DESIGN AND IMPLEMENTATION OF HYBRID FULL ADDER BASED 16-BIT MULTIPLICATION USING FPGA

A Project report submitted in partial fulfilment of the requirements for

the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

B USHA SRI (319126512007),

S SAI DEEPAK (319126512050),

G TARUN(319126512015)

I SATHYA SAGAR (319126512020).

Under the guidance of

Dr.K.V.Gowreesrinivas

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC AUTONOMOUS)

(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade)

Sangivalasa, Bheemili mandal, Visakhapatnam dist.(A.P) (2022-2023)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade)

Sangivalasa, Bheemili Mandal, Visakhapatnam dist.(A.P)

ANITS

CERTIFICATE

This is to certify that the project report entitled **"DESIGN AND IMPLEMENTATION OF HYBRID FULL ADDER BASED 16 BIT MULTIPLICATION USING FPGA"** submitted by **B USHA SRI(319126512007),S SAI DEEPAK (319126512050),G TARUN (319126512015),I SATHYA SAGAR(319126512020)** in partial fulfillment ofthe requirements for the award of the degree of **Bachelor of Engineering** in **Electronics & Communication Engineering** of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

Project Guide

K.V.G. Sri

Dr.K.V.GowreeSrinivas
Assistant Professor
Department of E.C.E
ANITS

Assistant Professor
Department of E.C.E.
Anil Neerukonda
Institute of Technology & Sciences
Sangivalasa, Visakhapatnam-531 162

Head of the Department

Dr. B.Jagadeesh
Professor
Department of E.C.E
ANITS

Head of the Department
Department of E C E
Anil Neerukonda Institute of Technology & Sciences
Sangivalasa - 531 162

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Dr.K.V.Gowreesrinivas** Assistant professor, Department of Electronics and Communication Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. B. Jagadeesh**, Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa ,** for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of ECE, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. Last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS

B USHA SRI (319126512007)
S SAI DEEPAK (319126512050)
G TARUN (319126512015)
I SATHYA SAGAR (319126512020)

ABSTRACT

In this,16 bit multiplication is implemented using higher order compressors such as 4:3, 5:3,6:3 ,7:3,8:4,9:4,10:4,11:4,15:4 and a higher order compressor based efficient 16-bit multiplier is proposed. Because of use of compressors, we obtain better area, better space than conventional multiplication. Further implementation of compressor based multiplication; we have introduced hybrid full adders using nand gates in our design. After introducing hybrid full adder's better area, better space is obtained. In this, 16 bit multiplication, we use Wallace tree structure. The Wallace tree method is applied to determine outcome of binary number multiplication. This approach transfers the result to the following stage after adding binary bits. All modules are implemented using verilog HDL and synthesized using Xilinx vivado16.1.

CONTENTS

List of Figures:

CHAPTER-1

INTRODUCTION

The multiplier is one of the most frequently used arithmetic understanding paths in modern digital design. Multiplication is a crucial and computationally challenging operation. The multiplication operation is advantageous to many parts for a digital system or computing device, especially signal processing, graphics, and mathematical calculations. The most crucial considerations in VLSI design today are power consumption, area, and speed. Particularly, in digital signal processing uses where plenty of multiplications are necessary, the multiplier design is crucial. hybrid adder is a good choice in terms of power consumption and speed than HPSC but at the cost of increased number of transistors in the design . We use hybrid full adders in the place of adders to reduce the power consumption and area. Multipliers require high amount of power and delay during the partial products addition. At this stage, most of the multipliers are designed with different kind of adders that are capable to add two/three or at most 4 bits by using 4-2 compressors. For higher order multiplications, a huge number of adders or compressors are used to perform the partial product addition.

1.1 PROJECT OBJECTIVE:

The main objective of this project is to study, design and implementation of hybrid full adder based 16-bit multiplication using FPGA. The Synthesis and Implementation is done for 16-bit Multiplier using hybrid full adders using Xilinx Vivado design suite. The performance comparison of proposed work on various FPGA families is observed in terms of area and power.

1.2 PROJECT OUTLINE:

This project report is presented over the remaining chapters.

Chapter 1 presents Introduction.

Chapter 2 describes introduction to hybrid full adders.

Chapter 3 explains Introduction to Multipliers and different types of multipliers.

Chapter 4 is about Introduction to Compressor Adders and its different types.

Chapter 5 describes Introduction to Verilog.

Chapter 6 mainly gives the description about working with XILINX ISE DESIGN SUITE.

Chapter 7 presents the simulation results which are simulated using Vivado Design Suite simulator. Finally, the results of the project work and conclusions are drawn.

CHAPTER 2

HYBRID FULL ADDERS

The majority of the the multiplier partial product bits are generated by NAND gates rather than AND gates thanks to hybrid adders.The hybrid full adder uses less energy and requires fewer transistors overall. Instead of the 6-transistor AND gates, a hybrid fulladder uses 4-transistor CMOS NAND gates. The primary distinction between hybrid adders and conventional full adders is the fact that a few of the signal inputs and output signals used by hybrid adders are inverted.

The location of an inverter that needs to be added to turn the suggested hybrid adder back into a regular full adder is indicated by a bubble at any output or input of the hybrid adder (Fig). Therefore, an AND gate's regular partial product (PP) bit must first be inverted before it can be connected to a bubbled pin. The PP is instead made from a NAND gate and connected directly, without inversion, to a bubbled pin. This is so that no inverters are required when a bubbled output is linked to a bubbled input (or two inverters are required but revoke one another out). In order to produce the multiplier's partial products, the newly introduced hybrid adders enable the use of NAND gates rather than AND gates.

For instance, the 3-2 adder of this type received its name because it requires three inverters at its X, Y, and Z the inputs and two inverters at its sum and Carry outputs in order to function as a standard full adder. Consequently, it executes the hybrid addition function.

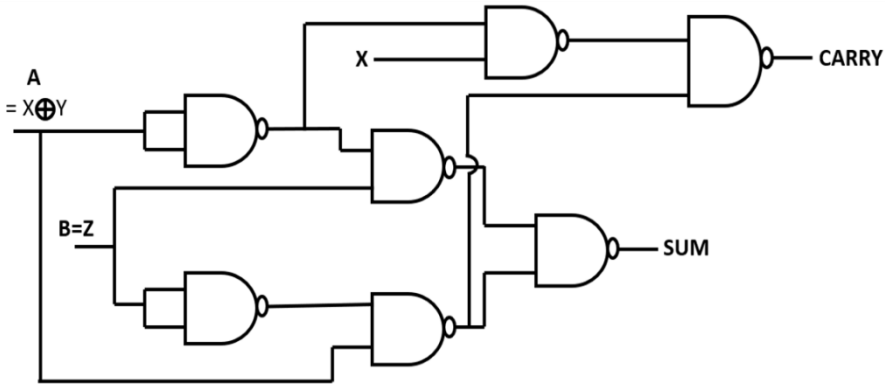


Fig1: Hybrid full adder

CHAPTER 3

INTRODUCTION TO MULTIPLIERS

Today, multipliers are extensively employed in areas such as digital signal analysis. In order to make multipliers suitable for various high speed, low power, and compact VLSI implementations, many researchers have tried and are still trying to creation of a multiplier which satisfy one of the two design targets listed as: excessive pace, minimal power use, consistency for the design and thus fewer space, or perhaps all of these goals within a single a multiplier. The "add and shift" technique of multiplication is the most popular. The most crucial factor affecting the performance of parallel multiplication is the amount of partial outcomes that must be merged. Well known technique for minimising the amount of incomplete products that must be appended is the Modified Booth algorithm.

The Wallace Tree method could be used to achieve faster speeds and fewer the steps of logical addition. We may also take advantage of the advantages of both techniques in creating one multiplier by merging the Wallace Tree method with the Modified Booth approach. fortunately as parallelism grows, so do the amount of changes among partial outcomes and intermediate sums that must be added, which may result in processing that is slower, requires more silicon because of irregular structure, and uses more power because of more link up because of the complicated route design.

While "serial-parallel" multipliers forgo performance in favor of improved space and power efficiency. Which type of multiplier is used—parallel or serial—with respect to the application. This lecture will discuss the design of multiplication algorithms and compare them on the basis of rapidity, space, power, and an assortment of all of these measures.

3.1 TYPES OF MULTIPLIERS

Serial Multiplier: When space and energy are critical and latency might be allowed, the serial multiplier is utilised. This arrangement uses a single adder to merge m * n partial products.

Serial/Parallel Multiplier: One component is pumped into the device serially whereas the additional operand is supplied in parallel via a serial/parallel multiplier. Each cycle results in the creation of N partial products. The M*N PPs multiplication table grows by one column with each succeeding cycle. After N+M cycles, the ultimate findings are saved in the outcome

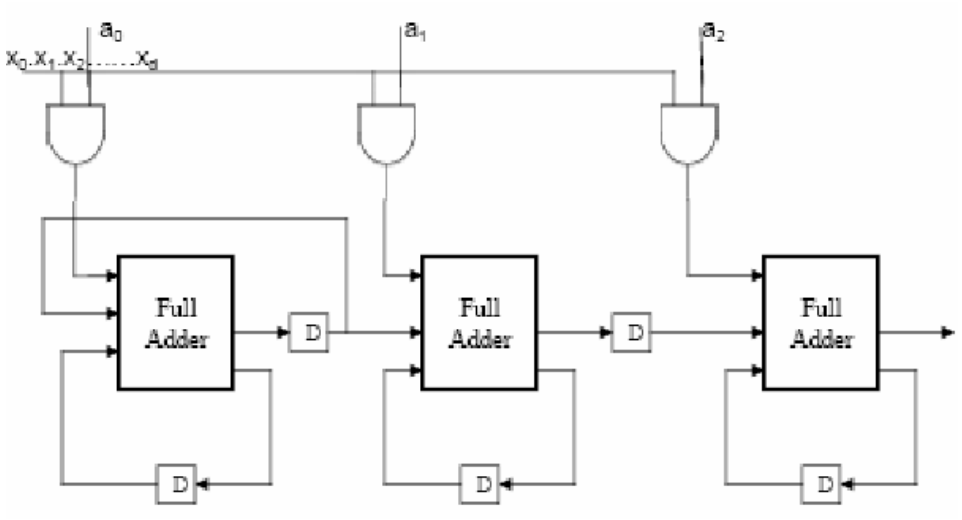


Fig2:Serial/parallelMultiplier

Shift and Add Multiplier: Based on the multiplier LSB bit value, a multiplicand value is introduced and compounded. The multiplier advances a single bit to the correct position with each minute of the clock, and its current position is verified. A shift operation is carried out if it is zero. The multiplicand is applied into the accumulation and migrated one bit to the right if the result is 1. The result is stored in the accumulator once each multiplier bit has been verified. The multiplier is initially stored in the N, LSBs of the accumulator, which has a size of 2N (M+N). N cycles of delay is the maximum.

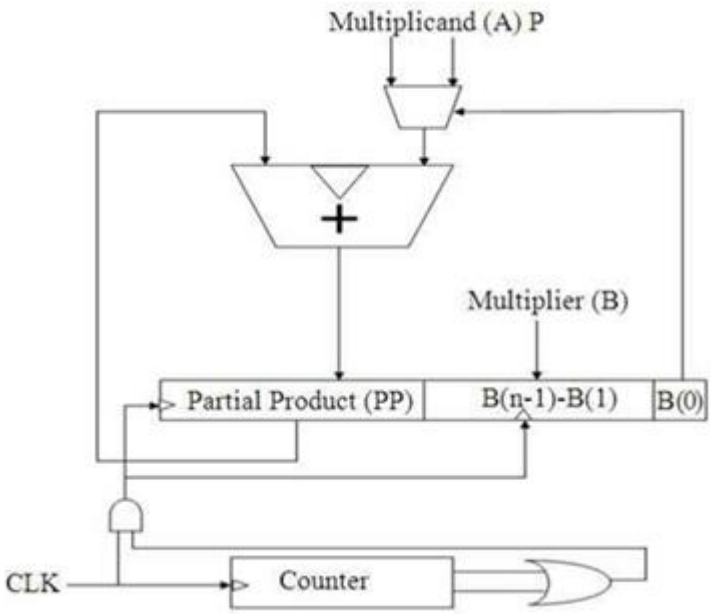


Fig3: Shift and add multiplier

Array Multiplier: The array multiplier is widely recognized because of its periodic construction. In the multiplier circuit, the add and shift algorithm is used. The multiplicand is multiplied by one multiplier bit to produce each partial product. Based on their bit ordering, the partial product is moved and subsequently added. A standard carry propagate adder can be used to perform the addition. M-1 adders are needed, where M is the multiplier length.

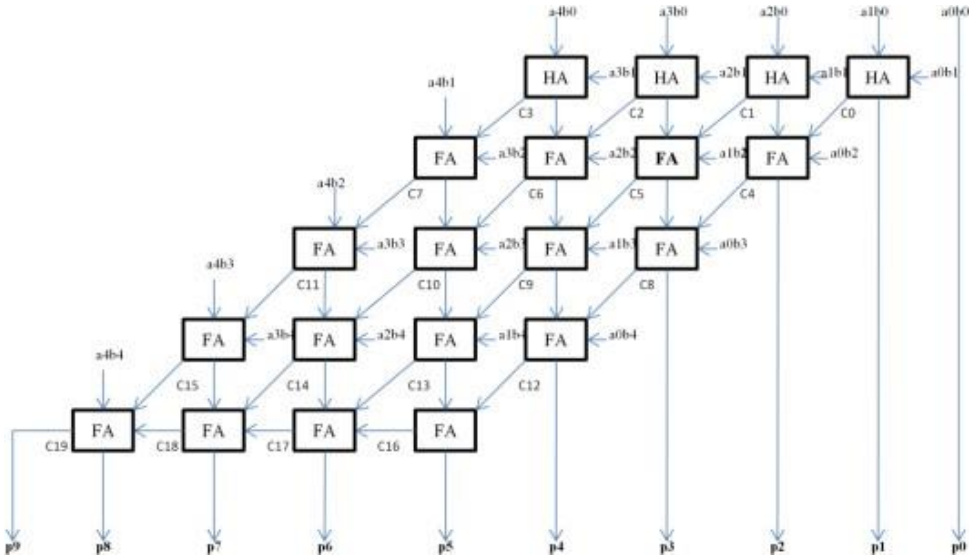


Fig 4: Arraymultiplier

Booth Multipliers: It is an efficient method for signed-number multiplication that evenly multiplies both neutral and positive numbers. Every multiplication value produces a multiple of the multiplicand which is appended to the intermediate result in a typical add-shift procedure. A lot of multiplicands must be added if the multiplier is really large. The amount of additions that must be made in this situation basically determines the multiplier delay. Minimizing the quantity of changes will boost efficiency. The multiplicand multiples can be reduced using the Booth algorithm. A more complex presentation radix necessitate a smaller number of digits to represent a given range of integers. Because high radix multiplication may handle several bits of the multiplier in each cycle, High radix multiplication can be used to deal with more than one bit of the multiplier in each cycle because a k-bit number in binary format can be symbolised as a radix-4 number with K/2 digits, a radix-8 number with K/3 digits, and furthermore.

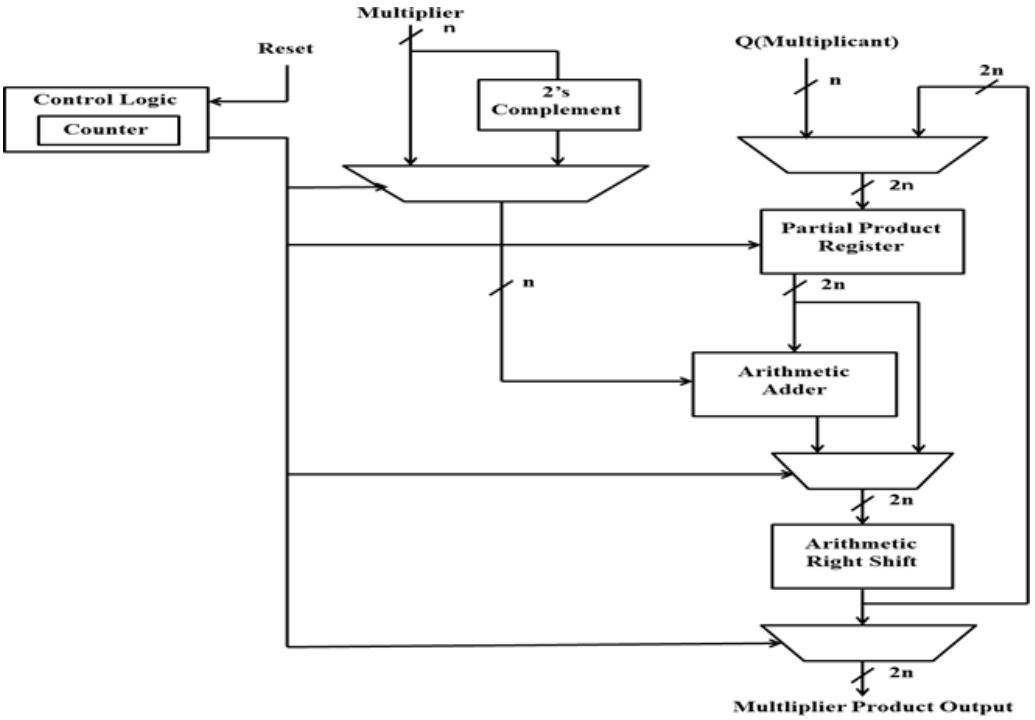


Fig 5: Booth Multiplier

Wallace tree Multiplier: A Wallace tree is a very effective hardware method for implementing an electronic circuit that multiply two different numbers. This was invented in 1964 by Chris Wallace, an Australian computer scientist**.**

The Wallace tree has following stages:

1. To get n 2 outcomes, divide each bit of one argument by each bit of the other. The wires' weights vary according on where the multiplied bits are located; for example, the wire holding the outcome of is 64. (Refer to the weights description following).

2. Layers of half and full adders limit the amount of partial products to two.

3.With a standard adder, divide the wires into two integers.

The second step is carried out as shown below. Add the next layer if there are three or more wires of the same weight.

• Any four identically weighed wires can be combined into a full adder. The outcome will be a heavier output wire and an output wire with the identical weight as each of the four input wires.

• If there are still two identical wires, feed them into a half-adder.

•Connect the final wire to the following layer if only one is left.

The Wallace tree benefits from having only reduction layers with an O (1) propagation delay.

Because the cost for producing partial products is O (1) and the cost of ultimate addition is O (log n), multiplication is just marginally worse than addition. (However, in the gate

count, much more expensive). O (log2 n) time is required for normal adders to add partial products.

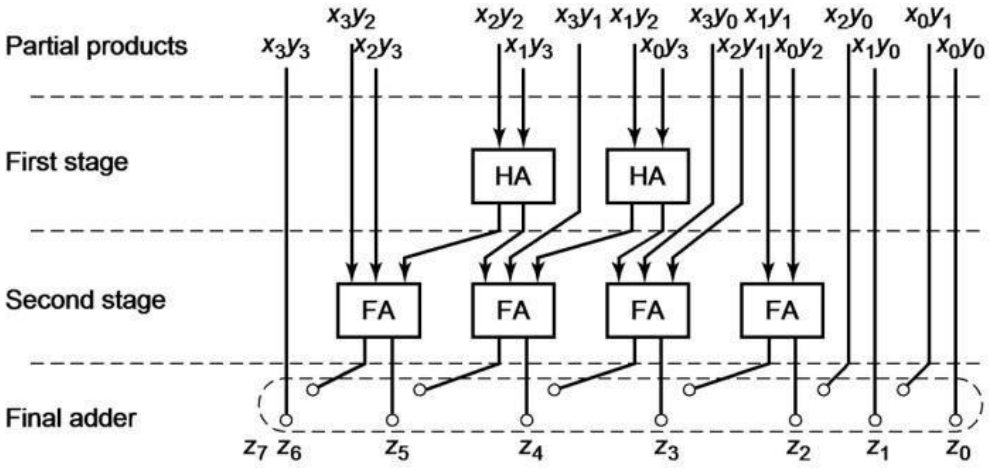


Fig 7:Wallace tree multiplier

CHAPTER 4
INTRODUCTION TO COMPRESSOR ADDERS

4.1 WHAT ARE COMPRESSOR ADDERS

In contrast to combinational circuits of half and full adders, compressor adders are fundamental circuits that add bits more than four at a time to produce better delay outcomes. The symbol for compressor design is N r, where 'N' stands for the number of bits fed in and 'r' stands for the overall number of 1s found in N bits. In contrast to adder circuits, it actually reduces gate counts and latency, hence the term compressor. The circuits of lower compressors have been improved in significant part through study. Higher compressors are also used in conjunction with this to add more bits. The most popular compressor architectures are 4-3, 5-3, 6-3, 7-3, 8-4, 9-4, and 10-4,11-4 and 15-4.

4.2 TYPES OF COMPRESSOR ADDERS

A. 4-3 COMPRESSOR ADDER: Four binary bits are added with the aid of a 4:3 compressor. It has three outputs and four inputs. Sum, Carry, and Auxiliary Carry are outputs. Utilizing a 4:3 compressor requires less wiring than using one full adder and three half adders independently.

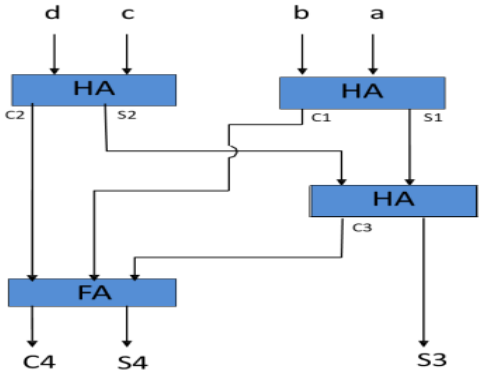


Fig 8: circuit diagram of 4-3 compressor adder

B. 5-3 COMPRESSOR ADDER: An 5:3 compressor combinational logic circuit accepts five inputs and generates three outputs. The three-bit output is created by adding the five input bits. So as to reduce stage delay, dissipation of power, and space in two ways while maximizing performance, we have given two topologies for 5:3 compressors.

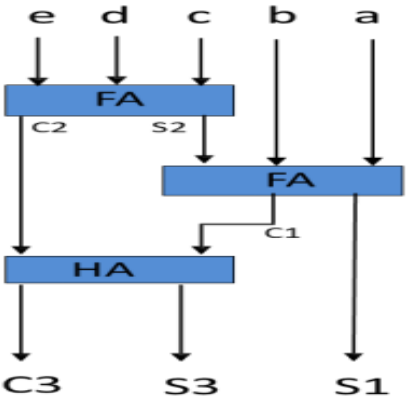


Fig 9: circuit diagram of 5-3 compressor adder

C. 6-3, 7-3 COMPRESSOR ADDER: The same function is likewise carried out by 6:3 and 7:3 compressors as it was by 4:3 and 5:3 before. However, some compressors have three outputs and additional inputs. A compressor combinational logic circuit with eight inputs , four outputs makes up an 8:4 compressor. It takes 8 bits as input and outputs its sum. Similar to this, a 6:3 and 7:3 compressor respectively takes 6 bit and 7 bit digits as inputs, performs operations on those bits, and outputs 3 bit.

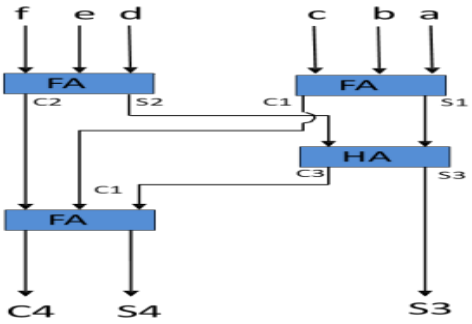


Fig 10: circuit diagram of 6-3 compressor

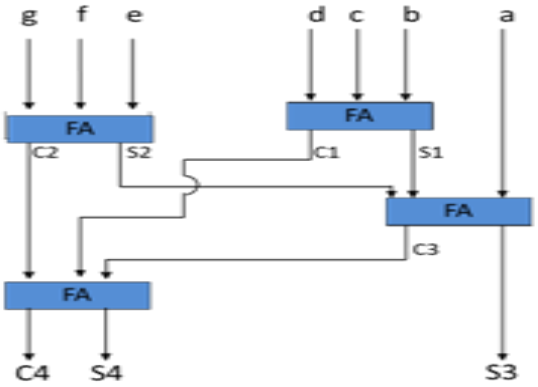


Fig 11: modified circuit diagram of 7-3 compressor

D. 8-4, 9-4 COMPRESSOR ADDER:A 8:4 compressor combinational logic circuit with eight inputs and four outputs makes up an 8:4 compressor. It takes 8 bits as input and outputs its sum.A8:4 and 9:4 compressor respectively takes 8 bit and 9 bit digits as inputs, performs operations on those bits, and outputs 4 bit.

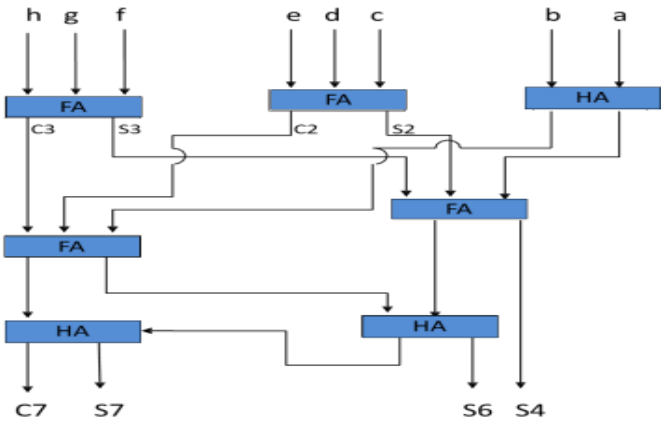


Fig 12: circuit diagram of 8-4 compressor

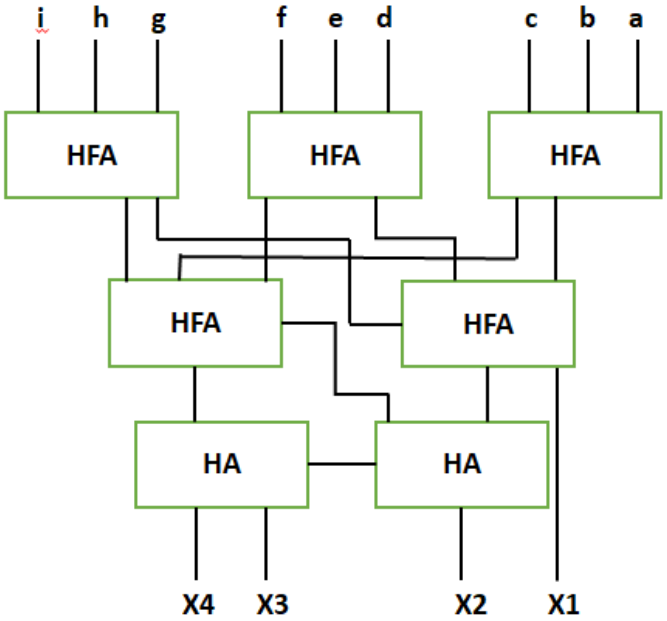


Fig 13: circuit diagram of 9-4 compressor

E. 10-4, 11-4 COMPRESSOR ADDER:A 10:4 compressor combinational logic circuit accepts ten inputs and generates four outputs. The four-bit output is created by adding the ten input bits. Similar to this, an 11:4 compressor takes eleven inputs and generates four outputs.

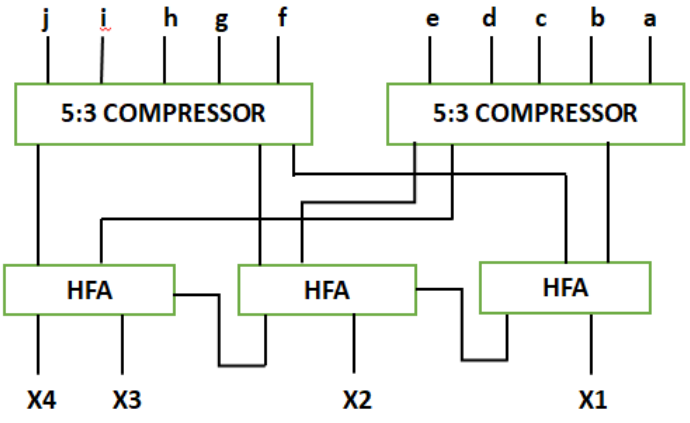


Fig 14: circuit diagram of 10-4 compressor adder

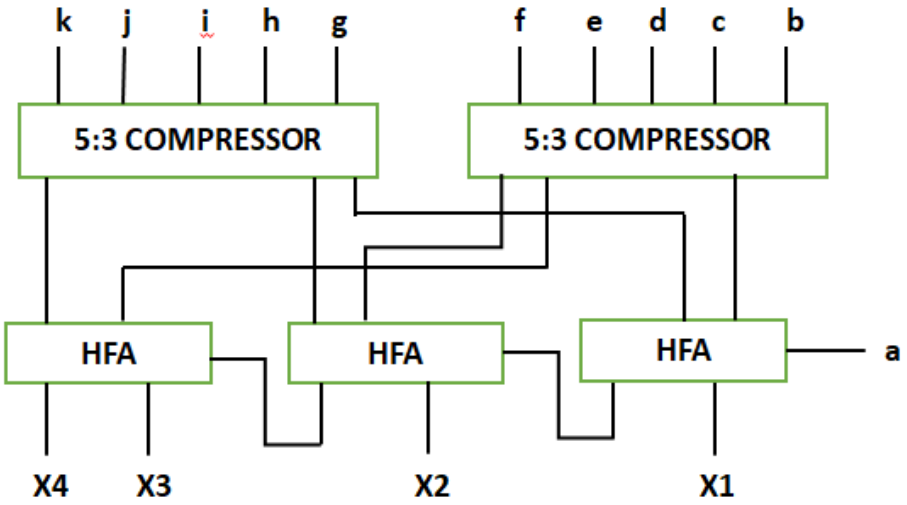


Fig 15: circuit diagram of 11-4 compressor adder

15-4 COMPRESSOR ADDER:A 15:4 compressor combinational logic circuit accepts fifteen inputs and generates four outputs. A 3 bit parallel adder is a combinational circuit accustomed to add three bit binary numbers. A 15:4 compressor combinational logic circuit accepts fifteen inputs and generates four outputs. The addition of bits is used to multiply two binary values. This bit addition can involve as little as two bits, as many as three bits, or as many as n bits. Before, only half adders and full adders could be accustomed to add any number of bits, which was complicated, time-consuming, and required a lot of design space. Here, we design a 16-bit multiplier for both high performance and space efficiency employing higher order compressors in two different ways.

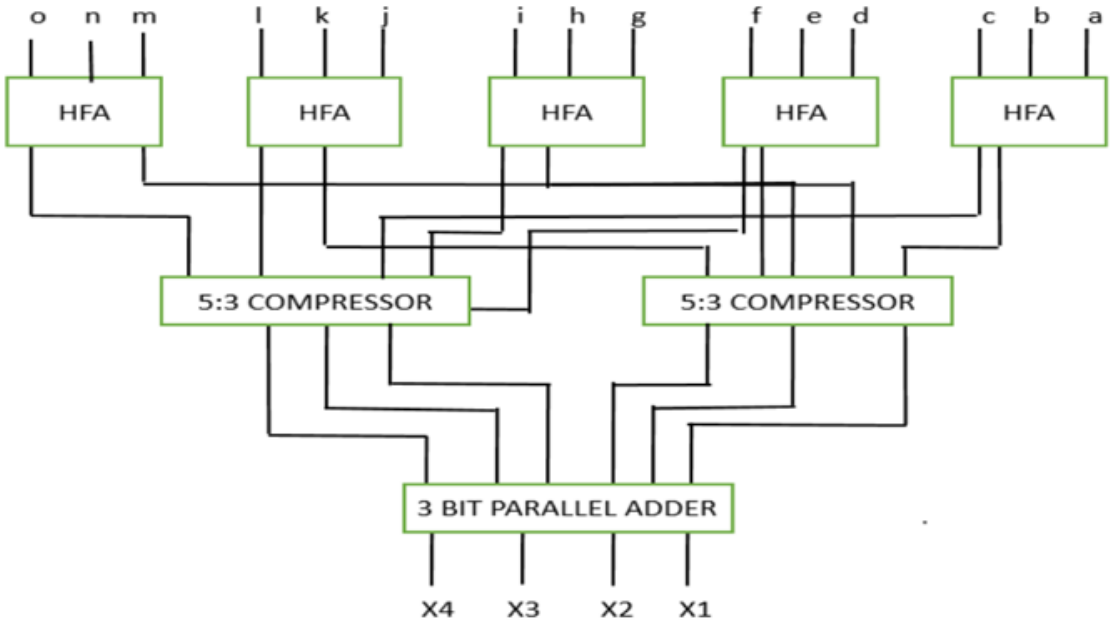


Fig 16: circuit diagram of 15-4 compressor adder

CHAPTER 5
INTRODUCTION TO VERILOG

5.1 DEFINITION:

The HARDWARE DESCRIPTION LANGUAGE (HDL) Verilog is utilised for describing a digital system, such as a flip-flop, microprocessors, network switch, or another component of electronic machinery. Verilog was created to streamline the procedure and boost the HDL's robustness and flexibility. Verilog was created to expedite the creation process by giving engineers the ability to define the capabilities of desired equipment and allowing tools for automation transform the behaviour into real-world components like a combination gates and sequential logic. Verilog is currently the HDL that is used and practised the most in the semiconductor industry. Similarly to every other hardware description language, Verilog has applications. It allows designers to create designs in either a bottom-up or top-down manner.Bottom-Up Design

Electronic design has typically been done from scratch. Each design is carried out using the common gates at the gate level. It is possible to develop new fundamental, arranged design techniques using this design.

Top-Down Design:

Numerous benefits are provided by it, such as early testing, easy technological advances switching, especially ordered system design, and others.

5.2 HISTORY OF VERILOG:

Verilog HDL can be traced back to the development of a language for describing hardware and a logic simulators called Verilog-XL by a business called Gateway Designing Technology in the 1980s. In 1989, Cadence Designing Systems purchased Gateway along with their authorizations for the simulators and language. Cadence released the language into the public domain in 1990 with the goal of making it an extensively used, not proprietary language. because Open Verilog International (OVI) and VHDL Internationally combined, the Verilog HDL is now maintained by the nonprofit organisationAccellera. OVI, who was responsible for the language's IEEE standardization, had to handle this. Verilog HDL got IEEE Std. 1364-1995 in the last month of 1995. In 2001, IEEE Std. 1364-2001, an edition that underwent significant revisions, was published.

One more modification was made in 2005, but it only made a few small adjustments. Accellera has created an entirely novel Verilog expansion standard called System Verilog. System Verilog was included in the IEEE standard (1800–2005) in 2005.

5.3 USES OF VERILOG:

It is simpler to conceal the specifics of Verilog's technological use due to its level of level of abstraction. For instance, a D flip-flop development would require knowledge of the transistor placement, in addition to the rise, fall, and CLK-Q times required to latch the current value onto a flop, in order to accomplish a positive-edge triggered FF. A deeper comprehension of a transistor's physical properties would additionally be necessary to control power consumption, timing, as well as the ability to drive nets along with other flops. Verilog will allow us to concentrate on thebehaviour while we take responsibility for the rest later.

5.4 FEATURES OF VERILOG:

- At Verilog Case-sensitivity is supported in Verilog.
- In Verilog, keywords are defined using letters that are lowercase.
- The majority of Verilog's the syntax is taken from the "C" language of programming.
- At the algorithm, RTL, gate, and switch levels, digital electronics can be modelled using Verilog.
- In Verilog, packaging is not a concept.
- The proficient simulations features TEXTIO, PLI, and UDPs are also supported.

5.5 DATA TYPES:

Several new data types are introduced by Verilog. RTL descriptions are simpler to write and comprehend thanks to these data types. The Verilog Hardware Description Language (HDL) uses data types to represent the data storage and transmission elements found in digital hardware.Data types in Verilog are separated into NETS and Registers. These kinds of data represent various hardware structures and can be stored and given values in various ways.

There are four fundamental values in the Verilog HDL value set:

Value	Description
0	Logic zero or false
1	Logic one or true
X	Unknown logical value
Z	The high impedance of the tri-state gate

5.5.1 INTEGER AND REAL DATA TYPES:

The vast majority of data types are known to C programmers. The concept is that if the data types in the two languages are the same, algorithms written in C can be translated into Verilog.Each of the bits in the brand-new two-state types of data that Verilog offers can only be either 0 or 1.

Simulators in RTL models might operate more effectively with two-state variables. Additionally, they have no impact on the outcomes of the the synthesis process.

Types	Description
bit	user-defined size
byte	8 bits, signed
shortint	16 bits, signed
int	32 bits, signed
longint	64 bits, signed

❖ **Two-state integer types:**

Verilog defines the total amount of bits for each of the fixed-width types, unlike C.

Types	Description
reg	user-defined size
logic	identical to reg in every way
integer	32 bits, signed

❖ **Four-state integer types:**

Regulation is inferior to logic, so we went with logic. Logic can be used in places where reg or wire was previously used.

Type	Description
time	64-bit unsigned
shortreal	like a float in C
shortreal	like double in C
realtime	identical to real

5.5.2 NON-INTEGER DATA TYPES:

❖ **Arrays:**

Variables, scalar networks, and vector networks can all be defined in Verilog. The memory arrays of values, which are arrays with one dimension of the variable type, can also be defined. Verilog removed some limitations on the use of memory arrays and permitted arrays with multiple dimensions of both nets and variables. Verilog, which enhances the idea of arrays and permits more array operations, improves this. Verilog arrays support both packed and unpacked dimensions.

❖ **Packed dimensions:**

- Undoubtedly organized in memory's sequential order.
- It can be used on any additional item that is packed.
- slicable
- ("part-selects") can only operate on "bit" types such as bit, logic, int, etc., certain of and this (like int) possess a fixed size.

❖ **Unpacked dimensions:**

- The simulator may decide how to set it up in memory. A reliable copy of an array onto another one of the same type can be made.
- There are guidelines for how to convert an unwrapped kind to a wrapped type when interacting with arrays of various types.
- Verilog encourages a variety of operations on whole unwrapped arrays and pieces of unwrapped arrays.
- The arrays or slices employed here must be of identical kind and shape, i.e., have a comparable number and length of unwrapped measurements.
- As long to be each array or a slice parts have the same amount of bits, the wrapped dimensions are flexible.

The permitted operations are:

- The whole array, pieces of an arrays, and individual array aspects can all be read and written to.
- Relationships between arrays of values, slices, to use and components that are equal.

Associative arrays and dynamic arrays, both of which have variable element counts during simulation, are also supported in Verilog.Verilog provides a wide range of arrays of querying functions and methods that accommodate each one of these array kinds.

5.6 NETS:

The association arrays and changing arrays, both of that feature non-contiguous ranges and variable element counts during simulation, are also supported in Verilog. For each one of these the array types, Verilog offers an assortment of arrays of searching operates and techniques.

When a signal is received, a net data type must:

motivates for a few IT devices.

As an in or outside the band port, it is listed.

To the left, right after the ongoing duty.

1. Wire:

When linking gates or modules, each wire in a circuit functions as a fictitious wire. Inside a function or block, the numerical value of a wire can be examine yet not was given. A wire is unable to be controlled by an ongoing duty declaration or through being linked to a gate's or module's results because it cannot keep its current value.

2. Wand (wired-AND)

The logical AND of all the drivers attached to a wand determines its value.

3. Wor (wired-OR)

Its value is established by the reasonable OR of every driver attached to wor..

4. Tri (three-state)

All drivers linked to the tri must have z, with the exception of the function that decides tri's value.

5. Supply0 and Supply1

The wires linked to logic zero and logic 1 are marked by supplies either 0 or 1 respectively.

5.7 REGISTERS:

The register class data object stores a value for upcoming procedural assignments. Just functions and legal blocks make use of them. A trigger in a method changes the contents of the data preservation element through the assignment declaration.

Reg was a variable type in Verilog, not an actual register by definition. Data is saved in multi-bit registers using unsigned numerals, and no sign extension is given to data that the user could have mistaken for two's complement. Reg, integer, duration, and real are examples of data types for registers.Regulated is the one that is used the most regularly. Reg is a term used to describe logic. An example of an universal variable is an integer. In particular, indexes variables, and constants are utilised by cycles. Instead of using the unsigned integers that explicitly specified reg types use, they maintain information as signed numbers. Their size is going to default to 32 bits if they contain digits that are not stated at the time of compilation. The synthesiser renders the necessary modifications to make these individuals the smallest possible width at compilation if they do contain constants. in the system's actual modules. To store modelling times in test benches, use Time and Realtime. The $time structure task and a period of time a 64-bit amount, can be paired for storing simulation time.

Note:A reg can also represent combinational logic, so it need not always represent a flip-flop. At the beginning of the simulation, the reg variables are set to x. The value x is present in any wire variable that isn't attached to anything. During the declaration, A register's or wire's size can be specified. When a register or wire has a size greater than one bit, it is referred to as a vector.

5.8 VERILOG STRING:

Reg is used to store strings, and the reg variable's width needs to be sufficient to hold the string. A string contains one byte and one ASCII value for each character. Verilog reduces the text's leftmost bits if a variable's size is smaller than the string's. Verilog, which provides zeros for the left side of the string if the parameter's size is larger than the string.

5.9 LEXICAL TOKENS:

The Verilog language for programming has similar lexical conventions to C. A stream of lexical tokens makes up the Verilog language's source text files.Any number of characters can make up a lexical token, and every one of them appears just once per token.

Strings, keywords, comments, numerals, and white space are all acceptable token types. Each line should end with a semicolon. (;). Verilog HDL is a case-sensitive language. Every keyword are lowercase as well.

White Space

Tab, blank, newline, and form feed characters can all be found in white space. Except when they are used to divide other tokens, these characters are ignored. But in strings, blanks and tabs are important.

Comments

The comments can be divided into two categories, including:

Comments appear on a single line, starting with the character // and finishing in the word return.

The single-line syntax, for instance, is //this.

The tokens /* and */ mark the beginning and end of multi-line comments, respectively.

/* this is multiline syntax/, for instance.

Identifiers

- The title provided to an object, such as a module, register, or function, serves as its identifier. Identifiers must start with an alphabetic letter or an underscore.
- Take A_Z and a_z as examples.
- Identifiers are made up of a dollar sign ($), an underscore, a number, and a string of letters. 1024 characters is the maximum they can have.
- Identifiers must start with an alphabetic character (a-z A-Z_) or an underscore.
- Identifiers may also include underscores, dollar signs, and alphabetic and numeric characters (a-z A-Z 0-9 _ $).

- Identifiers have a maximum length of 1024 characters.

Escaped Identifiers

By escape the identifier, Verilog HDL permits the use of every character in an identifier. Identifiers may include any accessible ASCII character, and this is what is meant by "escaped identifiers." Acceptable ranges are from 33 to 126 in decimal or from 21 to 7E in hexadecimal.Backslashes (/) are used to indicate that an identifier has been altered. The entire identifier escapes when there is a backslash.Commas, brackets, and semicolons grow an element of an escaped identifier until they are accompanied by white space, which concludes the fled identifier. After any escaped identifiers, add a space. If not, characters that should follow the identifier are taken into consideration to be asome of it.

5.10 OPERATORS

For defining control variables, special characters known as drivers are used. To manipulate variables, a single character, two, and even three different characters are used.

1. Arithmetic Operators

These people perform calculations. Both the + and - have binary (z-y) and unary (x) versions that can be used. Mathematical operators include addition, subtraction, multiplication, division, and the modulus.

2. Relational Operators

These operators compare two operands and give back just one bit, either a 1 or a 0, as a result. Numerous operators are used in relational operations:

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

3. Bit-wise Operators

Bit-by-bit comparison is what bit-wise operators do when comparing two operands. The bit-wise functioning includes the following operators:

&	Bit-wise AND
|	Bit-wise OR
~	Bit-wise NOT
^	Bit-wise XOR
~^ or ^~	Bit-wise XNOR

4. Logical Operators

Only single-bit operands are accepted by the logical operators, which are bit-wise operators. It returns one of the two bit values, either 0 or 1. It is possible to treat all values that are not zero as one, and they can be used with expressions, bits, and integers. The fact that logical operators work with expressions makes them frequently employed in statements with conditions. the individuals who perform logical operations.

The logical functioning includes a number of operators.:

!	logical NOT
&&	logical AND
||	logical OR

5.Reduction Operators

Reduction operators work on each bit in an operand vector and are the unary equivalent to bitwise operators. These also produce an amount of one bit.

The following individuals work in the reduction operation:

&	reduction AND
|	reduction OR
~&	reduction NAND
~|	reduction NOR
^	reduction XOR
~^ or ^~	reduction XNOR

7. Concatenation Operator

The concatenation operator combines two or more operands to produce a larger vector. The concatenation operation includes an additional operator:

8. Replication Operator

A given item is replicated in multiple copies by the replication operator. In the replication operation (n fold replication of an item), the operator "nitem" is utilised.

9. Conditional Operator

A multiplexer is produced by the conditional operator. It assesses one of both of the expressions based on the condition and is of identical type as that used in C/C++. (Condition)? Where operates the operator fit into conditional operations?

5.12 VERILOG MODULE:

A Verilog module is an element of code that executes a specific functionality. A more advanced section may interact with other lower- components using the input and output ports when they are embedded within one another.

Syntax

A module must be followed by the end-of-module and section keywords. The Following the module's keyword, the module's name should be stated. Alternatively, an array of ports may also be declared.

Note:The ports that appear on the set of port statements cannot be changed after they are officially declared.

```
module <name> ([port_list]);

    // Contents of the module
end module

// A module can have an empty port list

module name;
    // Contents of the module
end module
```

The module and end module keywords must contain all variable declarations, functions, tasks, dataflow statements, and cases of lower modules.

Purpose of a Module

A module is a design component that implements a particular set of behavioural traits and is later transformed into a circuit that is digital during synthesis. The module can receive any number of inputs, and it will output data in response.

It enables the reuse of a module to create bigger, more potent modules that utilise sophisticated hardware.

CHAPTER 6

XILINX TOOL

All facets of the design flow are managed by the ISE® Design Suite. You have access to all of the design execution tools and design entry through the Project Navigator interface. Additionally, you get entry to the papers and files linked to the project.

6.1 Project Navigator Interface

As demonstrated in Fig 4.1, the Project Navigator interface by default is divided across four group sub-windows. On the left side of the screen are the Designer documents, Begin, and Library panels, it's that offer accessibility to and representation for a project's source materials in addition to the ability to launch procedures for the present wanted source. The first panel enables speedy project opening in addition to easy access to tutorials, references, and documentation. Below the Projects Navigator, the Console, Issues, and notifications panels appear and provide status messages, issues, and alarms.A multi-document interaction (MDI) panel on the opposite side is the workspace. In the working area, you can view text files, layout accounts, simulation waveforms, and diagrams. Any panel may be expanded, added to Project Navigator, relocated about inside the main Project Navigator window, tiled, layered, or assigned to a particular position. Use the menu commands under View > Panels to unlock or dismiss windows. Go to Layout > Load Usual Structure to return the window structure to its normal state. These windows are explored in more detail in the sections that follow.

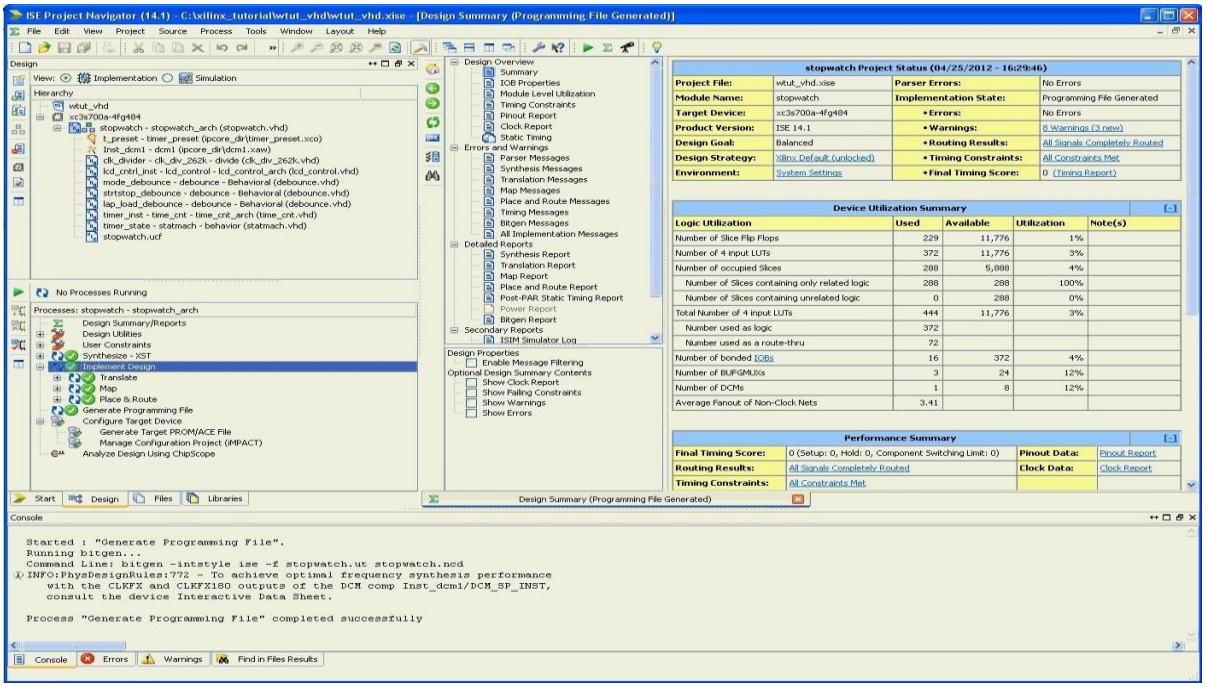


Fig 17: Project Navigator

Design Panel

Entry via the hierarchy, opinionand workflow windows is provided through the design panel.

View Panel

One may see the primary packages associated to the Simulator Design perspective or Implementation within the Hierarchialbar with the help of selection buttons located in the Check out pane, One should choose a test phase against the menu if you choose Simulation.

Hierarchy Pane

The target device, project name, design source files anduser documentsrelevant to the chosen designer perspective are all visible in Hierarchical view. One can only see source files related to chosen Designer perspective, such as installation and modelling, using the overview window near the highest point of the designer screen.

The Hierarchy pane's files each have a corresponding icon. The symbol denotes the kind of document. (HDL file, conceptual, centering on or textual data, for example). See the "Source File Types" item in the ISE Help for a comprehensive list of potential sources types and the icons that go with them. To view the ISE Help, choose Help > Help Topics from Project Navigator.

The picture bears a plus sign (+) at the far left of the caption whenever a document carriesreduced level of organisational structure. The hierarchy can be expanded by clicking the plus sign (+). Double-clicking on a file's name will open it for editing.

Processes Pane

The Sources pane's source type selection and the top-level module in the project determine the context-sensitive adjustments that the Processes pane makes. You might perform the methods essential to illustrate, run and analysethe design from the Processes pane. The following features are accessible through the Processes pane:

- Design Summary/Reports

Entry to messages, design reports and data for the outcome summary. Additionally, message filtering is possible.

- Design Utilities

access to simulation library compilation,command line history viewing,instantiation templates, and symbol generation

- User Constraints

Gives users entry to rectification time and location restrictions.

- Synthesis

Gives users access to synthesis reports, view RTL,check syntax, or technology schematics, and synthesis. The methods that are available depend on synthesis tools one can choose.

- Implement Design

Approach to tools for implementation and analysis after implementation.

- Produce Programming

Document generates approach to bitstream delivery.

- Set up destination unit

Approach to setupinstruments forcoding the device and producing programming files

Dependency management technology is used in the Processes pane. The tools maintain path of which methods turned out to be completed and which is essential to be completed. Visual indicators of position show the current condition of flow. When you choose a method in the path, the software conducts the operations required to reach the specified step. When you run the Implement Design process, for example, programme navigator likewise executes the analysis cycle since execution is related on current production findings.

Choose graphic tools and pick read Command Line record File to see an ongoing log of command line parameters utilisedfor the present task.

Records Module

The task's original files are all displayed in an unorganised column which may be filtered in the Records panel. Folders are able to categorisedutilising any of the rows in the display. You can access and modify a file's information by picking Resource Information within the corresponding context box whenever you right-clicking it.

.Libraries Panel

You may direct HDL libraries and the HDL source files that go with them using the libraries panel. You can make, see, and edit libraries and the sources that go with them.

Console Panel

The Dashboard is used for displaying every typical information via Project Navigator-run software. Notifications about oversights, alerts, and details are given. Inaccuracies are symbolised by a red X adjacent to the message, whereas cautionary statements are represented by a yellow exclamation symbol (!).

.Errors Panel

The Problems panel simply exposes errors. Additional terminals' data are intercepted by the filter used.

.Warnings Panel

The Alerts segment merely displays messages of caution. Other terminals' data are suppressed by the filter used.

Error Navigation to Source

Users can move through the spot of the issue in aexternal HDL document using a algorithm issue or alertnotification within the console, faults, or alerts panel. To do this, click the mouse once on the mistake or warning notice, then choose the option to open origin from the menu when you right click.The cursor advances to the line containing the error as the HDL source file opens.

Error Navigation to Answer Record

Users are able to navigate from a mistake or cautionary message displayed in the Console, Inaccuracies or alert window to pertinent Response Tracks within the Product Assistance and Reference portion of the Xilinx® website. To access the Response Record, hover your mouse the mouse on the error or alert message.All result Records relevant to this message are shown in the open Web browser by default.

Workspace

Design editors, viewers, and analysis tools unlocked in the workspace. They contains the ISE Text Editor, Timing Analyzer, Design Summary/Report Viewer, Constraint Editor, and RTL and Technology Viewers, Schematic Editor.

When activated, additional tools like the PlanAhead TM tool for input/output planning and floor planning, ISim, third-party text editors, XPower Analyzer, and Impact display in independent sessions away from the main Project Navigator environment. Design Summary/Report Viewer.

The Design Overview contains links to each of the notifications and reports produced by the synthesis and implementation tools, as well as an overview of the essential information about the design. A high degree information regarding the assignment are included in the brief, including an overview, a device use brief, outcome information from the Place and Route (PAR) report, limitations, and summary data from each report, along with connections to the respective sections. Details on the surroundings parameters and instrument parameters used during the design implementation may be found by following the link to the System Settings report. This view provides messaging functionality including progressive communication, messaging categorising, and text sorting.

6.2 HDL Based Design:

Overview of HDL-Based Design

The present section demonstrates HDL-based development process utilizing a timer for runners as an example. This tutorial's design serves as an example of numerous hardware, software, and development process principles which you’re able toutilise in your own designs. This design is intended for the Spartan®-3A device, but, unless otherwise stated, each of the ideas and processes are transferable to any Xilinx® device family.

Two cores and HDL components make up the design. Utilizing the software Synplify/Synplify Pro, Precision, or Xilinx Synthesis Technology (XST), you can synthesise the design.

You require Xilinx ISE® Design Suite loaded in order to complete this tutorial.

The programme is presumptively loaded in the standard directory, which is c:xilinxrelease_numberISE_DSISE, according to this guide. Replace your installation path in the subsequent steps if you installed the software somewhere else.

Note: The Xilinx Design Tools: Installation and Licensing Guide (UG798) offers from the Xilinx website and contains comprehensive instructions for installing software.

HDL:

This software works with both VHDL and Verilog designs simultaneously, pointing up discrepancies as necessary. The tutorial requires you to select the HDL language users intend to work through and acquire the language specific documents. A mixed-linguistic design can be synthesised by XST. The mixed language functionality fails to tackle in the tutorial, though.

Starting the ISE Design Suite

Install the ISE Design Suite by clicking twice Project Navigator on your computer's desktop, or go to the beginning > All Programs > Xilinx ISE Design Suite > Xilinx Design Suite 14 > ISE Design Tools > Project Navigator.

Fig 18:Project navigator Desktop

Create a New Project

Utilize the New Project Wizard to begin a new project by doing the following:

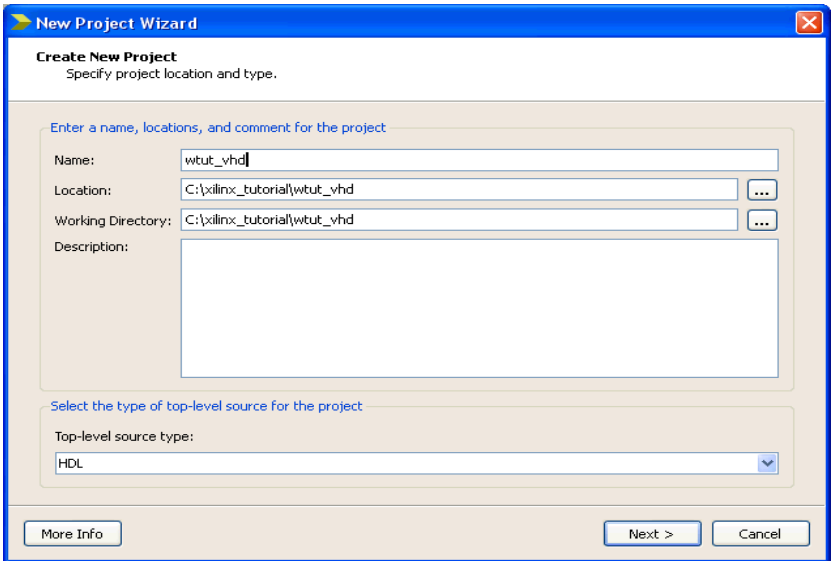


Fig 19: Create New Project Page with the New Project Wizard

File > New Project can be chosen from Project Navigator. The newly created Tasks Wizard emerges.

Browse to the project's installation directory at c:xilinx_tutorialor in the Location form.

1. Enter wtut_vhdorwtut_ver in the Name field.
2. After clicking next making sure HDL is taken as the Top-Level Source Type.
3. The Device Properties screen of the New Project Wizard displays.

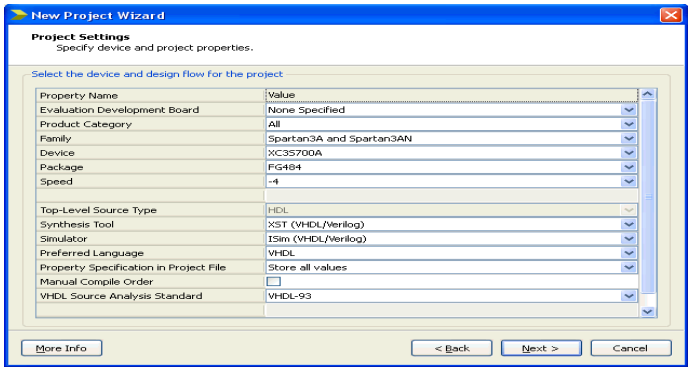


Fig 20: New Project Wizard- Device Properties Page

1. Select the relevant parameters on the Device Properties screen of the New Project Wizard
2. Product Category: **All**
3. Family: **Artix 7**
4. Device: **XC3S700A**
5. Package: **FG484**
6. Speed: **-4**
7. Synthesis Tool: **XST (VHDL/Verilog)**
8. Simulator: **ISim (VHDL/Verilog)**
9. Suggested Language: Verilog or VHDL, if desired. It will determine the language that every process that createHDLfiles will use by default..
10. You can leave other attributes' initial settings..
    11. To finish creating the project, click Proceed and Done.
    12. Termination of the Tutorial
    13. By choosing File > Save All, you can pause the lesson at any time and save your progress.
    14. Design Description 14.
    15. 15. During this course, hierarchical HDL-based design is used, which means that the highest-level layout document is an HDL file which accesses a variety of different smaller-scale expressions. The lower-level macros are either IP modules or HDL modules.
    16. The design is initially a work in progress. You will finish the design throughout the tutorial by creating some of the modules from new and finishing others with pre-existing files. You will replicate the design after it is finished to ensure that it functions as intended.

    The runner's stopwatch has four exterior output buses and five external input buses. An outside signal is used to generate the system clock. The design's input and output signals are listed in the table below

INPUT

The corresponding are incoming messages for the tutorial stopwatch design:

1. strtstop

the stopwatch is started and stoppedThis low-level operational signal performs as the stopwatch's start/stop button for runners.

2.reset

sets the time to 0:00:00 and switches the stopwatch into clocking mode.

3.clk

system clock produced outside.

4.mode

Converts between the timer and clock modes. Only when the clock or timer is not including is this input usable.

5.lap_load

This signal has two processes. When in clocking mode, the 'Lap' display section shows the current clock value. When the timer is not addingwhile in timer mode, it fills the pre-assigned values from ROM to timer display.

OUTPUT

The design's output signals include the following:

1. LCD_e, LCD_RS, and LCD_RW

These results are the signals that operate the Artix 7 demo board's LCD display, which shows the stopwatch times.

1. sf_d[7:0]

gives the LCD display the data values it needs.

Functional Blocks

The following functional units compromise the finished design:

1. clk_div_262k

macro that divides by 262,144 the frequency of a clock. converts a clock at 26.2144 MHz into a 50% duty cycle clock at 100 Hz.

2. The dcm1 Clocking Wizard macro features internal feedback, frequency-controlled output, and duty-cycle adjustment. The Spartan-3A demo board's 50 MHz clock is changed to 26.2144 MHz using the CLKFX_OUT output.

3. debounce This schematic module implements a straightforward debounce circuit for the input signals strtstop, mode, and lap_load.

1. lcd_control

Control module for the LCD display's output and initialization.

2. The HDL statmach status machine module regulates the stopwatch's status.

3. The 64x20 ROM timer_preset CORE GeneratorTM tool. The 64 preset times in this macro range from 0:00 to 9:59 and can be imported into thetimer.

4. time_cnt

Module for a counter that counts in decimal from 0 to 9:59:99. Five 4-bit outputs in this macro represent the stopwatch time digits.

Synthesizing the Design

You've been utilising Xilinx Synthesis Technology (XST) for syntactic verification so far. The design will then be synthesisedusingSynplify/Synplify Pro,either XST or Precision software. The netlistsynthesis software takes the HDL code from the design and evoloves a sustained netlist type (EDIF or NGC) for the Xilinx tools for execution. To generate the netlist, the synthesising tool goes through the series ofbroad phases (which are further broken down by all synthesis tools):

Analyze/Check Syntax

The origin code is checked by the syntax

1. Compiles the HDL code by translating and optimising it through a bunch of elements that the compilersystem can recognise.

2. Map Transforms the fundamental elements of the target technology into the elements obtained during the compilation step.

During the design flowthe simulation tool may be upgraded as desired.To change the synthesis tool, perform the following steps:

1. Select the targeted part in the Project Navigator Design panel's Hierarchy pane.

2. Choose properties of the design from the context menu.

3. In the Properties of the design dialogue box, choose the Synthesis Tool value then choose the synthesis tool that you desire from the available choices. using the pull-down arrow.

Note: If your synthesis tool does not appear in the record, you might not had theinstalled software installed or conFigd in the ISE Design Suite. In the Preferences dialogue box, you can adjust the synthesis tools. Select Edit > Preferences, then expand ISE General and then choose Tools.
The execution material gets wiped out when the proposed flow is changed.. In this tutorial, you have not so far generated any implementation data. If you want to make a backup of a project that contains execution material, Xilinx suggests users to made areplica of the project using File > Copy Project prior to proceeding.

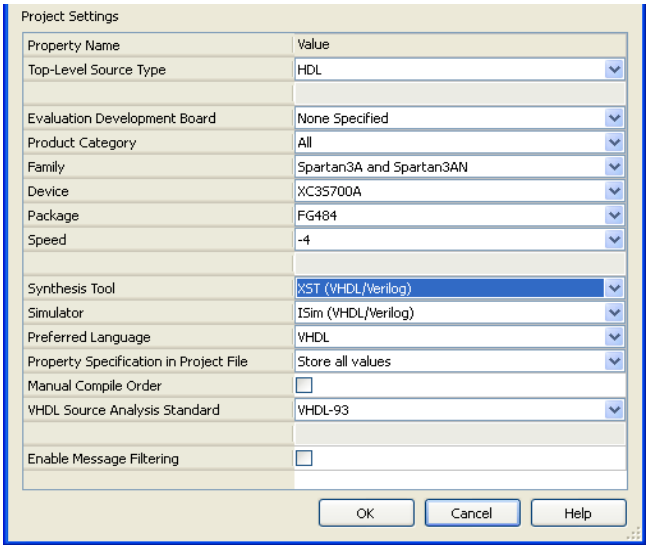


Fig 21: Choosing a Synthesis Tool and XST to Create the Design

After users generated and analysed the layout, the following stage is to synthesise it. The HDL documents are converted onto circuits and optimisedaccording to the intended design during synthesis.

The following processes are available for synthesis with XST:

• View RTL Schematic Displays a schematic representation of your RTL netlist.

• View Technology Schematic Displays a schematic representation of your technology netlist.

• Check Syntax Verifies that the HDL code has been admitted correctly.

• Produce a Post-Synthesis Simulation Model

Based on the synthesis netlist, it generates HDL simulation models.

Entering **Synthesis Options**

Synthesis settings allows to change the behaviour of synthesis tool to optimise it for the necessity of the design. Controlling synthesis to achieve optimisations depending on area or speed is a popular approach. Additional possibilities include managing the optimum fanout of a flip-flops result or choosing the layout's intended rate.

To access synthesis possibilities, perform the following steps:

1. In the Project Navigator Design panel's Hierarchy pane, choose stopwatch.vhd.
(Or stopwatch.v).

2. Hover your mouse on synthesize activity in the Processes window and choose Process Properties.

3. On the Synthesis Options tab, change the Netlist Hierarchy attribute to Reconstuct.
To utilize this property, make sure the Property presentation type is set as high

4.Press the OK button.

6.4 Synthesizing the Design

Now prepared to synthesisethe design. To produce a compliant netlist from the HDL code, perform the following:

1.Select stopwatch.vhd from the Hierarchy window.(or stopwatch.v). click twice on the Synthesize method in the Procedure pane. Using the RTL/Technology Viewer XST, you can create the schematic layout of the HDL code you provided. A schematic representation of the code aids in the analysis of your design by presenting a visual relationship among the many elements detected by XST. The two types of schematic depiction are as follows:

• RTL view

The HDL code is pre-optimized.

• Technology view

View of the HDL design after it has been translation onto the destination technology. Do the following to see a diagrammatic layout of your HDL code:

1. Expand Synthesize in the Processes pane and double-click View RTL Schematic or see Technology Schematic.

2. Select Start with the Explorer Wizard if the Set RTL/Tech Viewer Startup Mode dialogue opens.

3. On the Create Schematic start page, choose the clk_divider and lap_load_debounceelements within the possiblecomponents list, then press the Add icon to add them to the Chosen Components table.

4.Selectthe Create Schematic option.

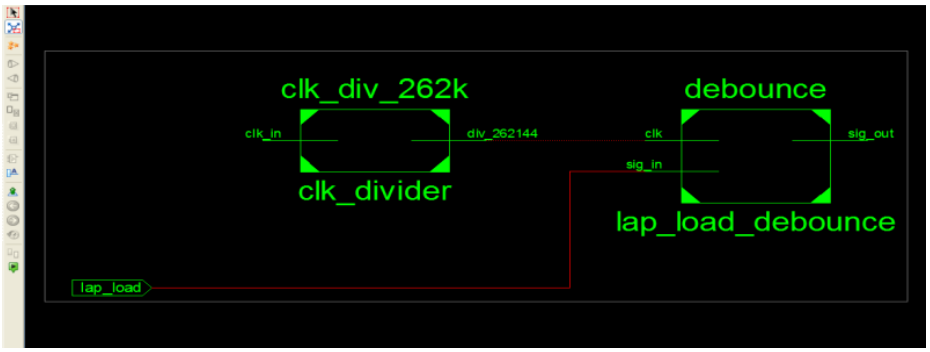


Fig 22:RTL Schematic

You can use the schematic viewer to choose which parts of the outline to exhibit as schematics. Double-click on the icon to keep into the layout and explore the various design features and interaction. Right-click the schematic to see the several functions available in the schematic viewer.

CHAPTER 7

PROPOSED WORK

In this project, we have analyzed 16-bit Vedic multiplier using higher order compressors. Fig 23 depicts that how partial products are grouped to compressor adders for addition.

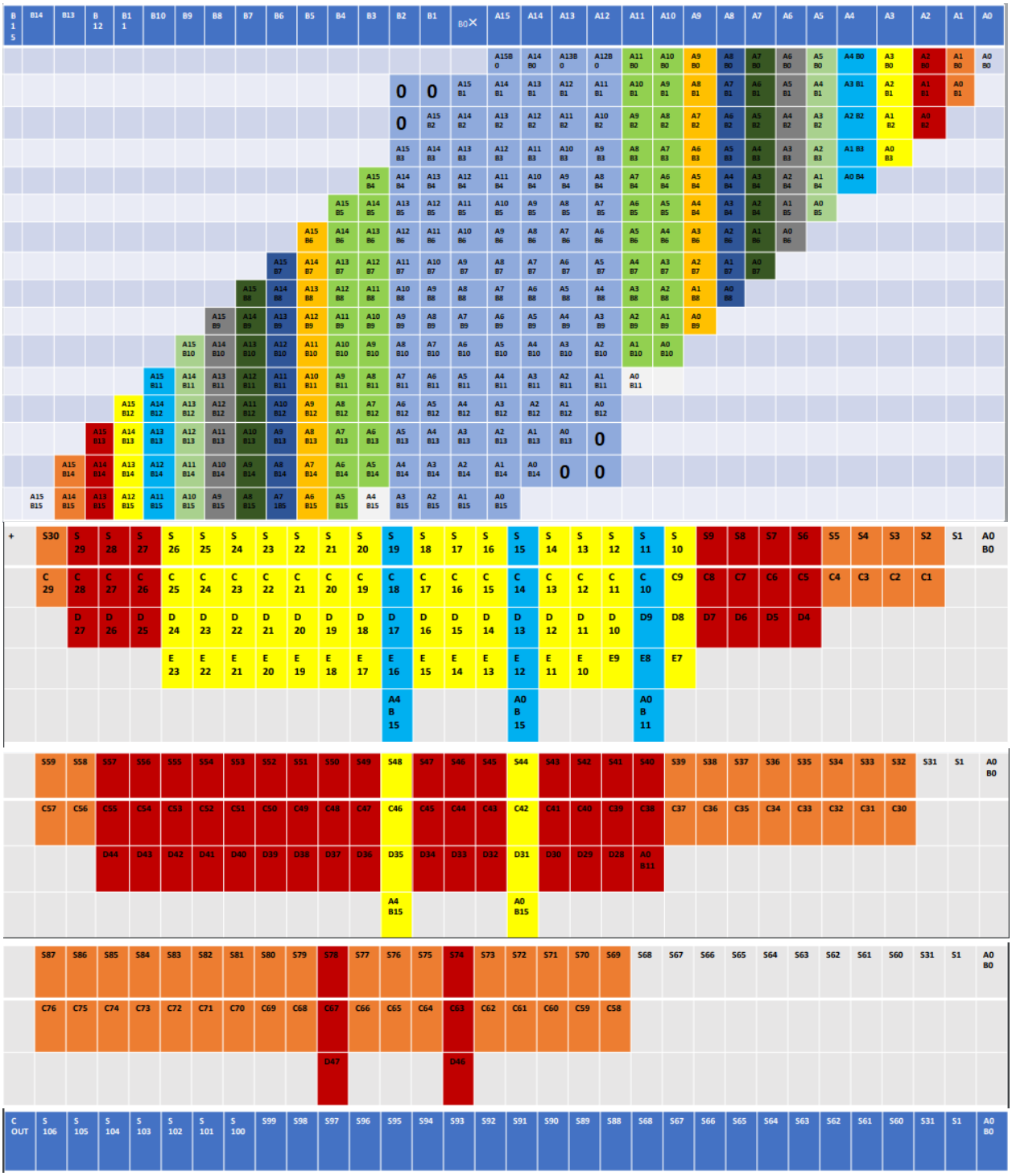


Fig 23: Compressor based 16-bit multiplier

In this approach, 16-bit multiplication is implemented using 4:3, 5:3, 6:3, 7:3, 8:4, 9:4, 10:4, 11:4, and 15:4 compressors, as shown in Fig 1. A half adder and a full adder are accustomed to sum up 5 bits when employing a 5:3 compressor, as an example. The situation with other compressors is similar. The Wallace tree multiplier method is accustomed to resolve the outcome of binary number multiplication. This approach transfers the result to the following stage after adding binary bits.

A full adder will generate carry and sum as its output. The sum is transferred to same level of next stage and carry is transferred to next level of next stage. In a similar manner, a 5:3 compressor will produce three outputs: sum, carry, and auxiliary carry. Now that the sum has been transferred to same level of next stage, the carry has been moved to the next stage of next level, and the auxiliary carry has been moved second next level of the next stage. We also employ 6:3, 7:3, 8:4, 9:4, and 10:4 compressors. Any bits that are still ungrouped after grouping are moved immediately to the next stage.

For instance, if we use an 11:4 compressor to group 12 bits, one bit remains ungrouped. The first four output bits of the 11:4 compressor use the same process as before, and the left bit is passed immediately to the following stage. If there aren't enough bits after grouping to add in a compressor, we add 0 as a bit instead. For instance, two spots are left if we aggregate 13 bits using a 15:4 compressor. So, we give the compressor two "0" bits. The outputs are transferred to the succeeding next step till we achieve the ultimate result, which is depicted in the picture. In a similar manner, we group the bits of the next stage using the same technique. There are three phases to 16-bit multiplication. At the very end, we employ a parallel adder.

CHAPTER 8

RESULTS AND DISCUSSIONS

In this chapter, 16-bit Multiplier using hybrid full adders are simulated using Xilinx Vivado design suite. The performance comparison of proposed work on various FPGA families is observed in terms of area and power.

7.1 SIMULATION RESULTS OF 16 BIT MULTIPLIER USING HYBRID FULL ADDER:

A) Using ARTIX 7 FPGA:

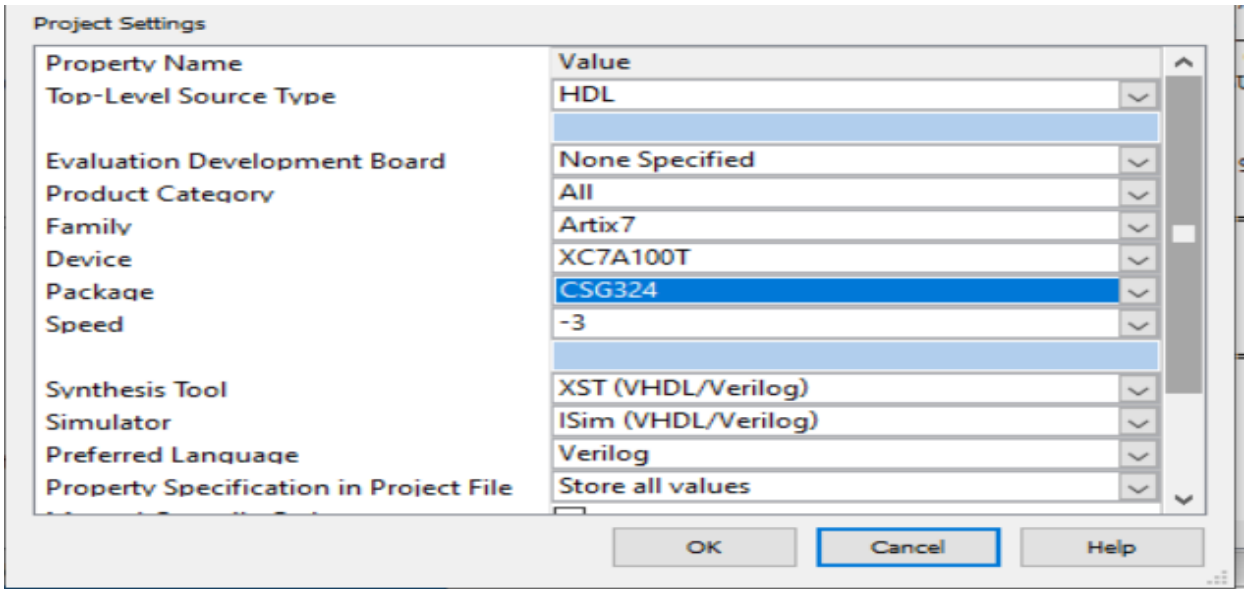


Fig 24: Device and Package selection of Artix7 FPGA

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	126,800	0%
Number of Slice LUTs	254	63,400	1%
Number used as logic	254	63,400	1%
Number using O6 output only	193		
Number using O5 output only	0		
Number using O5 and O6	61		
Number used as ROM	0		
Number used as Memory	0	19,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	129	15,850	1%
Number of LUT Flip Flop pairs used	254		
Number with an unused Flip Flop	254	254	100%
Number with an unused LUT	0	254	0%
Number of fully used LUT-FF pairs	0	254	0%
Number of slice register sites lost to control set restrictions	0	126,800	0%
Number of bonded IOBs	65	210	30%

Fig 25: Area utilization report

```
  Data Path: a<4> to pr<17>
                                Gate     Net
    Cell:in->out      fanout   Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             27   0.001   0.799  a_4_IBUF (a_4_IBUF)
     LUT6:I0->O             3   0.097   0.693  x14/x148/s1 (x14/w6)
     LUT6:I1->O             1   0.097   0.556  x14/x150/x110/s1 (x14/x150/w0)
     LUT6:I2->O             5   0.097   0.398  x14/x150/x111/s1 (x14/A0)
     LUT6:I4->O             4   0.097   0.697  x14/x152/xx2/s1 (c<14>)
     LUT5:I0->O             2   0.097   0.697  x43/x112/Mxor_s_xo<0>1 (c<43>)
     LUT6:I0->O             2   0.097   0.688  x72/Mxor_w1_xo<0>1 (x72/w1)
     LUT6:I1->O             1   0.097   0.279  x99/s1 (pr_16_OBUF)
     OBUF:I->O                  0.000          pr_16_OBUF (pr<16>)
    ----------------------------------------
    Total                       5.489ns (0.680ns logic, 4.809ns route)
                                        (12.4% logic, 87.6% route)

==========================================================================

Cross Clock Domains Report:
--------------------------

==========================================================================


Total REAL time to Xst completion: 17.00 secs
Total CPU time to Xst completion: 16.64 secs
```

Fig 26: Delay Report

A	B	C	D	E	F	G	H
Device			On-Chip	Power (W)	Used	Available	Utilization (%)
Family	Artix7		Logic	0.000	254	63400	0
Part	xc7a100t		Signals	0.000	347	---	---
Package	csg324		IOs	0.000	65	210	31
Temp Grade	Commercial		Leakage	0.082			
Process	Typical		Total	0.082			
Speed Grade	-3						
			Thermal Properties		Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
Environment					4.6	84.6	25.4
Ambient Temp (C)	25.0						
Use custom TJA?	No						
Custom TJA (C/W)	NA						
Airflow (LFM)	250						
Heat Sink	Medium Profile						
Custom TSA (C/W)	NA						
Board Selection	Medium (10"x10")						
# of Board Layers	12 to 15						
Custom TJB (C/W)	NA						
Board Temperature (C)	NA						

Fig 27: Area Utilization Report

7.2 VIRTEX 4

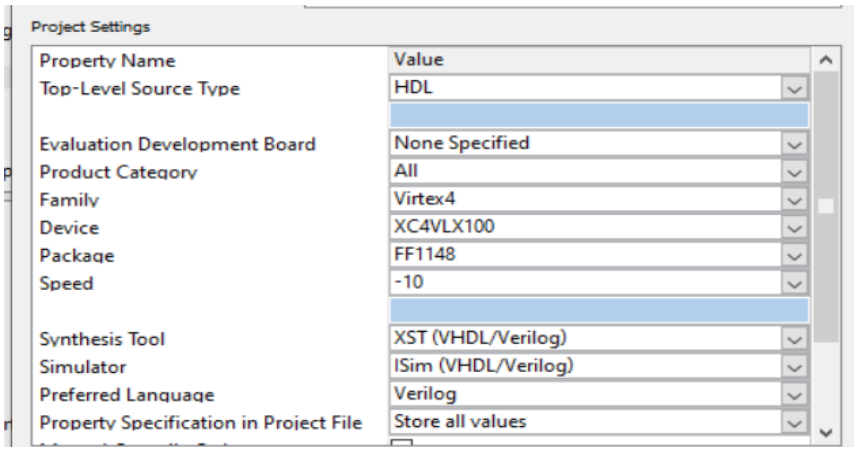


Fig 28: Device and Package selection of Virtex4 FPGA

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of 4 input LUTs	431	98,304	1%
Number of occupied Slices	226	49,152	1%
Number of Slices containing only related logic	226	226	100%
Number of Slices containing unrelated logic	0	226	0%
Total Number of 4 input LUTs	431	98,304	1%
Number of bonded IOBs	65	768	8%
Average Fanout of Non-Clock Nets	3.49		

Fig 29: Area utilization report

```
Data Path: a<14> to pr<20>
                                    Gate     Net
  Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O             26   0.965   1.129  a_14_IBUF (a_14_IBUF)
   LUT2:I0->O             2   0.195   0.705  _and01491 (_and0149)
   LUT4:I1->O             1   0.195   0.741  x17/x150/x111/Mxor_w1_Result21 (x17/N111)
   LUT4:I0->O             1   0.195   0.688  x17/x152/xx2/s11_SW0 (N15)
   LUT4:I1->O             2   0.195   0.540  x17/x152/xx2/s11 (x17/N3)
   LUT4:I3->O             5   0.195   0.711  x17/x152/xx2/s1 (c<17>)
   LUT4:I1->O             3   0.195   0.703  x46/x109/s1 (c<46>)
   LUT4:I1->O             1   0.195   0.585  x75/Mxor_w1_Result1 (x75/w1)
   LUT3:I1->O             3   0.195   0.756  x75/s1 (c<102>)
   LUT4:I0->O             1   0.195   0.000  x103/s12 (x103/s11)
   MUXF5:I0->O            1   0.382   0.360  x103/s1_f5 (pr_20_OBUF)
   OBUF:I->O                  3.957          pr_20_OBUF (pr<20>)
  ----------------------------------------
  Total                      13.977ns (7.059ns logic, 6.918ns route)
                                      (50.5% logic, 49.5% route)

=========================================================================


Total REAL time to Xst completion: 40.00 secs
Total CPU time to Xst completion: 39.53 secs
```

Fig 30: Delay Report

A	B	C	D	E	F	G	H
Device			On-Chip	Power (W)	Used	Available	Utilization (%)
Family	Virtex4		Logic	0.000	431	98304	0
Part	xc4vlx100		Signals	0.000	459	---	---
Package	ff1148		DCMs	0.000	0	12	0
Temp Grade	Commercial		IOs	0.000	65	768	8
Process	Typical		Leakage	0.795			
Speed Grade	-10		Total	0.795			
Environment			Thermal Properties		Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
Ambient Temp (C)	50.0				6.1	80.2	54.8
Use custom TJA?	No						
Custom TJA (C/W)	NA						
Airflow (LFM)	250						
Characterization							
PRODUCTION	v1.0,02-02-08						

Fig 31: Power Report

7.3 Spartan 6

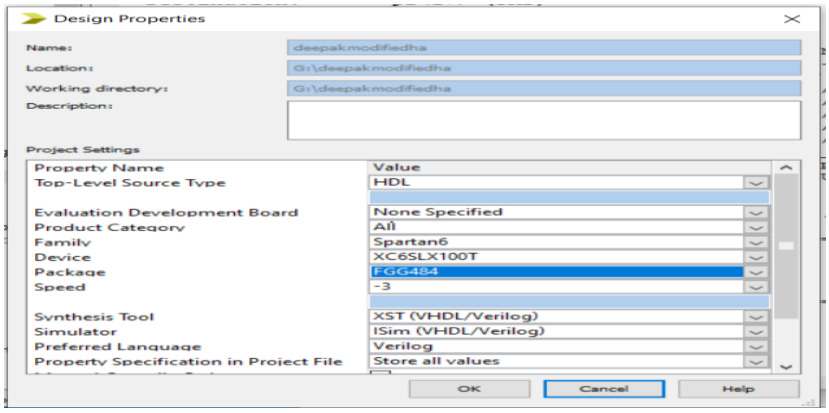


Fig 32: Device and Package selection of Spartan 6 FPGA

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	126,576	0%
Number of Slice LUTs	254	63,288	1%
Number used as logic	254	63,288	1%
Number using O6 output only	193		
Number using O5 output only	0		
Number using O5 and O6	61		
Number used as ROM	0		
Number used as Memory	0	15,616	0%
Number of occupied Slices	94	15,822	1%
Number of MUXCYs used	0	31,644	0%
Number of LUT Flip Flop pairs used	254		
Number with an unused Flip Flop	254	254	100%
Number with an unused LUT	0	254	0%
Number of fully used LUT-FF pairs	0	254	0%
Number of slice register sites lost to control set restrictions	0	126,576	0%
Number of bonded IOBs	65	296	21%

Fig 33: Area utilization report

```
Data Path: a<5> to pr<21>
                                        Gate     Net
   Cell:in->out        fanout    Delay    Delay   Logical Name (Net Name)
   ----------------------------------------  ------------
    IBUF:I->O               31    1.222    1.642   a_5_IBUF (a_5_IBUF)
    LUT6:I0->O               3    0.203    0.995   x18/x148/s1 (x18/w6)
    LUT6:I1->O               1    0.203    0.827   x18/x150/x110/s1 (x18/x150/w0)
    LUT6:I2->O               5    0.203    0.819   x18/x150/x111/s1 (x18/A0)
    LUT6:I4->O               3    0.203    0.995   x18/x152/xx2/s1 (c<18>)
    LUT5:I0->O               2    0.203    0.981   x47/x112/Mxor_s_xo<0>1 (c<47>)
    LUT6:I0->O               2    0.203    0.961   x76/Mxor_w1_xo<0>1 (x76/w1)
    LUT6:I1->O               1    0.203    0.579   x103/s1 (pr_20_OBUF)
    OBUF:I->O                     2.571            pr_20_OBUF (pr<20>)
   ----------------------------------------
   Total                         13.014ns (5.214ns logic, 7.800ns route)
                                          (40.1% logic, 59.9% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

=========================================================================

Total REAL time to Xst completion: 21.00 secs
Total CPU time to Xst completion: 21.29 secs
```

Fig 34: delay Report

\

7.4 VIRTEX 6 LOW POWER:

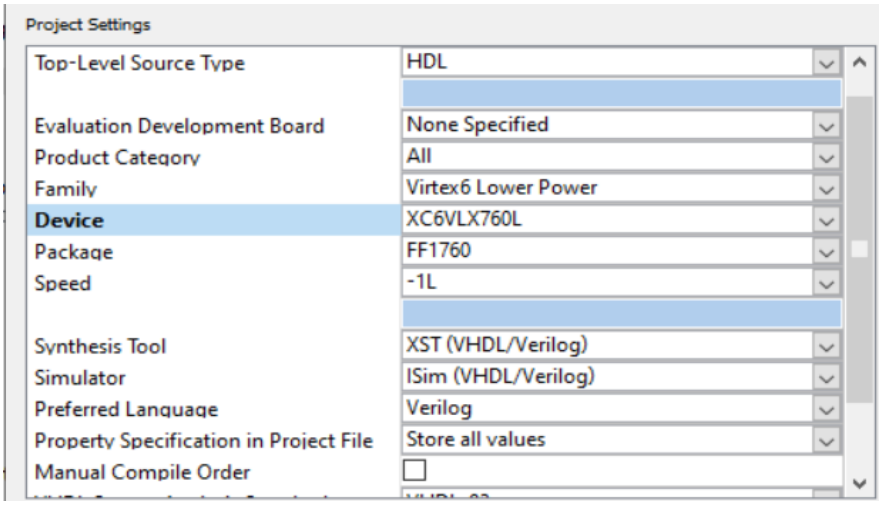


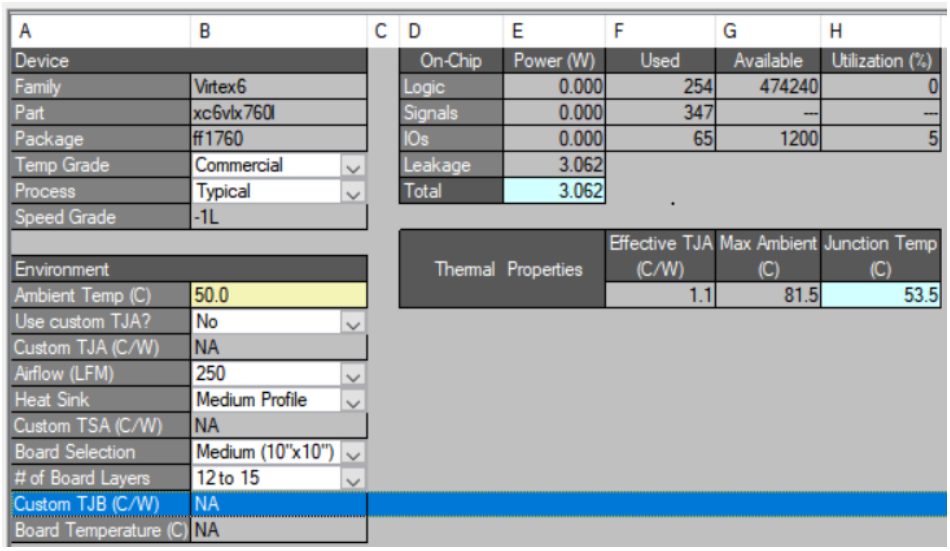
Fig 36: Device and Package selection of Virtex 6 low power FPGA

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	0	948,480	0%
Number of Slice LUTs	254	474,240	1%
Number used as logic	254	474,240	1%
Number using O6 output only	193		
Number using O5 output only	0		
Number using O5 and O6	61		
Number used as ROM	0		
Number used as Memory	0	132,480	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	101	118,560	1%
Number of LUT Flip Flop pairs used	254		
Number with an unused Flip Flop	254	254	100%
Number with an unused LUT	0	254	0%
Number of fully used LUT-FF pairs	0	254	0%
Number of slice register sites lost to control set restrictions	0	948,480	0%
Number of bonded IOBs	65	1,200	5%

Fig 37: Area Utilization Report

```
Data Path: a<4> to pr<17>
                                    Gate     Net
  Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O             27   0.003   0.865  a_4_IBUF (a_4_IBUF)
   LUT6:I0->O             3   0.053   0.726  x14/x148/s1 (x14/w6)
   LUT6:I1->O             1   0.053   0.601  x14/x150/x110/s1 (x14/x150/w0)
   LUT6:I2->O             5   0.053   0.504  x14/x150/x111/s1 (x14/A0)
   LUT6:I4->O             4   0.053   0.732  x14/x152/xx2/s1 (c<14>)
   LUT5:I0->O             2   0.053   0.720  x43/x112/Mxor_s_xo<0>1 (c<43>)
   LUT6:I0->O             2   0.053   0.718  x72/Mxor_w1_xo<0>1 (x72/w1)
   LUT6:I1->O             1   0.053   0.399  x99/s1 (pr_16_OBUF)
   OBUF:I->O                  0.003          pr_16_OBUF (pr<16>)
  ----------------------------------------
  Total                       5.642ns (0.377ns logic, 5.265ns route)
                                      (6.7% logic, 93.3% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

=========================================================================


Total REAL time to Xst completion: 51.00 secs
Total CPU time to Xst completion: 50.71 secs
```

Fig 38: Delay Report



A	B	C	D	E	F	G	H
Device			On-Chip	Power (W)	Used	Available	Utilization (%)
Family	Virtex6		Logic	0.000	254	474240	0
Part	xc6vlx760l		Signals	0.000	347	---	---
Package	ff1760		IOs	0.000	65	1200	5
Temp Grade	Commercial		Leakage	3.062			
Process	Typical		Total	3.062			
Speed Grade	-1L						
			Thermal Properties		Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
Environment					1.1	81.5	53.5
Ambient Temp (C)	50.0						
Use custom TJA?	No						
Custom TJA (C/W)	NA						
Airflow (LFM)	250						
Heat Sink	Medium Profile						
Custom TSA (C/W)	NA						
Board Selection	Medium (10"x10")						
# of Board Layers	12 to 15						
Custom TJB (C/W)	NA						
Board Temperature (C)	NA						

Fig 39: Power Report

Table 1: Manual area utilization report of 16-bit multiplier

Type of compressor	No of compressors
Half adder	34
Full adder	27
4:3	18
5:3	5
6:3	2
7:3	2
8:4	2
9:4	2
10:4	2
11:4	4
15:4	7

Table2: Implementation of 16-bit multiplication on various FPGA families

Performance Metrices	FPGA Family Used			
	Artix-7	Virtex-6	Spartan-6	Virtex-4
No.of Slices	254	254	254	431
No.of Occupied Slices	129	101	99	226
Delay(ns)	5.489	5.642	13.014	13.97
Power(Watts)	0.082	3.062	0.081	0.795

CONCLUSION

In this project, 16-bit multiplier based on higher order compressor is developed and implemented with Artix-7 FPGA. To further improvement in performance, a hybrid full adder is introduced in higher order compressors and the same will be used in 16-bit multiplier. By using Artix-7 FPGA we can observe that there is 57.82%enhancement in delay compared to Spartan 6, 60.7% enhancement in delay and 89.68%enhancementin power compared to Virtex-4. So far as power concern, Spartan 6 and Artix-7 provide better performance compared to Virtex-4 family. From the simulation results it is observed that multiplier based on hybrid full adder allows better results in respect to power and area.

References

[1] Y.Kim, Y.Zhang, and P. Li, ”An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems,” in proc. of International conference on Computer-Aided Design (ICCAD), USA, 2013, pp. 130-137.

[2] D. Baran, M. Aktan, and V.G. Oklobdzija,”Energy Efficient Implementation of Parallel CMOS Multipliers with Improved Compressors,” in proc. ACM/IEEE International Symposium on LowPower Electronics and Design (ISLPED),USA, 2010, pp. 147-152.

[3] S. Veeramachaneni, K. Krishna M, L. Avinash, S. R. Puppala, and M.B. Srinivas, ”Novel Architectures for High-Speed and Low-Power 3-2, 4-2 and 5-2 Compressors,” in proc. Of International Conference on VLSI Design (VLSID),Bangalore,2007, pp.324-329.

[4] R. Menon, and D. Radhakrishnan, ”High performance 5: 2 compressor architectures,” in proc. of IEE - Circuits, Devices and Systems, vol. 153,no. 5, pp. 447-452, Nov 2006.

[5] A. Pishvaie, G. Jaberipur, and A. Jahanian,”High Performance CMOS (4:2) compressors,” Int.journal of electronics, vol. 101, no. 11,1511-1525, Jan. 2014.

[6] O. Kwan, K. Nawka, and E. SwartzlanderJr,” A 16 bit by 16 bit MAC Design Using Fast 5:3 Compressor Cells,” Journal of VLSI Signal Processing, vol. 31, no. 2, 77-89,July 2002.

[7] S. Mehrabi, R.F Mirzaee, S. Zamanzadeh, K. Navi, and O. Hashemipour,”Design, analysis, and implementation of partial product reduction phase by using wide m:3 (4 m 10) compressors,” Int. Journal of High Performance System Arch, vol. 4, no. 4,231-241,Jan. 2013.

[8] A. Dandapat, P.Bose, S. Ghosh, P Sarkar, and D. Mukhopadhyay,(March 2009), ”A 1.2-ns 16 x 16 bit binary multiplier using high speed compressors,” World Academy of Science, Engineering and Technology, vol. 39, 627-632,March 2009.

[9] R. Marimuthu, M. Pradeepkumar, D. Bansal, S. Balamurugan, and P.S Mallick,”Design of high speed and low power 15-4 compressor,” in proc. International Conference on Communication and Signal Processing (ICCSP),Melmaruvathur, 2013, pp. 533-536.

[10] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi”Approximate XOR/XNOR-based adders for inexact computing,” in proc. of IEEE International Conference on Nanotechnology (IEEE - NANO),China,2013,pp. 690 - 693.

[11] H. Jiang, J.Han, and F. Lombardi,”A comparative review and evaluation of approximate adders,” in proc. of ACM Great Lakes Symposium on VLSI(GLSVLSI),New York, 2015 pp.343 - 348.

[12] C. Liu, J. Han, and F. Lombardi,”A Low-Power, HighPerformance Approximate Multiplier with Configurable Partial Error Recovery,” in proc. of International Conference on Design Automation and Test in Europe (TEST).Germany, 2014, pp. 1-4.

[13] Y. Bansal, and C. Madhu,”A novel high-speed approach for 16x16 vedic multiplication with compressor adders,” Computers and Electrical Engineering, vol. 49, 39-49,Jan. 2016.

[14] A. Momeni, J. Han, P. Montuschi, and F. Lombardi,”Design and analysis of approximate compressors for multiplication,” IEEE Trans.On Computers, vol. 64, no. 4,984-994, Apr. 2015.

[15] Valentina Bianchi,Ilaria De Munari, “A modular vedic multiplier architecture for model-based design and deployment on FPGA platforms” Microprocessors and Microsystems vol.76,2020

[16] Juili borkar,Dr.U.M.Gokhale,”Design and simulation of low power and area efficient 16x16 bit hybrid multiplier,”International Journal of Engineering Development and Research,vol.5,2017

[17] Sivanandam K, Kumar P,” Design and Performance Analysis of Reconfigurable Modified Vedic Multiplier with 3-1-1-2 Compressor”, Microprocessors and Microsystems, Volume 65, March 2019, Pages 97-106

[18] Yogita Bansal, Charu Madhu," A novel high-speed approach for 16 × 16 Vedic multiplication with compressor adders", Computers and Electrical Engineering 49, pp. 39–49, 2016

[19] Valentina Bianchi, Ilaria De Munari, "A modular Vedic multiplier architecture for model-based design and deployment on FPGA platforms",Microprocessors and Microsystems76(2020)103106(INCET)B

[20]Rakesh H.M and G.S.Sunitha,"Design and Implementation of Novel 32-Bit MAC Unit for DSP Applications",2020 International Conference for Emerging Technology elgaum, India. Jun 5-7, 2020