ABSTRACT

This project aimed to design and develop a high-speed multiplier for Digital Signal Processing (DSP) applications. The study began by understanding the basic process of multiplication and the importance of signed multiplication in DSP. Various multiplier architectures were compared, and the Bough-Wooley multiplier architecture was identified as the most suitable choice for the targeted application. The architecture was analyzed on an FPGA level, and computational results were obtained. Addition methods for the Bough-Wooley multiplier architecture were explored, and it was discovered that the propagate and generate based approach yielded the best results for optimized resource usage and high speed. The designed architecture was implemented on Cadence Virtuoso, providing a valuable tool for high-speed multiplication in DSP applications. Overall, this study provides a comprehensive understanding of the design and development of high-speed multipliers for DSP applications, with practical implications for real-world use.

TABLE OF CONTENTS

LIST OF TABLES

.

LIST OF FIGURES

.

Chapter 1

INTRODUCTION

1.1. ROLE OF MULTIPLIER IN DSP

Multiplication is the key circuit block in modern day chips. This block is used in digital computing systems, controllers, filters, signal processors and various logic blocks as well. Since technology is updating every moment it is our prime need to implement fast multiplier in modern days VLSI chips. Presently there is high expansion in computer applications such as Artificial intelligence, Computer graphics, Filter application and many more. VLSI technology has evolved in so many years which resulted in implementation of complex technology easily. A normal multiplication has two operations first is to generate the partial products and second is to add those partial products. If we want to implement a fast multiplier then we must reduce the number of partial products which are generated, or we must make their addition much faster.

In many Digital Signal Processing (DSP) applications, including convolution, Fast Fourier Transform (FFT), filtering, and in microprocessors in their arithmetic and logic unit, multiplication-based operations like multiply and Accumulate (MAC) and inner product are among the frequently used Computation-Intensive Arithmetic Functions (CIAF). A high speed multiplier is required since multiplication takes up the majority of the time spent running most DSP algorithms. The instruction cycle time of a DSP processor is now still mostly influenced by multiplication time. The expansion of computer and signal processing applications has led to an increase in the demand for high speed processing. Higher throughput arithmetic operations are necessary to attain the needed performance in many real-time signal and image processing applications. Multiplication is a crucial mathematical operation in many applications, and the creation of quick multiplier circuits has long been of interest. For many applications, minimizing time delays and power consumption is a crucial requirement. Optimizing at all design stages is necessary to reduce power consumption for digital systems. The technology utilised to create the digital circuits, the style and topology of the circuits, the architecture for doing so, and at the highest level, the algorithms being employed, is all included in this optimization. The most often utilized elements in any digital circuit design are digital multipliers. They are used to carry out any operation since they are quick, dependable, and effective components.

In electronics, a combinational path is a logical circuit that produces an output based solely on the values of its inputs. Combinational path delay is the amount of time it takes for the output of a combinational path to change in response to a change in its input.

The delay is caused by the propagation of signals through gates and wires, which have inherent delays due to their physical characteristics. The total combinational path delay is the sum of the delays of all the gates and wires in the path.

Delay can vary depending on factors such as the type of logic gate, the distance the signal must travel, and the resistance and capacitance of the wire. The speed of the logic gates themselves can also affect the delay.

Delay can be minimized by optimizing the circuit design, choosing the right type of gates and interconnects, and reducing the distance the signal must travel. However, it is not always possible to completely eliminate delay, and designers must take into account the worst-case delay when designing the circuit. Understanding combinational path delay is important for our design.

1.2. PROBLEM STATEMENT

Typical DSP algorithms like filtering or transformations involve multiplication as a primary process to generate a desired output. Multiplication being a critical block requires larger time to compute and produce results. Improving the performance of a multiplier improves the overall performance of a DSP block.

1.3. OBJECTIVES

- ✓ To design and develop a high speed multiplier block for DSP application.
- ✓ To analyse the power and delay characteristics on FPGA level using Xilinx ISE/Vivado Design Suits.
- ✓ To design and analyse the schematic of optimized high speed multiplier on Cadence Virtuoso platform and validate the performance matric.

1.4. ORGANIZATION OF REPORT

Here the abstract of the report explains the full flow of project and then we have moved on to chapter 1 that consists of introduction which consists of role of multiplier in DSP, problem statement , objectives. Post this the flow explains the literature review. The literature review is tabulated in chapter 2 which also consists of the literature summary. Further the design of multiplier for DSP application is explained in detail in chapter 3 which includes motivation for signed multiplication, different multiplier algorithms involved, Baugh-wooley multiplication algorithm and multiplication operation in DSP and finally the design of various full adders. This is followed by methodology and results and discussion in chapter 4 and 5 respectively. These consists of experimental set up, simulation results, comparative study and implementation details of both Xilinx ISE design software and cadence virtuoso respectively. Finally we have the results and discussion in chapter 6 and references in chapter 7 associated with the project.

Chapter 2

LITERATURE REVIEW

The following table consists of all the research papers that we have gone through in the project and are tabulated as follows,

SI. No	Title of Paper	Technique Used	Description	Technology Used	End Application
[1]	High-Performance Accurate and Approximate Multipliers for FPGA-based Hardware Accelerators	Bough-Wooley's Multiplication Algorithm	This proposed unsigned and signed accurate architecture provides up to 25% and 53% reduction in LUT Utilization.	7VX330T device of Virtex-7 family using Xilinx Vivado 17.4.	Image and Video Processing
[2]	Implementation of Bit Parallel Finite Field Multipliers on FPGA	Karatsuba–Ofmn multipliers (KOM)	Evaluate the LUT complexity and area-time product trade offs on FPGAs	Virtex-4 like V4LX40ff1148- 12 and V5LX50ff1153- 3, synthesized with Xilinx ISE	For efficient FPGA Implementation
[3]	FPGA Based Optimized Design of Montgomery Modular Multiplier	Montgomery Modular Multiplier (MMM), Elliptic Curve Cryptography (ECC)	Novel architecture of the proposed design enhanced the maximum frequency of design	The proposed design is coded in VHDL, and targeted Virtex- 6,	System on Chip (SoC) and IoT applications
[4]	Theoretical Modeling of Elliptic Curve Scalar Multiplier	Elliptic Curve Scalar Multiplier (ECSM)	Paper illustrates suitable scheduling	Xilinx Virtex 5	For high-speed Designs.
[5]	Design and Implementation of Fast Booth-2 Multiplier on Artix FPGA	Redundant Binary Signed Digit (RBSD) Booth Multiplier	Designed a fast multiplier which enables fast computation by reducing carry propagation time as carry is not propagated	Xilinx Artix FPGA with part number xc7a200tisbv4 and implemented using XILINX Vivado 2019.3	For efficient FPGA Implementation

SI. No	Title of Paper	Technique U sed	Description	Technology Used	End Applica tion
[6]	Throughput/A rea Efficient ECC Processor using Montgomery Point Multiplication on FPGA	Segmented Pipe-lining Based Digital Serial Multiplier	In this paper,an ECC processor Architecture over Galois Fields presented that achieves the best reported throghput/area performance on FPGA.	Implemented on Xilinx Virtex 4 , Virtex5 and Virtex7	For Elliptic Curve Cryptog raphy (ECC) hardwar e impleme ntations
[7]	Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials	Karatsuba Multiplier and Comba multiplier	Work presented a flexible and scalable template architec ture for the hardware implementation of large binary poly nomial multipliers.	Xilinx Artix-7 mid-range family which include Artix 12 and Artix 200 FPGA Boards	To design quantum resistant public-key cryptosy stem
[8]	Low-Delay FPGA-Based Implementatio n of Finite Field Multiplier	Low Delay FPGA Based Polynomial Multiplier	This exhibits the best delay, with a delay improvement of up to 4.7%, and the second best Area × Time	Implemented with Xilinx ISE 14.7 using XST synthesizer in the Artix-7 XC7A200T-FFG1156	For efficient FPGA Implem entation
[9]	Efficient FPGA-Implementatio n of Two's Complement Digit-Serial/Parallel Multipliers	Digit-serial/parallel Multipliers	Efficient implementation of digit-serial/parallel multipliers on 4-input look-up table	Intel (Altera) Stratix V 5SGXMA9N1 F45C2)	DSP applicati ons

The above papers reflects the literature review conducted and though which we could identify that Baugh-wooley multiplication algorithm is the most suitable for signed DSP operations by considering the area and delay parameters.

Chapter 3

DESIGN OF MULTIPLIER FOR DSP APPLICATION

3.1. MOTIVATION FOR SIGNED MULTIPLICATION

In digital signal processing (DSP) applications, signed multipliers are commonly used to perform various computational tasks. The motivation for using signed multipliers stems from the fact that many signals and data in DSP systems are represented using signed number formats.

Here are some key motivations for using signed multipliers in DSP applications:

- ✓ Representation of Signed Numbers: In DSP, it is often necessary to represent both positive and negative values. Signed number formats, such as two's complement, allow efficient representation of negative numbers. Signed multipliers are designed to handle these signed number representations and perform arithmetic operations accurately.
- ✓ Signal Processing Operations: DSP applications involve a wide range of signal processing operations, such as filtering, convolution, correlation, and modulation. These operations require multiplication of signed numbers. By using signed multipliers, these operations can be performed efficiently and accurately.
- ✓ Dynamic Range: Signed multipliers provide a wider dynamic range compared to unsigned multipliers. In DSP systems, signals often have a wide range of amplitudes, and signed multipliers enable precise multiplication without losing significant bits of information, allowing for higher precision and accuracy in the signal processing operations.
- ✓ Error Reduction: DSP algorithms often involve iterative calculations and feedback loops, where accumulated errors can impact the overall accuracy of the system. Using signed multipliers helps in minimizing the effects of errors introduced during multiplication. By maintaining the sign information, the error accumulation can be properly handled, leading to improved accuracy.
- ✓ Compatibility with Standard Formats: Many DSP algorithms and standards utilize signed number formats, such as the International Telecommunication Union-Telecommunication (ITU-T) G.711 standard for speech coding. By using signed

multipliers, DSP systems can efficiently implement these standards and ensure compatibility with existing systems and protocols.

The motivation for using signed multipliers in DSP applications lies in their ability to handle signed number representations, perform a wide range of signal processing operations accurately, provide a wider dynamic range, minimize error accumulation, and ensure compatibility with standard formats and we have the following multiplier architectures available.

3.2. DIFFERENT MULTIPLIER ALGORITHMS INVOLVED

In digital signal processing (DSP), multiplication is a fundamental operation that is used in various applications such as filtering, modulation, and signal analysis. There are different multiplier algorithms used in DSP, each with its own advantages and disadvantages. One such algorithm is the Baugh-Wooley multiplication algorithm.

1. Basic Multiplier Algorithm:

The basic multiplier algorithm, also known as the "schoolbook" algorithm or the "long multiplication" algorithm, involves multiplying each digit of one number with each digit of the other number and adding the partial products to obtain the final result. While this algorithm is simple to understand and implement, it is computationally expensive and requires a large number of clock cycles to perform multiplication.

2. Booth's Algorithm:

Booth's algorithm is a multiplication algorithm that reduces the number of partial products by using a recoding scheme. It is more efficient than the basic multiplier algorithm and requires fewer clock cycles to perform multiplication. Booth's algorithm is commonly used in hardware implementations of DSP systems.

3. Wallace Tree Multiplier:

The Wallace tree multiplier is a hardware-oriented multiplication algorithm that reduces the number of partial products and the number of stages required for addition. It uses a combination of compressors and adders to perform multiplication efficiently. The Wallace tree multiplier is often used in high-speed DSP applications.

4. Baugh-Wooley Algorithm:

The Baugh-Wooley algorithm is a signed multiplication algorithm specifically designed for binary numbers represented in two's complement form. It leverages the properties of two's complement arithmetic to reduce the number of partial products and eliminate redundant calculations. This algorithm is particularly useful in DSP applications that involve signed number processing

The significance of using the Baugh-Wooley multiplication algorithm in DSP applications lies in its ability to reduce computational complexity and improve efficiency. By eliminating redundant calculations, the Baugh-Wooley algorithm reduces the number of partial products and hence the overall number of additions required. This leads to faster multiplication operations, reduced power consumption, and more efficient hardware implementations.

The Baugh-Wooley multiplication algorithm is specifically designed for signed binary numbers and offers improved efficiency in DSP applications by reducing computational complexity and optimizing hardware implementations.

3.2.1. BAUGH-WOOLEY MULTIPLICATION ALGORITHM

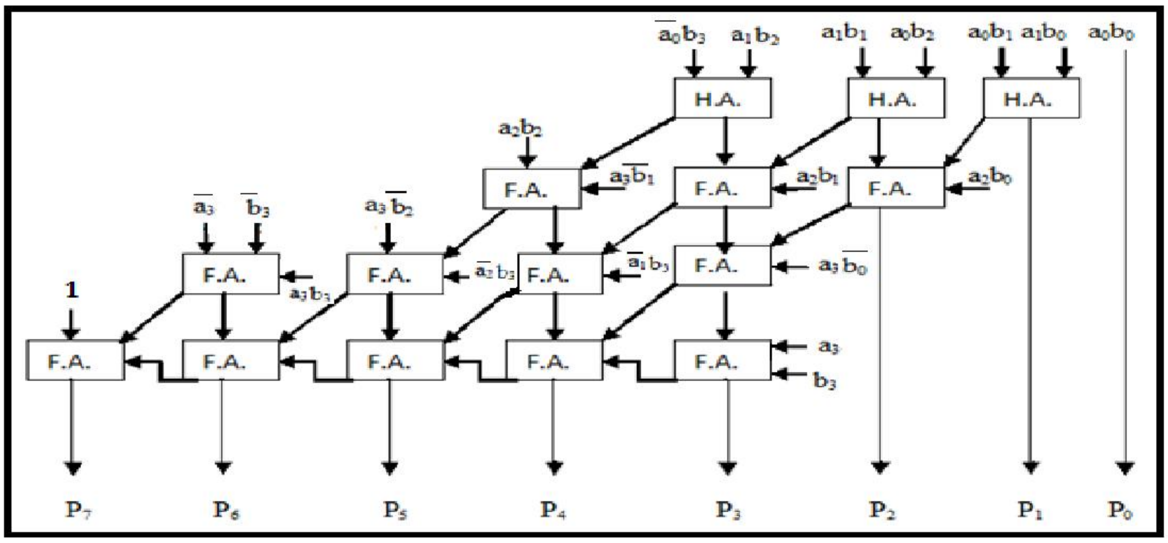


Figure 3.2.1. The Architecture of 4x4 Baugh-Wooley Multiplier

The Baugh-Wooley multiplier is a type of digital circuit that performs multiplication on two binary numbers. It is commonly used in microprocessors, digital signal processors, and other digital systems. The architecture of the Baugh-Wooley multiplier is based on the 2's complement multiplication algorithm.

The Baugh-Wooley multiplier is a partial product generator that generates partial products and adds them together to produce the final result. It consists of two main components: a partial product generator and a final adder.

The partial product generator takes two binary numbers, A and B, and generates partial products by multiplying each bit of A with each bit of B. The multiplication of two bits can result in three possible outcomes: 0, 1, or -1. To simplify the addition of partial products, the partial product generator produces only two types of partial products: positive and negative. Positive partial products are produced when two bits are multiplied and the result is either 0 or 1. Negative partial products are produced when two bits are multiplied and the result is -1.

3.2.2. BAUGH-WOOLEY MULTIPLICATION IN DSP

The Baugh-Wooley multiplier architecture is optimized for speed and efficiency, making it suitable for high-performance digital systems. It has a regular structure that allows for easy implementation using hardware description languages (HDLs) and programmable logic devices (PLDs).

The advantages offered by Baugh-wooley Multilpier when compared to other multipliers are:

- ✓ *Reduced number of partial products:* The Baugh-Wooley multiplier generates only (n/2) + 1 partial products for n-bit operands, which is less than other multipliers like the Wallace multiplier that generates 3n/2 partial products. This reduces the number of logic gates required and makes the circuit faster and more efficient.
- ✓ *Balanced delay:* The delay through the Baugh-Wooley multiplier is balanced, which means that the propagation delay is the same for all bits of the result. This makes it easier to optimize the clock frequency and timing of the circuit.
- ✓ *Regular structure:* The Baugh-Wooley multiplier has a regular and structured design that makes it easy to implement in hardware using HDLs or PLDs. It also allows for easy scaling to larger operand sizes.

- ✓ *Low power consumption:* Due to the reduced number of partial products and balanced delay, the Baugh-Wooley multiplier consumes less power compared to other multipliers.

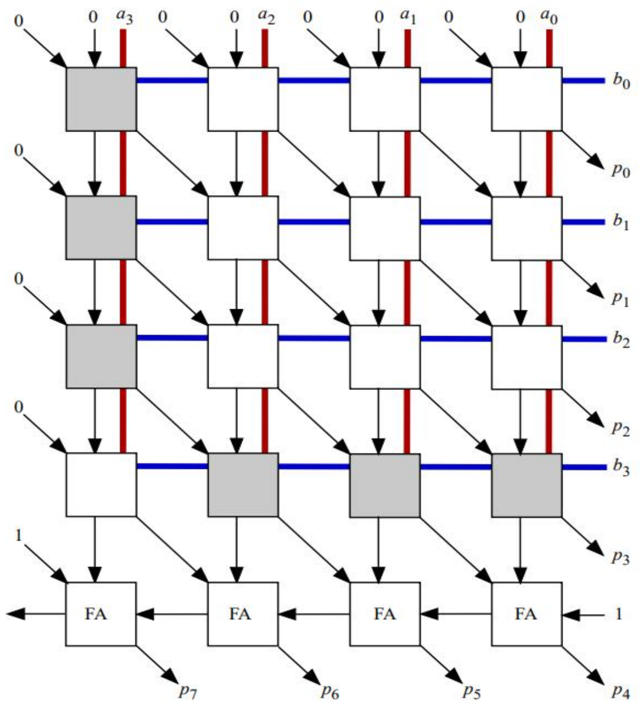


Figure 3.2.2. Block Diagram of 4x4 Baugh-Wooley Multiplier

The baugh-wooley multiplication algorithm is an efficient way to handle the sign bits. This technique has been developed in order to design regular multipliers suited for 2`s complement numbers. Let us consider two n-bit numbers, A and B to be multiplied, A and B can be represented as

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

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Where the a_i's and b_i's are the bits in A and B, respectively and a_{n-1} and b_{n-1} are the sign bits.

The products, P= A x B is then given by the following equation:

$$P = A \times B$$

$$= \left(-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i\right) \times \left(-b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j\right)$$

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

The final product P = A x B becomes:

$$P = a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} + 2^{n-1}\sum_{i=0}^{n-2} \overline{b_{n-1}a_i} 2^i + 2^{n-1}\sum_{j=0}^{n-2} \overline{a_{n-1}b_j} 2^j - 2^{2n-1} + 2^n$$

Let us assume that A and B are 4-bit binary numbers, then the product P = A x B is 8-bits long and is given by,

$$P = a_3 b_3 2^6 + \sum_{i=0}^{2}\sum_{j=0}^{2} a_i b_j 2^{i+j} + 2^3\sum_{i=0}^{2} \overline{b_3 a_i} 2^i + 2^3\sum_{j=0}^{2} \overline{a_3 b_j} 2^j - 2^7 + 2^4$$

The following figure shows the implementation of Baugh-Wooley multiplier using 16 process element blocks,

3.3. DESIGN OF FULL ADDER USING VARIOUS TECHNIQUES

The following section consists of different adder architectures in use,

A. Genetic Full Adder:

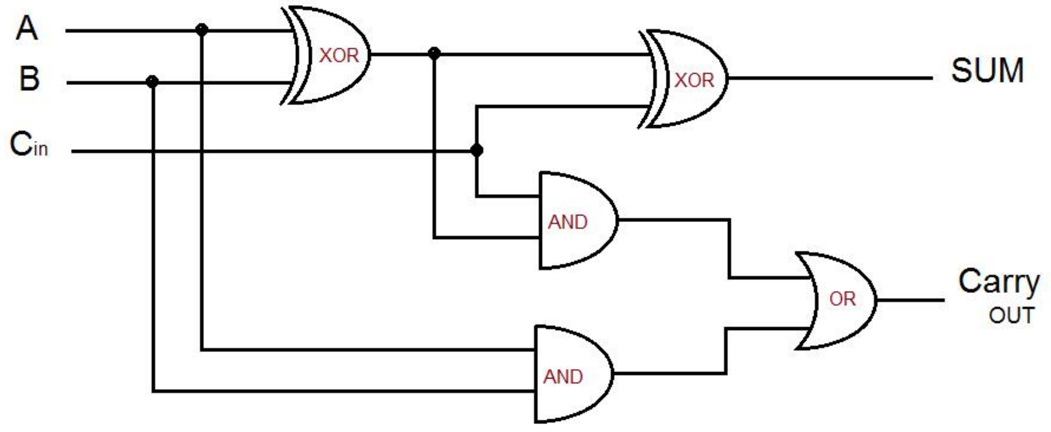


Figure 3.3.1 Architecture of Generic Full Adder

A generic full adder is a digital circuit that performs the addition of two binary numbers, considering a carry input from a previous stage. It consists of two main components: the sum output and the carry output. The architecture of a generic full adder can be explained using logic gates. Let's denote the two input bits to be added as A and B, and the carry input from the previous stage as Cin. The outputs are the sum, denoted as S, and the carry out, denoted as Cout.

The generic full adder architecture typically employs two stages: the sum stage and the carry stage.

1. Sum Stage:

The sum stage is responsible for calculating the sum output bit (S) of the full adder. It can be implemented using an XOR gate. The XOR gate takes the inputs A, B, and Cin and produces the sum output S.

$$S = A \oplus B \oplus Cin \text{ (} \oplus \text{ denotes XOR operation)}$$

2. Carry Stage:

The carry stage calculates the carry out bit (Cout) of the full adder. It involves two main components: the AND gate and the OR gate.

$$Cout = (A \text{ AND } B) \text{ OR } (Cin \text{ AND } (A \text{ OR } B))$$

The OR gate considers the case when either A and B have a carry or when there is a carry from the previous stage.

B. Full adder using two half adders:

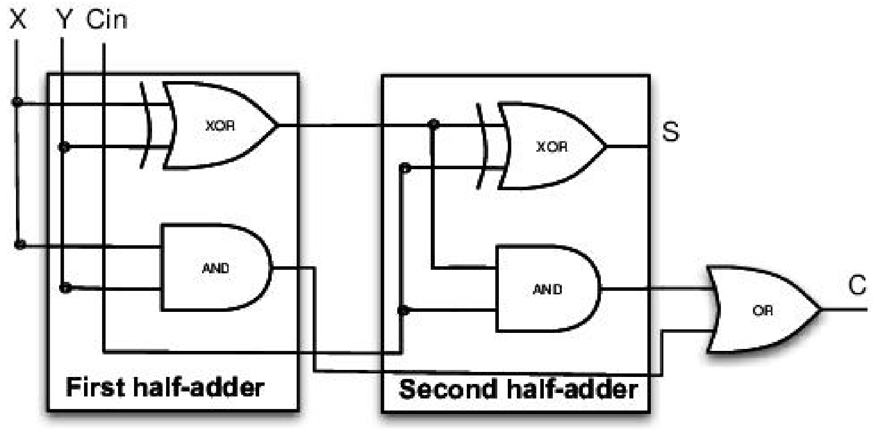


Figure 3.3.2. Architecture of Full Adder using two Half Adders

A half adder is a basic digital circuit that can add two binary bits, producing a sum and a carry output. To design a full adder using half adders, we can combine two half adders along with an additional OR gate to handle the carry input from the previous stage.

Let's denote the two input bits to be added as A and B, the sum output as S, and the carry output as Cout.

The architecture of a full adder based on half adders consists of two stages: the first stage computes the sum of A and B, and the second stage computes the carry output.

1. Sum Stage:
   In the sum stage, we use a half adder to calculate the sum output (S) of the full adder. The half adder takes inputs A and B and produces the sum output, which we denote as S1, and the carry output, which we denote as C1.

$$S1 = A \oplus B$$
$$C1 = A \text{ AND } B$$

2. Carry Stage:
   In the carry stage, we combine two half adders and an OR gate to calculate the final carry output (Cout) of the full adder. We take the carry output (C1) from the sum stage as an input, along with an additional carry input from the previous stage (Cin). The output Cout is the result of the OR gate.

$$S2 = S1 \oplus Cin$$
$$C2 = C1 + (S1 \text{ AND } Cin)$$
$$Cout = C2$$

The intermediate sum output (S2) is obtained by XORing the previous sum output (S1) with the carry input (Cin) from the previous stage.

The intermediate carry output (C2) is calculated by adding the carry output (C1) from the sum stage and the result of the AND gate between the previous sum output (S1) and the carry input (Cin).

The final carry output (Cout) is obtained by directly connecting the output of the OR gate to the Cout output of the full adder.

By combining the sum and carry stages, we obtain a full adder based on half adders. This design allows for cascading multiple full adders to create larger arithmetic circuits for binary addition, such as adders and multipliers.

C. MUX based Full Adder Design:

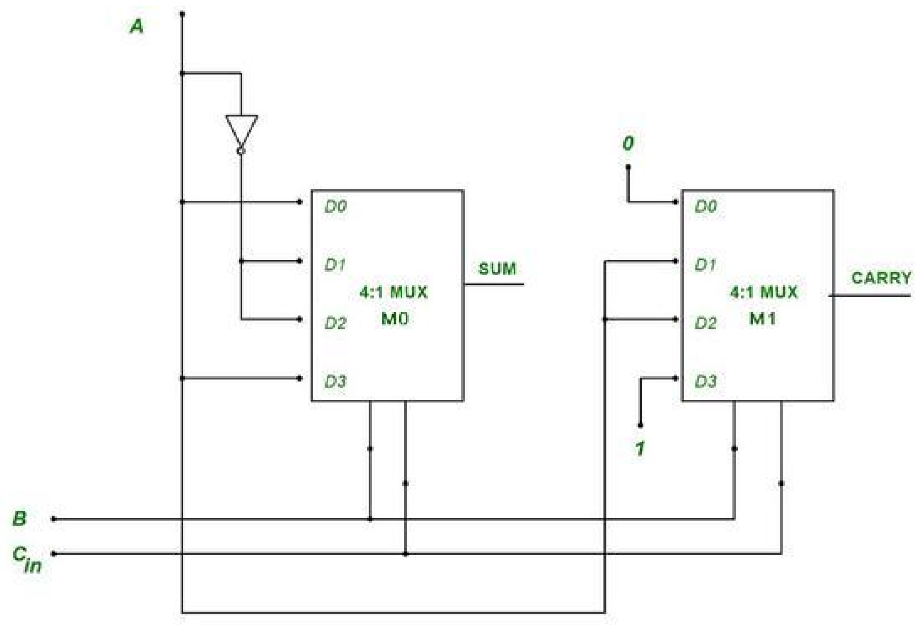


Figure 3.3.3. Architecture of Full Adder using MUX

A mux (multiplexer)-based full adder design is an alternative implementation of a full adder using multiplexers instead of logic gates. This design offers a different perspective on how to construct a full adder circuit.

A multiplexer is a digital circuit component that selects one of multiple input signals and routes it to the output based on a control signal. The control signal determines which input is chosen. In the context of a mux-based full adder, we use multiplexers to select and combine the appropriate input signals to generate the sum and carry outputs.

Let's denote the two input bits to be added as A and B, and the carry input from the previous stage as Cin. The outputs are the sum, denoted as S, and the carry out, denoted as Cout.

The mux-based full adder can be divided into two stages: the sum stage and the carry stage.

1. Sum Stage:

In the sum stage, we use two 2:1 multiplexers. One multiplexer selects the input A or B based on the control signal, and the other multiplexer selects the carry input Cin or the output of the first multiplexer, depending on the control signal.

The control signal for the first multiplexer is the input B, and the control signal for the second multiplexer is the input A. This arrangement ensures that when B is 0, the first multiplexer selects A, and when B is 1, it selects Cin. The second multiplexer selects Cin when A is 1; otherwise, it selects the output of the first multiplexer.

2. Carry Stage:

In the carry stage, we use three 2:1 multiplexers. These multiplexers are used to determine the carry output Cout. The control signal for the first multiplexer is A, the control signal for the second multiplexer is B, and the control signal for the third multiplexer is Cin. The inputs for the first multiplexer are A AND B and A XOR B. The multiplexer selects A AND B when A is 0 and A XOR B when A is 1. The inputs for the second multiplexer are A, B, and Cin. The multiplexer selects A when B is 0, B when B is 1, and Cin when B is 1 and A is 0.

The inputs for the third multiplexer are the outputs of the first two multiplexers. The multiplexer selects the output of the first multiplexer when Cin is 0 and the output of the second multiplexer when Cin is 1.

The output of the third multiplexer represents the carry output Cout.

D. Propagate and Generate Based Full Adder Design:

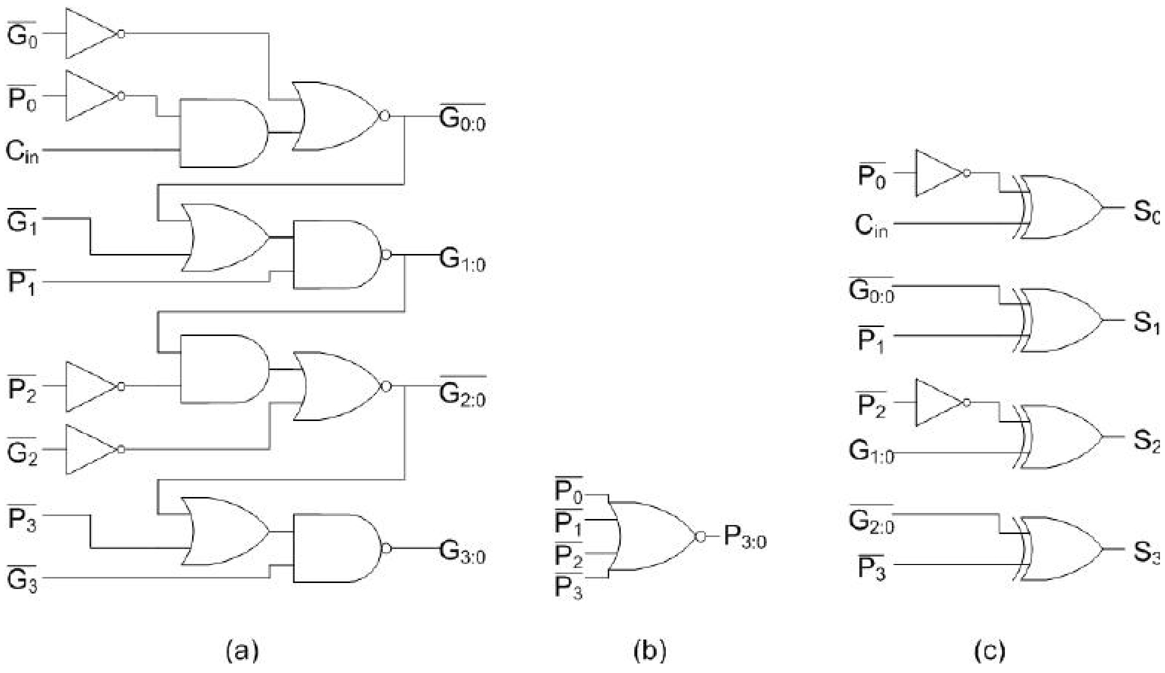


Figure 3.3.4. Architecture of Full Adder using PG

Propagate and Generate (P and G) based full adder design is an alternative approach to implementing a full adder circuit that focuses on optimizing the propagation delay. It reduces the number of gates compared to the traditional full adder architecture, resulting in improved performance.

The P and G based full adder design involves three main components: the P bits, the G bits, and the sum and carry outputs.

1. P (Propagate) bits:

The P bits represent the bits that propagate the carry signal. These bits are generated by evaluating the AND gate logic on the input bits A and B.

P = A AND B

The P bits indicate whether the corresponding input bits A and B will propagate a carry to the next stage.

2. G (Generate) bits:

The G bits represent the bits that generate a carry. These bits are generated by evaluating the AND gate logic on the input bits A and B, considering the carry input Cin.

G = A OR B

The G bits indicate whether the corresponding input bits A and B, along with the carry input Cin, will generate a carry.

3. Sum and Carry outputs:

The sum output (S) can be calculated by performing an XOR operation between the input bits A, B, and Cin, just like in the traditional full adder.

$$S = A \oplus B \oplus Cin$$

The carry output (Cout) is determined by evaluating the OR gate logic on the P and G bits. It considers the cases where either a carry is generated by the G bits or a carry is propagated by the P bits.

$$Cout = (G \text{ AND } Cin) \text{ OR } P$$

The carry output Cout is generated by the OR gate using the P and G bits and the carry input Cin.

By using the P and G based full adder design, the number of gates and the resulting propagation delay are reduced compared to the traditional full adder architecture.

E. Threshold Logic Based Design:

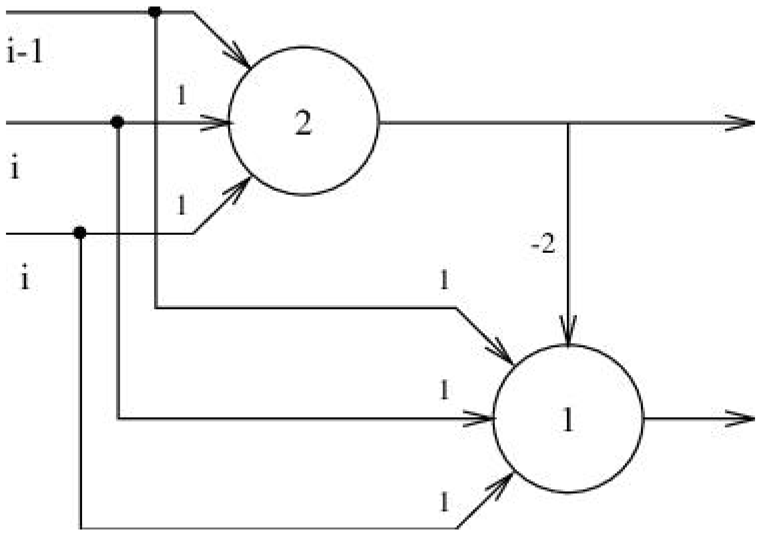


Figure 3.3.5. Architecture of Full Adder Using Threshold Logic

Threshold logic is a computational model that operates based on threshold functions. In threshold logic, the inputs and weights are summed up, and if the sum exceeds a certain threshold, the output is activated, otherwise, it remains deactivated. Let's explore how threshold logic can be used to design a full adder.

To design a threshold logic-based full adder, we can use threshold gates as the fundamental building blocks. A threshold gate takes multiple inputs, multiplies them by corresponding weights, sums up the weighted inputs, and compares the sum to a threshold value. If the sum exceeds the threshold, the gate outputs a high signal (1); otherwise, it outputs a low signal (0).

Chapter 4

METHODOLOGY

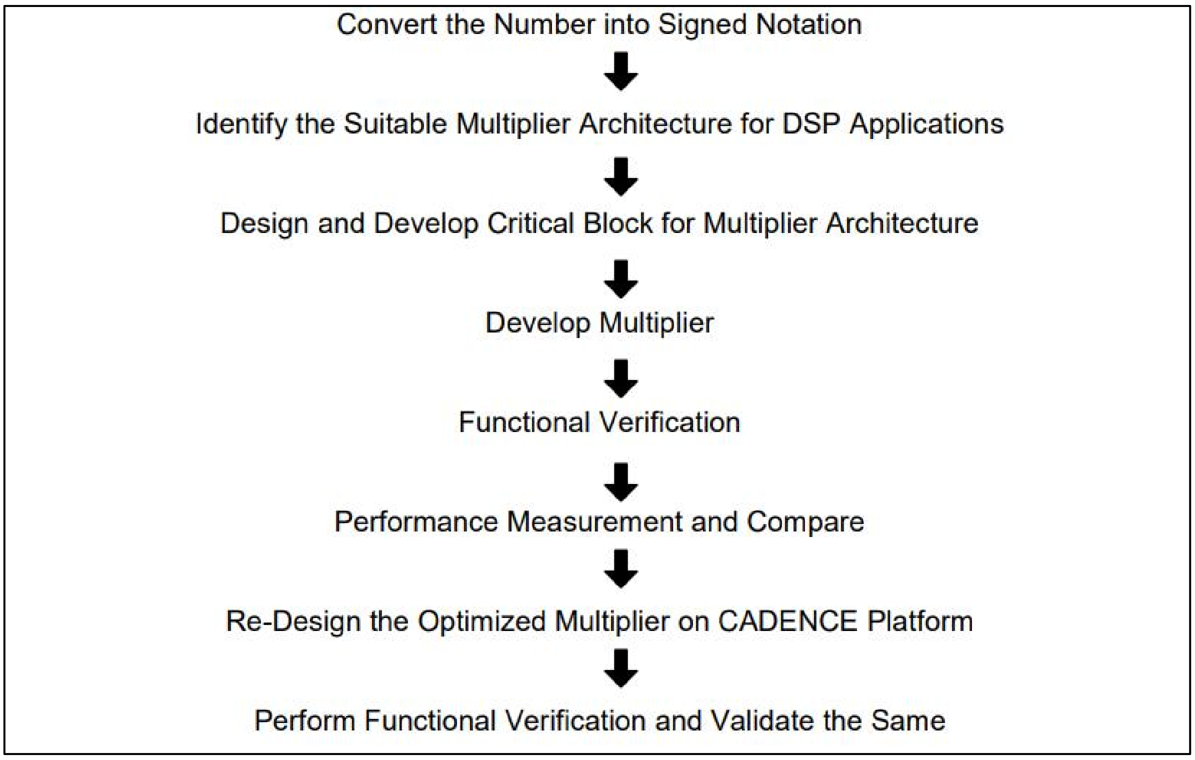

Convert the Number into Signed Notation
Identify the Suitable Multiplier Architecture for DSP Applications
Design and Develop Critical Block for Multiplier Architecture
Develop Multiplier
Functional Verification
Performance Measurement and Compare
Re-Design the Optimized Multiplier on CADENCE Platform
Perform Functional Verification and Validate the Same

Figure 4.1. Steps Involved in the Project

- ✓ **Conversion of numbers into signed notations:** Sign-magnitude notation is the simplest and one of the most common methods of representing positive and negative numbers either side of zero, (0). Thus negative numbers are obtained simply by changing the sign of the corresponding positive number as each positive or unsigned number will have a signed opposite, for example, +2 and -2, +10 and -10, etc. For signed binary numbers the most significant bit (MSB) is used as the sign bit. If the sign bit is "0", this means the number is positive in value. If the sign bit is "1", then the number is negative in value. The remaining bits in the number are used to represent the magnitude of the binary number in the usual unsigned binary number format way.

- ✓ **Identify the Suitable Multiplier Architecture for DSP Applications:** There have been many algorithms proposals in literature to perform multiplication, each offering different advantages and having trade-off in terms of speed, circuit complexity, and area and power consumption. The multiplier is a fairly large block of a computing system. Modified booth multiplier, array multiplier, binary multiplier are the classes of multipliers.

- ✓ **Design and Develop Critical Block for Matrix:** Two most important design criteria deciding the performance of processor are speed and power consumption. Many research efforts have been carried out in order to obtain energy efficient multiplier architectures. State-of-the-art designs focused mainly on reducing the silicon area but in the last decade the focus is primarily shifted toward speed and power. The complexity of the design directly depends on the speed of computation. High speed requirement results in increased complexity of the circuit, hence larger number of transistors will be required in the design which further results in high power dissipation. So there is a trade-off between speed and power dissipation.

- ✓ **Develop Multiplier:** Keeping the design constraints in mind a optimized multiplier must be developed.

- ✓ **Functional Verification:** For the multiplier design so developed, functional verification is done using the Xilinx ISE Design Suit.

- ✓ **Performance Measurement and Compare:** In many DSP algorithms, the multiplier lies in the critical delay path and ultimately determines the performance of algorithm. The speed of multiplication operation is of great importance in DSP as well as in general processor. In the past multiplication was implemented generally with a sequence of addition. Throughput is the measure of how many multiplications can be performed in a given period of time; multiplier is not only a high delay block but also a major source of power dissipation. That is why if one also aims to minimize power consumption, it is of great interest to reduce the delay by using various delay optimization.

- ➢ **Re-Design the Optimized Multiplier on CADENCE Platform:** In algorithmic and structural levels, a lot of multiplication techniques had been developed to enhance the efficiency of the multiplier; which encounters the reduction of the partial products and/or the methods for their partial products addition, but the principle behind multiplication was same in all cases.

- ✓ **Perform Functional Verification and Validate the Same:** Finally the plotted design is then validated on Cadence platform.

Chapter 5

RESULTS

This chapter contains experimental set up, simulation results, comparative study and implementation details of both Xilinx ISE design software and cadence virtuoso

5.1. DESIGN AND IMPLEMENTATION ON XILINX ISE DESIGN SUITE

This section contains experimental set up, simulation results, comparative study and implementation details of both Xilinx ISE design software.

5.1.1. EXPERIMENTAL SETUP

The hardware specification for the system is as follows: it belongs to the Virtex 7 family of devices and specifically utilizes the XC7A100T device. The package used is FGG676, and it operates at a speed grade of -3. The specific hardware configuration is identified as xc7a100t-3fgg676. The design environment employed for this system is the ISE Design Suite 14.7, which provides the necessary tools and resources for designing and programming the hardware.
Here the parameters under consideration are LUT's and Slices are area parameters and combinational path delay in the delay parameter which is further divided into logic delay and route delay.

5.1.2. SIMULATION RESULTS

Finally the most efficient design that is the Baugh-Wooley multiplier with propagate and generate signaling for a full adders was designed at transistor level using the Cadence virtuoso tool and the functional verification was done.

The output waveforms of a generic multiplier are an important consideration when designing new circuits that rely on multiplication. The waveforms provide information about the performance and behavior of the multiplier under different input conditions, which can help in optimizing its design.

The waveform of the output signal is influenced by several factors, including the amplitude and phase of the input signals, the type of multiplier used, and the design parameters of the multiplier. In the case of a generic multiplier, the output waveforms shown in the results can provide useful insights into its basic operating characteristics.

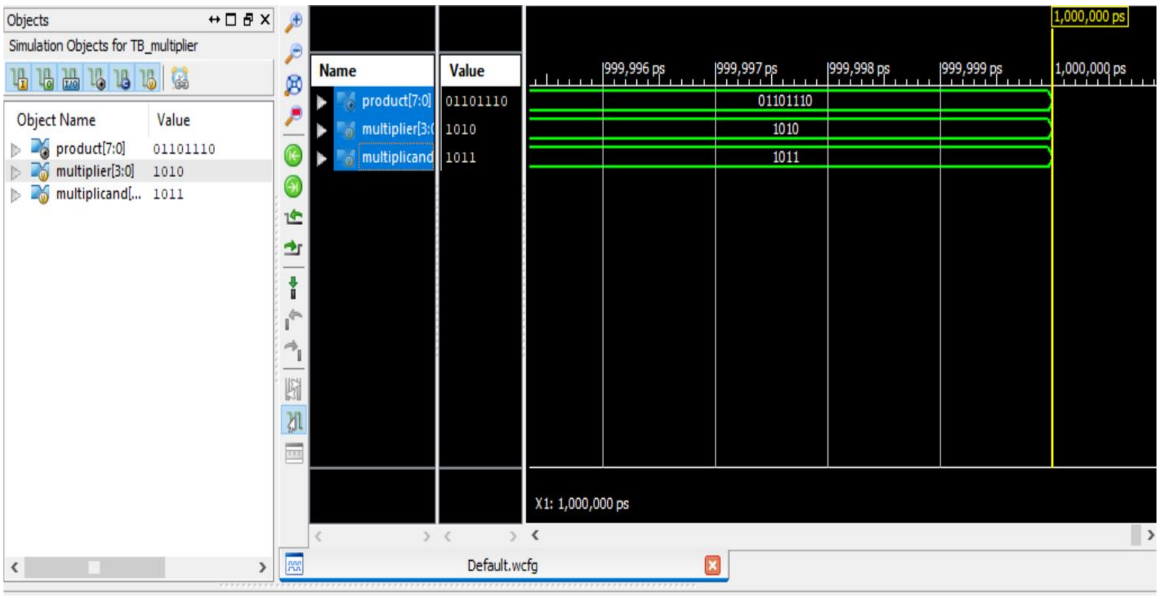


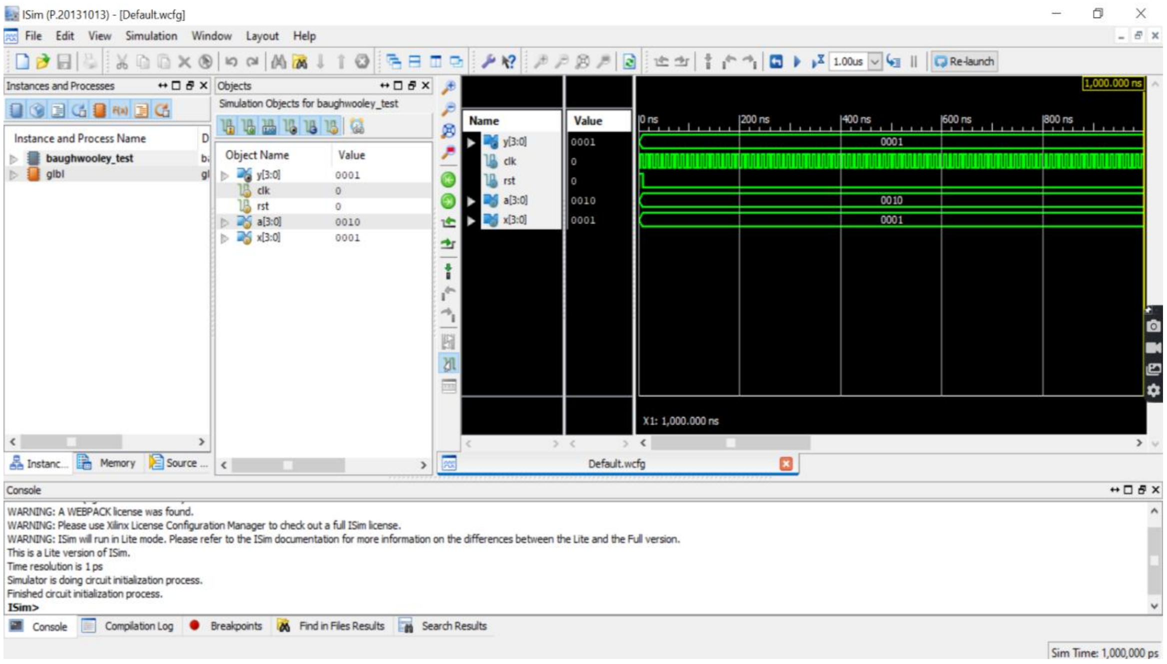
Figure 5.1.2.1. Output Waveform of Generic Multiplier

One important consideration when analyzing the output waveforms of a multiplier is the level of distortion or noise present in the signal. Distortion can be caused by a number of factors, including non-linearities in the multiplier circuitry or interference from other sources. The waveform results can be used to identify any sources of distortion or noise and to evaluate the effectiveness of any mitigation strategies that have been implemented.

Another important consideration is the frequency response of the multiplier. The waveform results can be used to analyze the frequency response of the circuit and to identify any frequency-dependent behavior or limitations. This information can be used to design filters or other signal conditioning circuits that can help to mitigate any frequency-dependent effects.

The output waveforms can also provide information about the accuracy and precision of the multiplier. For example, the results can be used to evaluate the linearity of the circuit and to identify any sources of error or bias. This information can be used to optimize the design of the multiplier and to improve its overall performance.

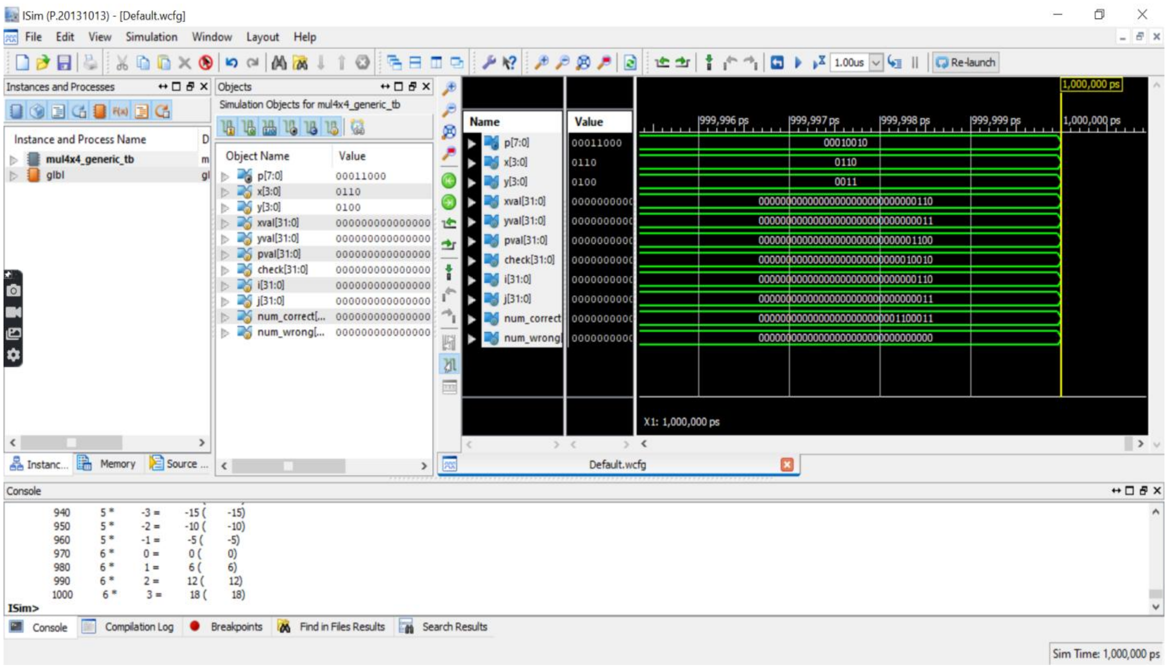
The output waveforms of a generic multiplier are an important tool for evaluating and optimizing the performance of the circuit. By carefully analyzing the waveform results, designers can identify sources of distortion and noise, evaluate the frequency response of the circuit, and optimize the accuracy and precision of the multiplier.



```
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Figure 5.1.2.2. Simulation Result of Generic 4x4 Baugh-Wooley Multiplier

The Baugh-Wooley multiplier is a well-known technique for performing binary multiplication using only bitwise operations as discussed in the above discussion of design methodology. In a 4x4 Baugh-Wooley multiplier, the input operands are 4 bits each, and the output is an 8-bit product. The propagate and generate method is a common approach for implementing the Baugh-Wooley multiplier.


ISim (P.20131013) - [Default.wcfg]

Simulation Objects for mul4x4_generic_tb

Object Name	Value
p[7:0]	00011000
x[3:0]	0110
y[3:0]	0100
xval[31:0]	000000000000000
yval[31:0]	000000000000000
pval[31:0]	000000000000000
check[31:0]	000000000000000
i[31:0]	000000000000000
j[31:0]	000000000000000
num_correct[...	000000000000000
num_wrong[...	000000000000000

Name	Value	Waveform
p[7:0]	00011000	00010010
x[3:0]	0110	0110
y[3:0]	0100	0011
xval[31:0]	000000000	00000000000000000000000000000110
yval[31:0]	000000000	00000000000000000000000000000011
pval[31:0]	000000000	00000000000000000000000000001100
check[31:0]	000000000	00000000000000000000000000010010
i[31:0]	000000000	00000000000000000000000000000110
j[31:0]	000000000	00000000000000000000000000000011
num_correct	000000000	00000000000000000000000001100011
num_wrong	000000000	00000000000000000000000000000000

X1: 1,000,000 ps

```
      940       5 *     -3 =     -15 (     -15)
      950       5 *     -2 =     -10 (     -10)
      960       5 *     -1 =      -5 (      -5)
      970       6 *      0 =       0 (       0)
      980       6 *      1 =       6 (       6)
      990       6 *      2 =      12 (      12)
     1000       6 *      3 =      18 (      18)
ISim>
```

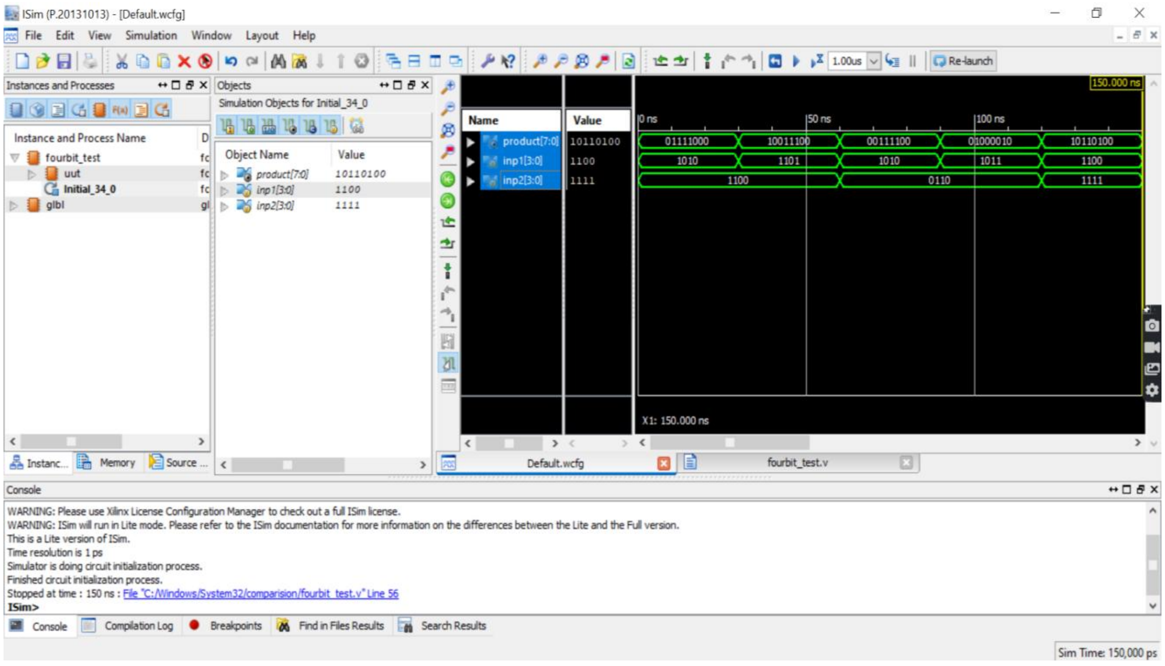
Sim Time: 1,000,000 ps

Figure 5.1.2.3. Simulation Result of 4x4 Baugh-Wooley Multiplier using propagate and generate method

The simulation results of the 4x4 Baugh-Wooley multiplier using the propagate and generate method indicate that it has the lowest number of LUTs (look-up tables) and the least path delay compared to other multiplication algorithms which is tabulated in Table 4 above. The number of LUTs is a measure of the amount of logic needed to implement the multiplier, and a lower number is generally better for performance and resource utilization. The path delay is a measure of the time it takes for a signal to propagate through the circuit, and a shorter delay means faster operation.

The reason for the superior performance of the Baugh-Wooley multiplier using the propagate and generate method is due to the nature of the algorithm itself. The technique is based on a clever use of logic gates to minimize the amount of computation required for each multiplication operation. By using only bitwise operations, the circuit can be implemented with fewer LUTs and a shorter path delay compared to other multiplication algorithms that require more complex operations.

Overall, the simulation results demonstrate that the 4x4 Baugh-Wooley multiplier using the propagate and generate method is an efficient and effective way to perform binary multiplication. The technique provides a good balance between performance and resource utilization, making it well-suited for a variety of applications in digital signal processing and other fields.

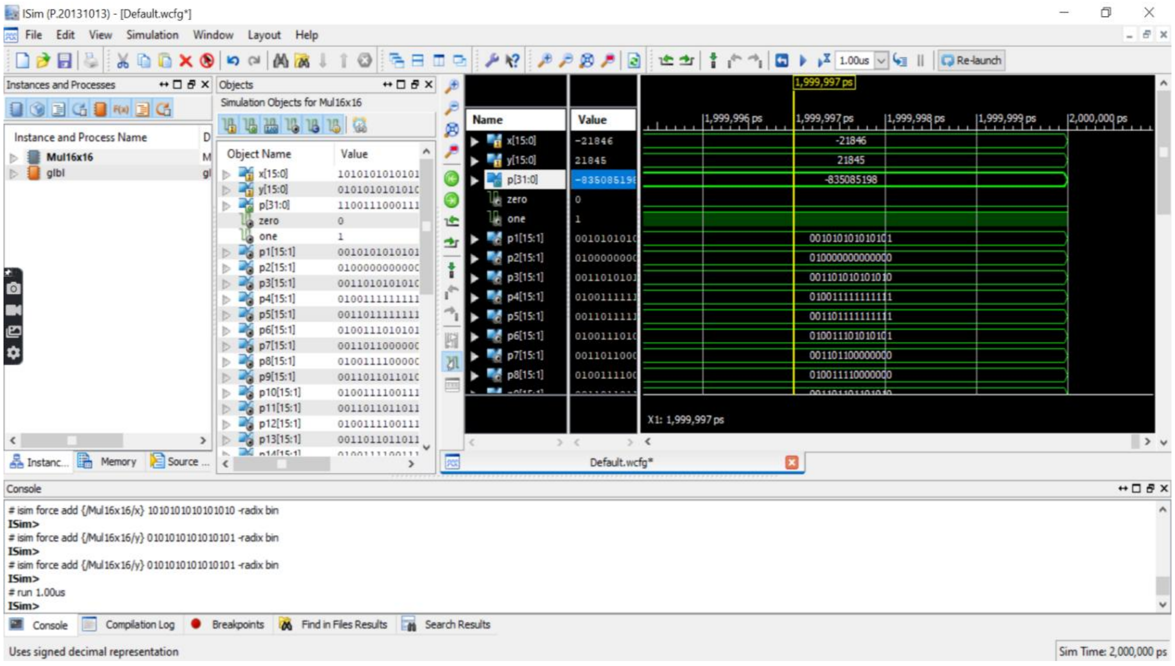


Object Name	Value
product[7:0]	10110100
inp1[3:0]	1100
inp2[3:0]	1111

```
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 150 ns : File "C:/Windows/System32/comparision/fourbit_test.v" Line 56
ISim>
```

Figure 5.1.2.4. Simulation Result of 4x4 Baugh-Wooley Multiplier using half adder based design

Although the above implementation is effective and gives desired results, it does not stand a chance when compared to our proposed method of 4x4 Baugh-Wooley Multiplier using propogate and generate.

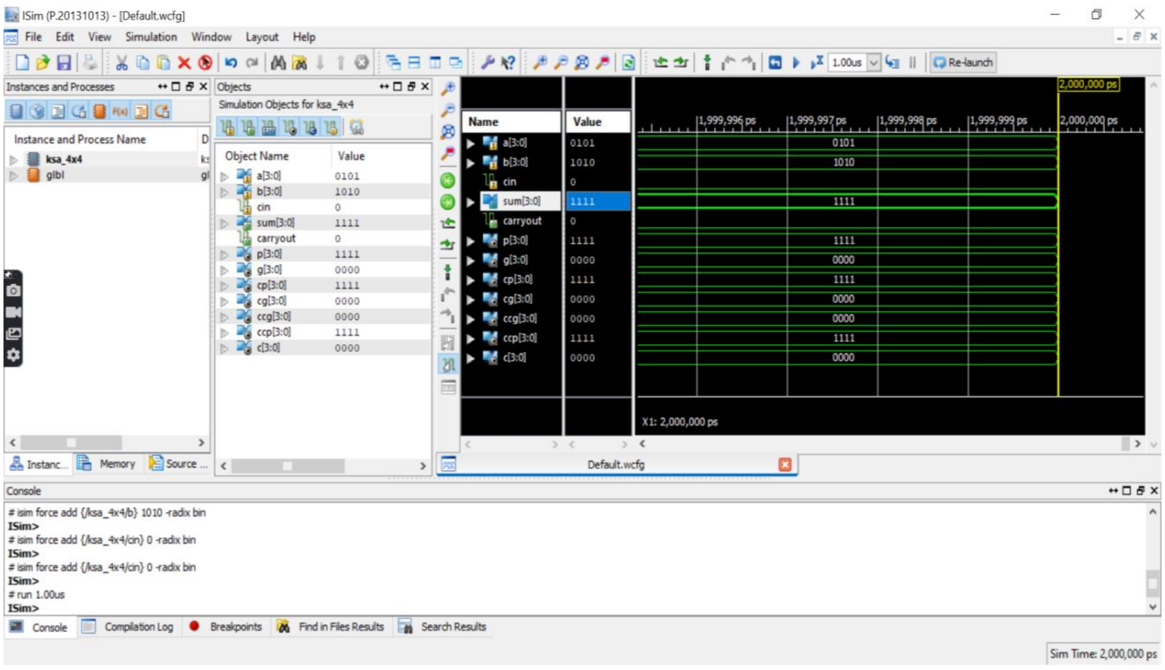


Object Name	Value
x[15:0]	1010101010101
y[15:0]	0101010101010
p[31:0]	1100111000111
zero	0
one	1
p1[15:1]	0010101010101
p2[15:1]	0100000000000
p3[15:1]	0011010101010
p4[15:1]	0100111111111
p5[15:1]	0011011111111
p6[15:1]	0100111010101
p7[15:1]	0011011000000
p8[15:1]	0100111100000
p9[15:1]	0011011011010
p10[15:1]	0100111100111
p11[15:1]	0011011011011
p12[15:1]	0100111100111
p13[15:1]	0011011011011

```
# isim force add {/Mul16x16/x} 1010101010101010 -radix bin
ISim>
# isim force add {/Mul16x16/y} 0101010101010101 -radix bin
ISim>
# isim force add {/Mul16x16/y} 0101010101010101 -radix bin
ISim>
# run 1.00us
ISim>
```

ISim (P.20131013) - [Default.wcfg*]
Mul16x16
glbl
Simulation Objects for Mul16x16
-21846
21845
-835085198
X1: 1,999,997 ps
Uses signed decimal representation
Sim Time: 2,000,000 ps

Figure 5.1.2.5. Simulation Result of 16x16 Baugh-Wooley Multiplier using conventional adders



Name	Value
a[3:0]	0101
b[3:0]	1010
cin	0
sum[3:0]	1111
carryout	0
p[3:0]	1111
g[3:0]	0000
cp[3:0]	1111
cg[3:0]	0000
ccg[3:0]	0000
ccp[3:0]	1111
c[3:0]	0000

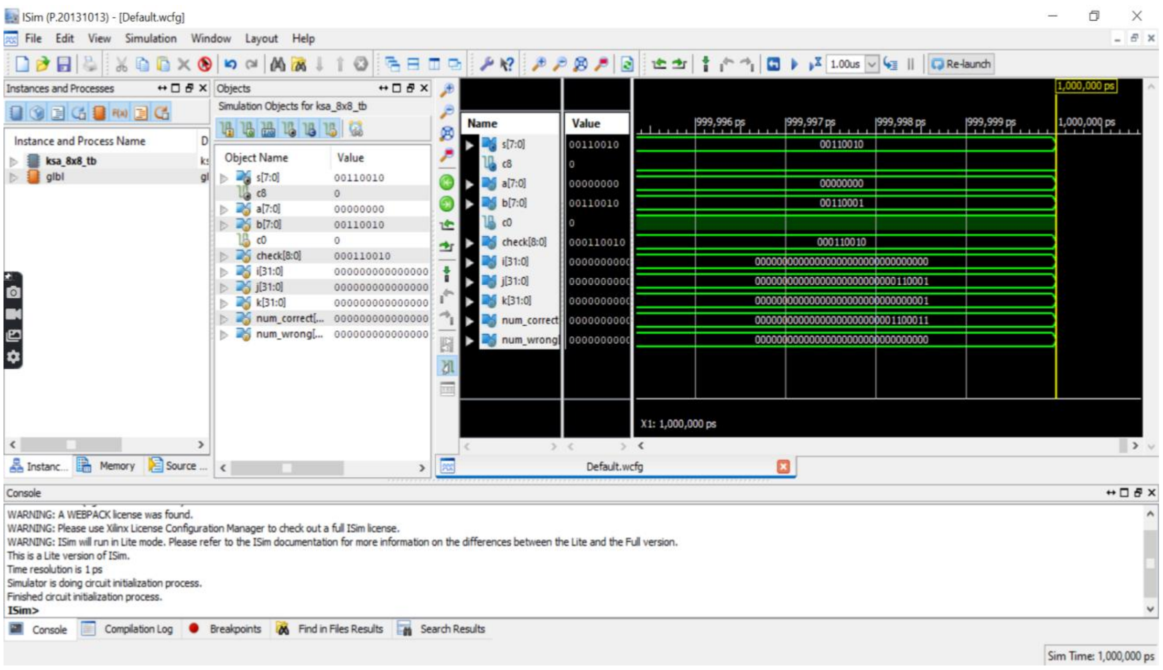
```
# isim force add {/ksa_4x4/b} 1010 -radix bin
ISim>
# isim force add {/ksa_4x4/cin} 0 -radix bin
ISim>
# isim force add {/ksa_4x4/cin} 0 -radix bin
ISim>
# run 1.00us
ISim>
```

Figure 5.1.2.6. Simulation Result of 4x4 Baugh-Wooley Multiplier using KSA

The Kogge-Stone adder is a type of parallel prefix adder that can perform binary addition quickly and efficiently. However, there are several disadvantages to this type of adder, including:

The Kogge-Stone adder requires a large number of logic gates and wires to implement, making it more complex than simpler adders such as ripple carry adders. This complexity can lead to higher power consumption and longer design times. The Kogge-Stone adder requires a larger area than some other types of adders, which can be a concern for applications with limited space or where minimizing chip size is important. While the Kogge-Stone adder is generally faster than ripple carry adders for large operands, it can still suffer from longer delays than other types of parallel prefix adders such as the Brent-Kung adder.

The Kogge-Stone adder does not include any error correction or detection mechanisms, which means that it may not be suitable for applications where error detection or correction is critical. The Kogge-Stone adder is a powerful tool for performing binary addition quickly and efficiently, but its complexity and larger area requirements may limit its applicability in certain contexts. Additionally, its lack of error correction or detection mechanisms may make it unsuitable for applications where data integrity is a concern.



Object Name	Value
s[7:0]	00110010
c8	0
a[7:0]	00000000
b[7:0]	00110010
c0	0
check[8:0]	000110010
i[31:0]	000000000000000
j[31:0]	000000000000000
k[31:0]	000000000000000
num_correct[...	000000000000000
num_wrong[...	000000000000000

```
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Figure 5.1.2.7. Simulation Result of 8x8 Baugh-Wooley Multiplier sung BKA

The Brent-Kung adder is a type of parallel prefix adder that is known for its efficiency and fast performance in binary addition. However, there are several disadvantages to this type of adder after observing above simulation and its results, including:

The Brent-Kung adder is more complex than simpler adders such as ripple carry adders, which can make it more difficult and time-consuming to design and implement. This complexity can also increase power consumption. The Brent-Kung adder requires a larger area than some other types of adders, which can be a concern for applications with limited space or where minimizing chip size is important. Cost: The Brent-Kung adder may be more expensive to implement than some other types of adders due to its complex design and area requirements.

The Brent-Kung adder may not be suitable for all input sizes. It is typically designed for larger inputs and may not be the best choice for smaller inputs. The Brent-Kung adder may still suffer from carry propagation delays in some cases, which can limit its performance in certain applications. The Brent-Kung adder is a powerful tool for performing binary addition quickly and efficiently, but its complexity, area requirements, and limitations on input sizes may limit its applicability in certain contexts.

5.1.3. COMPARATIVE STUDY

We started the comparison by comparing the different architectural blocks of a full adder as they perform as the basic building blocks of a multiplication process. The basic one of it included the generic addition method that involved generic full address to accomplish the addition process.

	Generic Method	MUX based approach	Propagate and Generate	Half adder based approach
Slice LUT's	2	2	2	2
Occupied Slices	2	2	2	2
Combinational Path delay	3.813ns	3.813ns	3.813ns	3.813ns
Macros	1 bit 3 input xor -2	1 bit 2 input xor-2	1 bit 2 input xor-2	1 bit 2 input xor-2

Table 5.1.3.1. Full Adder Design Analysis Tabulation

The second one of it was the MUX based design, the third one was using the propagate and generate pulses and the final one was the half adder base approach.

As the combination path delay here was platform specific we couldn't draw much differences in the combinational path delay of the existing for algorithms but the design comparison of the full adder blocks is shown above

As far as a literature review of the current design addition blocks and the multiplier architecture in general is concerned we found that Baugh-Wooley multiplication algorithm gave the most significant results as far as signed multiplication for digital signal processing was concerned. Hence removed ahead with the same design replacing the current design blocks for a full adders by the tested design blocks of the full adders. In the first case we took a 4 × 4 Baugh -wooley multiplier and we tried to analyse the different dimensions of path delay combinations for 5 different adder approaches. Generic addition, MUX based approach, propagate and generate based approach, half adder based approach, threshold logic based approach where the five different logics.

	Generic Addition	MUX based Addition	Propagate and Generate	Half adder based approach	Threshold Logic
Slice LUT's	16	13	14	14	114
Occupied Slices	9	5	5	5	37
Combinational Path delay	3.070ns (0.389ns logic, 2.681ns route) (12.7% logic, 87.3% route)	4.373ns (0.583ns logic, 3.790ns route) (13.3% logic, 86.7% route)	2.520ns (0.292ns logic, 2.228ns route) (11.6% logic, 88.4% route)	3.720ns (0.352ns logic, 3.368ns route) (12.2% logic, 87.8% route)	7.337ns (3.467ns logic, 3.870ns route) (47.3% logic, 52.7% route)
Macros	1-bit 2 input xor -27	1-bit 2-to-1 multiplexer -30 1-bit 2 input xor -15	1-bit 2-to-1 multiplexer-15 1-bit 2 input xor -30	1-bit 2 input xor -30	32-bit comparator lessequal-30 32-bit 2 input xor-30

Table 5.1.3.2. Computational Results of 4x4 Baugh-Wooley Multiplier

Out of the existing design blocks it was observed that propagate and generate method for full adder gives most efficient combinational path delay that is 2.520 nanoseconds. The results of the simulation are tabulated as above.

	Generic Addition	MUX based Addition	Propagate and Generate	Half adder based approach	Threshold Logic
Slice LUT's	78	77	78	78	468
Occupied Slices	27	24	34	42	151
Combinational Path delay	8.170ns (1.068ns logic, 7.102ns route) (13.1% logic, 86.9% route)	8.914ns (1.165ns logic, 7.749ns route) (13.1% logic, 86.9% route)	7.270ns (1.068ns logic, 6.202ns route) (14.3% logic, 85.7% route)	8.170ns (1.068ns logic, 7.102ns route) (13.1% logic, 86.9% route)	14.902ns (7.127ns logic, 7.776ns route) (47.8% logic, 52.2% route)
Macros	1-bit 2 input xor -118	1-bit 2-to-1 multiplexer-118 1-bit 2 input xor -59	1-bit 2-to-1 multiplexer-55 1-bit 2 input xor -118	1-bit 2 input xor -118	32-bit comparator lessequal-118 32-bit 2 input xor-118

Table 5.1.3.3. Computational Results of 8x8 Baugh-Wooley Multiplier

The same method was the extended to 8 × 8 Baugh-Wooley multiplier and the same results where drawn with propagate and generate design for a full adder giving the minimum combinational path delay being 7.270 nanoseconds. The same results are tabulated below,

	Generic Addition	MUX based Addition	Propagate and Generate	Half adder based approach	Threshold Logic
Slice LUT's	372	336	341	341	1970
Occupied Slices	201	185	184	182	644
Combinational Path delay	17.240ns (2.232ns logic, 15.008ns route) (12.9% logic, 87.1% route)	20.238ns (2.620ns logic, 17.618ns route) (12.9% logic, 87.1% route)	17.240ns (2.232ns logic, 15.008ns route) (12.9% logic, 87.1% route)	17.240ns (2.232ns logic, 15.008ns route) (12.9% logic, 87.1% route)	30.924ns (14.447ns logic, 16.477ns route) (46.7% logic, 53.3% route)
Macros	1-bit 2-to-1 multiplexer -486	1-bit 2-to-1 multiplexer -486 1-bit 2 input xor - 243	1-bit 2-to-1 multiplexer - 246 1-bit 2 input xor - 486	1-bit 2-to-1 multiplexer-486	32-bit comparator lessequal-486 32-bit 2 input xor-486

Table 5.1.3.4. Computational Results of 16x16 Baugh-Wooley Multiplier

Similarly for 16 × 16 Baugh-Wooley multiplier give minimum path delay for the same propagating generate design for a full letter which was 17.240 nanoseconds. The tabulation containing slice LUT's, occupied slices, combinational path delay and macros have been tabulated above.

This was the first method associated with replacing the existing architectural designs of a full adder by applied adder designs namely Generic addition, MUX based approach, propagate and generate based approach, half adder based approach, threshold logic based approach. The study was thus continued by considering the standard add a blocks such as Kogge Stone adder and Brent Kung adder for all 4 bit, 8 bit and 16 bit combinations. The generic design comparison for all the three combinations of full ladder have been shown in the table below which represents the Slice LUT's, occupied slices, combinatioal path delay and macross for both Kogge Stone adder and Brent Kung adder.

	4 bit KSA	8 bit KSA	16 bit KSA
Slice LUT's	4	10	16
Occupied Slices	2	5	9
Combinational Path delay	1.424ns (0.195ns logic, 1.229ns route) (13.7% logic, 86.3% route)	2.951ns (0.668ns logic, 2.283ns route) (22.6% logic, 77.4% route)	4.678ns (0.933ns logic, 3.748ns route) (23.2% logic, 76.8% route)
Macros	1-bit xor2 - 4 4-bit xor2 - 1	1-bit xor2 - 16	1-bit xor2 - 32

Table 5.1.3.5. Performance Analysis of Kogge Stone Adder

	4 bit BKA	8 bit BKA	16 bit BKA
Slice LUT's	4	10	16
Occupied Slices	2	5	9
Combinational Path delay	1.567ns (0.122ns logic, 1.445ns route) (12.1% logic, 87.9% route)	2.987ns (0.560ns logic, 2.427ns route) (21.4% logic, 78.6% route)	4.753ns (0.832ns logic, 3.921ns route) (22.8% logic, 77.2% route)
Macros	2-bit xor2 - 4 4-bit xor2 - 1	1-bit xor2 - 16	1-bit xor2 - 32

Table 5.1.3.6. Performance Analysis of Brent Kung Adder

Further the same design implementation was continued by replacing the existing adder blocks of 4 × 4, 8 × 8 and 16 × 16 Baugh-Wooley multiplier and out of the results obtained we still found that propagate and generate was giving us the minimum combinational path delay.

	4 bit BW	8 bit BW	16 bit BW
Slice LUT's	13	78	368
Occupied Slices	5	44	209
Combinational Path delay	4.231ns (0.679ns logic, 3.552ns route) (14.1% logic, 85.9% route)	8.390ns (1.045ns logic, 7.345ns route) (12.9% logic, 87.1% route)	17.245ns (2.351ns logic, 14.894ns route) (13.6% logic, 86.4% route)
Macros	1-bit 2 input xor -15	1-bit 2 input xor - 118	1-bit 2-to-1 multiplexer-486

Table 5.1.3.7 Computational Results of Baugh-Wooley Using Kogge Stone Adder (KSA)

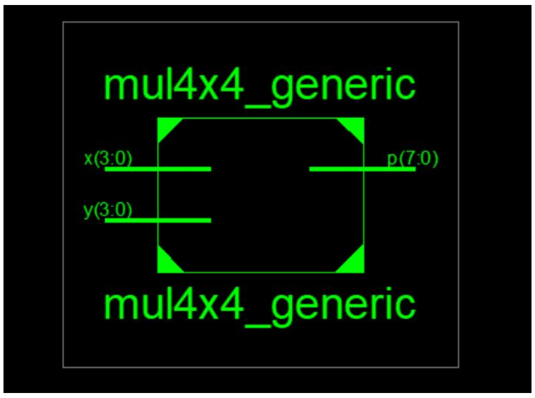
When compared to the standard architectures of Kogge Stone adder and Brent Kung adders. The systematic design implementation and the results obtained have been tabulated as below,

	4 bit BW	8 bit BW	16 bit BW
Slice LUT's	16	79	386
Occupied Slices	7	23	213
Combinational Path delay	3.103ns (0.283ns logic, 2.820ns route) (11.8% logic, 88.2% route)	8.039ns (1.257ns logic, 6.782ns route) (13.7% logic, 86.3% route)	17.340ns (2.371ns logic, 14.969ns route) (13.4% logic, 86.6% route)
Macros	1-bit 2 input xor - 27	1-bit 2 input xor -59	1-bit 2-to-1 multiplexer- 276 1-bit 2 input xor - 486

Table 5.1.3.8 Computational Results of Baugh-Wooley Using Brent Kung Adder (BKA)

5.1.4. IMPLEMENTAION DETAILS

The implementation RTL Schematics generated from the Xilinx ISE Design Suite is captured below to validate our behavioral model



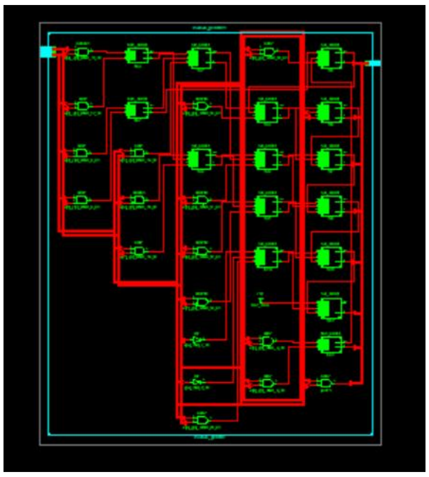
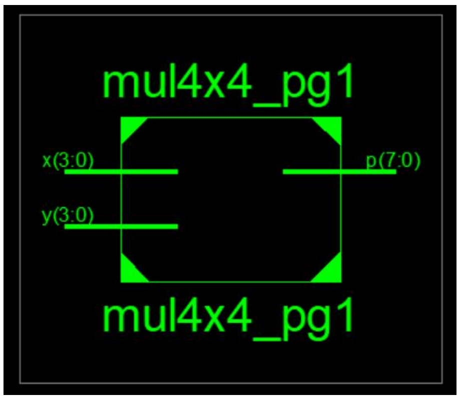


Figure 5.1.4.1 The RTL Schematic Top Level Module of Generic Multiplier



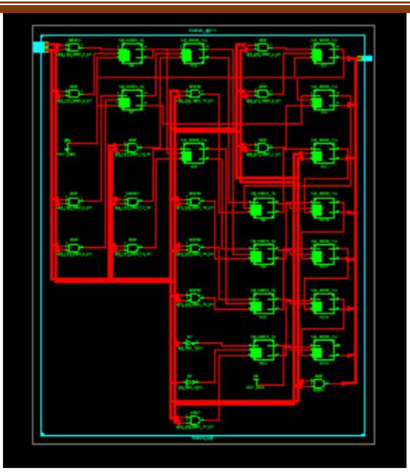


Figure 5.1.4.2 The RTL Schematic Top Level Module of PG Multiplier

5.2. DESIGN IMPLIMENTATION ON CADENCE VIRTUOSO

This section contains experimental set up, simulation results, comparative study and implementation details of both Cadence Virtuoso software.

5.2.1. EXPERIMENTAL SETUP

To begin the design implementation on Cadence Virtuoso, the process typically starts with creating a schematic representation of the desired circuit. The designer can use the Virtuoso Schematic Editor, which provides a user-friendly interface for drawing circuit schematics. The Schematic Editor allows the designer to place and connect various components, such as transistors, resistors, capacitors, and other IC building blocks, to create the desired circuit topology.

Once the schematic is complete, the designer can perform various simulations using Cadence Virtuoso's simulation tools. This allows for the analysis and verification of the circuit's behavior and performance. The designer can run transient analyses, AC analyses, DC analyses, and other simulations to evaluate the circuit's functionality and ensure it meets the design specifications.

5.2.2. SIMULATION RESULTS

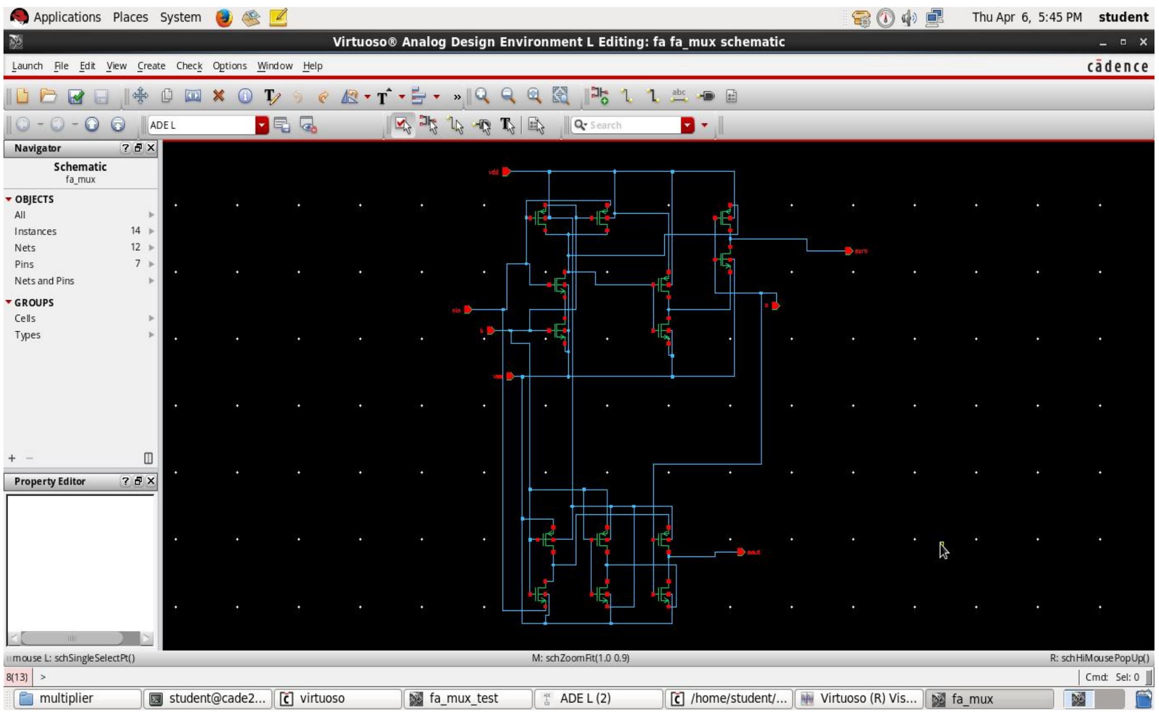


Figure 5.2.2.1. MUX based design schematic for a full adder plotted on cadence virtuoso

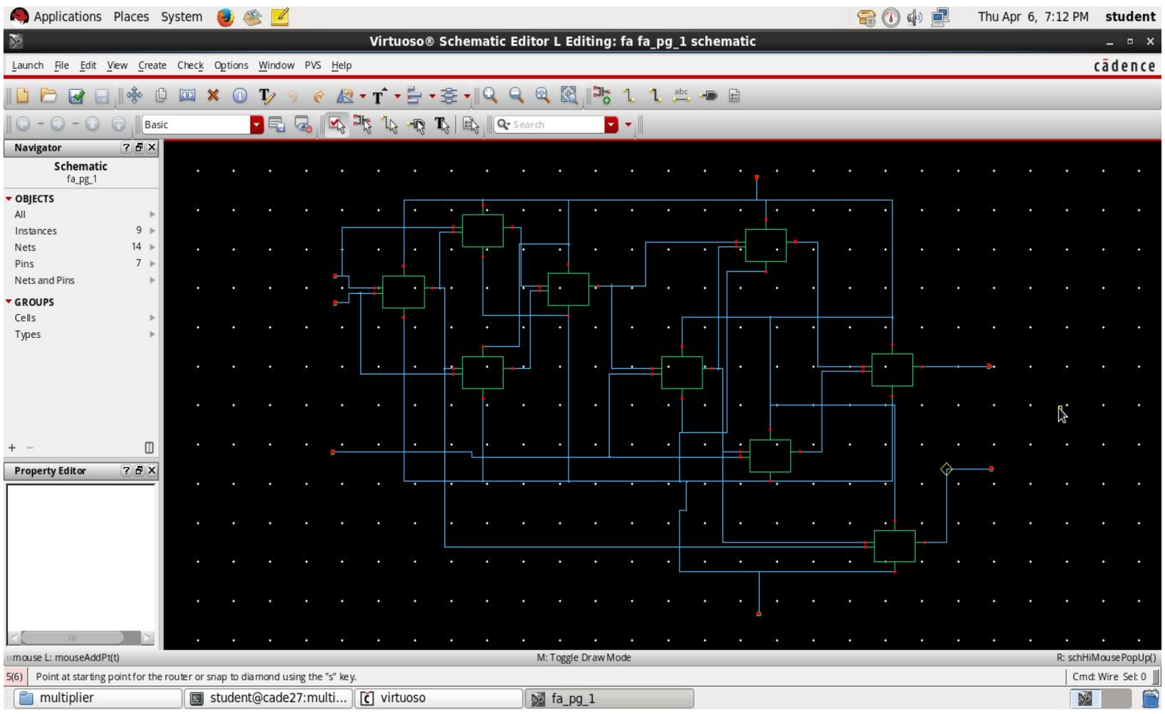

Applications Places System
Thu Apr 6, 7:12 PM student
Virtuoso® Schematic Editor L Editing: fa fa_pg_1 schematic
Launch File Edit View Create Check Options Window PVS Help
cadence
Basic
Navigator
Schematic
fa_pg_1
OBJECTS
All
Instances 9
Nets 14
Pins 7
Nets and Pins
GROUPS
Cells
Types
Property Editor
mouse L: mouseAddPt()
M: Toggle Draw Mode
R: schHiMousePopUp()
5(6) Point at starting point for the router or snap to diamond using the "s" key.
Cmd: Wire Sel: 0
multiplier
student@cade27:multi...
virtuoso
fa_pg_1

Figure 5.2.2.2. Propagate and generate based design schematic for a full adder plotted on cadence virtuoso

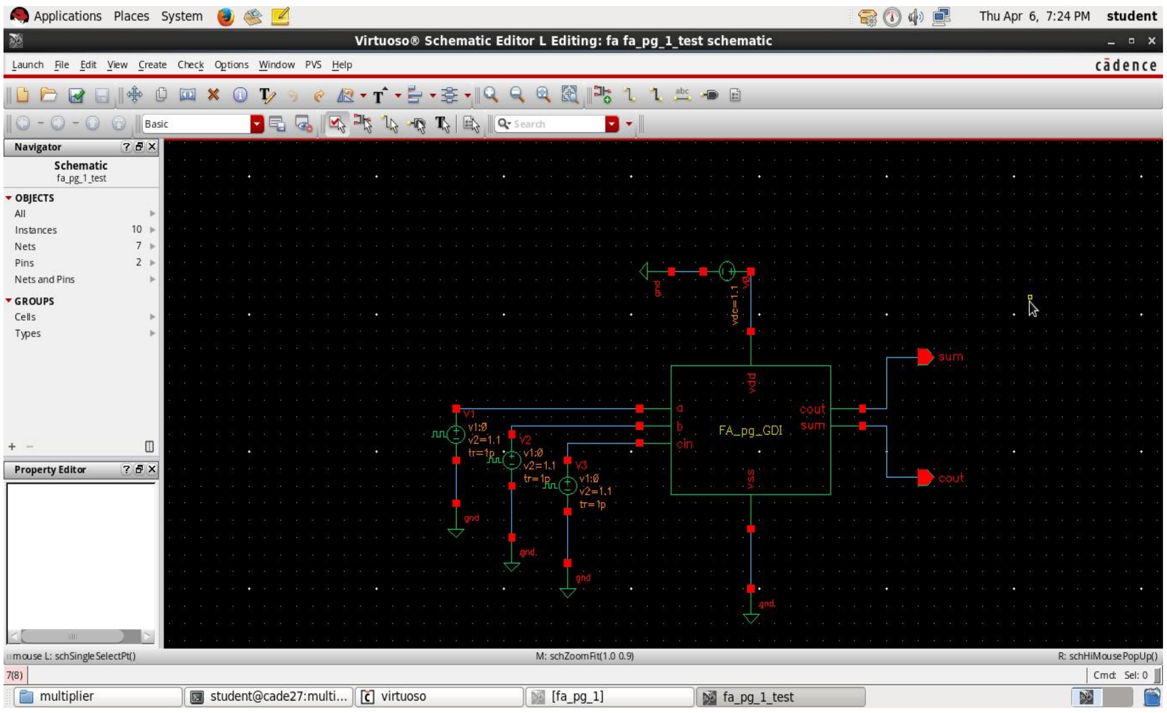


Figure 5.2.2.3. Propagate and generate based design test for a full adder plotted on cadence virtuoso

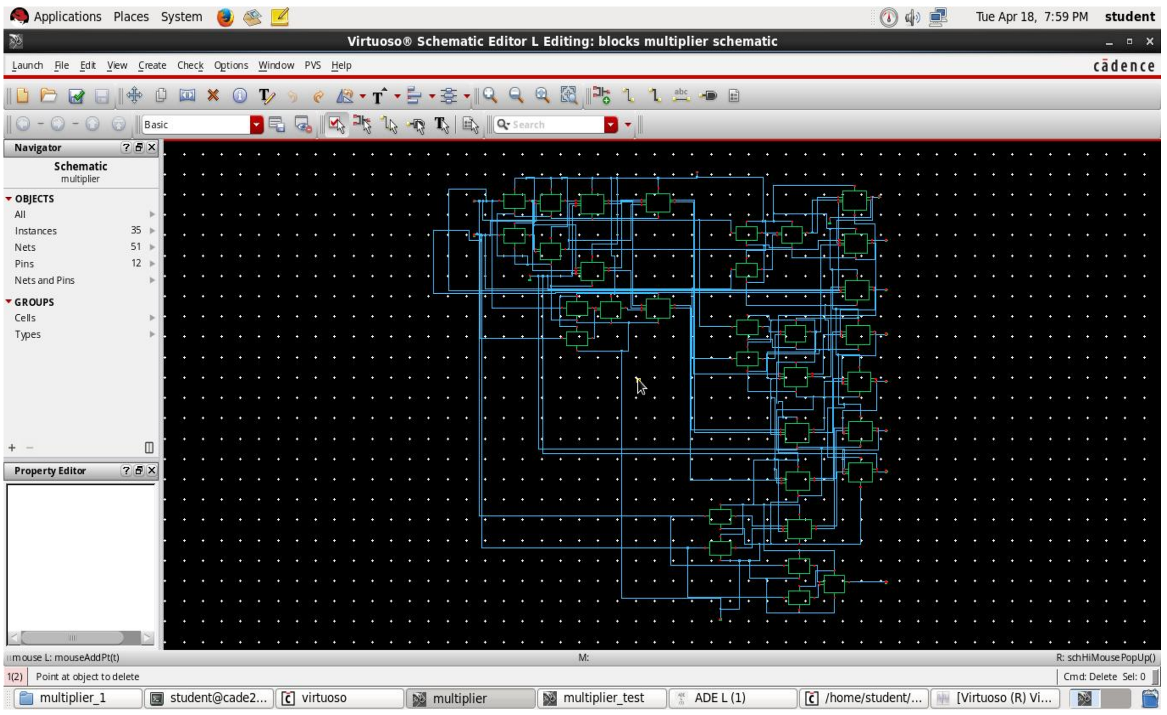


Figure 5.2.2.4. Multiplier design schematic plotted on cadence virtuoso

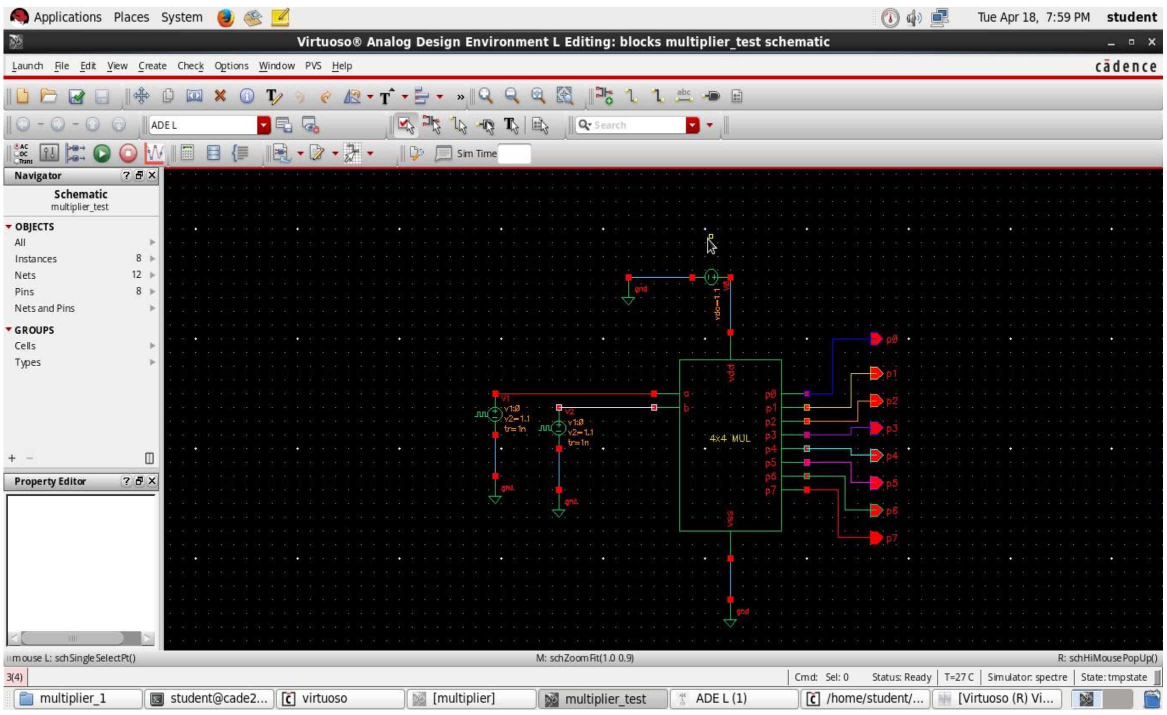


Figure 5.2.2.5. Multiplier design test plotted on cadence virtuoso

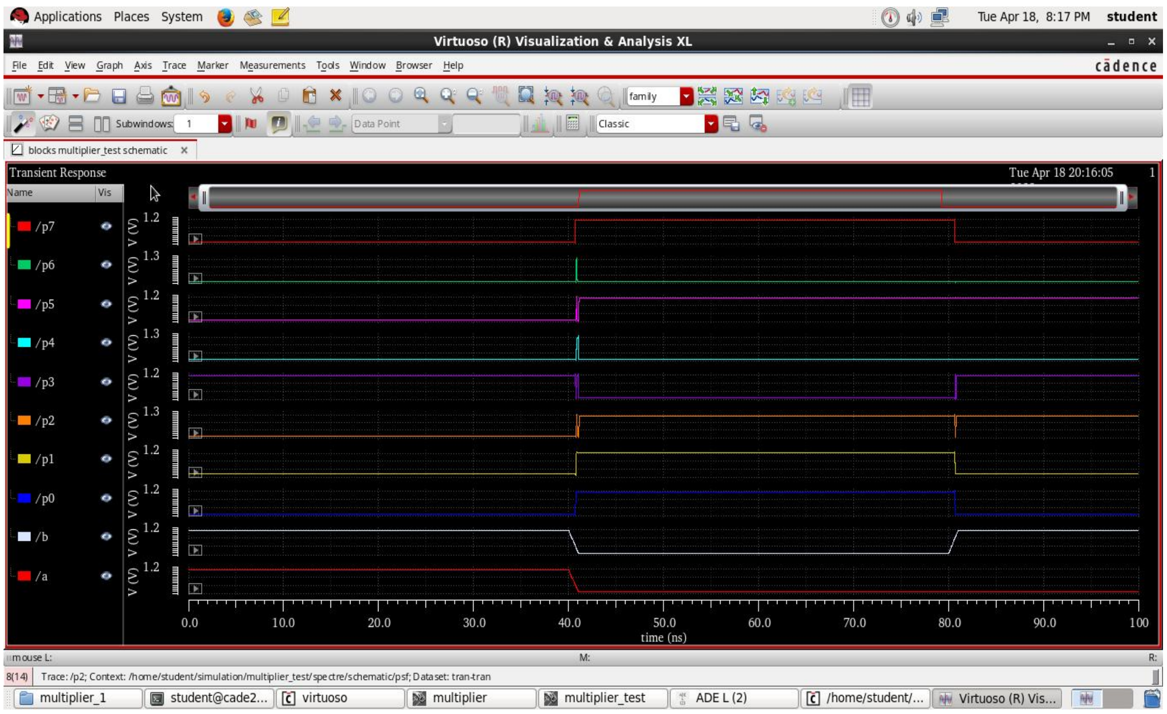

Virtuoso (R) Visualization & Analysis XL
File Edit View Graph Axis Trace Marker Measurements Tools Window Browser Help
blocks multiplier_test schematic
Transient Response
Tue Apr 18 20:16:05
/p7
/p6
/p5
/p4
/p3
/p2
/p1
/p0
/b
/a
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100
time (ns)
Trace: /p2; Context: /home/student/simulation/multiplier_test/spectre/schematic/psf; Dataset: tran-tran

Figure 5.2.2.6. Resulting waveform of a multiplier design plotted on cadence virtuoso

5.2.3. COMPARATIVE STUDY

It is important to note that accurately predicting power consumption in a Baugh-Wooley multiplier implemented on Cadence Virtuoso requires detailed analysis, simulation, and optimization specific to the design. Power estimation tools and methodologies provided by Cadence or other power analysis tools can assist in evaluating and optimizing power performance during the design phase.

4 bit Baugh-Wooley Multiplier		**MUX based Design**	**Propagate and Generate based Design**
Delay In Seconds	P0	$143.8e^{-12}$	$143.7e^{-12}$
	P1	$774.1e^{-12}$	$302.4e^{-12}$
	P2	$775.9e^{-12}$	$301.2e^{-12}$
	P3	$769.6e^{-12}$	$310.5e^{-12}$
	P4	$778.1e^{-12}$	490.1^{-12}
	P5	$779.6e^{-12}$	$649.3e^{-12}$
	P6	$887.8e^{-12}$	$302.1e^{-12}$
	P7	$489.3e^{-12}$	$889.7e^{-12}$
Number of Transistors		252	398
Static Power		$130.2e^{-6}$	$22.03e^{-6}$
Dynamic Power		$2.103e^{-4}$	$2.423e^{-5}$
Total Power		0.0003612	0.00007093

Table 5.2.3.1. Performance Tabulation on Cadence Virtuoso

Chapter 6

CONCLUSION AND FUTURE SCOPE

We have presented a detailed study on the design and development of a high speed multiplier typically for digital signal processing applications.
We have studied all the basic elements that make up a multiplier in a particular DSP unit.

Starting with the basic idea of what is the process of multiplication, we have covered aspects such as what is the necessity of signed multiplication, what are the different multiplier architectures and a detailed comparison of the advantages of one multiplier architecture over the other. We further came to the conclusion (based on literature survey) that Bough-Wooley multiplier architecture would be a suitable choice for our targeted application. We have gone further and analyzed the Bough-wooley multiplier architecture on FPGA level(xc7a100t-3fgg676) and obtained the computational results (LUT slices and delay parameters) for the same. We further explored various other ways of carrying out addition on the Bough-wooley multiplier architecture such as generic addition, propagate and generate approach, half adder approach, threshold logic based approach and MUX based addition. We discovered that propogate and generate based addition yielded the best result for optimized resource usage and high speed.

Further we have implemented the same architecture on Cadence Virtuoso

Chapter 7

REFERENCES

[1] B. Shivaprasad, K. Shinde, V. Muddi, Design and implementation of parallel floating point matrix multiplier for quaternion computation, Control, Instrumentation, Communication and Computational Technologies (ICCICCT),2015, pp. 289–293.

[2] R Zimmermann and W Fichtner, Fellow, IEEELow-Power" Logic Styles: CMOS Versus Pass-Transistor Logic," IEEE Journal Of Solid-State Circuits, vol. 32, no. 7,pp.1079- 90,1997.

[3] D Markovic, B Nikolic, and V G Oklobdzija, "A general method in synthesis of pass-transistor circuits," Microelectr. J, vol. 31,pp. 991-8,2000.

[2] C H Chang, J Gu, and M Zhang, "A review of O.J8-mm full adder performances for tree structured arithmetic circuits," IEEE Trans. Very Large Scale Integr. (VLSI) Syst. Vol. 13 pp. 686-95, 2005.

[3] Massimo Alioto, Member, IEEE, and Gaetano Palumbo, Senior Member, IEEE, "Analysis and Comparision on Full Adder Block in Submicron Technology," IEEE Transactions On Very Large Scale Integration (VLSI) Systems, vol. 10, no. 6, pp. 806-23, 2002.

[6] L Sousa and R Chaves, "A universal architecture for designing effiCient modulo 2n+ 1 multipliers, " IEEE Trans. Circuits Syst.I: Regular Papers, vol. 52, pp.1166-78, 2005.

[7] T Oscal, C Chen, S Wang, and Y W Wu, "Minimization of switching activities of Partial Products for Designing LowPower Multipliers, "IEEE Transaction on Very Large Scale Integration (VLSI) Systems,vol. II, no. 3, pp. 418-433,2008.

[8] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," IEE Proc.: Circuits, Devices, Syst., vol. 141, no. 5, pp. 407-413, Oct. 1994.