# VLSI IMPLEMENTATION OF OPTIMIZED DIRECT FORM & CASCADE FIR FILTER FOR LOW POWER, AREA AND DELAY

**A PROJECT REPORT**

*Submitted in partial fulfilment of the requirement for the award of the Degree of*

**BACHELOR OF TECHNOLOGY**
**IN**
**ELECTRONICS AND COMMUNICATION ENGINEERING**

*by*

**VADLAMUDI SHANMUKH PRANITH (18BEC1245)**
**SUDANAGUNTA NAGA DHEERAJ (18BEC1087)**
**PINJALA SRI HARSHA (18BEC1211)**

*Under the Guidance of*

**Dr. S M SAKTHIVEL**
ASSOCIATE PROFESSOR SENSE

SCHOOL OF ELECTRONICS ENGINEERING
VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI – 600127
*May 2022*

## CERTIFICATE

This is to certify that the Project work titled "**VLSI Implementation of Direct & Cascade Form FIR Filter for Low Power, Area and Delay**" that is being submitted by ***P. Sri Harsha (18BEC1211), V. Shanmukh Pranith (18BEC1245)*** and ***S. Naga Dheeraj(18BEC1087)*** is in partial fulfillment of the requirements for the award of **Bachelor of Technology in Electronics and Communication Engineering**, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

[illegible]

**Guide**

**The Project Report is satisfactory / unsatisfactory**

**Internal Examiner** **External Examiner**

**Approved by**

**Head of the Department**
B. Tech. (ECE)

**DEAN**
School of Electronics Engineering

## ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. S M Sakthivel**, Associate Professor, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work. We are extremely grateful to **Dr. Sivasubramanian A**, Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the school towards our project and for his unstinting support. We express our thanks to our Head of the Department **Dr. Vetrivelan P** for his support throughout the course of this project. We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the course. We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

| | | |
|---|---|---|
| **V. Shanmukh Pranith** | **S. Naga Dheeraj** | **P. Sri Harsha** |
| **(18BEC1245)** | **(18BEC1087)** | **(18BEC1211)** |

## ABSTRACT

This research focuses on the design and implementation of a low power, high speed 16 order FIR filter. For the multiplication of filter coefficient with filter input, different multiplication techniques such as Vedic multiplier, Array multiplier and Booth multiplier are used to optimize filter area, delay, and power. Various adders, including the ripple carry adder, carry select adder, carry select and carry look ahead adders and a few PP adders, are studied for optimal performance, the filter area and delay are optimized by using the add and shift method for multiplication, despite the fact that this increases the filter's power dissipation. Coefficients are represented in canonical signed digit representation, which is more efficient than traditional binary representation, to reduce the complexity of the filter. The MATLAB equiripple method is used to design the finite impulse-response (FIR) filter. We generate a signal from MATLAB and that would be used in Vivado 2021.2 to test the efficiency of all the combinations of FIR filters that we create helping us get a wide understanding and get a conclusion as to it. This is applicable in any industry that makes use of digital signal processing in any form or manner. The most prominent real-life implementations of this topic could be in the fields of bio- medical practices, audio technology, RADAR technology, and noise cancelling applications. This is also crucial for effecting and instantaneous communication in the military bases as they cannot afford to have any delays or lapses in their security systems.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

## LIST OF SYMBOLS AND ABBREVATIONS

| **ABBREVIATION** | **TITLE** |
|---|---|
| bm | BOOTH MULTIPLIER |
| am | ARRAY MULTIPLIER |
| vm | VEDIC MULTIPLIER |
| rca | RIPPLE CARRY ADDER |
| csa | CARRY SELECT ADDER |
| cla | CARRY LOOK-AHEAD ADDER |
| csia | CARRY SKIP ADDER |
| PP | PARALLEL PREFIX |
| KSA | KOGGE STONE ADDER |

# CHAPTER – 1

# INTRODUCTION

In VLSI circuit plans, the reduction of power, area, and delay parameters are expanding as the scope of complexity of utilizations. These days, there are some constant applications needs low power with high throughput than never prior to getting the reach. Here the FIR channel is utilized to plan a proficient advanced signal handling framework. In FIR channel, the right mixes of adders and multipliers are the significant parts to reduce the delay and area. In this project we are going to implement further combinations of filters that would get us a better understanding of the efficiency of FIR filters in direct and cascade forms.

## 1.1 Objectives

The following are the objectives of this project:

- To optimize the architecture of the FIR filters for high-speed and low power with efficient area-delay realization.
- Design FIR filter by coding in the required form in Xilinx using Verilog language.
- Generate the signal and extract the data from MATLAB.
- Make further analysis based on the several parameters like power, area and delay.
- Note down the results of above FIR filter combinations.

## 1.2 Background and Literature Survey

Some of our reference projects which describe the process of designing and implementation of a low power and high speed 16th order FIR filter with optimized filter area, delay and power. They have used different multiplication techniques such as Vedic multiplier, add and shift method and Wallace tree (WT) multiplier are used for the multiplication of filter coefficient with filter input. Various adders such as ripple carry adder, Kogge Stone adder, Brent Kung adder, Ladner Fischer adder and Han Carlson adder are analyzed for optimum performance study. Our project is an extended study of this topic made by considering direct and cascade forms of FIR filter.

**Table 1.1**

| **S.NO** | **TITLE** | **METHOD** | **PARAMETERS** | **INFERENCE** |
|---|---|---|---|---|
| 1. | COMPARATIVE STUDY OF 16-ORDER FIR FILTER DESIGN USING DIFFERENT MULTIPLICATION TECHNIQUES | add and shift method, Wallace tree (WT) multiplier, Vedic multiplier | Area, power and delay | Wallace Tree Multiplier is best among all the multiplication methods. |
| 2. | IMPLEMENTATION OF FIR FILTER FOR LOW POWER AND AREA MINIMIZATION USING SHIFT – ADD METHOD WITHOUT MULTIPLIERS | LC-CSLA-FIR method | Area, power | 180nm -11.56% of area, 7.25% of power 45nm -13.39% of area, 4.61% of power reduced in LC-CSLA-FIR method than conventional methods. |
| 3. | AREA-DELAY-POWER EFFICIENT VLSI ARCHITECTURE OF FIR FILTER FOR PROCESSING SEISMIC SIGNAL | CSD-based MCSE and retiming method | CPD: critical path delay. PDP: Power delay product. ADP: area-delay-product. | Involvement of HUB format, and method used fulfill the objective of a substantial reduction in the hardware complexity and CPD |
| 4. | VLSI IMPLEMENTATION OF FIR FILTER USING DIFFERENT ADDITION AND MULTIPLICATION TECHNIQUES | Dadda Multiplier, Vedic Multiplier, Wallace Multiplier and Modified Carry Select AN-ta | Delay, Area | For area FIR filter using Dadda MCSLA based MAC is efficient architecture and for highspeed FIR filter using Wallace MCSLA based MAC is efficient architecture |
| 5. | COMPARISON OF DIFFERENT TYPES OF MULTIPLIERS WITH RESPECT TO SPEED, AREA AND POWER | Array Multipliers, Baugh-Wooley Multiplier, Booth Multiplier (Radix-2), Modified Booth Multiplier (Radix-4,8,32,64) Braun Multipliers, Wallace Tree Multipliers, Dadda Multipliers | Speed, area, power | Modified Booth Multiplier is more efficient |

| | | | | |
|---|---|---|---|---|
| 6. | HIGH SPEED MODIFIED BOOTH ENCODER MULTIPLIER FOR SIGNED AND UNSIGNED NUMBERS | Modified Booth Encoding (MBE) multiplier and the Baugh-Wooley multiplier | Speed | The speed of the adder circuit and the number of partial products generated is the final speed of the multiplier circuit |
| 7. | VLSI IMPLEMENTATION OF PIPELINED FIR FILTER | non- pipelined and pipelined | Delay, power | The delay efficiency is 82.9 % using Radix-4 and 81.97 % using Radix-8 towards pipelined compare to non-pipelined technique |
| 8. | DESIGN AND IMPLEMENTATION OF FIR FILTER USING EFFICIENT MAC | Vedic-SQRT-CSLA, Hierarchy-CSLALOT, Vedic-CSLALOT, Hierarchy-SQRT-CSLA | Delay, Area | An area efficient MAC based FIR Filter is Vedic-CSLALOT MAC |
| 9. | DESIGN OF A HIGH SPEED AND AREA EFFICIENT NOVEL ADDER FOR AES APPLICATIONS | --------- | Delay | a novel technique is developed for addition which is implemented at gate level to decrease the time delay and reduces the area. |
| 10. | VLSI IMPLEMENTATION AND PERFORMANCE EVALUATION OF LOW PASS CASCADE & LINEAR PHASE FIR FILTER | Normal VHDL | Delay, power, area | as compared to Cascade Filter The product of delay and power of Linear Phase filter is reduced by 42.79 percent. |
| 11. | EFFICIENT DESIGN OF DIFFERENT FORMS OF FIR FILTER | Frequency Sampling method Parks –Mc Cleelen method, Window design method | Delay, Area | Considering all parameters either direct form or transposed form can be utilized in any DSP application |
| 12. | AREA–DELAY–POWER EFFICIENT CARRY-SELECT ADDER | CSLA | Delay, Area, Power, ADP, Energy, EADP EE | SQRT-CSLA design produced 44 percent more ADP and utilize 42 percent more energy than the proposed SQRT-CSLA on average for different bit widths. |

## 1.3 Need for Efficient FIR filters.

We need efficient FIR filters to implement audio or noise cancelling technology which can be useful in various fields for faster and clearer communication over distances. They are also crucial in the bio-medical industry where signals collected from patients for several of their diagnostics that could help them arrive at more precise conclusions faster.

## 1.4 Organization of the Report

The remaining chapters of the project report are described as follows:

- **Chapter 2: FIR Filters**
- **Chapter 3: Implementation of Direct and Cascade Form**
- **Chapter 4: Results and Discussion**
- **Chapter 5: Conclusion and Future work**
- **Chapter 6: References**

# CHAPTER 2

# FIR FILTERS

This chapter describes methodology and filter designing.

## 2.1 Methodology:

The idea of this project is to implement FIR Filters. Any frequency response can be implanted by a Finite Impulse Response Filter (FIR). A FIR Filter is typically executed by utilizing a progression of multiple delays, adders and multipliers to generate the Filter output.

**FIR advantages**:

FIR filter requires no feedback which justifies the term Finite impulse response. This implicates that no errors wouldn't be compounded in further procedure. A similar relative errors happens with every computation. This likewise simplifies execution. And they are stable in nature, since the output will be a sum of a finite no.of finite multiples of the input values, so is always smaller or equal than/to sigma{b(i)} times that of the biggest input.

### 2.1.1 Low Pass FIR Filter:

Low Pass Filters are used to filter out the noise from a circuit.A high frequency signal occurring randomly in the signal. This high frequency noise is removed when passed through the and a noiseless sound is produced and implementing a low pass filter with finite impulse response.

### 2.1.2 Direct form:

Higher order FIR Filters could be realized in the Direct Form as FIR filter is a digital filter with digital inputs. The impulse response of FIR filter is of finite duration. As the name suggests, these are Finite filters, means no need of feedback, thus non-recursive. Output of this FIR Filter is defined as:

$Y[n] = X[n] * H[n]$

In the above equation, x(n) is the output sequence, h(n) represents the coefficients of filter

$H(z) = (c_0 + c_1z^{-1} + c_2z^{-2} + c_3z^{-3}+c_4z^{-4} + ... + c_{12}z^{-12})$.

The general block diagram of a FIR Filter in direct form is depicted in the fig-2.1

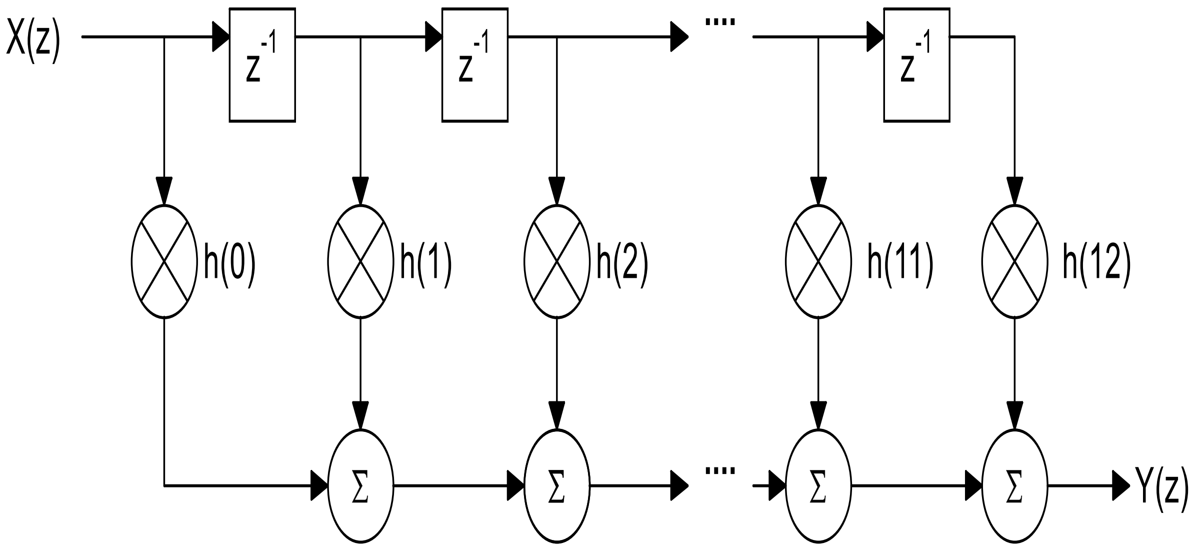


**Fig- 2.1**

## 2.1.3 Cascade:

Higher order FIR filter can be acknowledged in the cascaded form. In the cascade form, the higher order transfer function is acknowledged by cascading lower order FIR sections where each part acknowledges either a first order or second order transfer function.

$H(z) = (c_0 + c_1z^{-1} + c_2z^{-2})*( c_1z^{-1} + c_2z^{-2} + c_3z^{-3})*( c_1z^{-1} + c_2z^{-2} + c_3z^{-3})*( c_1z^{-1} + c_2z^{-2} + c_3z^{-3}+c_4z^{-4})$.

The general form of a cascade form FIR Filter is in the fig-2.2

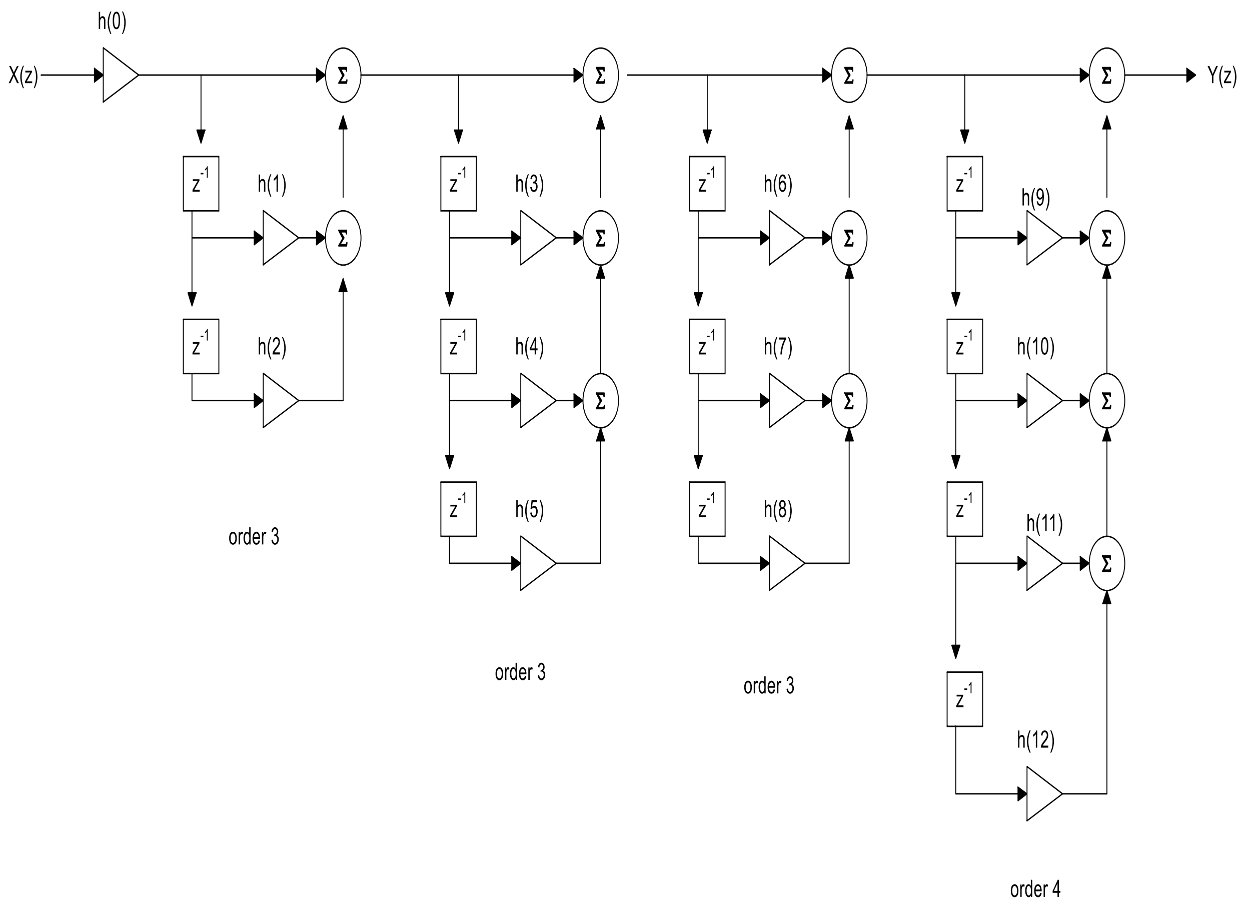


**Fig-2.2**

## 2.2 Multipliers:

### 2.2.1 Vedic Multiplier

Multipliers are an important component in designing digital systems. Several algorithms for implementing fast multipliers have been reported in the literature. The VEDIC algorithm for multiplication is another option for implementing effective multipliers. This task describes the VEDIC multiplier. Multiplication can be implemented in 3 ways in VEDIC maths alone. Only one of the three is a general method that applies to all cases, and the other two are for special cases. The key algo for VEDIC multiplication is Urdhva Triyakbhyamm. This is a common method for multiplication that can be applied for all multiplications. It very exactly means vertical as well as horizontal. Two operands can be multiplied using the VEDIC multiplier by multiplying them vertically and horizontally and all the results must be added. VEDIC can be multiplication is a effective alternate choice for other known popular multiplication algorithms. VEDIC multipliers reduce hardware and latency compared to other algorithms. A diagram representing the working process of a vedic multiplier is shown in the below fig-2.3

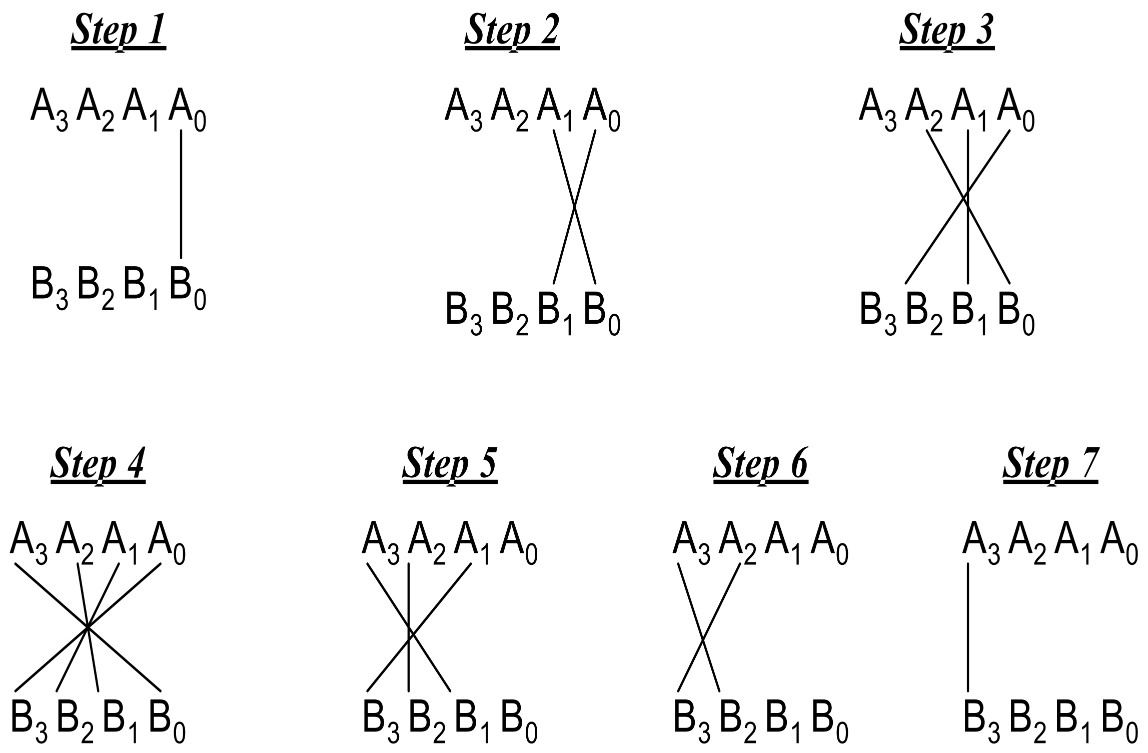


**Fig-2.3**

### 2.2.2 Array Multiplier

An Array Multiplier is digital combination circuit used to multiply two numbers in binary form using a bunch of full and half adders. This array then, is used to add various related product terms at about the same time. The AND gate array is used before the adder array to form the various product terms. Examining all the bits of multipliers one after another to form a partial product is a sequential operation that requires a series of addition and shift microoperations. Multiplication of two binary numbers can be performed in microoperations using combinatorial circuits that form product bits at once. This is an easy way to multiply two numbers, as the signal only takes time to propagate through the gates that make up the multiplication array. The below given fig - 2.4 is the diagram of an array multiplier.

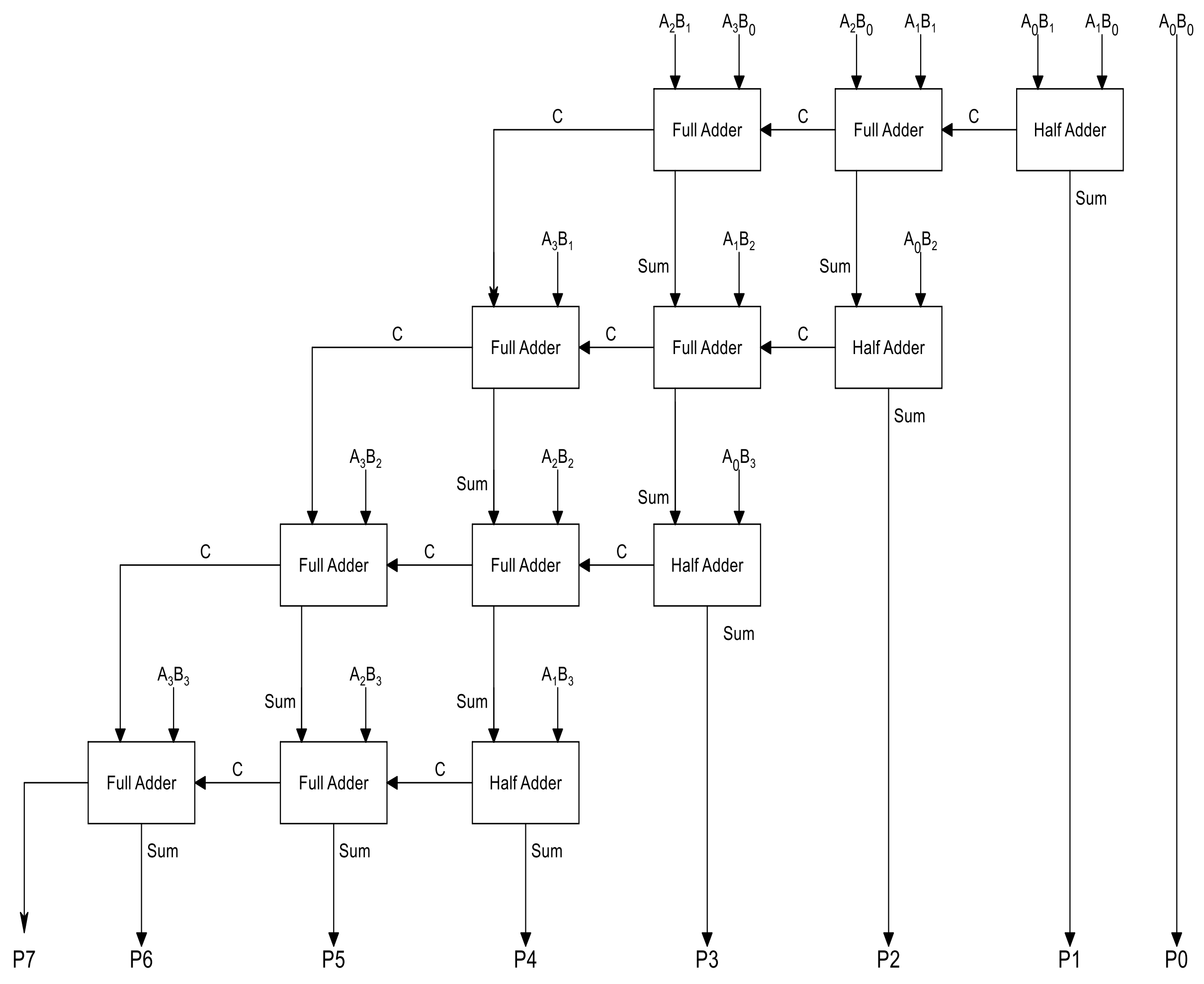


**Fig-2.4**

### 2.2.3 Booth Multiplier

The multiplicand and multiplier are set in the M and Q enlists individually. A 1 cycle register is put legitimately to one side of the LSB, i.e Q0 of Q register. This is indicated by Q-1. An and Q-1 are at first set to 0. Control rationale really takes a look at the two pieces Q0 and Q-1. In the event that the 2 pieces are same (00 or 11) each of the bits of A, Q, Q-1 are moved 1 digit to right side.On the off chance that they are not something similar and in the event that the mix is 10, the multiplicand is deducted from An and on the off chance that the blend is 01, the multiplicand is added with A. In both the cases results are put away in A, and after the expansion or deduction activity, A, Q, Q-1 are correct moved. The moving is the number juggling right shift activity where the left most piece specifically, An-1 isn't just moved into An-2 yet additionally stays in An-1. This is to safeguard the indication of the number in An and Q. The consequence of the augmentation will show up in the An and Q. The block diagram of booth multiplier is shown below in fig-2.5

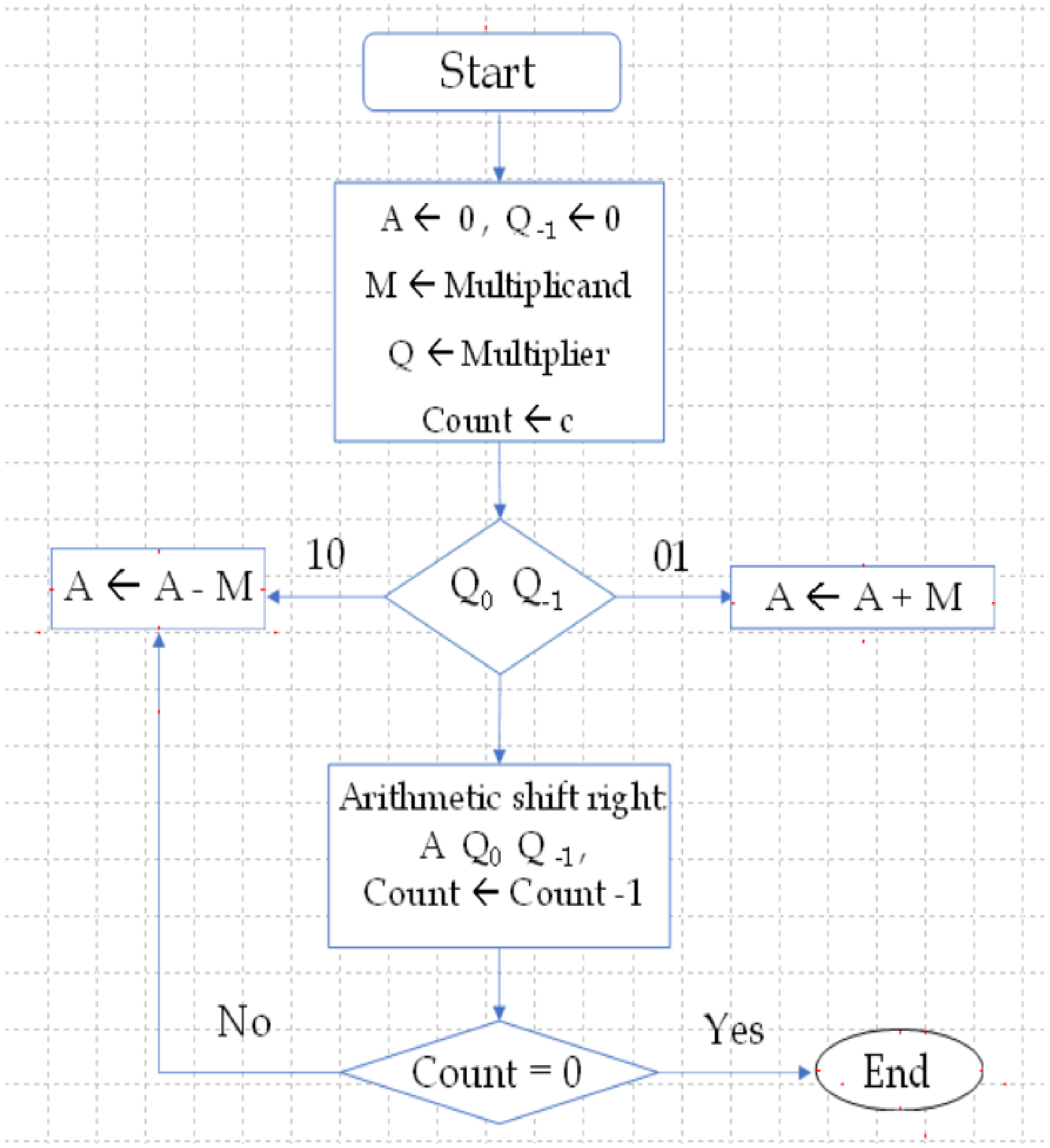


**Fig-2.5**

## 2.3 ADDERS

### 2.3.1 CARRY LOOK AHEAD ADDER:

The carry look-ahead adder, also known as the fast adder, increases the speed required to determine the carry bit. We know that computers perform their activities by arithmetic operations such as division, addition, multiplication, and subtraction. Therefore, division repeats subtraction, and multiplication repeats addition. A adder circuit is required to perform these repeat functions. These are half adders, full adders, and Carry Look-Ahead adders. The Carry Look-Ahead adder definition is a faster circuit for performing binary addition using the concepts of carry generation and carry propagation. CLA is called the successor to the Ripple Carry Adder. The CLA circuit minimizes propagation delay time by implementing a complex circuit. The diagram of Carry Look Ahead adder is depicted below in the fig-2.6

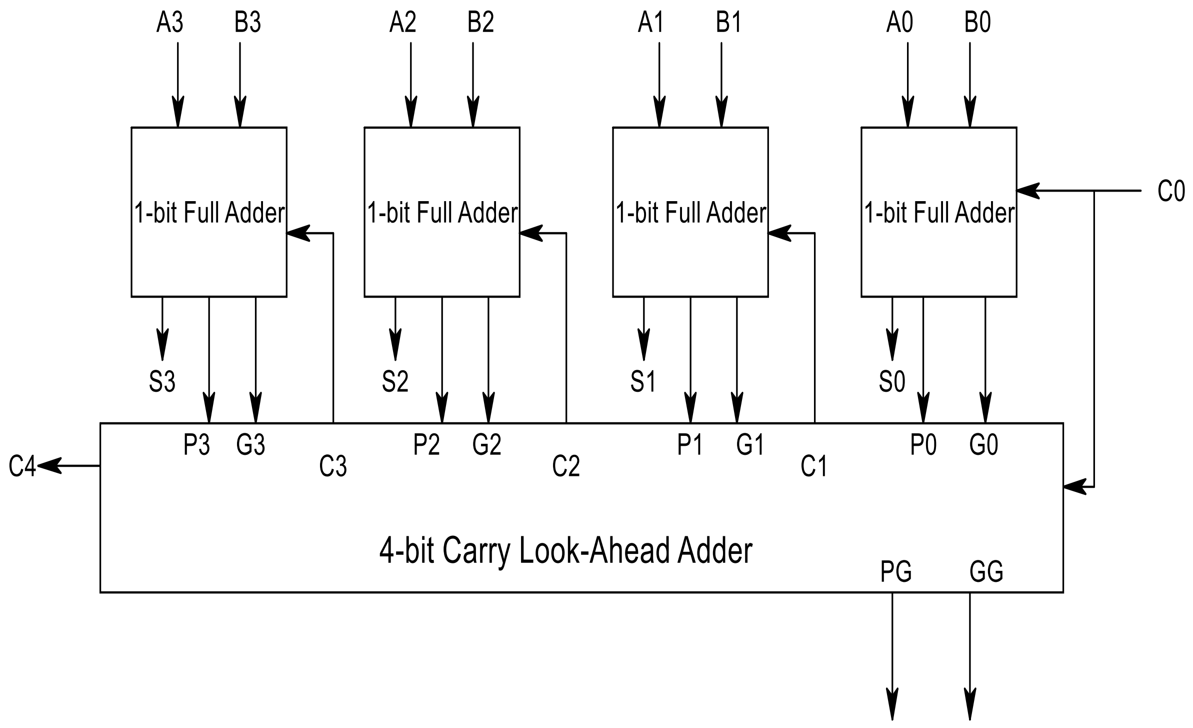


**Fig-2.6**

### 2.3.2 RIPPLE CARRY ADDER

Several FA circuits could be cascaded all in parallel in order to add all the N bits. For N-bit parallel adders, the same number of FA circuits are needed. A ripple carry adder is a logical circuit in which the output carry of each one of the full adder is an input for the next FA. It is called a ripple carry adder because each carry bit is rippled to the next stage. In a Ripple Carry adder, both sum and carry bits of Half Adder stages are not valid till a $C_{in}$ for that particular stage happens.

The reason is the propagation delay in the logic circuit. The Propagation delay is the time taken starting from the sending input to the generation of the respective output. If the input given is "0", the output is "1" and also works the other way around. After applying the binary "1" as an input for NOT gate, the time taken for the output of NOT gate to reach "0" is the propagation delay in this. The same way, the propagation delay of carry is the elapsed time from the assertion of the carry-in signal to the occurrence of the Cout. The sample diagram of a ripple carry adder is depicted by the below fig-2.7.

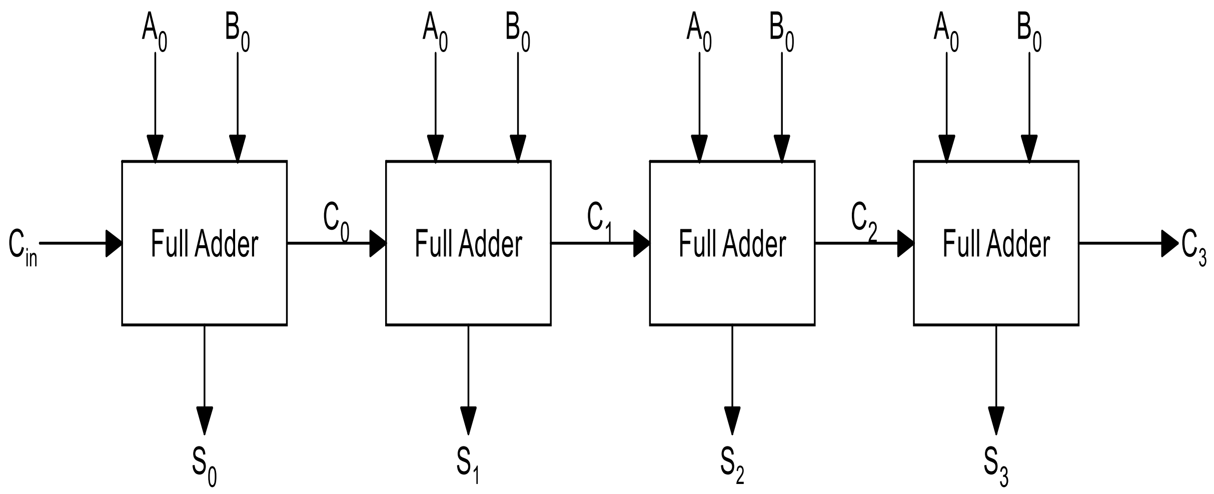


**Fig-2.7**

## 2.3.3 CARRY SELECT ADDER:

A Carry Select Adder generally made with a multiplexer and two RCAs. Adding two n-bit numbers with a carry select adder is finished with two RC adders, so as to do the calculation two times, just only once for the idea of the carry-in to be zero and the other assumption will automatically be 1. Once the calculation of the two results is completed, the proper sum, yet because the correct carryout, will then be calculated along with the mux once the right carry-in would be realized.

Every carry select block is allowed to have the same (uniform case) or different number of bits. In the uniform case, the best delay condition takes place for a block size of sqrt(n).In case of variable block size, a delay is required from the adder inputs A and also B to a carry-out which is equal to the chain of multiplexers supplied to it so that the Cout is calculated with in the required time. The O (√n) delay is deduced from the unit size where the optimal no.of full adder elements per one block is same as the sqrt(no.of bits added).The reason for this is having the equal no.of MUX delays. The diagram of a Carry Select Adder is shown in the below fig-2.8

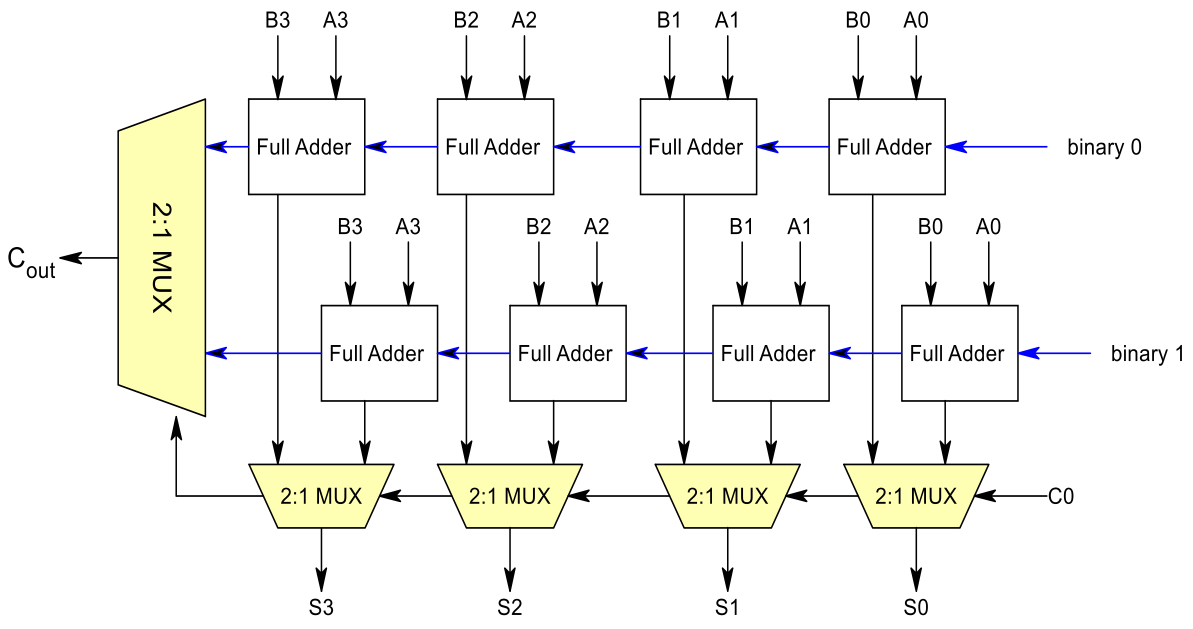


**Fig-2.8**

### 2.3.4 CARRY SKIP ADDER:

The constituents of an n-bit-carry skip adder n-input AND gate, and a multiplexer. All the propagation bits provided by all the carry ripple blocks are connected to an n-input AND gate below. The resultant bit of this AND gate is used as the selector to decide whether the last carry bit $C_n$ or carry-in $C_0$ to the carry-out signal $C_{out}$.

This lets the carry bits of all the blocks to "skip" between blocks if the cluster propagation signal is ready to Logic 1(in distinction with the ripple carry chain containing carry), therefore via the vital path. The latency of the additional adder is greatly reduced. Loop equally of the adder. the quantity of AND circuit inputs is that the same because the dimension of the adder. this can be impractical and incurs further delay, as AND gates have to be compelled to be designed as a tree if they're wide. If the add logic is that the same depth as that of the n-input AND circuit and electronic device, an affordable dimension will be obtained. The below fig-2.9 depicts the diagram of a carry-skip adder.

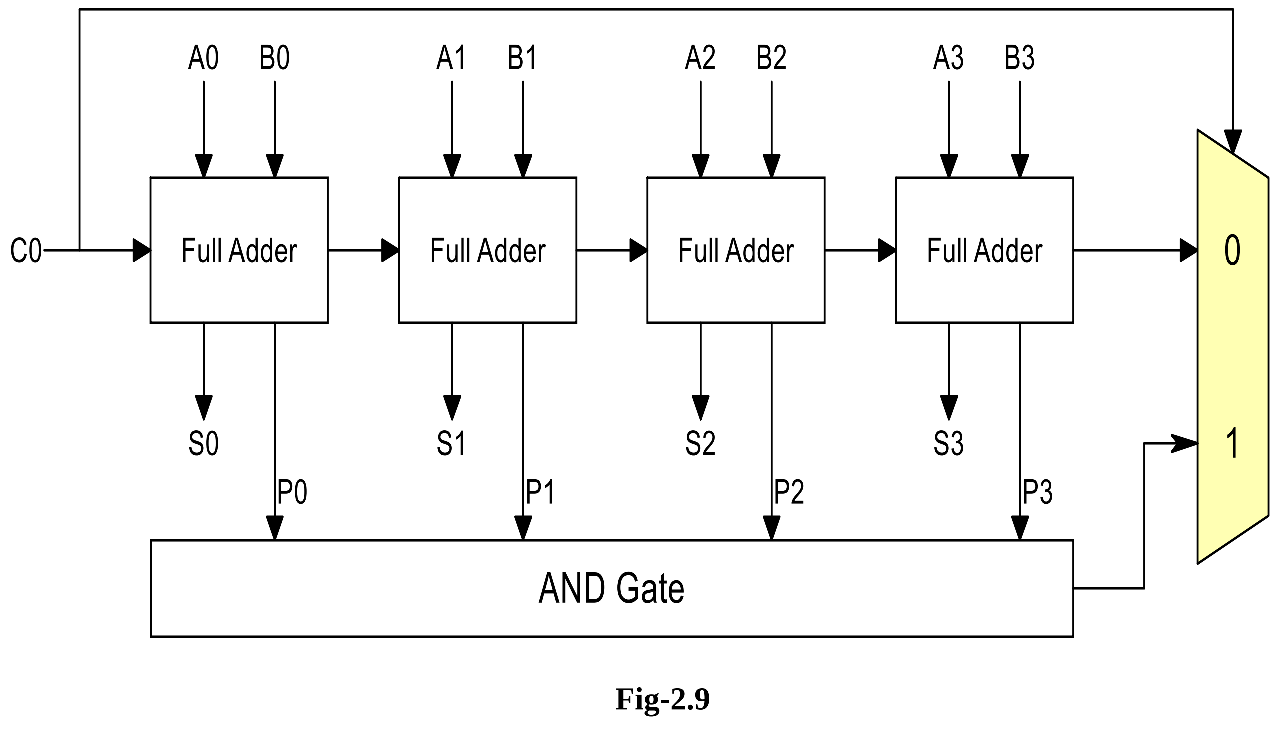


**Fig-2.9**

### 2.3.5 KOGGE STONE ADDER

The KoggeStone adder is a PP extention of Carry Look-Ahead adder. The carry signals are produced in O(log$_2$N) time, and it can be arguably considered the fastest adder design in the industry. It is the most common choice for high performing adders for practical industry use. M. Kogge and Harold S have introduced the Kogge Stone adder. In Koggestone adder, large area is the trade off for fast carry generation by parallel processing. The tree designs of carry propagate will produce signals in 8bit KSA.

The network of Carry generation is the main part in tree adders and Black cell, Grey cell and Buffer are its most important constituents. Black cells are used for processing of propagate and generate signals. Gray cells are used for generated calculations. Post-processing stage requires the signal to calculate the sum. Buffer used to compensate the load effect. The block diagram of a Kogge-stone adder is shown in the fig-2.10

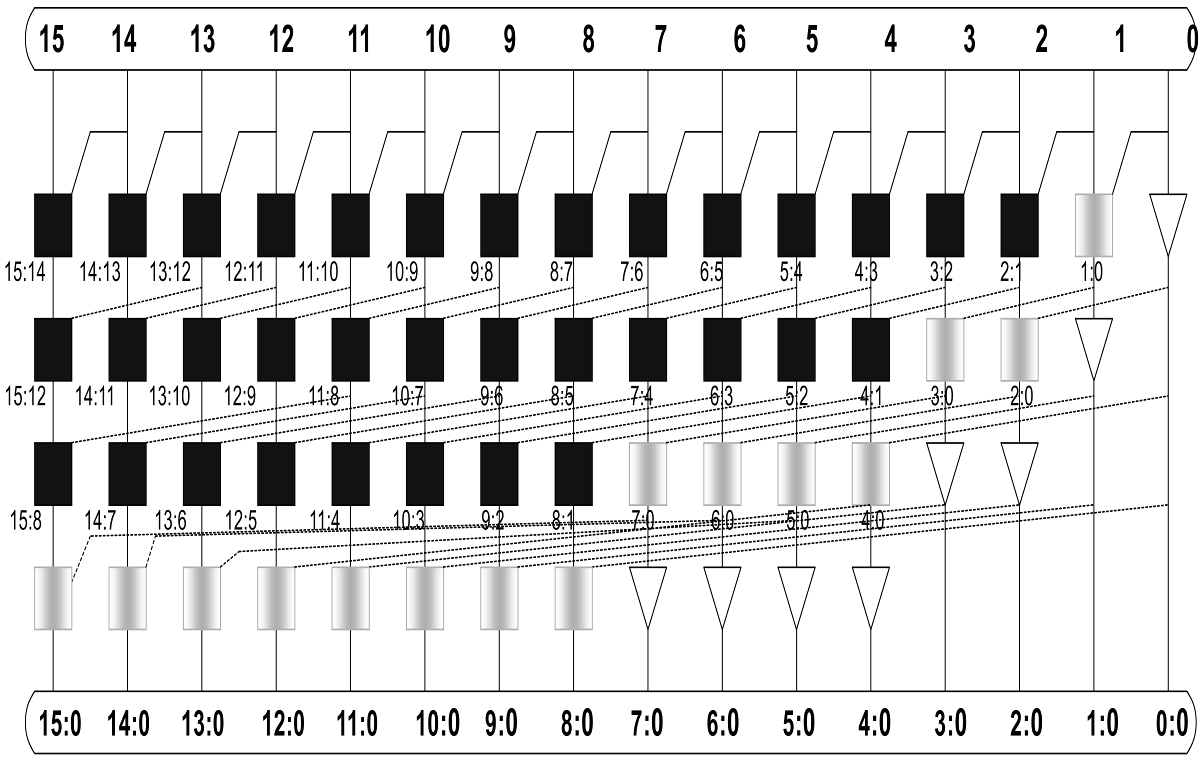


**Fig-2.10**

## 2.3.6 BRENT KUNG ADDER

The Brent Kung adder is a carry-lookahead adder with a PP adder (CLA). In comparison to the Kogge–Stone adder, Richard Peirce Brent and Hsiang Te Kung had proposed it in the year 1982. It increased constancy to structure of adder and decreases the wiring complexity, resulting in higher efficiency and lower chip area required for its implementation (KSA). It's also remarkably faster than RCA. Brent–Kung adders have higher performance due to their tree structure of carry propagation, which also leads to lower power consumption which enables the carry signal to go through less stages, resulting in lower switching of transistors. The reduced quantity of wire and also fan out also helps it consume lower power compared to CLA adder.

A well-known structure with a sparse network is the Brent-Kung prefix tree. As f = 0, the fan out is among the smallest. The same is true for the wire tracks where t = 0. The additional Level 1 logic levels come at a cost. Figure 6 depicts a 16-bit example. The critical path has been represented in the figure by a thick grey line owing to the higher no.of logic levels, the Brent-Kung adder is the most difficult to construct. The block diagram representing the Brent Kung adder is given in the below fig-2.11

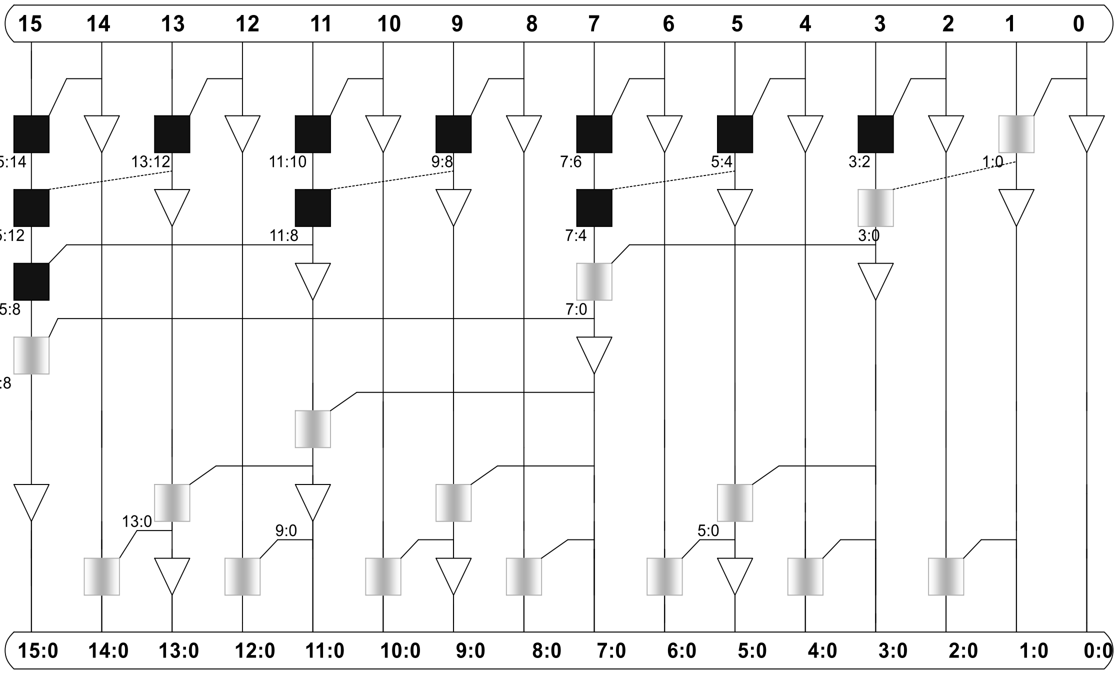


**Fig-2.11**

## 2.3.7 HAN CARLSON ADDER

Several parameters like no. of black cells, logic levels and fan out are all well-balanced in the Han-Carlson adder. As a result, the Han Carlson adder can match the Kogge-Stone adder's speed while consuming less power and covering less space. Executing a imaginary Han-Carlson adder in this way is amusing. It can be said that error detection is reasonably simplified and can be significantly fast when compared to Kogge-Stone.

The concept of the Han Carlson adder is similar to Kogge-Stone structure in that it has a highest fan out of 2. The main difference lies in the usage of lesser wire tracks and cells than the Kogge Stone. The Han-Carlson prefix tree can be thought of as a more sparse version of the Kogge-Stone prefix tree. In fact, the fan out is the same for all logic levels. The pseudo-code for the structure of Kogge Stone adder is easily adjustable to create a Han-Carlson prefix tree. The main difference is that the Han Carlson prefix adder places cells all bits in every logic level, and the final logic-level accounts for the missing carries. The block diagram of Han Carlson adder is shown in the fig-2.12

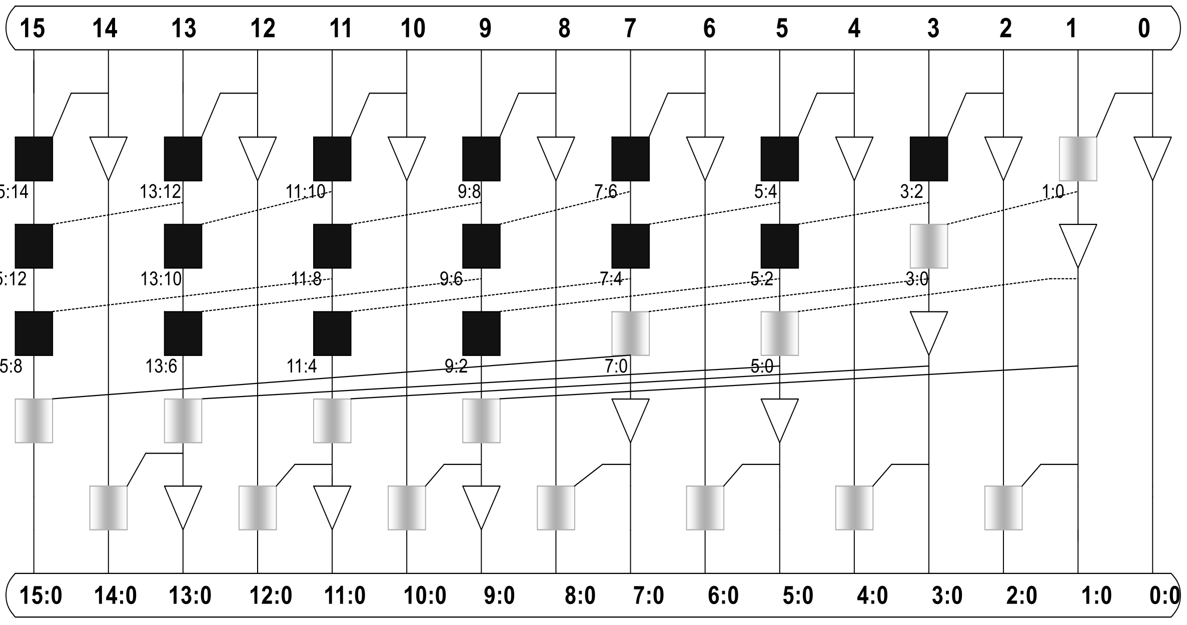


Fig-2.12

### 2.3.8 HARRIS ADDER

The idea from harris about prefix tree is to try to balance the wires, fanout and logic levels. Harris proposed a cube structure for prefix trees. If we consider the harris tree as cube then brent kung at the logic level extreme , sklansky adder standing in at the fan out extreme and kogge stone at the wire path extreme.

The balanced PP adder structure is close to centroid of cube.

(1) Logic levels= $\log_2 N$

(2) max. fan-out=3

(3) wire track=n/6

### 2.3.9 KNOWLES ADDER

In terms of speed, parallel-prefix tree adders outperform other adders because of its low $O(\log_2 N)$ complexity delay (Kogge stone) through out its carry path. This has similarities with kogge stone adder, but the complexity is halved in the final stage. However, it doubles the loading.

The Kogge-Stone adder has the simplest design and one of the fastest possible critical paths of any tree adder. One of the major disadvantage of execution of Kogge-Stone adder is its huge requirements of area and a bit more complicated routing. The block diagram of Knowles adder is shown below in fig-2.13

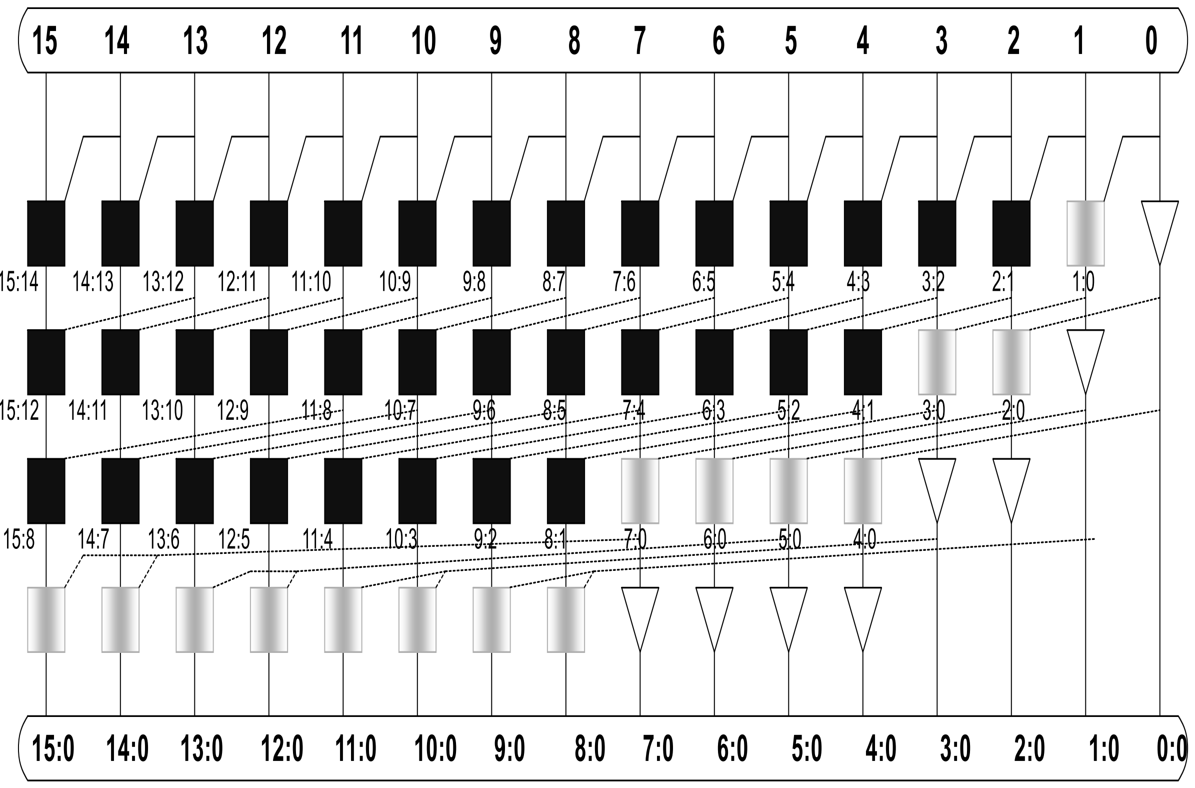


**Fig-2.13**

### 2.3.10 LADNER FISCHER ADDER

Ladner–Fischer adder is one of the many PP adders. These PP adders are used to accelerate arithmetic operations. Ripple carry adders can add some carry propagation delay, which can be removed with these adders. Ladner-Fischer adder is a PP adder that executes addition. It is divided into three stages: (1) pre-processing, (2) generation carry, and (3)final processing stage. The efficient design of this adder works in high speed and occupies less memory for the functioning of 16-bit addition. The block diagram of Ladner Fischer Adder is shown in the Fig - 2.14 and Fig - 2.15

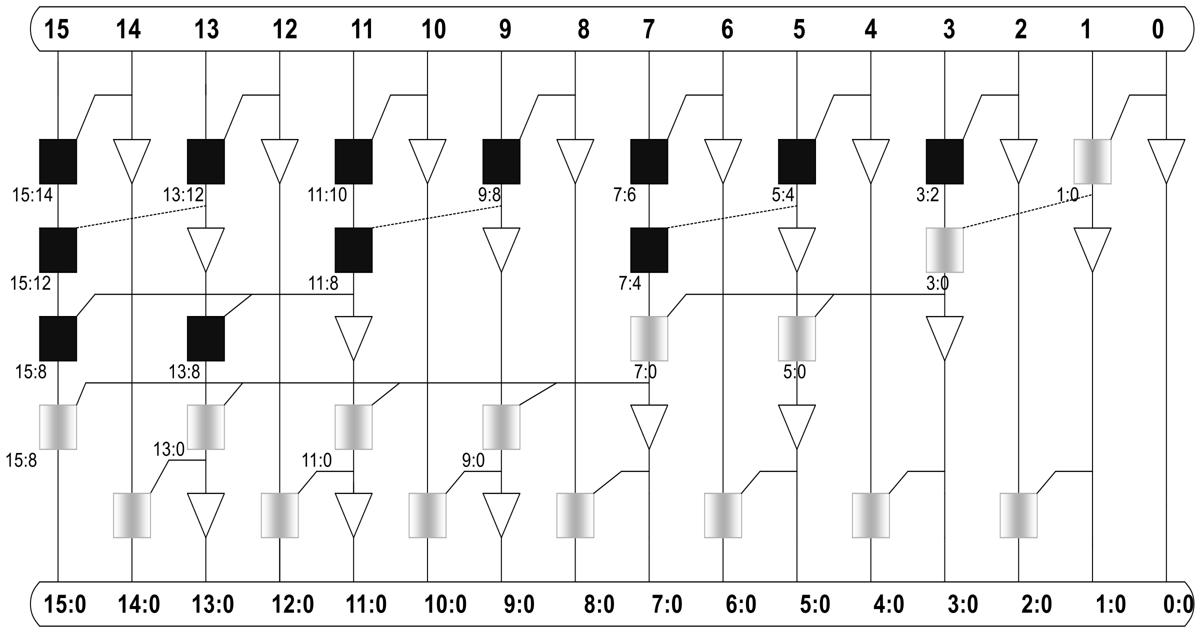


**Fig-2.14**

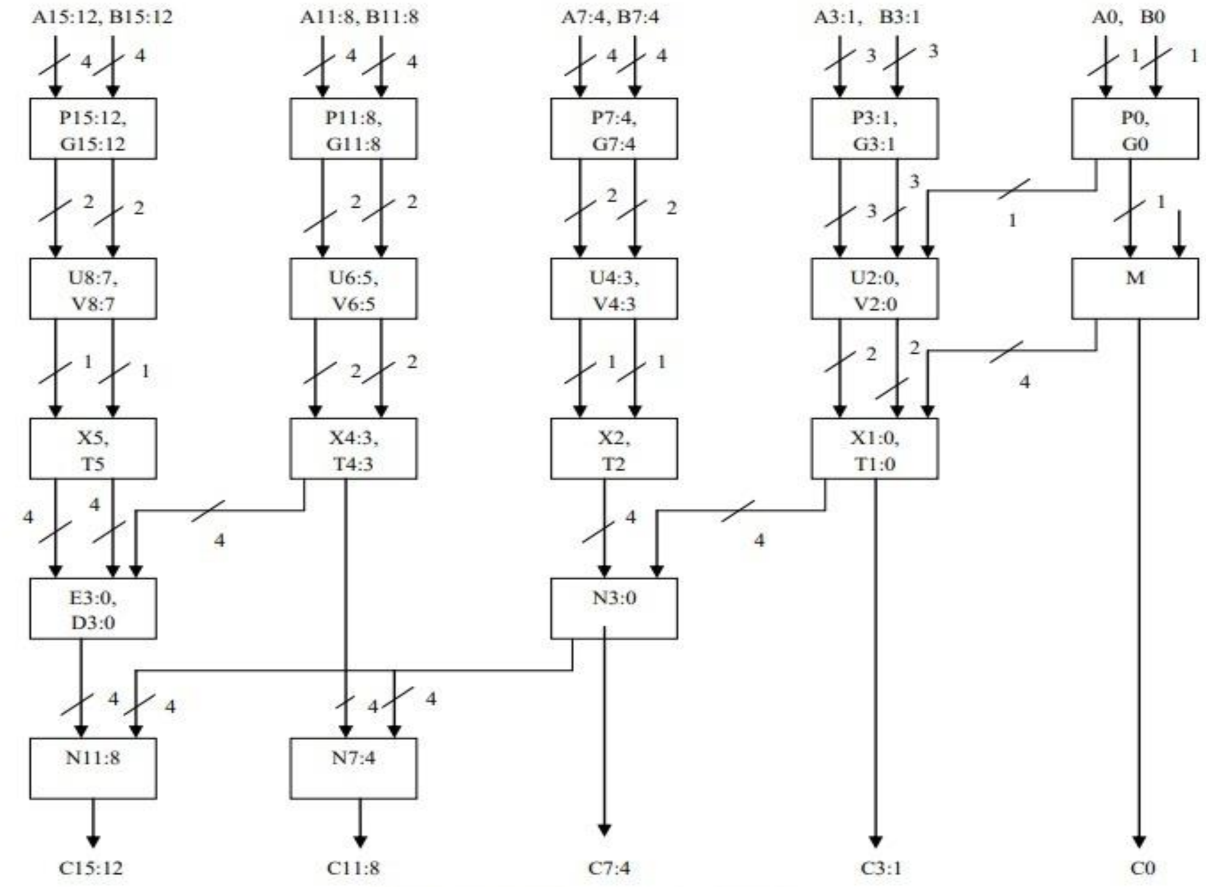


**Fig-2.15**

## 2.3.11 SKLANSKY ADDER

The sklansky adder has a very low delay of $\log_2 N$, and hence it is also called as divide & conquer tree adder method. It can simultaneously calculate mid and large group prefixes. At each stage, fanout is increased two fold. High fan outs can hint at poor performance for wide adders. In order to increase efficiency, we can restrict fan out and a buffer can be used for critical signals so that they can be used for further proceeding for intermediate prefixes.

Sklansky prefix trees have the fewest logic levels and use the fewest cells compared to Kogge-Stone and Knowles prefix trees. The main issue with the Sklansky prefix tree is its large fan-out. To address this issue, the Ladner-Fischer prefix tree is proposed. In order to decrease fan out without adding extra cells, its necessary to increase the number of logic levels. The block diagram of sklansky adder in the below fig – 2.16

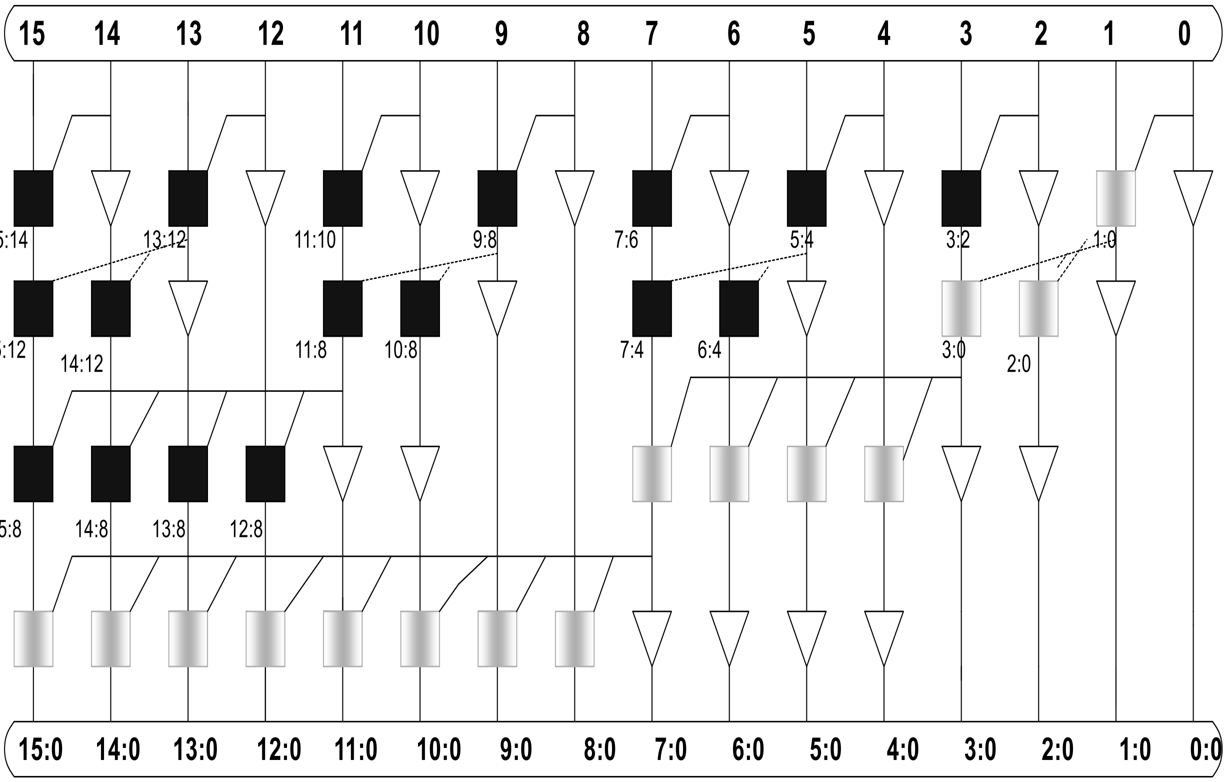
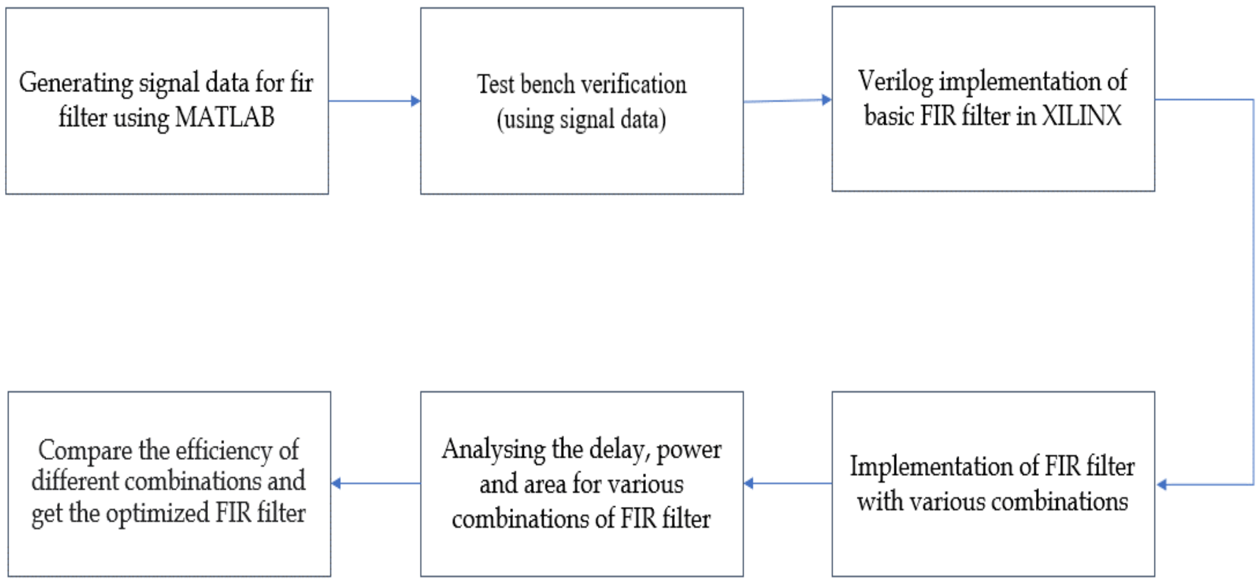


**Fig-2.16**

# CHAPTER 3

## Implementation of direct form and cascade form

This chapter describes the implementation of both direct and cascade forms of FIR filter with various adders and multipliers. The block diagram of the implementation steps of the procedure in the fig-3.1



Generating signal data for fir filter using MATLAB
Test bench verification (using signal data)
Verilog implementation of basic FIR filter in XILINX
Implementation of FIR filter with various combinations
Analysing the delay, power and area for various combinations of FIR filter
Compare the efficiency of different combinations and get the optimized FIR filter

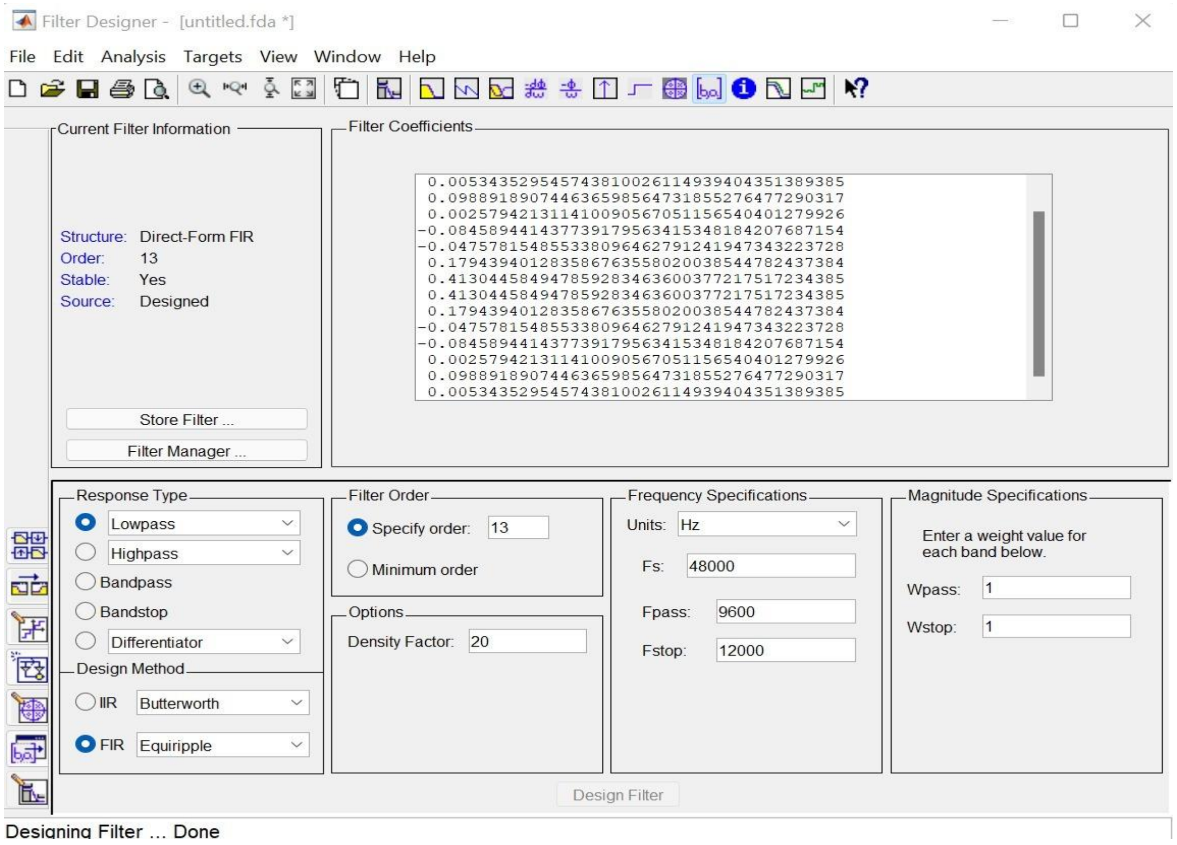
**Fig-3.1**

## 3.1 SIGNAL GENERATION

We generated a signal using MATLAB and then extracted the values and use that data for the simulation of our FIR filters.

- Using Filter Builder option in MATLAB.
- select the filter type if its low pass or high pass.
- we select the form of FIR Filter, whether it is direct or cascade.
- Then we go on to set the frequency as a result of which a signal will be generated.
- In the generated signal window, we have to open the co-efficient and extract them for later use.
- Select the order of the filter and we will have generated the 13th order 16-bit FIR filter

The procedure to build a low pass FIR filter and generate a signal in MATLAB is shown in fig-3.2


Filter Designer - [untitled.fda *]
File Edit Analysis Targets View Window Help
Store Filter ...
Filter Manager ...
Design Filter

Current Filter Information

Structure: Direct-Form FIR
Order: 13
Stable: Yes
Source: Designed

Filter Coefficients

```
 0.005343529545743810026114939404351389385
 0.098891890744636598564731855276477290317
 0.002579421311410090567051156540401279926
-0.084589441437739179563415348184207687154
-0.047578154855338096462791241947343223728
 0.179439401283586763558020038544782437384
 0.413044584947859283463600377217517234385
 0.413044584947859283463600377217517234385
 0.179439401283586763558020038544782437384
-0.047578154855338096462791241947343223728
-0.084589441437739179563415348184207687154
 0.002579421311410090567051156540401279926
 0.098891890744636598564731855276477290317
 0.005343529545743810026114939404351389385
```

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR Equiripple]

Filter Order: [✓ Specify order: 13] Minimum order

Options: Density Factor: 20

Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 9600; Fstop: 12000

Magnitude Specifications: Enter a weight value for each band below. Wpass: 1; Wstop: 1

Designing Filter ... Done

**Fig-3.2**

## 3.2 STEP-WISE IMPLEMENTATION

The steps of the procedure mentioned below this point would describe the implementation of building the filters and getting the reports.

- The figure- 3.3 depicts the process to initate a vivado project and select RTL project option to create a project file.

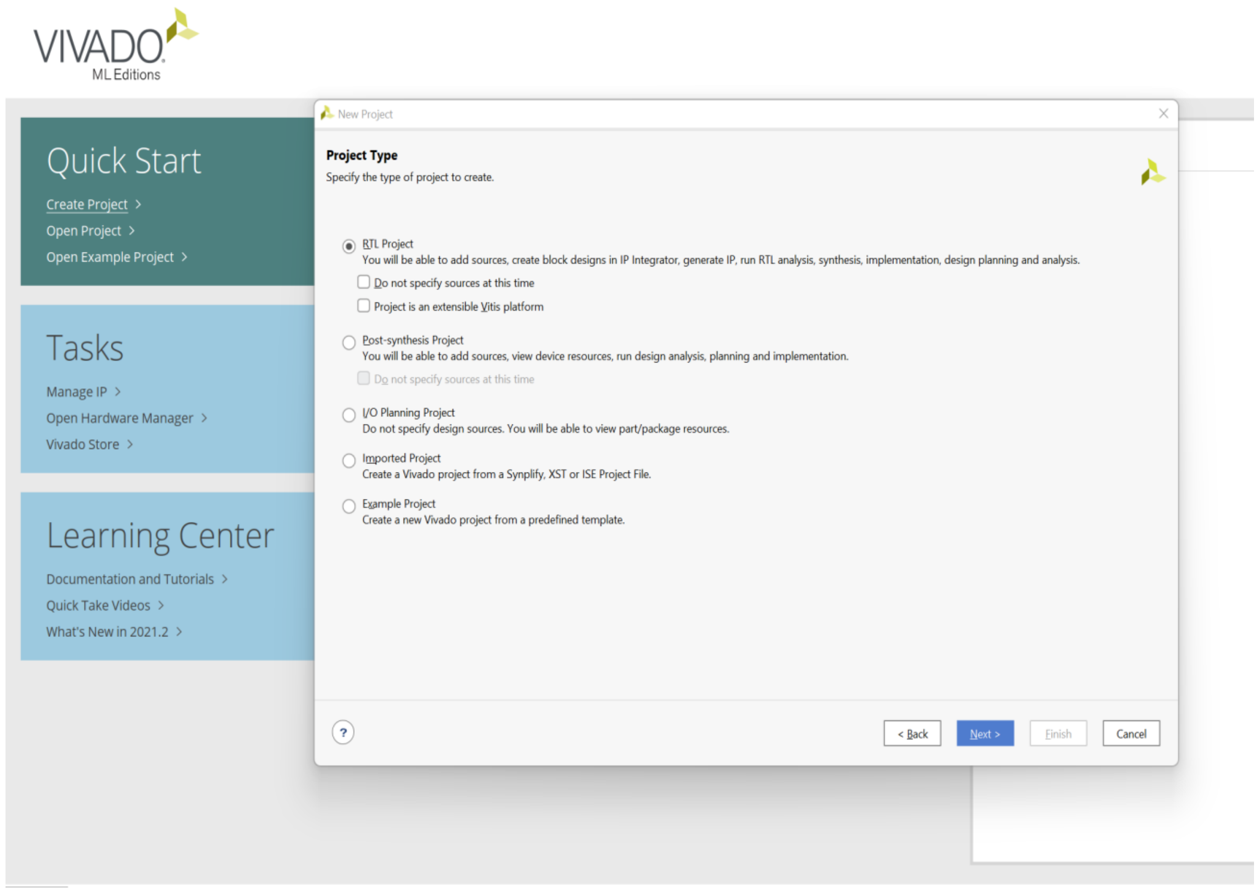
- Click next and select the add source option to create a new source file.

- It is in this file that we write the code for the intended FIR filter combination

- These steps will be further repeated several times for all the different FIR filters that we create using each and every set of adder and multiplier.

- Then we will create a test bench of our choice to verify the working of the code which has been written earlier.

The figure-3.3 depicts how a new RTL project would be created.



**Project Type**

Specify the type of project to create.

[✓ RTL Project]
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☐ Do not specify sources at this time

☐ Project is an extensible Vitis platform

Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.

☐ Do not specify sources at this time

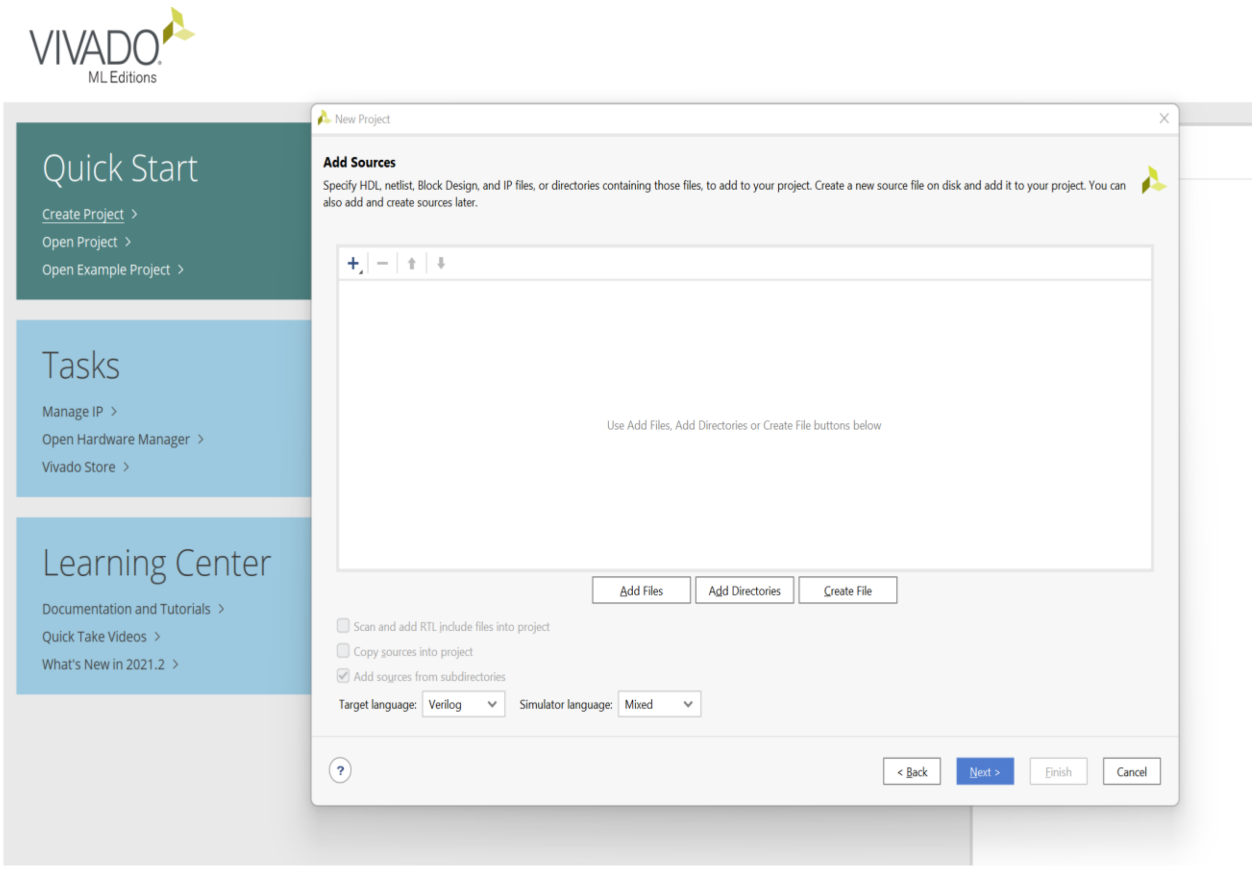
I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

Example Project
Create a new Vivado project from a predefined template.

**Fig-3.3**

The figure 3.4 below describes how to include the necessary source files.



**Add Sources**

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

Scan and add RTL include files into project

Copy sources into project

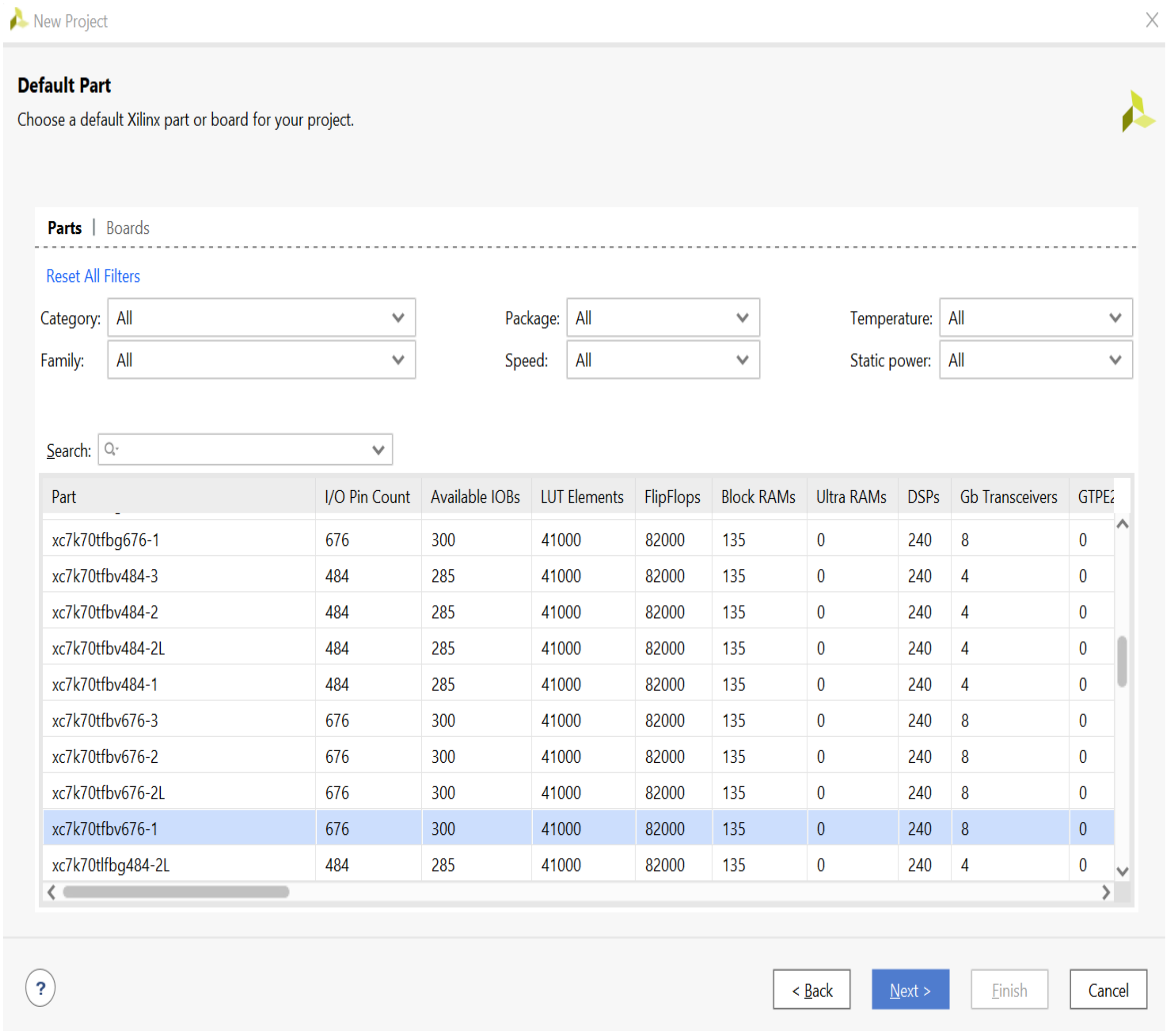
☑ Add sources from subdirectories

Target language: Verilog    Simulator language: Mixed

**Fig-3.4**

- A window will present itself showing us the summary of all the selections we made in the process of creating the RTL project file.

This figure 3.5 depicts the selection of default part or board we choose to use.



**Default Part**

Choose a default Xilinx part or board for your project.

Category: All
Family: All
Package: All
Speed: All
Temperature: All
Static power: All

Search:

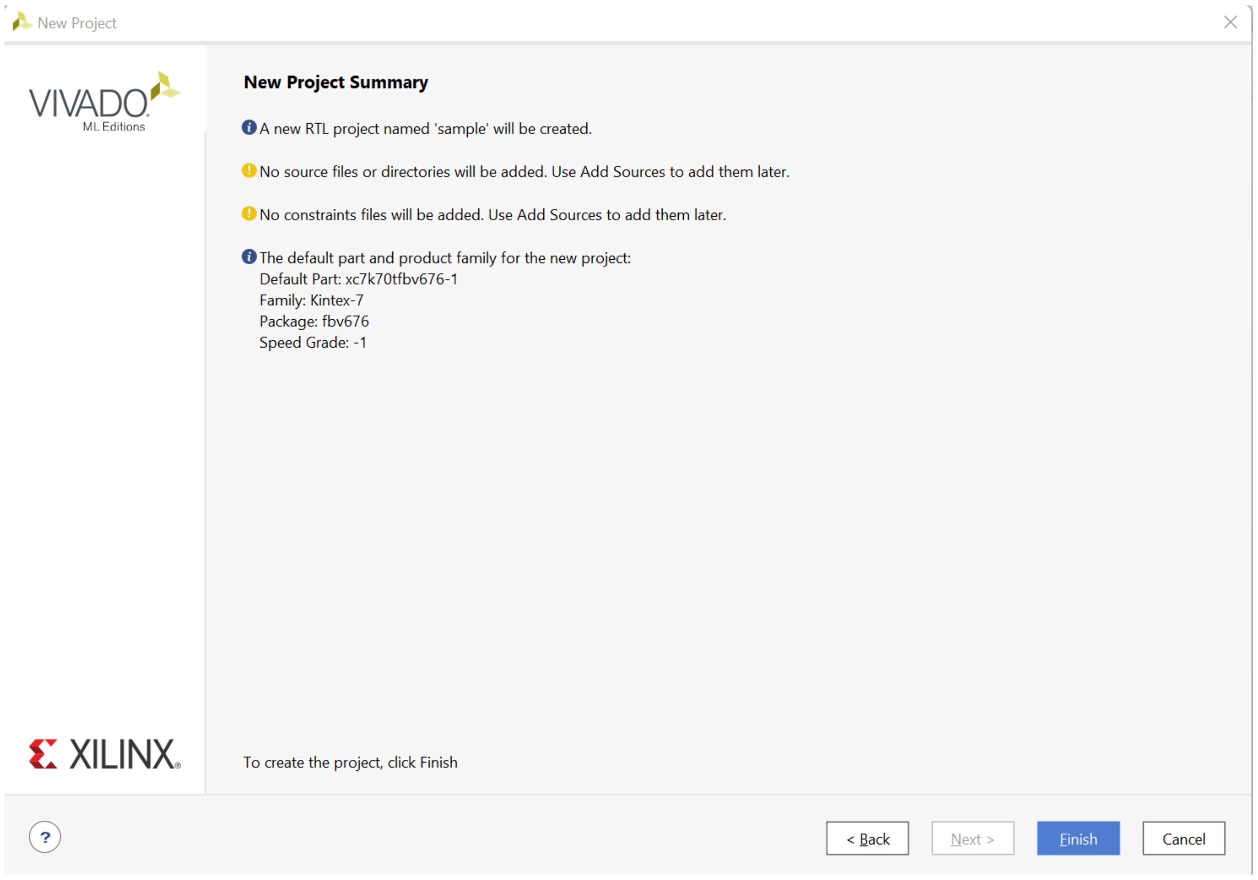
| Part | I/O Pin Count | Available IOBs | LUT Elements | FlipFlops | Block RAMs | Ultra RAMs | DSPs | Gb Transceivers | GTPE2 |
|---|---|---|---|---|---|---|---|---|---|
| xc7k70tfbg676-1 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 8 | 0 |
| xc7k70tfbv484-3 | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 4 | 0 |
| xc7k70tfbv484-2 | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 4 | 0 |
| xc7k70tfbv484-2L | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 4 | 0 |
| xc7k70tfbv484-1 | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 4 | 0 |
| xc7k70tfbv676-3 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 8 | 0 |
| xc7k70tfbv676-2 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 8 | 0 |
| xc7k70tfbv676-2L | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 8 | 0 |
| xc7k70tfbv676-1 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 8 | 0 |
| xc7k70tlfbg484-2L | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 4 | 0 |

**Fig-3.5**

- We need to look for any changes to made, and if every option is the favourable choice for the further proceedings of our project, we can go through with it.

- Then we can just select the FINISH option at the bottom right part of the same window.

- Upon clicking it, a new page will be ready for us to write the code and build the wanted FIR filter with our remaining constraints and requirements with required adder and multipliers.



New Project Summary

A new RTL project named 'sample' will be created.

No source files or directories will be added. Use Add Sources to add them later.

No constraints files will be added. Use Add Sources to add them later.

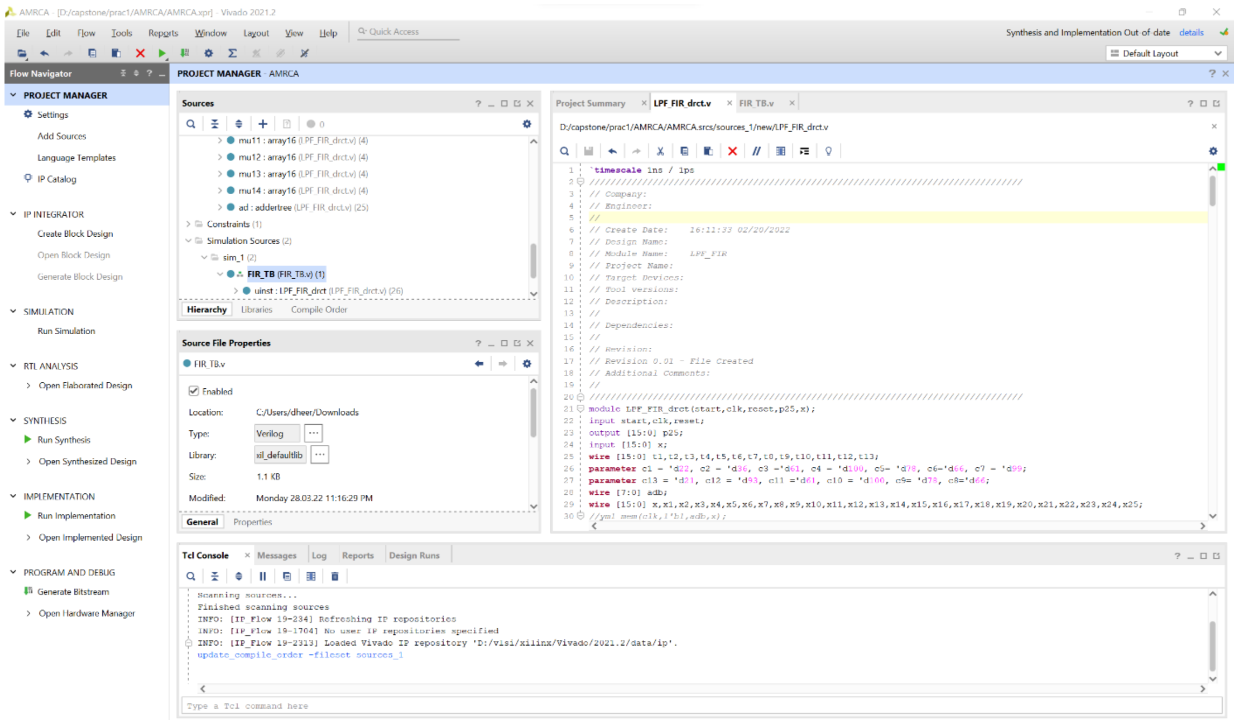
The default part and product family for the new project:
Default Part: xc7k70tfbv676-1
Family: Kintex-7
Package: fbv676
Speed Grade: -1

To create the project, click Finish

**Fig-3.6**

- This will takes us to the next steps of the process where we implement the filter code and execute it and does the analysis.

- Write the verilog code for FIR filters with all the combinations of adders and multipliers as shown in fig-3.7



Enabled
Location: C:/Users/dheer/Downloads
Type: Verilog
Library: xil_defaultlib
Size: 1.1 KB
Modified: Monday 28.03.22 11:16:29 PM

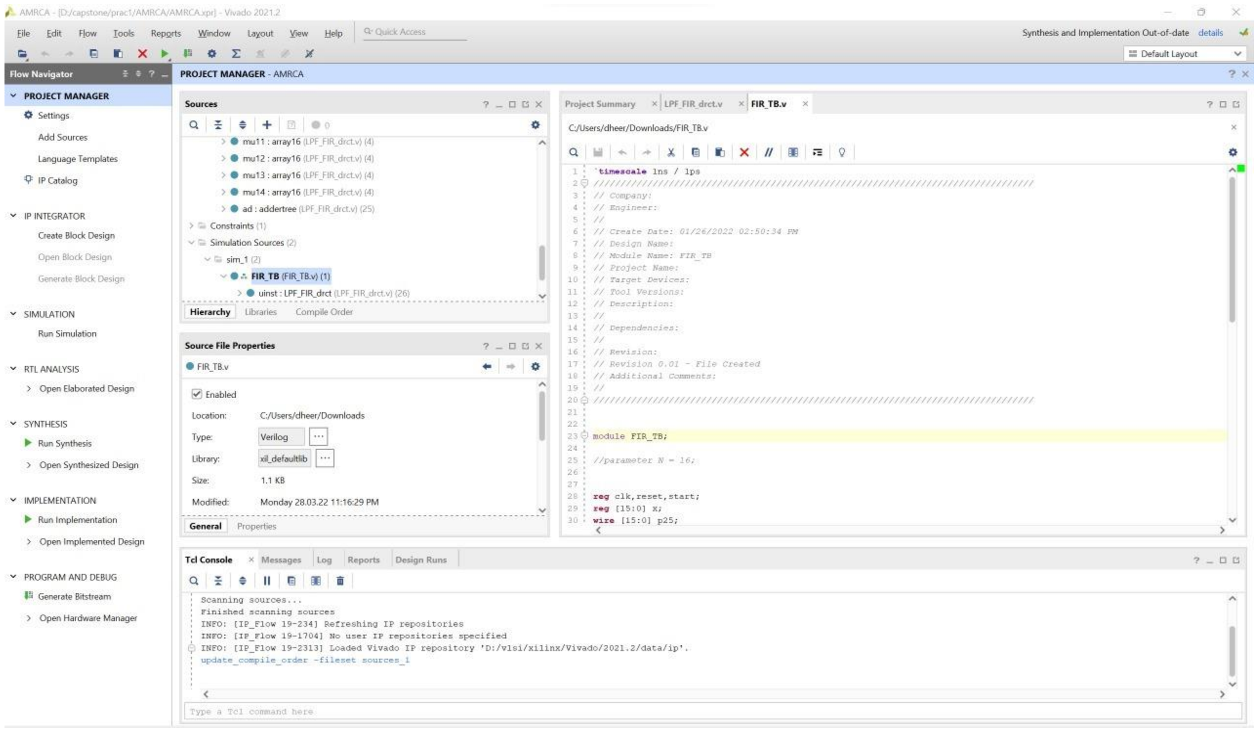
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:11:33 02/20/2022
// Design Name:
// Module Name:    LPF_FIR
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module LPF_FIR_drct(start,clk,reset,p25,x);
input start,clk,reset;
output [15:0] p25;
input [15:0] x;
wire [15:0] t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13;
parameter c1 = 'd22, c2 = 'd36, c3 ='d61, c4 = 'd100, c5= 'd78, c6='d66, c7 = 'd99;
parameter c13 = 'd21, c12 = 'd93, c11 ='d61, c10 = 'd100, c9= 'd78, c8='d66;
wire [7:0] adb;
wire [15:0] x,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,x22,x23,x24,x25;
//yml mem(clk,1'b1,adb,x);
```

```
Scanning sources...
Finished scanning sources
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'D:/vlsi/xilinx/Vivado/2021.2/data/ip'.
update_compile_order -fileset sources_1
```

Type a Tcl command here

**Fig-3.7**

- Add the test bench to the directory to test the results as shown in fig-3.8



Enabled
Location: C:/Users/dheer/Downloads
Type: Verilog
Library: xil_defaultlib
Size: 1.1 KB
Modified: Monday 28.03.22 11:16:29 PM

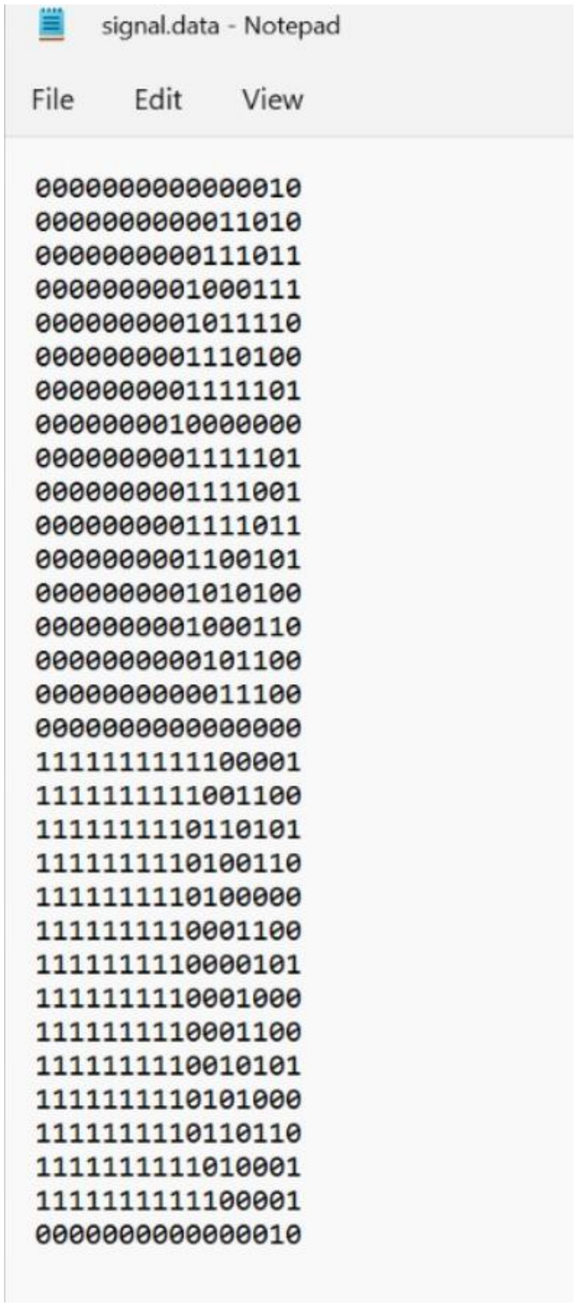
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/26/2022 02:50:34 PM
// Design Name:
// Module Name: FIR_TB
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FIR_TB;

//parameter N = 16;


reg clk,reset,start;
reg [15:0] x;
wire [15:0] p25;
```

```
Scanning sources...
Finished scanning sources
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'D:/vlsi/xilinx/Vivado/2021.2/data/ip'.
update_compile_order -fileset sources_1
```

Type a Tcl command here

**Fig-3.8**

- Add the signal file generated with the data extracted from matlab which is run with our FIR filter code and test bench to process.The extracted values are shown in the fig-3.9



```
0000000000000010
0000000000011010
0000000000111011
0000000001000111
0000000001011110
0000000001110100
0000000001111101
0000000010000000
0000000001111101
0000000001111001
0000000001111011
0000000001100101
0000000001010100
0000000001000110
0000000000101100
0000000000011100
0000000000000000
1111111111100001
1111111111001100
1111111110110101
1111111110100110
1111111110100000
1111111110001100
1111111110000101
1111111110001000
1111111110001100
1111111110010101
1111111110101000
1111111110110110
1111111111010001
1111111111100001
0000000000000010
```

**Fig-3.9**

The picture attached below figure – 3.10 depicts all the various features available in the software for us to explore and use to get the appropriate analysis made.

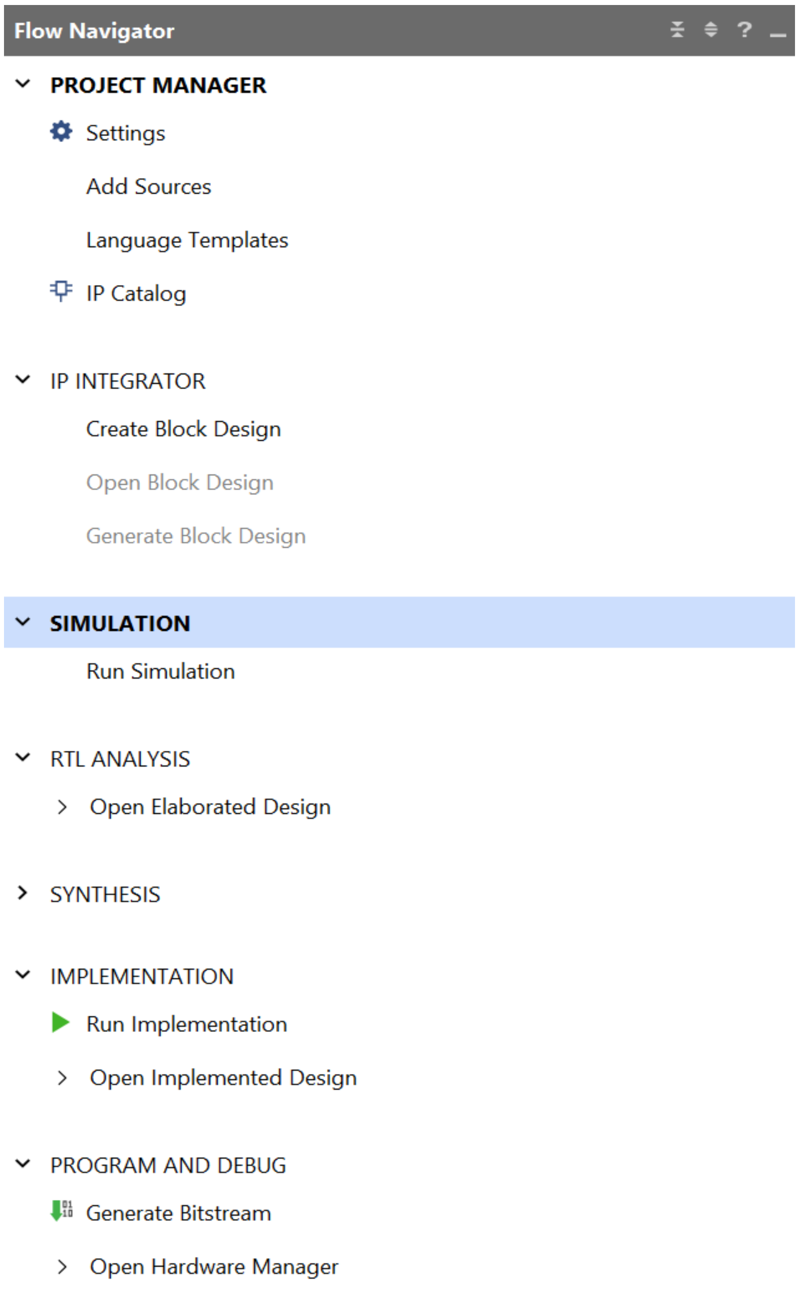
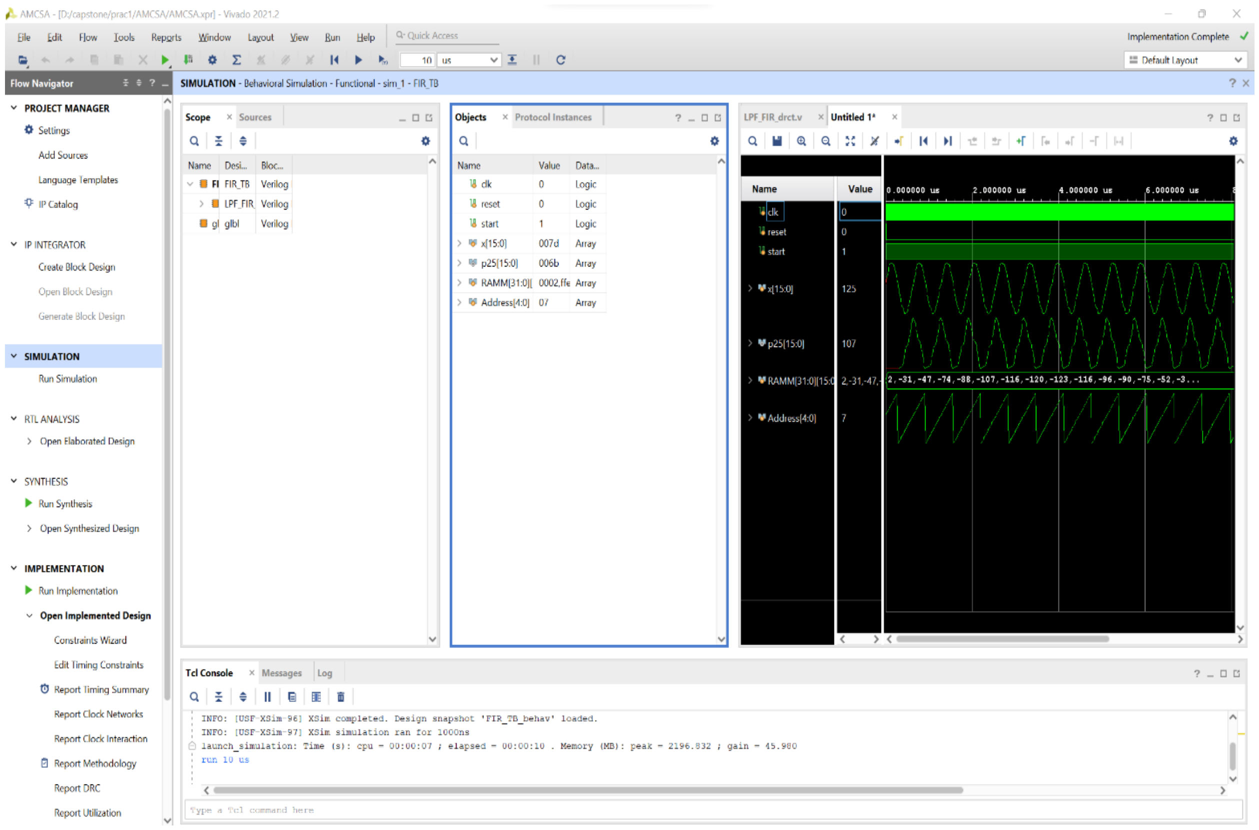


**Fig-3.10**

- Run implementation in vivado for all the prepared combinations of FIR filters and we will get the results as graphs as shown in the fig-3.11,3.12.
- These can also be used to get an understanding of the objects operating for the purpose of this filter like:

  - ➢ A list of wires used
  - ➢ The no.of inputs bits denoted as x
  - ➢ The no.of outputs bits denotes as p
  - ➢ Address array



| Name | Desi... | Bloc... |
|---|---|---|
| FI | FIR_TB | Verilog |
| | LPF_FIR | Verilog |
| gl | glbl | Verilog |

| Name | Value | Data... |
|---|---|---|
| clk | 0 | Logic |
| reset | 0 | Logic |
| start | 1 | Logic |
| x[15:0] | 007d | Array |
| p25[15:0] | 006b | Array |
| RAMM[31:0][ | 0002,ffe | Array |
| Address[4:0] | 07 | Array |

| Name | Value |
|---|---|
| clk | 0 |
| reset | 0 |
| start | 1 |
| x[15:0] | 125 |
| p25[15:0] | 107 |
| RAMM[31:0][15:0 | 2,-31,-47, |
| Address[4:0] | 7 |

```
2,-31,-47,-74,-88,-107,-116,-120,-123,-116,-96,-90,-75,-52,-3...
```

```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'FIR_TB_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:07 ; elapsed = 00:00:10 . Memory (MB): peak = 2196.832 ; gain = 45.980
run 10 us
```

**Fig-3.11**

- The underneath figure3.12 shows a filters output signal that we get after filtering singal for the filter which has been built by us.

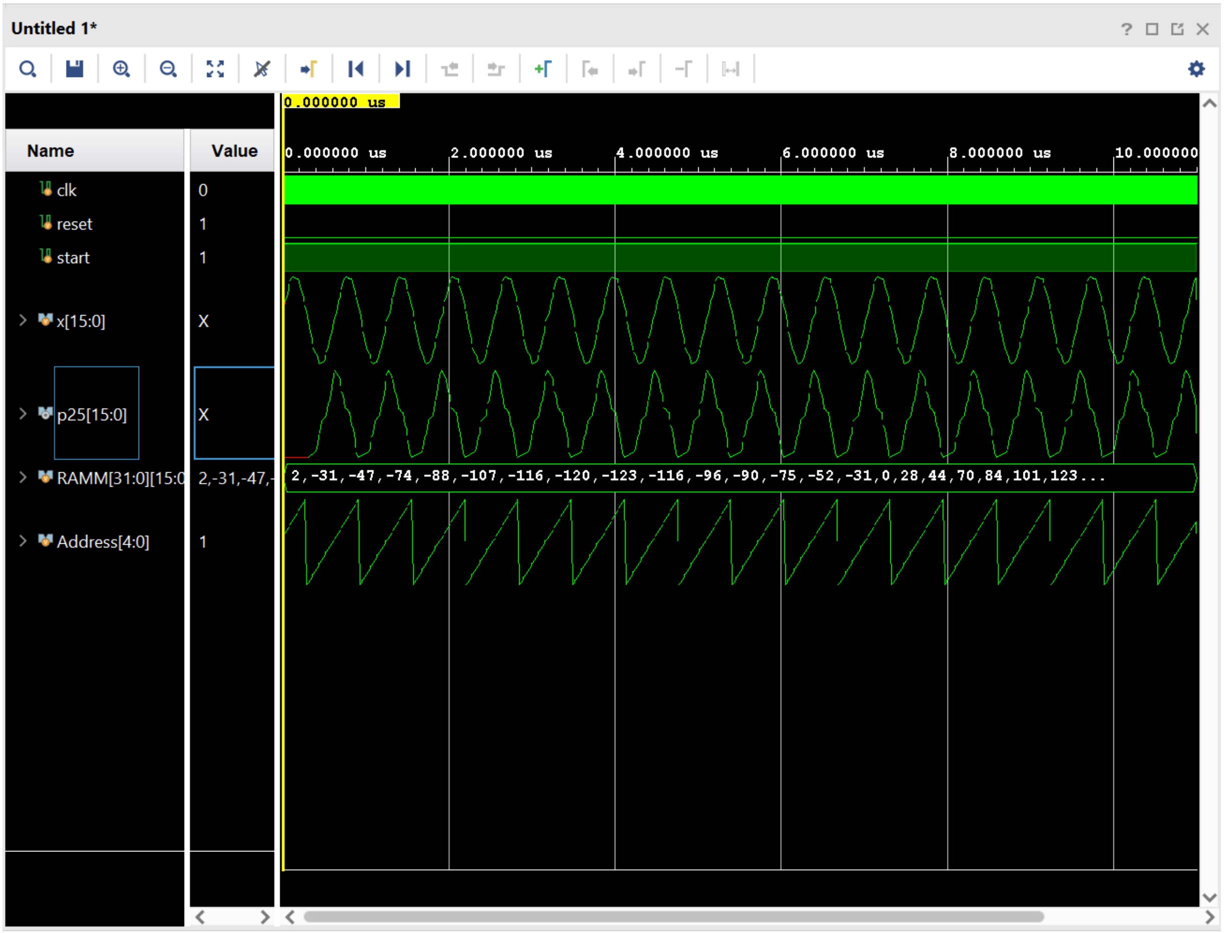
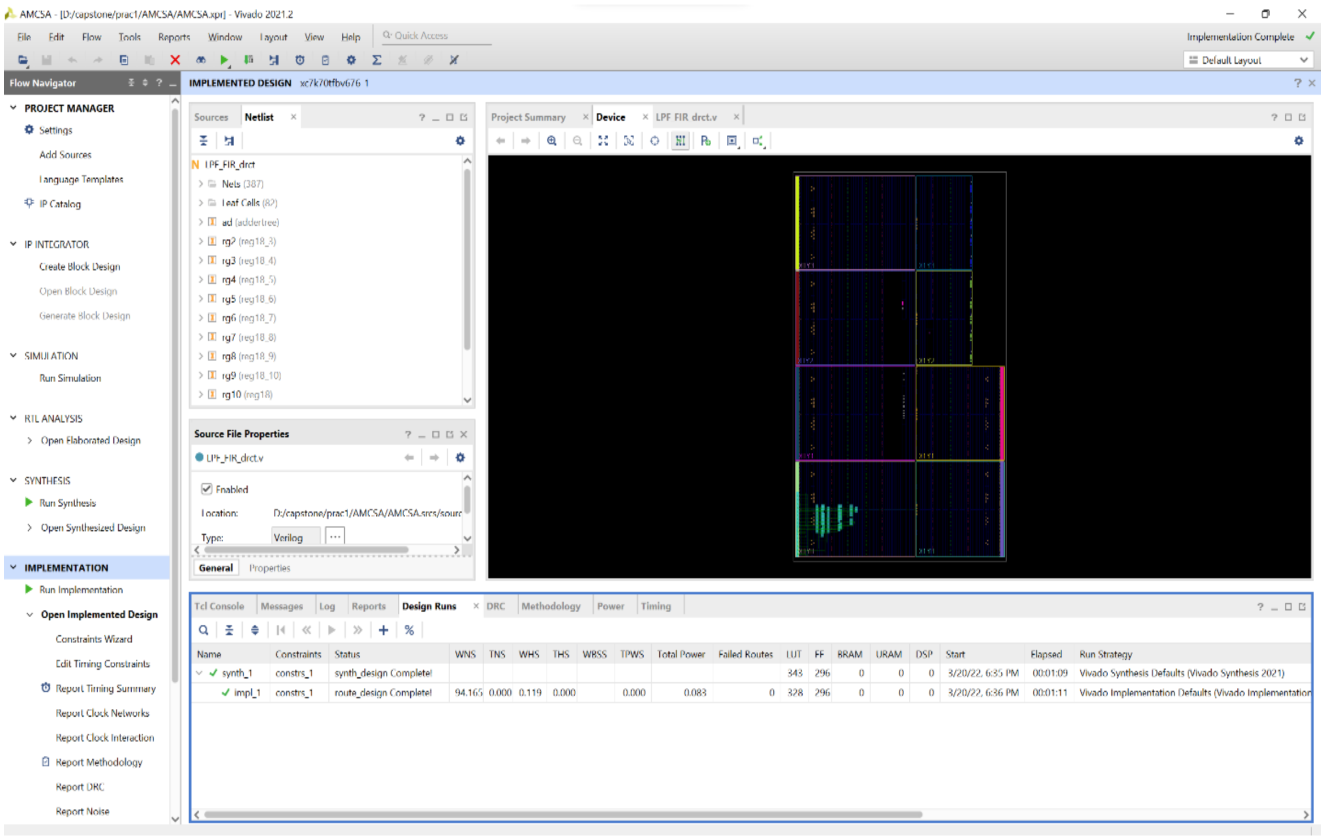


**Fig-3.12**

This software also enabled us to uses multiple functions use various of its attributes that would give us an idea of the schematics of the device upon implementation.

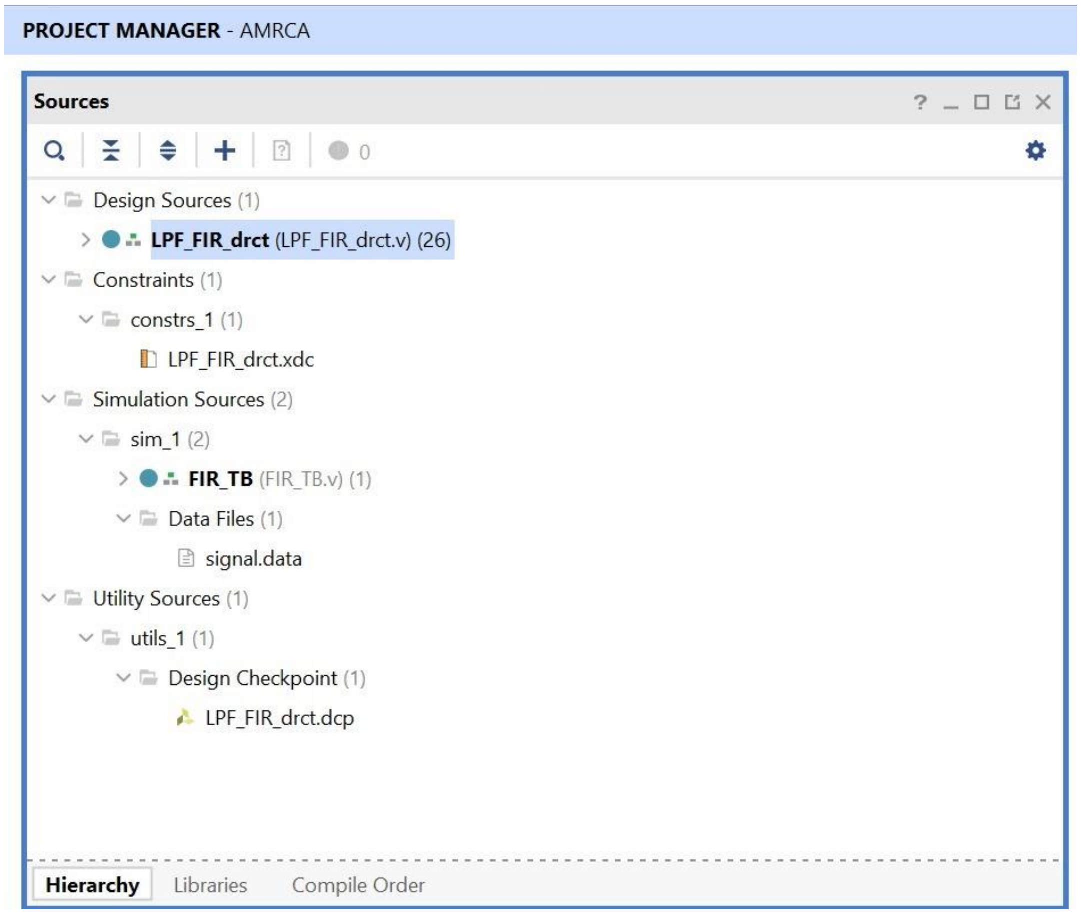
- Get various reports from implementation reports as per our requirements.
- The below figure-3.13 depicts the schematic of the designs as mentioned above.



| Name | Constraints | Status | WNS | TNS | WHS | THS | WBSS | TPWS | Total Power | Failed Routes | LUT | FF | BRAM | URAM | DSP | Start | Elapsed | Run Strategy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ synth_1 | constrs_1 | synth_design Complete! | | | | | | | | | 343 | 296 | 0 | 0 | 0 | 3/20/22, 6:35 PM | 00:01:09 | Vivado Synthesis Defaults (Vivado Synthesis 2021) |
| ✓ impl_1 | constrs_1 | route_design Complete! | 94.165 | 0.000 | 0.119 | 0.000 | | 0.000 | 0.083 | 0 | 328 | 296 | 0 | 0 | 0 | 3/20/22, 6:36 PM | 00:01:11 | Vivado Implementation Defaults (Vivado Implementation |

**Fig-3.13**

- We need to makes sure that the files created are placed in proper directories, otherwise the files will not be run.
- The below picture (figure-3.14) indicates the proper positioning of the tests bench, verilog file, and also singal data file which ensures the proper working of our filers.


PROJECT MANAGER - AMRCA

Sources

0

- Design Sources (1)
  - **LPF_FIR_drct** (LPF_FIR_drct.v) (26)
- Constraints (1)
  - constrs_1 (1)
    - LPF_FIR_drct.xdc
- Simulation Sources (2)
  - sim_1 (2)
    - **FIR_TB** (FIR_TB.v) (1)
    - Data Files (1)
      - signal.data
- Utility Sources (1)
  - utils_1 (1)
    - Design Checkpoint (1)
      - LPF_FIR_drct.dcp

**Hierarchy** Libraries Compile Order

**Fig-3.14**

- Take the values from timing report, power report, and also DRC reports. We can get the RTL schematic as shown in fig-3.15

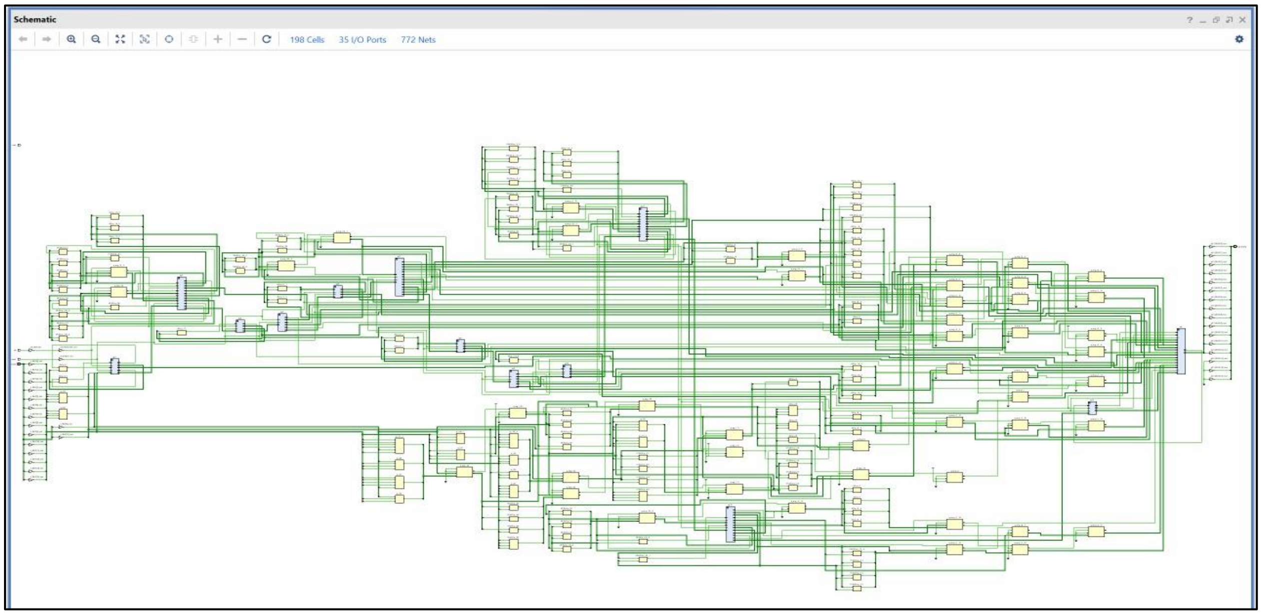
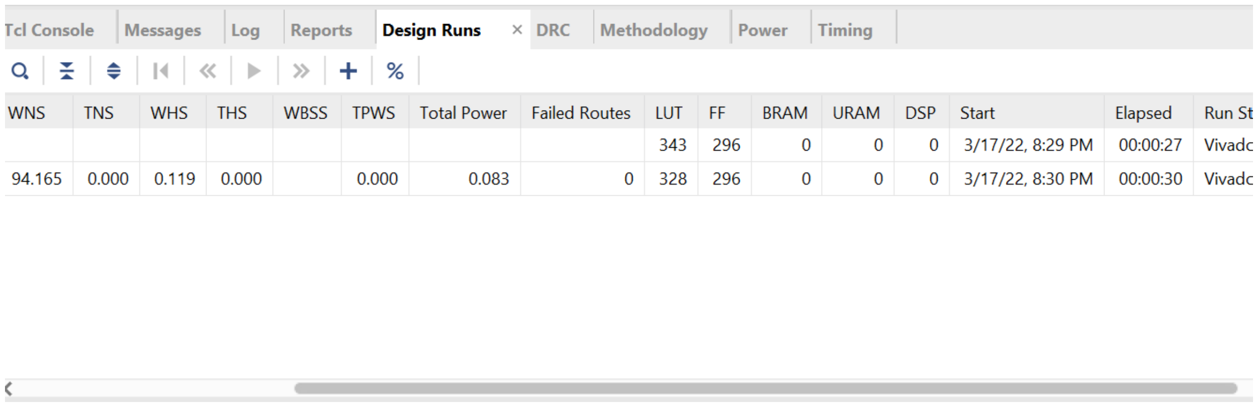


**Fig-3.15**

- Note down the values of different variables in all the combinations of FIR filters like Delay, no. of LUTs, Power.

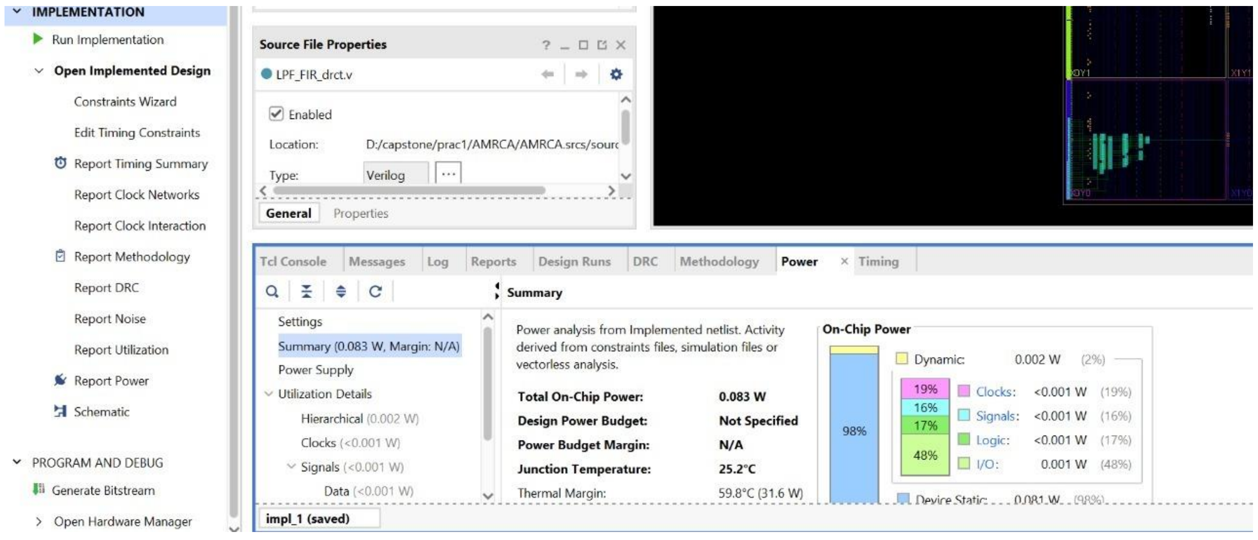


Tcl Console | Messages | Log | Reports | Design Runs × | DRC | Methodology | Power | Timing

| WNS | TNS | WHS | THS | WBSS | TPWS | Total Power | Failed Routes | LUT | FF | BRAM | URAM | DSP | Start | Elapsed | Run St |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  | 343 | 296 | 0 | 0 | 0 | 3/17/22, 8:29 PM | 00:00:27 | Vivadc |
| 94.165 | 0.000 | 0.119 | 0.000 |  | 0.000 | 0.083 | 0 | 328 | 296 | 0 | 0 | 0 | 3/17/22, 8:30 PM | 00:00:30 | Vivadc |

**Fig-3.16**

- In the below figure-3.17 we will see how to check the power analysis, to get that we will select the “report-power” option in the implementation section in the left part of the window.



IMPLEMENTATION
- Run Implementation
- Open Implemented Design
  - Constraints Wizard
  - Edit Timing Constraints
  - Report Timing Summary
  - Report Clock Networks
  - Report Clock Interaction
  - Report Methodology
  - Report DRC
  - Report Noise
  - Report Utilization
  - Report Power
  - Schematic

PROGRAM AND DEBUG
- Generate Bitstream
- Open Hardware Manager

Source File Properties

LPF_FIR_drct.v

Enabled

Location: D:/capstone/prac1/AMRCA/AMRCA.srcs/sourc

Type: Verilog

General | Properties

Tcl Console | Messages | Log | Reports | Design Runs | DRC | Methodology | Power × | Timing

Settings
Summary (0.083 W, Margin: N/A)
Power Supply
Utilization Details
  Hierarchical (0.002 W)
  Clocks (<0.001 W)
  Signals (<0.001 W)
    Data (<0.001 W)

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

| | |
|---|---|
| Total On-Chip Power: | 0.083 W |
| Design Power Budget: | Not Specified |
| Power Budget Margin: | N/A |
| Junction Temperature: | 25.2°C |
| Thermal Margin: | 59.8°C (31.6 W) |

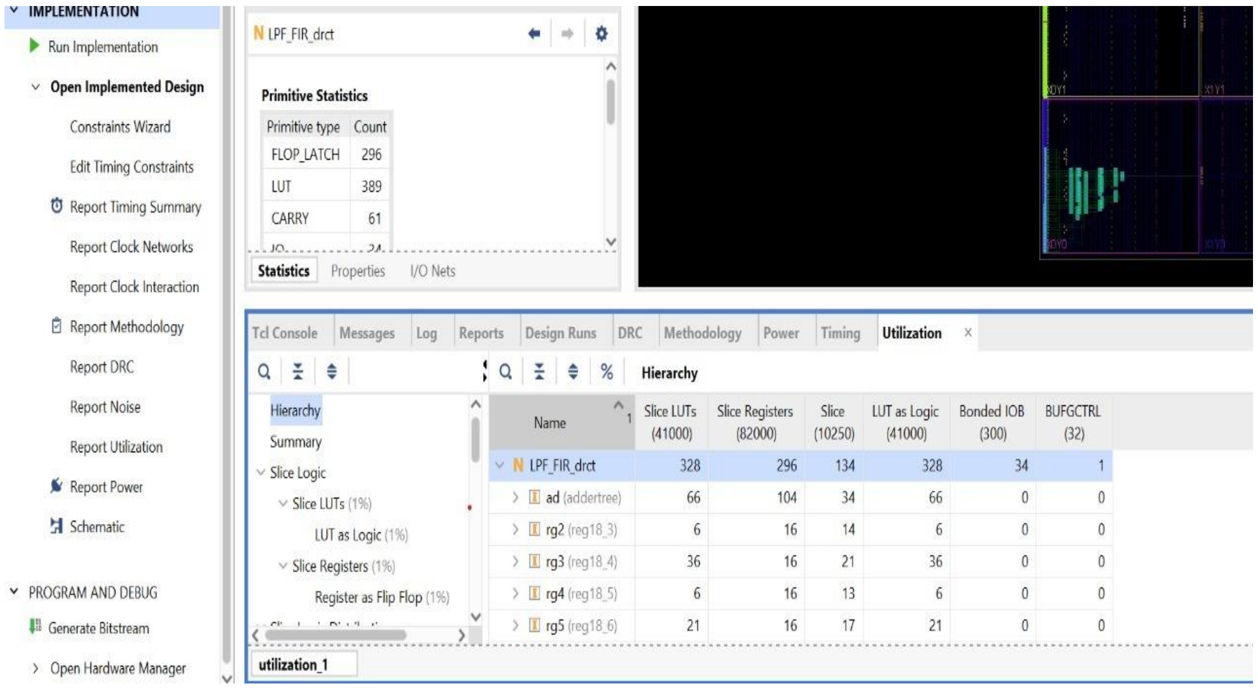
On-Chip Power

| | | |
|---|---|---|
| Dynamic: | 0.002 W | (2%) |
| Clocks: | <0.001 W | (19%) |
| Signals: | <0.001 W | (16%) |
| Logic: | <0.001 W | (17%) |
| I/O: | 0.001 W | (48%) |
| Device Static: | 0.081 W | (98%) |

impl_1 (saved)

**Fig-3.17**

- To get the no.of LUTs, which would implicate the area consumed, click on the "Report-Unilization" in the implementation design section in the left part of the window which is called flow navigator the way it has been depicted in the below figure-3.18.
-



Primitive Statistics

| Primitive type | Count |
|---|---|
| FLOP_LATCH | 296 |
| LUT | 389 |
| CARRY | 61 |

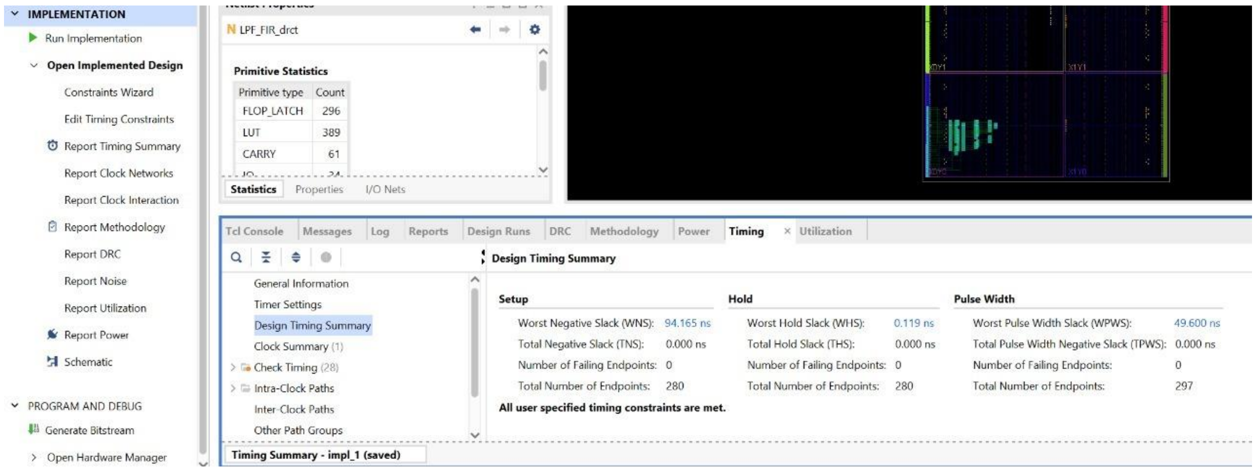
Hierarchy

| Name | Slice LUTs (41000) | Slice Registers (82000) | Slice (10250) | LUT as Logic (41000) | Bonded IOB (300) | BUFGCTRL (32) |
|---|---|---|---|---|---|---|
| LPF_FIR_drct | 328 | 296 | 134 | 328 | 34 | 1 |
| ad (addertree) | 66 | 104 | 34 | 66 | 0 | 0 |
| rg2 (reg18_3) | 6 | 16 | 14 | 6 | 0 | 0 |
| rg3 (reg18_4) | 36 | 16 | 21 | 36 | 0 | 0 |
| rg4 (reg18_5) | 6 | 16 | 13 | 6 | 0 | 0 |
| rg5 (reg18_6) | 21 | 16 | 17 | 21 | 0 | 0 |

**Fig-3.18**

- To get the timing details of the analysis, we have to select the "Report-Timing-summary" in the implementation section in the flow navigator.
- There we will be shown a term called "worst-negative-slack" which would indicate the delay of the filter.

- The below figure-3.19 represents the “timing report summary” of the filter.



Primitive Statistics

| Primitive type | Count |
|---|---|
| FLOP_LATCH | 296 |
| LUT | 389 |
| CARRY | 61 |

Design Timing Summary

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 94.165 ns | Worst Hold Slack (WHS): | 0.119 ns | Worst Pulse Width Slack (WPWS): | 49.600 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 280 | Total Number of Endpoints: | 280 | Total Number of Endpoints: | 297 |

All user specified timing constraints are met.

**Fig-3.19**

# CHAPTER 4

# RESULTS AND DISCUSSION

## DIRECT FORM:

| Filt. combinations | no. of LUTS | ON-chip power(W) | worst negative slack(WNS)(ns) |
|---|---|---|---|
| bmrca | 2638 | 0.088 | 76.828 |
| bmcsa | 2847 | 0.09 | 77.276 |
| bmcla | 2547 | 0.089 | 77.525 |
| bmcsia | 2529 | 0.089 | 76.768 |
| bmbrentkung | 2704 | 0.09 | 76.986 |
| bmhancarlson | 3273 | 0.091 | 76.08 |
| bmharris | 3295 | 0.091 | 77.038 |
| bmknowles | 2835 | 0.09 | 77.005 |
| bmladnerficsher | 2774 | 0.09 | 77.508 |
| bmsklansky | 3225 | 0.091 | 76.639 |
| bmkogge | 2748 | 0.09 | 76.469 |
| | | | |
| | | | |
| amrca | 328 | 0.083 | 94.795 |
| amcsa | 329 | 0.083 | 94.165 |
| amcla | 343 | 0.081 | 95.206 |
| amcsia | 341 | 0.083 | 94.795 |
| ambrentkung | 360 | 0.083 | 94.043 |
| amhancarlson | 340 | 0.083 | 93.902 |
| amharris | 340 | 0.083 | 93.902 |
| amknowles | 339 | 0.083 | 93.66 |
| amladnerficsher | 360 | 0.083 | 94.043 |
| amsklansky | 357 | 0.083 | 93.47 |
| amkogge | 369 | 0.083 | 93.818 |
| | | | |
| | | | |
| vmrca | 619 | 0.083 | 94.013 |
| vmcsa | 619 | 0.084 | 92.33 |
| vmcla | 631 | 0.083 | 92.19 |
| vmcsia | 619 | 0.084 | 92.289 |
| vmbentkung | 677 | 0.084 | 92.653 |
| vmhancarlson | 629 | 0.084 | 92.229 |
| vmharris | 628 | 0.084 | 92.766 |
| vmknowles | 629 | 0.084 | 92.669 |
| vmladnerficsher | 677 | 0.084 | 92.448 |
| vmsklansky | 682 | 0.084 | 92.203 |
| vmkogge | 732 | 0.084 | 92.864 |

**Table-4.1**

**Area:**

The area occupied according to the number of LUTs shown in the below fig-4.1

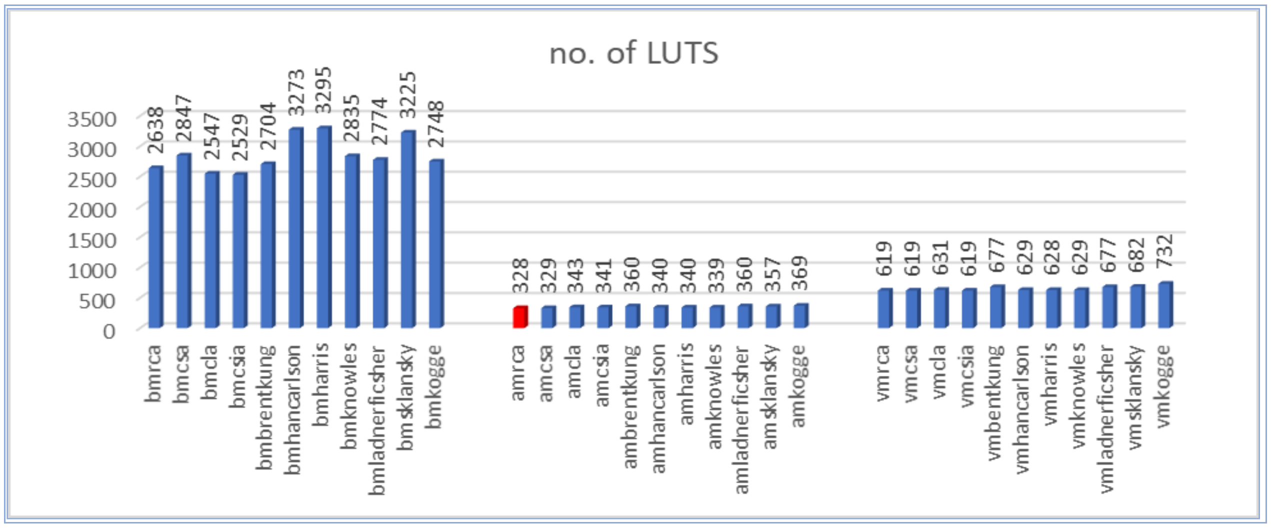


**Fig-4.1**

**Power:**

The on chip power of all the combinations in the Direct form with the least power consuming filter highlighted is shown in the fig-4.2

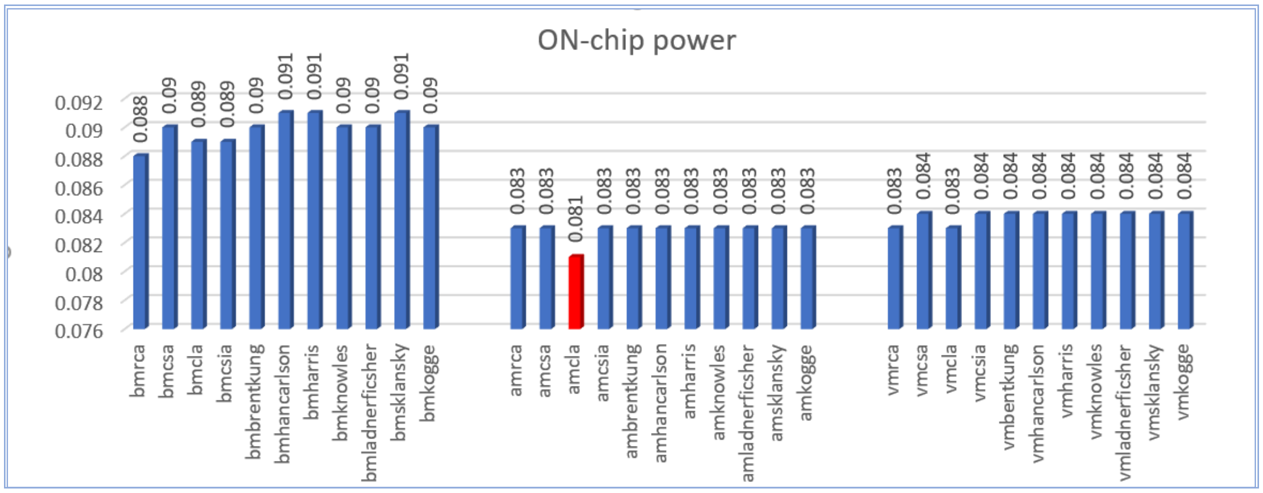


**Fig-4.2**

**Delay:**

The delay of the filter of all implementations are shown in the fig-4.3

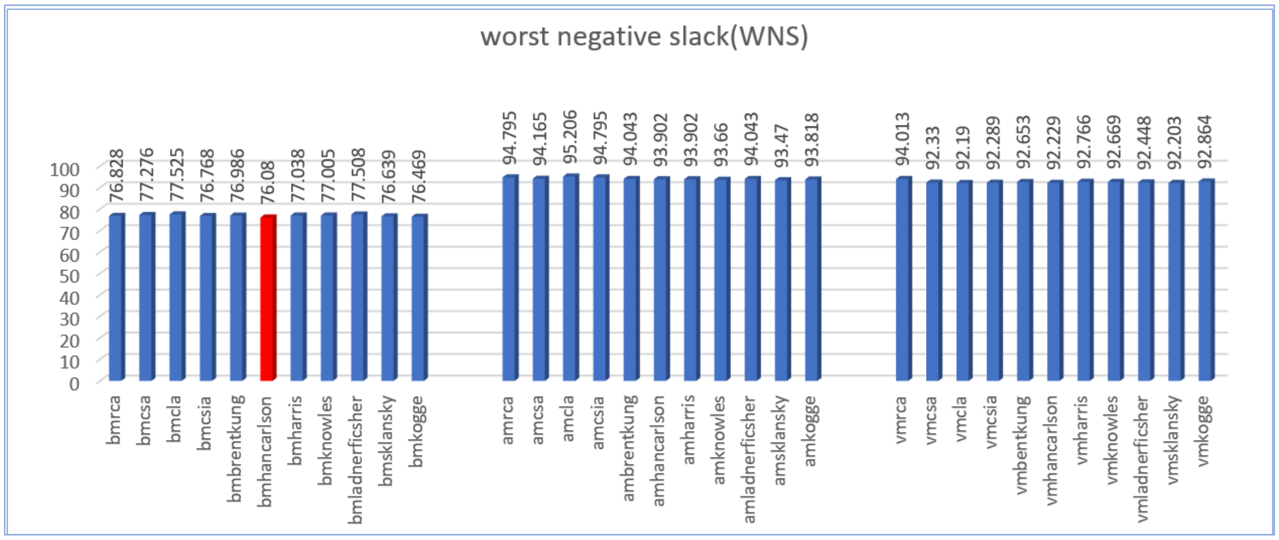


**Fig-4.3**

**Discussion:**

The array multiplier with Ripple carry adder consumes the least area as it has only 328 LUTs, which is the least amongst all the combinations that we have implemented. The power consumption is the least for FIR filter made with the Array Multiplier with Carry look-ahead adder which gives only 0.081 and the FIR filter made with booth multiplier in combination with Knowles adder will result in the least delay of 71.512.

## CASCADE FORM:

| Filt. combinations | no. of LUTS | ON-chip power(W) | worst negative slack(WNS)(ns) |
|---|---|---|---|
| bmrca_cascad | 3192 | 0.096 | 75.055 |
| bmcsa_cascad | 3292 | 0.094 | 80.63 |
| bmcla_cascad | 3356 | 0.096 | 74.449 |
| bmcsia_cascad | 3212 | 0.096 | 74.783 |
| bmbrentkung_cascad | 3234 | 0.096 | 74.83 |
| bmhancarlson_cascad | 3380 | 0.096 | 74.202 |
| bmharris_cascad | 3392 | 0.096 | 73.573 |
| bmknowles_cascad | 3228 | 0.096 | 73.49 |
| bmladnerficsher_cascad | 3302 | 0.096 | 75.601 |
| bmsklansky_cascad | 3291 | 0.096 | 74.9 |
| bmkogge_cascad | 3334 | 0.096 | 75.381 |
| | | | |
| | | | |
| amrcs_cascad | 547 | 0.083 | 92.113 |
| amcsa_cascad | 577 | 0.079 | 92.335 |
| amcla_cascad | 546 | 0.083 | 89.307 |
| amcsia_cascad | 528 | 0.083 | 88.94 |
| ambrentkung_cascad | 567 | 0.324 | 93.821 |
| amhancarlson_cascad | 719 | 0.324 | 92.81 |
| amharris_cascad | 707 | 0.324 | 93.437 |
| amknowles_cascad | 815 | 0.084 | 88.759 |
| amladnerficsher_cascad | 595 | 0.083 | 89.87 |
| amsklansky_cascad | 603 | 0.083 | 89.582 |
| amkogge_cascad | 641 | 0.083 | 89.946 |
| | | | |
| | | | |
| vmrca_cascad | 561 | 0.084 | 88.534 |
| vmcsa_cascad | 561 | 0.084 | 89.199 |
| vmcla_cascad | 562 | 0.084 | 89.202 |
| vmcsia_cascad | 555 | 0.084 | 89.001 |
| vmbrentkung_cascad | 555 | 0.324 | 94.131 |
| vmhancarlson_cascad | 587 | 0.324 | 94.377 |
| vmharris_cascad | 684 | 0.324 | 92.749 |
| vmknowles_cascad | 758 | 0.325 | 93.39 |
| vmladnerficsher_cascad | 573 | 0.324 | 94.469 |
| vmsklansky_cascad | 562 | 0.324 | 94.634 |
| vmkogge_cascad | 660 | 0.084 | 91.473 |

**Table-4.2**

**Area:**

The areas based on the no.of LUTs are shown in the fig-4.4

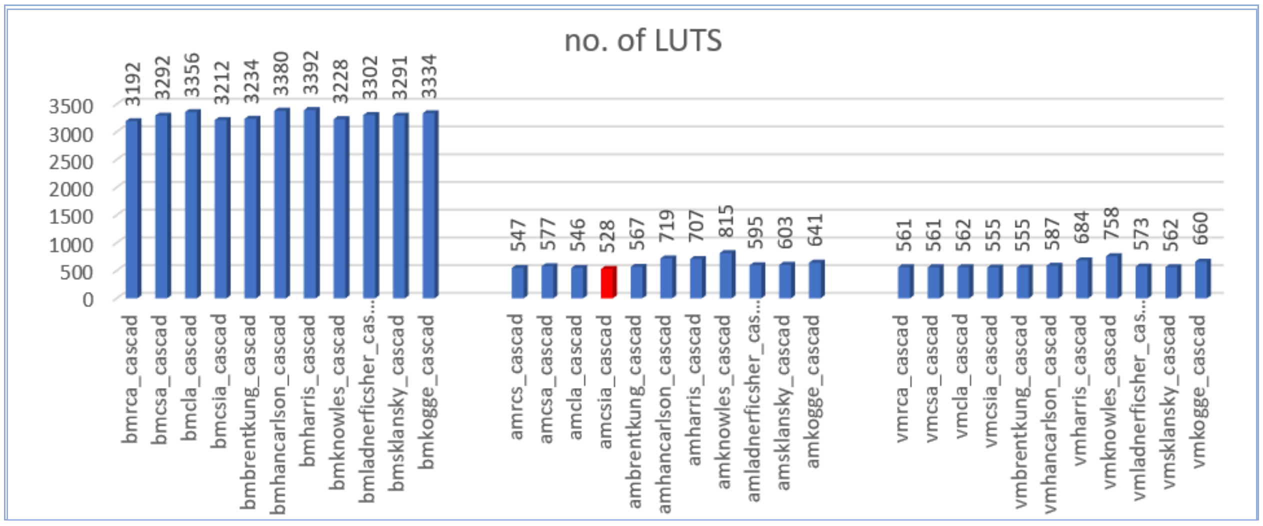


**Fig-4.4**

**Power:**

The power consumption of all the combinations of filters implemented shown in fig-4.5

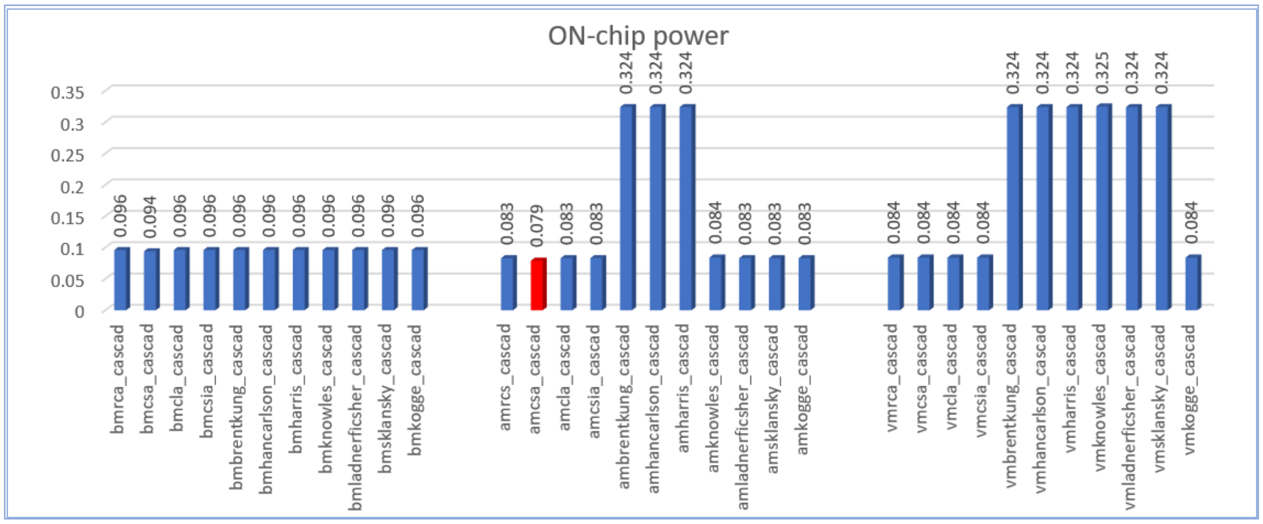


**Fig-4.5**

**Delay:**

The delay of all the filters are presented below in the fig-4.6

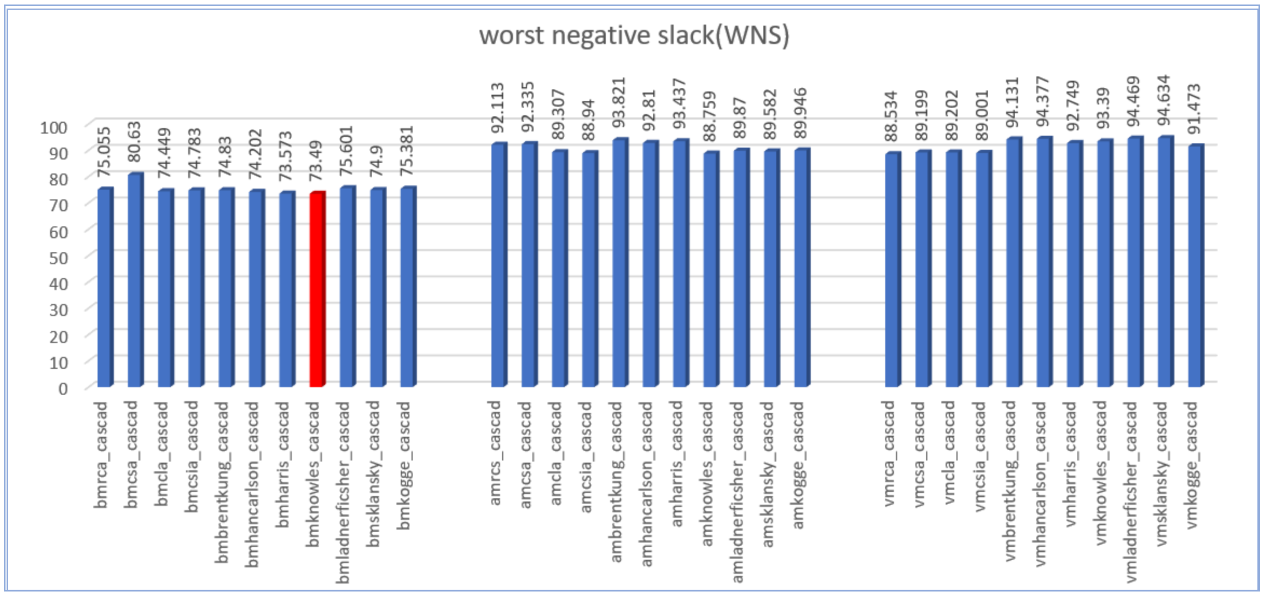


**Fig-4.6**

**Discussion:**

The least area is consumed by the FIR filter made with the combination of array multiplier along with carry skip adder as it is made with only 528 adders. Array multiplier with carry select adder would result in the FIR filter that consumes least power, in cascaded form, of only 0.079. The least delay of 73.49 can be noticed to be taken by the FIR filter implemented with the combination of booth multiplier and Knowles adder.

# CHAPTER 5

# CONCLUSION AND FUTURE WORKS

## 5.1 Conclusion

The proposed system analyzes all the combinations of different address and multipliers. This could help us. Pick the best combinations of adders and multipliers that would result in the efficient formation of a FIR filter. This work can be implemented in any application that uses digital signals, as this makes the chip design efficient with respect to all the parameters taken into consideration like power consumption, area occupied by the design and the delay aspects of the design, all of which can be crucial to determine the efficiency. These are applied in bio-medicinal field, one of all such applications in the field is ECG signal processing that would be very helpful, and any percentage of increase in the efficiency can help with the immediate diagnosis and treatment for a patient and further saves their lives. This also. Can be used to constantly monitor seismic activities. They are also applied to use in all forms of audio technology. Noise cancellation can be crucial at times for military purposes where the communication cannot be done as a normal in land communication. Because it cannot be regulated as much in some remote areas.

## 5.2 Future Works

FIR filters have a wide range of real-life implementations most of which are not only commercial products but also in crucial professional works across various fields. They are used in Noise cancelling and all other technologies of audio processing. These can be used in military applications and also medical technology equipment used for processing ECG signals. Further extensions of this project can be implemented by considering other forms of FIR filters like Linear, direct form-2, and other forms of FIR filters and get a deeper understanding of the efficiency of the filters. The use of Cadence tool would help us achieve better results but as the software is rarely available, it is hard to go through that process.

# CHAPTER 6

# REFERENCES

[1] Anubhuti Mittal , Ashutosh Nandi , Disha Yadav,: "Comparative study of 16-order FIR filter design using different multiplication techniques" ,ET Circuits Devices Syst., 2017, Vol. 11 Iss. 3, pp. 196-200.

[2] Sumalatha Madugula, Panchala Venkata Naganjaneyulu,Kodati Satya Prasad,: "Implementation of FIR Filter for Low Power and Area Minimization Using Shift –Add Method without Multipliers", International Journal of Intelligent Engineering and Systems, Vol.10, No.6, 2017.

[3] S. Bose, A. De and I. Chakrabarti, "Area-Delay-Power Efficient VLSI Architecture of FIR Filter for Processing Seismic Signal," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 11, pp. 3451-3455, Nov. 2021, doi: 10.1109/TCSII.2021.3081257.

[4] Udaya Kumar N ,Subbalakshmi U, Surya Priya B,Bala Sindhuri K,: "VLSI Implementation of FIR Filter Using Different Addition and Multiplication Techniques", International Conference on Soft Computing Systems, ICSCS 2018: Soft Computing Systems pp 483-490.

[5] Madhu Gudhimetla, CM Ananda,: "Comparison Of Different Types Of Multipliers With Respect To Speed, Area And Power", Proceedings Of 35 th IRF International Conference, 06th August, 2017, Bengaluru, India

[6] R. P. Rajput and M. N. S. Swamy, "High Speed Modified Booth Encoder Multiplier for Signed and Unsigned Numbers," 2012 UKSim 14th International Conference on Computer Modelling and Simulation, 2012, pp. 649-654, doi: 10.1109/UKSim.2012.99.

[7] Aarti Sharma , Sanjay Kumar,: "VLSI Implementation Of Pipelined FIR Filter", International Journal Of Innovative Research In Electrical, Electronics, Instrumentation And Control Engineering Vol. 1, Issue 5, August 2013

[8] Bala Sindhuri Kandula, K Padma Vasavi, I Santi Prabha,: "Design and Implementation of FIR Filter using Efficient MAC", International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-9 Issue-3, January 2020

[9] A. Radha, K.S.N. Murthy,: "Design of a High Speed and Area Efficient Novel Adder for AES Applications", International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-2, July 2019.

[10] Jaya Gupta, Arpan Shah, Ramesh Bharti,: “VLSI Implementation and Performance Evaluation of Low Pass Cascade & Linear Phase FIR Filter”, International Journal of Engineering and Technical Research (IJETR) ISSN: 2321-0869, Volume-3, Issue-6, June 2015.

[11] D. Bharti and K. N. Gupta, "Efficient design of different forms of FIR filter," 2014 International Conference on Recent Trends in Information Technology, 2014, pp. 1-4, doi: 10.1109/ICRTIT.2014.6996204.

[12] B. K. Mohanty and S. K. Patel, "Area–Delay–Power Efficient Carry-Select Adder," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 61, no. 6, pp. 418-422, June 2014, doi: 10.1109/TCSII.2014.2319695.

[13] Muthyala Sudhakar, Sreenivaas; Chidambaram, Kumar P.; Swartzlander, Earl E. (2012). [IEEE 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS) - Boise, ID, USA (2012.08.5-2012.08.8)] 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS) - Hybrid Han-Carlson adder. , (), 818–821. doi:10.1109/MWSCAS.2012.6292146

[14] Moghaddam, M.; Ghaznavi-Ghoushchi, M. B. (2011). [IEEE IEEE EUROCON 2011 - International Conference on Computer as a Tool - Lisbon (2011.04.27-2011.04.29)] 2011 IEEE EUROCON - International Conference on Computer as a Tool - A new low-power, low-area, parallel prefix Sklansky adder with reduced inter-stage connections complexity. , (0), 1–4. doi:10.1109/eurocon.2011.5929280

[15] R.Gowrishankar, Dr.N.Sathish Kumar, Dr.B.Senthilkumar, “ Performance Evaluation of New Adder Designed for Electronic Circuits” Brent kung, Knowles and Kogge Stone. International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-3, September 2019. DOI:10.35940/ijrte.C5681.098319.

## BIODATA

| | | |
|---|---|---|
| Name | : | Pinjala Sri Harsha |
| Mobile no | : | 8106802543 |
| Email | : | sriharshapinjala@gmail.com |
| Address | : | Chirala, Prakasam, Andhra Pradesh |

| | | |
|---|---|---|
| Name | : | Sudanagunta Naga Dheeraj |
| Mobile no | : | 9666290267 |
| Email | : | dheerajsudanagunta19@gmail.com |
| Address | : | Ongole, Prakasam, Andhra Pradesh |

| | | |
|---|---|---|
| Name | : | Vadlamudi Shanmukh Pranith |
| Mobile no | : | 9940439166 |
| Email | : | shanmukhpranith@gmail.com |
| Address | : | Nellore, Andhra Pradesh |