Study of 64x64 bit Multipliers

An Internship Report

submitted by

UMESH KUMAR YADAV (EVD18I027)

in partial fulfilment of requirements

for the award of the dual degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

Department of Electronics and Communication Engineering
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING KANCHEEPURAM

February 2023

DECLARATION OF ORIGINALITY

I, **Umesh Kumar Yadav**, with Roll No: **EVD18I027** hereby declare that the material presented in the Project Report titled **Study of 64x64 bit Multipliers** represents original work carried out by me in the **Department of Electronics and Communication Engineering** at the Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Umesh Kumar Yadav

Place: Chennai

Date: February 2023

CERTIFICATE

This is to certify that the report titled **Study of 64x64 bit Multipliers**, submitted by **Umesh Kumar Yadav (EVD18I027)**, to the Indian Institute of Information Technology, Design and Manufacturing Kancheepuram, for the award of the dual degree of **BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY** is a bonafide record of the work done by him/her under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Binsu J Kailath
Guide
Designation
Department of ECE
IIITDM Kancheepuram, 600 127

Place: Chennai

Date: February 2023

ACKNOWLEDGEMENTS

I am grateful to the **Electronics and Communication Engineering Department of Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram** for offering me the chance to work on a research project as an intern under the guidance of **Prof. Binsu J Kailath**.

I want to express my heartfelt appreciation to **Prof. Binsu J Kailath** for granting me this fantastic opportunity to carry out my internship project under her esteemed mentorship. Her unwavering encouragement and valuable insights have enhanced my understanding of the project's challenges. Despite being busy, she always takes time to answer all my questions with patience. I am especially grateful for the WhatsApp messages, phone calls, and project discussions we had.

I also wish to express my gratitude to my college for providing support throughout the project. Additionally, my acknowledgements would be incomplete without recognizing the contributions of my family and friends. I am particularly grateful to my **elder brother** for motivating and encouraging me, as well as providing different ideas and cheering me up during the various stages of the project.

ABSTRACT

In modern high-speed Digital Signal Processing, Speech recognition and other applications, multipliers are critical components. Multiplication of two binary numbers is a fundamental and commonly used arithmetic operation. The objective of this paper is to examine the principles behind four different kind of multipliers: Array multiplier, Booth multiplier, Baugh-Wooley multiplier, and Wallace tree multiplier. Additionally, this paper includes the implementation of each multiplier using Verilog HDL in Xilinx Vivado 2020.2.

KEYWORDS: Array multiplier; Booth multiplier; Baugh-Wooley multiplier; Wallace tree multiplier; Verilog HDL.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
ALU	Arithmetic logic unit
DSP	Digital Signal Processing
HDL	Hardware Description Language
LSB	Least Significant Bit
LUT	Look Up Table
MAC	Multiply ACcumulate
MSB	Most Significant Bit
RADAR	RAdio Detection And Ranging
RCA	Ripple Carry Adder
RSA	Right Shift Arithmetic
RSC	Right Shift Circular
SONAR	SOund NAvigation Ranging
VLSI	Very Large Scale Integration

CHAPTER 1

Introduction

The semiconductor device industry has grown significantly, leading to the creation of portable systems with enhanced data transmission reliability. Multiplication is a commonly used arithmetic operation in various scientific and signal processing applications. multipliers are crucial components in digital signal processing, including digital image processing, telecommunications, audio signal processing, SONAR, RADAR, speech recognition systems, digital filtering, spectral analysis, seismology, and others. Presently, numerous DSP applications focus on battery-powered portable systems, making power consumption a vital design consideration. Since multipliers are intricate circuits that typically operate at high system clock rates, minimizing the multiplier delay is essential to meet overall design requirements.[2]

Digital signal processors are microprocessors that are purposely designed for processing signals, such as convolution, Fast Fourier Transform, de-convolution, filtering, and others. These tasks require significant mathematical computations that are not present in general-purpose microprocessors, making DSPs a better option. DSPs have unique features, including efficient implementation of fundamental mathematical operations. With the increasing demand for high-speed processors in the semiconductor industry and daily life, there is a growing need for DSPs to execute these operations more effectively. Consequently, developing high-speed algorithms for fundamental operations as Addition, Subtraction, Multiplication, and Division, is crucial to fulfill the industry's demand for high-speed processors.

The multiply-accumulate (MAC) unit are the primary building blocks of a DSP processor, which conducts multiplication and accumulation operations. The unit includes combinational logic that comprises a multiplier, adder, and accumulator register to store the result. Improving the multiplier's latency can significantly boost the DSP processor's speed. In DSP and ASIC, the Arithmetic Logic Unit (ALU) is the primary component utilized for comparison, convolution, correlation, and digital filtering. An

ALU integrates various arithmetic and logic operations into a single unit. The multiplier circuit's performance heavily influences the ALU's speed. Consequently, there is a growing demand for high-speed multipliers that also exhibit less number of LUTs and moderate consumption of power.[2]

The objective of the paper here is to introduce several multiplication algorithms and architectures and assess them based on a blend of metrics, such as area, speed, power consumption, and a combinations of all these factors.

1.1 Background

Digital signal processing relies heavily on multipliers, which are electronic circuits that multiply two numbers and generate a result. multipliers play a vital role in various applications such as audio and image processing, digital communications, and radar and sonar systems. As the speed and power consumption of multipliers significantly impact digital signal processing performance, researchers and engineers are continuously working to design more efficient and effective multiplier circuits.

The most commonly used multiplication technique in digital signal processing is the "add and shift" algorithm, which involves generating partial products based on the binary representation of the numbers being multiplied and then accumulating them to obtain the final result. However, other multiplication algorithms and architectures have been developed to meet specific requirements, such as reducing the number of partial products or minimizing power consumption.

A crucial consideration when designing multipliers is balancing the trade-off between speed, power consumption, and silicon area. There are two main types of multiplier circuits: serial & parallel. Each configuration has its own set of advantages & disadvantages. Serial multipliers are often more space-efficient but slower than parallel multipliers, which require more space but can be faster.

Because multipliers are such a crucial component of digital signal processing, there is a continuous effort to design multiplier circuits that are more efficient, faster, and have a smaller power and area footprint.

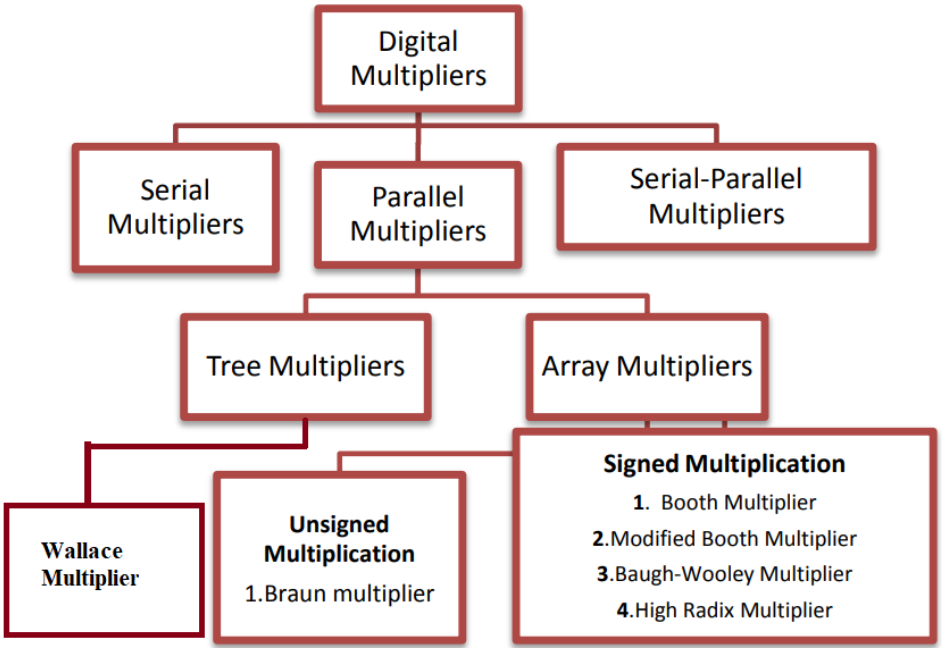


Figure 1.1: Classification of Digital multipliers[2]

The objective of paper is to understand and compare the algorithms and architecture of four different types of multipliers, considering factors such as speed, area, power consumption, and combinations thereof.

- Array multiplier.
- Booth multiplier.
- Baugh-Wooley multiplier.
- Wallace multiplier.

1.2 Motivation

Digital signal processing and other modern applications rely heavily on multiplication as a crucial operation. In microprocessors and digital signal processors (DSPs), multiplication and accumulation of binary numbers are crucial arithmetic operations that are frequently used. Research indicates that over 70% of instructions in microprocessors and most DSP algorithms involve multiplication and addition, making these operations the most time-consuming.

Designing multipliers with low power consumption is a crucial objective for designers. This is because reducing the number of operations performed can result in a

decrease in dynamic power consumption, which accounts for a significant proportion of the overall power consumption. Therefore, the demand for multipliers that are both fast and consume less power has risen. As a result, designers prioritize the development of circuits that are efficient in terms of both speed & power consumption.

The goal of an efficient multiplier is to provide -

- Accuracy: - A good multiplier should produce accurate output.
- Speed: - The multiplier should execute the operation quickly.
- Area: - A multiplier should be designed to occupy a small number of slices and LUTs (Look Up Tables)
- Power: - A multiplier should have low power consumption.

1.3 Objectives of the work

The main objectives of this project are:-

- Understanding logic behind each one multiplier.
  - Array multiplier.
  - Booth multiplier.
  - Baugh-Wooley multiplier.
  - Wallace multiplier.
- Understanding different techniques to reduce the latency of all these multipliers.
- Implementing these multipliers for binary number system in xilinx Vivado using verilog HDL.
- Performing the delay and area analysis of these multipliers.

CHAPTER 2

Methodology

2.1 For designing an Array Multiplier

Below is the algorithm for multiplying an N-bit multiplicand by an N-bit multiplier:

$A = A_{m-1}A_{m-2}.....A_2A_1A_0$ //multiplicand

$B = B_{n-1}B_{n-2}.....B_2B_1B_0$ //multiplier

- Here we generate partial products using AND gates. In case the multiplicand has M bits and the multiplier has N bits, the partial products number generated would be $M \times N$.
- Array multiplier is a commonly used multiplier design that employs the add and shift algorithm for its multiplication circuit. all bit of the multiplier is utilized to multiply the multiplicand and generate a partial product. The resulting partial products are then shifted according to their respective bit positions and summed together. The array multiplier has a regular structure and is widely acknowledged.

2.2 For designing a Booth Multiplier

The Booth Algorithm is method for multiplying binary integers represented in 2's complement form. This technique is efficient as it requires fewer addition and subtraction operations. The algorithm achieves multiplication by performing a small number of shift and addition operations. The underlying concept behind the algorithm is that it is possible to handle groups of 0's in the multiplier by just shifting, while strings of 1's between bit weight 2^k and 2^m can be treated as 2^{k+1} to 2^m. As a result of this approach, the Booth algorithm can decrease the number of multiplicand and multiplier operations required.

Like other methods of multiplication, the Booth algorithm involves scrutinizing the bits of the multiplier and modifying the partial product by shifting. However, before

performing the shift operation, the multiplicand can either be added to the partial product, subtracted from it, or left unchanged, based on the following principles.

- If a series of 1's in the multiplier is detected, beginning from the least significant bit, then the multiplicand is subtracted from the partial product.
- If a sequence of 0's in the multiplier is found after a previous '1,' the multiplicand is added to the partial product.
- partial product remains unchanged when the current multiplier bit is the same as the previous one.

2.3 For designing a Baugh-Wooley Multiplier

The Baugh-Wooley multiplication technique is a useful method for handling signed numbers. It was designed to produce multipliers that are appropriate for 2's complement numbers and have a uniform structure. The Baugh-Wooley multiplier solely uses full adders, and all bit products are generated in parallel and then combined through an array of full adders.

The Baugh-Wooley algorithm can be broken down into three distinct stages:

- In the Baugh Wooley algorithm, the partial-products of each row except the last row's most significant bit (MSB) and all bits of the last partial-product row except for its MSB are inverted.
- A '1' is added to the Nth column.
- The most significant bit of the final result is inverted.

2.4 For designing a Wallace Tree Multiplier

The Wallace tree architecture concurrently sums all the bits from each column's partial products by utilizing a collection of counters, without any carry-over. Afterward, another set of counters is utilized to reduce the resulting matrix, and this procedure is repeated until a two-row matrix is obtained. Full Adders such as the 3:2 counter are typically employed as counters. The final outcomes are then added using a fast adder.

The operation of the Wallace multiplier is accomplished through three distinct stages -

- Produce all the partial products.
- The partial products obtained from the multiplication operation are combined and reduced using full adders and half adders in a tree structure until only two terms are left.
- In the end, a quick adder is employed to sum up the two terms.

CHAPTER 3

Implementation of Binary Multiplier

The logic required for implementing Array multiplier, Booth multiplier, Baugh-Wooley multiplier and Wallace Tree multiplier is as follows -

3.1 Designing an Array Multiplier

The algorithm being discussed is similar to the conventional multiplication method that employs the add-and-shift technique, which can be applied to any number system. This algorithm involves using a combination of full adders and half adders arranged in an array to determine the product. Each bit of the multiplier is multiplied with the entire multiplicand input, resulting in multiple partial products that are obtained through sequential shifting. The final step is to sum all the partial products together to yield the multiplication result.

3.1.1 Construction and Working of a 4×4 Array Multiplier

The add shift algorithm principle serves as the basis for the design structure of the regular array multiplier.

partial product = Number of multiplicand bit $\times$ Number of multiplier bit

In an array multiplier, the product is calculated using AND gates, and the partial products are added using Full Adders and Half Adders, while the partial product is shifted based on their bit position. For an $M \times N$ array multiplier, $M \times N$ AND gates are used to generate partial products, and the addition of these partial products can be performed using $N \times (M-2)$ Full adders and N Half adders. In the case of a 4x4 array multiplier, there are eight inputs and eight outputs.

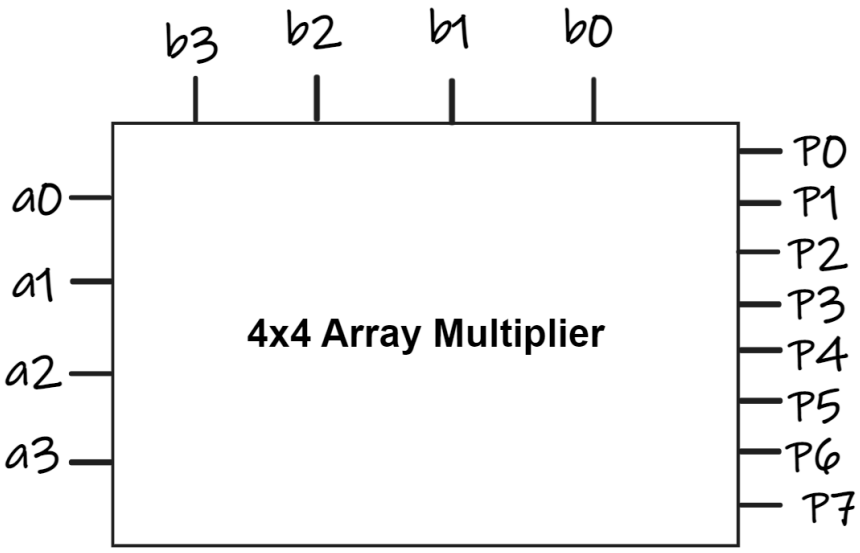


Figure 3.1: 4x4 bit Array multiplier

3.1.2 Building Blocks of 4×4 Array Multiplier

The full adder is a fundamental component in the design of array multipliers, having three input lines and two output lines. The following is an illustration of a 4x4 bit array multiplier, where the partial product's LSB is represented by the leftmost bit.

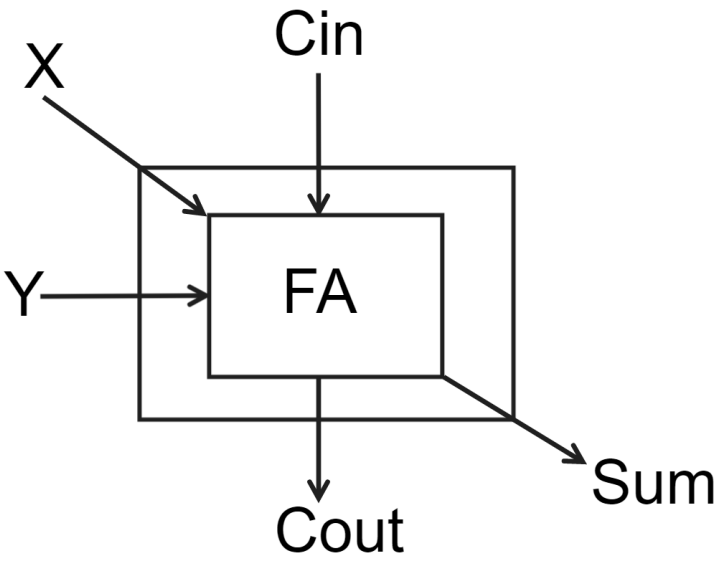


Figure 3.2: Full Adder Block Diagram for construction of Array multiplier

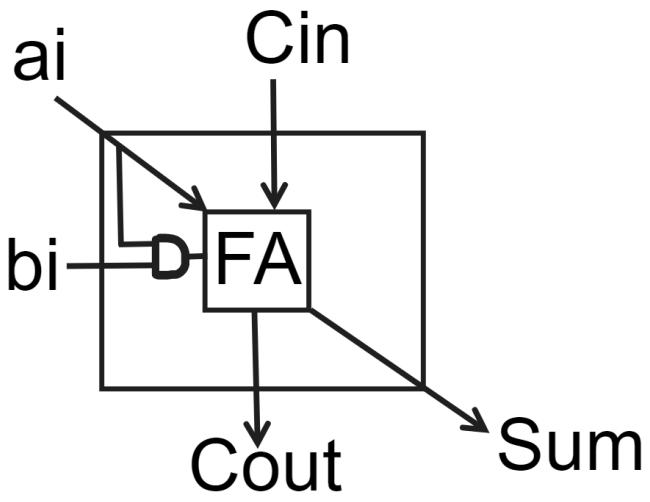


Figure 3.3: Array multiplier's Block Diagram

During multiplication, the partial products that are produced are shifted towards the left, and the most significant bit is positioned on the rightmost side of the partial product, while the least significant bit is on the leftmost side. These partial products are then added together to obtain the final product. This process is repeated until there are no more pairs of partial products to be added.

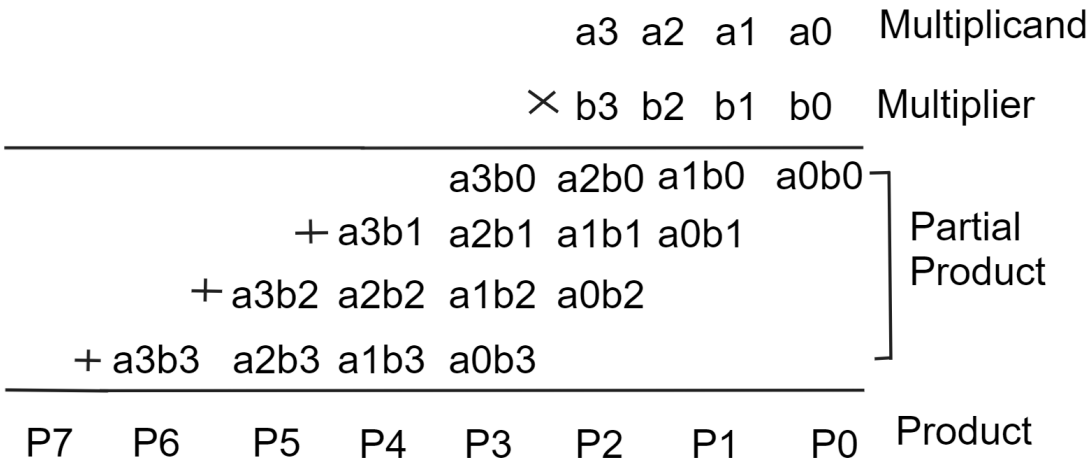


Figure 3.4: 4x4 bit Array multiplier Methodology

3.1.3 Working on the Array Multiplier Algorithm

When a Half Adder or Full Adder generates a carry in a previous stage, it must be propagated to the next stage. This carry propagation can occur in any Half Adder or Full Adder. When there are two additions in a column, two carries may be generated, and each of these carries must be propagated horizontally to the next column. The carry is always one bit position higher in the hierarchy, so the carry from one column must be propagated to the next column, while the carry from the next column must be propagated to the column after that. While the sum moves vertically, the carry must move horizontally to the next position.

The diagram presented illustrates a way to construct the 4x4 bit array multiplier using a combination of half adders and full adders. These adders play a crucial role in adding the partial products to obtain the ultimate output of the multiplication operation.

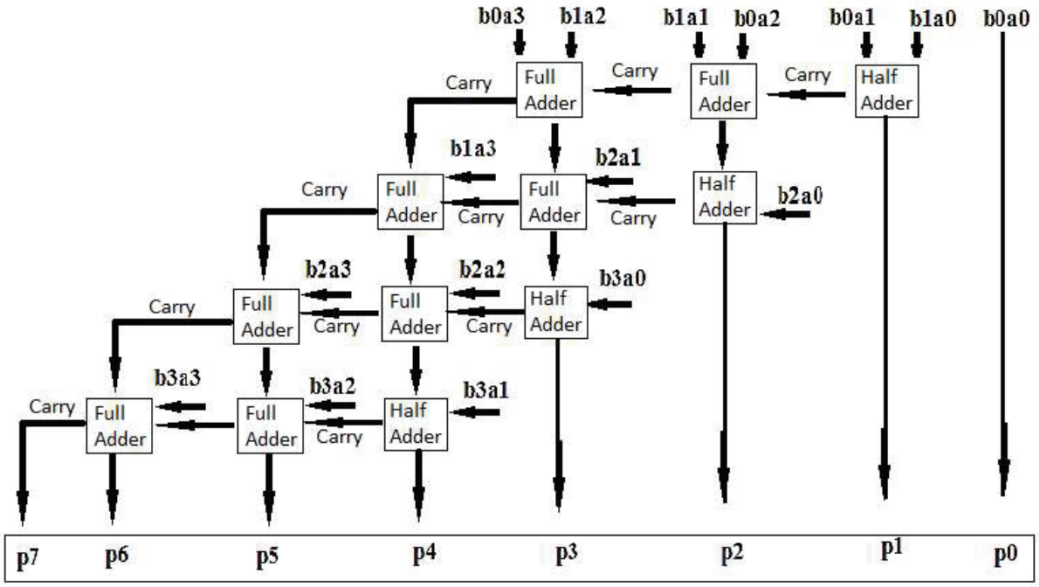


Figure 3.5: 4x4 bit Array multiplier Implementation[1]

The multiplicand and multiplier are represented as a0,a1,a2,a3 and b0,b1,b2,b3 respectively. The products obtained by multiplying each bit of the multiplier with the entire multiplicand input are the partial products. The sum of all partial products is the result of the multiplication.

A 4x4 Array multiplier requires 16 AND gates along with 4 Half Adders (HAs) and 8 Full Adders (FAs) to implement the multiplication process. In total, 12 adders are required.

Assume,

M = Number of bits in multiplicand

N = Number of bits in multiplier

then,

$M \times N$ = Number of partial Products

$M \times N$ = AND gates

N = Number of Half adders

$(M-2) \times N$ = Number of Full adders

$(M-1) \times N$ = Total Number of adders

T_c = Carry propagation time of an adder

T_s = Sum propagation time of an adder

T_A = Propagation time of AND gate

if, $T_c > T_s$

Total Multiplication time = $T_A + (M+N-2) \times T_c$

if, $T_c < T_s$

Total Multiplication time = $T_A + (N-1) \times T_s + (M-1) \times T_c$

3.1.4 Simulation Results of 64x64 bit Array Multiplier

This section displays the simulation results of the 64-bit binary unsigned array multiplier. The multiplier used in this simulation is a 64x64 bit array multiplier, which can be observed in the simulation waveform.

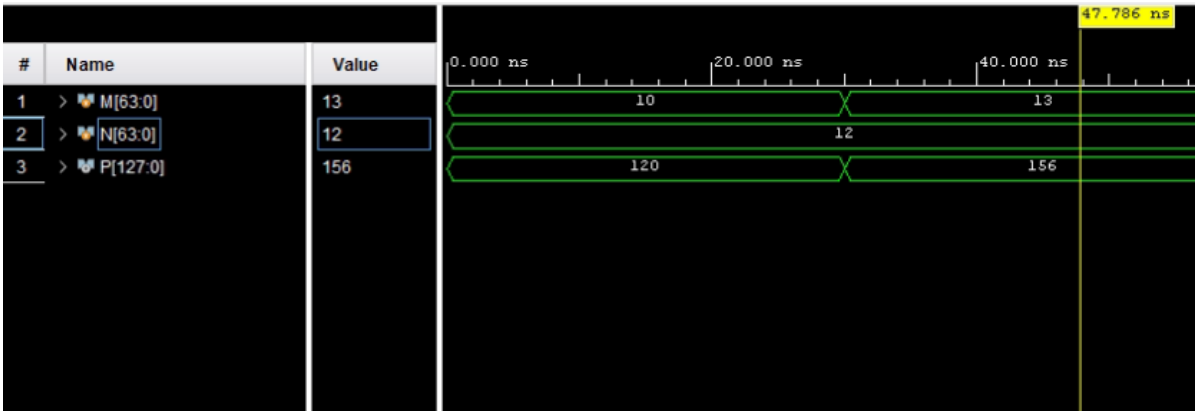


Figure 3.6: 64x64 bit Array multiplier simulation output waveform

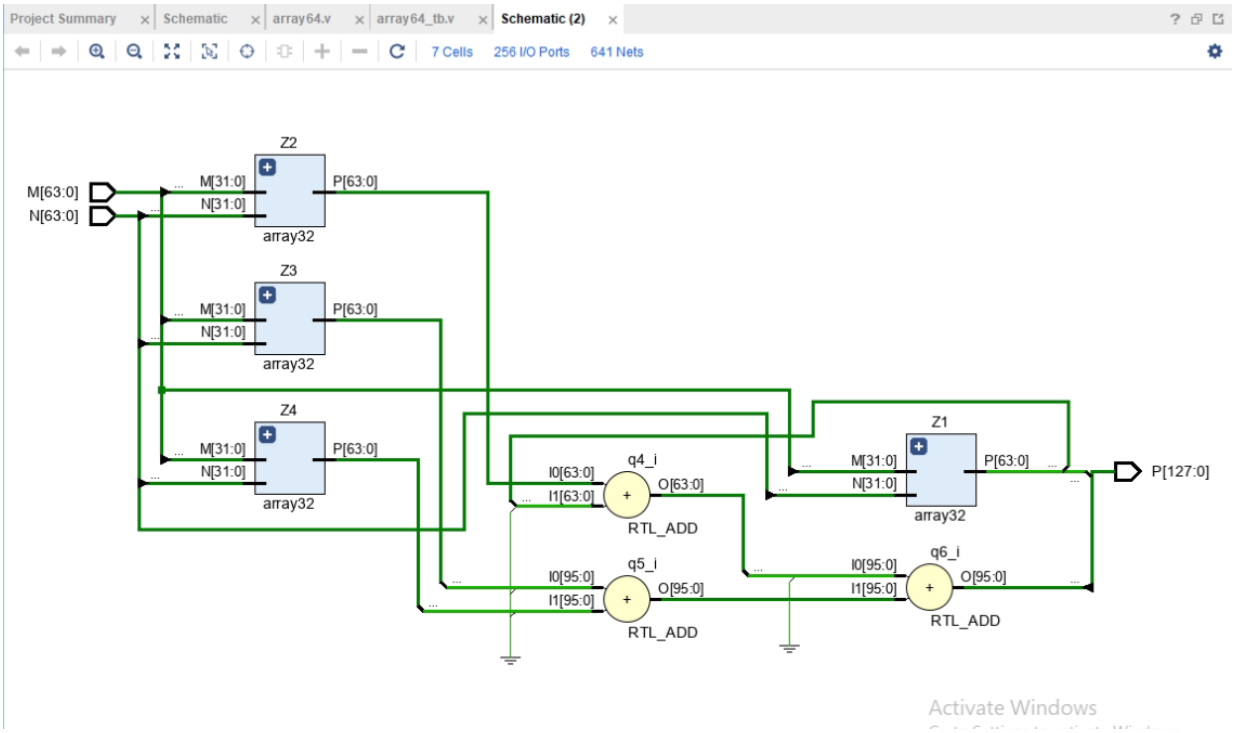


Figure 3.7: 64x64 bit Array multiplier schematic elaborated design

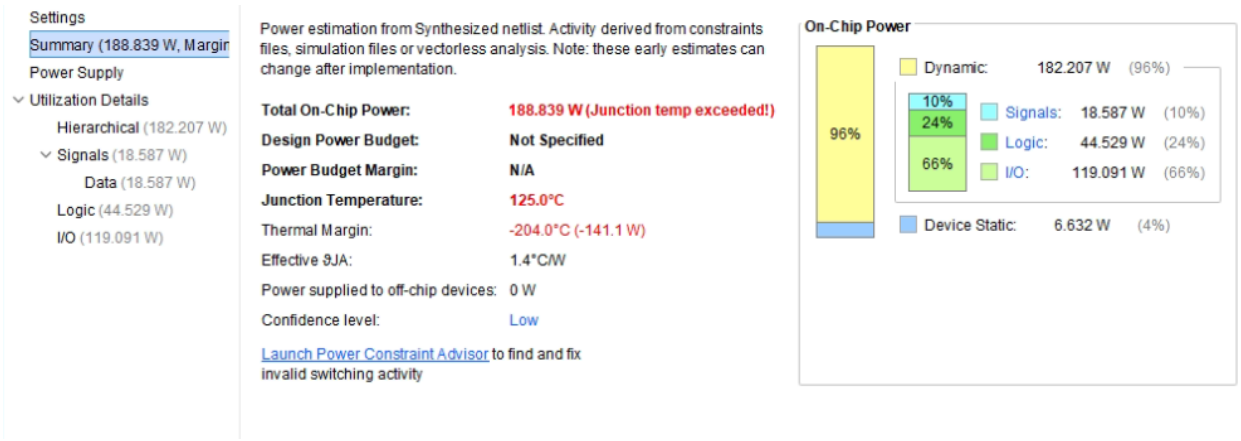


Figure 3.8: 64x64 bit Array multiplier power estimation from synthesized netlist

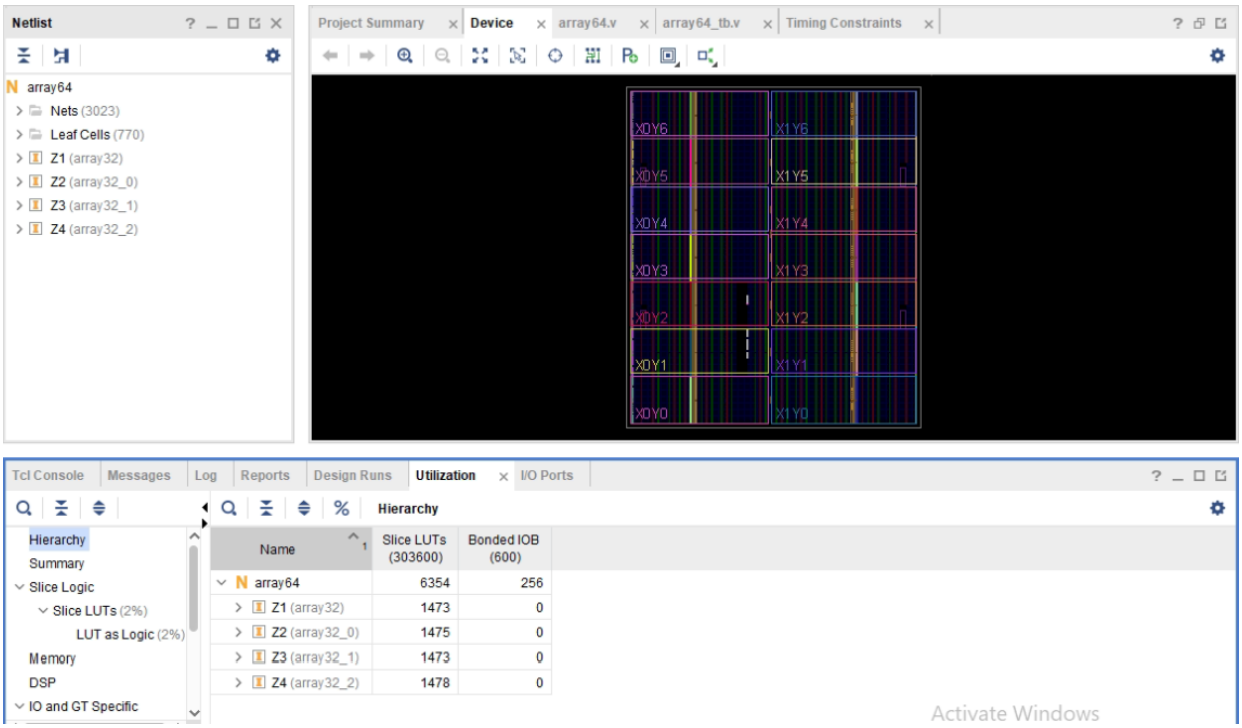


Figure 3.9: 64x64 bit Array multiplier Utilization Report

3.2 Designing a Booth Multiplier

The booth algorithm is a multiplication method used to multiply two signed binary integers represented in 2's complement. It is known for being highly efficient and can speed up the multiplication process. This algorithm works by analyzing the sequence of 0's in the multiplier, which only requires right-shift operations and no addition. It also examines the sequence of 1's in a multiplier bit weight from 2^k to 2^m, which can be simplified as $2^{k+1} - 2^m$.

Here is a visual depiction of Booth's Algorithm:

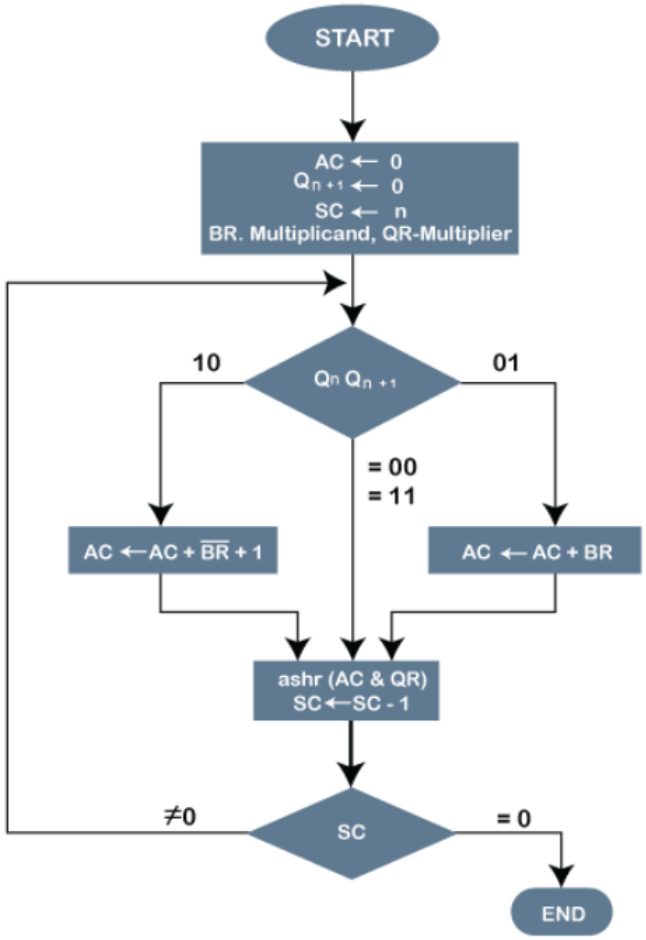


Figure 3.10: Booth's Algorithm Flowchart[1]

The flowchart presented above illustrates how the Booth algorithm works for multiplying two binary integers. Initially, the bits for AC and Q_{n+1} are set to 0, and SC is a sequence counter that represents the total number of bits set to n, which is equal to the number of bits in the multiplier. The multiplicand bits are represented by BR, and the

multiplier bits by QR. When two bits of the multiplier, Q_n and Q_{n+1}, are encountered, where Q_n represents the last bit of QR, and Q_{n+1} represents the incremented bit of Q_n by 1, the algorithm performs the following: If the two bits are $(10)_2$, it subtracts the multiplier from the partial product in AC and performs an arithmetic shift operation. If the two bits are $(01)_2$, it adds the multiplicand to the partial product in AC and performs an arithmetic shift operation that includes Q_{n+1}. The arithmetic shift operation shifts AC and QR bits to the right by one, while keeping the sign bit in AC unchanged. The sequence counter is decremented continuously until the computational loop is repeated equal to the number of bits (n).[1]

3.2.1 Working on the Booth Algorithm[1]

- Assign binary bits M and Q to represent the multiplicand and multiplier, respectively.
- At the start, we assign a value of 0 to both the AC and Q_{n+1} registers.
- The sequence counter, SC, is used to keep track of the number of bits in the multiplier (Q). It continuously decreases until it equals the number of bits (n) or reaches 0.
- Q_n refers to the final bit of the multiplier (Q), while Q_{n+1} represents the next bit after Q_n, incremented by 1.
- During each cycle of the Booth algorithm, the following parameters will be examined for the Q_n and Q_{n+1} bits:
  - If both Q_n and Q_{n+1} are either 00 or 11, we can perform an arithmetic shift right operation on the partial product AC. Additionally, we must increment the bits of Q_n and Q_{n+1} by 1 bit.
  - If the Q_n and Q_{n+1} bits are 01, we add the multiplicand bits (M) to the Accumulator register (AC). Then, we perform a right shift operation on both the AC and QR bits by 1.
  - If the Q_n and Q_{n+1} bits are 10, we subtract the multiplicand bits (M) from the Accumulator register (AC). Then, we perform a right shift operation on both the AC and QR bits by 1.
- The process continues until we reach the $n - 1$ bit in the Booth algorithm.
- The Multiplication binary bits' results will be saved in the AC and QR registers.

3.2.2 Methods used in Booth's Algorithm

RSC (Right Shift Circular)

"Right shift circular" refers to a bitwise operation in which the bits in a binary number are shifted one position to the right, and the rightmost bit is moved to the leftmost position, becoming the new most significant bit. This operation is also called a circular shift or a bitwise rotation.

The algorithm moves the rightmost bit of the binary number to the leftmost position and adds it to the beginning of the binary bits.

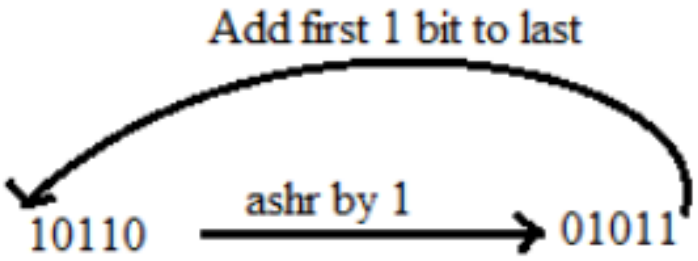


Figure 3.11: RSC (Right Shift Circular) for Booth multiplier

RSA (Right Shift Arithmetic)

"Right Shift Arithmetic" is a bitwise operation in which the bits in a binary number are shifted one position to the right. In contrast to a logical right shift, which shifts in 0's to the leftmost positions, a right shift arithmetic shifts in the sign bit to the leftmost positions. This maintains the sign of the original value and is often used in integer division by powers of two or in signed integer arithmetic operations.

The algorithm adds the two binary bits, and then the resulting sum is shifted one bit position to the right.

Example: 0100 + 0110 => 1010, After adding the binary number, shift each bit one position to the right, and place the first bit of the result at the beginning of the new bit.

3.2.3 Understanding Booth's Algorithm with Example

Example: Useing Booth's multiplication algorithm to multiply the numbers 23 and -9.

Here, M = 23 = $(010111)_2$ and Q = -9 = $(110111)_2$

Q_n	Q_{n+1}	M = 0 1 0 1 1 1 M' + 1 = 1 0 1 0 0 1	AC	Q	Q_{n+1}	SC
		Initially	000000	110111	0	6
1	0	Subtract M	101001			
			101001			
		Perform Arithmetic right shift operation	110100	111011	1	5
1	1	Perform Arithmetic right shift operation	111010	011101	1	4
1	1	Perform Arithmetic right shift operation	111101	001110	1	3
0	1	Addition (A + M)	010111			
			010100			
		Perform Arithmetic right shift operation	001010	000111	0	2
1	0	Subtract M	101001			
			110011			
		Perform Arithmetic right shift operation	111001	100011	1	1
1	1	Perform Arithmetic right shift operation	**111100**	**110001**	**1**	0

Figure 3.12: Understanding Booth's Algorithm with Example[1]

Q_{n+1} **= 1**, This implies that the output is a negative value.

Hence, 23$\times$ (-9) = 2's complement of $(111100110001)_2$ —> $(00001100111)_2$

3.2.4 Simulation Results of 64x64 bit Booth Multiplier

This section demonstrates the simulation of a 64-bit binary signed Booth multiplier. The simulation waveforms depict the 64x64 bit Booth multiplier.

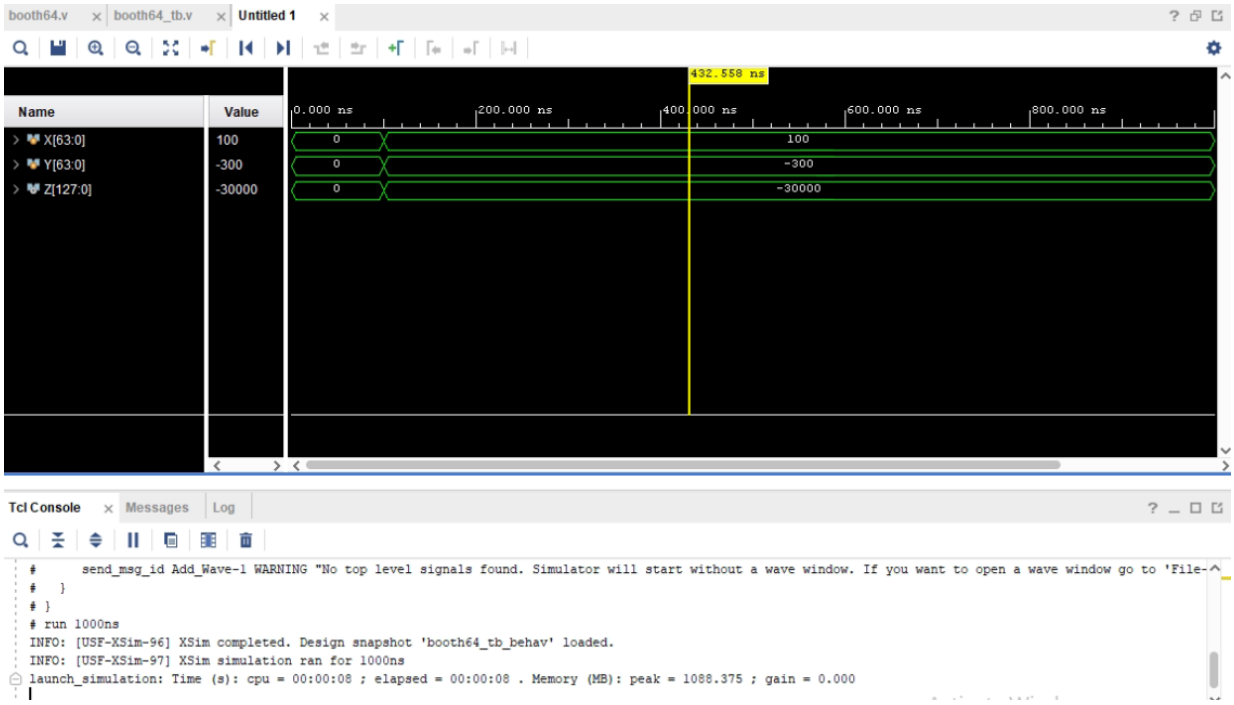


Figure 3.13: 64x64 bit Booth multiplier simulation output waveform

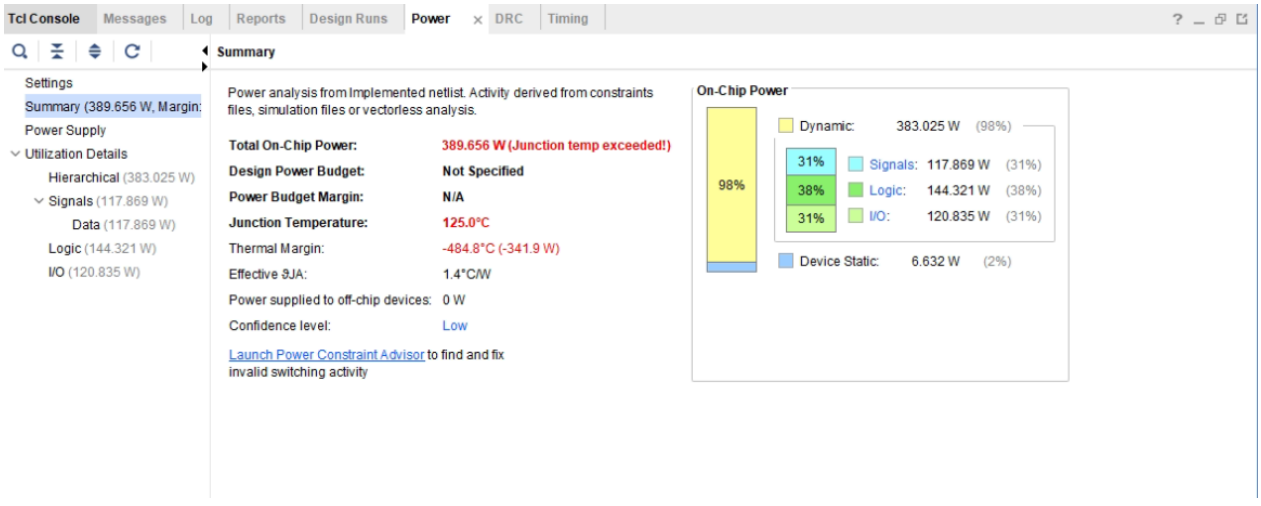


Figure 3.14: 64x64 bit Booth multiplier power estimation from synthesized netlist

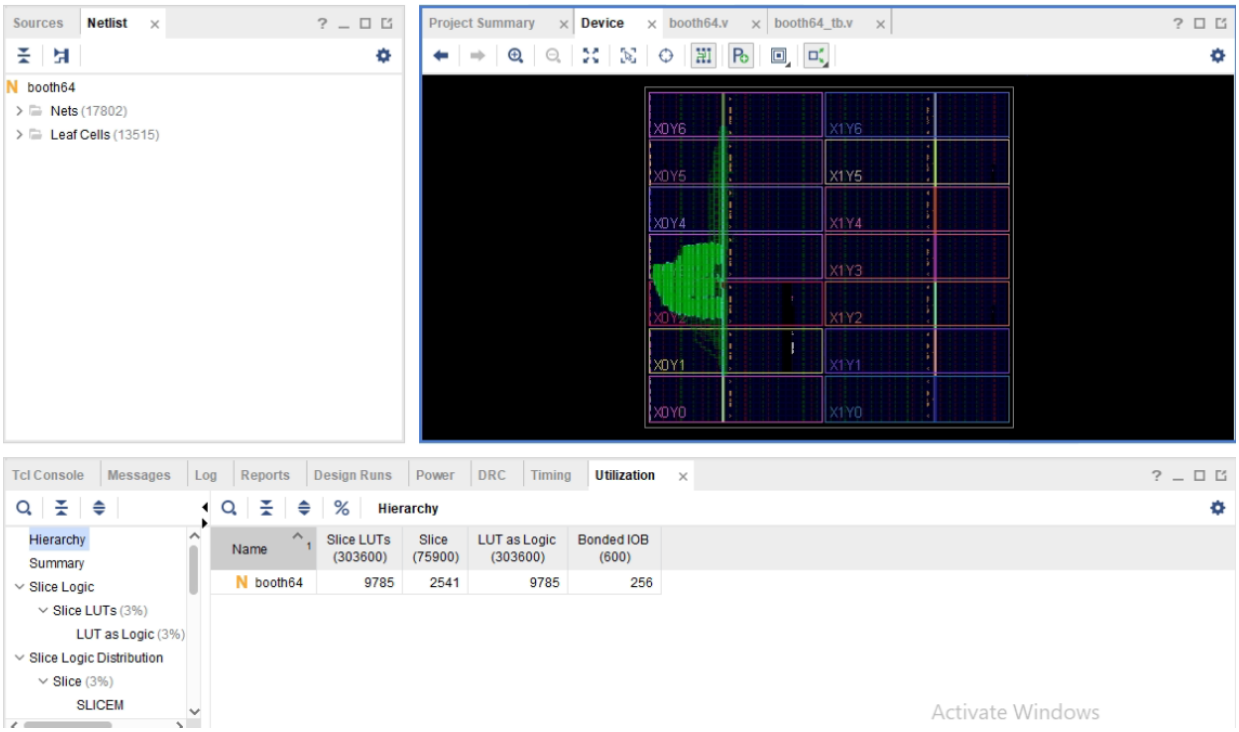


Figure 3.15: 64x64 bit Booth multiplier Utilization Report

3.3 Designing a Baugh-Wooley Multiplier

When carrying out multiplication with signed numbers, the number of partial products and their size can become extremely large. To overcome this, Baugh and Wooley devised an algorithm for directly multiplying two's complement arrays, which is an efficient way of dealing with sign bits. The major advantage of their approach is that all the summands have positive signs, which enables the array to be constructed completely with conventional full adders. This method was developed to create regular multipliers that are appropriate for 2's complement numbers.

The Baugh-Wooley approach was specifically developed for creating direct multipliers for Two's complement numbers. Since each partial product is a signed number when multiplying Two's complement numbers directly, it must be sign-extended to the final product's size to ensure accurate summation by the Carry Save Adder (CSA) tree. To avoid dealing with the negatively weighted bits in the partial product matrix, the Baugh-Wooley technique suggests adding additional entries to the bit matrix in a more efficient manner.[3]

Figure 3.16 & 3.17 depict partial product arrays for 5x5 bit representations of both Unsigned and Signed bits.

					a_4	a_3	a_2	a_1	a_0
					x_4	x_3	x_2	x_1	x_0
					a_4x_0	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				a_4x_1	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			a_4x_2	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		a_4x_3	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	a_3x_4	a_2x_4	a_1x_4	a_0x_4				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Figure 3.16: partial Product arrays of 5x5 bits Unsigned bit for Baugh-Wooley multiplier[3]

					a_4	a_3	a_2	a_1	a_0
					x_4	x_3	x_2	x_1	x_0
					$-a_4x_0$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$-a_4x_1$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$-a_4x_2$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$-a_4x_3$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	$-a_3x_4$	$-a_2x_4$	$-a_1x_4$	$-a_0x_4$				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Figure 3.17: partial Product arrays of 5x5 bits signed bit for Baugh-Wooley multiplier[3]

3.3.1 Building Blocks of 4×4 Baugh-Wooley Multiplier

Figure 3.18 presents the depiction of Baugh Wooley algorithm using Hatamian's approach, which can be broken down into three individual stages.[4]

- The Baugh Wooley algorithm involves inverting all bits of the last partial-product row, except for its most significant bit (MSB), as well as the MSB of the partial-products in each of the N-1 rows.
- A '1' is appended to the Nth column.
- The most significant bit (MSB) of the final outcome is inverted.

The HPM technique is a convenient and uncomplicated method to execute the Baugh Wooley multiplier. It can be effortlessly implemented by employing AND gates to compute partial products and NAND gates to compute inverted products. Figure 3.18 illustrates the block diagram of a 10-bit Baugh Wooley multiplier employing HPM, depicting the addition of "1" and the partial products.

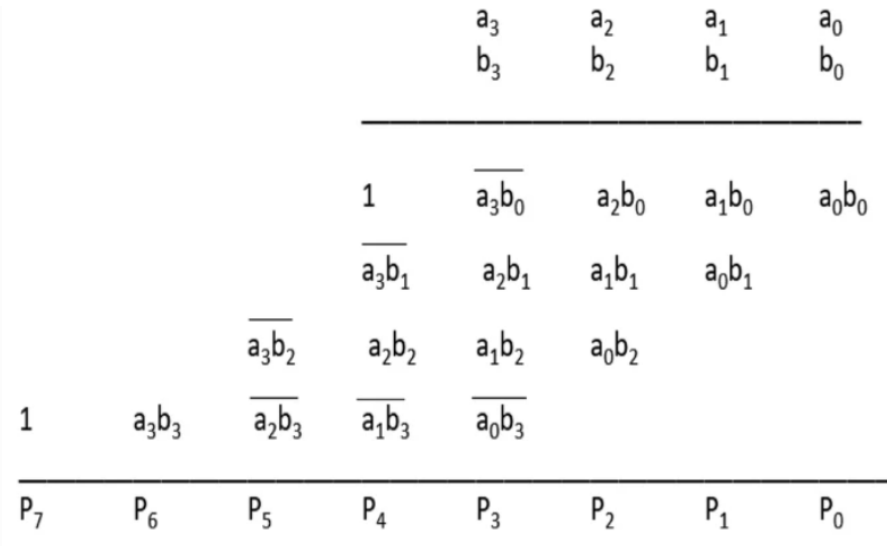


Figure 3.18: 4x4 bit Multiplication Example of Baugh-Wooley multiplier[4]

Figure 3.19 illustrates the block diagram of a 4-bit Baugh Wooley multiplier, while Figure 3.20 displays the comprehensive arrangement of each individual block.

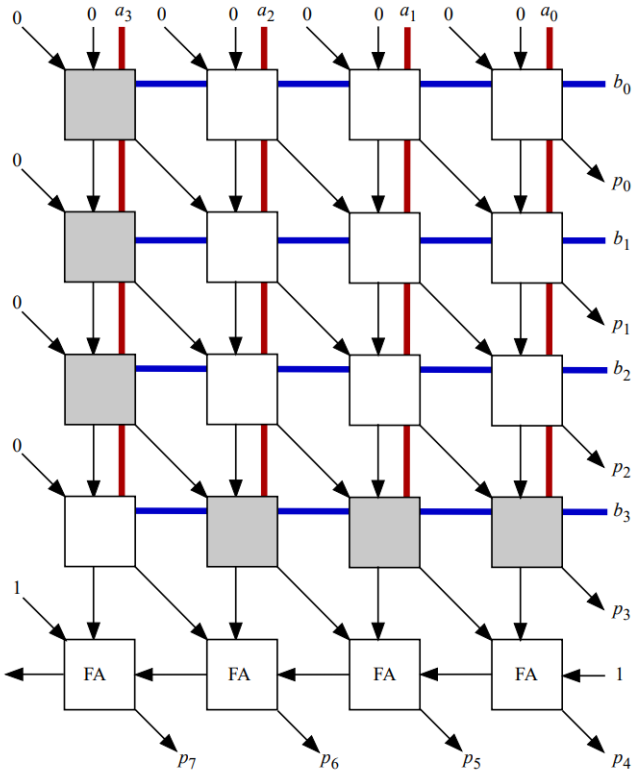
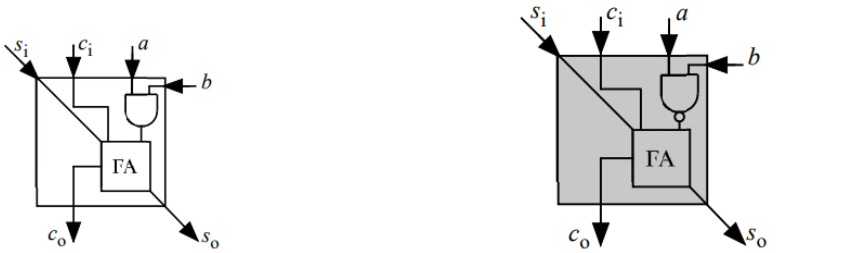


Figure 3.19: Block Diagram of 4x4 bit Baugh-Wooley multiplier[4]



(a) Baugh-Wooley multiplier white-cell (b) Baugh-Wooley multiplier gray-cell

Figure 3.20: Baugh Wooley multiplier basic cell construction[4]

3.3.2 Simulation Results of 8x8 bit Baugh-Wooley Multiplier

In this section, a simulation of the Baugh-Wooley multiplier for 8-bit binary signed numbers is demonstrated. The simulation waveforms depict the 8x8 bit Baugh-Wooley multiplier.

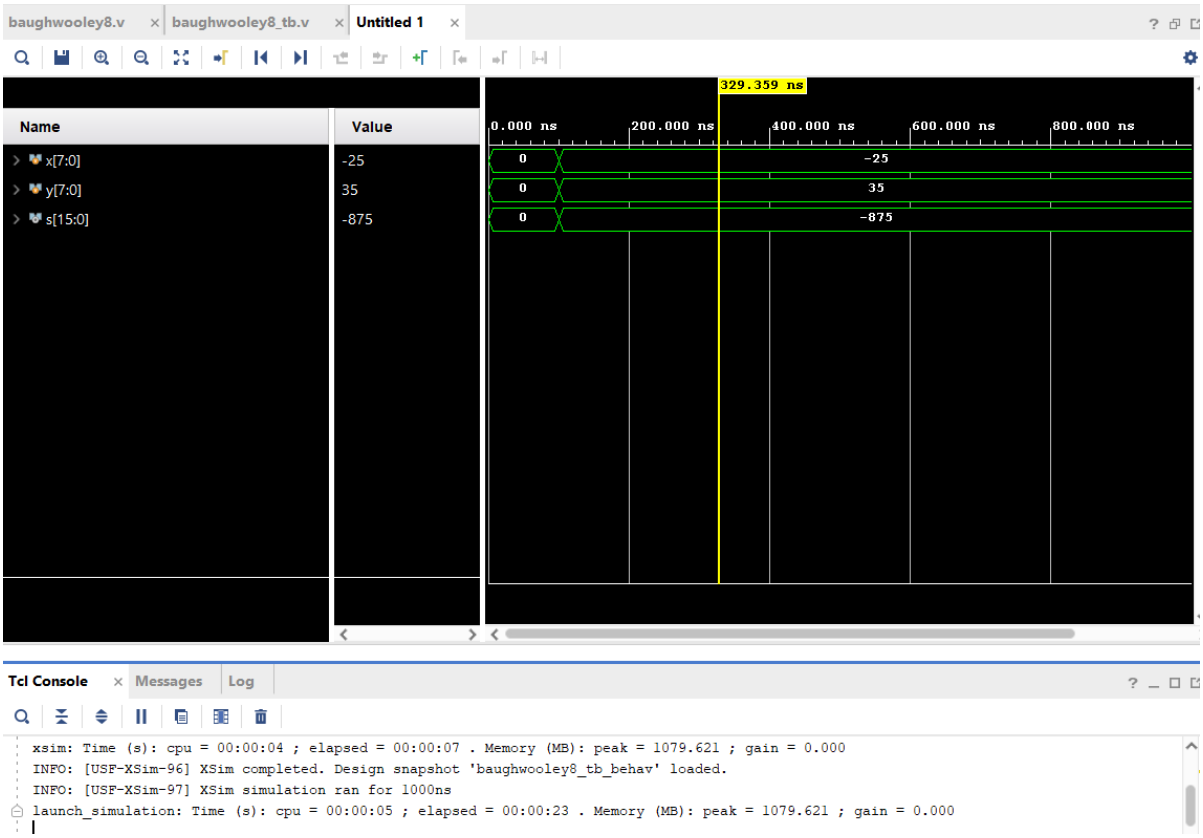


Figure 3.21: 8x8 bit Baugh-Wooley multiplier simulation output waveform

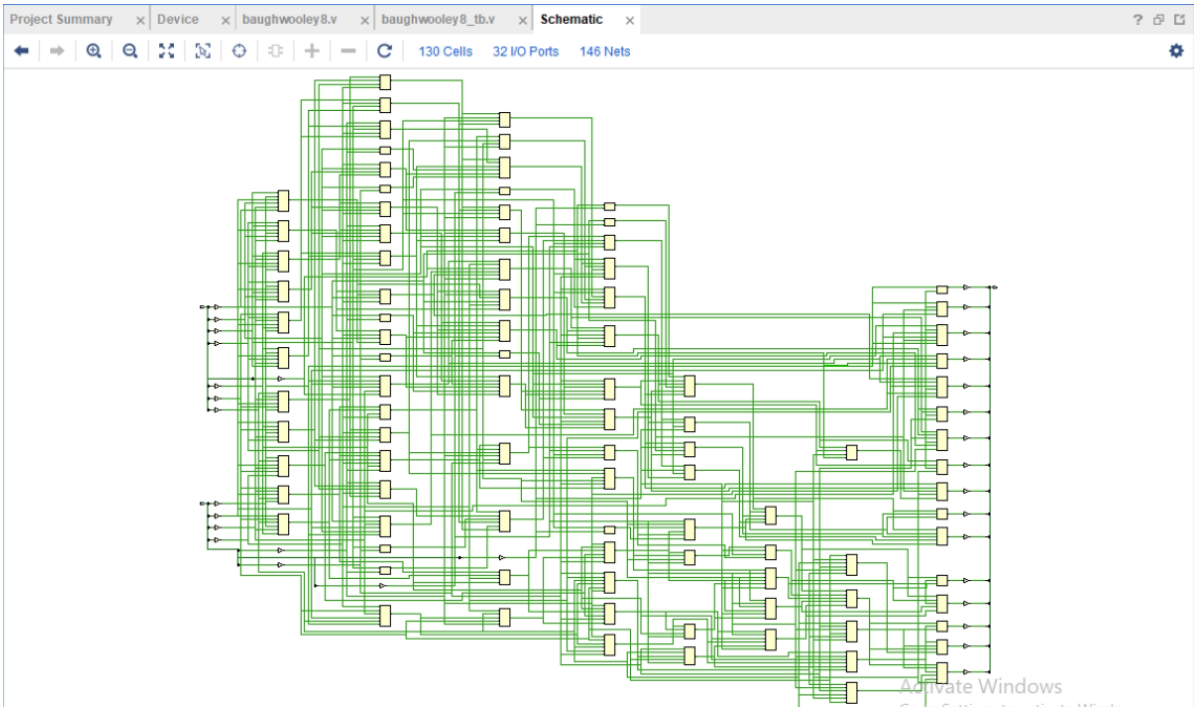


Figure 3.22: 8x8 bit Baugh-Wooley multiplier schematic elaborated design

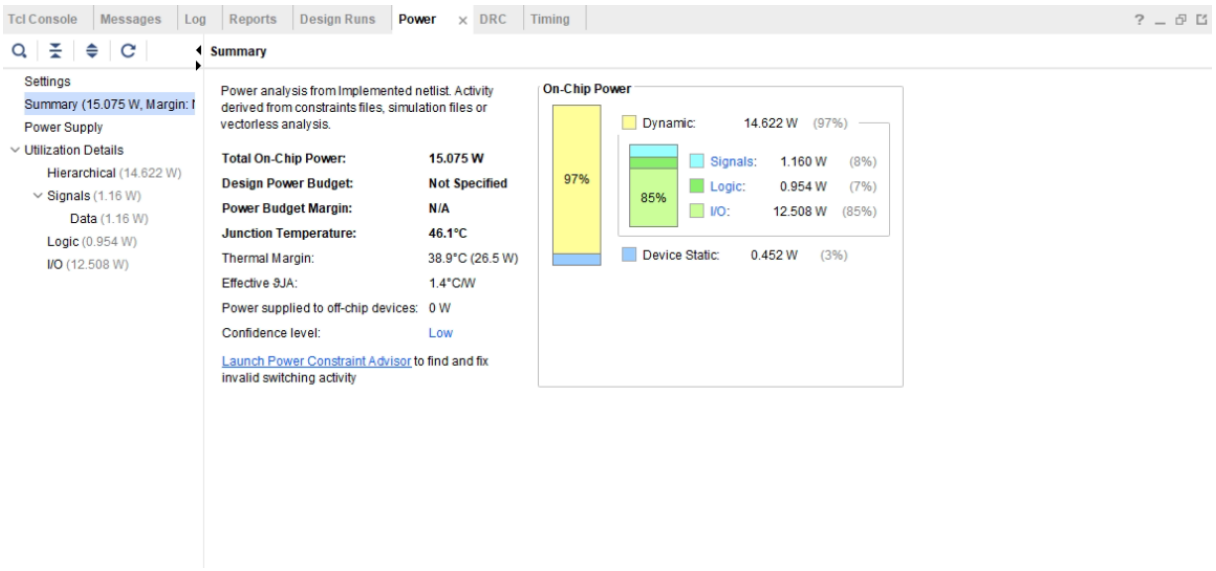


Figure 3.23: 8x8 bit Baugh-Wooley multiplier power estimation from synthesized netlist

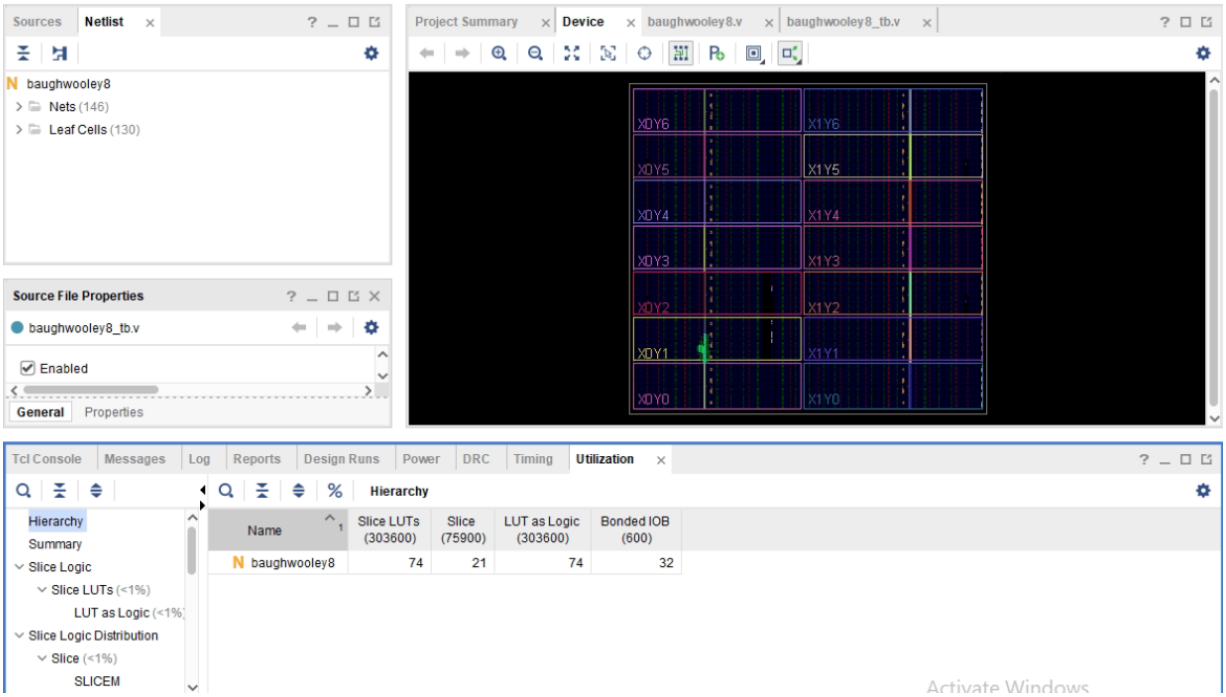


Figure 3.24: 8x8 bit Baugh-Wooley multiplier Utilization Report

3.4 Designing a Wallace Tree Multiplier

The Wallace tree architecture concurrently sums all the bits from each column's partial products by utilizing a collection of counters, without any carry-over. Afterward, another set of counters is utilized to reduce the resulting matrix, and this procedure is repeated until a two-row matrix is obtained. Full Adders such as the 3:2 counter are typically employed as counters. The final outcomes are then added using a fast adder.[5]

The operation of the Wallace multiplier is accomplished through three distinct stages -

- Produce all the partial products.
- The partial products obtained from the multiplication operation are combined and reduced using full adders and half adders in a tree structure until only two terms are left.
- In the end, a quick adder is employed to sum up the two terms.

3.4.1 Understanding logic behind Wallace Tree multiplier

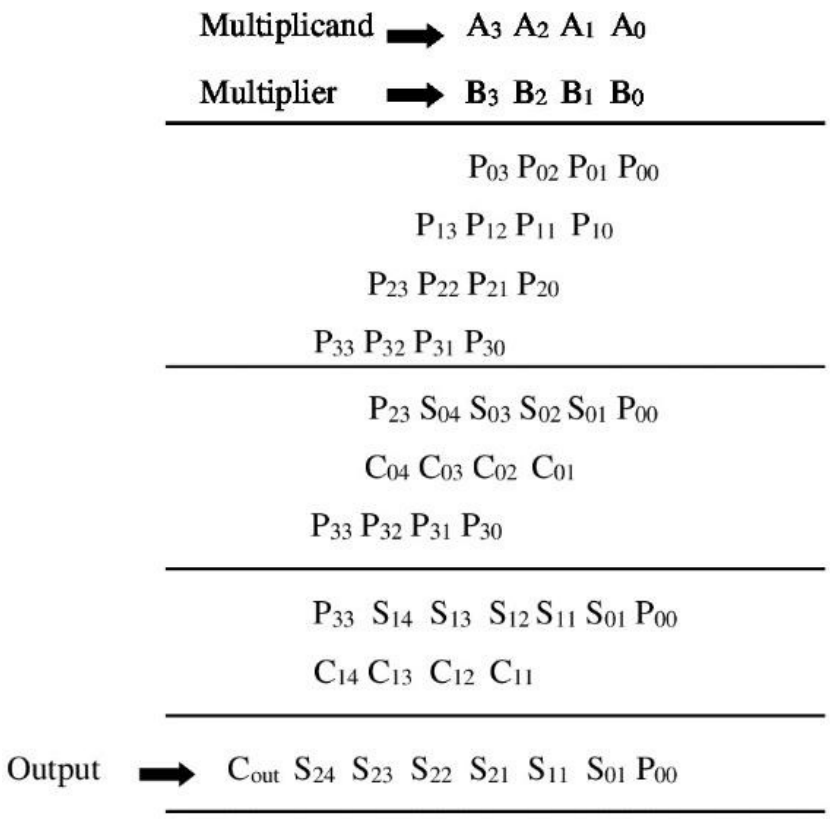


Figure 3.25: Operation of 4x4 bit Wallace Tree multiplier[5]

To begin the process, the first step involves generating partial products by multiplying every individual bit of the multiplier with the matching bit of the multiplicand. Following this, groups of three adjacent rows are merged, and the use of half adders and full adders is employed to reduce each group of three rows.

In the process of multiplication, each column consists of two or three bits. If a column has two bits, a half adder is employed to perform the addition, and if it has three bits, a full adder is used. For columns with only one bit, the bit is passed on to the next stage unchanged. This reduction process is repeated in each subsequent stage until only two rows remain. In the last stage, these two rows are added together, resulting in an 8-bit number as shown in Figure 3.26.[5]

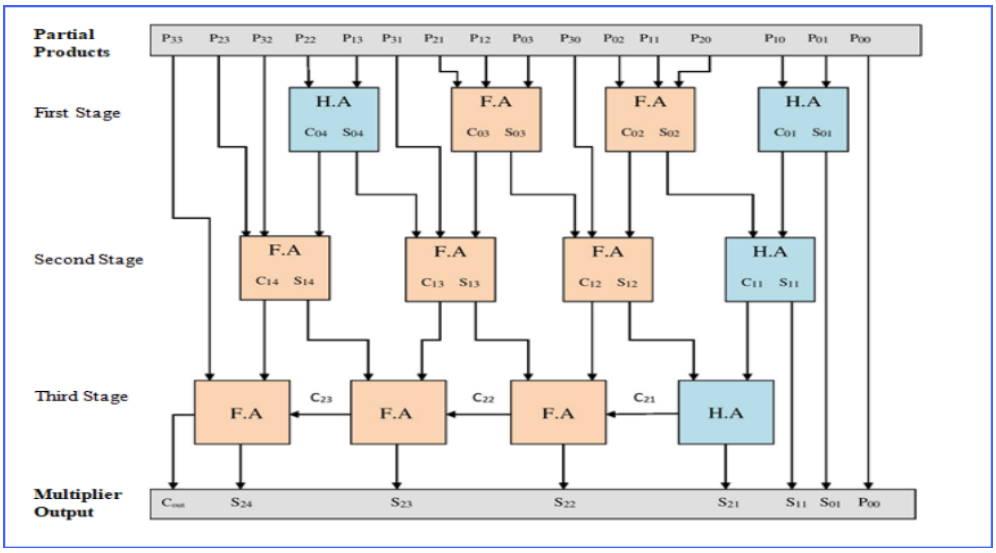


Figure 3.26: Wallace Tree multiplier using Half and Full Adder

3.4.2 Simulation Result of 4x4 bit Wallace Tree Multiplier

This section displays the simulation results of the 4-bit binary Wallace multiplier. The simulation waveforms depict the 4x4 bit Wallace multiplier.

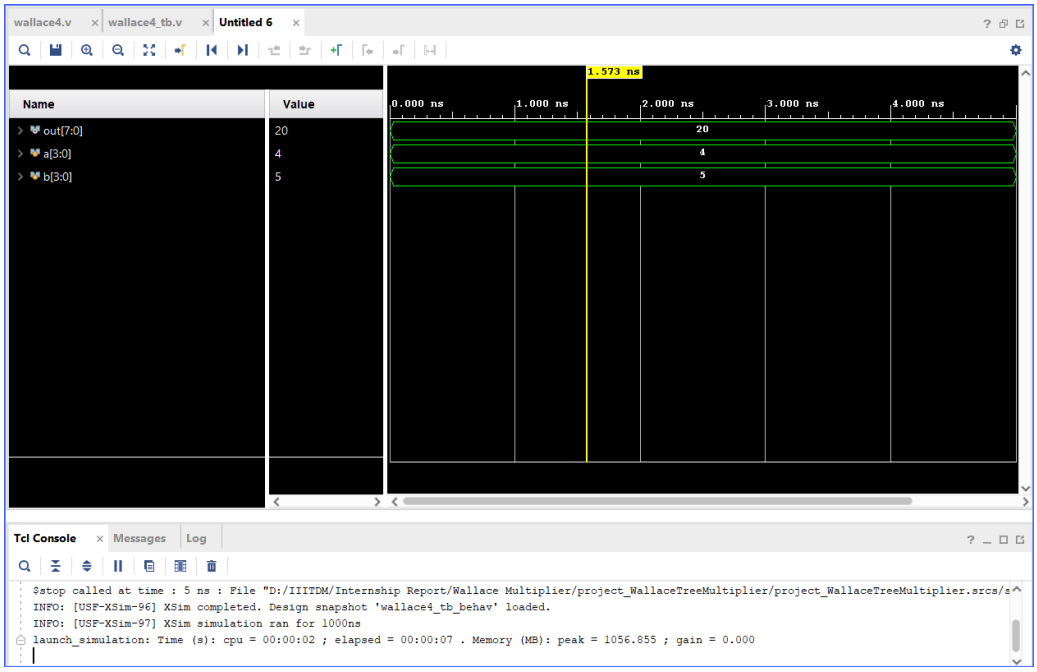


Figure 3.27: 4x4 bit Wallace Tree multiplier simulation output waveform

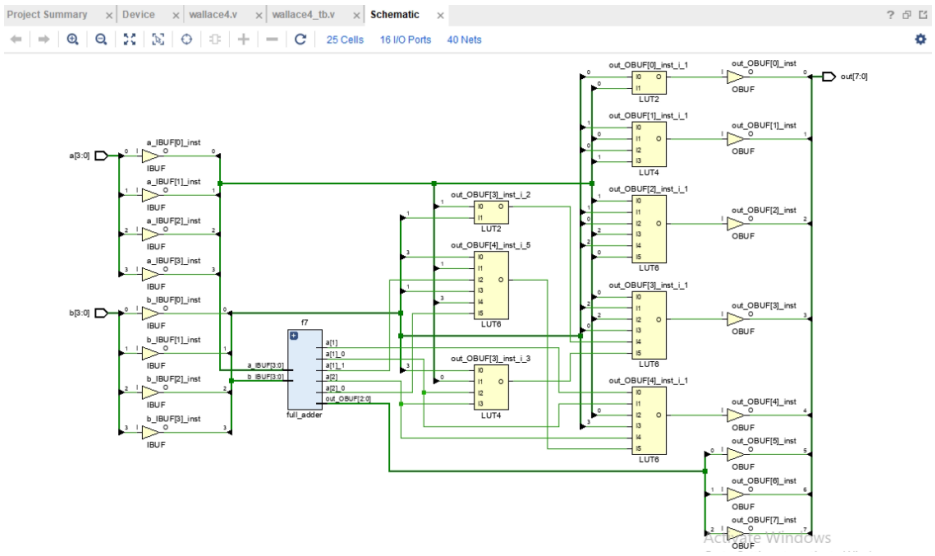


Figure 3.28: 4x4 bit Wallace multiplier schematic elaborated design

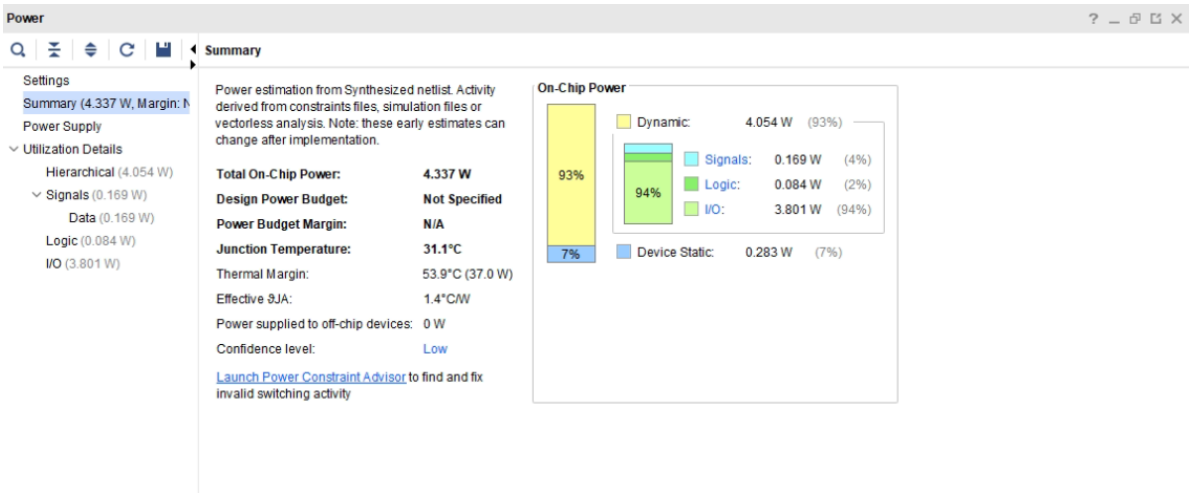


Figure 3.29: 4x4 bit Wallace multiplier power estimation from synthesized netlist

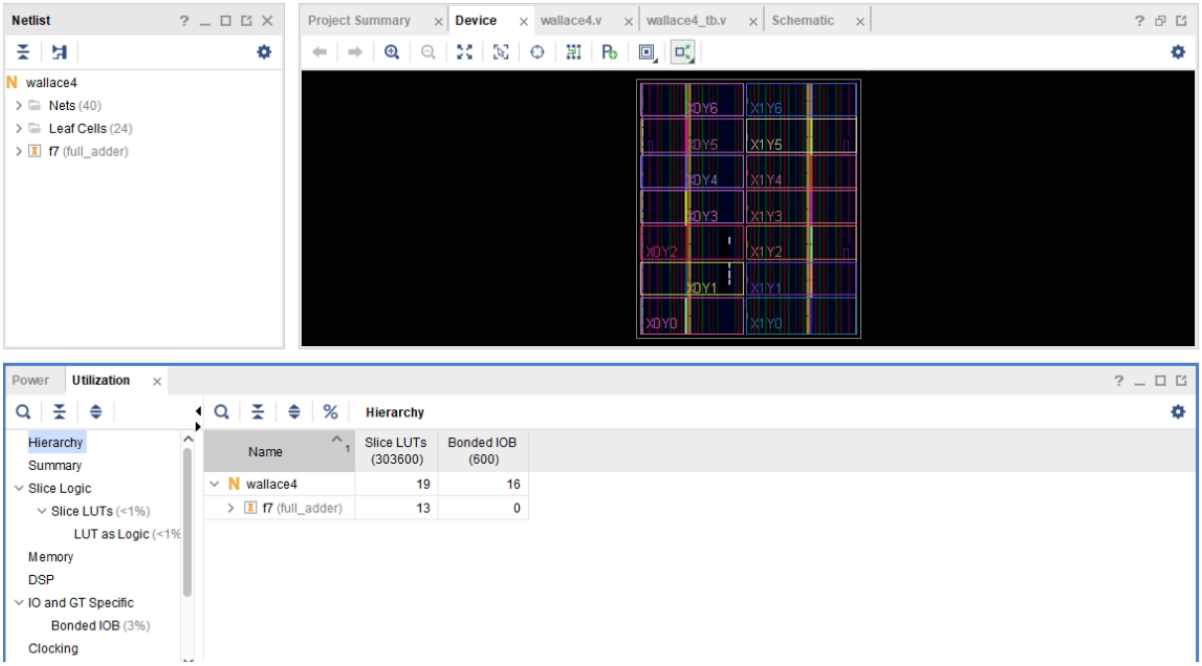


Figure 3.30: 4x4 bit Wallace multiplier Utilization Report

CHAPTER 4

Conclusions and Extensions

In this report we have implemented the different types of multipliers include Array multiplier, Booth multiplier, Baugh-Wooley multiplier, and Wallace multiplier. From these 4 multipliers Array multiplier and Booth multiplier are implemented in 64x64 bit in xilinx vivado 2020.2 using Verilog HDL. Baugh-Wooley is impleted in 8x8 bit and Wallace Tree multiplier is implemented in 4x4 bit in xilinx vivado 2020.2 using Verilog HDL.

By incorporating the multiplier into the arithmetic logical unit and multiply accumulator unit designs, as well as comparing the outcomes with those of similar designs, the project could be expanded even further.

REFERENCES

[1] A. Gunturu, "Analysis of booth's multiplier algorithm vs array multiplier algorithm and their fpga implementation," *Youngstown State University*, 2019.

[2] S. D. Kale and G. N. Zade, "Design of baugh-wooley multiplier using verilog hdl."

[3] R. Kumar and P. Kumar, "An efficient baugh-wooley multiplication algorithm for 32-bit synchronous multiplication," *International Journal of Advanced Engineering Research and Science (IJAERS)*, 2014.

[4] J. Antony and J. Pathak, "Design and implementation of high speed baugh wooley and modified booth multiplier using cadence rtl," *Int. J. Res. Eng. Technol 3.8 (2014): 56-63*, 2014.

[5] M. Esa and K. Achyut, "Design and verification of 4 x 4 wallace tree multiplier," *International Journal of Analytical and Experimental Modal Analysis (IJAEMA) 11.10 (2019)*, 2019.