- Optimization Goal : Speed
- Design Strategy : Balanced
- Total Slices : 4656
- Total LUTs : 9312
- Modelsim6.3f

**a. Carry Chain Adder**

The table 6.1 shows the various synthesis results for different size Carry Chain Adder.

**Table 6.1:** Synthesis Results of Carry Chain Adder

| Size | 4-bits | 8-bits | 16-bits | 32-bits | 64-bits |
|---|---|---|---|---|---|
| No. of Slices | 4 | 9 | 18 | 37 | 74 |
| No. of LUTs | 8 | 16 | 32 | 64 | 128 |
| LUT Utilization (%) | 0 | 0 | 0 | 0 | 1 |
| Bonded IOs | 14 | 26 | 50 | 98 | 194 |
| Level of Logic | 6 | 10 | 18 | 34 | 66 |
| Logic Delay (%) | 74.6 | 69.6 | 65.6 | 62.8 | 61.2 |
| Route Delay (%) | 25.4 | 30.4 | 34.4 | 37.2 | 38.8 |
| Max.Comb. Delay (ns) | 9.794 | 14.538 | 24.026 | 43.002 | 80.954 |

**b. Carry-look Ahead Adder**

The table 6.2 shows the various synthesis results for different size Carry-look Ahead Adder.

**Table 6.2:** Synthesis Results of Carry-look Ahead Adder

| 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 |

**c. Carry Select Adder**

The table 6.3 shows the various synthesis results for different size Carry Select Adder.

**Table 6.3:** 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 |

**d. Carry Skip Adder**

The table 6.4 shows the various synthesis results for different size Carry Skip Adder.

**Table 6.4:** 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 |

**e. Analysis**

The Benchmarking of results shows that as the operand size of Adder increases the area in term of LUTs increases and also the combinational path delay increases. For 4-bit operands adder CLA and CCA uses less no. of LUTs and have less combinational path delay than CSLA and CSKA. As the operand's size increases CCA and CLA have more combinational path delay than CSLA due to high level of logic in the critical path. So CCA and CLA are suitable for the addition of operands having less no. of bits. 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.

### 6.1.2 Synthesis Results on Design Compiler

Synopsys DC tool version D-2010.03-SP1 is used for synthesis. The Working Environment is :

- Operating Condition Name : WCCOM
- Library : fsd0c_a_generic_core_ss1p08v125c
- Process : 1.00
- Temperature : 125.00 $^0$C
- Voltage : 1.08 V
- Interconnect Model : Wire load (worst case tree)
- Technology: 90 nm
- Timing Analysis Effort : Medium
- Power Analysis Effort : Low
- Wire Load Model : G10K

**a. Carry Chain Adder**

The table 6.5 shows the various synthesis results for different size Carry Chain Adder.

**Table 6.5:** Synthesis Results of Carry Chain Adder

| Size | 4-bits | 8-bits | 16-bits | 32-bits | 64-bits |
|---|---|---|---|---|---|
| Level of Logic | 6 | 10 | 18 | 34 | 66 |
| Critical Path Length | 0.91 | 1.46 | 2.55 | 4.74 | 9.11 |
| Leaf Cell Count | 13 | 27 | 55 | 111 | 223 |
| Total no. of NET | 22 | 44 | 88 | 176 | 352 |
| Comb.Area | 91.728 | 184.24 | 369.264 | 739.312 | 1479.408 |
| Dyn. Power(uW) | 12.5528 | 26.3535 | 54.2728 | 109.6783 | 220.0941 |
| Cell Lkg.Power(nW) | 58.3107 | 117.9836 | 237.2104 | 475.7574 | 952.9311 |

**b. Carry-look Ahead Adder**

The table 6.6 shows the various synthesis results for different size Carry-look Ahead Adder.

**Table 6.6:** Synthesis Results of Carry-look Ahead Adder

| Size | 4-bits | 8-bits | 16-bits | 32-bits | 64-bits |
|---|---|---|---|---|---|
| Level of Logic | 5 | 9 | 17 | 33 | 64 |
| Critical Path Length | 0.63 | 1.15 | 2.18 | 5.54 | 12.13 |
| Leaf Cell Count | 12 | 24 | 48 | 96 | 191 |
| Total no. of NET | 21 | 41 | 81 | 161 | 319 |
| Comb. Area | 87.808 | 175.616 | 351.232 | 715.008 | 1432.368 |
| Dyn. Power (uW) | 11.31 | 23.55 | 48.48 | 98.08 | 195.25 |
| Cell Lkg.Power(nW) | 56.89 | 113.78 | 227.58 | 455.66 | 904.96 |

**c. Carry Select Adder**

The table 6.7 shows the various synthesis results for different size Carry Select Adder.

**Table 6.7:** Synthesis Results of Carry Select Adder

| Size | 4-bits | 8-bits | 16-bits | 32-bits | 64-bits |
|---|---|---|---|---|---|
| Level of Logic | 7 | 9 | 11 | 15 | 23 |
| Critical Path Length | 1.05 | 1.3 | 1.76 | 2.67 | 4.49 |
| Leaf Cell Count | 34 | 68 | 136 | 272 | 544 |
| Total no. of NET | 43 | 85 | 169 | 337 | 673 |
| Comb. Area | 224.224 | 448.448 | 896.896 | 1793.792 | 3587.584 |
| Dyn. Power (uW) | 29.39 | 60.41 | 123.08 | 248.55 | 497.66 |
| Cell Lkg.Power | 133.91nW | 267.79 nW | 535.55 nW | 1.07 uW | 2.14uW |

**d. Carry Skip Adder**

The table 6.8 shows the various synthesis results for different size Carry Skip Adder.

**Table 6.8:** Synthesis Results of Carry Skip Adder

| Size | 4-bits | 8-bits | 16-bits | 32-bits | 64-bits |
|---|---|---|---|---|---|
| Level of Logic | 7 | 11 | 19 | 35 | 67 |
| Critical Path Length | 0.79 | 1.32 | 2.37 | 4.47 | 8.68 |
| Leaf Cell Count | 16 | 32 | 64 | 128 | 256 |
| Total no. of NET | 25 | 49 | 97 | 193 | 385 |
| Comb. Area | 128.576 | 257.150 | 514.304 | 1028.608 | 2057.216 |
| Dyn. Power (uW) | 14.60 | 30.24 | 61.57 | 124.13 | 248.44 |
| Cell Lkg. Power | 70.78 nW | 141.58 nW | 0.283 uW | 0.566 uW | 1.13uW |

**e. Analysis**

The tables show that as the operand size increases the level of logic, leaf cell count, critical path length and combinational path area increases for every type of adder architecture. But for 4-bit and 8-bit operands size CCA and CLA has less value of critical path so have less delay but as the operands size increases these architectures become less effective in term of delay as compared to CSLA. However the CSLA has comparatively low value of critical path length hence less combinational path delay but it has higher no. of leaf cell count and combinational path area. It also has high dynamic power than other adder architectures. It's clear from the tables that CLA and CSKA architectures can be used for low power applications as it has low value of dynamic as well cell leakage power.

## 6.2 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.

### 6.2.1 16-bit Multiply and Accumulator

**a. Synthesis Results on FPGA**

The table 6.9 shows the various synthesis results of 16-bit MAC Unit for different final stage and accumulator adder.

**Table 6.9:** FPGA Synthesis Results of 16-bit MAC Unit

| Adder Type | CCA | CLA | CSLA | CSKA |
|---|---|---|---|---|
| No. Of Slices | 547 | 553 | 560 | 565 |
| Slice Flip-Flops | 97 | 97 | 117 | 97 |
| 4 input LUTs | 1047 | 1057 | 1077 | 1081 |
| LUT Utilization % | 11 | 11 | 11 | 11 |
| Bonded IOs | 67 | 67 | 67 | 67 |
| Level of Logic | 28 | 17 | 28 | 20 |
| Min. Clock Period (ns) | 19.698 | 19.695 | 18.451 | 23.355 |
| Max. Frequency (MHz) | 50.768 | 50.774 | 54.198 | 42.817 |
| Offset In before clock( ns) | 1.946 | 1.946 | 2.057 | 1.946 |
| Offset Out after clock (ns) | 23.458 | 22.09 | 18.46 | 28.986 |
| Total Power (mW) | 176 | 214.38 | 210.03 | 165.36 |

**b. Synthesis Results on DC**

The table 6.10 shows the various synthesis results of 16-bit MAC Unit for different final stage and accumulator adder on Design Compiler.

**Table 6.10:** DC Synthesis Results of 16-bit MAC Unit

| Adder Type | CC A | CLA | CSLA | CSKA |
|---|---|---|---|---|
| Levels of Logic | 29 | 31 | 38 | 27 |
| Critical Path Length | 2.49 | 2.47 | 2.6 | 2.7 |
| Critical Path Slack | -0.11 | -0.16 | -0.29 | -0.39 |
| CLK Period(ns) | 2.5 | 2.5 | 2.5 | 2.5 |
| Total Negative Slack | -1.75 | -2.27 | -6.37 | -6.23 |
| No. of Violating Paths | 21 | 21 | 32 | 22 |
| Leaf Cell Count | 4509 | 4205 | 4344 | 3837 |
| Comb. Area(C) | 34249.823 | 31606.959 | 32868.416 | 28158.143 |
| Non Comb. Area (S) | 1352.4 | 1466.864 | 1366.512 | 1467.648 |
| C/S Ratio | 25.32 | 21.54 | 24.05 | 19.18 |
| Cell or Design Area | 35602.223 | 33073.82396 | 34234.92801 | 29625.792 |
| Total Number of Nets | 4543 | 4239 | 4378 | 3872 |
| Total Dynamic Power | 5.1854 mW | 4.5968 mW | 5.0742 mW | 4.2472 mW |
| Cell Leakage Power | 28.3131 uW | 25.4987 uW | 26.1717 uW | 22.4250 uW |

**c. Analysis**

The FPGA's synthesis results show that the MAC with CSLA architecture has higher frequency of operation than the MAC with other adder architectures. However it has more LUTs than others. However MAC with CSKA required less power supply than MAC with other adder architectures.

According to DC synthesis Results for a fixed clock period, MAC with CCA has less value of critical path length so have less value of critical path slack as well as total negative slack value so it can run on higher frequency than other 16-bit MAC architectures. However CCA has more leaf cell count and design area than other three architectures. So area and power wise the 16-bit MAC with CSKA is better as compared to other architectures.

### 6.2.2 32-bit Multiply and Accumulator

**a. Synthesis Results on FPGA**

**i. For Design Strategy: Balanced**

The table 6.11 shows the various synthesis results of 32-bit MAC Unit for different final stage and accumulator adder.

**Table 6.11:** FPGA Synthesis Results of 32-bit MAC Unit.

| Adder Type | CCA | CLA | CSLA | CSKA |
|---|---|---|---|---|
| No. of Slices | 2051 | 2056 | 1871 | 2114 |
| Slice Flip-Flops | 243 | 244 | 237 | 229 |
| 4 input LUTs | 3974 | 3982 | 3646 | 4084 |
| LUT Utilization % | 42 | 42 | 39 | 43 |
| Bonded IOs | 131 | 131 | 131 | 131 |
| Level of Logic | 46 | 39 | 47 | 32 |
| Min.Clock Period (ns) | 24.921 | 24.35 | 24.127 | 28.165 |
| Max. Frequency (MHz) | 40.126 | 41.068 | 41.447 | 35.505 |
| Offset In before clock (ns) | 2.057 | 2.113 | 2.057 | 2.057 |
| Offset Out after clock (ns) | 22.054 | 20.868 | 24.557 | 28.972 |
| Total Power (mW) | 282.69 | 269.89 | 279.26 | 290.67 |

**ii. For Design Strategy: Performance with Physical Synthesis**

The table 6.12 shows the various synthesis results of 32-bit MAC Unit for different final stage and accumulator adder with Performance Design Strategy.

**Table 6.12:** FPGA Synthesis Results of 32-bit MAC Unit.

| Final Stage Adder | CCA | CLA | CSLA | CSKA |
|---|---|---|---|---|
| No. Of Slices | 2179 | 2229 | 2071 | 2195 |
| Slice Flip-Flops | 735 | 927 | 776 | 719 |
| 4 input LUTs | 4193 | 4303 | 4013 | 4231 |
| LUT Utilization % | 45% | 46% | 43% | 45% |
| Bonded IOs | 131 | 131 | 131 | 131 |
| Level of Logic | 9 | 12 | 12 | 14 |
| Min.Clock Period( ns) | 15.994 | 14.577 | 14.081 | 15.614 |
| Max. Frequency (MHz) | 62.523 | 68.601 | 71.018 | 64.045 |
| Offset In before clk( ns) | 1.946 | 1.946 | 1.946 | 1.946 |
| Offset Out after clk( ns) | 28.44 | 29.481 | 36.141 | 36.485 |
| Total Power (mW) | 356.92 | 363.97 | 376.67 | 394.25 |

b. **Synthesis Results on DC**

The table 6.13 shows the various synthesis results of 32-bit MAC Unit for different final stage and accumulator adder on Design Compiler.

**Table 6.13:** DC Synthesis Results of 32-bit MAC Unit.

| Adder Type | CCA | CLA | CSLA | CSKA |
|---|---|---|---|---|
| Levels of Logic | 37 | 35 | 54 | 49 |
| Critical Path Length | 3.27 | 3.24 | 3.82 | 4.02 |
| Critical Path Slack | -0.46 | -0.44 | -1 | -1.21 |
| CLK Period(ns) | 3 | 3 | 3 | 3 |
| Total Negative Slack | -21.36 | -20.34 | -43.46 | -41.41 |
| No. of Violating Paths | 57 | 57 | 79 | 62 |
| Leaf Cell Count | 17026 | 17360 | 13518 | 12936 |
| Comb. Area (C) | 127823.359 | 130473.279 | 98513.520 | 89910.688 |
| Non Comb. Area (S) | 2751.8400 | 2696.1760 | 2698.5280 | 2698.5280 |
| C/S Ratio | 46.45 | 48.39 | 36.50 | 33.31 |
| Cell or Design Area | 130575.199 | 133169.455 | 101212.048 | 92609.216 |
| Total Number of Nets | 17092 | 17428 | 13584 | 13002 |
| Total Dyn. Power(mW) | 15.7928 | 15.7229 | 13.0224 | 11.46 |
| Cell Leakage Power(uW) | 101.5989 | 103.3779 | 74.9818 | 67.33 |

**c. 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. However MAC with CLA require less power supply than MAC with other adder architectures.

If the Design Strategy, Performance with Physical Synthesis used then the frequency can be increased by 30 MHz approx. but LUTs and power supply requirements also increases. In this strategy tool optimize the design for high speed.

According to DC synthesis Results for a fixed clock period, MAC with CLA has less value of critical path length so have less value of critical path slack as well as total negative slack value so it can run on higher frequency than other 32-bit MAC architectures. However CSKA has less value of leaf cell count and design area than other three architectures. So area and power wise the 32-bit MAC with CSKA is better as compared to other architectures.

## 6.3 32-bit Conventional Booth Wallace MAC with Carry Select Adder

### 6.3.1 Simulation Results

a : Input Data 32-bit

b : Input Data 32-bit

clk :Input Clock

z: : Output Data 64-bit

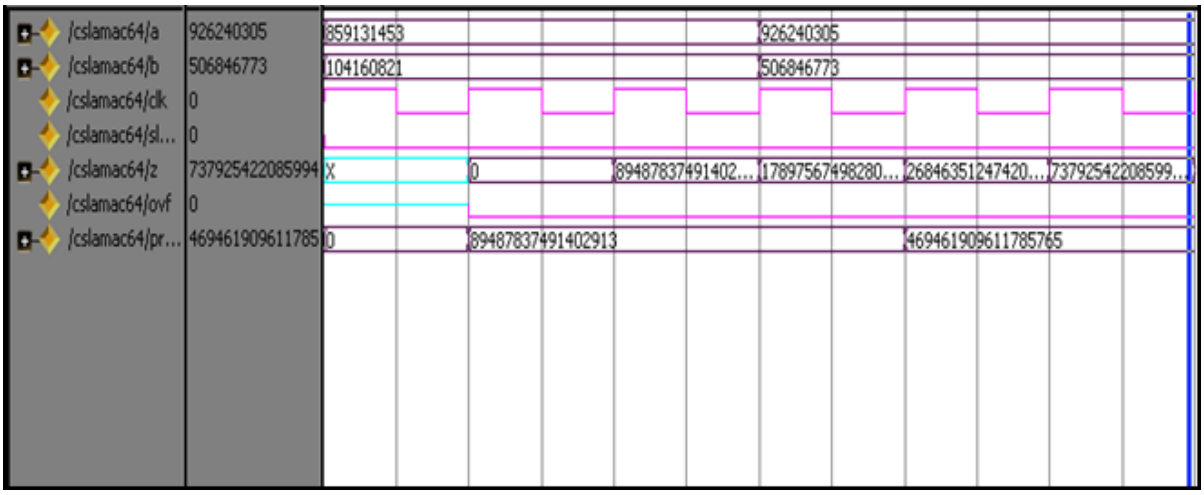



**Figure 6.1:** Simulated Waveform of 32-bit Conventional MAC Unit

The table 6.14 shows the simulation result verification of 32-bit Conventional Booth Wallace MAC Unit. There is latency of 2 clock cycle means we get the final output of MAC in 3rd clock cycle which is clear from the table 6.14 and from Figure 6.1.

**Table 6.14:** Simulation verification of 32-bit Conventional Booth Wallace MAC Unit

| | clk1 | clk2 | clk3 | clk4 | clk5 | clk6 |
|---|---|---|---|---|---|---|
| a | 859131453 | 859131453 | 859131453 | 926240305 | 926240305 | 926240305 |
| b | 104160821 | 104160821 | 104160821 | 506846773 | 506846773 | 506846773 |
| z | x | 0 | 894878374 91402913 | 1789756749 82805826 | 2684635124 74208739 | 7379254220 85994504 |

### 6.3.2 Synthesis Results on FPGA

The table 6.15 shows the synthesis result of 32-bit Conventional Booth Wallace MAC unit.

**Table 6.15:** 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 |

**a. Keyboard and LCD Interfacing**


**Figure 6.2:** LCD Display of 32-bit Booth Wallace MAC Unit

The table 6.16 shows the synthesis results of 32-bit MAC with Keyboard and LCD interfacing. Three global clocks of FPGA has been used one for LCD, 2nd for Keyboard and 3rd one for the MAC architecture.

**Table 6.16:** FPGA Synthesis Results with Keyboard and LCD interfacing of Conventional Booth Wallace MAC Unit

| | |
|---|---|
| No. Of Slices | 2073 |
| Slice Flip-Flops | 570 |
| 4 input LUTs | 4020 |
| LUT Utilization % | 43 |
| Bonded IOs | 16 |
| Min.Clock Period( ns) | 24.406 |
| Max. Frequency (MHz) | 40.97 |
| GCLKs | 3 |

### 6.3.3 Synthesis Results on DC

The table 6.17 shows the synthesis results of 32-bit Conventional Booth Wallace MAC Unit in default condition i.e. all the timing and design constraints have their default value from the library.

**Table 6.17:** DC Synthesis Results of 32-bit Conventional Booth Wallace MAC Unit

| | Worst Case | Best Case | Average Case |
|---|---|---|---|
| CLK Period(ns) | 3.97 | 1.61 | 2.35 |
| Levels of Logic | 51 | 51 | 49 |
| Critical Path Length | 3.78 | 1.55 | 2.25 |
| Critical Path Slack | 0.00 | 0.00 | 0.00 |
| Total Negative Slack | 0.00 | 0.00 | 0.00 |
| No. of Violating Paths | 0 | 0 | 0 |
| Leaf Cell Count | 13613 | 12532 | 13043 |
| Comb. Area(C) | 99313.200 | 88679.024 | 91855.792 |
| Non Comb.Area (S) | 2686.768 | 2655.408 | 2648.352 |
| C/S Ratio | 36.96 | 33.39 | 34.68 |
| Cell or Design Area | 101999.968 | 91334.432 | 94504.144 |
| Total Number of Nets | 13681 | 12598 | 13109 |
| Total Dynamic Power | 10.085 mW | 33.379 mW | 18.821mW |
| Cell Leakage Power | 74.829 uW | 179.578 uW | 61.434 uW |
| Frequency (MHz) | 251.88 | 621.11 | 425.53 |

### 6.3.4 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 in case of best case analysis (BCCOM) the MAC met all the constraints at the clock period 1.61 ns or at the frequency 621.11 MHz and in worst case scenario (WCCOM) all the constraints at the clock period 3.97 ns or at the frequency 251.88 MHz and in average case scenario (TCCOM) all the constraints at the clock period 2.35 ns or at the frequency 425.53 MHz.

## 6.4 32-bit Pipelined Booth Wallace MAC Unit with Carry Select Adder

### 6.4.1 Simulation Results

a : Input Data 32-bit

b : Input Data 32-bit

clk :Input Clock

z: : Output Data 64-bit

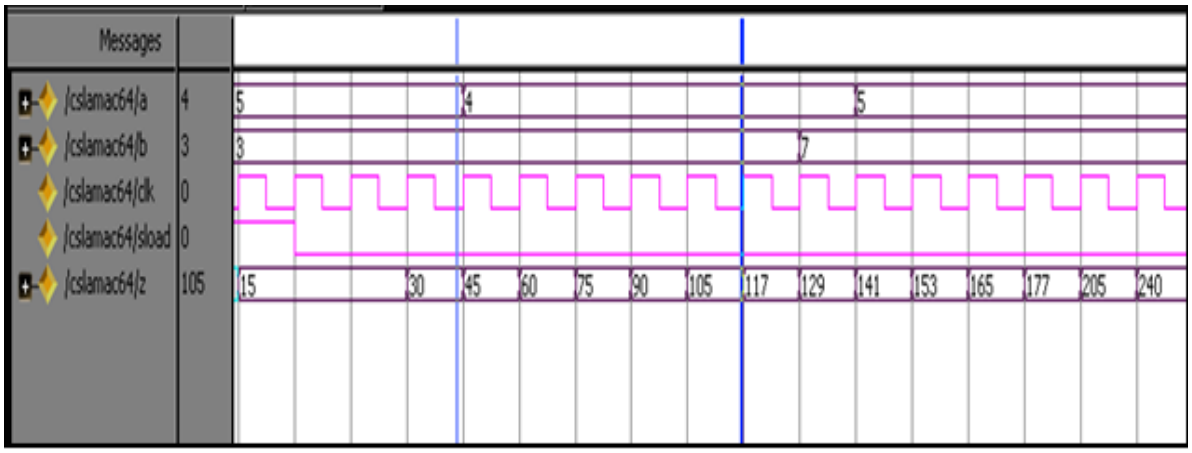



**Figure 6.3:** Simulated Waveform of 32-bit Pipelined Booth Wallace MAC Unit

Table 6.18 shows the simulation verification of 32-bit Pipelined Booth Wallace MAC Unit.

**Table 6.18:** Simulation result verification of 32-bit Pipelined Booth Wallace MAC Unit

| | clk1 | clk2 | clk3 | clk4 | clk5 | Clk6 | Clk7 | Clk8 | Clk9 |
|---|---|---|---|---|---|---|---|---|---|
| a | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 5 |
| b | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 7 |
| z | 15 | 30 | 45 | 60 | 75 | 90 | 105 | 117 | 129 |

There is latency of 5 clock cycle means we get the final output of MAC in 6th clock cycle which is clear from the table 6.18 and from Figure 6.3.

### 6.4.2 Synthesis Results on FPGA

The table 6.19 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.

**Table 6.19:** 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 |

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.

**a. Keyboard and LCD Interfacing**

The table 6.20 shows the synthesis results of 32-bit Pipelined MAC with Keyboard and LCD interfacing on FPGA. Three global clocks of FPGA has been used one for LCD, 2nd for Keyboard and 3rd one for the MAC architecture.

**Table 6.20:** FPGA Results of 32-bit Pipelined MAC with Keyboard and LCD interfacing

| | |
|---|---|
| No. Of Slices | 2308 |
| Slice Flip-Flops | 1716 |
| 4 input LUTs | 3776 |
| LUT Utilization % | 40 |
| Bonded IOs | 16 |
| Min.Clock Period( ns) | 10.935 |
| Max. Frequency (MHz) | 91.449 |
| GCLKs | 3 |

**b. RTL View**

The RTL view of 32-bit Pipelined Booth Wallace MAC on FPGA is shown in Figure 6.4.

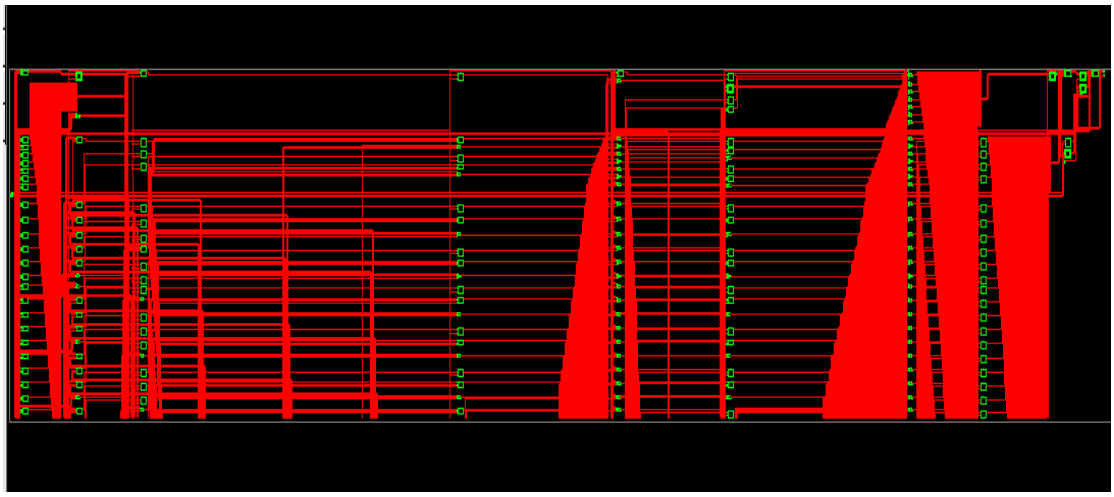


**Figure 6.4:** RTL View of 32-bit Pipelined Booth Wallace MAC Unit

**c. Floor Plan Area**

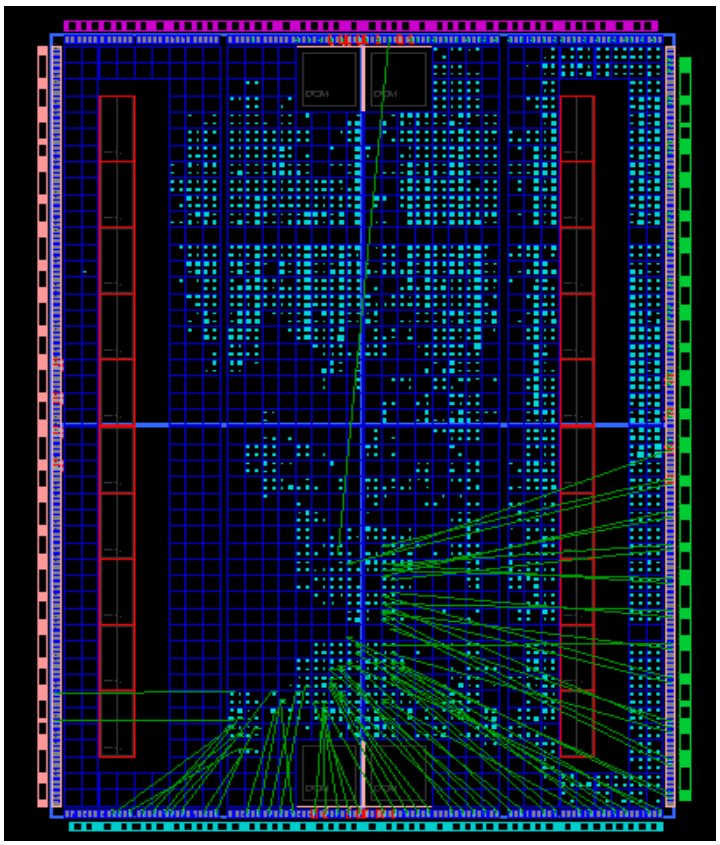


**Figure 6.5:** Floor Plan Area of 32-bit Pipelined Booth Wallace MAC Unit

**d. Static Timing Analysis**

- Clock Period = 15 ns.
- Offset = out 25 ns after Clock
- Offset = in 5.5 ns valid 5 ns before Clock
- System Jitter= 0.05 ns
- Maximum Skew= 0.1 ns
- Temperature= 50 $^0$C
- Voltage= 1.14V
- Maximum Fanout = 4

**Table 6.21:** Static Timing Results of 32-bit Pipelined Booth Wallace MAC Unit on FPGA

| | Result after Synthesis | Result After PAR |
|---|---|---|
| No. Of Slices | 2928 | 3710 |
| Slice Flip-Flops | 2948 | 2948 |
| 4 input LUTs | 3632 | 3616 |
| LUT Utilization % | 39 | 38 |
| Bonded IOs | 131 | 131 |
| Min.Clock Period( ns) | 10.635 | 14.648 |
| Max. Frequency (MHz) | 94.029 | 69.06 |
| Offset In before clk( ns) | 4.777 | 5.526 |
| Offset Out after clk( ns) | 16.225 | 22.150 |
| Total Power (mW) | 433.62 | 433.62 |

It's clear from the table that the result after synthesis and after PAR are different. The actual frequency of operation and actual no. of LUTs are determined after PAR.

If the input constraints are applied to use the MAC for the practical scenario, to meet all that constraints the clock period have to increased to avoid slack in the data path, so the frequency of operation is changed with input constraints, from the table 6.21 the MAC has frequency 69.06MHz when input constraints are applied. The best achievable input offset delay is 5.526 ns and output offset delay is the 22.150 ns.

### 6.4.3 Synthesis Result on DC

The table 6.22 shows the synthesis results of 32-bit Pipelined Booth Wallace MAC Unit in default case i.e. all the constraints, timing and design have their default value from the library.

**Table 6.22:** DC Synthesis Results of 32-bit Pipelined Booth Wallace MAC Unit

| | Worst Case | Best Case | Average Case |
|---|---|---|---|
| CLK Period(ns) | 2.43 | 1.1 | 1.45 |
| Levels of Logic | 22 | 33 | 38 |
| Critical Path Length | 2.23 | 1.04 | 1.38 |
| Critical Path Slack | 0.00 | 0.00 | 0.00 |
| Total Negative Slack | 0.00 | 0.00 | 0.00 |
| No. of Violating Paths | 0 | 0 | 0 |
| Comb. Area(C) | 56189.280 | 55696.928 | 56130.480 |
| Non Comb.Area (S) | 23781.072 | 23728.544 | 23759.904 |
| C/S Ratio | 2.36 | 2.34 | 2.36 |
| Cell or Design Area | 79970.352 | 79425.472 | 79890.384 |
| Total Number of Nets | 11215 | 10668 | 11058 |
| Frequency (MHz) | 411.52 | 909.09 | 689.65 |
| Total Dynamic Power (mW) | 17.3468 | 60.9256 | 36.5846 |
| Cell Leakage Power (uW) | 37.5806 | 124.1783 | 39.7328 |

The design compiler results shows that in case of best case analysis (BCCOM) the MAC met all the constraints at the clock period 1.1 ns or at the frequency 909.09 MHz and in worst case scenario (WCCOM) all the constraints at the clock period 2.43 ns or at the frequency 411.52 MHz and in average case scenario (TCCOM) all the constraints at the clock period 1.45 ns or at the frequency 689.65 MHz.

**a. Static Timing Analysis**

- Clock Latency : 1.0 ns
- Input Delay : 2.0 ns
- Output Delay : 2.0 ns
- Clock Uncertainty : 0.1 ns
- Clock Transition : 0.5 ns
- Maximum Net Transition: 0.1 ns
- Maximum Capacitance: 0.5pF
- Fan-out : 4
- Load : 4
- Maximum Area : 0.0
- Wire Load Model : G10K

**Table 6.23:** Static Timing Results of 32-bit Pipelined Booth Wallace MAC Unit on DC

| | |
|---|---|
| CLK Period(ns) | 3.60 |
| Levels of Logic | 36 |
| Critical Path Length | 1.5 |
| Leaf Cell Count | 9615 |
| Comb. Area(C) | 58587.537 |
| Non Comb. Area (S) | 23694.832 |
| C/S Ratio | 32.33 |
| Cell or Design Area | 82282.369 |
| Total Number of Nets | 10979 |
| Total Dynamic Power (mW) | 19.05 |
| Cell Leakage Power (uW) | 40.80 |

**b. RTL View**

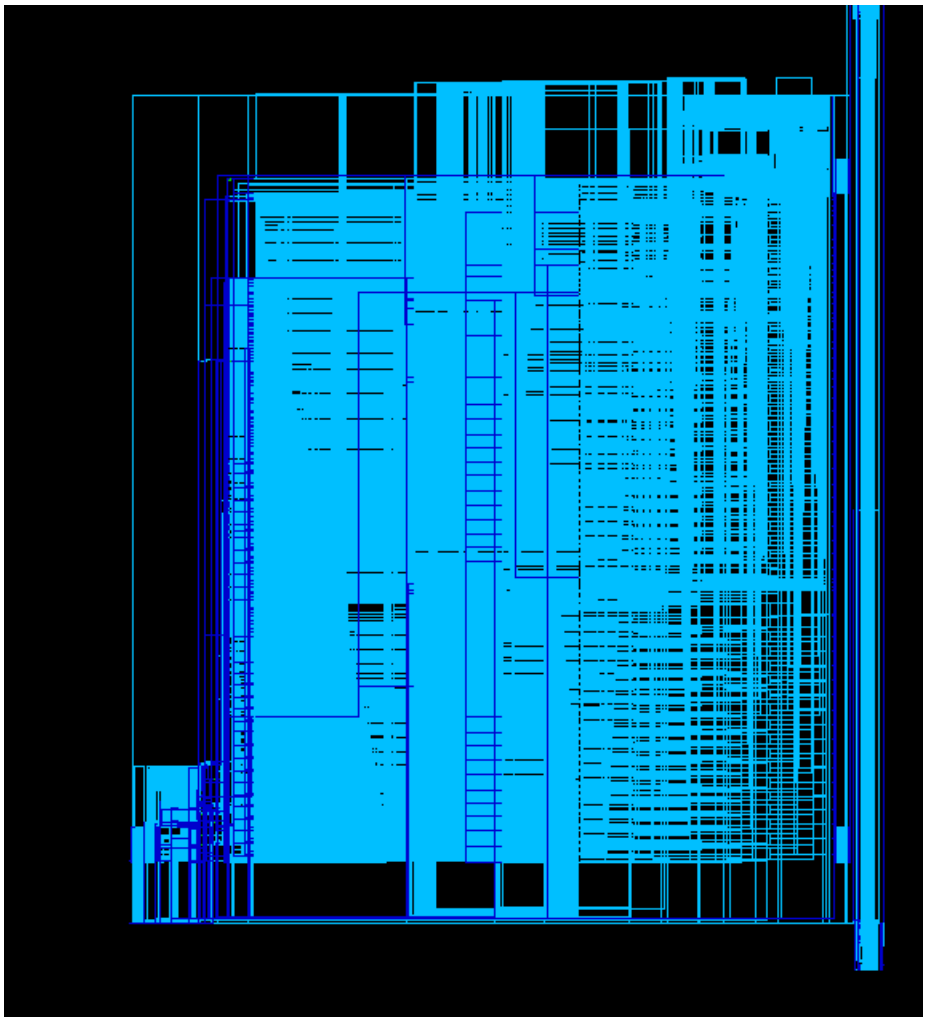


**Figure 6.6:** RTL View of 32-bit Pipelined Booth Wallace MAC Unit

# CHAPTER 7
# CONCLUSION

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

### 7.1 Conclusion

The design of 4 bits,8 bits,16 bits,32 bits and 64 bits Carry Chain Adder, Carry-look Ahead Adder, Carry Select Adder and Carry Skip Adder have been implemented on Spartan XC3S500-4FG320 device. By analyzing the result, 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 7.1 shows the synthesis results of different adder architectures and table 7.2 shows the results of reference [10] for 16-bit operand size .

**Table 7.1:** Synthesis Results of Adders

| Adder | CCA | | | CLA | | | CSLA | | | CSKA | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Size (bit) | 16 | 32 | 64 | 16 | 32 | 64 | 16 | 32 | 64 | 16 | 32 | 64 |
| Slices | 18 | 37 | 74 | 19 | 39 | 78 | 29 | 56 | 113 | 30 | 61 | 120 |
| Delay(ns) | 24.0 | 43.0 | 80.9 | 21.0 | 35.9 | 70.9 | 16.6 | 26.6 | 48.3 | 24.3 | 43.4 | 80.9 |

**Table 7.2:** Comparisons with other adders [10]

| Adders | Slices | Delay (ns) | Delay (ns) |
|---|---|---|---|
| Block Size=4 bits | | Sum | Carry |
| RCA | 24 | 24.1 | 23.5 |
| CLA | 26 | 20.9 | 20.6 |
| CSLA | 27 | 17.1 | 17.7 |
| CSKA | 23 | 23.2 | 23.1 |

By comparing the both table it is clear that CSLA has less delay than other adder architectures.

After that 16 bits and 32 bits Booth Wallace MAC unit are designed using each type of adder architectures 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 other adder architectures. All these designs are also synthesized using Synopsys Design Compiler to calculate power, design area and other critical parameters. Table 7.3 shows that MAC with CSLA is faster for 16 bits and 32 bits.

**Table 7.3:** Synthesis Results of 16-bit and 32-bit MAC Unit

| MAC | MAC with CCA | | MAC with CLA | | MAC with CSLA | | MAC with CSKA | |
|---|---|---|---|---|---|---|---|---|
| Size | 16-bit | 32-bit | 16-bit | 32-bit | 16-bit | 32-bit | 16-bit | 32-bit |
| LUTs | 1047 | 3974 | 1057 | 3982 | 1077 | 3646 | 1081 | 4084 |
| Delay (ns) | 19.7 | 24.9 | 19.7 | 24.3 | 18.4 | 24.1 | 23.3 | 28.1 |
| Frequency (MHz) | 50.76 | 40.1 | 50.77 | 41.0 | 54.2 | 41.5 | 42.8 | 35.5 |
| Power (mW) | 176 | 282.7 | 214.3 | 269.9 | 210 | 279.2 | 165.4 | 290.7 |

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 and finally Floor plan and static timing analysis of the 32-bit Pipelined Booth Wallace MAC has been done using Xilinx ISE 12.4 and Synopsys Design Compiler.

**Table 7.4:** 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 7.4 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.

The 32-bit Pipelined Booth Wallace MAC with CSLA also synthesized on Design Compiler and its results compared with the 32-bit conventional Booth Wallace MAC unit.

**Table 7.5:** Design Compiler Synthesis Results of Conventional and Pipelined 32-bit MAC

| | Conventional Booth Wallace MAC | | | Pipelined Booth Wallace MAC | | |
|---|---|---|---|---|---|---|
| | Worst Case | Best Case | Avg. Case | Worst Case | Best Case | Avg Case |
| Clock Period(ns) | 3.97 | 1.61 | 2.35 | 2.43 | 1.1 | 1.45 |
| Design Area | 101999.96 | 91334.43 | 94504.14 | 79970.35 | 79425.47 | 79890.38 |
| Dynamic Power(mW) | 10.085 | 33.379 | 18.821 | 17.3468 | 60.9256 | 36.5846 |
| Leakage Power(uW) | 74.829 | 179.578 | 61.434 | 37.5806 | 124.1783 | 39.7328 |
| Frequency (MHz) | 251.88 | 621.11 | 425.53 | 411.52 | 909.09 | 689.65 |
| Speed Improvement | - | - | - | x1.63 | x1.46 | x1.62 |

**Table 7.6:** Design Compiler Synthesis Results 16-bit Pipelined Multiplier[18]

| Type | Case | # of stages | Gate count | Area increase | Delay(ns) | Speed improvement |
|---|---|---|---|---|---|---|
| A | - | - | 3,629 | - | 2.40 | - |
| B | - | 3 | 4,017 | x1.11 | 1.66 | x1.45 |
| C | 1 | 5 | 4,526 | x1.25 | 0.82 | x2.93 |
| | 2 | 5 | 4,528 | x1.25 | 0.91 | x2.64 |
| | 3 | 5 | 4,528 | x1.25 | 0.91 | x2.64 |
| | 4 | 4 | 4,452 | x1.23 | 1.05 | x2.29 |
| | 5 | 4 | 4,268 | x1.17 | 1.10 | x2.18 |
| | 6 | 9 | 6,069 | x1.67 | 0.47 | x5.11 |

### 7.2 Future 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 variable stage carry select adder is used. Also the speed can be further increased by applying the pipelining in full adder in Wallace Tree. This MAC unit can be optimized for low power by using CSKA architectures. Using Synopsys Physical Compiler and Prime time the GDS level netlist and layout can also be generated.

# 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] Goel. A., and Bapat, P., "A New Time-Position Algorithm for the Modeling of multilevel Carry Skip Adders in VHDL", Canadian Conference on Electrical and Computer Engineering, pp. 158-61, 1996.

[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] R.P.P.,Singh; B. Singh and P. Kumar, "Performance Analysis Of Fast Adders Using VHDL", International Conference on Advances in Recent Technologies in Communication and Computing, 2009.

[11] R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI Synthesis", A dissertation submitted to the Swiss Federal Institute of Technology, Zurich for the degree of Doctor of technical sciences, 1997.

[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] 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.

[14] 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.

[15] A.D.Booth, "A Signed Binary Multiplication Technique", Quarterly J.Mechan Appl.Math., vol.IV, 1951.

[16] Macsorley, O.L., "High-Speed Arithmetic in Binary Computers", Proceedings of the IRE ,vol. 49, pp 67 – 91, 1961.

[17] K.C. Chang, "Digital system design with VHDL and Synthesis" An integrated Approach IEE Computer Society, pp 408-437, 1999.

[18] Kim Soojin; Cho Kyeongsoon; "Design of High-speed Modified Booth Multipliers Operating at GHz Ranges", World Academy of Science, Engineering and Technology, Issue 61, January 2010.

[19] 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.

[20] Yongserk Lee, "Design of MIPS Processor Supporting MAC with FPGA" A thesis submitted to Taewoo Han School of Electrical and Electronic Engineering College, Yonsei University, June 2007.

[21] 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.

[22] Design Compiler User Guide v1999.10.

[23] Himanshu Bhatnagar; "Advanced ASIC Chip Synthesis" using Synopsys Design Compiler, Physical Compiler and Prime Time, 2nd ed..Kluwer Academic Publishers, 2002.

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

[25] Hima Bindu Kommuru Hamid Mahmoodi, "ASIC Design Flow Tutorial Using Synopsys Tools", Nano-Electronics & Computing Research Lab School of Engineering San Francisco State University San Francisco, CA Spring 2009.