ACO Based FIR Filter Implementation on FPGA

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Ch. Rohit (318126512127)
Ch. Phani (318126512128)
N. Pujitha Vaidya (318126512161)
Y. Sarvendra (318126512179)

Under the guidance of
Dr.S. Srinivas
Associate Professor

PRAGNANAM BRAHMA
ANITS

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 'B+' Grade*)
Sangivalasa, Bheemili mandal, Visakhapatnam dist. (A.P)
2021-2022

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Dr.S.Srinivas,** Associate Professor, Department of Electronics and Communication Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. V. Rajya Lakshmi**, 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. Finally we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS:
Ch. Rohit (318126512127),
N.Pujitha Vaidya (318126512161),
Ch. Phani (318126512128),
Y. Sarvendra (318126512179)

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 'B+' Grade*)

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

PRAGNANAM BRAHMA

ANITS

CERTIFICATE

This is to certify that the project report entitled "**ACO Based FIR Filter Implementation on FPGA**" Submitted by **Ch. Rohit (318126512127), N. Pujitha Vaidya (318126512161), Ch. Phani (318126512128), Y. Sarvendra (318126512179)** in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Technology** in **Electronics & Communication Engineering** of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

[illegible]

Project Guide

Dr. S. Srinivas
Associate Professor
Department of E.C.E
ANITS

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

[illegible]

Head of the Department

Dr. V. Rajya Lakshmi
Professor & Head of the Department
Department of E.C.E
ANITS

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

CONTENTS

APPENDIX

ABSTRACT

In this modern era of rapid development, the usage of VERY LARGE-SCALE INTEGRATION (VLSI) architecture is in high demand and because of this rapid advancement in VLSI design posed a number of challenges. It is important to optimize the usage of different design areas such as chip size, component separation, inter-connected length. FIR filter can be designed by formulation of specifications which are for a particular application requirement. In this project, an efficient FIR filter will be designed using ANT COLONY OPTIMIZATION ALGORITHM. Also, an optimization environment will be designed such that filter components are upgraded on the VLSI design metrics such as area, speed or power and synthesizable code in Hardware Description Language [HDL] will be generated.

PURPOSE: This project aims to concentrate on an efficient FIR filter architecture in combination with the differential evolution ant colony algorithm (DE-ACO). For the design of FIR filter, ACO is found to be very efficient because of its non-conventional, nonlinear, multi-modal and non-differentiable nature. It converges to find the optimal final solution.

APPROACH: FIR filters are extensively used for many low power, low complexities, less area and high-speed digital signal processing applications since it can assure stability and recognize linear phase characteristics. Ant Colony algorithms form a class of proposed metaheuristics for solving difficult optimization problems.

KEYWORDS: VLSI DESIGN, DE-ACO, FIR FILTER, XILINX ISE, METAHEURISTIC.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Finite Impulse Response (FIR) filters are extensively used in digital signal processing applications since it can assure stability and recognize linear phase characteristics. The circuit scale tends to be large while implementing high order FIR filter in hardware which increases the design cost and power consumption. The number of digits involved without zero in each coefficient corresponds to multiple shifters in the multiplier. Various heuristic methods have been recommended to solve this problem in the equitable computation time and applied. Artificial bee colony (ABC) and particle swarm optimization (PSO) are the approaches applied to a continuous optimization problem. The difficulty of the existing technique is solved by using a useful ant colony optimization (ACO) algorithm. ACO, genetic algorithm (GA), simulated annealing and other heuristic methods are known to generate acceptable solutions within a possible time. A combinatorial optimization problem has been solved by the proposed ACO, and it has demonstrated excellent performance in comprehensive problems. The design of high-order discrete coefficient FIR filters is a large-scale problem, and the application of ACO can be very effective in solving the above problem. The Z-plane transfer function of FIR filter with only zeros and linear phase described by is used to find out the magnitude and frequency response. The FIR filter has N number of points, and its z-transform is given in eq. (1):

$$H(z) = \sum_{n=0}^{N-1} h(n) z^{-n} \qquad (1)$$

There is no feedback required for FIR filters. At present, ACO is widely used for various optimization applications such as optimal design and scheduling problem of thermal units, traveling salesman problem and quadratic assignment problem. This paper presents a novel DEACO technique for optimizing power consumption. The research comprises of the establishment of novel algorithm termed as DE-ACO which combines state of the art to the power consumption and frequency domain specifications optimizing.

1.2 BACKGROUND WORK

Fir filters are stable, linear and easy to implement and hence they are widely used in many consumer electronics and DSP applications. But, the main problem here is that the FIR takes more time to execute and since there is no feedback it needs more number of coefficients. For every extra coefficient there will be an extra memory requirement and hence for a demanding system, the speed and memory requirements to implement an FIR system can make the system unfeasible.

So, our project aimed to reduce the time of execution by minimizing the number of coefficients needed. This can be done using ANT COLONY OPTIMIZATION technique which reduces coefficients and gives only the required ones. So, our project aimed to focus on designing some efficient reconfigurable FIR filter architectures.

CHAPTER 2

INTRODUCTION TO FIR FILTERS

2.1 INTRODUCTION

The digital filter is a discrete system, and it can do a series of mathematic processing to the input signal, and therefore obtain the desired information from the input signal. The transfer function for a linear, time-invariant, digital filter is expressed in eq. (2)

$$H(z) = \frac{\sum_{J=0}^{M} b_j z^{-1}}{1 + \sum_{i=0}^{N} a_i z^{-1}} \qquad (2)$$

where a_i and b_j are coefficients of the filter in Z-transform

There are many kinds of digital filters, and also many different ways to classify them. According their function, the FIR filters can be classified into four categories, which are lowpass filter, high pass filter, bandpass filter, and band stop filter.

According to the impulse response, there are usually two types of digital filters, which are finite impulse response (FIR) filters and infinite impulse response (IIR) filters.

According to the formula above, if a_i is always zero, then it is a FIR filter, otherwise, if there is at least one none-zero a_i, then it is an IIR filter. Usually, we need three basic arithmetic units to design a digital filter, which are the adder, the delay, and the multiplier.

The following are the steps to design a digital filter:

1. Make sure of the property of a digital filter according to the given requirements.
2. Use a discrete linear time-invariant system function to approach to the properties.
3. Make use of algorithms to design the system function.
4. Use a computer simulation or hardware to achieve it.

2.2 FIR FILTER

FIR filters are stable, linear and easy to implement and hence they are widely used in many consumer electronics and DSP applications [8]. The finite impulse response (FIR) filter is one of the most basic elements in a digital signal processing system, and it can guarantee a strict linear phase frequency characteristic with any kind of amplitude frequency characteristic. Besides, the unit impulse response is finite; therefore, FIR filters are stable system. The FIR filter has a broad application in many fields, such as telecommunication, image processing, and so on. The system function of FIR filter is given by eq. (3):

$$H(z) = \sum_{n=0}^{L-1} h[n]z^{-n} \qquad (3)$$

where L is the length of the filter, and h$[n]$ is the impulse response.

2.3 IIR FILTER

The Infinite Impulse Response (IIR) filter is recu rsive structure, and it has afeedback loop. The precision of amplitude frequency characteristic is very high, and IIR filters are not linear phase.

2.4 FIR AND IIR COMPARISON

(1) Under the same conditions as in the technical indicators, output of the IIR filterhas feedback to input, so it can meet the requirements better than FIR. The storage units are less than those of IIR, the number of calculations is also less, and it’s more economical.

(2) The phase of FIR filter is strictly linear, while the IIR filter is not. The better the selectivity of IIR filter is, the more serious the nonlinearity of the phase will be.

(3) The FIR filter is non-recursive structure, finite precision arithmetic error is very small. While IIR filter is recursive structure, and parasitic oscillation may occur in theoperation of IIR filter.

(4) Fast Fourier Transformation can be used in FIR filter, while IIR cannot.

(5)The IIR filter can use formulas, data and tables of the analog filter, andonly a small amount of calculation. While FIR filter design may always make use of the computer to calculate, and the order of FIR filter could be large to meet the design specifications.

2.5 DEFINITION OF FIR FILTER

In signal processing, a finite impulse response (FIR) filter from (figure1) is a filter whose impulse response (or response to any finite length input) is of *finite* duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

The impulse response of an Nth-order discrete-time FIR filter (i.e., with a Kronecker delta impulse input) lasts for N + 1 samples, and then settles to zero.
FIR filters can be discrete-time or continuous-time, and digital or analog.

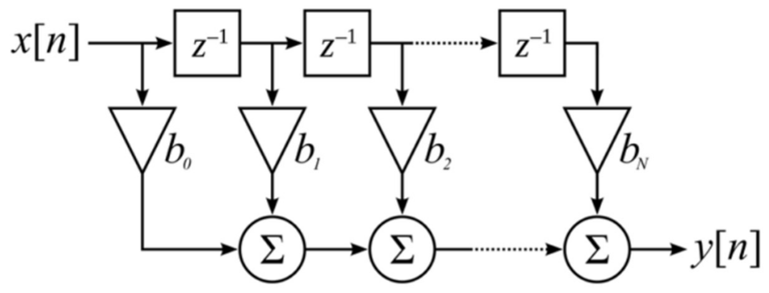


Figure 1: FIR Filter Block Diagram

A discrete-time FIR filter of order N. The top part is an N-stage delay line with N + 1 taps. Each unit delay is a z−1 operator in the Z-transform notation. The output y of a linear time invariant system is determined by convolving its input signal x with its impulse response

For a discrete-time FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. The operation is described by the following equation, which defines the output sequence y[n] in terms of its input sequence x[n] as in eq. (4):

$$y[n] = b_0\, x[n] + b_0\, x[n-1] + \cdots + b_0\, x[n-N] = \textstyle\sum_{i=0}^{N} b_i\, x[n-1] \qquad (4)$$

Where, x[n] is the input signal, y[n] is the output signal, b(i) are the filter coefficients, also known as tap weights, that make up the impulse response and N is the filter order; an th-order filter has (N+I) terms on the right-hand side. The x(n-i) in these terms is commonly referred to as taps, based on the structure of a tapped delay line that in many implementations or block diagrams provides the delayed inputs to the multiplication operations. One may speak of a 5th order/6-tap filter, for instance.

2.6 FIR PROPERTIES

An FIR filter has a number of useful properties which sometimes make it preferable to an infinite impulse response (IIR) filter. FIR filters:

Require no feedback: This means that any rounding errors are not compounded by summed iterations. The same relative error occurs in each.

Inherent stability: This is due to the fact that, because there is no required feedback, all the poles are located at the origin and thus are located within the unit circle (the required condition for stability in a Z transformed system).

Phase Issue: can easily be designed to be linear phase by making the coefficient sequence symmetric; linear phase, or phase change proportional to frequency, corresponds to equal delay at all frequencies. This property is sometimes desired for phase-sensitive applications, for example data communications, crossover filters, and mastering.

The main disadvantage of FIR filters is that **considerably more computation power** in a general purpose processor is required compared to an IIR filter with similar sharpness or selectivity, especially when low frequency (relative to the sample rate) cutoffs are needed. However, many digital signal processors provide specialized hardware features to make FIR filters approximately as efficient as IIR for many applications.

2.7 FIR APPLICATIONS

FIR applications mainly involve in digital communications in the intermediate frequency stages of the receiver. For instance, a digital radio receives and converts the analog signal to the intermediate frequency and then converts it to digital using with a digital to analog converter. Then uses the finite impulse response to choose the preferred frequency. It is used in software radio, that permits easily adaptable filters with good rejection and without changing hardware.

Spatial Beamforming:

It is a signal processing technique used in sensor arrays for directional signal transmission or reception.[1] This is achieved by combining elements in an antenna array in such a way that signals at particular angles experience constructive interference while others experience destructive interference. Beamforming can be used at both the transmitting and receiving ends in order to achieve spatial selectivity. The improvement compared with omnidirectional reception/transmission is known as the directivity of the array. Beamforming can be used for radio or sound waves. It has found numerous applications inradar, sonar, seismology, wireless communications, radio astronomy, acoustics and biomedicine. Adaptive beamforming is used to detect and estimate the signal of interest atthe output of a sensor array by means of optimal (e.g. least-squares) spatial filtering and interference rejection.

Linear Predictive Coding

It is a method used mostly in audio signal processing and speech processing for representing the spectral envelope of a digital signal of speech in compressed form, using the information of a linear predictive model. It is one of the most powerful speech analysis techniques, and one of the most useful methods for encoding good quality speech at a low bit rate and provides highly accurate estimates of speech parameters.

Linear Interpolation

In mathematics, linear interpolation is a method of curve fitting using linear polynomials to construct new data points within the range of a discrete set of known data points.

Speech Analysis

Speech analysis is the process of analyzing the speech signal to obtain relevant informationof the signal in a more compact form than the speech signal itself.

Speech Modelling

Modeling is the process of utilizing your complete speech and language system to help your child's developing speech and language system grow

Multi rate Signal processing

Iterate simply means "multiple sampling rates". A multirate DSP system uses multiple sampling rates within the system. Whenever a signal at one rate has to be used by a system that expects a different rate, the rate has to be increased or decreased, and some processing isrequired to do so.

Averaging Filter

A Special implementation of a low pass algorithm is the averaging filter. It calculates the output sample using the average from a finite number of input samples. The averaging filter is used in situations where is necessary to smooth data that carrying high frequency distortion.

2.8 ADVANTAGES OF FIR FILTERS

FIR filter is always stable and simple. FIR filters have linear phase response. It is easy to optimize and is Noncausal. Round of noise error is minimum. Both recursive, as well as non-recursive filter, can be designed using FIR designing techniques.

For designing a filter having any arbitrary magnitude response, FIR designing techniques can be easily applied and it gives good performance, robust in nature and the ease of computational techniques for filter implementation. It has the requirement of large storage and the incapability of linear phase response.

2.9 DISADVANTAGES OF FIR FILTERS

Require large storage requirements and cannot simulate prototype analog filter. For the implementation of FIR filter complex computational techniques are required, it is hard to implementation than IIR and is expensive due to large order, require more memory and a time-consuming process.

CHAPTER 3

MODIFIED DIFFERENTIAL EVOLUTION ANT COLONY OPTIMIZATION ALGORITHM

3.1 INTRODUCTION

The algorithmic world is beautiful with multifarious strategies and tools being developed round the clock to render to the need for high-performance computing. In fact, when algorithms are inspired by natural laws, interesting results are observed. Evolutionary algorithms belong to such a class of algorithms. These algorithms are designed so as to mimic certain behaviours as well as evolutionary traits of the human genome. Moreover, such algorithmic design is not only constrained to humans but can be inspired by the natural behaviour of certain animals as well. The basic aim of fabricating such methodologies is to provide realistic, relevant and yet some low-cost solutions to problems that are hitherto unsolvable by conventional means.

In computer science and operations research, the **Ant Colony Optimization** algorithm (**ACO**) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. Artificial ants stand for multi-agent methods inspired by the behaviour of real ants. The pheromone-based communication of biological ants is often the predominant paradigm used. Combinations of artificial ants and local search algorithms have become a method of choice for numerous optimization tasks involving some sort of graph, e.g., vehicle routing and internet routing.

Different optimization techniques have thus evolved based on such evolutionary algorithms and thereby opened up the domain of metaheuristics. **Metaheuristic** has been derived from two Greek words, namely, **Meta** meaning **one level above** and **heuriskein** meaning **to find**. Algorithms such as the Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) are examples of swarm intelligence and metaheuristics. The goal of swarm intelligence is to design intelligent multi-agent systems by taking inspiration from the collective behaviour of social insects such as ants, termites, bees, wasps, and other animal societies such as flocks of birds or schools of fish.

In the natural world, ants of some species (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but instead to follow the trail, returning and reinforcing it if they eventually find food.

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. The influence of pheromone evaporation in real ant systems is unclear, but it is very important in artificial systems.

The overall result is that when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to many ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve.

3.2 WORKING OF ACO

Ant Colony Optimization technique is purely inspired from the **foraging** behaviour of ant colonies, first introduced by Marco Dorigo in the 1990s. Ants are eusocial insects that prefer community survival and sustaining rather than as individual species. They communicate with each other using sound, touch and pheromone. **Pheromones** are organic chemical compounds secreted by the ants that trigger a social response in members of same species. These are chemicals capable of acting like hormones outside the body of the secreting individual, to impact the behaviour of the receiving individuals. Since most ants live on the ground, they use the soil surface to leave pheromone trails that may be followed (smelled) by other ants.

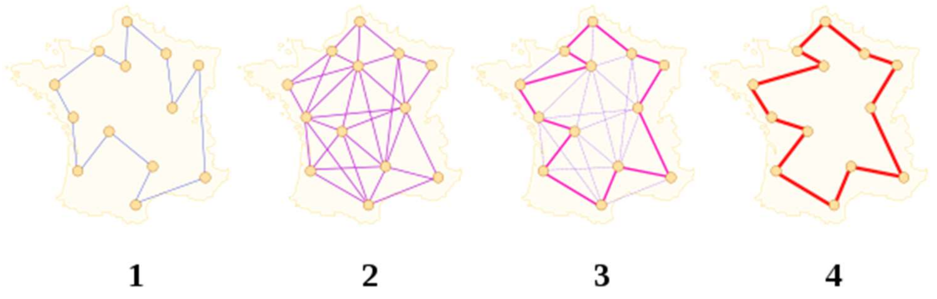


Figure 2: ACO Algorithm for Shortest Path

Ants live in community nests and the underlying principle of ACO is to observe the movement of the ants from their nests in order to search for food in the shortest possible path from (figure 2). Initially, ants start to move randomly in search of food around their nests. This randomized search opens up multiple routes from the nest to the food source. Now, based on the quality and quantity of the food, ants carry a portion of the food back with necessary pheromone concentration on its return path. Depending on these pheromone trials, the probability of selection of a specific path by the following ants would be a guiding factor to the food source from (figure 3). Evidently, this probability is based on the concentration as well as the rate of evaporation of pheromone. It can also be observed that since the evaporation rate of pheromone is also a deciding factor, the length of each path can easily be accounted for.

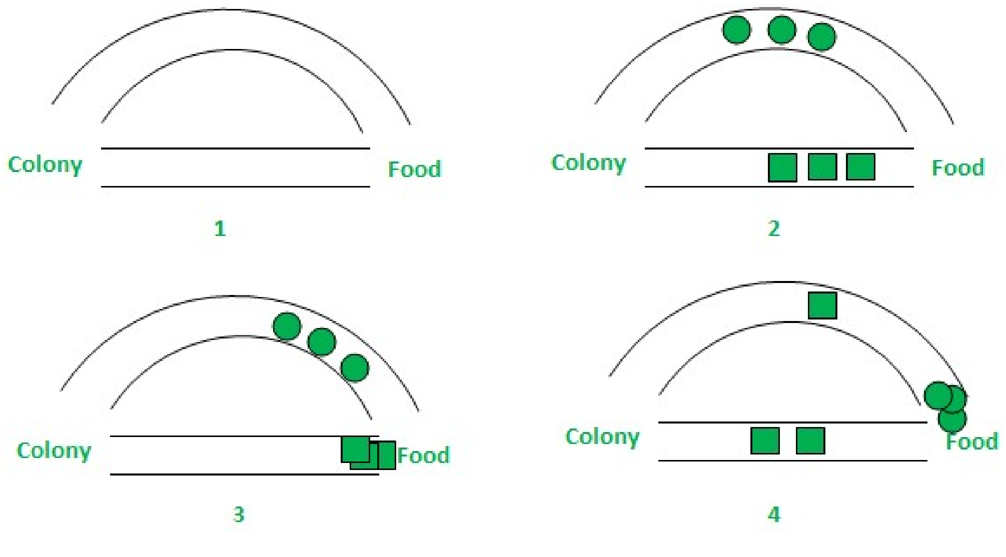


Figure 3: Different Stages of Ant Colony Algorithm

Stage 1: All ants are in their nest. There is no pheromone content in the environment. (For algorithmic design, residual pheromone amount can be considered without interfering with the probability).

Stage 2: Ants begin their search with equal (0.5 each) probability along each path. Clearly, the curved path is the longer and hence the time taken by ants to reach food source is greater than the other.

Stage 3: The ants through the shorter path reaches food source earlier. Now, evidently they face with a similar selection dilemma, but this time due to pheromone trail along the shorter path already available, probability of selection is higher.

Stage 4: More ants return via the shorter path and subsequently the pheromone concentrations also increase. Moreover, due to evaporation, the pheromone concentration in the longer path reduces, decreasing the probability of selection of this path in further stages. Therefore, the whole colony gradually uses the shorter path in higher probabilities. So, path optimization is attained.

3.3 ALGORITHM AND FORMULA

The standard approach for solving an optimization problem is to design an objective function which can replace the problem's objectives while limited to its constraints. The DE algorithm is a heuristic algorithm based on the population like GAs using the three operators; **Mutation, Crossover and Selection** to optimize the cost function or detached function over the successive generations.

DIFFERENTIAL EVOLUTIONARY algorithm is a heuristic algorithm which follows 3 steps for solving an optimized problem.

MUTATION: Minimizes power consumption and reduces ripple errors of pass and stop band. Due to which, ants select components according to previous phermone distribution and releases pheromone on components they select.

CROSSOVER: New child is created by mutation of parents, which are selected in random, bunch of 4 each time.

SELECTION: To select the shortest path which gives the required nodes.

As for the linear phase digital FIR filter design, the inputs of the algorithm are a group of coefficients {h (0), h (1), ..., h (N - 1)}. Moreover, at the end of the optimization, only one group with optimum fitness value will be obtained. In every generation, to produce a trial vector each vector of the parent parameter is targeted for the crossover with a vector. The resultant trial vector is called a child of the two parent vectors, and it will compete with the target vector in the following steps. CR is the probability of crossover which the parameter values of fraction controls and succeeded by the mutant. After mutation and crossover operations violate the bounds; it is limited, and likewise the bounds are regulated. The child's vectors are used to calculate the objective function values.

ACO is used to calculate the shortest path between source and destination. The two functions are mainly done by the ant that is in drift until finding the source of food (F) and then back to the nest (N) depending on the pheromone trail on random. It is a chemical substance that is released by ants when it goes into the search of food. Pheromone trails discover the shortest path. Pheromones show in some path the common storage. It remains extremely elementary. The optimum solutions from a finite number of solutions are calculated by combinatorial optimization. It works on the domain of those optimization problems that have set of appropriate solutions that will be discrete. Better results can be obtained for lesser appropriate discrete solutions.

Compute probability P according to the probability equation using eq. (5):

$$P_{n,k}^{lm}(t) = \frac{\tau^{l1}n,k(l)}{\tau^{l1}n,k(t)+\tau^{l0}n,k(l)+\tau^{l1}n,k(l)} \qquad (5)$$

According to pheromone updation, the equation is eq. (6):

$$\tau^{l1}n,k(t+1) = \rho\tau^{l1}n,k(t) + \sum_{u-1}^{\sigma-1}\Delta\tau_{n,k}^{lm,u}(t) + \Delta\tau_{n,k}^{lm,*}(t) \qquad (6)$$

The desired frequency domain specifications and power minimization are the dual objectives to be considered while designing a filter. The proposed DE-ACO has a more excellent performance with strong ability to find optimal solution and quick convergence speed. The contribution of the present work is three-fold, firstly a novel optimization technique based on the hybridization of a heuristic technique, i.e., ant colony algorithm and an Evolutionary technique, i.e., DE algorithm is proposed, which conserves the individual advantages of both the techniques, i.e., it improves the effectiveness; reduce the design error (ACO) and speeds up execution process (DE), while discarding their limitations. The design of FIR filter is based on the approach of optimization. It satisfies the dual objectives of minimizing power consumption during implementation and reducing ripples in the stop band and pass band. In this approach, ants in colony randomly select components according to previous pheromone distribution and release pheromone on the components they select. The ants will eventually converge on a set of components, thus completing the optimization. Secondly, a hybrid of DE and ACO gives a new method of optimization called the DE-ACO. In DEACO, new offspring is created by the mutation of the parent. In this work, gbest is taken as the parent. Gaussian distribution has been considered. For mutation, four particles are chosen in random from the population. The mutation of the parent creates an offspring using the weighted error between these particles' positions. Weighted error between the position of child particles is eq. (7):

$$If\ (rand_{()} < CR\ OR\ d == k)$$
$$then\ T_{id} = P_{gd} + \delta_{2,d}$$
$$\delta_{2,d} = \frac{(P_{1,d} - P_{2,d}) + (P_{3,d} - P_{4,d})}{2} \quad (7)$$

3.4 PSEUDO CODE FOR DE-ACO ALGORITHM

```
For each particle
     Initialize particle
     End
     Do
     For each particle
     Calculate Fitness value
     If the fitness value is better than the
     best fitness value
      (pbest) in memory
     Set current value as the new pbest
     Endif
     End
     Choose the particle with the best fitness
   value of all the particles as the gbest
     For each particle
     Calculate Pheromone update using (11)
     Calculate pheromone trial updating of
   ants using (12)
     End For each particle in parent set,
     Select 4 (or 6 for Gaussian distribution)
   particles
     Evaluate weighed differences
```

```
Mutate the parent
End
For each particle in offspring set,
Calculate the fitness
If fitness(parent) < fitness(offspring) then
Replace parent with offspring
End
```

3.5 APPLICATIONS

Ant colony optimization algorithms have been applied to many combinatorial optimization problems, ranging from quadratic assignment to protein folding or routing vehicles and a lot of derived methods have been adapted to dynamic problems in real variables, stochastic problems, multi-targets and parallel implementations. It has also been used to produce near-optimal solutions to the travelling salesman problem. They have an advantage over simulated annealing and genetic algorithm approaches of similar problems when the graph may change dynamically; the ant colony algorithm can be run continuously and adapt to changes in real time. This is of interest in network routing and urban transportation systems. The first ACO algorithm was called the ant system and it was aimed to solve the travelling salesman problem, in which the goal is to find the shortest round-trip to link a series of cities. The general algorithm is relatively simple and based on a set of ants, each making one of the possible round-trips along the cities. At each stage, the ant chooses to move from one city to another according to some rules:

1. It must visit each city exactly once;
2. A distant city has less chance of being chosen (the visibility);
3. The more intense the pheromone trail laid out on an edge between two cities, the greater the probability that that edge will be chosen;
4. Having completed its journey, the ant deposits more pheromones on all edges it traversed, if the journey is short;
5. After each iteration, trails of pheromones evaporate.

CHAPTER 4

FIR FILTER USING DE-ACO ALGORITHM

4.1 DESIGN OF FINITE IMPULSE RESPONSE FILTER

Finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration as in eq. (8), because it settles to zero in finite time. The output of the FIR filters depends on the present, and past input values as in eq. (9): and these filters are named as non-recursive filters [1]. The multipliers, series of delays, and adders are used to implement the FIR filter to create the filter's output from (figure4). It is known as an all-zero filter because it does not have any poles. They are also known as feed forward or transversal filters.

$$h(n) = \begin{cases} b_{k,} 0 \leq n \leq N-1 \\ 0, otherwise \end{cases} \qquad (8)$$

h(n) represents impulse responses at n = 1,2,3, ...N – 1 and N is the order of the filter

$$y(n) = \sum_{k=0}^{N-1} b_k \, x(n-k) \qquad (9)$$

where y(n) represents output of fir filter

$$H(z) = \sum_{n=0}^{N-1} h(n) z^{-n} \qquad (10)$$

H(z) represents transfer function of the FIR filter from eq. (10)

FIR filter is designed on the ideal filter approximation according to given design conditions. The following parameters can specify an FIR digital filter: the orders of the filter = 15, pass band cut-off frequency = 200kHz, stop band cut-off frequency = 355kHz, by using this stop band and pass band ripple errors are minimized from (figure 5). The optimization is aiming to reduce the ripple error of the pass band and stop band [10] while keeping a sharp transition band.

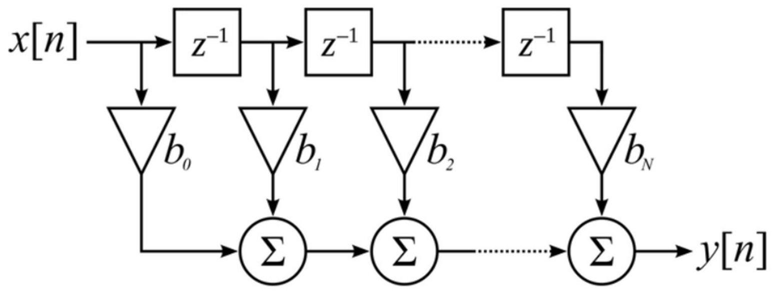


Figure 4. Basic Structure of FIR filter

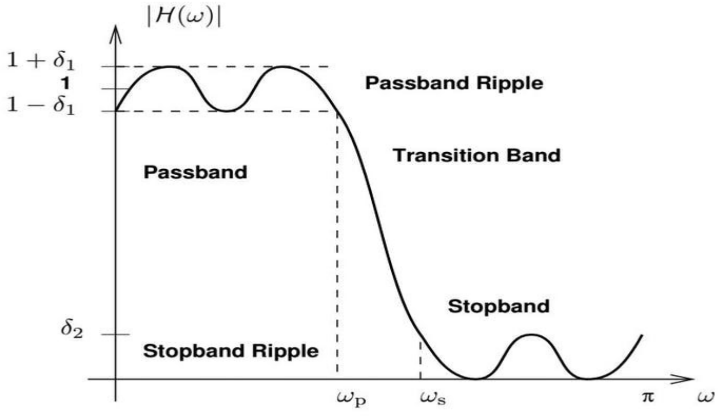


Figure 5.Frequency Response of Low Pass Filter

The main focus of this project is on the implementation of a FIR in HDL, which can be broken down into three main logic components

- A circular buffer to clock each sample into that properly accounts for the delays of the serial input
- Multipliers for each of the taps' coefficient value
- The accumulator registers for the summing result from each tap's output.

Coefficients optimization is the main and high-effective aspect which effects the output of the filter. Also, the cost function is introduced here. There are many ways to design Digital FIR filters, such as windowing functions, frequency sampling method, and non-linear optimization algorithm method. Evolving algorithms have been proved to be the digital FIR filter design. Moreover, the following part will introduce the two of them namely Ant Colony Algorithm (ACO) and DE to design FIR filters.

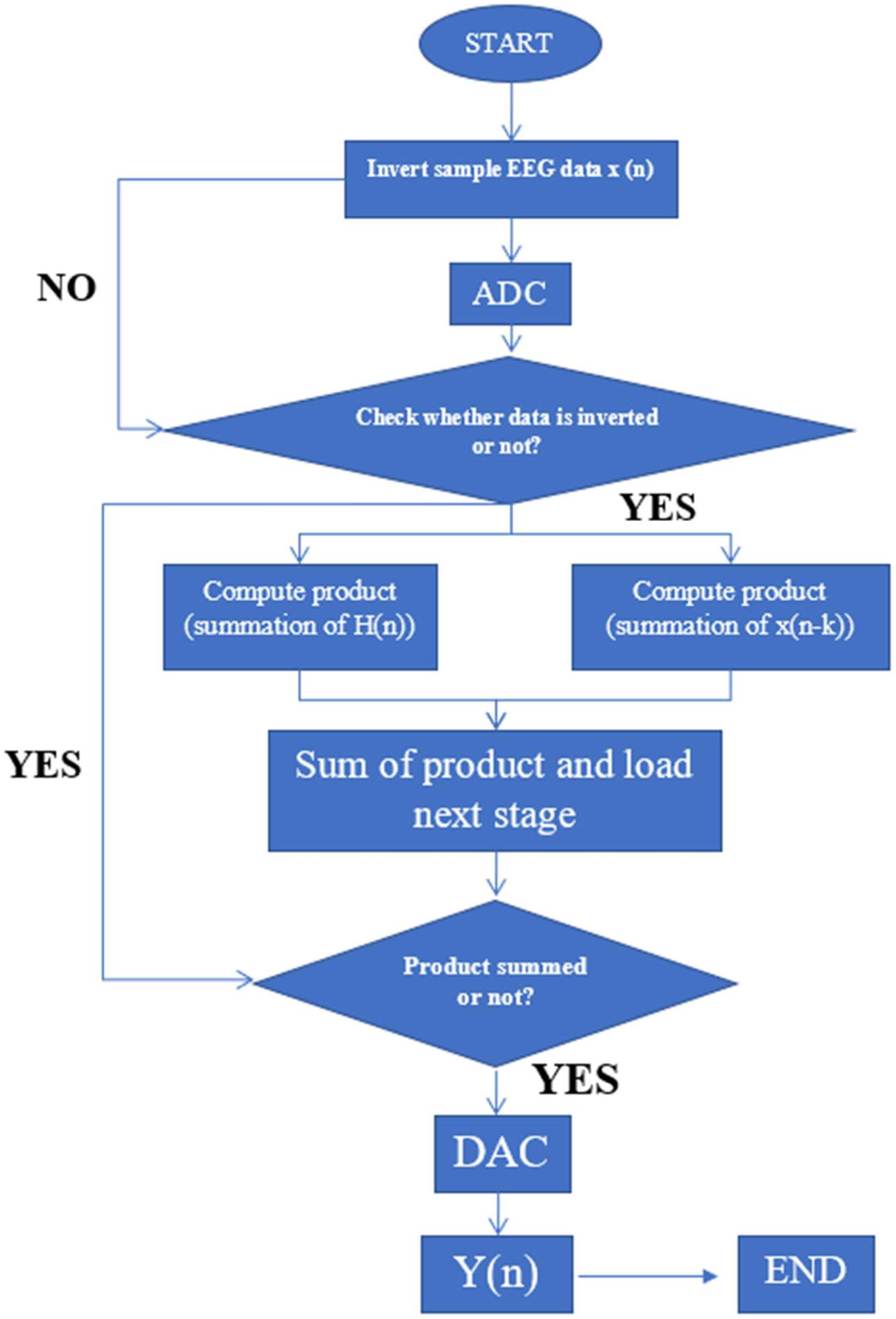

START
Invert sample EEG data x (n)
NO
ADC
Check whether data is inverted or not?
YES
Compute product (summation of H(n))
Compute product (summation of x(n-k))
YES
Sum of product and load next stage
Product summed or not?
YES
DAC
Y(n)
END

Figure 6.Flow Chart to Design FIR filter

4.2 FILTER OPTIMIZATION TECHNIQUES

The various iterative solutions are compared to get the optimum solution or satisfactory solution for a given application which is done by the optimization algorithm. The algorithms are deterministic which deal with specific rules for moving from one solution to the other. The algorithm which has the probabilistic transition rule is known as stochastic.

USING A DIFFERENTIAL EVOLUTIONARY ALGORITHM

The standard approach for solving an optimization problem is to design an objective function which can replace the problem's objectives while limited to its constraints. The DE algorithm is a heuristic algorithm based on the population using the three operators; **Mutation, Crossover and Selection** to optimize the cost function or detached function over the successive generations [3]. As for the linear phase digital FIR filter design, the inputs of the algorithm are a group of coefficients {h (0), h (1), h (2), ..., h(N-1)}. Moreover, at the end of the optimization, only one group with optimum fitness value will be obtained. In every generation, to produce a trial vector each vector of the parent parameter is targeted for the crossover with a vector. The resultant trial vector is called a child of the two parent vectors, and it will compete with the target vector in the following steps. After mutation and crossover operations violate the bounds; it is limited, and likewise the bounds are regulated [7]. The children vectors are used to calculate the objective function values.

A. Mutation operation

There are several versions of mutation operation. All of them aim to produce new values from the original ones. An initial mutation individual vector y_i is generated by eq. (11) randomly selecting three members of the population, x_{i1}, x_{i2} and x_{i3}. Then y_i is produced as

$$y_i = x_{i1} + F\,(x_{i2} - x_{i3}) \qquad (11)$$

Where F is a positive scale factor, effective values for which are typically less than one. All the individuals will experience the mutation until D mutations have been made.

The most useful strategies are:

"Best/1."

$$y_i = x_{best} + F\,(x_{i1} - x_{i2}) \qquad (11.1)$$

"Current to best/1."

$$y_i = x_i + F\,(x_{i1} - x_{i2}) + F\,(x_{best} - x_i) \qquad (11.2)$$

"Best/2."

$$y_i = x_{best} + F\,(x_{i1} - x_{i2}) + F\,(x_{i3} - x_{i4}) \qquad (11.3)$$

"Rand/2."

$$y_i = x_{i1} + F\,(x_{i2} - x_{i3}) + F\,(x_{i4} - x_{i5}) \qquad (11.4)$$

Where the index $i1$, $i2$, $i3$, $i4$, $i5$ are different with the current population i which represent the random different integers generated within range [1, NP], NP is the number of parameter vectors.

B. Crossover operation:

After the mutation operation, a value $v_{ji}{}^{G}$ is produced which is in the position of $x_{ji}{}^{G}$. Crossover operation is then applied to each pair of the target value $x_{ji}{}^{G}$ and its corresponding mutant value $v_{ji}{}^{G}$ to decide a new value: $u_{ji}{}^{G}$ from eq.(12) . DE employs the crossover as the basic version:

$$u_{ji}^{G} = \begin{cases} x_{ji}^{G}, & if\ rand < CR\ or\ j = jrand \\ v_{ji}^{G}, & otherwise \end{cases} \qquad (12)$$

CR is defined within the range (0.7, 1]. $jrand$ is a randomly chosen integer in the range [1, D].

C. Selection Operation

After the mutation and crossover operations, some new parameters values are added to the current population. Meanwhile, we will remove some of the members which exceed the search space. Then, the cost function values of all the vectors are calculated. After that, a selection operation is performed. The objective function value of each member ($uji\ G$) is compared to that of its corresponding target vector $f(xji\ G$) one by one in the current population. If the target vector has better objective function value than the corresponding trial vector, the target vector will maintain and the trial vector will not enter the population of the next generation. Otherwise, the trial vector will replace the target vector in the population for the next generation. The selection operation can be expressed as eq. (13):

$$x_{ji}^{G+1} = \begin{cases} u_{ji}^{G}, \ if\ f\left(u_{ji}^{G}\right) > f(x_{ji}^{G}) \\ x_{ji}^{G}, \ otherwise \end{cases} \qquad (13)$$

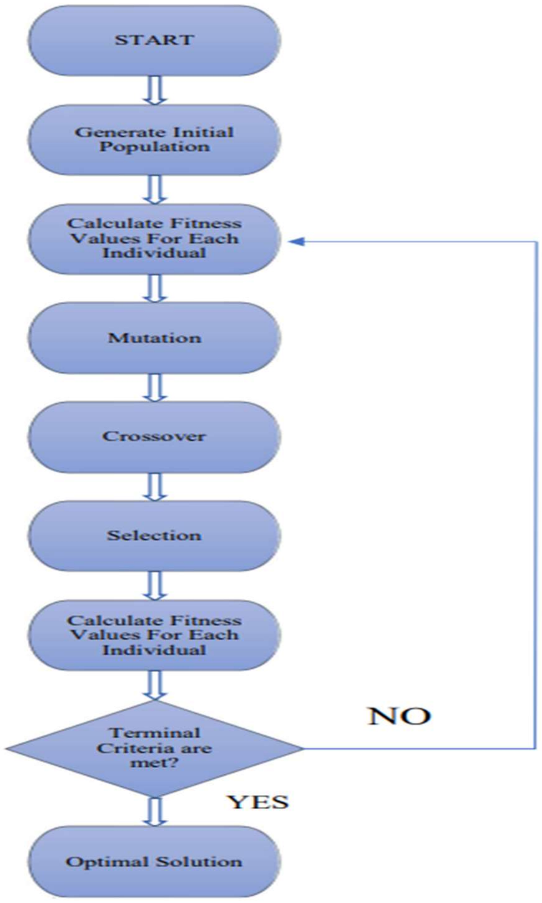

START
Generate Initial Population
Calculate Fitness Values For Each Individual
Mutation
Crossover
Selection
Calculate Fitness Values For Each Individual
Terminal Criteria are met?
NO
YES
Optimal Solution

Figure 7. Flow chart of DE Algorithm

Control parameters:

Suitable control parameters are important for DE algorithm to increase the speed of searching.
1) $F \in [0.5, 1]$
2) $CR \in [0.8, 1]$
1) $Np = 10\square D$
Liu and L Ampinen in [14] set control parameters to F =0.9, CR = 0.9.
In this chapter, we use the constant control parameter mechanism during the iteration and the control parameter Np keeps no change as well. Both F and CR are applied at the individual level. The original DE algorithm has three control parameters that need to be

adjusted by the user. Different parameters could have influenced in different function problems. When we implement a self-adaptive parameters mechanism in this design, there is no apparent effectiveness in the result. So we still adapt the constant values for this design.

USING ANT COLONY OPTIMIZATION ALGORITHM

ACO is used to calculate the shortest path between source and destination [4]. The two functions are mainly done by the ant that is in drift until finding the source of food and then back to the nest depending on the pheromone trail on random from eq. (14). It is a chemical substance that is released by ants when it goes into the search of food [6]. Pheromone trails discover the shortest path upon evaporation and reapplication of it which is updated frequently by eq. (15). Pheromones show in some path the common storage. The optimum solutions from a finite number of solutions are calculated by combinatorial optimization. It works on the domain of those optimization problems that have set of appropriate solutions that will be discrete. Better results can be obtained for lesser appropriate discrete solutions.

$$P_{n,k}^{lm}(t) = \frac{\tau^{l1}n,k(l)}{\tau^{l1}n,k(t)+\tau^{l0}n,k(l)+\tau^{l1}n,k(l)} \qquad (14)$$

$$\tau^{l1}n,k(t+1) = \rho\tau^{l1}n,k(t) + \sum_{u-1}^{\sigma-1}\Delta\tau_{n,k}^{lm,u}(t) + \Delta\tau_{n,k}^{lm,*}(t) \qquad (15)$$

4.3. DIFFERENTIAL EVOLUTION-ANT COLONY OPTIMIZATION ALGORITHMS

The desired frequency domain specifications and power minimization are the dual objectives to be considered while designing a filter. The proposed DE-ACO has a more excellent performance with strong ability to find optimal solution and quick convergence speed. The contribution of the present work is three fold process.

Firstly a novel optimization technique based on the hybridization of a heuristic technique, i.e. ant colony algorithm and an Evolutionary technique, i.e. DE algorithm is proposed, which conserves the individual advantages of both the techniques, i.e. it improves the effectiveness; reduce the design error (ACO) [5] and speeds up execution process (DE), while discarding their limitations. The design of FIR filter is based on the approach of optimization. It satisfies the dual objectives of minimizing power consumption during implementation and reducing ripples in the stop band and pass band.

In this approach, ants in colony randomly select components according to previous pheromone distribution and release pheromone on the components they select. The ants will eventually converge on a set of components, thus completing the optimization.

Secondly, a hybrid of DE and ACO gives a new method of optimization called the DE-ACO. In DEACO, new offspring is created by the mutation of the parent. In this work, gbest is taken as the parent. Gaussian distribution has been considered. For mutation, four particles are chosen in random from the population. The mutation of the parent creates an offspring using the weighted error between these particles' positions.

The mutation takes place according to:

$$If\ (rand_{()} < CR\ OR\ d == k)$$
$$then\ T_{id} = P_{gd} + \delta_{2,d}$$
$$\delta_{2,d} = \frac{(P_{1,d} - P_{2,d}) + (P_{3,d} - P_{4,d})}{2} \qquad (16)$$

Where δ_2 is the weighted error in different dimensions ,T_{id} is the offspring ,P_{gd} is the best position of the parent from eq. (16). The random number between [0, 1] is less than the reproduction rate or the position of the particle in any one randomly chosen dimension, k is mutated then the mutation takes place. The parent might not be the same as offspring. Fitness function is also evaluated. The parent is replaced by the only offspring and thus has better fitness. If no replacement takes place, then it will be retained for next iteration.

4.4 DE-ACO-BASED FIR FILTER DESIGN

Thirdly, DE-ACO based FIR filter design is formulated.

The transfer function of the FIR filter is as follows eq. (17):

$$\frac{y_n}{x_n} = a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n} \qquad (17)$$

The particles are dispersed in a D dimensional search space, where D = N for FIR filter. The new coefficient is used to calculate the particles fitness and iteration which improves the search. The error obtained is below certain limit or the error is below after some iteration to obtain the result, and it will be considered as the final result.

An error function is an approximate error used in the Parks McClellan algorithm for filter design:

$$E(w) = G(w)[H_d(e^{jw}) - (H(e^{jw})] \qquad (18)$$

Where, G (w) is used to provide different weights for the relative errors in different frequency bands as the weighting function from eq. (18), $H_d(e^{jw})$ Is the frequency response of the desired filter and $H(e^{jw})$ is the frequency response of the approximate filter.

$$F_1 = \max_{w \le w_p}(|E(w)| - \delta_p) + \max_{w \le w_s}(|E(w)| - \delta_s) \qquad (19)$$

Where, δ_p and δ_s are the ripples in pass and stop bands and w_p and w_s are pass and stop band cut-off frequencies respectively from eq. (19). The error is minimized by algorithms, and thus it increases the fitness.

4.5 ADVANTAGES OF DE-ACO ALGORITHM

DE algorithm is a new heuristic approach mainly having three advantages.

Finding the true global minimum of a multimodal search space regardless of the initial parameter values. Fast convergence. It uses only few control parameters [9]. Advantages of the ACO is inherent parallelism, positive feedback accounts for rapid discovery of good solutions and used in dynamic applications.

4.6 FIR FILTER DESIGN

The FIR filters are designed to optimize the coefficients that give the best frequency response. The filter is implemented by Xilinx [2] integrated synthesis environment tool is shown in (figure 8). FIR filter is introduced to reduce the ripple constant and also to reduce the power consumption. The filter is sensitive to the positive edge of the clock and negative edge of reset. s_axis_tdata represents input data to the filter and output data is in m_axis_tdata. The obtained output with a delay is 20 ns.

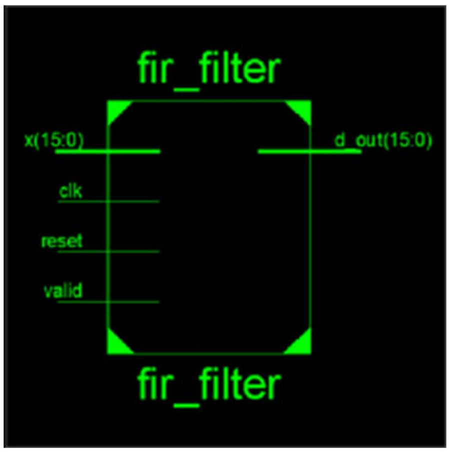
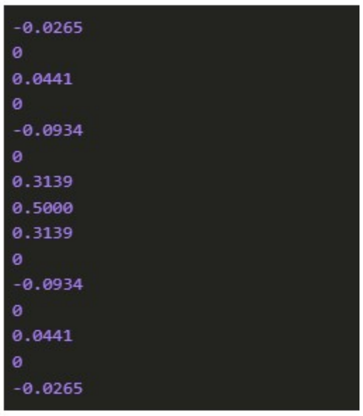


Figure 8.Register Transfer Level Logic Diagram

The FIR designed is composed of three main logic components: a circular buffer to clock each sample into that properly accounts for the delays of the serial input, multipliers for each of the taps' coefficient value, and the accumulator register for the summing result from each tap's output. Both DE and ACO are implemented in MATLAB, and the coefficients are quantized to minimum word length. The realizations of the filters are done using the transposed form structure. In the constant coefficient FIR Filters, adders /subtractors replaced the multipliers which are called multiplier adders (MA). Adders are realized using ripple carry adders.

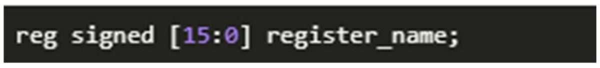
Implemented a simple 15-tap low pass filter from (figure 9) FIR sampling at 1Ms/s with a passband frequency of 200 kHz and a stop band frequency of 355kHz which gave me the following coefficients:



```
-0.0265
0
0.0441
0
-0.0934
0
0.3139
0.5000
0.3139
0
-0.0934
0
0.0441
0
-0.0265
```

Figure 9. 15 Tap Low Pass Filter Coefficients

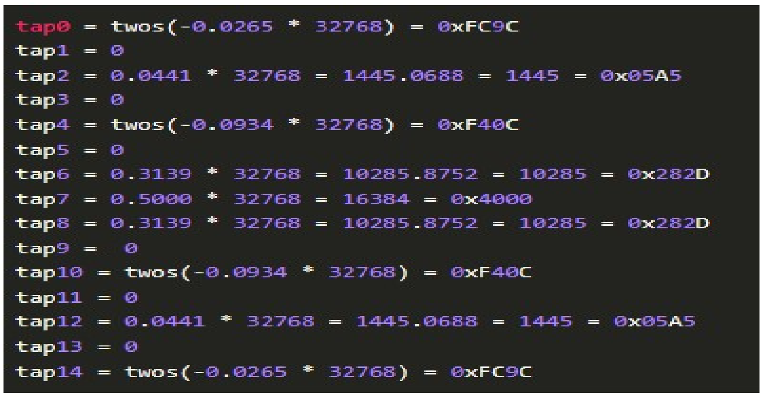
Now after deciding on the order (number of taps) for your FIR and obtaining your coefficient values, the next set of parameters that must be defined is the bit width of the input samples, output samples, and the coefficients themselves. For this FIR, input sample and coefficient registers to be 16 bits wide and my output sample register to be 32 bits since the product of two 16-bit values is a 32-bit value (the widths of the two values being multiplied add to give the width of the product, so if chosen 16-bit input samples with 8-bit taps then the output samples would be 24 bits wide).These values are also all signed, thus the MSB is used as the sign bit and the lower remaining bits are what the value must fit into. To set these values as signed data type in Verilog, the keyword *signed* is used as in (figure 10):



```
reg signed [15:0] register_name;
```

Figure 10. Signed Keyword in Verilog

The next thing to address is to handle the coefficient values in Verilog, the decimal point values need to be converted to fixed point values. Since all of the coefficient values are less than one, all 15 bits (the MSB of the total 16 bits is the signed bit) of our registers can be dedicated to fractional bits. The no. of bits in the register you want to dedicate to the integer part of the number vs the fractional part of the number is decided. Therefore the math to convert the fractional value taps is: (fractional coefficient value)*(2^(15)) from (figure 11) ,where any decimal value of this product is rounded off and the two's compliment of the value is calculated if the coefficient is negative.

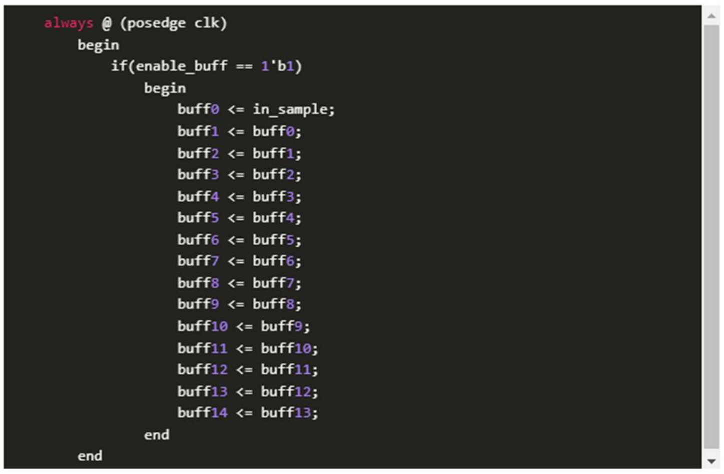


```
tap0 = twos(-0.0265 * 32768) = 0xFC9C
tap1 = 0
tap2 = 0.0441 * 32768 = 1445.0688 = 1445 = 0x05A5
tap3 = 0
tap4 = twos(-0.0934 * 32768) = 0xF40C
tap5 = 0
tap6 = 0.3139 * 32768 = 10285.8752 = 10285 = 0x282D
tap7 = 0.5000 * 32768 = 16384 = 0x4000
tap8 = 0.3139 * 32768 = 10285.8752 = 10285 = 0x282D
tap9 =  0
tap10 = twos(-0.0934 * 32768) = 0xF40C
tap11 = 0
tap12 = 0.0441 * 32768 = 1445.0688 = 1445 = 0x05A5
tap13 = 0
tap14 = twos(-0.0265 * 32768) = 0xFC9C
```

Figure 11.Conversion of Fractional Value TAPS to Hex Form

4.7 LOGIC OF THE FIR MODULE

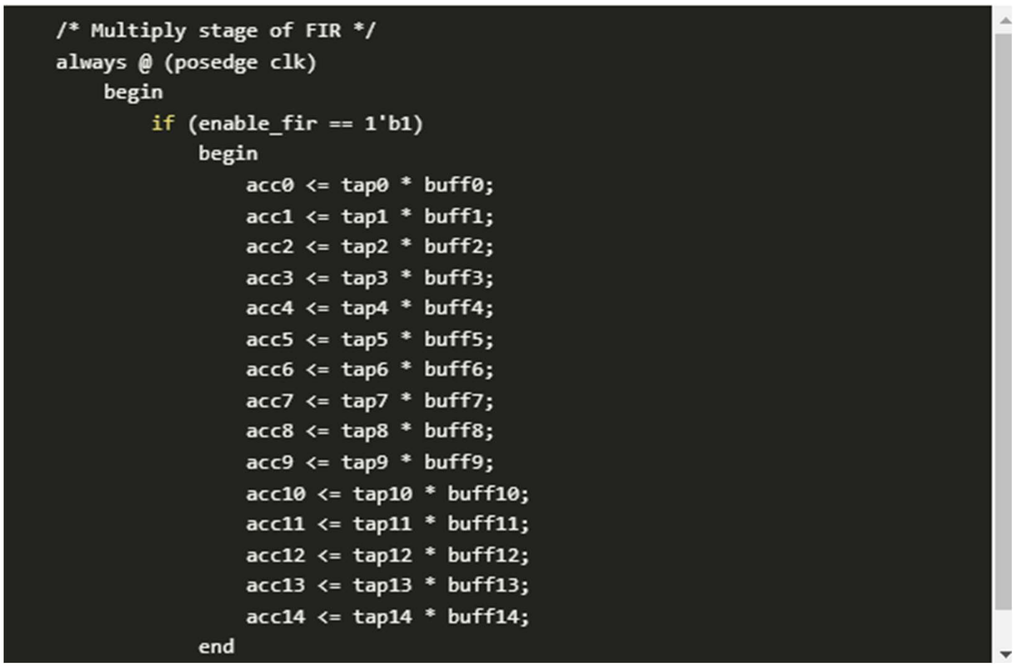
The first of which is the circular buffer which brings in a serial input sample stream from (figure 12) and creates an array of 15 input samples for the 15 taps of the filter.



```
always @ (posedge clk)
    begin
        if(enable_buff == 1'b1)
            begin
                buff0 <= in_sample;
                buff1 <= buff0;
                buff2 <= buff1;
                buff3 <= buff2;
                buff4 <= buff3;
                buff5 <= buff4;
                buff6 <= buff5;
                buff7 <= buff6;
                buff8 <= buff7;
                buff9 <= buff8;
                buff10 <= buff9;
                buff11 <= buff10;
                buff12 <= buff11;
                buff13 <= buff12;
                buff14 <= buff13;
            end
    end
```

Figure 12. 15 Input Samples for 15 Tap FIR Filter

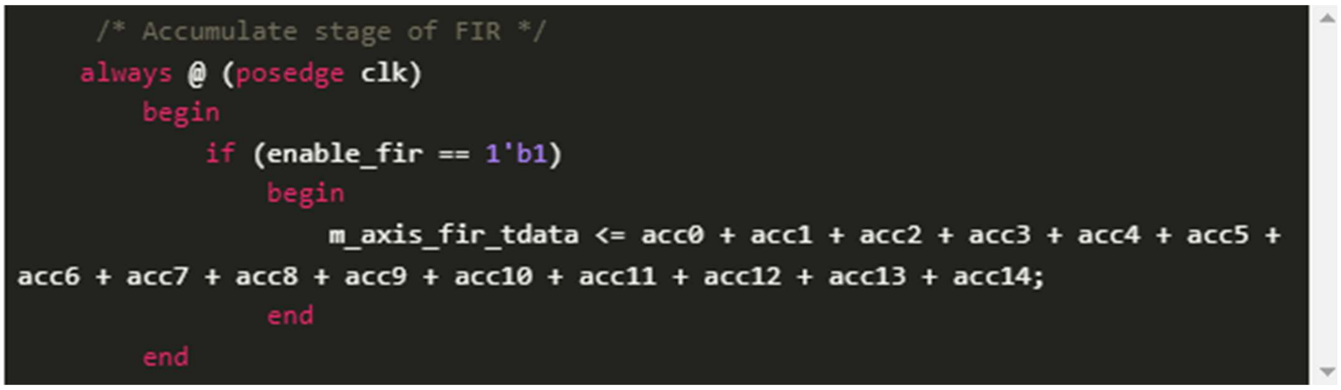
Next, the multiply stage multiplies each sample by each of the coefficient values as in below (figure 13):



```
/* Multiply stage of FIR */
always @ (posedge clk)
    begin
        if (enable_fir == 1'b1)
            begin
                acc0 <= tap0 * buff0;
                acc1 <= tap1 * buff1;
                acc2 <= tap2 * buff2;
                acc3 <= tap3 * buff3;
                acc4 <= tap4 * buff4;
                acc5 <= tap5 * buff5;
                acc6 <= tap6 * buff6;
                acc7 <= tap7 * buff7;
                acc8 <= tap8 * buff8;
                acc9 <= tap9 * buff9;
                acc10 <= tap10 * buff10;
                acc11 <= tap11 * buff11;
                acc12 <= tap12 * buff12;
                acc13 <= tap13 * buff13;
                acc14 <= tap14 * buff14;
            end
```

Figure 13.Multiplying Each Input Sample with Coefficient Values

The resulting values from the multiply stage are accumulated from (figure 14) by addition in a register which ultimately is the output data stream from the filter.



```
    /* Accumulate stage of FIR */
    always @ (posedge clk)
        begin
            if (enable_fir == 1'b1)
                begin
                    m_axis_fir_tdata <= acc0 + acc1 + acc2 + acc3 + acc4 + acc5 +
acc6 + acc7 + acc8 + acc9 + acc10 + acc11 + acc12 + acc13 + acc14;
                end
        end
```

Figure 14.Output Data Stream from FIR filter

Finally, the last part of the logic is the interface to stream data to and from the FIR module. The AXI Stream interface is one of the most common, thus what I chose to implement. The key aspects are the valid and ready signals which allow for the control of the flow of data between upstream and downstream devices from (figure15). This means the FIR module needs to provide a valid signal to its downstream device to indicate that it's output is valid data, as well as be able to pause (but still retain) its output if the downstream device de-asserts its ready signal. The FIR module must also be able to behave this same way with its upstream device on its master side interface.

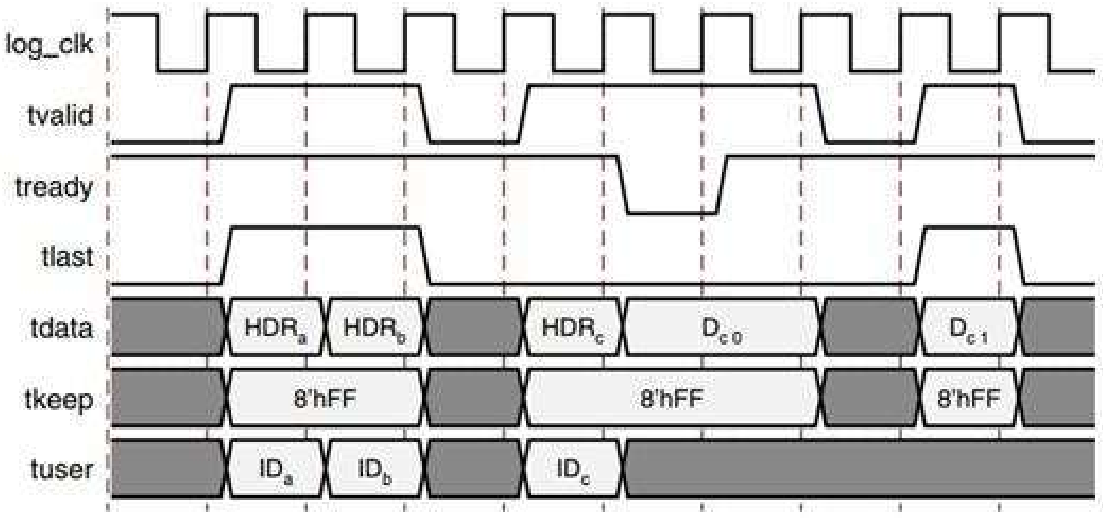


Figure 15.AXI Stream Timing Diagram

There are two main things that need to be tested in the FIR module: the filter math and the AXI stream interface. To accomplish this, a state machine in the test bench that generates a simple 200kHz sine wave is created and also toggles the valid signal on the slave side and the ready signal on the master side of the FIR's interface. Under the simulation sources in the Sources window, set the testbench module as the top-level file by right-clicking on it and selecting *Set as Top* from the reference of the (figure 16).

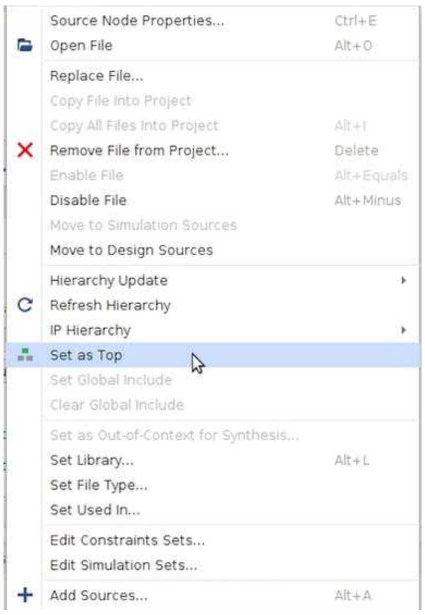


Figure 16.Setting Test Bench as Top

$$\sum_{n=0}^{N} h(n)z^{-n}, n = 0,1 \ldots . N \qquad (20)$$

where N is the order of the filter which has (N) number of coefficients. h (n) is the filter's impulse response. It is calculated by applying an impulse signal at the input. The values of h(n) from eq. (20) will find the type of the filter.

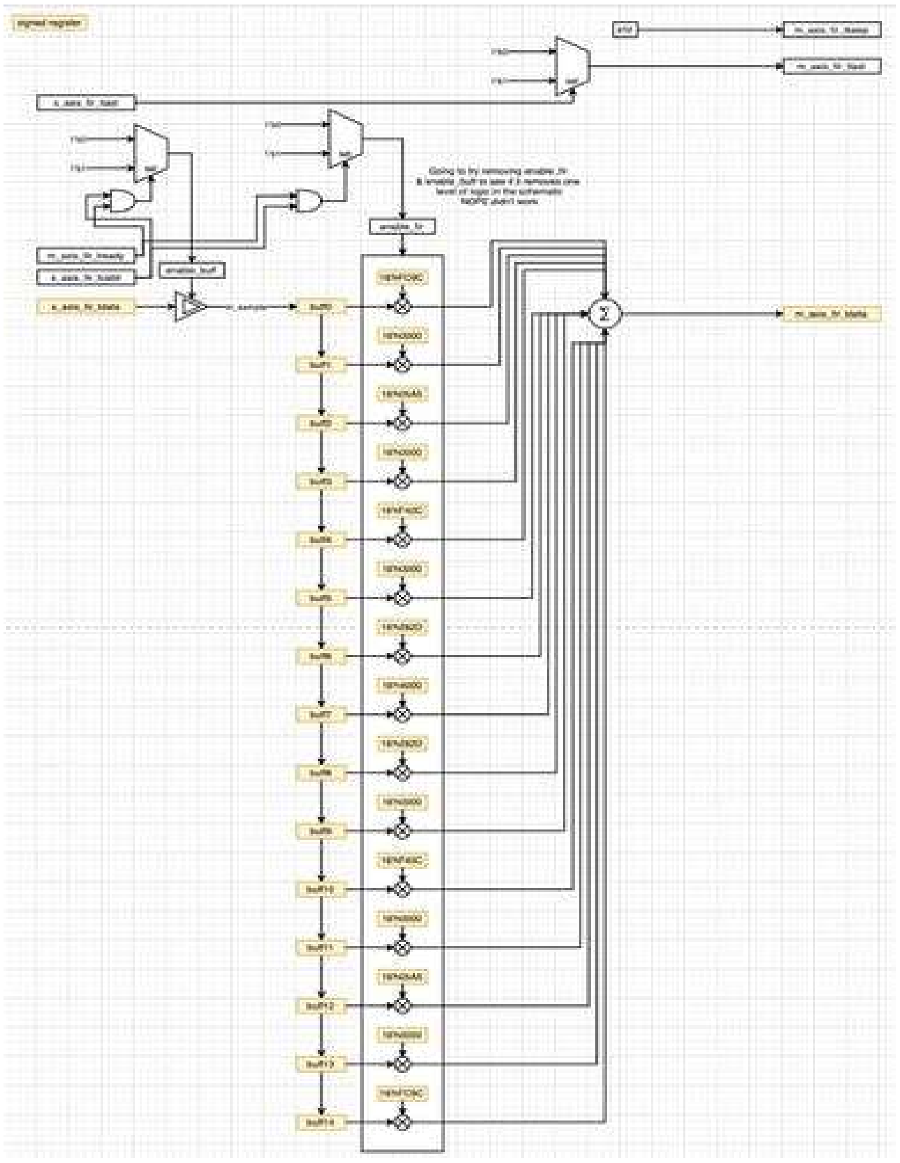


Figure 17.Logic Design of 15 tap FIR Module

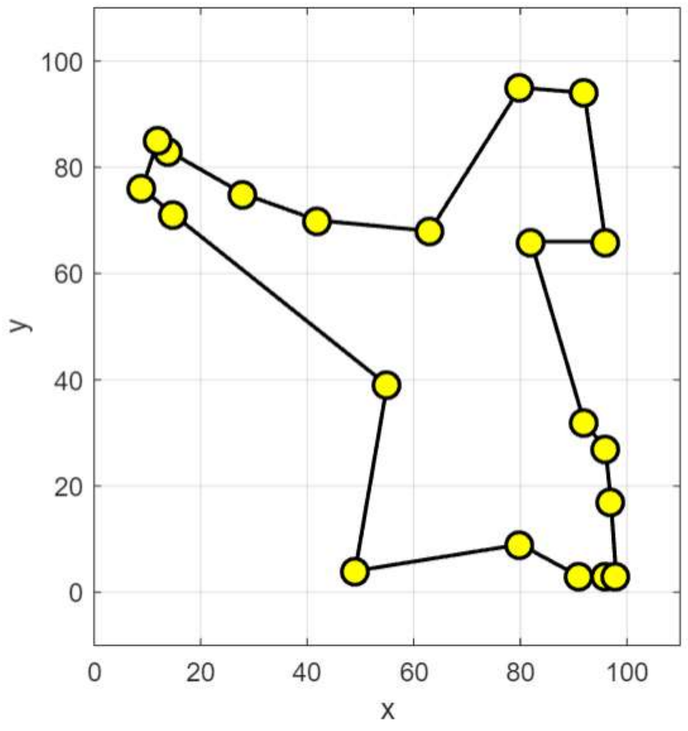
The DE-ACO algorithm, the individual represents h(n). In each iteration, these individuals are updated. Fitness of particles is calculated using the new coefficients. In each iteration, this fitness is used to improve the search and results obtained after a certain number of iterations or after the error is below a certain limit is considered to be the optimal result. Fitness of the individuals is calculated using the filter coefficients. Because the FIR filter uses a linear symmetric structure from (figure 17), the impulse response coefficients have symmetrical features. Then we symmetrically add the pre-input x(n) for simplifying the design.

By using DE-ACO the desired magnitude response and filter coefficients are obtained. Filter designed by the DE-ACO method produce better response in terms of minimum stop band ripple magnitude and maximum stop band attenuation. Both DE and ACO are implemented in MATLAB, and the coefficients are quantized to minimum word length. The realizations of the filters are done using the transposed form structure. In the constant coefficient FIR Filters, adders/ subtractors replaced the multipliers which are called as multiplier adders (MA). Adders are realized using ripple carry adders .The fitness based adaptive DE with ACO known as fitness based adaptive DEACO is used for the design of 15th order FIR filters. It is revealed that DE-ACO has the ability to converge to the best quality near-optimal solution and possesses the best convergence characteristics among the algorithms. DEACO is more efficient in optimizing the filter coefficients successfully. Maximum power consumption occurs during multiplication operations. Hence, to reduce power consumption, the number of multiplications has to be minimized. The number of SPT terms in the filter coefficients has to be optimally minimized, without compromising on the filter response.

CHAPTER 5

RESULTS

5.1 SIMULATED RESULTS FROM MATLAB FOR ACO



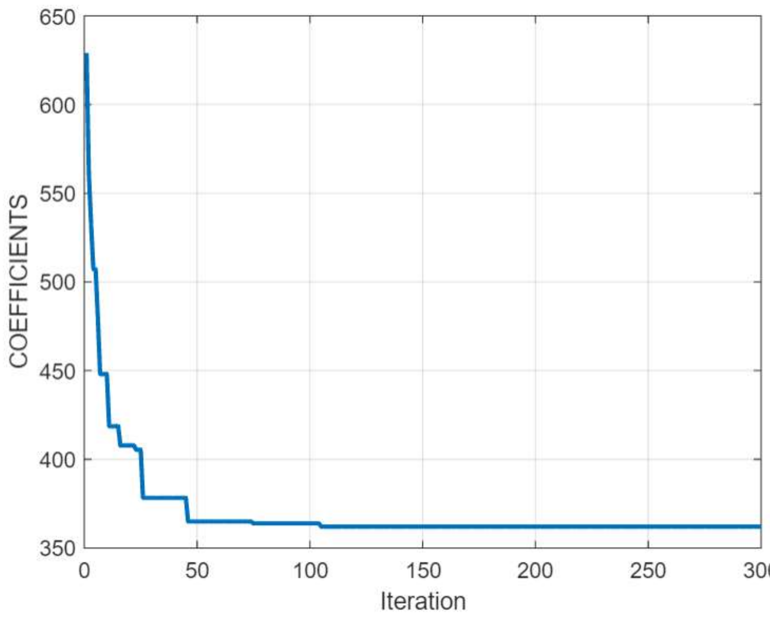
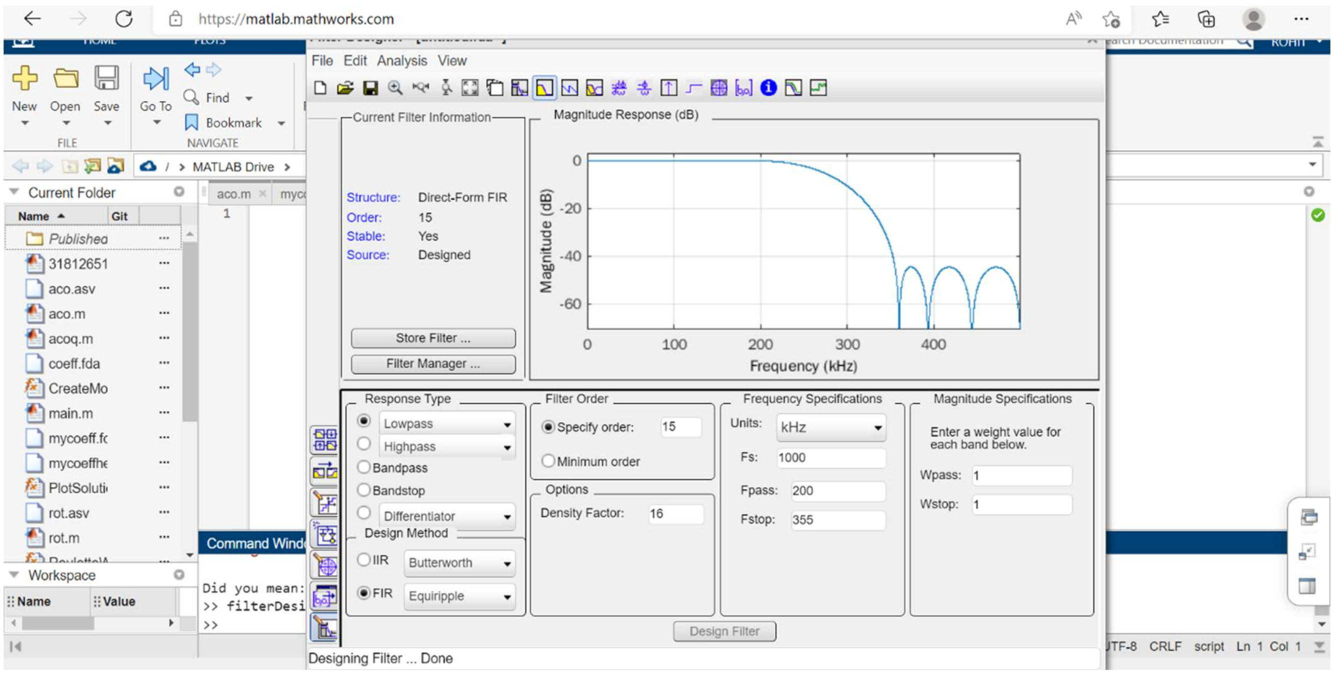


Figure 18.Optimized Outputs from ACO Algorithm

SIMULATED RESULTS FROM MATLAB FOR FIR FILTER


Current Filter Information

Structure: Direct-Form FIR
Order: 15
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR] Equiripple

Filter Order: [✓ Specify order: 15] Minimum order

Options: Density Factor: 16

Frequency Specifications: Units: kHz; Fs: 1000; Fpass: 200; Fstop: 355

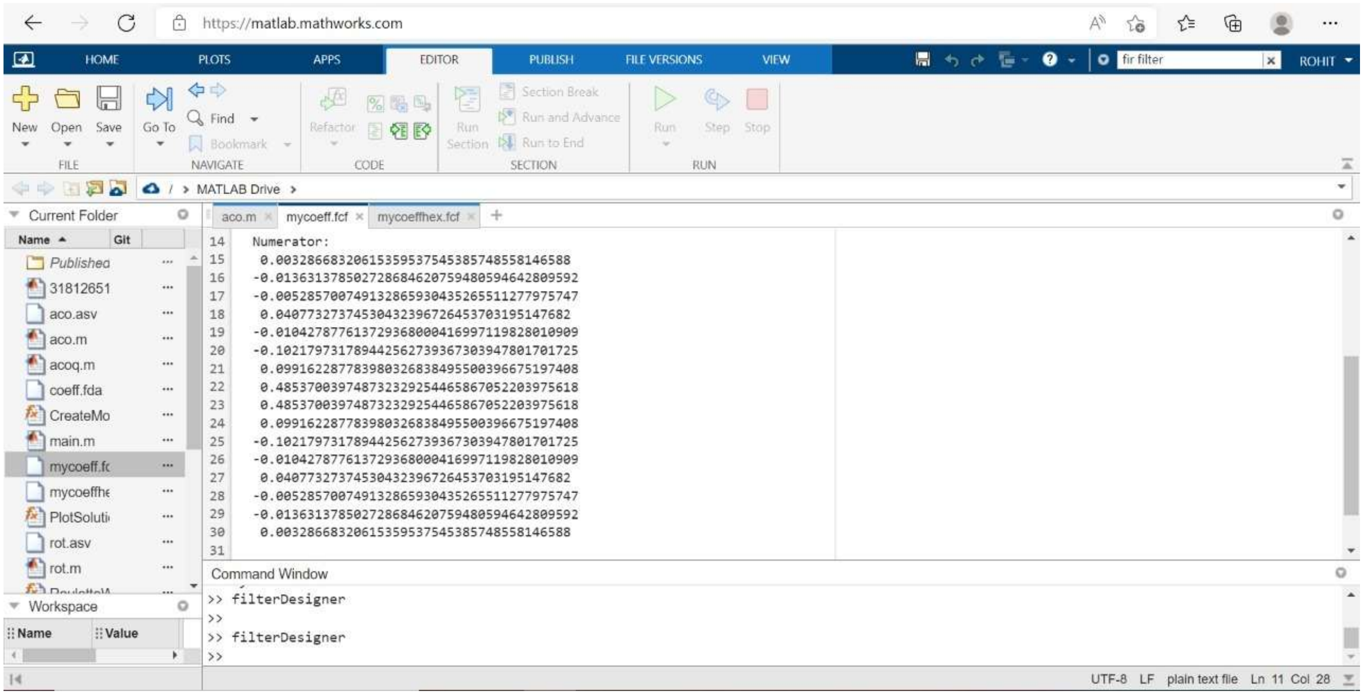
Magnitude Specifications: Enter a weight value for each band below. Wpass: 1; Wstop: 1

```
Did you mean:
>> filterDesi
>>
```

Designing Filter ... Done

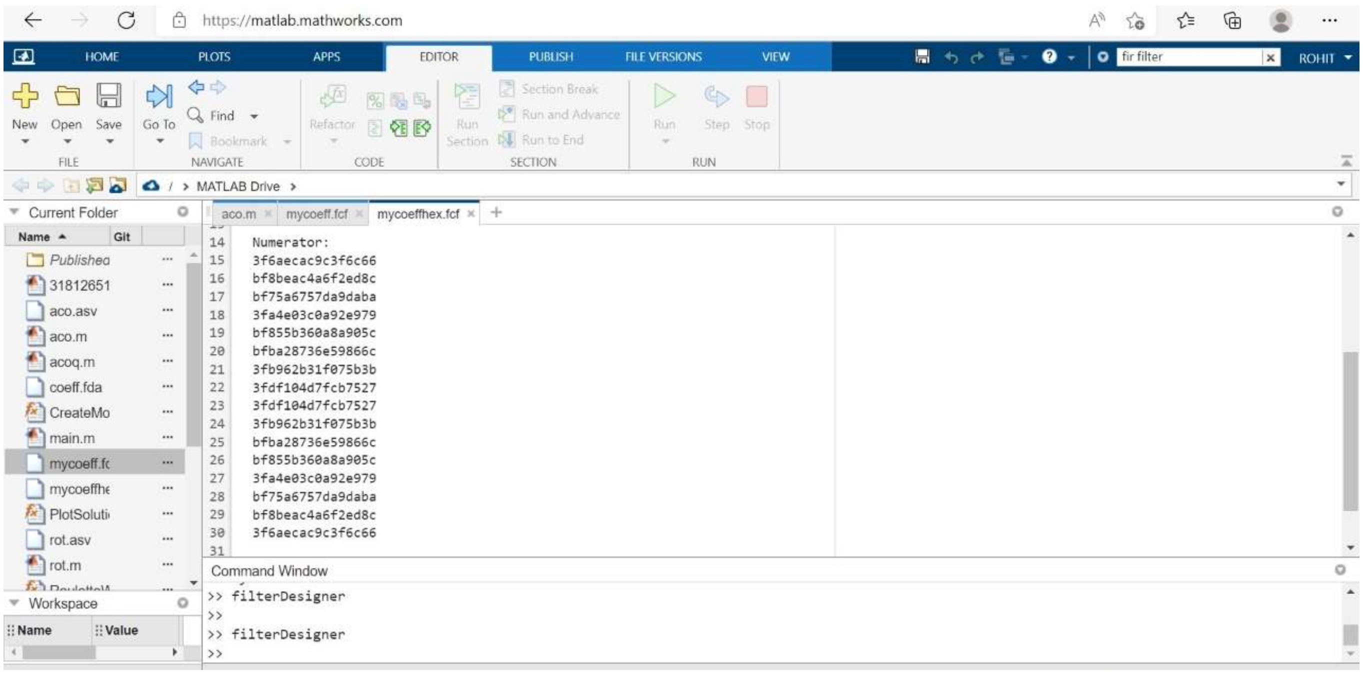
Figure 19.FIR Filter Designing Using Parameters

COEFFICIENTS OBTAINED FROM FILTER DESIGNER


```
Numerator:
  0.00328668320615359537545385748558146588
 -0.013631378502728684620759480594642809592
 -0.005285700749132865930435265511277975747
  0.04077327374530432396726453703195147682
 -0.010427877613729368000416997119828010909
 -0.102179731789442562739367303947801701725
  0.099162287783980326838495500396675197408
  0.485370039748732329254465867052203975618
  0.485370039748732329254465867052203975618
  0.099162287783980326838495500396675197408
 -0.102179731789442562739367303947801701725
 -0.010427877613729368000416997119828010909
  0.04077327374530432396726453703195147682
 -0.005285700749132865930435265511277975747
 -0.013631378502728684620759480594642809592
  0.00328668320615359537545385748558146588
```

```
>> filterDesigner
>>
>> filterDesigner
>>
```

Figure 20.Coefficients Obtained from Filter Design

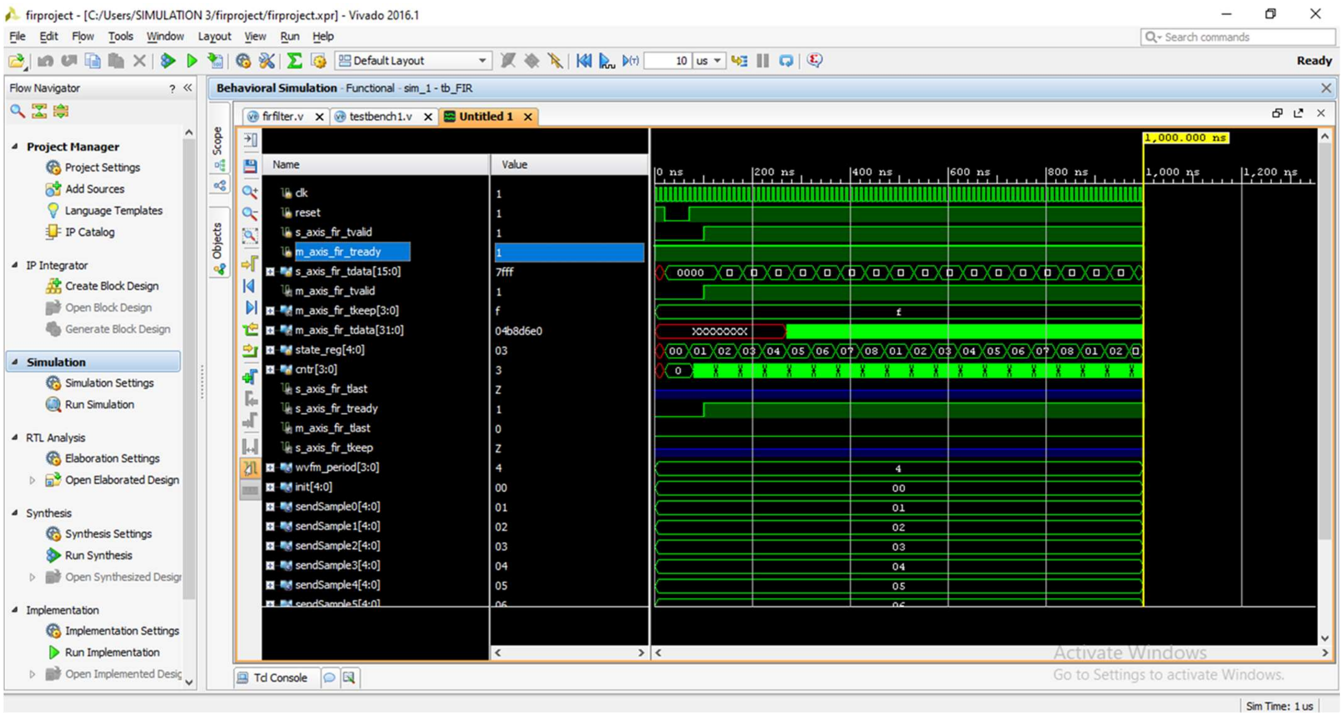


```
Numerator:
3f6aecac9c3f6c66
bf8beac4a6f2ed8c
bf75a6757da9daba
3fa4e03c0a92e979
bf855b360a8a905c
bfba28736e59866c
3fb962b31f075b3b
3fdf104d7fcb7527
3fdf104d7fcb7527
3fb962b31f075b3b
bfba28736e59866c
bf855b360a8a905c
3fa4e03c0a92e979
bf75a6757da9daba
bf8beac4a6f2ed8c
3f6aecac9c3f6c66
```

```
>> filterDesigner
>>
>> filterDesigner
>>
```

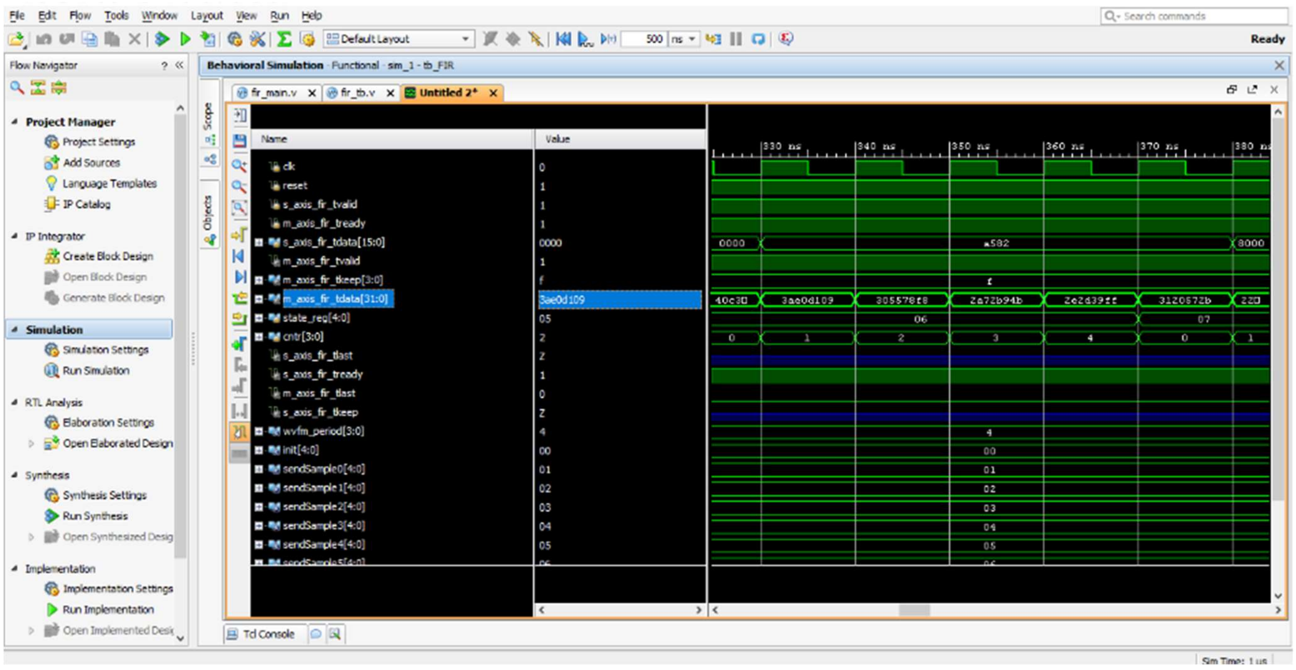
Figure 21. Coefficients Obtained from Filter Design in Hexadecimal

5.2 FIR FILTER OUTPUTS FROM VIVADO



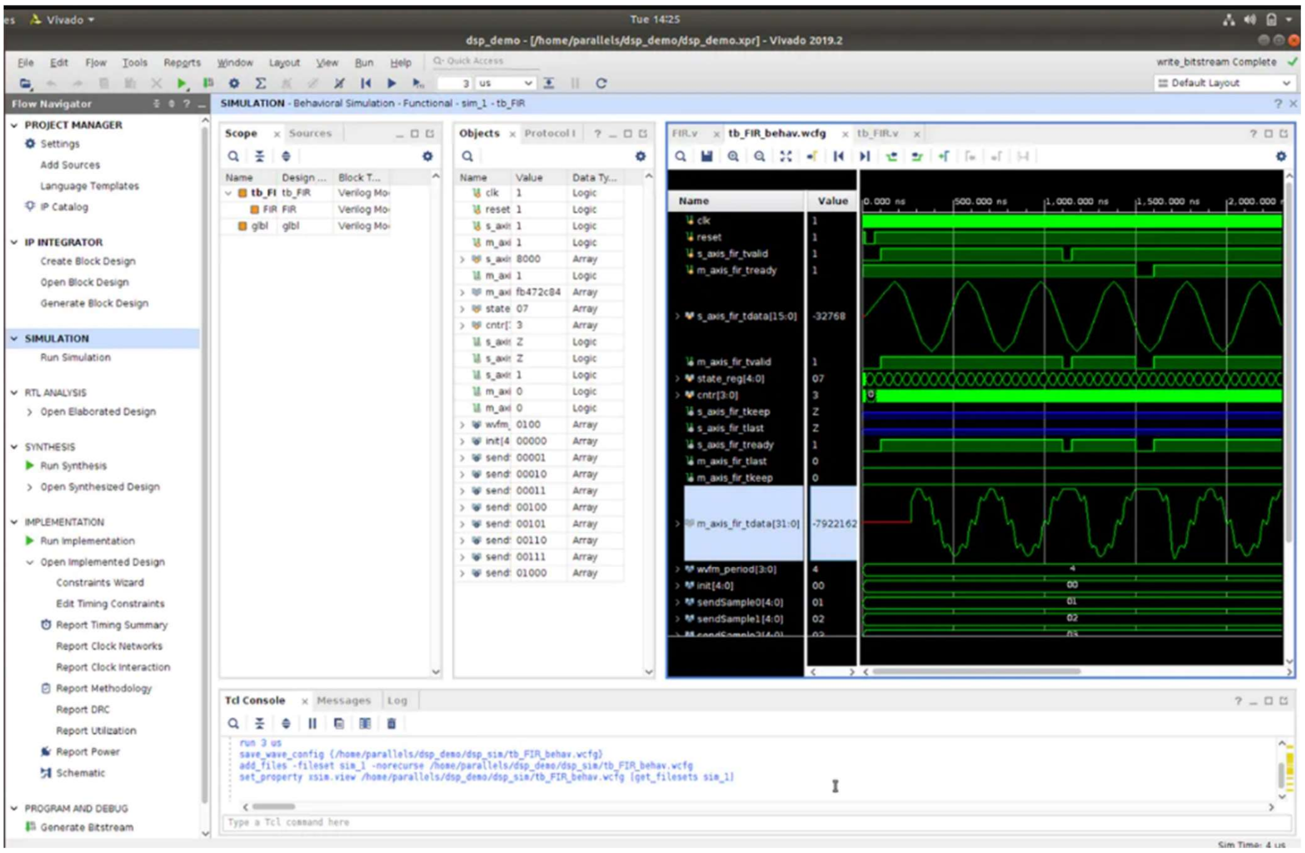
```
Name                    Value
clk                     1
reset                   1
s_axis_fir_tvalid       1
m_axis_fir_tready       1
s_axis_fir_tdata[15:0]  7fff
m_axis_fir_tvalid       1
m_axis_fir_tkeep[3:0]   f
m_axis_fir_tdata[31:0]  04b8d6e0
state_reg[4:0]          03
cntr[3:0]               3
s_axis_fir_tlast        Z
s_axis_fir_tready       1
m_axis_fir_tlast        0
s_axis_fir_tkeep        Z
wvfm_period[3:0]        4
init[4:0]               00
sendSample0[4:0]        01
sendSample1[4:0]        02
sendSample2[4:0]        03
sendSample3[4:0]        04
sendSample4[4:0]        05
```

Figure 22.1. FIR Filter Outputs from Vivado



```
Name                     Value
clk                      0
reset                    1
s_axis_fir_tvalid        1
m_axis_fir_tready        1
s_axis_fir_tdata[15:0]   0000
m_axis_fir_tvalid        1
m_axis_fir_tkeep[3:0]    f
m_axis_fir_tdata[31:0]   3ae0d109
state_reg[4:0]           05
cntr[3:0]                2
s_axis_fir_tlast         z
s_axis_fir_tready        1
m_axis_fir_tlast         0
s_axis_fir_tkeep         z
wvfm_period[3:0]         4
init[4:0]                00
sendSample0[4:0]         01
sendSample1[4:0]         02
sendSample2[4:0]         03
sendSample3[4:0]         04
sendSample4[4:0]         05
```

Figure 22.2. FIR Filter Outputs from Vivado



```
run 3 us
save_wave_config {/home/parallels/dsp_demo/dsp_sim/tb_FIR_behav.wcfg}
add_files -fileset sim_1 -norecurse /home/parallels/dsp_demo/dsp_sim/tb_FIR_behav.wcfg
set_property xsim.view /home/parallels/dsp_demo/dsp_sim/tb_FIR_behav.wcfg [get_filesets sim_1]
```

Figure 22.3. FIR Filter Outputs from Vivado

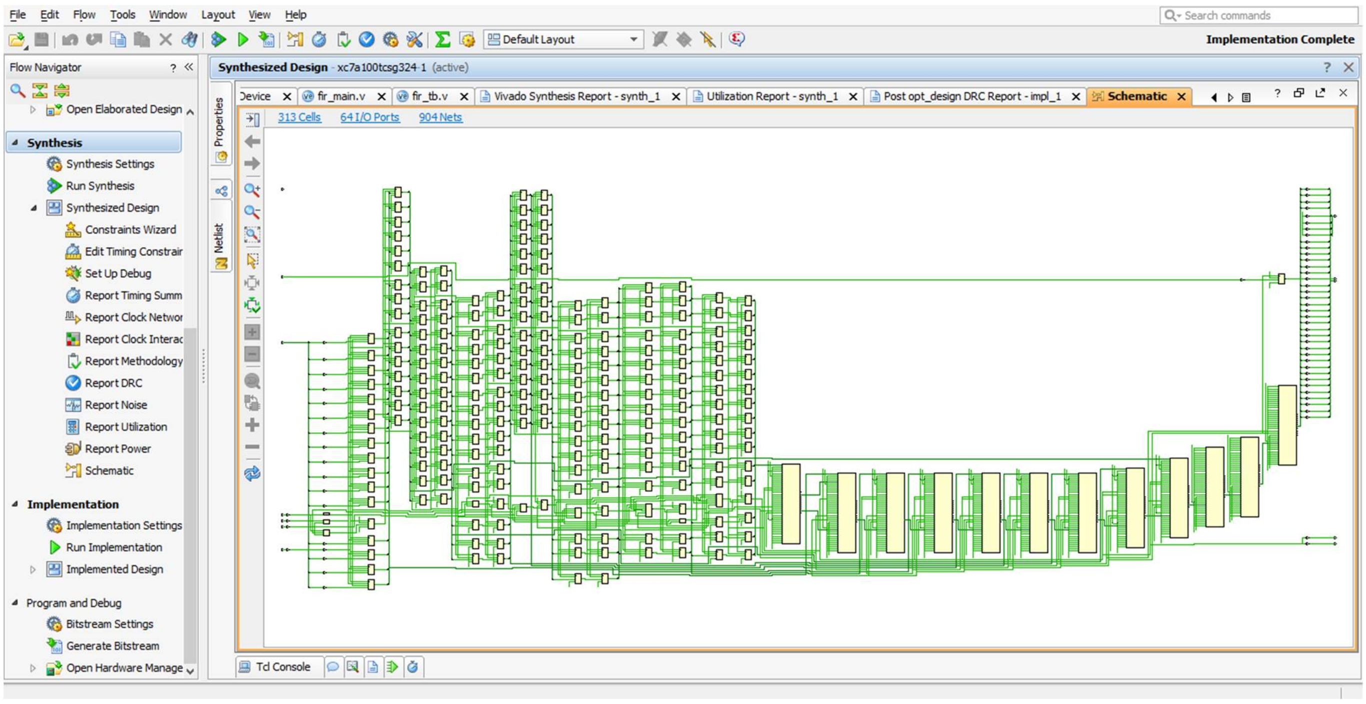


Figure 23.1. Schematic Diagram of FIR Filter Implementation

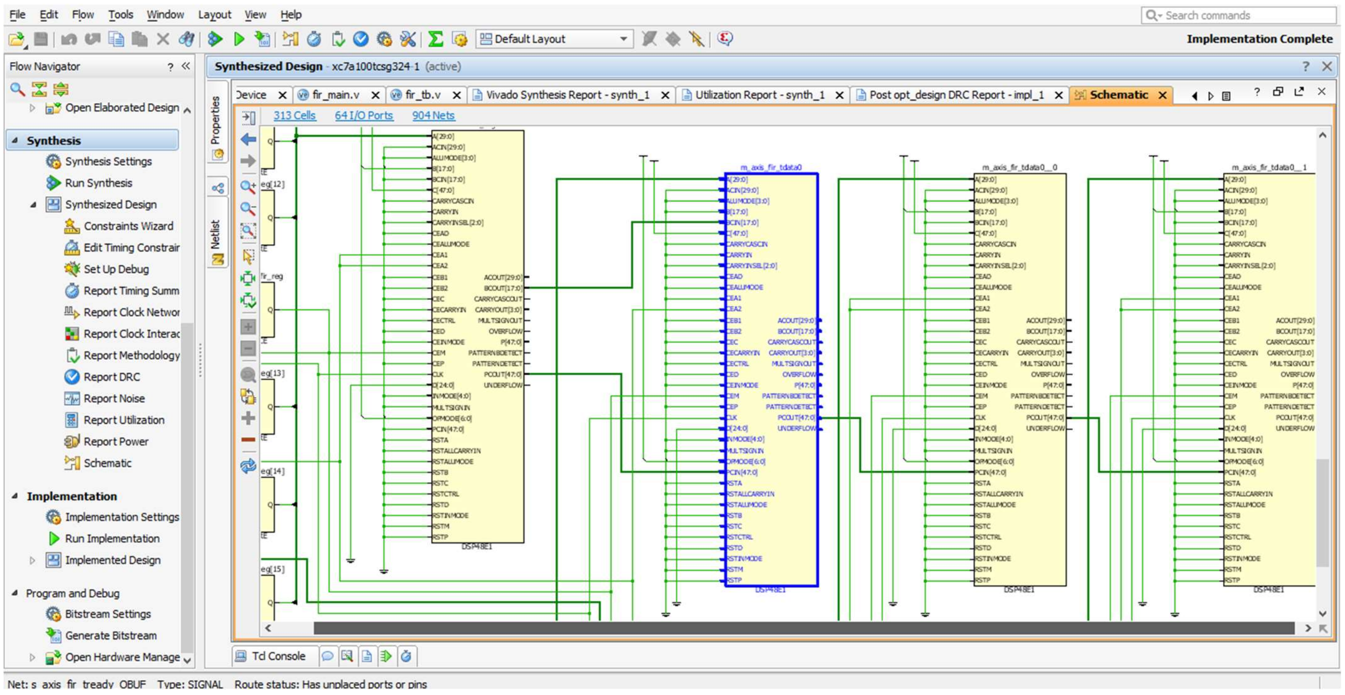

File Edit Flow Tools Window Layout View Help
Default Layout
Implementation Complete
Flow Navigator
Synthesized Design - xc7a100tcsg324-1 (active)
Schematic
313 Cells
64 I/O Ports
904 Nets
m_axis_fir_tdata0
m_axis_fir_tdata0__0
m_axis_fir_tdata0__1
DSP48E1
Tcl Console
Net: s_axis_fir_tready_OBUF Type: SIGNAL Route status: Has unplaced ports or pins

Figure 23.2. Schematic Diagram of FIR Filter Implementation

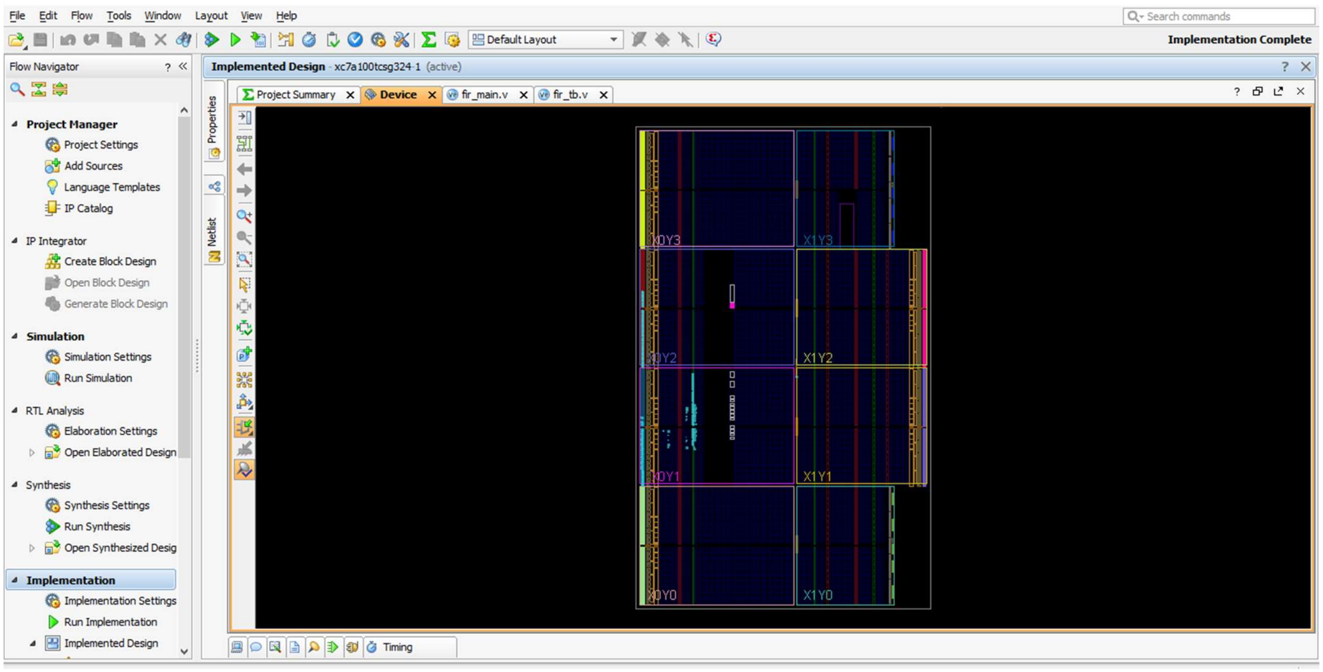


Figure 24.1. Implementation Design

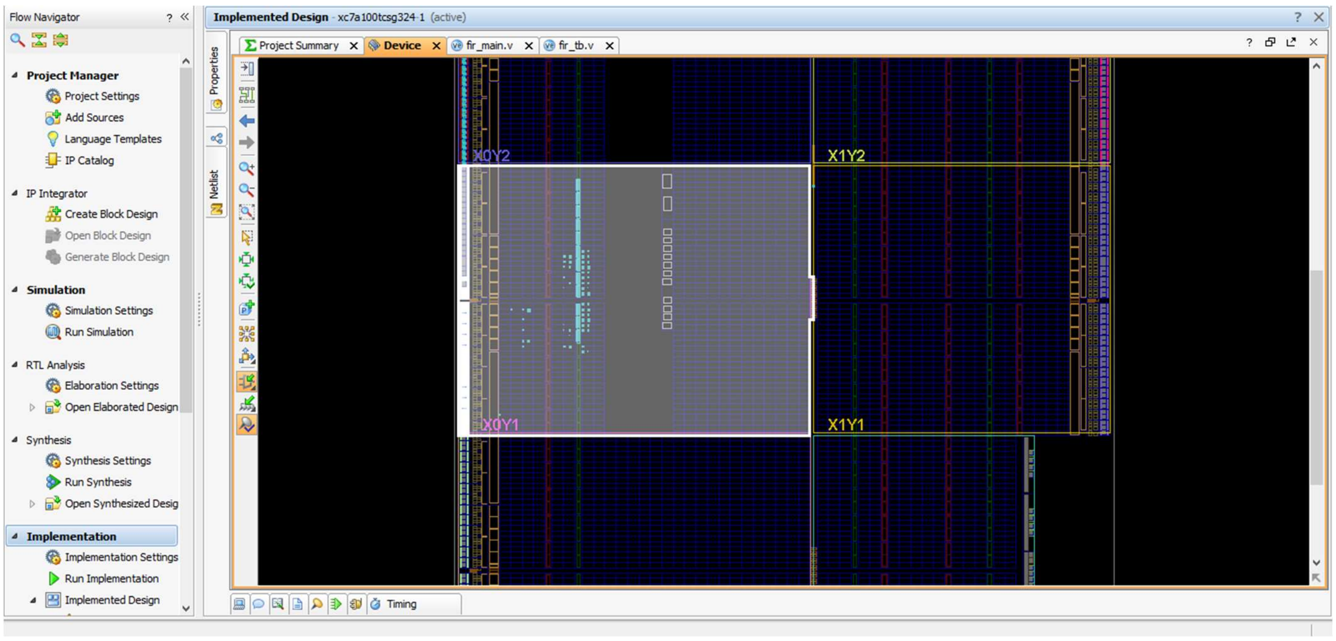
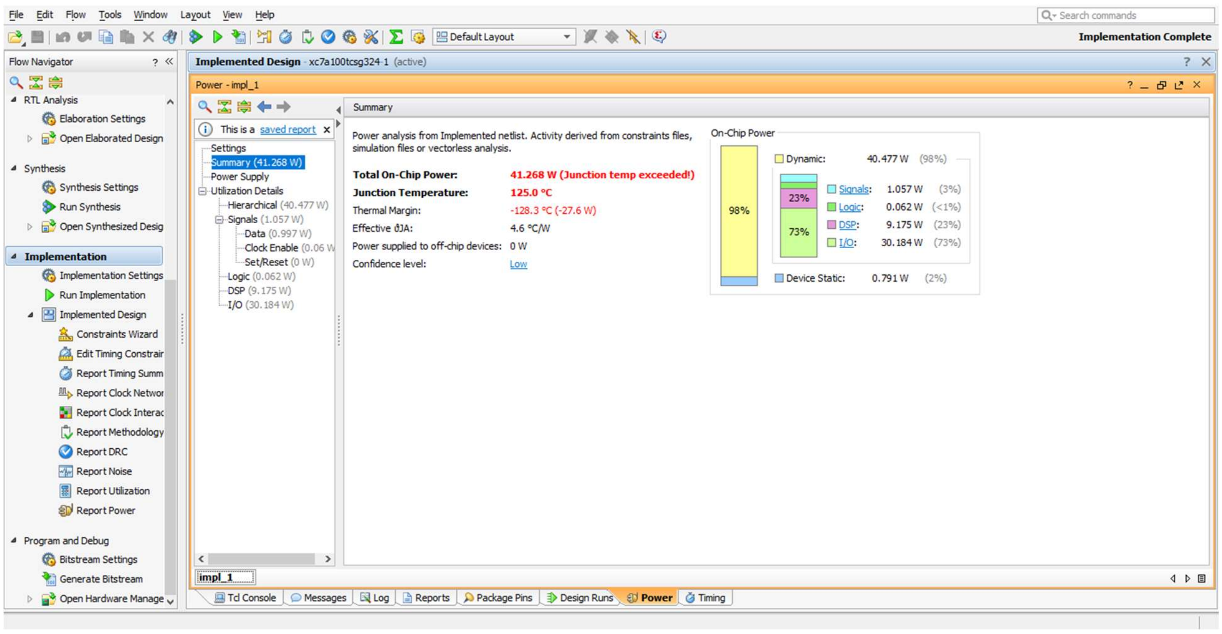


Figure 24.2. Implementation Design



Implemented Design - xc7a100tcsg324-1 (active)

Power - impl_1

This is a saved report

```
Settings
Summary (41.268 W)
Power Supply
Utilization Details
    Hierarchical (40.477 W)
    Signals (1.057 W)
        Data (0.997 W)
        Clock Enable (0.06 W
        Set/Reset (0 W)
    Logic (0.062 W)
    DSP (9.175 W)
    I/O (30.184 W)
```

Summary

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

Total On-Chip Power:	41.268 W (Junction temp exceeded!)
Junction Temperature:	125.0 °C
Thermal Margin:	-128.3 °C (-27.6 W)
Effective ϑJA:	4.6 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

On-Chip Power

Dynamic:	40.477 W	(98%)
Signals:	1.057 W	(3%)
Logic:	0.062 W	(<1%)
DSP:	9.175 W	(23%)
I/O:	30.184 W	(73%)
Device Static:	0.791 W	(2%)

Figure 25. Power Utilization Details

VIVADO SYNTHESIS REPORT

Start RTL Component Statistics

Detailed RTL Component Info:

```
+---Adders :
	   2 Input      4 Bit       Adders := 1
+---Registers :
	                32 Bit    Registers := 1
	                16 Bit    Registers := 10
	                 4 Bit    Registers := 2
	                 1 Bit    Registers := 3
+---Muxes :
	   2 Input      4 Bit        Muxes := 2
	   3 Input      1 Bit        Muxes := 1
```

Finished RTL Component Statistics

Start RTL Hierarchical Component Statistics

Hierarchical RTL Component report
Module FIR
Detailed RTL Component Info :

```
+---Adders :
2 Input      4 Bit       Adders := 1
+---Registers :
32 Bit    Registers := 1
16 Bit    Registers := 10
4 Bit    Registers := 2
1 Bit    Registers := 3
+---Muxes :
2 Input      4 Bit        Muxes := 2
3 Input      1 Bit        Muxes := 1
---------------------------------------------------------------------------------
Finished RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
```

Figure 26. Vivado Synthesis Report

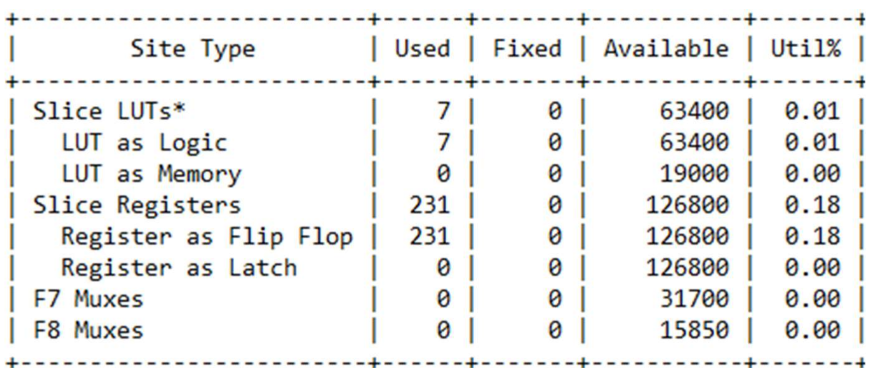
<u>UTILIZATION REPORT FOR THE DESIGN</u>

```
-------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------
```



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |    7 |     0 |     63400 |  0.01 |
|   LUT as Logic          |    7 |     0 |     63400 |  0.01 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |  231 |     0 |    126800 |  0.18 |
|   Register as Flip Flop |  231 |     0 |    126800 |  0.18 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 21    |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 210   |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+


2. Memory
---------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       135 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       135 |  0.00 |
|   RAMB18       |    0 |     0 |       270 |  0.00 |
+----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can
accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a
Block RAM Tile, that tile can still accommodate a RAMB18E1


3. DSP
------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| DSPs           |   12 |     0 |       240 |  5.00 |
|   DSP48E1 only |   12 |       |           |       |
+----------------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------
```

```
+-----------------------------+------+-------+-----------+-------+
|          Site Type          | Used | Fixed | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Bonded IOB                  |   60 |     0 |       210 | 28.57 |
| Bonded IPADs                |    0 |     0 |         2 |  0.00 |
| PHY_CONTROL                 |    0 |     0 |         6 |  0.00 |
| PHASER_REF                  |    0 |     0 |         6 |  0.00 |
| OUT_FIFO                    |    0 |     0 |        24 |  0.00 |
| IN_FIFO                     |    0 |     0 |        24 |  0.00 |
| IDELAYCTRL                  |    0 |     0 |         6 |  0.00 |
| IBUFDS                      |    0 |     0 |       202 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |     0 |        24 |  0.00 |
| PHASER_IN/PHASER_IN_PHY     |    0 |     0 |        24 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |     0 |       300 |  0.00 |
| IBUFDS_GTE2                 |    0 |     0 |         4 |  0.00 |
| ILOGIC                      |    0 |     0 |       210 |  0.00 |
| OLOGIC                      |    0 |     0 |       210 |  0.00 |
+-----------------------------+------+-------+-----------+-------+


5. Clocking
-----------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    1 |     0 |        32 |  3.13 |
| BUFIO      |    0 |     0 |        24 |  0.00 |
| MMCME2_ADV |    0 |     0 |         6 |  0.00 |
| PLLE2_ADV  |    0 |     0 |         6 |  0.00 |
| BUFMRCE    |    0 |     0 |        12 |  0.00 |
| BUFHCE     |    0 |     0 |        96 |  0.00 |
| BUFR       |    0 |     0 |        24 |  0.00 |
+------------+------+-------+-----------+-------+


6. Specific Feature
-------------------

+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Fixed | Available | Util% |
+-------------+------+-------+-----------+-------+
| BSCANE2     |    0 |     0 |         4 |  0.00 |
| CAPTUREE2   |    0 |     0 |         1 |  0.00 |
| DNA_PORT    |    0 |     0 |         1 |  0.00 |
| EFUSE_USR   |    0 |     0 |         1 |  0.00 |
| FRAME_ECCE2 |    0 |     0 |         1 |  0.00 |
| ICAPE2      |    0 |     0 |         2 |  0.00 |
| PCIE_2_1    |    0 |     0 |         1 |  0.00 |
| STARTUPE2  |    0 |     0 |         1 |  0.00 |
| XADC       |    0 |     0 |         1 |  0.00 |
+-------------+------+-------+-----------+-------+



7. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDRE     |  210 |        Flop & Latch |
| OBUF     |   39 |                  IO |
| IBUF     |   21 |                  IO |
| FDCE     |   21 |        Flop & Latch |
| DSP48E1  |   12 |    Block Arithmetic |
| LUT6     |    2 |                 LUT |
| LUT3     |    2 |                 LUT |
| LUT2     |    2 |                 LUT |
| LUT5     |    1 |                 LUT |
| LUT4     |    1 |                 LUT |
| LUT1     |    1 |                 LUT |
| BUFG     |    1 |               Clock |
+----------+------+---------------------+


8. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


9. Instantiated Netlists
------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```

Figure 27. Utilization Report for The Design

CONCLUSION

The optimized outputs from ACO are obtained. The gaussian distribution equation is converted into behavioral modelling of Verilog. These are the required coefficients given as an input for the Verilog code in behavioral modelling. Along with these obtained coefficients the FIR FILTER code is written in Verilog using Xilinx Vivado software. And this is convolved with the FIR code, which improved the effectiveness, decreased design error (ACO) and speed up execution process (DE) while discharging their limitations.

Hence by passing the optimized outputs from ACO and by giving them as input to FIR filter code in Verilog in behavioral modelling using Vivado software, we obtained an optimized VLSI ARCHITECTURE FOR FIR FILTER USING ANT COLONY OPTIMIZATION.

APPENDIX

A. OVERVIEW OF DIGITAL SYSTEM DESIGN AND VLSI ARCHITECTURE

EVOLUTION OF COMPUTER SYSTEM DESIGN

Digital circuit design has evolved rapidly over the last 25 years. The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. The first integrated circuit(IC) chips were SSI (Small Scale Integration) chips where the gate count was very small.

As technologies become sophisticated, designers were able to place circuits with hundreds of gates on a chip. These chips were called MSI (Medium Scale Integration) chips. With the advent of LSI (Large Scale Integration), designers could put thousands of gates on a single chip. At this point, design processes started getting very complicated, and designers felt need to automate these processes. Computer Aided Design (CAD) techniques began to evolve. Chips designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about 100 transistors. The circuits were still tested on the breadboard, and the layout was done on paper by hand on a graphic computer terminal.

With the advent of VLSI (Very Large Scale Integration) technology, designers could design single chips with more than 100,000 transistors. Because of the complexity of these circuits, it was not possible to verify these circuits on the breadboard. Computer-aided technologies became critical for verification and design of VLSI digital circuits. Computer programs to do automatic placement and routing of circuit layouts also became popular. The designers were now building gate-level digital circuits manually on graphic terminals. They would build small building blocks and then derive higher-level blocks from them. This process would continue until they had built the top-level block. Logic simulators came into existence to verify the functionality of these circuits before they were fabricated on chip.

DESIGN SPECIFICATION

Digital design flow begins with specification of the design at various level of abstraction. It describes the functionality, interface and overall architecture of digital circuits to be designed. The design specifications generally presented as a document describing a set of functionalities that the final solution will have to provide and set constraints that it must satisfy. At this point, architects do not need to think about how they will implement the circuit.

BEHAVIOUR DESCRIPTION

In this process, circuit details and electronic components are not specified. Instead, the behavior of each block at the highest level of abstraction is modeling. The behavioural approach to modeling hardware components is different from circuit design in that it does not necessarily reflect hoe the design is implemented. It is basically an algorithmic and black box approach to modeling. It accurately models what happens on the inputs and outputs of the box. For example, if you wish to simulate the operation of your custom design connected to a commercial part like a microprocessor. In this case, the microprocessor is complex and its internal operation is irrelevant (only the external behavior is important). Behaviour descriptions are important as they corroborate the integrity of design idea.

BEHAVIOUR DESCRIPTION TO RTL

The designer starts with an abstract description of the circuit called the behavioural model. This kind of description is used primarily to verify the methodology and functioning of the circuit. The next step is to transform this description something closer to electronic circuitry. In other words, the behavioural description needs to be converted to RTL (Registered transfer language) a functional or RTL description describes a circuit in terms of its registers and the combinational logic between the registers. This behavior synthesis can either be done manually or automatically by software, The essential goal of doing this is to use logic synthesizers that makes this form of description and synthesizes it to sets of registers and combinational logic, which can be readily shipped to FPGA.

FUNCTIONAL VERIFICATION AND TESTING

With the RTL design, the functional design of our digital system ends and its verification begins. From this point onward, the design process is done with the assistance of CAD tools. The underlying motivation is to remove all possible design errors before proceeding to the expensive chip manufacturing. Verification methodology still lacks any standard or even commonly accepted approach. The industrial approach to verification is functional validation. The functional model of the design is simulated with meaningful input stimuli and the output is checked for expected behavior. This model used for simulation is the RTL.

VLSI ARCHITECTURE

Very-large-scale integration (VLSI) is the process of creating an Integrated Circuit (IC) by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device.

Before the introduction of VLSI technology, most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.

The electronics industry has achieved a phenomenal growth over the last few decades, mainly due to the rapid advances in large scale integration technologies and system design applications. With the advent of very large scale integration (VLSI) designs, the number of applications of integrated circuits (ICs) in high-performance computing, controls, telecommunications, image and video processing, and consumer electronics has been rising at a very fast pace.

The current cutting-edge technologies such as high resolution and low bit-rate video and cellular communications provide the end-users a marvelous amount of applications, processing power and portability. This trend is expected to grow rapidly, with very important implications on VLSI design and systems design.

VLSI DESIGN FLOW

The VLSI IC circuits design flow is shown in the figure below. The various levels of design are numbered and the blocks show processes in the design flow.

Specifications comes first, they describe abstractly, the functionality, interface, and the architecture of the digital IC circuit to be designed.

Behavioral description is then created to analyze the design in terms of functionality, performance, compliance to given standards, and other specifications.

RTL description is done using HDLs. This RTL description is simulated to test functionality. From here onwards we need the help of EDA tools.

RTL description is then converted to a gate-level netlist using logic synthesis tools. A gatelevel netlist is a description of the circuit in terms of gates and connections between them, which are made in such a way that they meet the timing, power and area specifications.

Finally, a physical layout is made, which will be verified and then sent to fabrication.

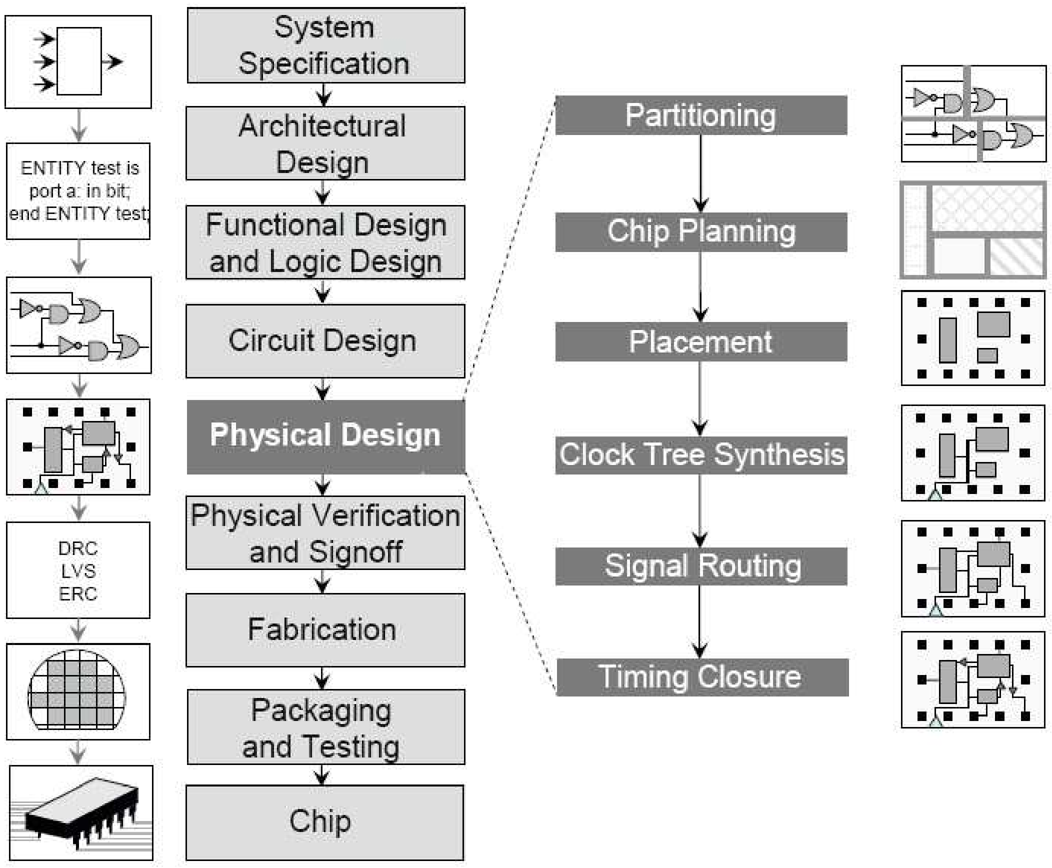


Figure 28. VLSI Design Flow

B. INTRODUCTION TO VERILOG

INTRODUCTION

Verilog, standardized as IEEE 1364, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in design and verification of digital circuits at the regular -transfer level of abstraction. It is also used in verification of analog circuits and mixed signal circuits HDL's allows the design to be simulated earlier in the design circuits in order to correct errors or experiments with different architectures.

Designs described in HDL are technology independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits. Verilog can be used to describe designs at four levels if abstractions:

1) Algorithmic level (much like as code if, case and loop statements).
2) Register transfer level (RTL uses registers connected by Boolean equations)
3) Gate level (interconnected AND, NOR etc.).
4) Switch level (the switches are MOS transistors inside gates).

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output and bidirectional ports.

Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies).

FEATURES OF VERILOG HDL

Verilog HDL offers many useful features for hardware design

- Verilog (verify logic) HDL is general purpose hardware description language that is easy to learn and easy to use. It is similar in syntax to C programming language. Designers with C programming experience will find it easy to learn Verilog HDL.
- Verilog HDL allows different levels of abstraction to be mixed in the same model. Thus a designer can define a hardware model in terms of switches, gates, RTL, or behaviour

code. Also a designer needs to learn only one language for stimulus and hierarchical designs.

- Most popular logic synthesis tools support Verilog HDL. This makes it the language of choice for designers.
- All fabrication vendors provide Verilog HDL libraries for post logic synthesis simulation. Thus designing a chip in Verilog HDL allows the widest choice of vendors.
- The Programming language interface (PLI) is a powerful feature that allows the user to custom C code to interact with the internal data structures of Verilog. Designers can customize a Verilog HDL simulator to their need with the Programming language interface (PLI).

MODULE DECLARATION

A module is the principal design entity in Verilog. The first line of a module declaration specifies the name and port list (arguments). The next few lines specifies the I/O type (**input**, **output** or **inout**) and width of each port. The default port with is 1 bit. Then the port variables must be declared **wire**, **reg**. The default is **wire**. Typically inputs are **wire** since their data is latched outside the module. Outputs are type **reg**if their signals were stored inside always or **initial** block.

Syntax

module model_ name(port_list);

input[msb:lsb] input_port_list;

output[msb:lsb] output_port_list;

inout[msb:lsb] inout_port_list;

…….statements…………

endmodule

Example

module add_sub(add, in1, in2, oot);

inputadd;//defaults to wire

input[7:0] in1, in2**,** ***wire*** in1, in2;

Output[7:0] oot;

regoot;

………statements……..

Endmodule

VERILOG MODELLING

Verilog has four levels of modelling:

1) The switch level Modeling.
2) Gate- level Modeling.
3) The Data-Flow level.
4) The behavioural or procedural level.

Switch level Modeling:

A circuit is defined by explicity showing how to constrct it using transistors like pmos and nmos, predefined modules.

Example

module inverter (out,in);

output out;

input in;

Supply0gnd;

Supply1vdd;

nmosx1(out, in, gnd);

pmosx2(out, in, vdd);

endmodule

Gate level modelling

A circuit is defined by explicitly showing how to correct it using logic gates. Predefined modules, and the connections between them. In this first we think of our circuit as a box or module which is encapsulated from its outer environment, in such a way that its only communication with the outer environment, is through input and output ports. We then set out to describe structure within the module by explicitly describing its gates and sub modules, and how they connect with one another as well as to the module ports.In other words, structural modelling is used to draw a schematic diagram for the circuit. As an example, consider the full-adder below.

Example

```
module fulladder (a, b, sum, Cout);
input a, b;
output sum, Cout;
xor x1(a, b, y);
xor x2(a, b, y);
endmodule
```

<u>Data-flow modelling</u>

Dataflow modeling uses Boolean expressions and operators. In this we use assign statement.

Example

```
module fulladder (a, b, sum,  Cout);
input a, b;
output sum, Cout;
assign sum=a^b;
assign Cout=a^b;
endmodule
```

<u>Behavioural modeling</u>

It is higher level of modeling where behaviour of logic is modelled. Verilog behavioural Code is inside procedure blocks, but there is an exception:some behavioural code also exist outside procedure blocks. There are two types of procedural blocks in Verilog .

Initial: initial blocks execute only once at time zero (start execution at time zero)
Always: always blocks loop to execute over and over again; in other words, as other words as the name suggests, it executes always.

An always statement executes repeatedly, it starts and its execution at other 0 ns

Syntax: *always@* (sensitivity list)

```
Begin
 Procedural statements
  end
```

Example

```
module fulladder (a, b, clk, sum);
input a, b, clk;
 output sum;
 always@ (posedgeclk)
  begin
 sum= a+b;
 endmodule
```

SOFTWARE DESIGN AND DEVELOPMENT

A description of the hardware's structure and behaviour is written in a high-level hardware description language (usually VHDL or Verilog) and those codes is then compiled and downloaded prior to execution. Although schematic capture can be used for design entry but due to more complex designs and the improvement of the language-based tools it has become less popular.

The most distinct difference between hardware and software design is the way a developer must think about the problem. Software developers tend to think sequentially, even when they are tasked to program a multithread application. Most of the time, the source code is always executed in that order. At the design entry phase, hardware designers must think and program in parallel.

All of the input signals are processed in parallel: inside each ore is a series of macro cells and interconnections routed toward their destination output signals. Therefore, the statement of a hardware description language creates structures, all of which are process at the very same time. (Normally the link between each macro cell to another macro cell usually -synchronized to some other signal, like a common clock).

In a typical design, after each design entry is completed, the next step is to perform periods of functional simulation. This is where a simulator comes in place. It is used to execute the design and confirm that the correct/required outputs are produced for a given set of test inputs.

This step is to ensure the designer that his/her logic is functionally correct before going on to the next stage development. This is a good practice as compared to simulating a full-scale design entry. As the design entry gets more complex, the troubleshooting will be much difficult and time consuming.

SOFTWARE TOOLS USED

XILINX VIVADO

MATLAB

VIVADO enables developers to synthesize their designs, perform timing analysis examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Vivado is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors.
MATLAB is used for proof of concept.

LANGUAGE SUPPORT

The Vivado High-Level Synthesis compiler enables C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to manually create RTL. Vivado HLS is widely reviewed to increase developer productivity, and is confirmed to support C++ classes, templates, functions and operator overloading.
Xilinx vivado enables simulation, verification and synthesis for the following languages

VHDL

Verilog

System Verilog

C. XILINX VIVADO

XILINX VIVADO ISE DESIGN SUITE (16.1version)

Xilinx is a powerful software tool that is used to design, synthesize, simulate, test and verify digital circuit designs. The designer can describe the digital design by either using the schematic entry tool or a hardware description language. In this software we will create VHDL design input files – the hardware description of the logic circuit, compile VHDL source files, create a test bench and simulate the design to make sure of the correct operation of the design (functional simulation). The purpose of this is to give new users an exposure to the basic and necessary steps to implement and examine your own designs using ISE environment. In this, we will design one simple module (OR gate); however, in the future, you will be designing such modules and completing the overall circuit design from these existing files. A VHDL input file in the Xilinx environment consists of: Entity Declarations: module name and interface specifications (I/O) – list of input and output ports; their mode, which is direction of data flow; and data type. Architecture: defines a component's logic operation.

There are different styles for the architecture body: (i)Behavioural – set of sequential assignment statements (ii) Data Flow – set of concurrent assignments o Structural – set of interconnected components A combination of these could be used, but in this tutorial we will use Dataflow. In its simplest form, the architectural body will take the following format, regardless of the style: architecture architecture_name of entity_name is begin … -- statement end architecture_name;

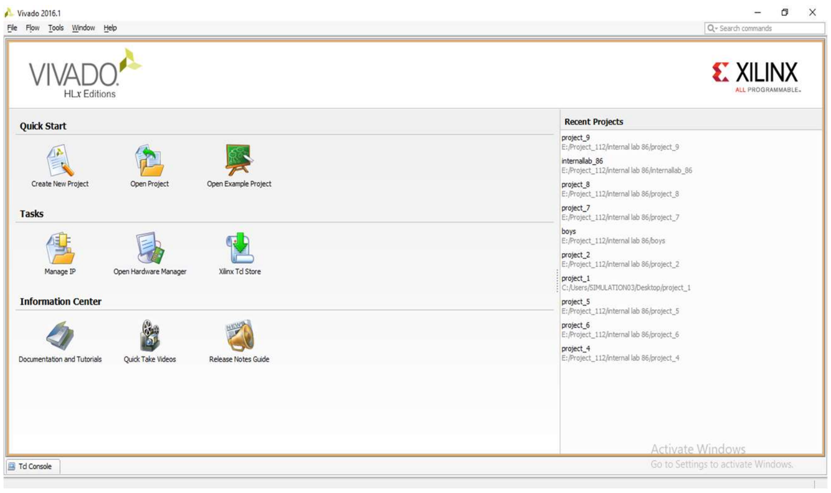
ISE (Integrated Software Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different simuli, and configure the target device with the programmer.

Xilinx is an American technology company, primarily a supplier of programmable logic devices. It is known for inventing FPGA.

The Xilinx ISE is primarily used for circuit synthesis and design, while the Modelsim logic simulator is used for system-level testing.

ISE PROJECT NAVIGATOR

In this section, we introduce the reader to the main components of an "ISE Project Navigator" window, which allows us to manage our design files and move our design process from creation to synthesis and to simulation phase.



project_9
E:/Project_112/internal lab 86/project_9
internallab_86
E:/Project_112/internal lab 86/internallab_86
project_8
E:/Project_112/internal lab 86/project_8
project_7
E:/Project_112/internal lab 86/project_7
boys
E:/Project_112/internal lab 86/boys
project_2
E:/Project_112/internal lab 86/project_2
project_1
C:/Users/SIMULATION03/Desktop/project_1
project_5
E:/Project_112/internal lab 86/project_5
project_6
E:/Project_112/internal lab 86/project_6
project_4
E:/Project_112/internal lab 86/project_4

Activate Windows
Go to Settings to activate Windows.

Figure 29. Xilinx Vivado Project Navigation Window

By opening the Xilinx vivado ISE suite, we will come to see the 3 main points. They are

1) Quick start
2) Tasks
3) Information Center

In the Quick start block, We have create a new project, open project and open example project.

In the Tasks, We have Manage IP, open hardware manager, xilinx Td store.

In the Information center, we have documentation and tutorials,quick take videos and release notes guide.

This section describes the four basic steps to working with a project.

Step 1—— Creating a New Project

This creates .xpr file and a working library.

Step 2—— Adding Items to the project

Projects can reference or include source files, folders for organizations, simulations, and any other files you want to associate with the project. You can copy files into the project directory or simply create mappings to files in other locations.

Step 3—— Compiling the Files

This checks syntax and semantics and creates the pseudo machine code that Viavdo uses for simulation.

Step 4—— Simulating a Design

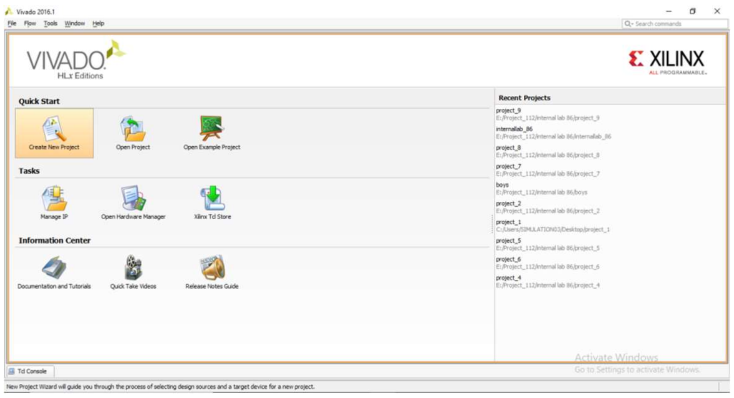
This specifies the design unit you want to simulate and opens a structure tab in the workspace pane.

you specify will be used to create a working library subdirectory within the Project

In order to start ISE double click the desktop icon: Or click**:**

<u>**Creating a New Project**</u>

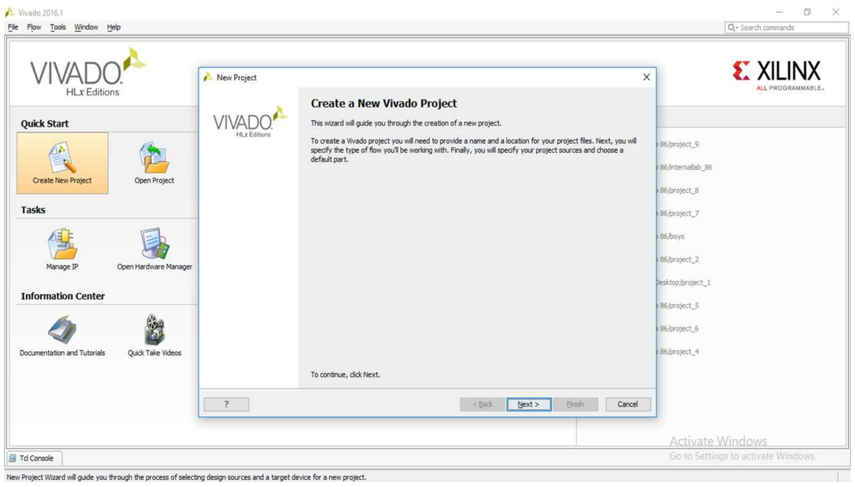
After launching Vivado, from the startup page click the "Create New Project" icon. Alternatively, you can select **File -> New Project**



New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

Figure 30. Creating New Project Window

The New Project wizard will launch, click the "Next >" button to proceed



Create a New Vivado Project

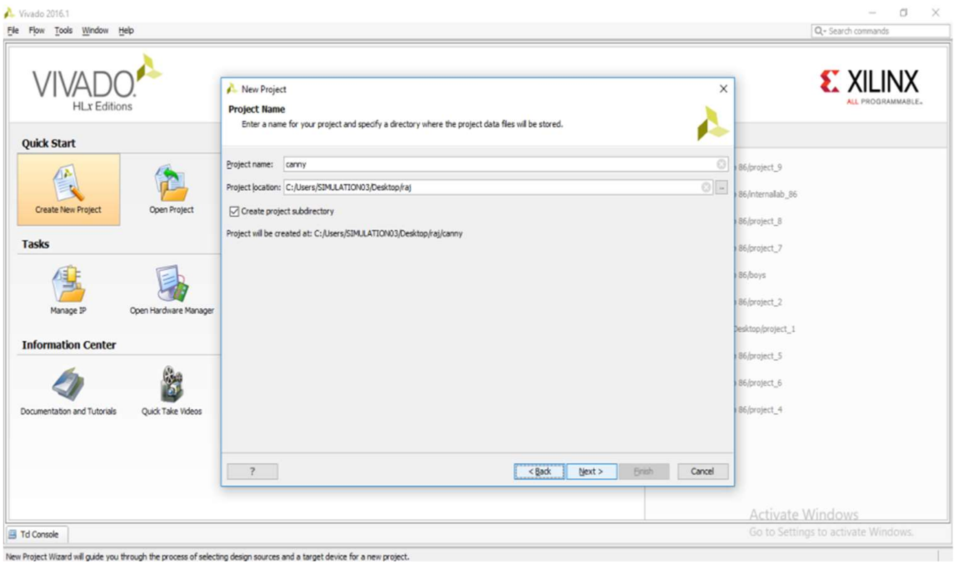
This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, click Next.

Figure 31. Guiding Wizard of the Project

Enter a project name and select a project location. **Make certain there are NO SPACES in either!** It's not a bad idea to only use letters, numbers, and underscores as well. If necessary simply create a new directory for your Xilinx Vivado projects in your root drive (e.g. C:\Vivado). You will likely always want to select the "Create project sub-directory" check-box as well. This keeps things neatly organized with a directory for each project and helps avoid problems. Click the "Next >" button to proceed.



New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: canny

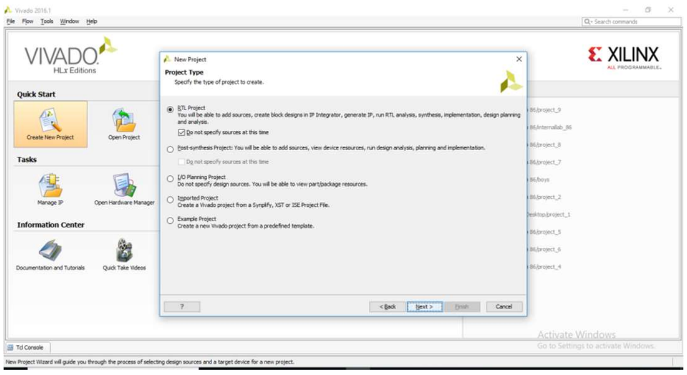
Project location: C:/Users/SIMULATION03/Desktop/raj

[✓] Create project subdirectory

Project will be created at: C:/Users/SIMULATION03/Desktop/raj/canny

Figure 32. Creating a Project Name

Select the "RTL Project" radial and select the "Do not specify sources at this time" check-box. If you don't select the check-box the wizard will take you through some additional steps to optionally add pre existing items such as VHDL or Verilog source files, Vivado IP blocks, and .XDC constraint files for device pin and timing configuration. For this first project you will add the necessary items later. Click the "Next >" button to proceed.



New Project

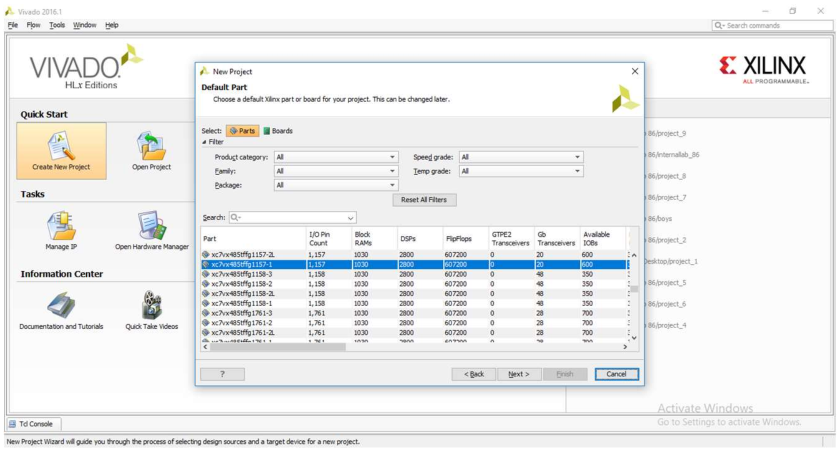
Project Type

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
- 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.

Figure 33. Specifying the RTL project

You need to filter down to and select the specific part number for your project. You can physically read the markings on your chip or refer to your board's documentation to find its part number. In the case of the Basys 3 it's the Artix-7 chip that's on the board, and the filters shown will help you get to the correct device that's highlighted. Once you select the correct device click the "Next >" button to proceed.


New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts Boards

Filter

Product category: All
Family: All
Package: All
Speed grade: All
Temp grade: All

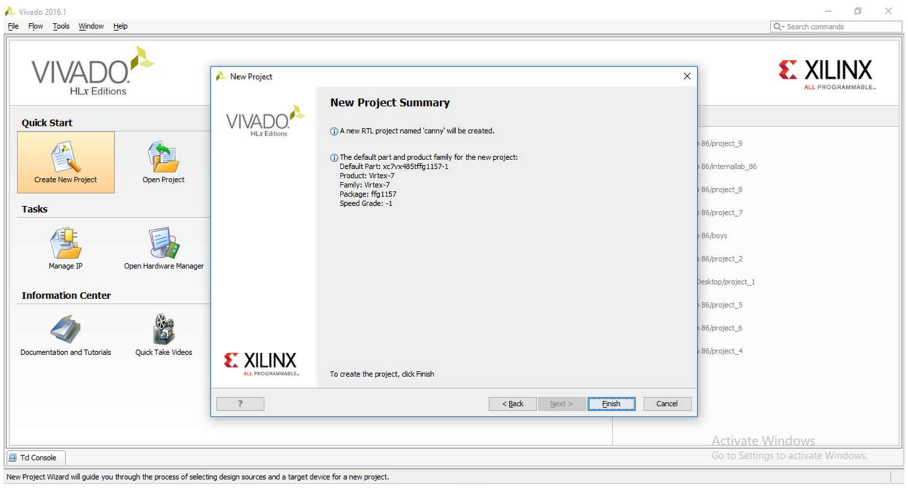
Search:

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs
xc7vx485tffg1157-2L	1,157	1030	2800	607200	0	20	600
xc7vx485tffg1157-1	1,157	1030	2800	607200	0	20	600
xc7vx485tffg1158-3	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1158-2	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1158-2L	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1158-1	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1761-3	1,761	1030	2800	607200	0	28	700
xc7vx485tffg1761-2	1,761	1030	2800	607200	0	28	700
xc7vx485tffg1761-2L	1,761	1030	2800	607200	0	28	700

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

Figure 34. Choosing a Board for The Project

Click the "Finish" button and Vivado will proceed to create your project as specified.


New Project

New Project Summary

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

The default part and product family for the new project:
Default Part: xc7vx485tffg1157-1
Product: Virtex-7
Family: Virtex-7
Package: ffg1157
Speed Grade: -1

To create the project, click Finish

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

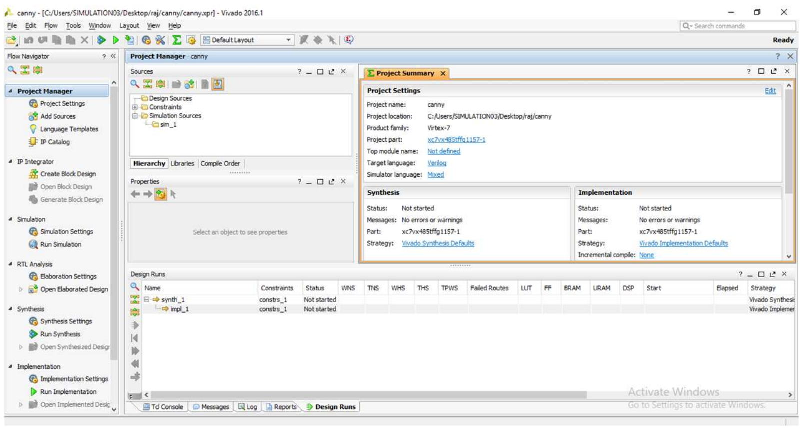
Figure 35. Project Summary

STEPS FOR DESIGN ENTRY

Working through the Basic Project Flow

The Vivado project window contains a lot of information, and the information displayed can change depending on what part of the design you currently have open as you work through the steps of your project. Keep this in mind as you work through this guide, because if you don't see a specific sub-window or sub-window tab it's possible you aren't in the correct part of the design.

The "Flow Navigator" on the left side of the screen has all the major project phases organized from top to bottom in their natural chronological order. You begin in the "Project Manager" portion of the flow and the header at the top of the screen next to the Flow Navigator reflects this. This header and the corresponding highlighted section in the Flow Navigator will tell you which phase of the design you have open.


canny - [C:/Users/SIMULATION03/Desktop/raj/canny/canny.xpr] - Vivado 2016.1

Project Settings
Project name: canny
Project location: C:/Users/SIMULATION03/Desktop/raj/canny
Product family: Virtex-7
Project part: xc7vx485tffg1157-1
Top module name: Not defined
Target language: Verilog
Simulator language: Mixed

Synthesis
Status: Not started
Messages: No errors or warnings
Part: xc7vx485tffg1157-1
Strategy: Vivado Synthesis Defaults

Implementation
Status: Not started
Messages: No errors or warnings
Part: xc7vx485tffg1157-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

Design Runs
Name	Constraints	Status	Strategy
synth_1	constrs_1	Not started	Vivado Synthesis
impl_1	constrs_1	Not started	Vivado Implemen

Figure 36. Main Window for The Project

<u>Project Manager</u>

Project Settings:

Begin by clicking on "Project Settings" under the Project Manager phase of the Flow Navigator

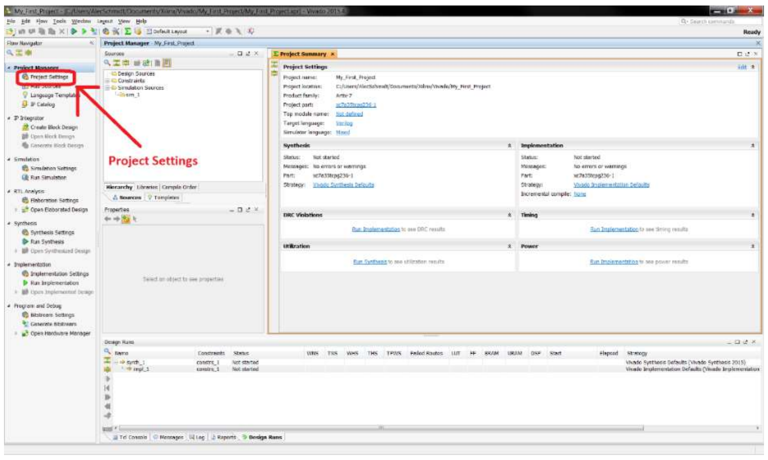
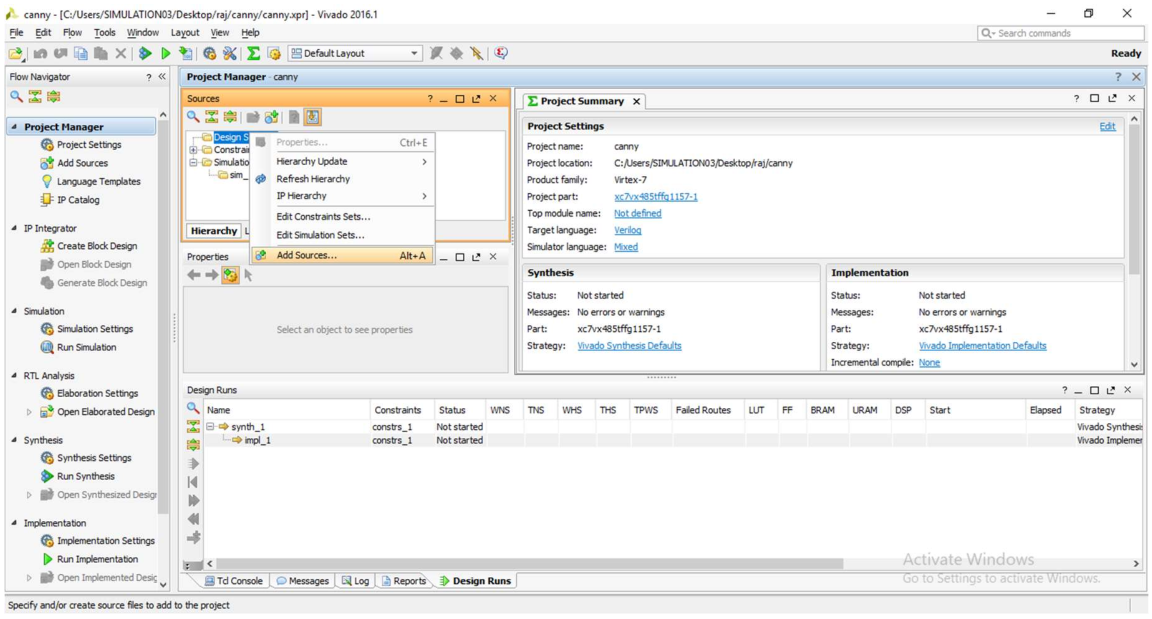


Figure 37. Project Settings Window

There are a lot of settings available here for all phases of the project flow, but for now just select "System Verilog" from the drop-down for the "Target language" in the "General" project settings and click the "OK" button.

Add Sources:

Now click on "Add Sources" under the Project Manager phase of the Flow Navigator



Project Settings

Project name:	canny
Project location:	C:/Users/SIMULATION03/Desktop/raj/canny
Product family:	Virtex-7
Project part:	xc7vx485tffg1157-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7vx485tffg1157-1
Strategy:	Vivado Synthesis Defaults

Implementation

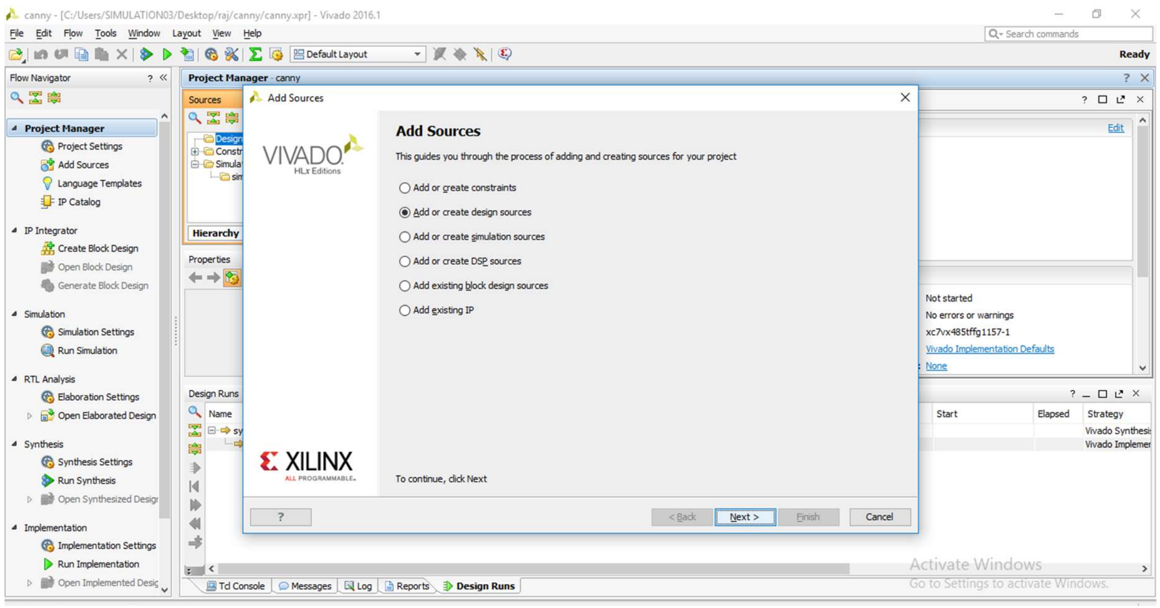
Status:	Not started
Messages:	No errors or warnings
Part:	xc7vx485tffg1157-1
Strategy:	Vivado Implementation Defaults
Incremental compile:	None

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Strategy
synth_1	constrs_1	Not started														Vivado Synthesis
impl_1	constrs_1	Not started														Vivado Implemen

Figure 38. Adding Source Files

Select the "Add or create design sources" radial and then click the "Next >" button



Add Sources

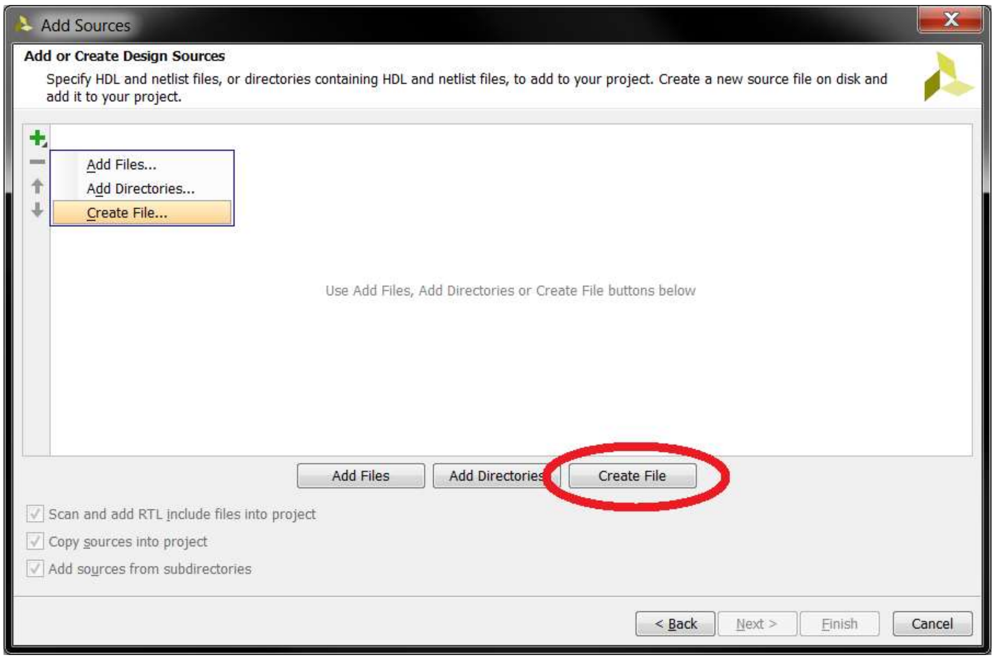
This guides you through the process of adding and creating sources for your project

- ○ Add or create constraints
- ● Add or create design sources
- ○ Add or create simulation sources
- ○ Add or create DSP sources
- ○ Add existing block design sources
- ○ Add existing IP

To continue, click Next

Figure 39. Wizard That Shows the Design Rules

Click the “Create File” button or click the green “+” symbol in the upper left corner and select the “Create File…” option


Add Sources

Add or Create Design Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

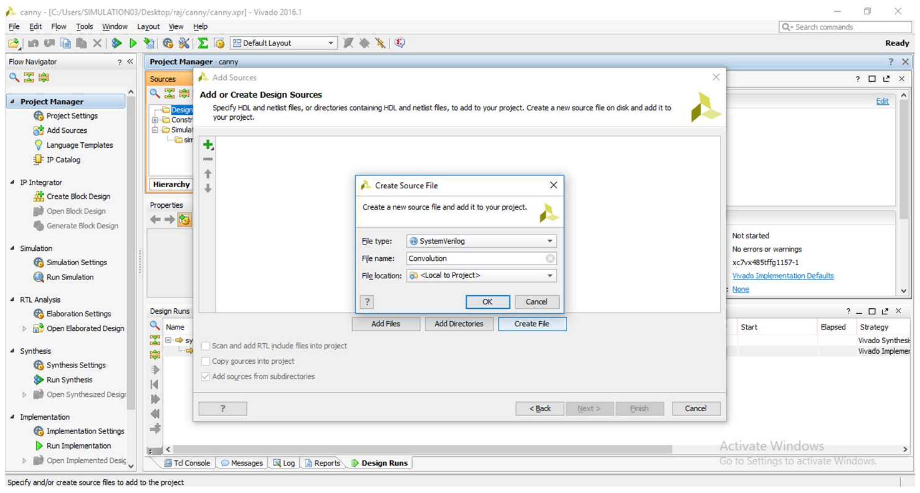
Use Add Files, Add Directories or Create File buttons below

- [✓] Scan and add RTL include files into project
- [✓] Copy sources into project
- [✓] Add sources from subdirectories

Figure 40. Creating New Name for New Design

Make sure the options shown are selected in the “Create Source File” popup, and for the sake of following along enter “convolution(Gaussian filter)” for the “File name”. Click the “OK” button when finished.

You can normally enter anything you like for the “File name” as long as it’s valid, but **always make certain there are NO SPACES!**


Create Source File

Create a new source file and add it to your project.

File type: SystemVerilog

File name: Convolution

File location: <Local to Project>

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [✓] Add sources from subdirectories

Figure 41. Selecting The Type of File And Location

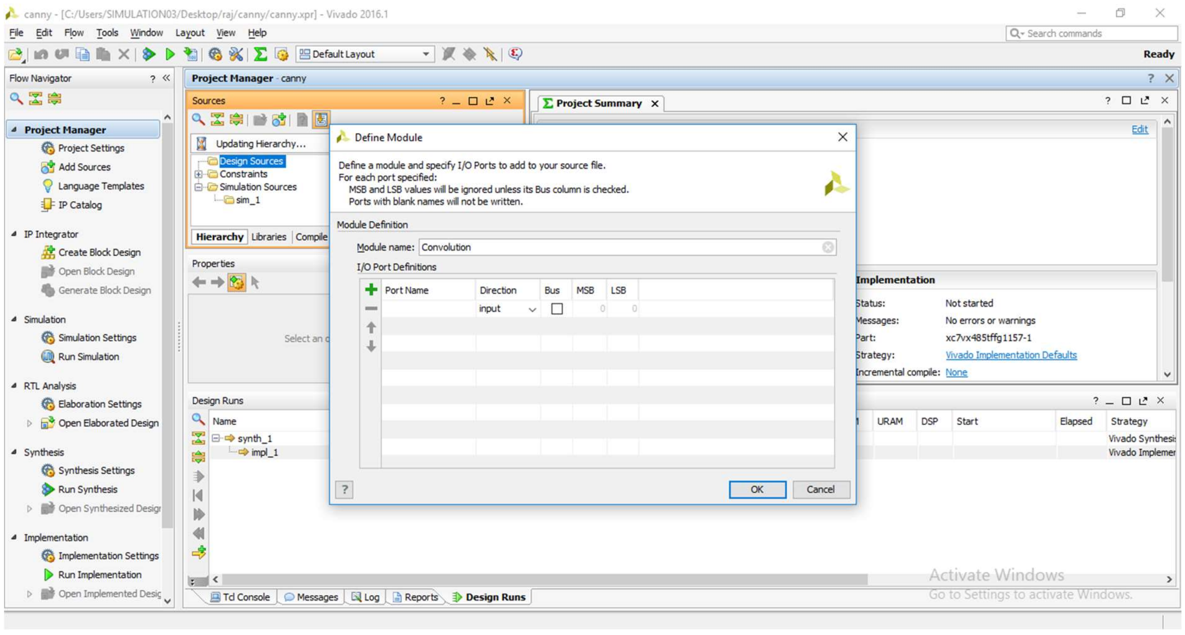
Click the “Finish” button and Vivado will then bring up the “Define Module” window.

Define Module:

You can use the “Define Module” window to automatically write some of the VHDL code for you. Additional “I/O Port Definitions” can be added by either clicking the green “+” symbol in the upper left or by simply clicking on the next empty line. The “Entity name” and “Architecture name” will be the corresponding Verilog HDL identifiers used in the code, as will whatever is typed in for each “Port Name”. Any valid verilog HDL identifier can be used for any of these, but for the sake of following along enter the information as shown. Make sure the proper “Direction” is set for each. Click the “OK” button when finished.

Note that if you would rather write your own code from scratch you can simply click the “Cancel” button and Vivado will create a completely blank System verilog VHDL source file inside your project. If you click the “OK” button without defining any “I/O Port Definitions” Vivado will still write the basic Verilog HDL code structure but the port definition will be empty and commented out for you to uncomment and fill later.

Also note that the port names here match the silkscreen reference designators of the switches and LEDs on the Basys 3 board that will be utilized for the example. This is for the convenience of those following along with the Basys 3, but should not be inferred as a requirement by beginners; each name is simply an arbitrary identifier.


canny - [C:/Users/SIMULATION03/Desktop/raj/canny/canny.xpr] - Vivado 2016.1
File Edit Flow Tools Window Layout View Help
Default Layout
Ready
Flow Navigator
Project Manager - canny
Sources
Project Summary
Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: Convolution

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0

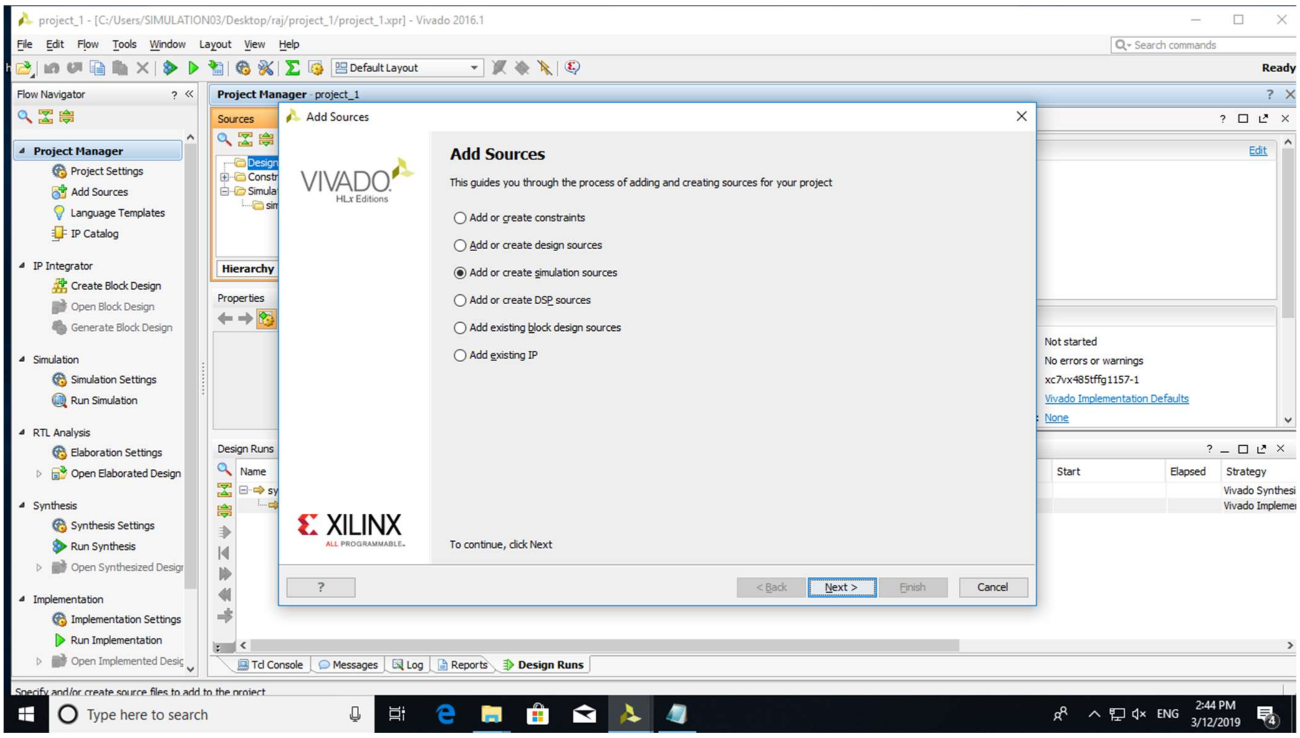
Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7vx485tffg1157-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

Figure 42. Module Defining with Ports

The System Verilog HDL source file generated will be added to your project in the "Design Sources" folder as shown. Double click it and it will open up in a new tab for you to view/edit. All the code here was generated by the previous "Define Module" window, and for this example you only need to manually enter the three highlighted lines between the "begin" and "end" keywords.

If we want to create a simulation source, we have to select a new simulation source by right clicking the add source block in the panel.



Add Sources

This guides you through the process of adding and creating sources for your project

- ○ Add or create constraints
- ○ Add or create design sources
- ● Add or create simulation sources
- ○ Add or create DSP sources
- ○ Add existing block design sources
- ○ Add existing IP

To continue, click Next

Not started
No errors or warnings
xc7vx485tffg1157-1
Vivado Implementation Defaults
None

Specify and/or create source files to add to the project

Figure 43. Creating the Simulation Source

REFERENCES

1. Tintu mary john, Chacko, Shanty (2020) , " Efficient VLSI Architecture For FIR FILTER Design Using Modified Differential Evolution Ant Colony Optimization Algorithm" Emerald publications.
2. Kumar, N.R. (2018), "A review of low-power VLSI technology developments", Innovations in Electronics and Communication Engineering, Springer, pp. 17-27.
3. Kaur, S. Singh, B. and Singh, M. (2016), "Different design approaches for the optimization of FIR filter coefficients", International Journal of Engineering Research & Technology (IJERT), Vol. 1 No. 7.
4. Rani, T. and Bansal, P. (2017), "Applications and variants of ant colony optimization in design of digital filters: a review", International Journal of Electronics, Electrical and Computational System.
5. Xu et al. (2011) proposed an empirical approach depending on ACO for the design of multiplier-less FIR filters with signedpower-of-two (SPT) coefficient. The number of adders needed for filtering with constant SPT coefficients can be reduced by minimizing non-zero SPT, through gathering the essential specifications of the filter.
6. Sasahara and Suyama (2017) studied design of FIR filters with CSD coefficients using ACO technique.
7. Chattopadhyay et al. (2014) proposed DE-optimized filter as a pulse-shaping filter in a Quadrature Phase Shift Keying modulated system.
8. Pusegaonkar et al. (2014) implemented low power digit-serial FIR filters using Multiple Constant Multiplication techniques.
9. Aggarwal et al. (2013) implemented the FIR filter using Simulink and FPGA which increases the speed of operation by its parallel processing capability.
10. Kaur et al. (2016) discussed the designing approaches for linear phase FIR filter. They also compared the existing systems in terms of maximum stop band ripple, maximum pass band ripple, transition width, and maximum stop band attenuation.

PUBLICATION

1. N.Pujitha Vaidya, Ch.Rohit, Ch.Phani, Y.Sarvendra and Srinivas Sabbavarapu, ”ACO Based FIR Filter Implementation on FPGA”, Manuscript submitted to Intelligent Systems with Applications (ISWA).