VLSI Design and Implementation of Statistical Multiplexer

Abhilasha Rani Goel

Electronics and Communication Engineering

Jaypee University of Information Technology

This dissertation is submitted for the degree of

M.Tech

2014

I would like to dedicate this thesis to my loving parents ...

Certificate

This is to certify that the work titled **"VLSI Design and Implementation of Statistical Multiplexer"** submitted by **"Abhilasha Rani Goel"** in partial fulfillment for the award of degree of M.Tech of Jaypee University of Information Technology, Waknaghat has been carried out under our supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor	
Name of Supervisor	Mr. Akhil Ranjan
Designation	Assistant Professor (Grade II)
Date	

Signature of Supervisor	
Name of Supervisor	Mr. Mohd. Wajid
Designation	Assistant Professor (Grade II)
Date	

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation “VLSI Design and Implementation of Statistical Multiplexer with QoS Parameter Control” are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Abhilasha Rani Goel
2014

Acknowledgements

No work is a single man's success. Apart from the hard work and personal efforts, which are key ingtedients, it requires the knowledge, guidance, encouragement and support. A work on completion signifies the joint efforts of the persons involved in it.

The satisfaction that accompanies the successful completion of every task during this project work wolud be incomplete without the mention of the people who made it possible. I consider it my privilege to express my gratitude and respect to all who guided, inspired and helped me in completion of my project work.

I am extremely thankful to my supervisor **Mr. Akhil Ranjan**, assistant professor and my co-guide **Mr. Mohd. Wajid**, assistant professor of ECE department, JUIT for their valuable guidance, constant support, keen interest and moral encouragement which has helped me to bring out this project in the present shape.

I wish to express my gratitude and high regards to **Prof. Dr. Sunil Bhooshan** head of ECE department, JUIT.

I express my heart-felt thanks to **Dr. D. S. Saini** for his kind encouragement in completion of the project work.

I would like to express my profound sense of gratitude to all the faculty members for their cooperation and encouragement throughout my course.

I am also grateful to the Department Library, JUIT, for providing me the required books and also grateful to all lab incharges for their support.

I am also thankful to my family and freinds for their love, affection and their support, which has been a constant source of encouragement in all my educational pursuits.

Abhilasha Rani Goel
2014

Abstract

Next Generation technology has the need of a very high speed information networks. When these networking services are built for end users, minimization of available resources without compromising on quality is an important aspect of the technological era. Till now, the most common form of multiplexing is that of time division multiplexing (TDM). In this multiplexing technique , each source is allocated a small time period to transmit information of fixed length. The packets available on the channel result from different sources, which send the packets simultaneously. The packets of the different sources are mixed on the channel e.g.- a packet is transmitted for first source, then a packet for a third source, next for the first, then two packets in a row for second source, and so on. This technique of mixing is referred to as “Statistical Multiplexing”. A statistical multiplexer has the advantage that all channels will not have peak demand of bit rate at the same time. Channel which has more need gets more bandwidth and less needed channels will be allocated less bandwidth, which provides the better quality of service (QoS).

In this technological era, QOS is an important parameter, which a user has to consider for each and every application. Implementing QOS issue is much more complicated because quality requirements differ from user to user. One critical problem in networking world is that how many users can be allowed in the network without diminishing the QOS levels.

Generally hardware implementation is faster than software implementation, because it takes less time to perform an algorithm than software. So, we proposed VLSI hardware architecture of statistical multiplexer and implemented on FPGA using Xilinx ISE. The data and coefficient widths are adjustable in the range 8 to 32. It has pipelined mode operation, each result is outputted in one clock cycle. The period of the clock is very less roughly in the range of ‘ns’, the frequency of the hardware will be in 100 MHZ to 1 GHz and now a days in communication this range of frequency is used.

Contents

List of Figures

List of Tables

Chapter 1

Literature review

1.1 Introduction

In twenty-first century communications will be directed to intelligent high-speed information networks. Networks will be enable to integrate and transmit media-rich services within specified standards of delivery due to all over access to the network through wired and wireless technology, adaptable network systems, and broadband transmission capacities. This vision has developed the standardization and evolution of broadband integrated services network (B-ISDN) technology since the early 1990s. The narrowband ISDN proposal for integrating voice, data, and video on the telephone line was the pioneer to broadband services and asynchronous transport mode (ATM) technology [1]. For efficient utilization of the transmission link capacity, integrated services traffic is multiplexed onto a single common channel. This concept is the basis of design of ATM and ISDN network. In ATM, the time slots of time- division multiplexed frame are allocated asynchronously for servicing the variable-bit-rate (VBR) traffic generated from video, voice and data services. The transport of information is done by using fixed-size cells of 53 bytes in length and the application of fast cell-switching architectures. It uses the concepts of both circuit and packet switching by creating virtual circuits that carry VBR streams generated by multiplexing ATM cells from voice, video, and data sources [2, 3].

The ATM architecture is designed to efficiently transport traffic sources that alternate between bursts of transmission activity and periods of no activity means bursty information which is shown in fig. 1.1 [4]. It handles various classes of traffic with different bit-rates and quality of service (QoS) requirements and the traffic source with continuously changing transmission rates. For example, voice traffic has the bit-rate of several Kbps and is delay sensitive, while high-speed data traffic used for file transfer or LAN interconnection has the

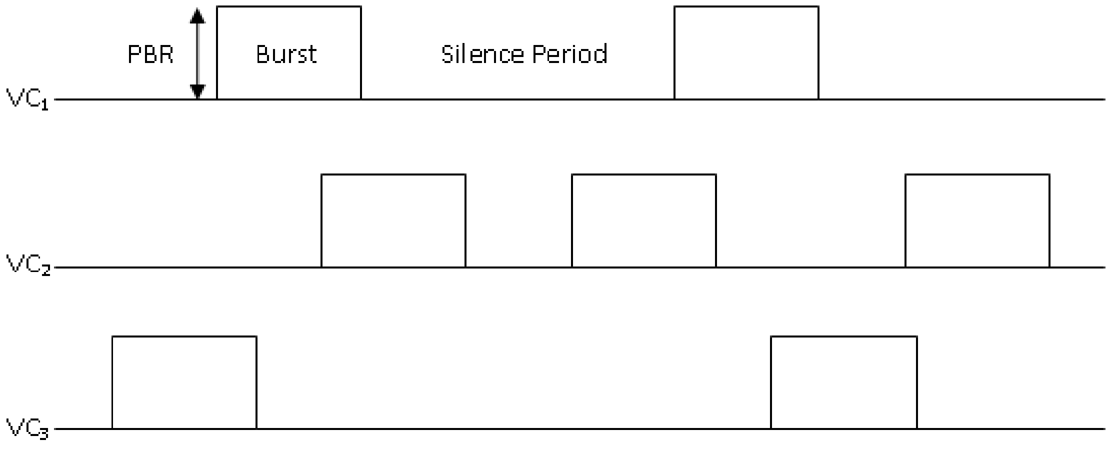


Fig. 1.1 Burst Model

bit-rate of hundreds of Mbps and is loss sensitive. Taffic burstiness is measured in terms of the ratio of peak to average rate of source. A circuit-switched network allocates capacity to each source equal to its peak rate. In this case, full resource utilization takes place only when all of the sources transmit at their peak rates. This is a low-probability event when the sources are statistically independent of each other [5]. However, statistical multiplxer allocates tha capacity between the peak rate and the average rate. When demand exceeds the channel capacity, the traffic is stored in the buffer. The multiplexed traffic should have a smaller variance in the limit of mean rate as the number of sources multiplexed increase to a large value. So the probability of occurrence of source rates that are greater than the available capacity is decreased.

Statistical multiplexer has been an integral component in packet switches and routers on data network since the 1960s. But, the use of statistical multiplexer increases since 1990 with the availability of broadband transmission speeds exceeding 155 Mbps and ranging upto 10 Gbps in the core of the network. However, since broadband traffic features are highly unpredictable, the quality of service such as packet delay and loss probabilities must be controlled by intelligent and adaptable protocols. These protocols include integrated (Intserv) and differentiated (Diffserv) services, multiprotocol label switching (MPLS), and resource reservation protocols (RSVPs). The design and performance of these protocols and services depend on the traffic patterns of voice, video and data sources and their influence on queues in statistical multiplexers [6].

1.2 Multiplexing

Many applications generate the information in a bursty manner. These bursts of information are formatted into packets and then transmitted over a communication line. If we assign a dedicated line to each user it becomes highly inefficient basically in long distance communication. Because, the bursty data has the long idle periods between the bursts so it is possible to send the data from all users over a single communication channel. Secondly, most of the individual data-communicating devices require modest data rate. But, communication media usually have much higher bandwidth. As a consequence, two communicating stations do not utilize the full capacity of a data link. Moreover, when many nodes compete to access the network, some efficient techniques for utilizing the data link are very essential [7, 8]. When the bandwidth of a medium is greater than individual signals to be transmitted through the channel, a medium can be shared by more than one channel of signals. So, these two things develop the idea of "Multiplexing". Basically, multiplexing means mixing, it mix the information from two or more sources and transmits over a single communication link. It is efficient in terms of capacity, resource utilization and cost. Most common use of multiplexing is in long- haul communication using coaxial cable, microwave and optical fibre. Multiplexing is the technique where two or more separate communications channels are supported across a single transmission medium. A well known example from the telephone network is the support of multiple telephone conversations on a single high bandwidth trunk [9, 10].

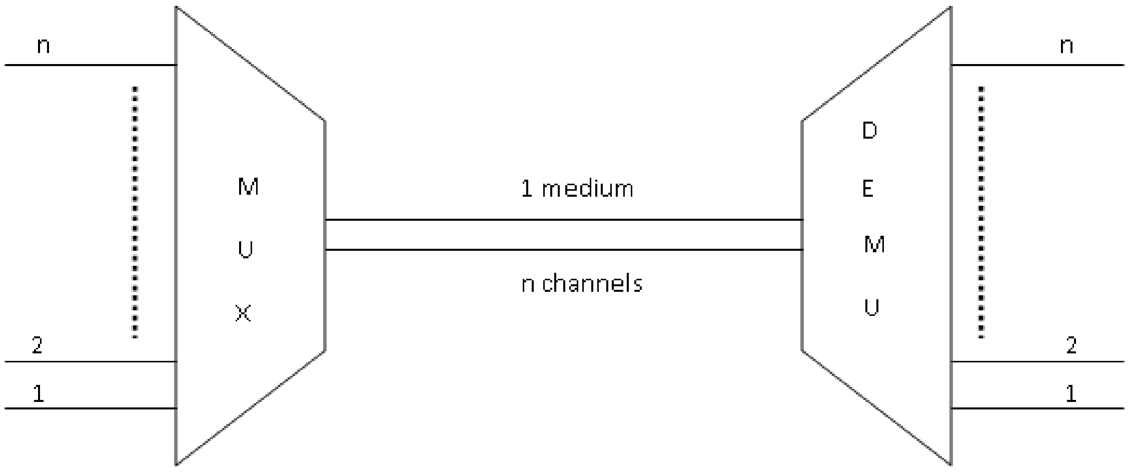


Fig. 1.2 Basic Concept of Multiplexing

Fig. 1.2 depicts the functioning of multiplexing in general. Multiplexing is done by using a device called multiplexer (MUX) that combines n input lines to genrate one output line i.e (many to one).Therefore multiplexer (MUX) has serval inputs one output. On the other hand, the demultiplexing is done using a device called demultiplexer (DEMUX) that splits 1 input line to generate n output lines i.e. (one to many). The multiplexer is connected to the demultiplexer by a single data link. The multiplexer combines (multiplexes) data from these 'n' input lines and transmits them through the high capacity data link, which is being demultiplexed at the other end and is delivered to the appropriate output lines. There are many types of multiplexing named as follow:

1. Frequency Division Multiplexing (FDM)
2. Wavelength Division Multiplexing (WDM)
3. Time Division Multiplexing (TDM)
   (a) Synchronous Time Division Multiplexing
   (b) Statistical or Asynchronous Time Division Multiplexing
4. Space Division Multiplexing (SDM)
5. Code Division Multiplexing (CDM)

Early multiplexing systems for use in the analogue telephone network employed frequency division multiplexing (FDM) in which each separate channel was transmitted at a different carrier frequency. An analogous technique currently being developed for use in optical communications systems is wavelength division multiplexing (WDM) in which the various channels are carried on different optical wavelengths. In digital communications systems by far the most common form of multiplexing is that of time division multiplexing (TDM) [11, 12].

1.3 Time Division Multiplexing

Time-division multiplexing (TDM) is a digital process in which all signals operate with same frequency at different times. It allows several connections to share the high bandwidth of a link. Each connection occupies a portion of time in the link. This is a base band transmission system, where an electronic commutator sequentially samples all data source and combines them to form a composite base band signal. This composite base band signal travels through the shared medium and is being demultiplexed into appropriate independent message signals by the corresponding commutator at the receiving end. Composite data rate must be at least equal to the sum of the individual data rates. The time division multiplexing operation is shown in fig. 1.3 [13].

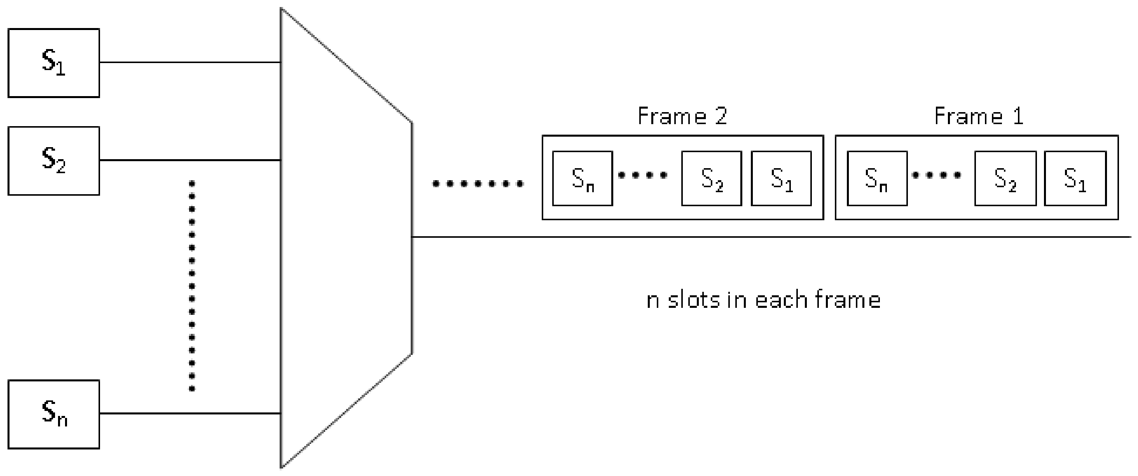


Fig. 1.3 Time Division Multiplexing Operation

As shown in the fig. 1.3 the composite signal has some dead space between the successive sampled pulses, which is essential to prevent interchannel cross talks. Along with the sampled pulses, one synchronizing pulse is sent in each cycle. Sampled pulses and synchronizing pulse along with the control information form a frame. Each of these frames contain a cycle of time slots and in each frame, one or more slots are dedicated to each data source. The maximum bandwidth (data rate) of a TDM system should be at least equal to the same data rate of the sources. The precise synchronization is needed in TDM. All senders have precise clocks and scheduling or there has to a way to distribute synchronization signals. Receivers have to listen to the right frequency at exactly the right points in time [14].
Time division multiplexing (TDM) can be divided into two schemes: Synchronous and Statistical [15].

1.3.1 Synchronous Time Division Multiplexing

In synchronous TDM the entire capacity of the shared transmission medium is allocated to each source for a short duration which is sufficient for the source to transmit a burst of information of fixed length. This is called synchronous because each time slot is preassigned to a fixed source. The time slots are transmitted irrespective of whether the sources have any data to send or not, so the channel capacity is wasted [16, 17]. Sources of different data rates can be handled by assigning fewer slots per cycle to the slower input devices than the faster devices. For example, a current European TDM transmission standard employs a 2.048 Mbps digital carrier divided into frames of length 125 μsec, as shown in fig. 1.4. Each frame is divided into 32 timeslots each of length 8 bits. In every frame, timeslot 0 is reserved for synchronization and maintenance purposes, timeslot 16 is allocated to signalling and all other timeslots allocated to traffic sources. When allocated a channel, the source is given the timeslot number and it fills the appropriate timeslot in every frame with 8 bits of data. Each channel thus carries 64 Kbps of traffic.

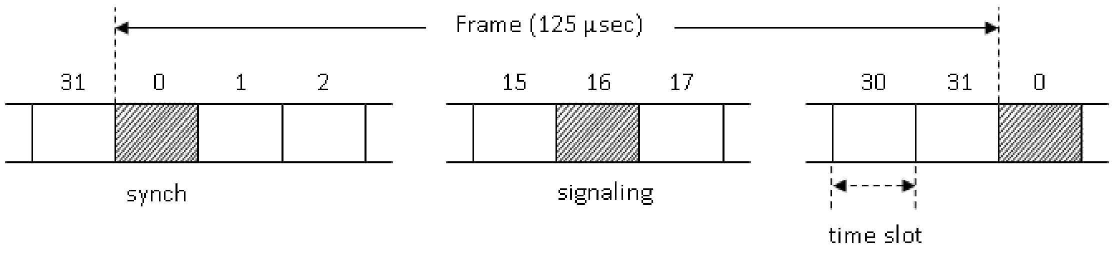


Fig. 1.4 An Example of Synchronous TDM

In synchronous TDM, the data flow of each input is divided into units, where each input occupies one input time slot. A unit can be 1 bit, one character, or one block of data. Each input unit becomes one output unit and occupies one output time slot. The duration of an output time slot is n times shorter than the duration of an input time slot. If an input time

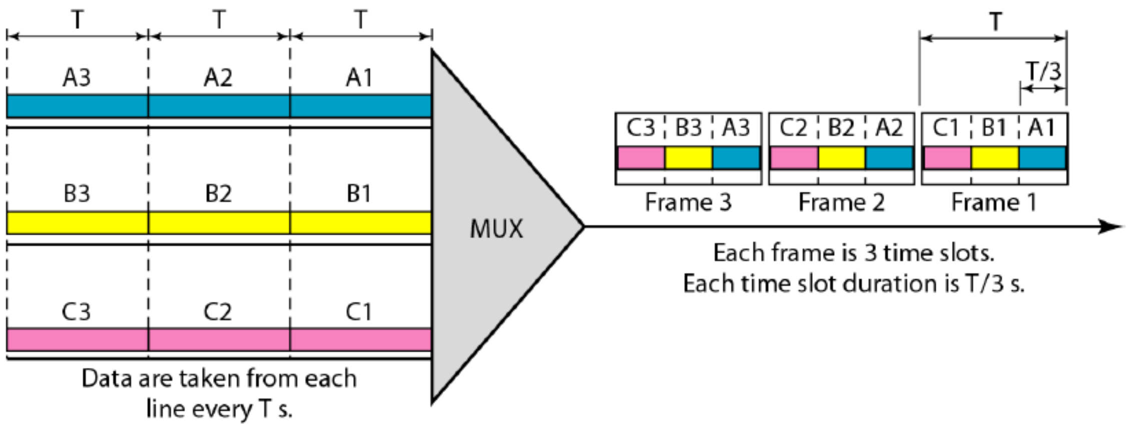


Fig. 1.5 Synchronous TDM with 3 Connections

slot is T s, the output time slot is T/n s, where n is the number of inputs. The fig. 1.5 shows an example of synchronous TDM where n is 3. In this technique, data units from each input is collected into a frame. If there are n connections, a frame is divided into n time slots and one slot is allocated for each input. If the duration of the input unit is T, the duration of each slot is T/n and the duration of each frame is T [18].

Interleaving

TDM can be seems as two fast-rotating switches, one on the multiplexing side and the other on the demultiplexing side which are synchronized with each other rotate at the same speed, but in opposite directions. On the multiplexing side, as the switch opens in front of a connection, that connection has the opportunity to send a unit onto the path. This process is called interleaving. On the demultiplexing side, as the switch opens in front of a connection, that connection has the opportunity to receive a unit from the path. The fig. 1.6 shows the interleaving process.

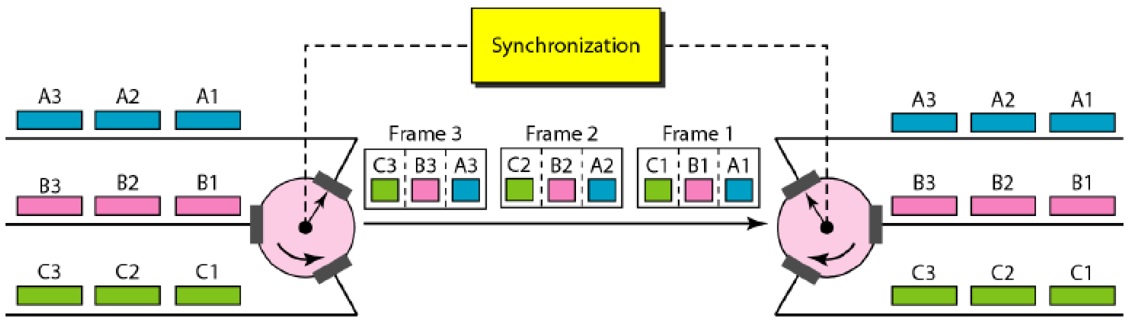


Fig. 1.6 Interleaving in Synchronous TDM

Data Rate Management

One Problem in TDM is how to handle the disparity of the data rate. If the data rate of all inputs are not same, three strategies or a combination of them can be used. The three strategies are multilevel multiplexing, pulse stuffing and multiple slots allocation.

Multilevel Multiplexing: This technique is used when data rate of one input is factor of data rate of other inputs. For example, in fig. 1.7 we have two inputs of data rate 30 Kbps and two inputs of data rate 60 Kbps. Inputs of data rate 30 Kbps can be multiplexed to provide the data rate of 60 Kbps that is equal to the other two inputs. A second level of multiplexing can be used to provide a data rate of 180 Kbps.

Multiple Slots Allocation: This technique is used when the data rate of one input line is multiple of data rate of other input lines. Allocate the more than one time slot to that input

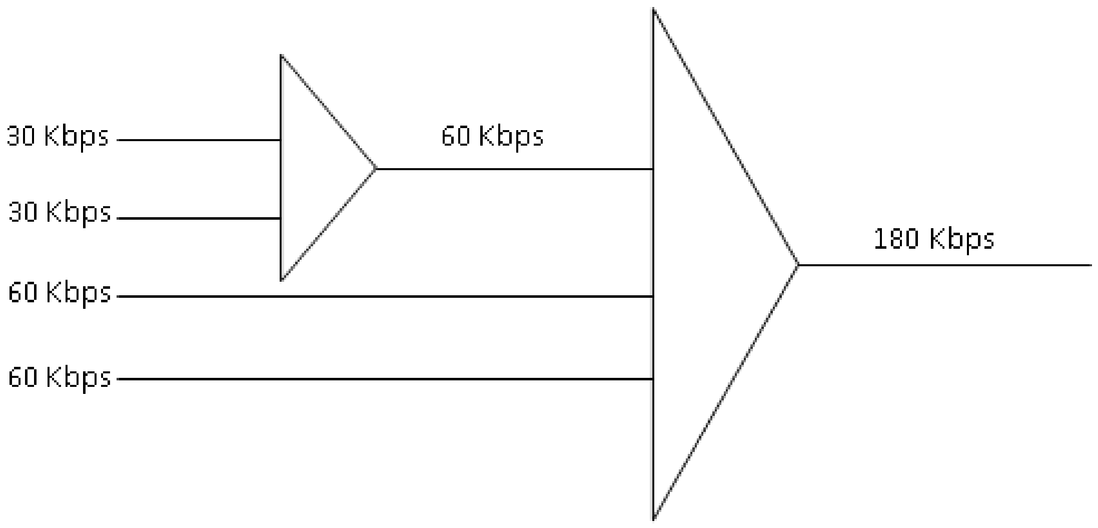


Fig. 1.7 Multilevel Multiplexing

in each frame. For example, in fig. 1.8 one input line has a data rate of 60 Kbps and other input data lines have data rate of 30 Kbps. We allocate the two time slot in each frame to the input line with data rate 60 Kbps. A serial to parallel converter is used to make the two inputs out of one.

Pulse Stuffing: This technique is used when the data rate of inputs are not integer multiple

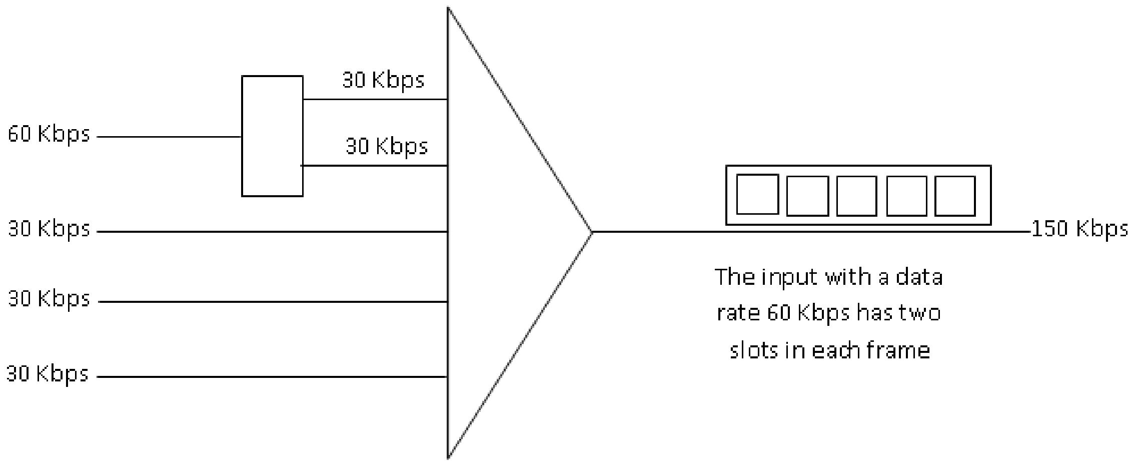

30 Kbps
60 Kbps
30 Kbps
30 Kbps
30 Kbps
30 Kbps
150 Kbps
The input with a data rate 60 Kbps has two slots in each frame

Fig. 1.8 Multiple Slots Allocation

of each other. In this case, the above two techniques cannot be applied. We make the highest data rate as the dominent data rate and then add some dummy bits to the lower data rate to make their data rate is equal to the highest data rate. This technique is called pulse stuffing, bit stuffing or bit padding. For example, in fig. 1.9 the input with data rate 26 Kbps is bit stuffed to increase its data rate to 30 Kbps, now multiplexing can take place.

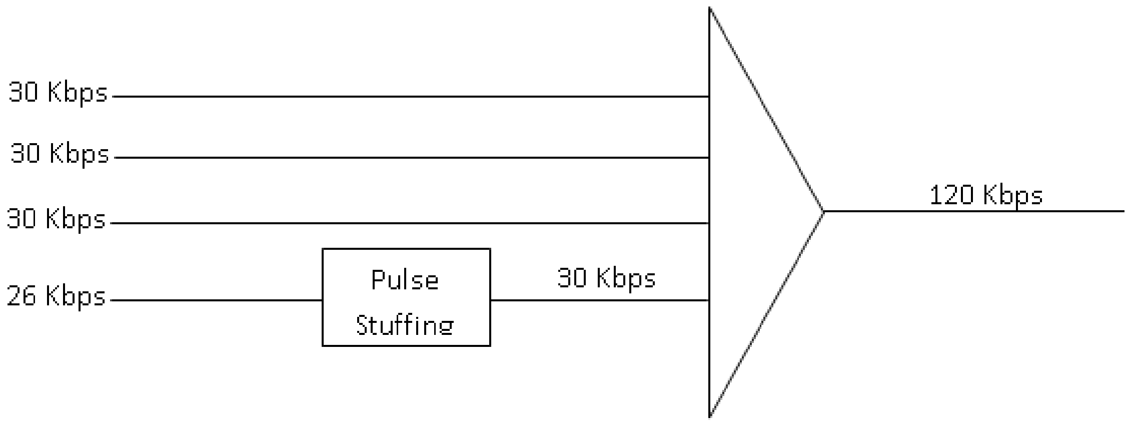


Fig. 1.9 Pulse Stuffing

Frame Synchronization

Synchronization between multiplexer and demultiplexer is the major issue in TDM. If the multiplexer and demultiplexer are not synchronized, bit belonging to one channel can be received by another channels. So, one or more bits called synchronizing bits are added at the starting of each frame. These bits also called framing bits and follow a pattern from frame to frame. It allows demultiplexer to synchronize with incoming bit stream so that it can identify the time slots accurately. In most cases, this synchronizing bit consist only one bit in each frame and alternates between 0 and 1 as shown in fig. 1.10.

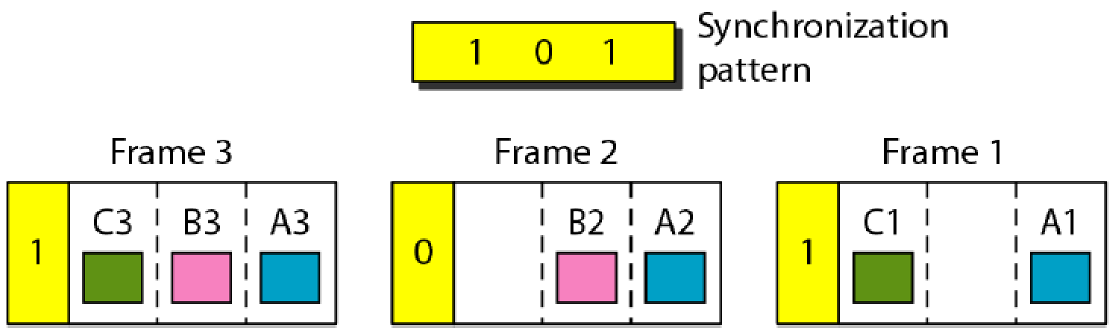


Fig. 1.10 Frame Synchronization

Empty Time Slots

If a source does not have data to send then its corresponding slot in output frame will be empty. Fig. 1.11 shows tha case when one input line has no data to send and another input line has discontinuity in data, so their corresponding time slots are empty in corresponding frames. The first output frame has three time slots filled, second frame has two time slots filled and third frame has three time slots filled, no frame is full [19, 20].

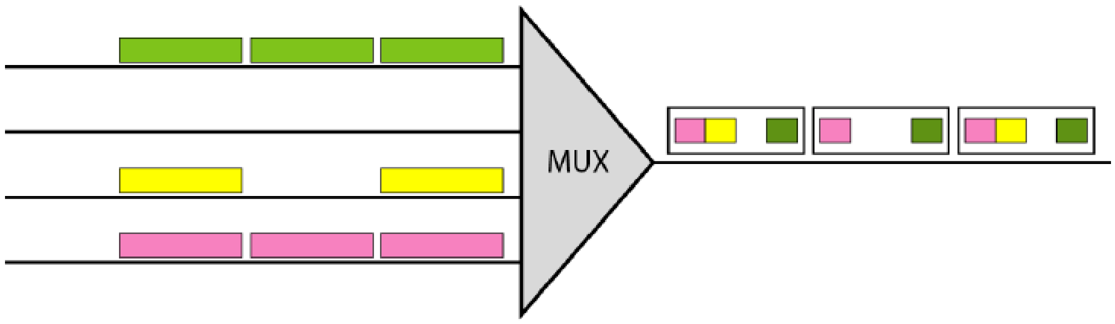


Fig. 1.11 Empty Time Slots

Statistical time division multiplexing can increase the efficiency of the system by removing these empty time slots from the frame.

1.3.2 Statistical or Asynchronous Time Division Multiplexing

One drawback of synchronous TDM is that many of the time slots in the frame are wasted. If a particular input has no data to sent at particular time instant, an empty time slot will be transmitted. The solution of this problem is statistical TDM also known as asynchronous TDM or intelligent TDM and is shown in fig. 1.12 [21]. In statistical multiplexing the channels are no longer of fixed bandwidth but each source receives as much transmission capacity as it requires instantaneously, means it dynamically allocates the time slots to each input based on demand, thus saving the channel capacity [22]. Only when an input line has data to transmit, the time slot is allocated to that source in output farme. Sources continue to transmit information in bursts but these bursts are not necessarily of equal length and sources may submit bursts of information in any order and at a rate that reflects the instantaneous bandwidth required. Sources generally queue for access to the shared transmission medium on a first come first serve basis but some sources may be allocated priority. In synchronous TDM, when a source is idle the time slot allocated to it is unused and is not available to

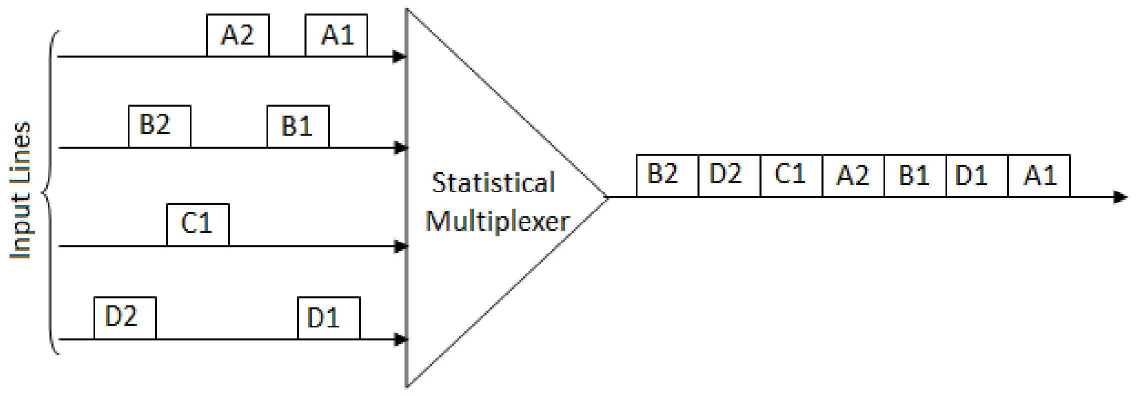


Fig. 1.12 Statistical TDM

other sources whereas in statistical TDM a source only consumes the time slot when it has information to send. In statistical multiplexing, the number of slots in each frame is less than the number of input lines. The multiplexer checks each input line in round robin fashion. It allocates a slot for an input line if the line has data to send otherwise it skips the line and checks the next line. The difference between synchronous TDM and asynchronous TDM is shown in fig. 1.13. It may be noted that many slots remain unutilised in case synchronous TDM, but the slots are fully utilized in statistical TDM, leading to smaller time for transmission and better utilization of bandwidth of the medium [3, 23, 24].

Addressing

The fig. 1.14 also shows a major difference between slots in the synchronous TDM and asynchronous TDM. In synchronous TDM the output slot is totally occupied by the data but in statistical TDM data as well as address of destination is transmitted in one time slot.
In synchronous TDM, synchronization and preassigned relationship between input and output serves as an address. For example, we know that the input from input line 1 goes to the output line 1, if the multiplexer and demultiplexer are synchronized it is guaranteed. In statistical multiplexer there is no fixed relationship between input and output because there is no preassigned and fixed time slot. So, it is needed to include the address of the receiver in each time slot to show that where it has to be delivered. To represent the address of N output lines we need n bits as $N = log_2 n$. This leads to more overhead per slot as shown in fig. 1.15. Relative addressing can be used to reduce overhead [25, 26].

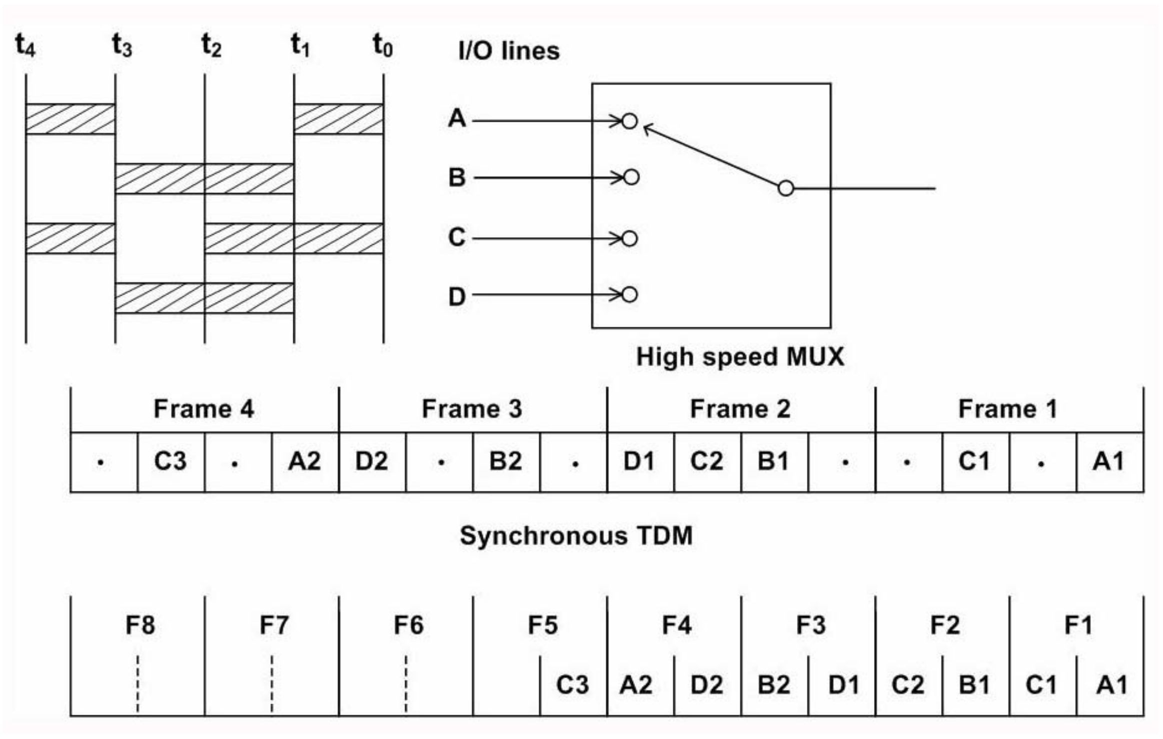


Fig. 1.13 Synchronous vs Asynchronous TDM

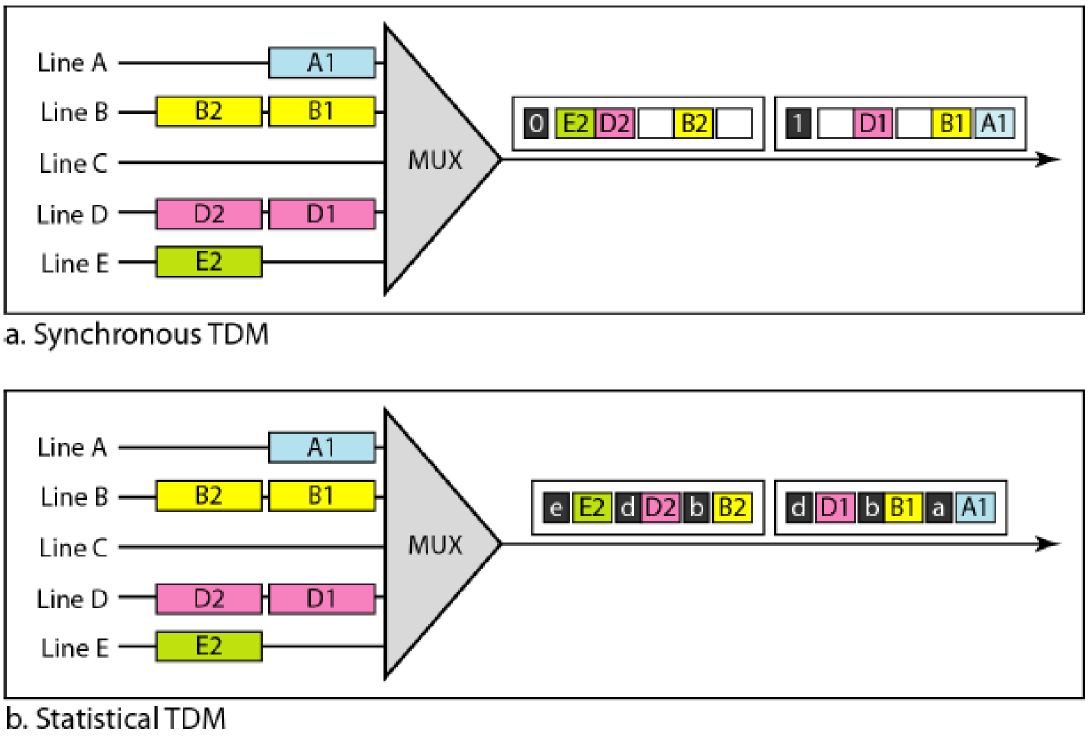


Fig. 1.14 Addressing in Statistical TDM

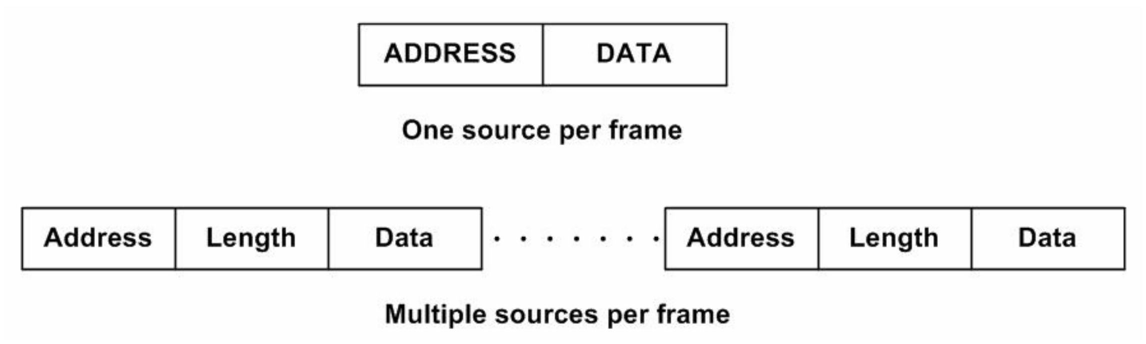


Fig. 1.15 Address Overhead in Statistical TDM

Slot Size

Since a slot carries both data and an address in statistical TDM, the ratio of the data size to address size must be reasonable to make transmission efficient. For example, it would be inefficient to send 1 bit per slot as data when the address is 3 bits. This would mean an overhead of 300 percent. In statistical TDM, a block of data is usually many bytes while the address is just a few bytes [27].

No Synchronization Bit

In statistical multiplexer, there is no need of synchronization between multiplexer and demultiplexer. So, we don't need to add synchronizing bit at the starting of each frame.

Bandwidth

In statistical TDM, the capacity of the link is normally less than the sum of the capacities of each channel. The designers of statistical TDM define the capacity of the link based on the statistics of the load for each channel. If on average only x percent of the input slots are filled, the capacity of the link reflects this. Of course, during peak times, some slots need to wait [28, 29].

1.3.3 Synchronous vs Statistical Multiplexing

In synchronous TDM the offered traffic load can never exceed the capacity of the shared transmission medium, a utilisation of 100% may be supported indefinitely, delay is deterministic and jitter is very low. However, with statistical multiplexing, for short periods the offered traffic load can exceed the capacity of the transmission medium which may results in either loss of information or delay or both. Statistical multiplexing can support only 80% maximum utilisation of the transmission medium. Delay is dependent upon the mean traffic load and the source traffic characteristics and jitter may be high. Despite these apparent disadvantages, statistical multiplexing is very fexible, supports traffic sources which vary widely in their bandwidth requirements and source traffic characteristics and handles bursty sources efficiently [30–32].

1.4 Motivation and Objective

Now a days, we are working in the frequency range from 100 Mbps to several Gbps. We also want that each user gets better quality of service (QoS) without bearing the quality of service of each other. Some applications are delay sensitive and some applications are loss sensitive, some applications generate bursty information. Todays, networks are able to carry all types of information due to development of statistical multiplexing. Statistical multiplexer also uses the buffer to store the packets, so there is a delay in the transmission of the information. Software implementation of statistical multiplexer can take several minutes to hours. It is, therefore necessary to implement statistical multiplexer so that the processing time is reduced to the order of seconds. Furthermore, these implementations should not be cost effective. So, this thesis presents a field-programmable gate array (FPGA) based implementation of statistical multiplexer. The reported implementation provides a speedup in process, thus reducing the time from minutes to several neno seconds.

1.4.1 Hardware vs Software

Software is a general term used to describe a collection of computer programs, procedures, and documentation that perform some task on a computer system. Practical computer systems divide software systems into three major classes: system software, programming software, and application software. Software is an ordered sequence of instructions for changing the state of the computer hardware in a particular sequence. It is usually written in high-level programming languages that are easier and more efficient for humans to use (closer to natural language) than machine language. High-level languages are compiled or interpreted into machine language object code. Software may also be written in an assembly language, essentially, a mnemonic representation of a machine language using a natural language alphabet.

Hardware is best described as a device that is physically connected to the computer or something that can be physically touched. A CD-ROM, monitor, printer, and video card are all examples of computer hardware. Without any hardware, computer would not exist and software would have nothing to run on. It is the physical part of a computer, including the digital circuitry, as distinguished from the computer software that executes within the hardware.

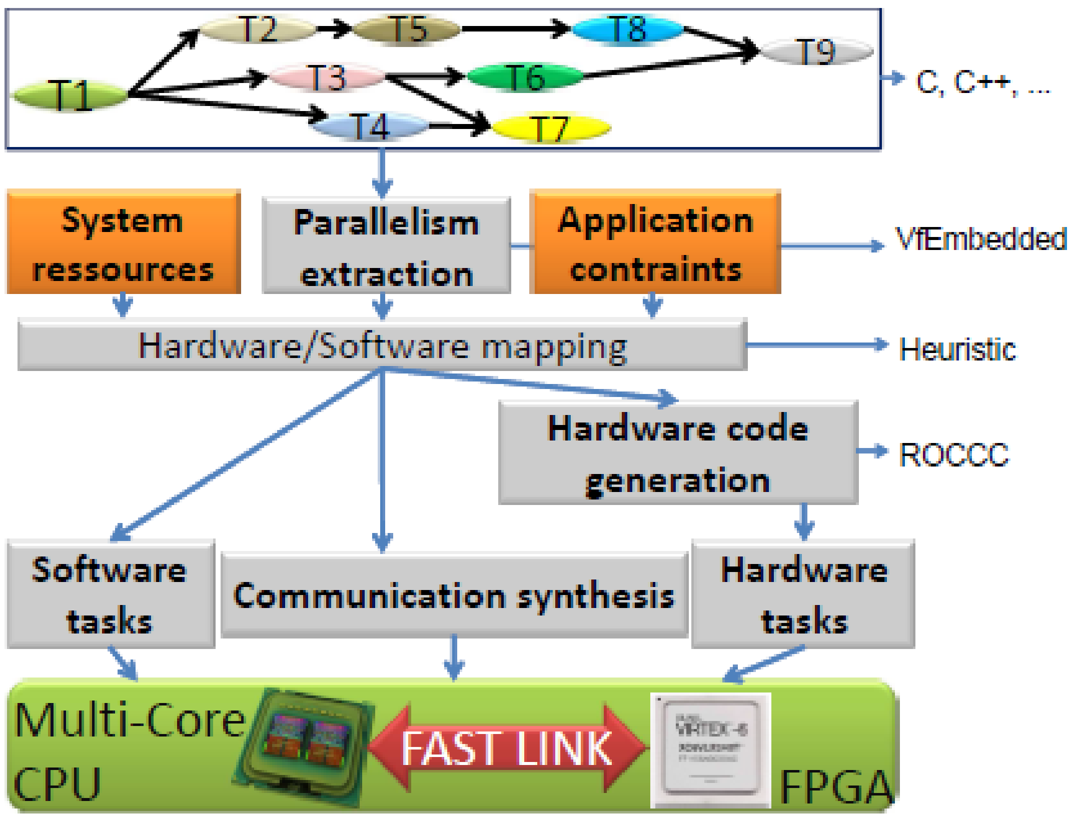


Fig. 1.16 Design Methodology for Efficient Test Mapping

Writh's law states that, "Software is getting slower more rapidly than hardware becomes faster." or, colloquially, "Software gets slower faster than hardware gets faster". It is stated by Niklaus Wirth in 1995. Computer hardware has become faster over time, and some of that development is quantified by Moore's law. So, the hradware implementation is always faster than the software implementation. Reconfigurable logic provides high performance per watt, execution time in neno seconds and high adaptability to application constraint. FPGA technology could offer better performances comparing to CPUs or GPUs up to 10x. George AFONSO, Rabie Ben Atitallah and Jean-Luc Dekeyser in [33] perform a experiment and prove that the execution time of hardware implementation is much more less than the execution time of software implementation. They also present a design methodology that covers the different development steps from software specification to the system implementation as shown in fig. 1.16 [33].

1.4.2 Why FPGA?

A field-programmable gate array (FPGA) is an integrated circuit which is designed to be configured by the customer or designer after manufacturing. FPGA can be used to implement any logic function. This project is targeted for FPGA for the following reasons:

1. FPGA represent a new direction for DSP, communication and real time systems and there is much original work to be done in terms of optimized algorithms for this type of systems.

2. They fill a need in the design space of digital systems, complementary to the role of micro-processors. Since, micro-processors can be used in variety of, but they rely on the software environment to implement functions. Hence, micro-processors are generally slower & more power hungry than the custom chips.

3. There is no wait from completing the design to obtain the working chip, that is design can be programmed into FPGA & tested immediately.

4. They are excellent prototyping vehicles. When used in final design, the jump from prototype to product is much smaller and easier to negotiate.

5. In several different designs, they are used to reduce the inventory cost.

They were often used in prototype because they could be programmed and inserted into board, in a few seconds/minutes. But they did not make it into the final product. And now a days they are used in all sort of digital systems: as video accelerator in home, personal video recorder (PVR's). They provides programmable logic elements & programmable interconnects to build the multilevel logic function [34].

1.5 Organization of thesis

The thesis has been organized as follow: In chapter 2, a brief introduction about the algorithms and proposed hardware architecture is discussed. The blocks used in the hardware architecture is also studied for the purpose of understanding their working which is important for implementation. Chapter 3 describes the hardware & software tools used in the implementation. Preference of verilog in Xilinx ISE instead of VHDL is also studied. Features of FPGA implementation are also discussed. Finally, in chapter 4, all modules are implemented in Xilinx ISE. Their schematic, RTL behavior and simulation is studied.

Chapter 2

Algorithms & Proposed Hardware Architecture

Multimedia applications ranging from telephone to video have become widespread, and networks today are filled with streaming flows having various bandwidths. Quality of service (QoS) must be controlled under streaming traffic conditions. Bandwidth management in high speed networks is also considered as a key technology in providing services to different traffics such as video, voice and data [35, 36]. These traffic types have very different needs as well as quality of service (QoS). In order to guarantee QoS to every transport connection established across the network, we need to have call admission control (CAC) which decides if a new call can be established with QoS required by the call without affecting the QoS of existing connections.

We have a lot of algorithms for call admission control [37–42]. In this theis we are describing two algorithms for call admission control: one is based on the effective bandwidth, and second is based on the tail probability of queue length. These two algorithms are the basic algorithms for call admission control.

2.1 Algorithm Based on Tail Probability of Queue Length

The authors in [43, 44] has proposed an algorithm for call admission control based on the tail distribution of the queue length in the corresponding infinite-buffer queue. In [43] it is assumed that the time is divided into slots and the length of one slot is equal to a unit time and cells arrives in batches. Two types of queuing models are defined based on the timing of arrivals: the early arrival model and the late arrival model. In the early arrival model, an arrival of a batch in the nth slot occurs immediately after the beginning of the nth slot. On the other hand, in the late arrival model, an arrival of a batch in the nth slot occurs immediately before the end of the nth slot [45, 46]. In this paper only early arrival model is considered.

The probability that a random variable deviates by a given amount from its expectation is referred to as a tail probability. Let x_k denote a $1\ X\ M$ vector whose $j^t h$ element represents the joint stationary probability of k cells in the system and the underlying Markov chain being in state j in the corresponding discrete-time queue with infinite buffer. Let ρ denote the traffic intensity, N is the number of cell in the system and e is an $M\ X\ 1$ vector with all elements equal to one. The tail probability T_k $(k \geq 0)$ is defined as:

$$T_k = \sum_{m=k+1}^{\infty} x_m e \tag{2.1}$$

Let P_{loss} denote the cell loss probability in the early arrival model and is defined as:

$$P_{loss} \approx \frac{(1-\rho)T_N}{\rho(1-T_N)} \tag{2.2}$$

Eq. 2.2 gives the approximate value of the cell loss probability. Also, it is difficult to calculate the tail probability of the queue with infinite buffer length. It is also not possible practically to implement the queue of infinite length.

2.2 Algorithm Based on Effective Bandwidth

Suppose we have a system as shown in fig. 2.1. There is M no. of sources, which have data that is statistical in nature, means they don't transmit data with constant periodic bit rate, bit

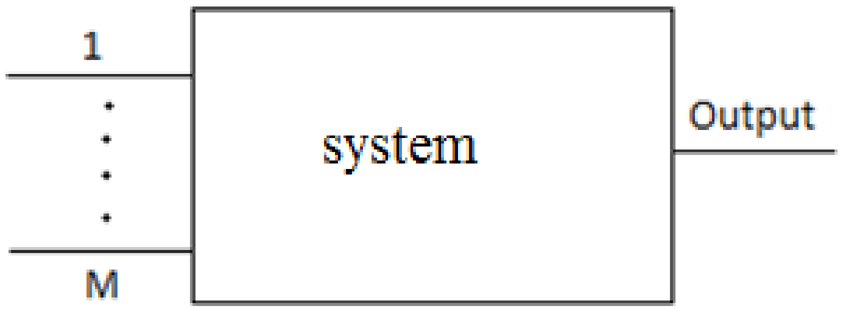


Fig. 2.1 An Example of a System

rate is variable in nature. Let the bit rate of source one is R_1, bit rate of source two is R_2 and so on, bit rate of source M is R_M. L is the maximum bit rate of output and Y is the combined bit rate of inputs, means

$$Y = R_1 + R_2 + ... + R_M \tag{2.3}$$

Suppose new user wants to enter in the system. So, the network problem is to determine whether new user can admit or not. In other way the network problem is to determine that how many numbers of users can admit in the system so that the packet loss probability is less than some threshold.

We consider that the PDF of bit rate of all users is known and PDF of i^{th} source, data rate is represented by $f_{R_i}(r_i)$. Because, the bit rate of users is a random variable and independent to each other, so from eq. 2.3

$$f_Y(y) = f_{R_1}(r_1) * f_{R_2}(r_2) * .... * f_{R_M}(r_M) \tag{2.4}$$

$$Probability[Y > L] => packetlossprobability$$

if,

$$packetlossprobability < \delta_{threshold}$$

Only then new user can enter in the system otherwise not. This technique has been defined in detail in [47].

2.3 Proposed VLSI Architecture

The algorithm describe in section 2.1 cannot be implemented on the hardware. So, we proposed a VLSI hardware architecture Based on the principle described in section 2.2 as shown in fig. 2.2 to control the new user admission in statistical multiplexing and to provide better QoS in terms of packet loss probability. This reconfigurable hardware architecture has been implemented on FPGA. The FPGA hardware implementation is faster than the software implementation because it takes less time in the range of ns. The working of this architecture is based on the flow chart as shown in fig. 2.3

If new user wants to enter in the system then it will send its packet to the system. Packet has the source address, by which system will know that it is not the packet from existing users. So, the convolution unit in the system computes the convolutio of all PDF of source data rate, which is already stored in the memory. Its output is fed to the probability computation unit. Probability computation unit compute the probability of combining data rate Y

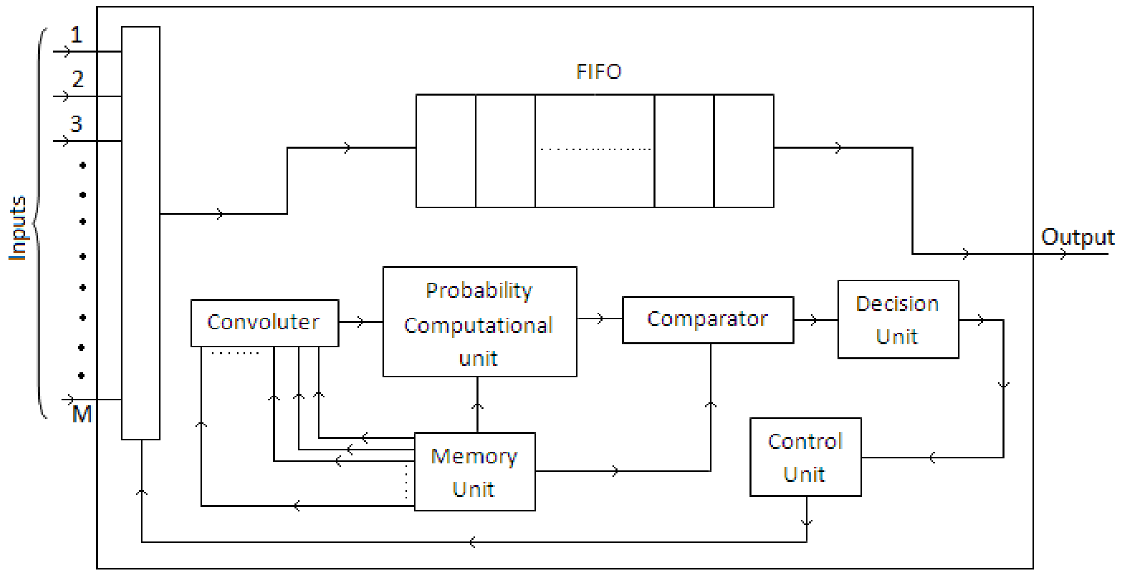


Fig. 2.2 Proposed Hardware Architecture

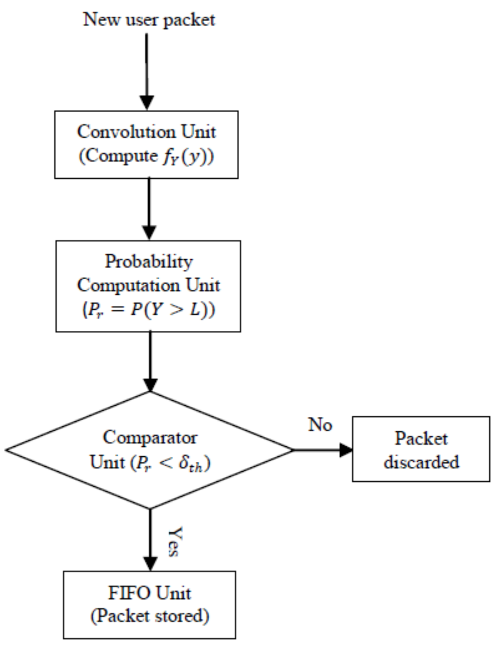

New user packet
Convolution Unit (Compute $f_Y(y)$)
Probability Computation Unit ($P_r = P(Y > L)$)
Comparator Unit ($P_r < \delta_{th}$)
No
Packet discarded
Yes
FIFO Unit (Packet stored)

Fig. 2.3 Flow Chart

is greater than the maximum data rate of the output, i.e.

$$P(Y > L) = \int_{L}^{\infty} f_Y(y)dy \tag{2.5}$$

The comparator unit compares the output of probability computation unit with a threshold value which is again stored in the memory. Based on the output of comparator, decision unit decide whether the packet of new user will stored in the buffer or it will be discarded. If the output of comparator is 1, it means that the combined bit rate of inputs is less than the maximum bit rate of the output so the packet of new user can be stored in the buffer means new user can enter in the system. Decision unit gives the input to control unit to store the packet in the buffer. If the output of comparator unit is 0, then decision unit gives the input to control unit to discard the packet.

2.4 Different Units in Hardware Architecture

As this project is focused on hardware implementation of statistical multiplexer for controlling the user admission. The proposed hardware architecture is shown in fig. 2.2. This architecture for statistical multiplexer has several basic bolcks:

1. Convolution Unit
2. Probability Computational Unit
3. Comparator
4. Decision Unit
5. FIFO Buffer
6. Memory Unit
7. Control Unit

2.5 Convolution Unit

Convolution is a mathematical operation on two functions, producing a third function that is typically viewed as a modified version of one of the original function, giving the area overlap between the two functions. It is also defined as an array operation where each output data element is a weighted sum of a collection of neighbouring input elements. The weight use in the weighted sum calculation are defined by an input array, commanly referred to as convolution kernel. For the discrete time signal it is defined as convolution sum and for the continuous time signal it is defined as convolution integral [50]. If we have two discrete time signal $x[n]$ and $h[n]$ and if the output is denoted as $y[n]$ then convolution sum is given as:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{2.6}$$

If two continuous time signal is defined as $x(t)$ and $h(t)$ and output is defined as $y(t)$, then convolution integral is defined as:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau) \tag{2.7}$$

The properties of convolution is given as below:

2.5.1 Properties of Convolution

- Commutative Property:
$$x[n] * h[n] = h[n] * x[n] \tag{2.8}$$
- Associative Property:
$$x[n] * (y[n] * h[n]) = (x[n] * y[n]) * h[n] \tag{2.9}$$
- Distibutive Property:
$$x[n] * (y[n] + h[n]) = x[n] * y[n] + x[n] * h[n] \tag{2.10}$$
- Constant Multiplier:
$$(ax[n]) * h[n] = a(x[n] * h[n]) \tag{2.11}$$
- Sum of the sample values in the resultant of convolution is equal to the product of the sum of the sample values of individual signal being convolved.

The same properties also valid for convolution integral [51].

2.6 Probability Computational Unit

If an experiment is repeated n times under the same conditions, and if n_A is the number of occurences of event A, then the probability of A, i.e. $P(A)$ is defined as [52]:

$$P(A) = \lim_{n \to \infty} \frac{n_A}{n} \tag{2.12}$$

This is also called the relative frequency of event A. It has the following properties for an event A:

1. $0 \leq P(A) \leq 1$, where $P(A) = 0$ if A occurs in none of the n repeated trials and $P(A) = 1$ if A occurs in all of the n repeated trials.

2. Probability of a certain event is equal to 1

$$P(S) = 1, \qquad \text{where } S \text{ is the certain event} \tag{2.13}$$

3. If A and B are the mutually exclusive events, then

$$P(A+B) = P(A) + P(B) \tag{2.14}$$

2.6.1 Elementary Properties of Probability

The useful properties of probability are given as [52]:

1. $P(\overline{A}) = 1 - P(A)$

2. $P(\phi) = 0$, where ϕ is a null event.

3. $P(A) \leq P(B) \qquad \text{if } A \subset B$

4. $P(A) \leq 1$

5. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

2.7 Comparator Unit

Comparator basically compares the two input and gives the output that the one input is either greater or smaller or equal to the second input. Mathematically, if $A \geq B$, then the output of comparator is one and if $A < B$, then comparator output is zero. It is not the hard and fast rule that if $A \geq B$, then only its output is one, it may be that if $A \leq B$, then also its output is one otherwise zero, it depends on the requirement of the system. The basic concept behind the comparator is that it perform the operation $A - B$. If the result of this operation is either zero or positive then it gives the output one and if the result of operation is negative then it gives the output zero.

2.8 Decision Unit

Decision unit simple takes the decision that which operation has to perform. In this there is one or more select lines based on the number of operations. We assign a unique operation for a particular value of the select lines. Basically, it works as a multiplexer, in which we have two or more inputs, one or more select lines and one output. Suppose, we have four operations *A, B, C, & D* to be performed. If the select line has value 1 then perform operation *A* , if select line has a value 2 then perform operation *B*, if select line is equal to 3 then perform operation *C*, & if the select line is equal to 4 then perform operation *D*.

2.9 FIFO Buffer

FIFO stands for First In First Out. It is an approach to handle the program from queue or stack, so that the oldest request is handled first. It is a memory in which the data word that is written in first also comes out first when the memory is read. The real life examples of FIFO are printer, a queue of people at ticket window, vehicles on toll-tax bridge, phone answering system, luggage checking machine and patients waiting outside the doctor's clinic. The data flow in a FIFO is illustrated in fig. 2.4 [53].

There are three kinds of FIFO:

1. Shift register
2. Exclusive read/write FIFO
3. Concurrent read/write FIFO

2.9.1 Shift register

In this type of FIFO the number of stored data words are fixed. So, there is a proper synchronism between the read and write operation, means a data word must be read at the same

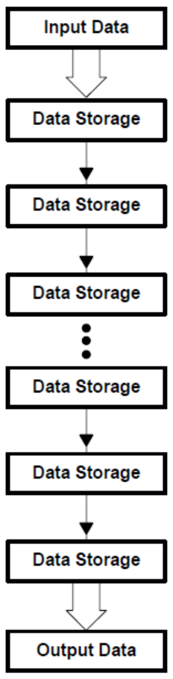


Fig. 2.4 First-in First-out Data Flow

time when a data word is written in FIFO.

2.9.2 Exclusive Read/Write FIFO

In this type of FIFO the number of stored data word are variable in nature. But, there is also a proper synchronism between the read and write operation, means the read and write operation are not independent to each other. There are timing relationships between the write clock and the read clock. For the use of such type of FIFO between two systems which are not synchronized with each other, an external circuit is required for synchronization. But, this external synchronization circuit reduces the data rate.

2.9.3 Concurrent Read/Write FIFO

In this type of FIFO also the number of stored data word are variable but the read and write operation are asynchronised, means there is no dependence between the read and write operation. So, the two systems with different frequencies can be connected to this type of FIFO. Depending on the write control and read control signals, concurrent read/write FIFO divided into two groups:

- Synchronous FIFO

- Asynchronous FIFO

The block diagram in fig. 2.5 shows the control lines of an asynchronous FIFO, and fig. 2.6 shows the typical timing on these lines in a read and write operation. The control

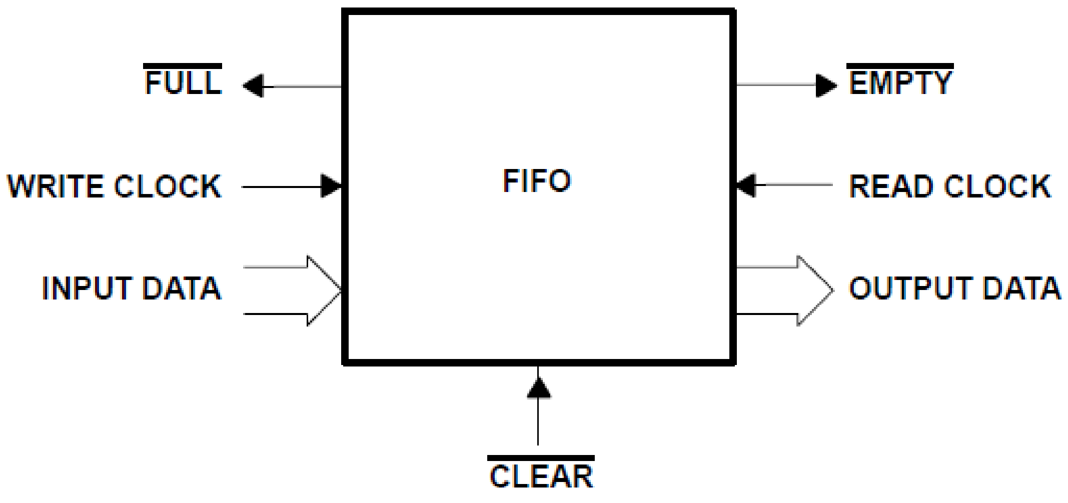


Fig. 2.5 Block diagram of a FIFO

lines WRITE CLOCK and $\overline{FULL}$ are used to write data. Before writing the data in a FIFO, it is neccessary to check whether there is space available in FIFO or not by checking the $\overline{FULL}$ status. If it is low it means that there is no space available in the FIFO. If it is high, means the FIFO is not full, so we can write the data in FIFO by the clock edge of the WRITE CLOCK input. Similarly, the control lines READ CLOCK and $\overline{EMPTY}$ are used to read data. Before, reading the data from the FIFO, we have to check whether the data is stored in the FIFO or not, means whether the FIFO is empty or not. This is done by quering the $\overline{EMPTY}$ status. If the FIFO is not empty, the data can be read by the clock egde of the READ CLOCK input. The timing diagram in fig. 2.6 shows that first we have to reset the FIFO. Then, three data words D1 through D3 are stored in the FIFO by the clock edge of the WRITE CLOCK. Once the first data i.e. D1 is written in the FIFO the $\overline{EMPTY}$ signal changes from low level to high level. Another two data wirds are written into the FIFO after the first read cycle. The reading out of first data by the clock edge of READ CLOCK does not change the status signals. After writing the two data words, FIFO is full which is indicated by the $\overline{FULL}$ signal. Finally, the four data words D2 through D5 remaining in the FIFO are read out. Thus, the FIFO is empty again, so the $\overline{EMPTY}$ status line changes again from high level to low level [54].

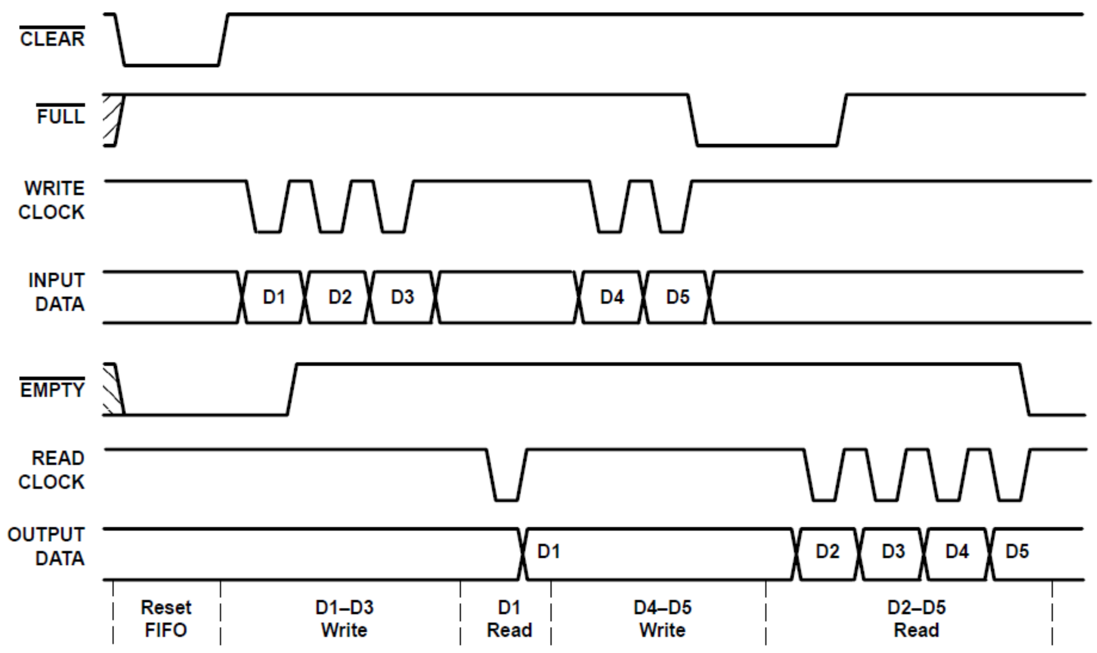


Fig. 2.6 Timing Diagram for FIFO of Length 4

2.10 Control Unit

To control all the sub-modules we need a control unit. Basically, it changes the status signal of a unit based on the status signal of other units. Like, if we have an adder, a subtractor, a multiplier and a division unit. Decision unit takes the decision that the subtractor operation has to be performed. So, the based on the output of the decision unti it applies the input to the subtractor unit. In this project, it changes the status signal of FIFO, gives the command to the convolution unit to perform the data fetch operation from the memory and many more.

Chapter 3

Hardware Tools and Implementation Platform

3.1 Hardware Description Language (HDL)

In electronics, a hardware description language or HDL is any language from a class of computer languages, specification languages, or modeling languages for formal description and design of electronic circuits, and most-commonly, digital logic. It can describe the circuit's operation, its design and organization, and tests to verify its operation at any level. The automatic translation of design description into a set of logic equation is performed by HDL. It is used to describe the architecture and behavior of discrete electronic system. The main difference with the traditional programming languages is HDL's representation of extensive parallel operations whereas traditional ones represents mostly serial operations. The design flow using HDL is given in fig. 3.1 [55].

3.1.1 Benefits of HDL

- We can verify design functionality early in the design written as an HDL description.
- Design simulation at this higher level before implementation at gatelevel, allow us to test architecture and design decision.
- The language content can be stored and retrieved easily and processed by computer software in an efficient manner.
- Reduced non-recurring engineering costs.

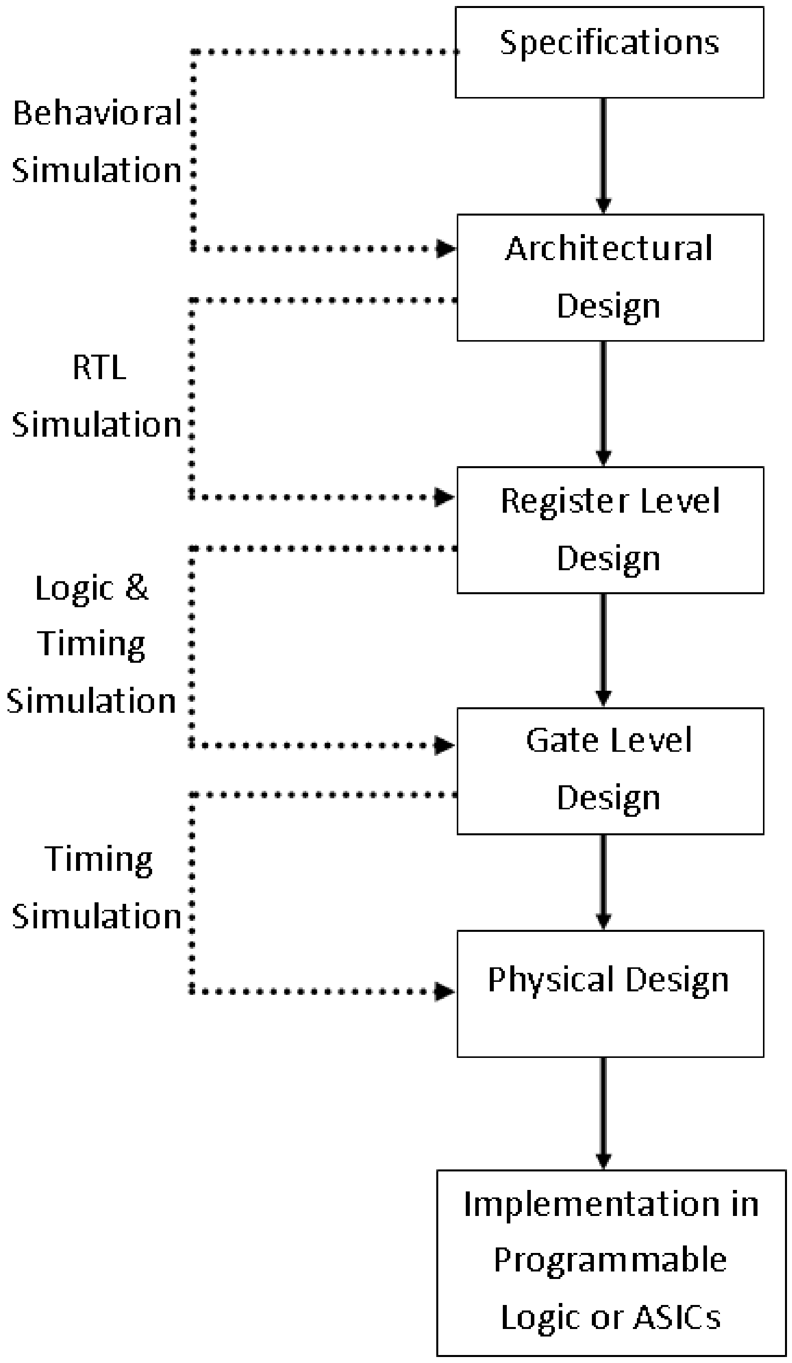


Fig. 3.1 Product Development & Design Verification Cycle using HDLs

- Design reused is enabled.
- Increase flexibility to design changes.
- Better and easier design auditing and verification.
- Base line testing of lower level design representations - example: gate level or register level design.
- Ability to manage/develop complex designs.
- Hardware/software co-design

3.1.2 Types of HDL

There are two types of hardware description language (HDL):

- VHDL – VHSIC Hardware Description Language
  VHSIC – Very High Speed Integrated Circuit
  developed from an initiative by US. Dept. of Defense.
  gate level through system level design and verification
- Verilog is other main HDL
  Primarily targeted for design of ASICs
  developed by Cadence Data systems and later transferred to a consortium called Open Verilog International (OVI).
  ASIC – Application Specific Integrated Circuit

3.1.3 Difference Between Verilog and VHDL

We can use any of the hardware description language (HDL) to design a circuit, but there is some differences between verilog and VHDL which are described in table 3.1.

In this thesis verilog is used to design the project. So, it is described in breif in next section.

3.1.4 Application of HDL Processing

There are two application of HDL processing: simulation and synthesis. The flow of both application is given in fig. 3.2.

Table 3.1 Difference Between Verilog and VHDL

S.No.	Verilog	VHDL
1	depends on C	depends on ADA
2	case-sensitive	case-insensitive
3	easy to learn	difficult to learn
4	more constructs are synthesizable	less constructs are synthesizable
5	data types are only built-in	complex, abstract, flexible, own definitions possible
6	no packages are required	required for functions, constants, types etc.
7	automatic conversion and declaration	conversion functions and all signals must be declared

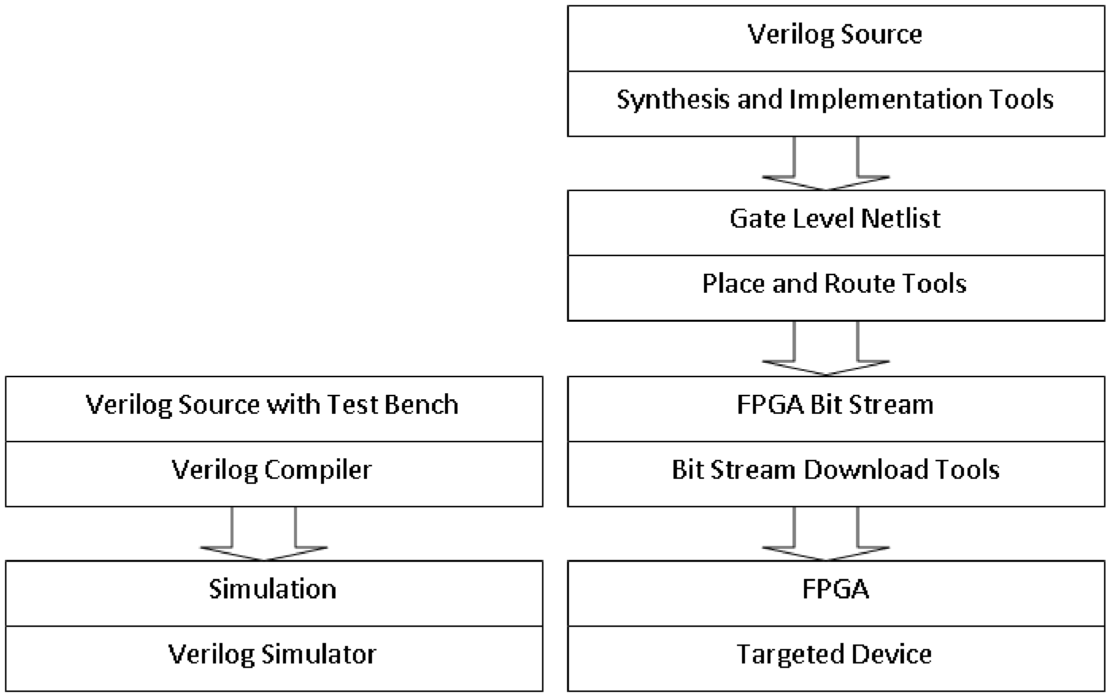


Fig. 3.2 Simulation Flow (left) and Synthesis Flow (right)

Simulation tool simulates in software the actual behavior of the hardware circuit for certain input conditions, which we describe in a testbench. Because compiling our HDL for the simulation tool is relatively fast, we primarily use simulation tools when we are testing our design.

Synthesis tool is used to turn high level HDL code to a low level gate netlist. A mapping tool then maps the netlist to the applicable resources on the targeting device. Finally, we download a bitstream, describing the way in which the targeting device should be reconfigured onto the FPGA, resulting in an actual digital circuit.

3.2 Verilog

Verilog means Verifying Logic. It was developed by Gateway Design Automation in 1985 by Phil Moorby as a simulation language. In 1989, Cadence Design Systems bought it after that it is also used as a synthesis language. It was a proprietary language, being the property of Cadence Design Systems. In the late 1980's it seemed evident that designers were going to be moving away from proprietary languages like n dot, HiLo and verilog towards the US Department of Defence standard VHDL. So, Cadence Design Systems opened verilog to public in 1990. It becomes an IEEE standard named as IEEE-1364 in 1995 and then revised in 2000 and till now it is an IEEE standard.

The Verilog Hardware Description Language (HDL) is used to describe a hardware design and a simulator that allows for testing the logic of a hardware design. Thus the designer is able to accurately simulate the operation of a wide variety of digital networks such as combinatorial logic, sequential networks and asynchronous networks. A description of a hardware design using the Verilog HDL is called a Verilog model and is the basic building block for simulation. This model can describe both the function of a design as well as the components and connections of components. Thus Verilog HDL is both a behavioral and a structural language [56].

Verilog models can be developed at several levels of abstraction. At the highest level, the designer can develop algorithmic that implement a design as an algorithm in a high level language similar to the C programming language. The next lower level would be to develop a model that describes the flow of data between registers and how a design processes that data. This type of modeling uses the languge which is called Register Transfer Language (RTL). The designer can develop more concrete models at the gate-level that describe the individual logic gates and the connections between logic gates in a design. Finally, the designer can develop very concrete models at the switch-level that describe the individual transistors and storage nodes in a device and the connections between them.

There are three main steps in the process of logic simulation in Verilog. First, the hardware is modeled using Verilog HDL at an appropriate level of abstraction. Second, the Verilog Simulator is told how to apply stimuli to the design. Third, Verilog Simulator is told how to report the results of the simulation.

Verilog Language uses a hierarchical, functional unit based design approach. The whole design consists of several smaller modules. The complexity of the modules is decided by the designer. Verilog modules define the input and output ports and logical relationship between the input and output ports. The module is always begin with *module* , ends with *endmodule* and may contain several statements. All statements as well as *module* terminate with a semi-colon while *endmodule* does not terminate with a semi-colon.

Verilog statements are concurrent in nature; except for code between begin and end blocks, there is no defined order in which they execute. In comparison, most languages like C consist of statements that are executed sequentially; the first line in main() is executed first, followed by the line after that, and so on. Synthesizable Verilog code is eventually mapped to actual hardware gates. Compiled C code, on the other hand, is mapped to some bits in storage that a CPU may or may not execute [57].

3.3 Xilinx Integrated Software Environment (ISE)

The Xilinx ISE system is an integrated design environment that consists of a set of programs to create (capture), simulate and implement digital designs in a field programmable gate array (FPGA) or complex programmable logic device (CPLD) target device. It produced for synthesis and analysis of HDL designs, which enables the developer 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. All the tools use a graphical user interface (GUI) that allows all programs to be executed from toolbars, menus or icons [58].

This tool used for the designing of both combinational and sequential circuits using a hardware description language (HDL) design specifications. In this thesis verilog is used to design the whole project, so the steps used in the design procedure are listed below:

1. Create Verilog design input file(s) using template driven editor.

2. Compile and implement the Verilog design file(s).

3. Create the test-vectors and simulate the design (functional simulation) without using a PLD (FPGA or CPLD).

4. Assign input/output pins to implement the design on a target device.

5. Download bitstream to an FPGA or CPLD device.

6. Test design on FPGA/CPLD device.

A Verilog input file in the Xilinx software environment consists of the following segments:

Header: module name, list of input and output ports.

Declarations: input and output ports, registers and wires.

Logic Descriptions: equations, state machines and logic functions.

End: endmodule

The WebPack is a free version of Xilinx ISE. This edition provides synthesis and programming for limited number of Xilinx devices. Particularly, devices with lots of I/O and huge gate matrix are disabled. The low-cost Spartan family of FPGAs, and the family of CPLDs is fully supported by this edition, that means small developers and educational institutions have no overheads from the cost of development software. License registration to use the WebPack of Xilinx ISE is required, which is free and can be renewed an unlimited number of times.

3.3.1 Design Flow of Xilinx

The following steps are involved in the realization of a digital system using Xilinx FPGAs and CPLDs, as illustrated in the fig. 3.3.

Design Entry

The first step is to enter the design which has to be analyzed. This can be done by creating “Source” files. Source files can be created in different formats such as a schematic, or a Hardware Description Language (HDL) such as VHDL, Verilog or ABEL. A project design will consist of a top-level source file and various lower-level source files. Any of these files can be either a schematic or a HDL file.

Design Synthesis

The synthesis step creates netlist files from the various source files. It is basically description of logic cells and their connections. The netlist files can serve as input to the implementation module.

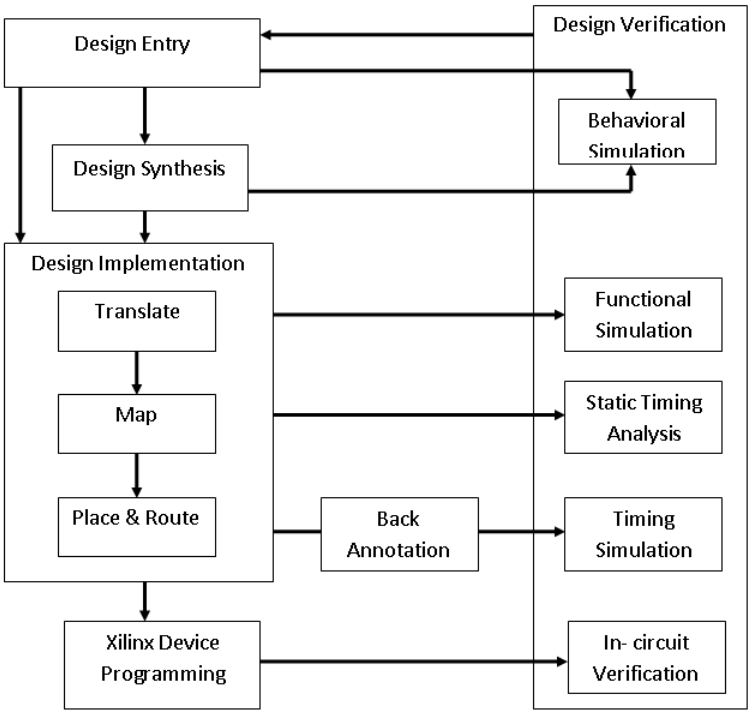


Fig. 3.3 Various Steps Involved in the Design Flow of a Digital System

Design Verification (Simulation)

This is an important step that should be done at various stages of the design. The simulator is used to verify the functionality of a design (functional simulation), the behavior and the timing (timing simulation) of the circuit. Timing simulation needs to know the actual placement and routing to find out the exact speed and timing of the circuit, so it is done after implementing the circuit in FPGA or CPLD.

Design Implementation

After generating the netlist file (synthesis step), the implementation will convert the logic design into a physical file that can be downloaded on the target devices (e.g. Virtex FPGA). This steps involves three sub-steps: Translating the netlist, Mapping and Place & Route.

Device Configuration

This refers to the actual programming of the target device by downloading the programming file to the Xilinx FPGA or CPLD.

3.3.2 Project Navigator Window

The above steps are managed through a central ISE Project Navigator window. It is divided into four panel sub-windows as shown in fig. 3.4. On the top left are the Start, Design, Files, and Libraries panels, which include display and access to the source files in the project as well as access to running processes for the currently selected source. The Start panel provides quick access to opening projects as well as frequently access reference material, documentation and tutorials. At the bottom of the Project Navigator are the Console, Errors, and Warnings panels, which display status messages, errors, and warnings. To the right is a multi-document interface (MDI) window referred to as the Workspace. The Workspace enable us to view design reports, text files, schematics, and simulation waveforms. Each window can be resized, undocked from Project Navigator, moved to a new location within the main Project Navigator window, tiled, layered, or closed.

Design Panel

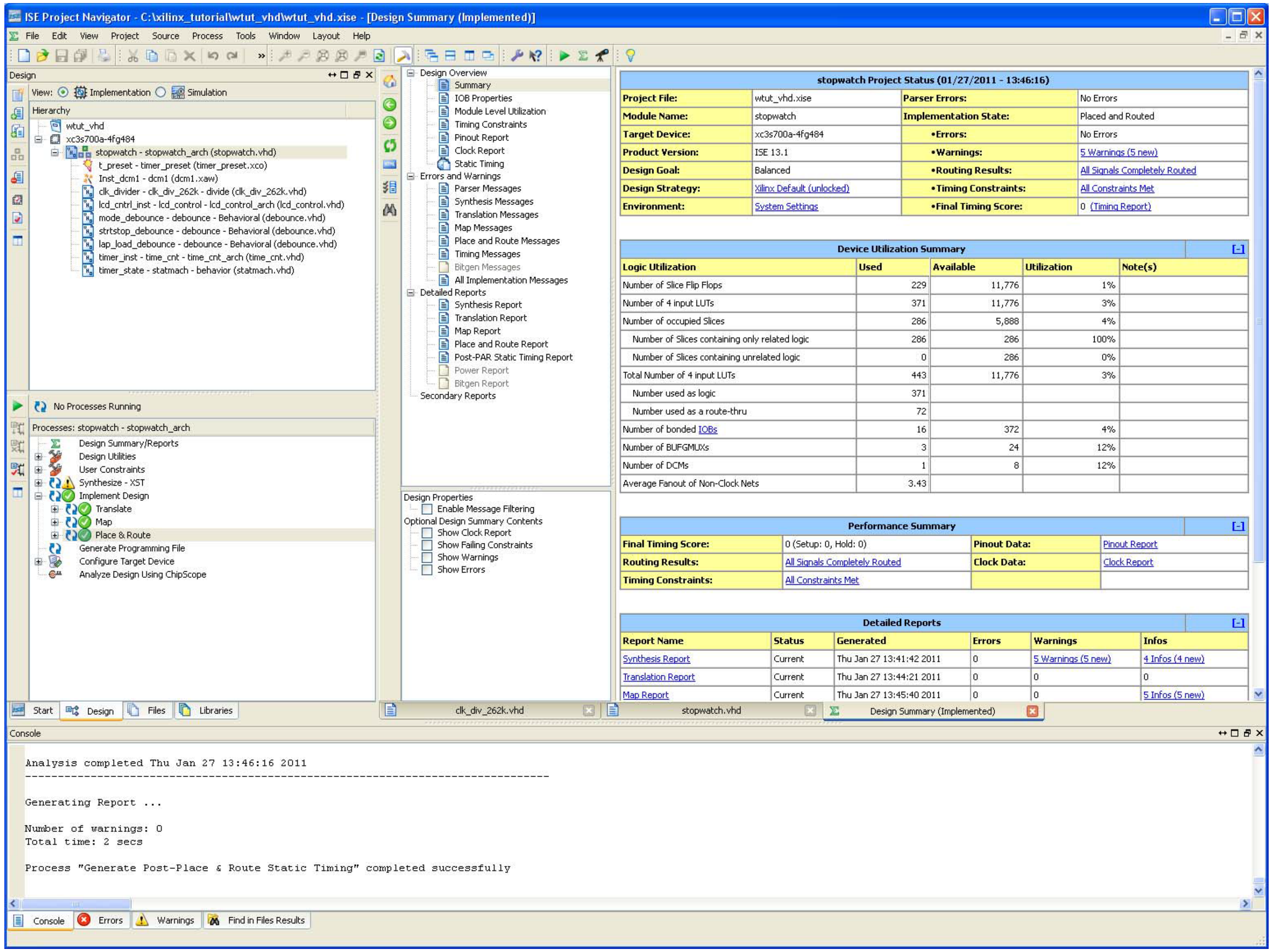
The Design panel provides access to the View, Hierarchy, and Processes panes.

View Pane radio buttons enable us to view the source modules associated with the Implementation or Simulation Design View in the Hierarchy pane. If we select Simulation, we must select a simulation phase from the drop-down list.

Hierarchy Pane displays the project name, the target device, user documents, and design source files associated with the selected Design View. The View pane at the top of the Design panel allows us to view only those source files associated with the selected Design View, such as Implementation or Simulation. Each file in the Hierarchy pane has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file, for example). If a file contains lower levels of hierarchy, the icon has a plus symbol (+) to the left of the name. We can expand the hierarchy by clicking the plus symbol (+) and a file could be opened for editing by double-clicking on the filename.

Processes Pane is context sensitive, and it changes based upon the source type selected in the Sources pane and the top-level source in the project. From the Processes pane, we can run the functions necessary to define, run, and analyze the design. The Processes pane provides access to the following functions:

- **Design Summary/Reports** provides access to design reports, messages, and summary of results data. Message filtering can also be performed.


ISE Project Navigator - C:\xilinx_tutorial\wtut_vhd\wtut_vhd.xise - [Design Summary (Implemented)]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
wtut_vhd
xc3s700a-4fg484
stopwatch - stopwatch_arch (stopwatch.vhd)
t_preset - timer_preset (timer_preset.xco)
Inst_dcm1 - dcm1 (dcm1.xaw)
clk_divider - clk_div_262k - divide (clk_div_262k.vhd)
lcd_cntrl_inst - lcd_control - lcd_control_arch (lcd_control.vhd)
mode_debounce - debounce - Behavioral (debounce.vhd)
strtstop_debounce - debounce - Behavioral (debounce.vhd)
lap_load_debounce - debounce - Behavioral (debounce.vhd)
timer_inst - time_cnt - time_cnt_arch (time_cnt.vhd)
timer_state - statmach - behavior (statmach.vhd)
No Processes Running
Processes: stopwatch - stopwatch_arch
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Design Overview
Summary
IOB Properties
Module Level Utilization
Timing Constraints
Pinout Report
Clock Report
Static Timing
Errors and Warnings
Parser Messages
Synthesis Messages
Translation Messages
Map Messages
Place and Route Messages
Timing Messages
Bitgen Messages
All Implementation Messages
Detailed Reports
Synthesis Report
Translation Report
Map Report
Place and Route Report
Post-PAR Static Timing Report
Power Report
Bitgen Report
Secondary Reports
Design Properties
Enable Message Filtering
Optional Design Summary Contents
Show Clock Report
Show Failing Constraints
Show Warnings
Show Errors
Start
Design
Files
Libraries
clk_div_262k.vhd
stopwatch.vhd
Design Summary (Implemented)
Console
Errors
Warnings
Find in Files Results

stopwatch Project Status (01/27/2011 - 13:46:16)			
Project File:	wtut_vhd.xise	**Parser Errors:**	No Errors
Module Name:	stopwatch	**Implementation State:**	Placed and Routed
Target Device:	xc3s700a-4fg484	**•Errors:**	No Errors
Product Version:	ISE 13.1	**•Warnings:**	5 Warnings (5 new)
Design Goal:	Balanced	**•Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	All Constraints Met
Environment:	System Settings	**•Final Timing Score:**	0 (Timing Report)

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	229	11,776	1%	
Number of 4 input LUTs	371	11,776	3%	
Number of occupied Slices	286	5,888	4%	
Number of Slices containing only related logic	286	286	100%	
Number of Slices containing unrelated logic	0	286	0%	
Total Number of 4 input LUTs	443	11,776	3%	
Number used as logic	371			
Number used as a route-thru	72			
Number of bonded IOBs	16	372	4%	
Number of BUFGMUXs	3	24	12%	
Number of DCMs	1	8	12%	
Average Fanout of Non-Clock Nets	3.43			

Performance Summary			
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Thu Jan 27 13:41:42 2011	0	5 Warnings (5 new)	4 Infos (4 new)
Translation Report	Current	Thu Jan 27 13:44:21 2011	0	0	0
Map Report	Current	Thu Jan 27 13:45:40 2011	0	0	5 Infos (5 new)

```
Analysis completed Thu Jan 27 13:46:16 2011
--------------------------------------------------------------------------------

Generating Report ...

Number of warnings: 0
Total time: 2 secs

Process "Generate Post-Place & Route Static Timing" completed successfully
```

Fig. 3.4 Project Navigator

- **Design Utilities** provides access to symbol generation, instantiation templates, viewing command line history, and simulation library compilation.
- **User Constraints** provides access to editing location and timing constraints.
- **Synthesis** provides access to Check Syntax, Synthesis, View RTL or Technology Schematic, and synthesis reports. Available processes vary depending on the synthesis tools used to analyze the design.
- **Implement Design** provides access to implementation tools and post implementation analysis tools.
- **Generate Programming File** provides access to bitstream generation.
- **Configure Target Device** provides access to configuration tools for creating programming files and programming the device.

The Processes pane incorporates dependency management technology. The tools keep track of which processes have been run and which processes need to be run. Graphical status indicators display the state of the flow at any given time. When a process in the flow is selected, the software automatically runs the processes necessary to get to the desired step. For example, when we run the Implement Design process, Project Navigator also runs the Synthesis process because implementation is dependent on up-to-date synthesis results.

Files Panel

The Files panel provides a flat, sortable list of all the source files in the project. Files can be sorted by any of the columns in the view. Properties for each file can be viewed and modified by right-clicking on the file and selecting **Source Properties** .

Libraries Panel

The Libraries panel enables us to manage HDL libraries and their associated HDL source files. We can create, view, and edit libraries and their associated sources.

Console Panel

The Console provides all standard output from processes run from Project Navigator. It displays errors, warnings, and information messages. Errors are signified by a red X next to the message; while warnings have a yellow exclamation mark (!).

Errors Panel

The Errors panel displays only error messages. Other console messages are filtered out.

Warnings Panel

The Warnings panel displays only warning messages. Other console messages are filtered out.

Workspace

The Workspace is where design editors, viewers, and analysis tools open. These include ISE Text Editor, Schematic Editor, Constraint Editor, Design Summary/Report Viewer, RTL and Technology Viewers, and Timing Analyzer.

Design Summary/ Report Viewer

The Design Summary provides a summary of key design data as well as access to all of the messages and detailed reports from the synthesis and implementation tools. The summary lists high-level information about the project, including overview information, a device utilization summary, performance data gathered from the Place and Route (PAR) report, constraints information, and summary information from all reports with links to the individual reports. Messaging features such as message filtering, tagging, and incremental messaging are also available from this view.

3.4 Field Programmable Gate Array (FPGA)

Field Programmable Gate Arrays (FPGA) are a relatively new class of integrated circuits. Firstly, it is introduced by the Xilinx company in 1985. Many different competing desings are developed by companies like Actel, Advanced Micro Devices, Algotronix, Altera, Atmel, AT&T, Crosspoint Solutions, Cypress, Intel, Lattice, Motorola, QuickLogic, and Texas Instruments. A generic FPGA consists of an array of logic elements together with an interconnect network which can be configured by the user at the point of application. User programming specifies both the logic function of each block and the connections between the blocks. Programming can take two forms; one-time programmable chips (analogous to fusible link PROMs), and reprogrammable chips (analogous to EEPROMs or Static RAMs).

The Programming technologies which allow these devices to be "field programmable" are EPROM and EEPROM transistors, anti-fuses, and Static Random Access Memory (SRAM) cells. The selection of a programming technology is important because it affects chip density, signal and logic propagation delays, and chip power consumption. FPGAs can be programmed by changing the characteristics of a switching element using EPROM and EEPROM programming technologies. The alternative method to design FPGA device is to use anti-fuse element. The anti-fuse resistance which is normally very high ($> 100M\Omega$) is permanently changed to a low resistance ($200 - 500\Omega$) by the application of appropriate programming voltages. The programmed anti-fuse is used to make a direct electrical connection between two metal lines.

The Static Random Access Memory (SRAM) FPGA programming technology is first introduced by Xilinx. It is also used in designs by Algotronix, Plessey, AT&T, and others. Programmable connections in these FPGAs are made using multiplexers, transmission gates, or pass transistors that are controlled by information stored in SRAM cells. Since, SRAM is volatile in nature, these FPGAs must be programmed to set the circuit configuration each time that power is applied to the chip. Table 3.2 gives the characteristics of four FPGA programming technologies which is discussed above [59].

Table 3.2 Characteristics of FPGA Programming Technologies

Programming Technology	**Re-Programmable**	**Volatile Storage**	**Series Resistance**	**Capacitance in ff**	**cell area**
Static RAM	in-circuit	yes	1KΩ	15	5X
Anti-Fuse	no	no	50-500Ω	1.2-5.0	1X
EPROM	outside circuit	no	2KΩ	10	1X
EEPROM	in-circuit	no	2KΩ	10	2X

3.4.1 Characteristics of FPGA

The characteristics of FPGA are given below:

1. They are standard parts: i.e. not designed for any particular function but are programmed by the customer for a particular purpose.

2. They implement multilevel logic: Means logic blocks inside FPGA can be connected in a network of arbitrary depth.

   Because FPGA implements multilevel logic, they generally need both programmable logic blocks and programmable interconnects. PLD's use fixed interconnects and

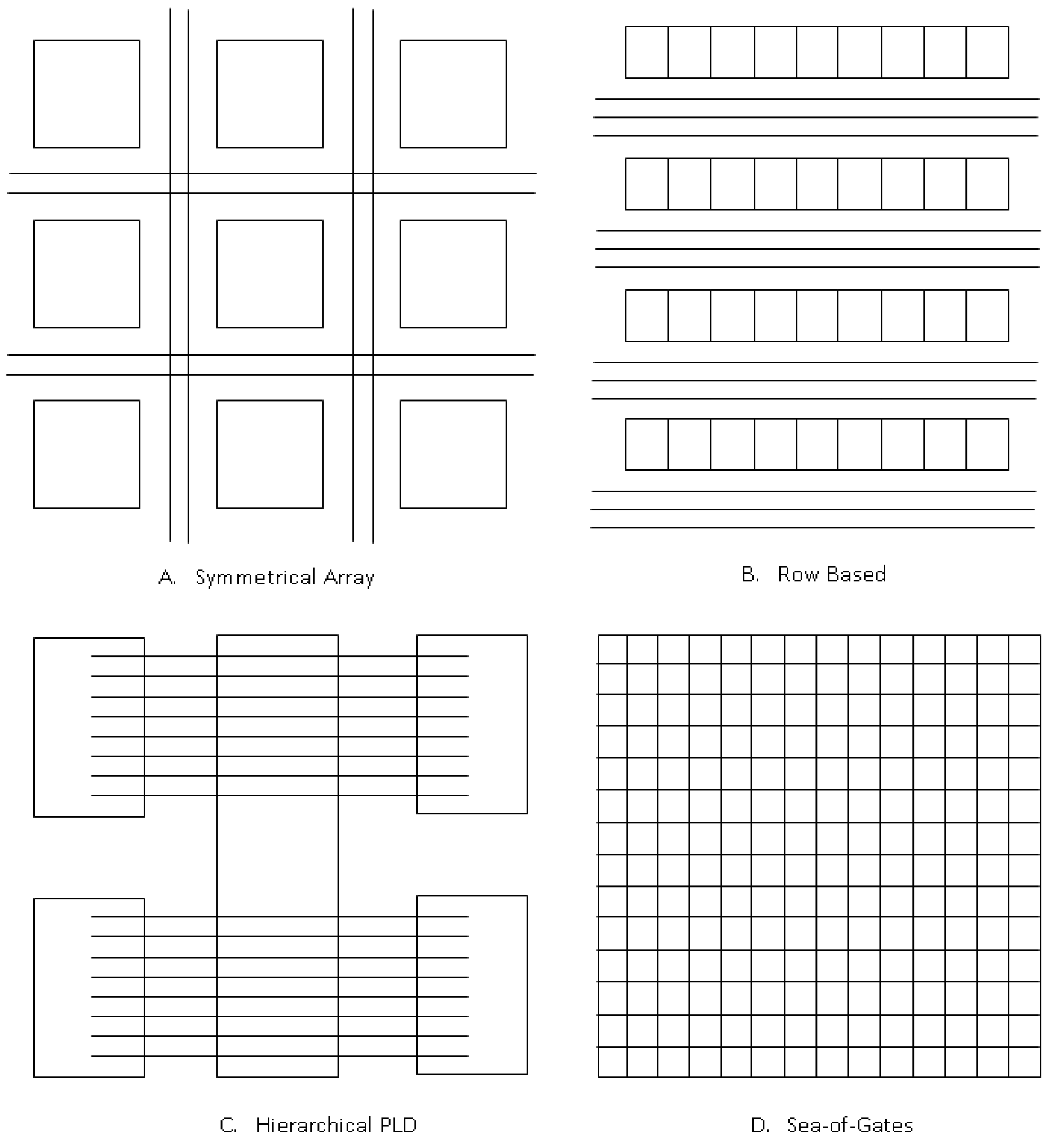


Fig. 3.5 Four FPGA Structure Classes

simply change the logic function attached to the wires. FPGA, in contrast requires programming logic blocks and connecting them together in order to implement function.

3. One of major characteristic of FPGA is that it can be programmed, which is very different than in microprocessor. Because FPGA's program are directly interwoven into logic structure of FPGA.

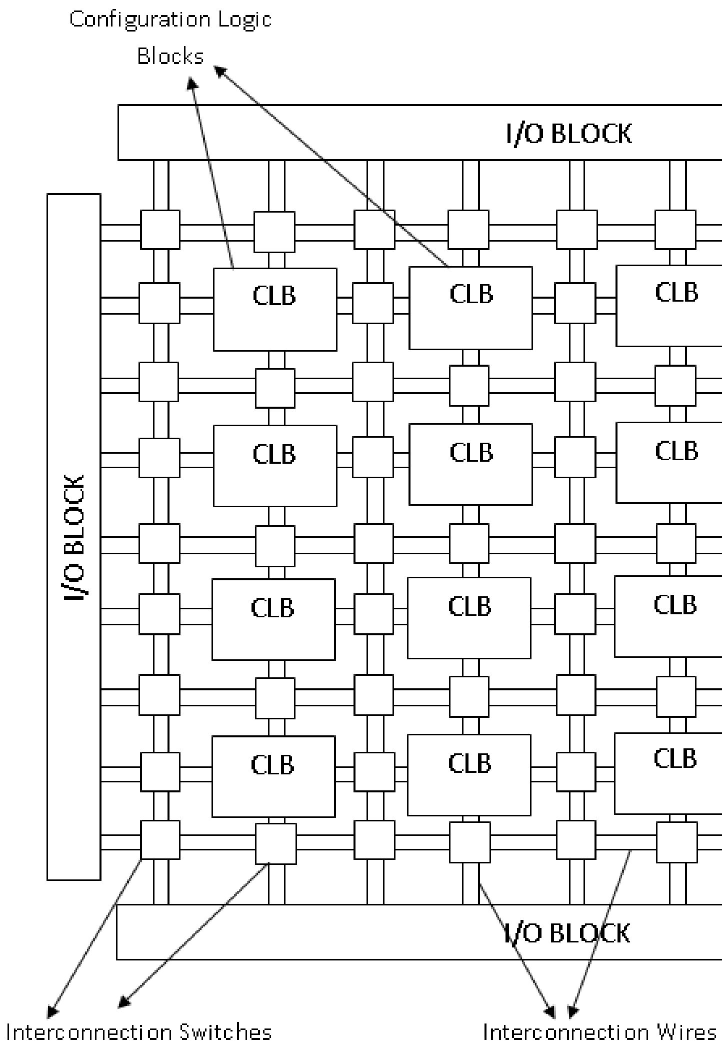


Fig. 3.6 Basic Structure of FPGA

3.4.2 FPGA Architectures

FPGAs are commonly available in many different architectures. FPGA architectures can be generally classified into one of four categories: Symmetrical Array, Row Based, Hierarchical PLD, and Sea of Gates. Fig. 3.5 illustrates this classification based on the general internal organization of the design.

The general structure of FPGA is shown in fig. 3.6. It Contain three main types of resources: configuration logic blocks (CLBs), I/O blocks for connecting to the pins of package, and interconnection wires and switches. The configuration logic blocks (CLBs) are arranged in a two-dimensional array. Each CLB contain flip-flop(s), multiplexers, and a combinatorial function block which operates as a SRAM based look-up table. The interconnecting wires are organized as horizontal and vertical routing channels between rows and columns of configuration logic blocks. The routing channels contain wires and programmable switches that allow the logic blocks to be interconnected in many ways. Programmable connections also exist between I/O blocks and the interconnecting wires [60].

3.4.3 FPGA Design Flow

Although early FPGA designs were generated largely by hand, access to today's complex programmable logic devices requires the use of an integrated Computer-Aided Design (CAD) system. Fig. 3.7 illustrates the sequence of operations needed to convert application design described in a Hardware Description Language (HDL) to a stream of bits that is eventually programmed on the FPGA. Both commercial CAD tool vendors and FPGA companies offer appropriate tools. For example, traditional Electronic Design Automation (EDA) vendors such as Cadence, Mentor Graphics, Synopsys, and ViewLogic all offer tools to support FPGA design. These tools are typically used for the front-end design entry and simulation operations and provide the necessary interfaces to vendor-specific back-end tools for chip placement and routing. Examples of vendor specific back-end tools are the Xilinx ISE system and the Altera MAX+PLUS II software.

The starting point in any logic or digital system design is a set of architectural or behavioral specifications. Traditionally, a designer uses schematic capture tools for graphical entry of a logic design which has been manually generated to meet the architectural or behavioral specifications. The upper left hand arrow in fig. 3.7 identifies some of the commercial CAD tools available for FPGA schematic capture. A behavioral design specification is created using a Hardware Description Language (HDL), and then a synthesis tool automatically compiles the gate level schematic or netlist from the behavioral description.

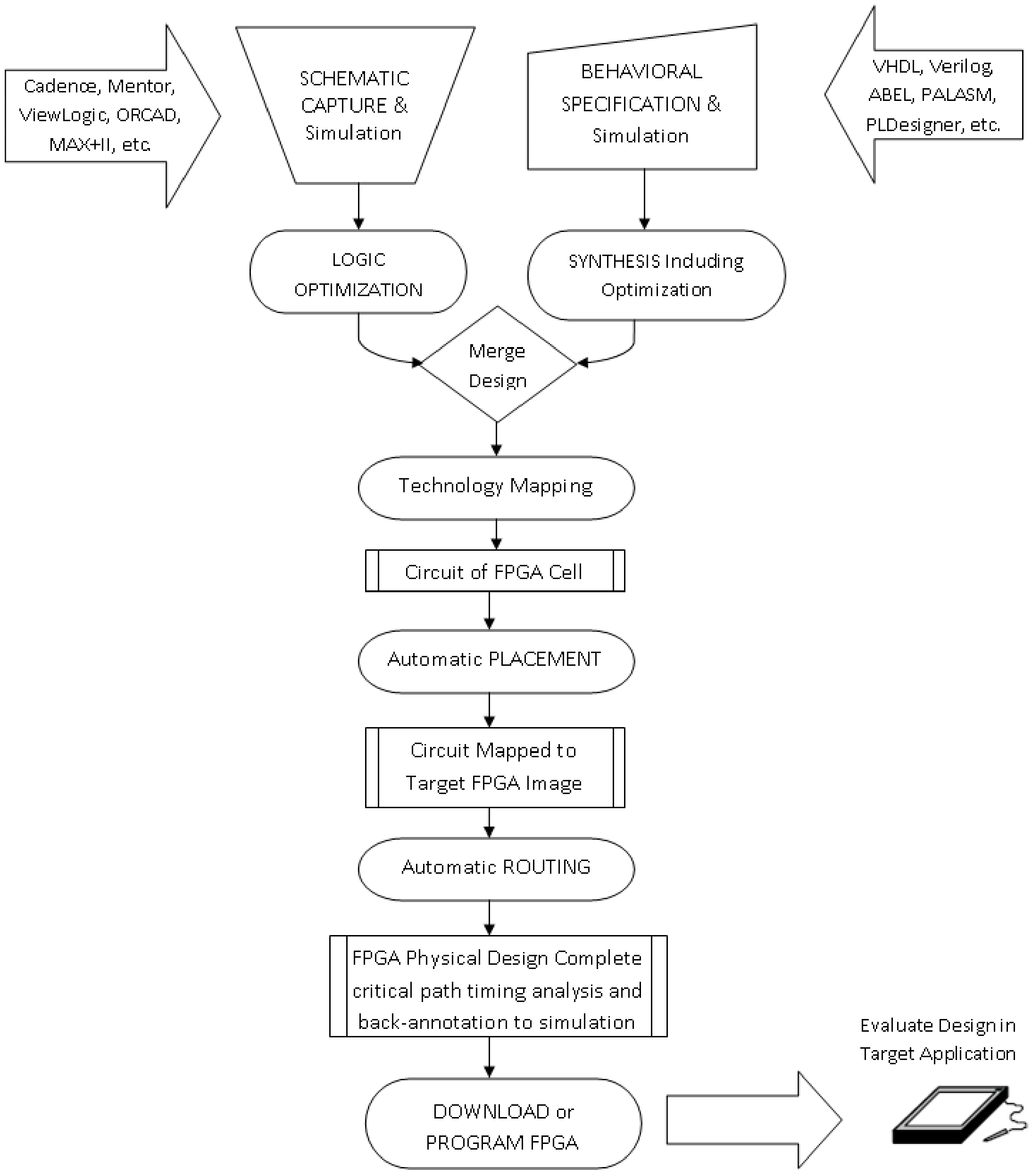

Cadence, Mentor, ViewLogic, ORCAD, MAX+II, etc.
SCHEMATIC CAPTURE & Simulation
BEHAVIORAL SPECIFICATION & Simulation
VHDL, Verilog, ABEL, PALASM, PLDesigner, etc.
LOGIC OPTIMIZATION
SYNTHESIS Including Optimization
Merge Design
Technology Mapping
Circuit of FPGA Cell
Automatic PLACEMENT
Circuit Mapped to Target FPGA Image
Automatic ROUTING
FPGA Physical Design Complete critical path timing analysis and back-annotation to simulation
DOWNLOAD or PROGRAM FPGA
Evaluate Design in Target Application

Fig. 3.7 CAD System Design Flow for FPGA

The upper right hand arrow in fig. 3.7 indicates some of the HDLs currently being used for FPGA behavioral modeling.

The behavioral description of designs can be done in many ways which include the VHSIC Hardware Description Language (VHDL), the Verilog hardware description language, timing diagrams and logic state diagrams. Which ever behavioral design entry method is chosen, the design system provides logic synthesis which automatically creates gate-level schematics. The next step in FPGA design is to translate the entire design into a standard form which can be processed by a logic optimization tool. The goal of logic optimization is to perform minimization of the Boolean expressions and eliminate redundancy, thus minimizing the area of the final circuit. The tool may also be constrained to maximize speed by limiting the number of logic levels. Simulation is performed both before and after the logic optimization steps to verify that the design meets the original system requirements for functionality and timing. The next step is to convert the gate level design into one which uses the FPGA circuit building blocks of the target technology. For example, the Xilinx ISE design system flow will be used to illustrate the steps needed to go from logic design to programmed FPGA. In the Xilinx design flow, the native format of the logic design must first be translated into the Xilinx Netlist Format (XNF) which is understood by the Xilinx tools. Next, the XNF circuit description must be mapped into Xilinx Configurable Logic Blocks (CLBs). This is the technology mapping step referred in fig. 3.7.

The next step is to place and route the design on the selected chip image. There are two types of placement and routing: manual and automatic. In the automatic placement operation, each CLB generated during the "technology mapping" step is assigned to a physical location on the chip. Interconnections between the CLBs must be routed using the available interconnect segments and switch matrix elements. With the physical placement and routing completed, exact timing values can now be used to determine chip performance. The final step in the Xilinx design flow is the creation of the "BIT" file which contains the binary programming data needed to configure the SRAM bits of the target chip. This file is then downloaded to configure the chip for final functional and timing tests of the programmed chip [61].

Chapter 4

Synthesis, Simulation and Timing Results

In this project we have taken some assumptions given as below:

1. Three users are already in the network and fourth user wants to admit in the network.
2. The PDF of bit rate of all the users is already known to us and is stored in the memory.

We divide our project into four sub-modules and then call these sub-modules into the final integrated unit. The sub-modules of this project are:

1. Convolution Unit
2. Probability Computational Unit
3. Comparator Unit
4. FIFO Buffer

4.1 Convolution Unit

The convolution unit is used to convolve the PDF of bit rate of users. When fourth user comes to the network with its packet which contain the address through which network get to know that it is not the packet of the existing user, then it gives the control signal to the convolution unit. Convolution unit take the PDF of bit rate of all the users from the meory and start its function. The simulation and synthesis results are given below:

4.1.1 Block Diagram

The block diagram of convolution unit is shown in fig. 4.1. It shows the number of inputs and ouputs in the unit and also shows that which input or output is represented by how many number of bits.

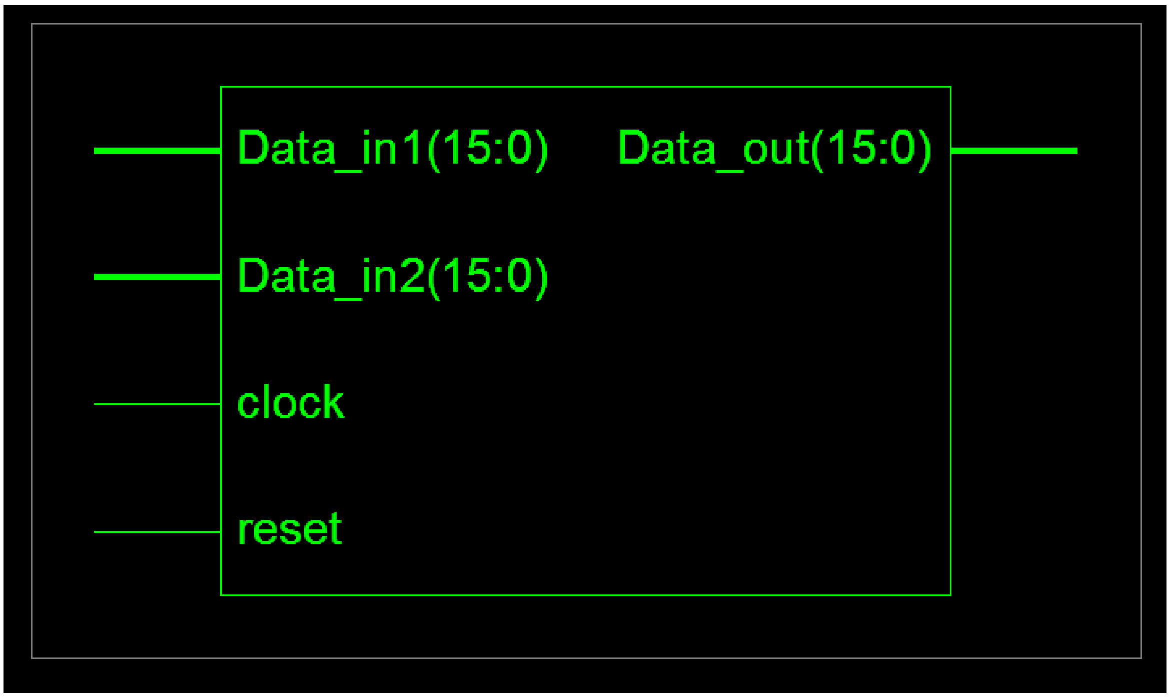


Fig. 4.1 Block Diagram of Convolution Unit

4.1.2 RTL Schematic

The RTL schematic of convolution unit is shown in fig. 4.2. It gives the full schematic veiw of the convolution unit.

4.1.3 Simulation Result

The simulation result of the convolution unit is shown in fig. 4.3. For the simulation of convolution unit we have taken two inputs, each have 9 samples of 16 bits. The output of convolution unit has 17 samples of 16 bits.

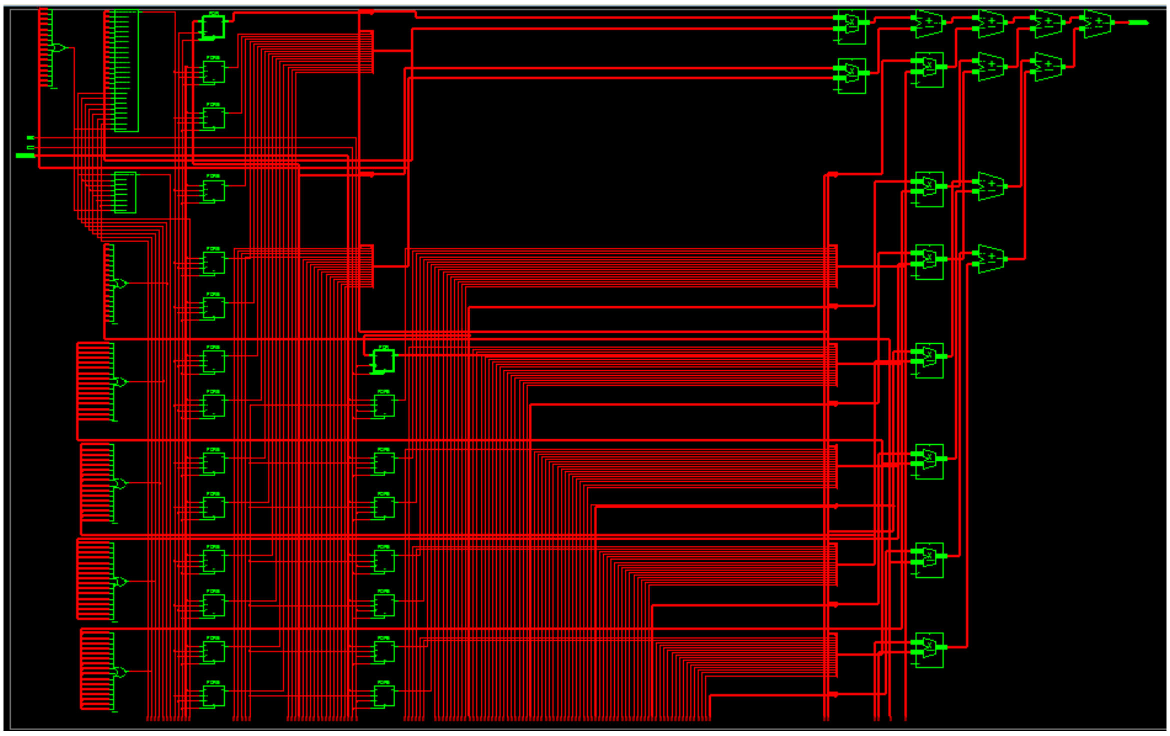


Fig. 4.2 RTL Schematic of Convolution Unit

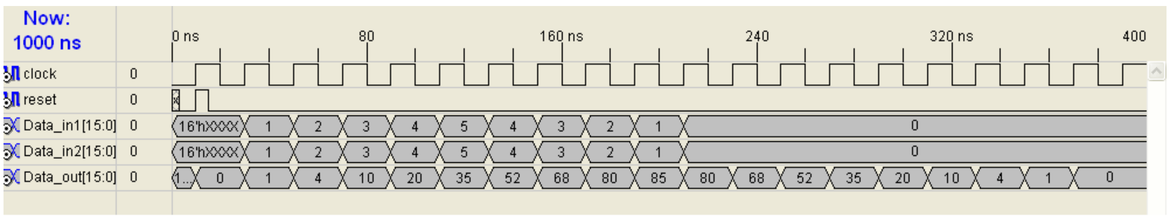
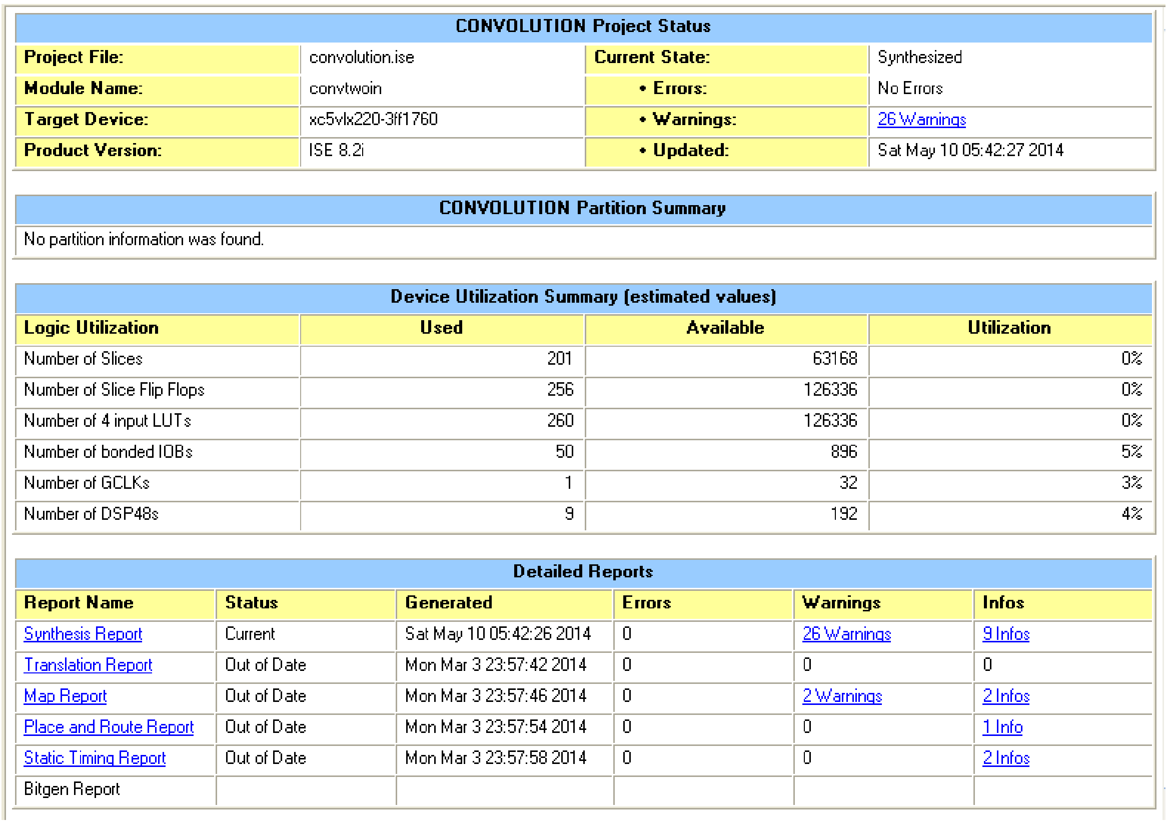


Fig. 4.3 Simulation Result of Convolution Unit

4.1.4 Synthesis Report

Fig. 4.4 shows the synthesis report of the convolution unit. Basically, the synthesis report shows the utilization of the logic devices like no. of slices which contain a pair of lookup tables and flip flops, no. of I/O bounds, no. of clock when we implement the hardware of the device. The minimum clock period of convolution unit is 5.34 *ns*.



CONVOLUTION Project Status			
Project File:	convolution.ise	**Current State:**	Synthesized
Module Name:	convtwoin	• **Errors:**	No Errors
Target Device:	xc5vlx220-3ff1760	• **Warnings:**	26 Warnings
Product Version:	ISE 8.2i	• **Updated:**	Sat May 10 05:42:27 2014

CONVOLUTION Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	201	63168	0%
Number of Slice Flip Flops	256	126336	0%
Number of 4 input LUTs	260	126336	0%
Number of bonded IOBs	50	896	5%
Number of GCLKs	1	32	3%
Number of DSP48s	9	192	4%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat May 10 05:42:26 2014	0	26 Warnings	9 Infos
Translation Report	Out of Date	Mon Mar 3 23:57:42 2014	0	0	0
Map Report	Out of Date	Mon Mar 3 23:57:46 2014	0	2 Warnings	2 Infos
Place and Route Report	Out of Date	Mon Mar 3 23:57:54 2014	0	0	1 Info
Static Timing Report	Out of Date	Mon Mar 3 23:57:58 2014	0	0	2 Infos
Bitgen Report					

Fig. 4.4 Synthesis Report of Convolution Unit

4.2 Probability Computational Unit

The probability computational unit takes the input from the convolution unit and it calculates the probability of combined bit rate of all users is less than the maximum output bit rate. Basically it calculates the sum of samples of convolution output upto the no. of maximum output bit rate. The simulation and synthesis results are given below:

4.2.1 Block Diagram

The block diagram of probability computational unit is shown in fig. 4.5.

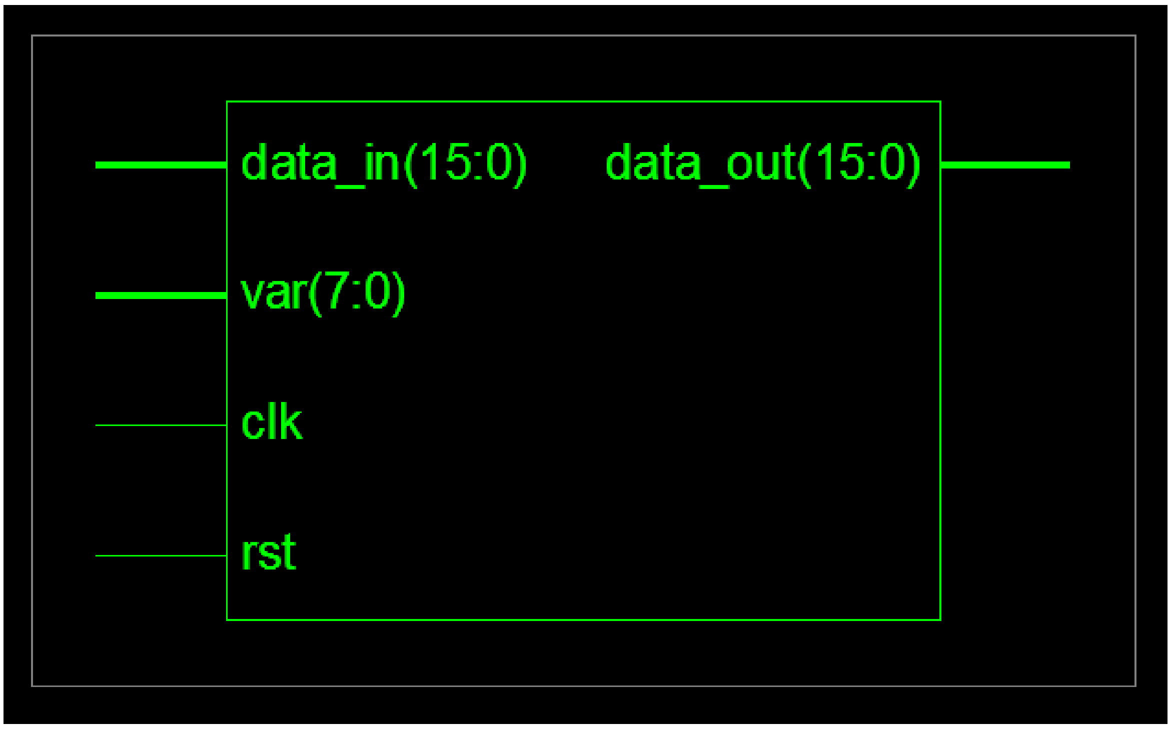


Fig. 4.5 Block Diagram of Probability Computational Unit

4.2.2 RTL Schematic

The RTL schematic of probability computational unit is shown in fig. 4.6.

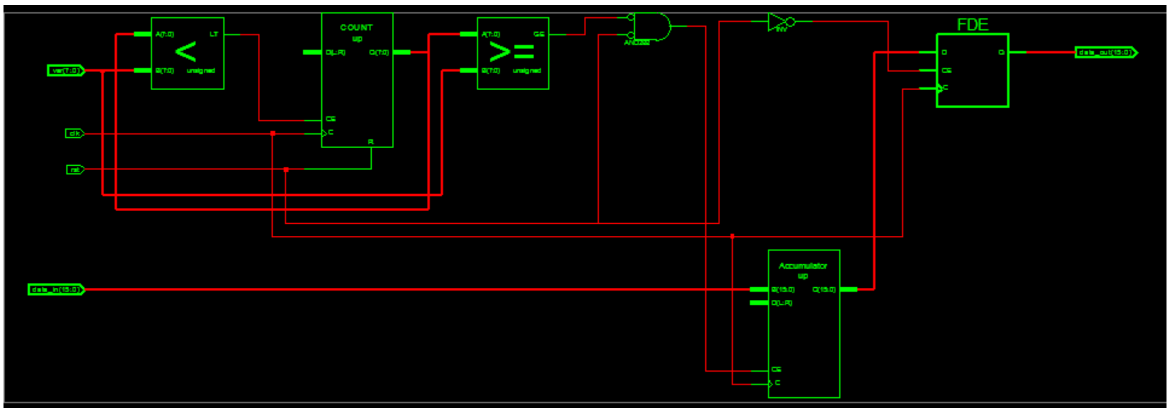


Fig. 4.6 RTL Schematic of Probability Computational Unit

4.2.3 Simulation Result

The simulation result of the probability computational unit is shown in fig. 4.7. For the demonstration purpose we add only 5 samples of the output of convolution unit. The output of probability computational unit gives the addition of 5 samples.

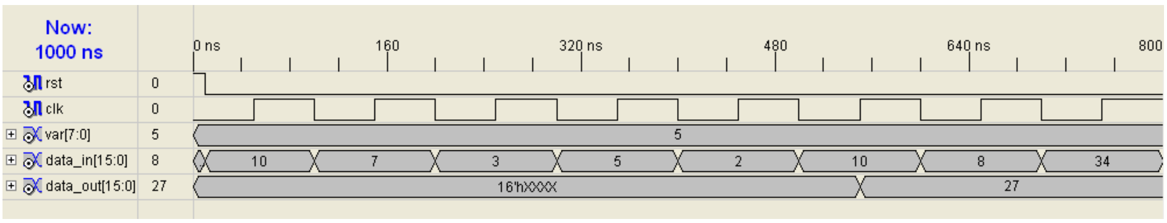
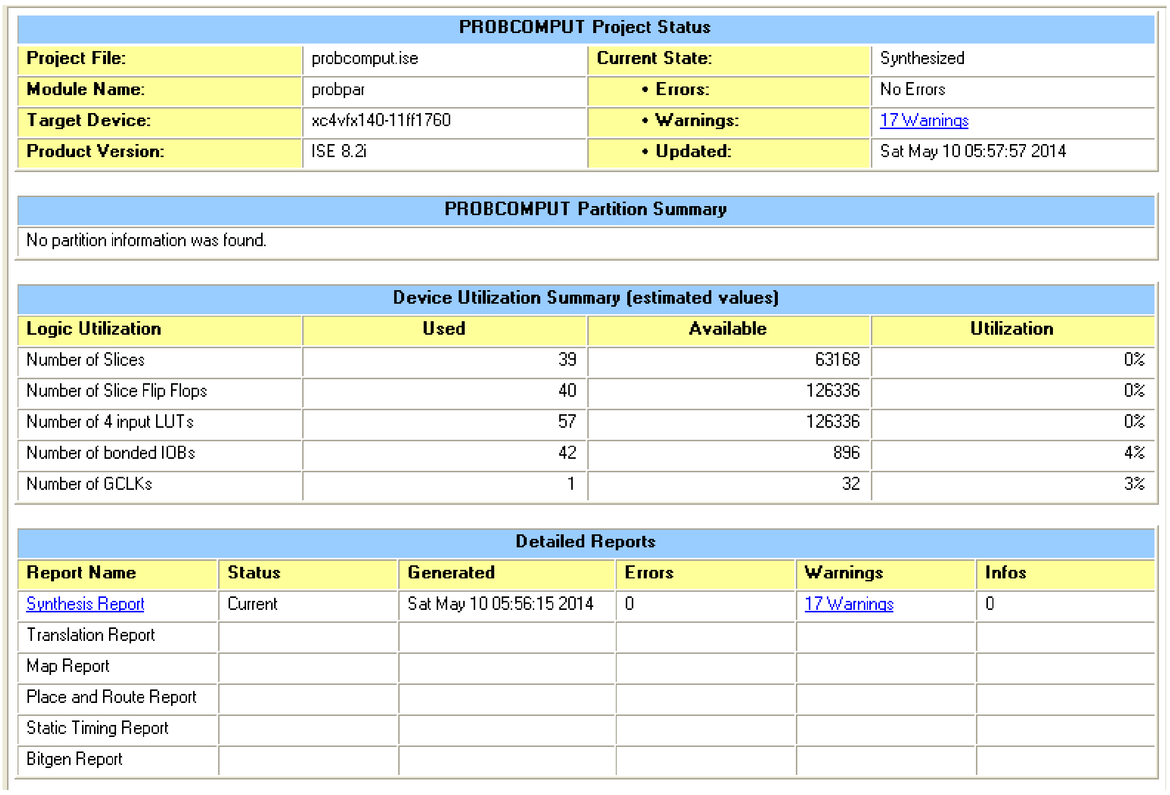


Fig. 4.7 Simulation Result of Probability Computational Unit

4.2.4 Synthesis Report

Fig. 4.8 shows the synthesis report of the probability computational unit. The minimum clock period of probability computational unit is 3.61 *ns*.



PROBCOMPUT Project Status			
Project File:	probcomput.ise	**Current State:**	Synthesized
Module Name:	probpar	• **Errors:**	No Errors
Target Device:	xc4vfx140-11ff1760	• **Warnings:**	17 Warnings
Product Version:	ISE 8.2i	• **Updated:**	Sat May 10 05:57:57 2014

PROBCOMPUT Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	39	63168	0%
Number of Slice Flip Flops	40	126336	0%
Number of 4 input LUTs	57	126336	0%
Number of bonded IOBs	42	896	4%
Number of GCLKs	1	32	3%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat May 10 05:56:15 2014	0	17 Warnings	0
Translation Report					
Map Report					
Place and Route Report					
Static Timing Report					
Bitgen Report					

Fig. 4.8 Synthesis Report of Probability Computational Unit

4.3 Comparator Unit

The comparator unit takes one input from the probability computational unit and other input from the memory. Other input indicates the threshold value. If the output of probability computational unit is less than the threshold value only then the user can admit in the network. The simulation and synthesis results are given below:

4.3.1 Block Diagram

The block diagram of comparator unit is shown in fig. 4.9.

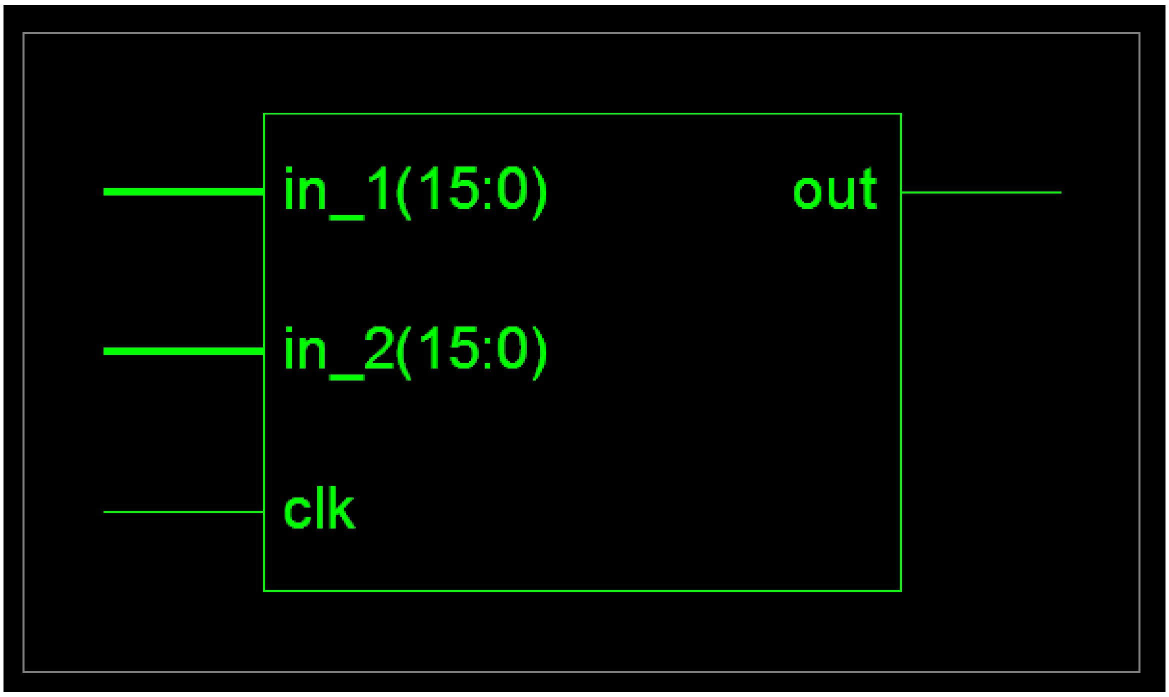


Fig. 4.9 Block Diagram of Comparator Unit

4.3.2 RTL Schematic

The RTL schematic of comparator unit is shown in fig. 4.10.

4.3.3 Simulation Result

The simulation result of the comparator unit is shown in fig. 4.11.

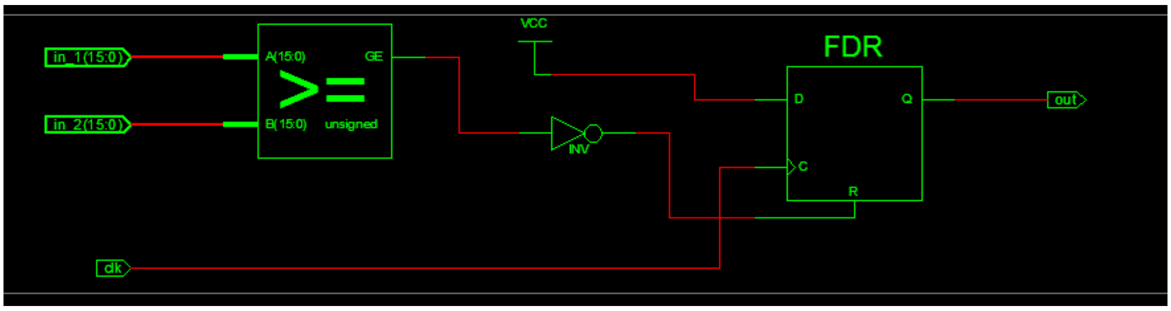


Fig. 4.10 RTL Schematic of Comparator Unit

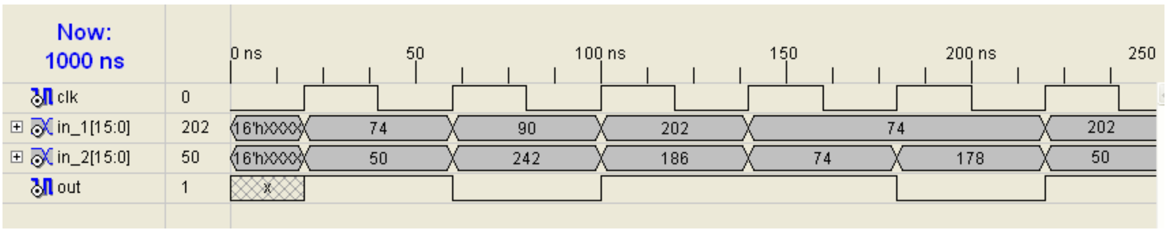


Fig. 4.11 Simulation Result of Comparator Unit

4.3.4 Synthesis Report

Fig. 4.12 shows the synthesis report of the comparator unit.

4.4 FIFO Buffer Unit

When the output line is busy and when the packet generation time overlap, the packets are stored in the buffer. The simulation and synthesis results are given below:

4.4.1 Block Diagram

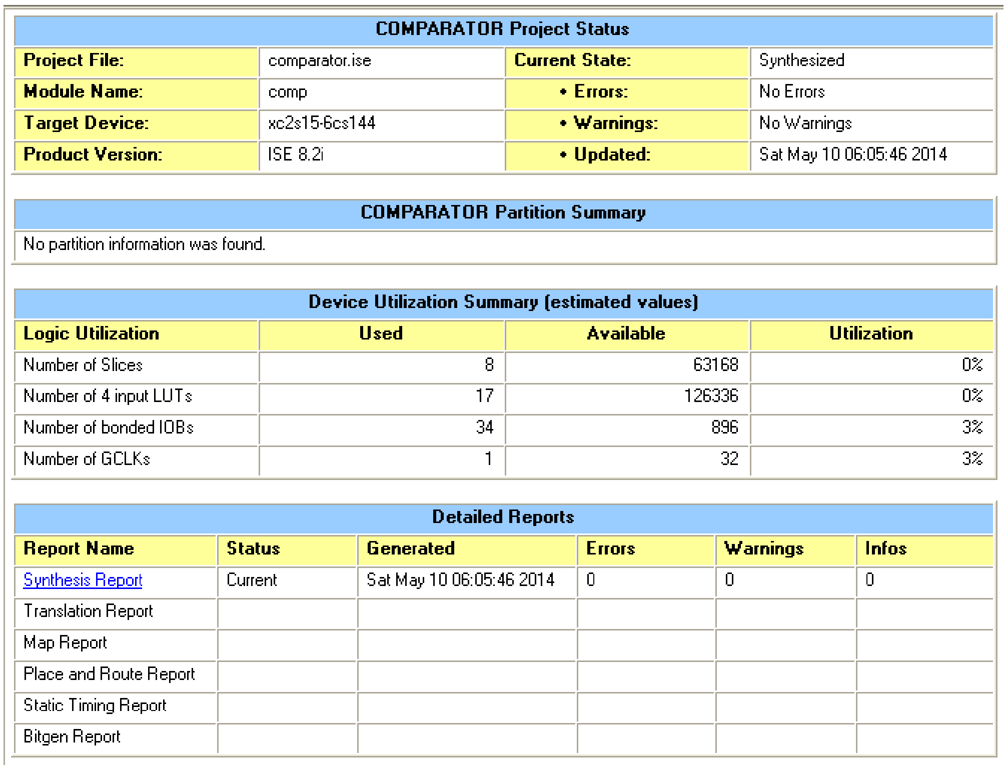
The block diagram of FIFO is shown in fig. 4.13.

4.4.2 RTL Schematic

The RTL schematic of FIFO is shown in fig. 4.14.

4.4.3 Simulation Result

The simulation result of the FIFO is shown in fig. 4.15. For the demonstration purpose we have assumed that the FIFO can only store the 8 samples of 16 bit.



COMPARATOR Project Status			
Project File:	comparator.ise	**Current State:**	Synthesized
Module Name:	comp	• **Errors:**	No Errors
Target Device:	xc2s15-6cs144	• **Warnings:**	No Warnings
Product Version:	ISE 8.2i	• **Updated:**	Sat May 10 06:05:46 2014

COMPARATOR Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	8	63168	0%
Number of 4 input LUTs	17	126336	0%
Number of bonded IOBs	34	896	3%
Number of GCLKs	1	32	3%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat May 10 06:05:46 2014	0	0	0
Translation Report					
Map Report					
Place and Route Report					
Static Timing Report					
Bitgen Report					

Fig. 4.12 Synthesis Report of Comparator Unit

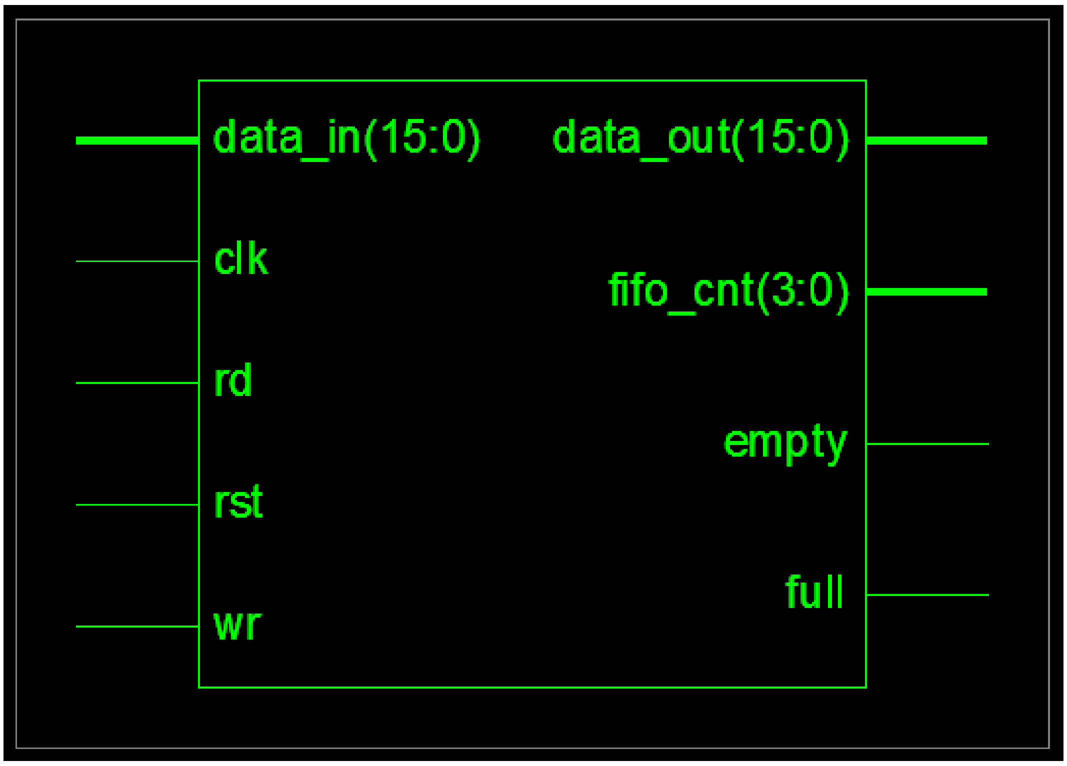


Fig. 4.13 Block Diagram of FIFO

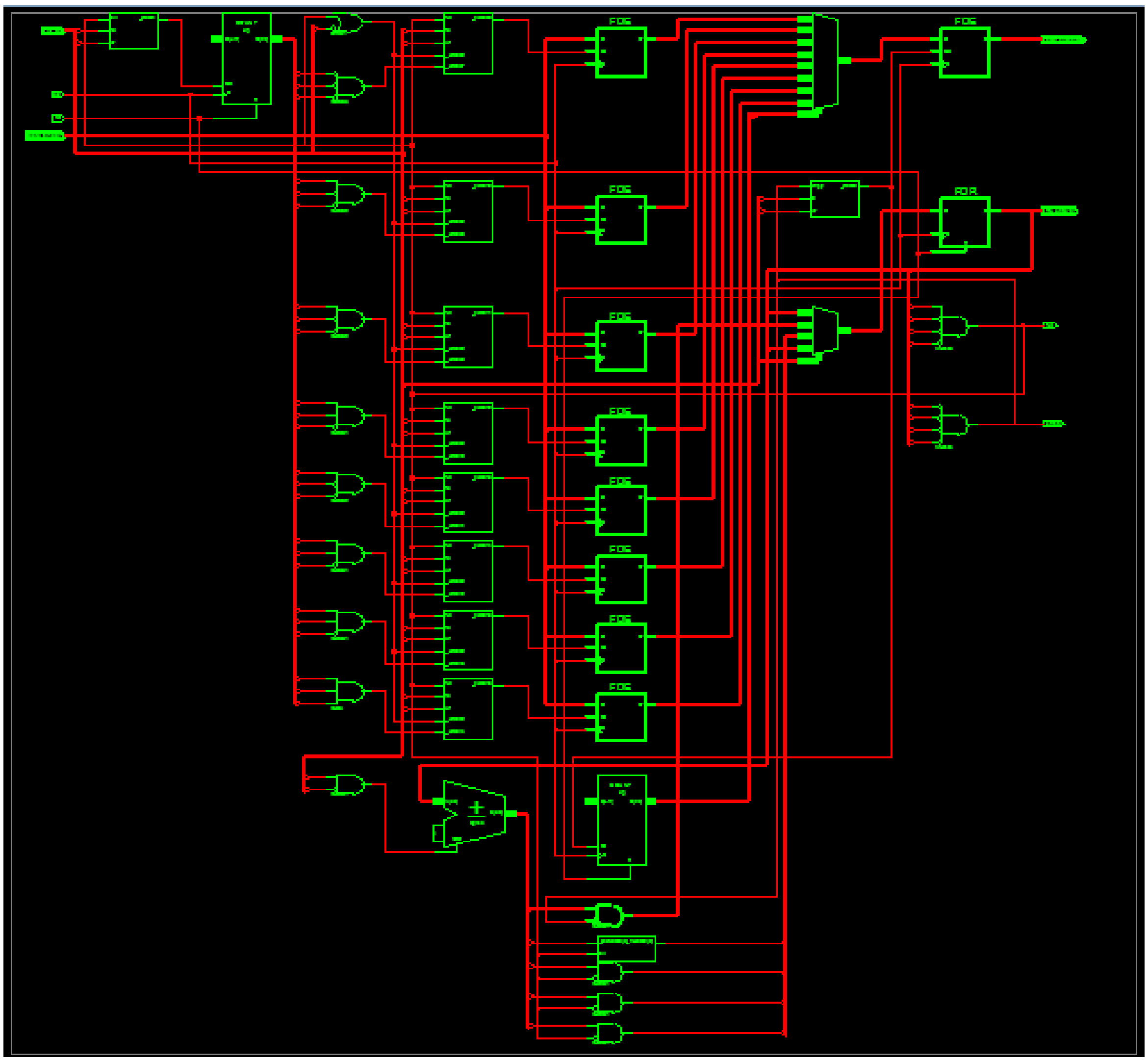


Fig. 4.14 RTL Schematic of FIFO

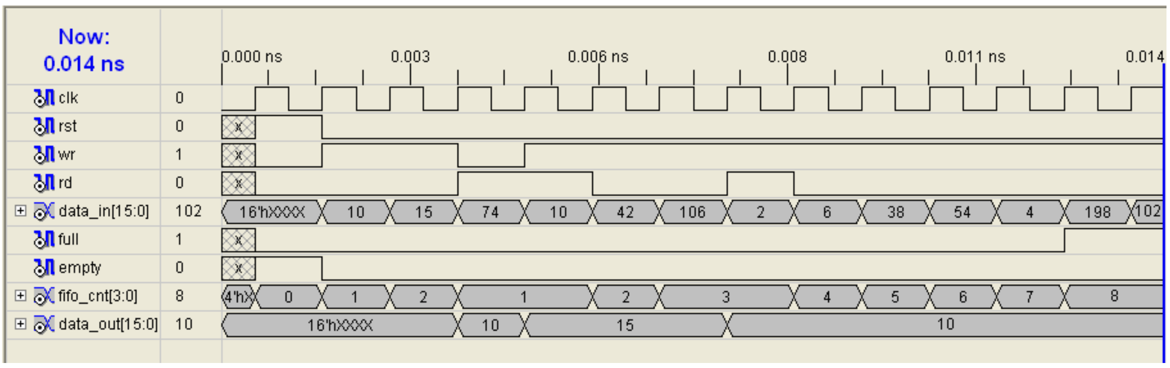
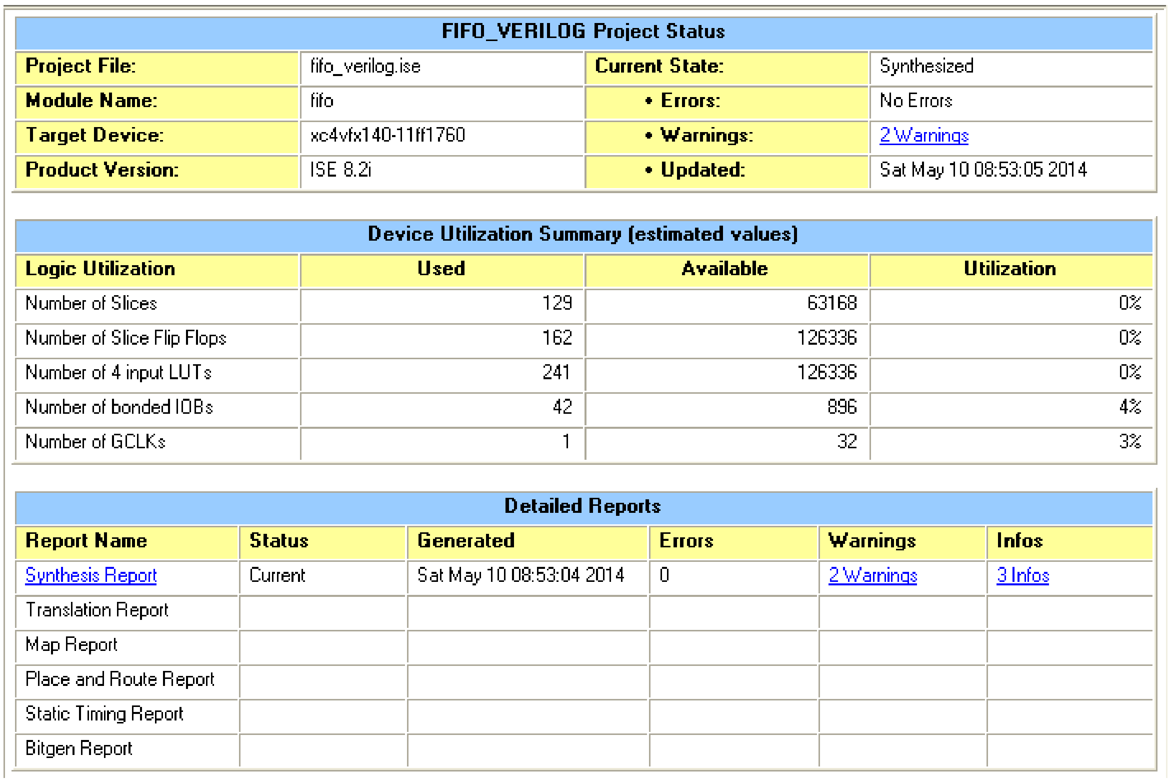


Fig. 4.15 Simulation Result of FIFO

4.4.4 Synthesis Report

Fig. 4.16 shows the synthesis report of the FIFO. The minimum clock period of FIFO unit is 3.33 *ns*.



FIFO_VERILOG Project Status			
Project File:	fifo_verilog.ise	**Current State:**	Synthesized
Module Name:	fifo	• **Errors:**	No Errors
Target Device:	xc4vfx140-11ff1760	• **Warnings:**	2 Warnings
Product Version:	ISE 8.2i	• **Updated:**	Sat May 10 08:53:05 2014

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	129	63168	0%
Number of Slice Flip Flops	162	126336	0%
Number of 4 input LUTs	241	126336	0%
Number of bonded IOBs	42	896	4%
Number of GCLKs	1	32	3%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat May 10 08:53:04 2014	0	2 Warnings	3 Infos
Translation Report					
Map Report					
Place and Route Report					
Static Timing Report					
Bitgen Report					

Fig. 4.16 Synthesis Report of FIFO

4.5 Final Integrated System

All the sub-modules described above are interfaced in the final integrated unit. For the convolution of PDF of bit rate of 4 users, convolution unit is called three times in the final integrated unit. All the sub-modules work parallely in the final integrated unit, means one module will not wait for the whole output of second module. When it got the one or two samples from the previous module it starts its working. The simulation and synthesis results of the final integrated unit are given below:

4.5.1 Block Diagram

The block diagram of final integrated unit is shown in fig. 4.17.

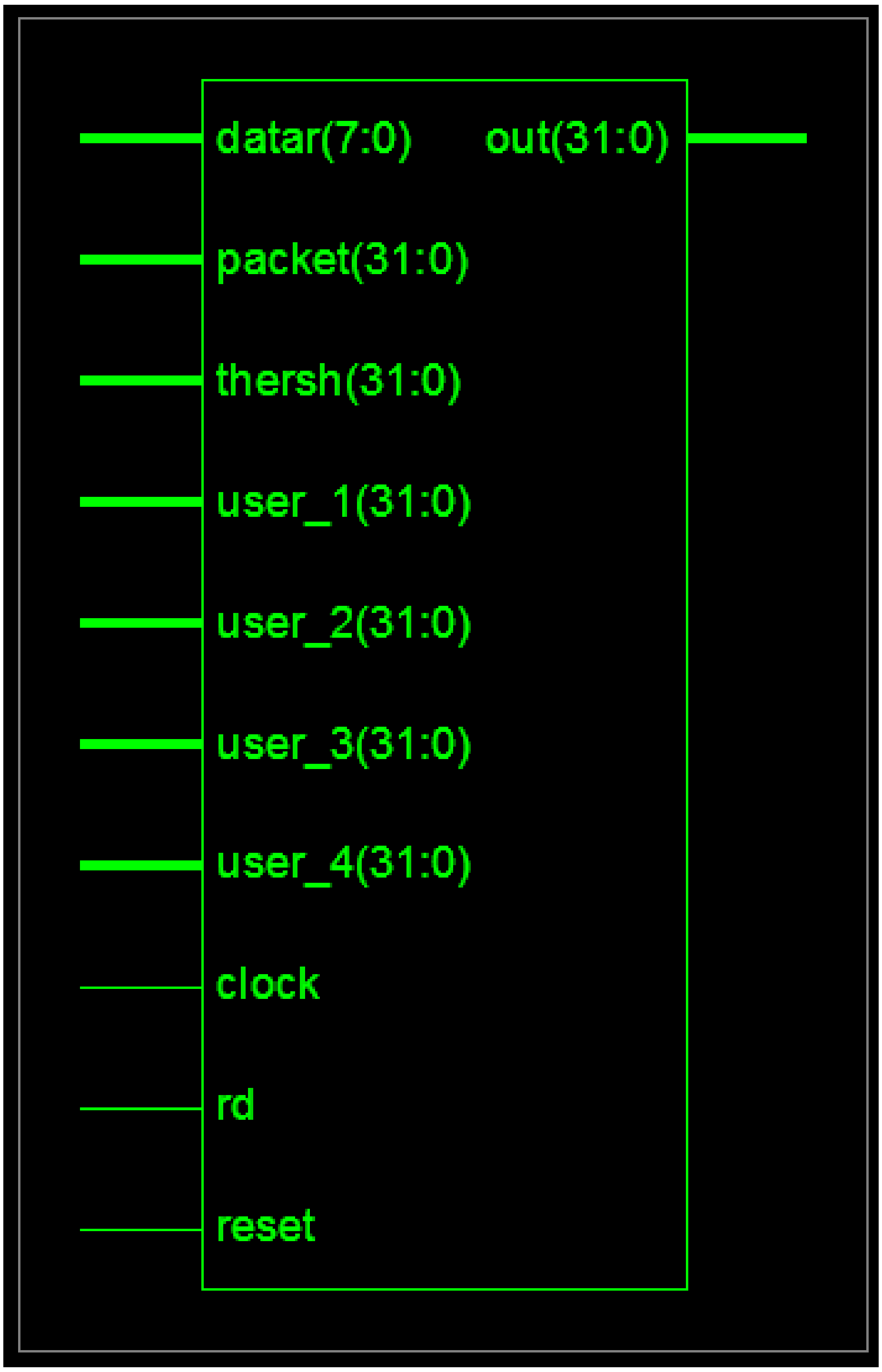


Fig. 4.17 Block Diagram of Final Integrated Unit

4.5.2 RTL Schematic

The RTL schematic of final integrated unit is shown in fig. 4.18.

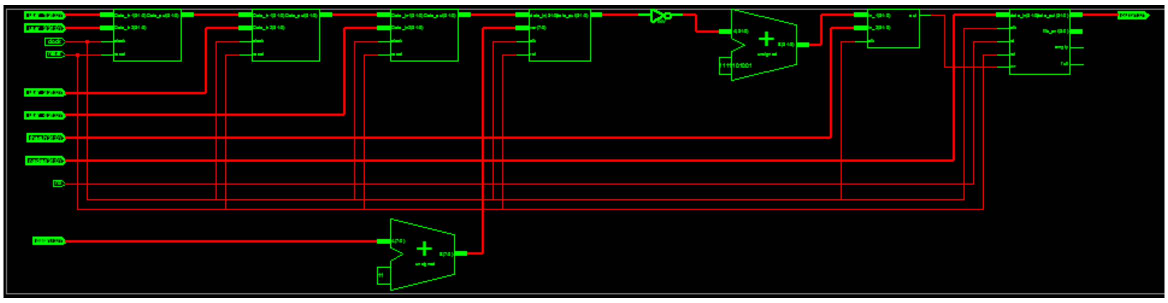


Fig. 4.18 RTL Schematic of Final Integrated Unit

4.5.3 Simulation Result

The simulation result of the final integrated unit is shown in fig. 4.19.

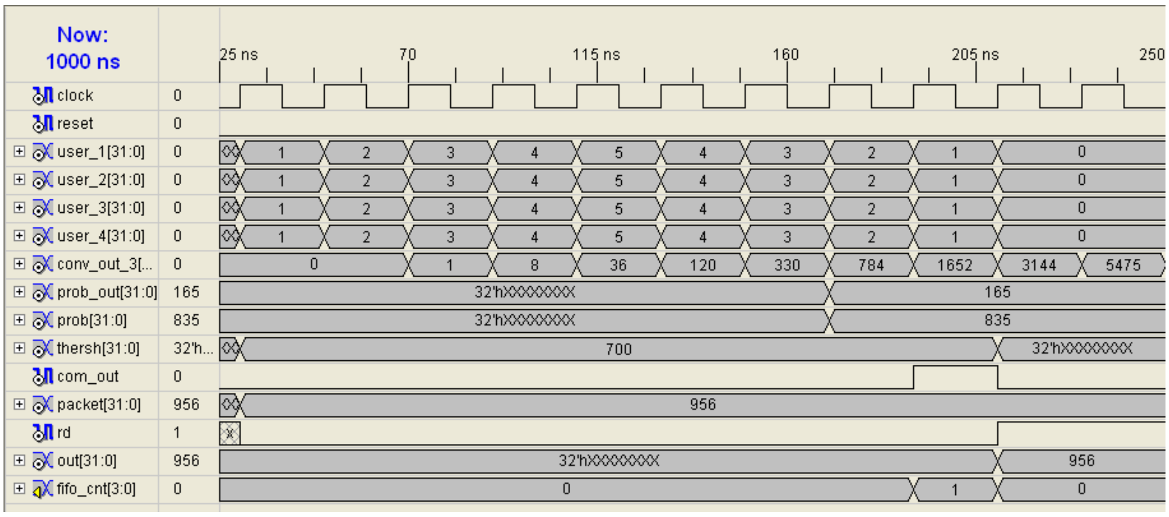
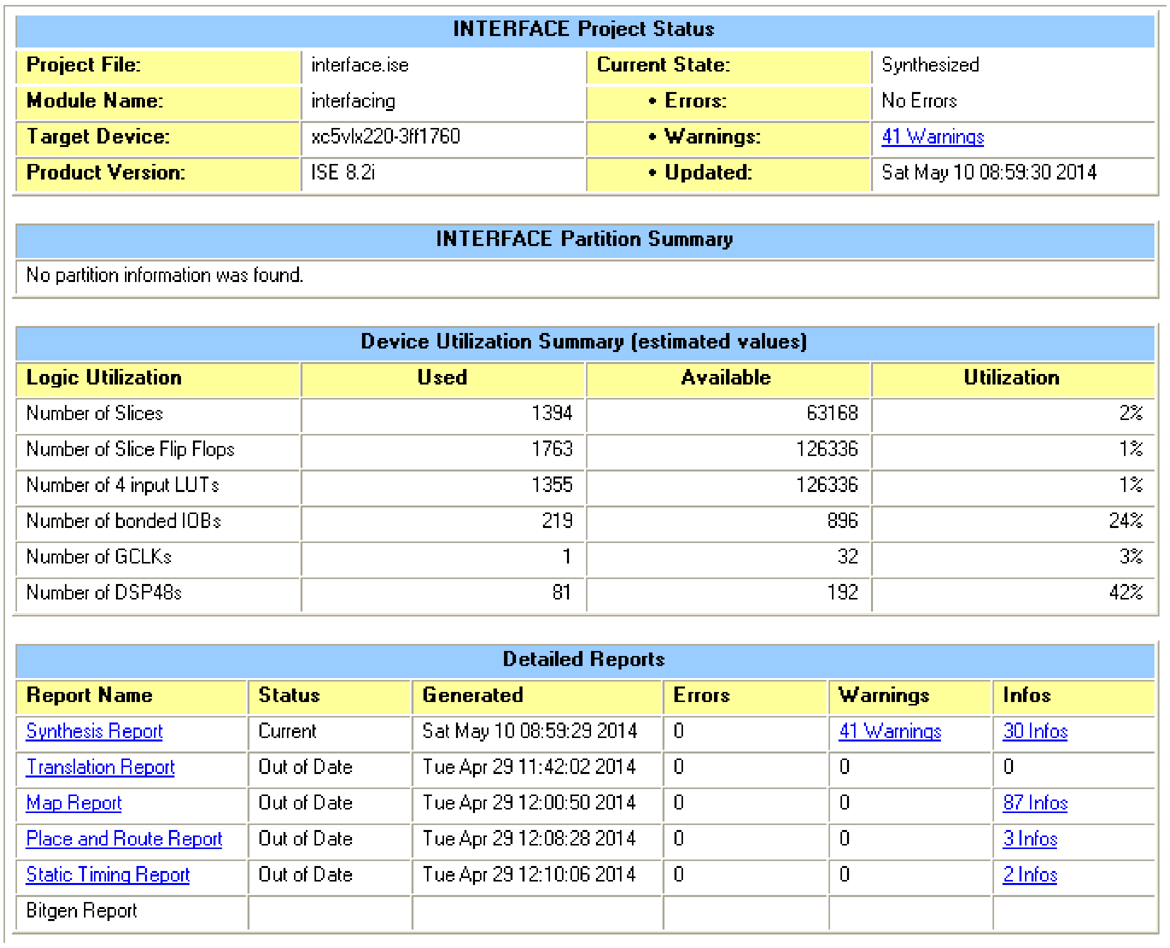


Fig. 4.19 Simulation Result of Final Integrated Unit

4.5.4 Synthesis Report

Fig. 4.20 shows the synthesis report of the final integrated unit. The minimum clock period of final integrated unit is 9.297 *ns*.



INTERFACE Project Status			
Project File:	interface.ise	**Current State:**	Synthesized
Module Name:	interfacing	• **Errors:**	No Errors
Target Device:	xc5vlx220-3ff1760	• **Warnings:**	41 Warnings
Product Version:	ISE 8.2i	• **Updated:**	Sat May 10 08:59:30 2014

INTERFACE Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1394	63168	2%
Number of Slice Flip Flops	1763	126336	1%
Number of 4 input LUTs	1355	126336	1%
Number of bonded IOBs	219	896	24%
Number of GCLKs	1	32	3%
Number of DSP48s	81	192	42%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat May 10 08:59:29 2014	0	41 Warnings	30 Infos
Translation Report	Out of Date	Tue Apr 29 11:42:02 2014	0	0	0
Map Report	Out of Date	Tue Apr 29 12:00:50 2014	0	0	87 Infos
Place and Route Report	Out of Date	Tue Apr 29 12:08:28 2014	0	0	3 Infos
Static Timing Report	Out of Date	Tue Apr 29 12:10:06 2014	0	0	2 Infos
Bitgen Report					

Fig. 4.20 Synthesis Report of Final Integrated Unit

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

The design and implementation of “Statistical Multiplexer” has been done. The description was made by Verilog-HDL in Xilinx ISE on Vertex-series family. We divided this project into different module and find out satisfactory results for each module in terms of minimum clock period and simplicity. All the modules are performing their operation in very less time only in the range of “ns”. Here we find out the minimum clock period for each modules are-

- Convolution Unit- 5.34 ns
- Probability Computational Unit- 3.61 ns
- FIFO Buffer- 3.33 ns
- Final Integrated Unit- 9.297 ns

After synthesizing the all module we find out, all the modules are synthesized, mapped and fit for the implementation on the FPGA platform. FPGA implementation is faster than the software implementation.

5.2 Future Scope

Even though we achieved our aim as we design a high speed statistical multiplexer. But still works are going on to increase more and more speed and to reduce area. Statistical multiplexer has many applications like digital TV transmission, digital broadcasting, asynchronous transfer mode, UDP-TCP protocol, X.25 and frame relay. Its a challenging area where reaserch is going on to increase the speed provided by the service provider.

References

[1] Yanxiang, Zhen, Su Fang, and Xu Huimin. "The Probability Distribution Modeling and Parameter Estimation Method of ROD in Different Packet Loss Pattern for Heterogeneous Network." In *Communications and Mobile Computing, 2009. CMC'09. WRI International Conference on*, vol. 2, pp. 185-190. IEEE, 2009.

[2] Chaki, S-I., Hiroshi Saito, Kou Miyake, and H. Ohnishi. "ATM network for high-speed data communication." In *Networks, 1993. International Conference on Information Engineering'93.'Communications and Networks for the Year 2000', Proceedings of IEEE Singapore International Conference on*, vol. 1, pp. 123-127. IEEE, 1993.

[3] Changuel, Nesrine, Bessem Sayadi, and Michel Kieffer. "Joint encoder and buffer control for statistical multiplexing of multimedia contents." In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1-6. IEEE, 2010.

[4] Saito, Hiroshi. "Call admission control in an ATM network using upper bound of cell loss probability." *Communications, IEEE Transactions on* 40, no. 9 (1992): 1512-1521.

[5] D'Apice, C., R. Manzo, S. Salerno, and N. Likhanov. "Network Traffic Modeling and Packet-Loss Probability Approximation." *Journal of Mathematical Sciences* 132, no. 5 (2006): 590-601.

[6] Lombardo, A., S. Palazzo, and G. Schenbra. "Management of wireless ATM networks loaded by mobile ABR source traffic." In *Global Telecommunications Conference, 1999. GLOBECOM'99*, vol. 5, pp. 2758-2762. IEEE, 1999.

[7] Sarajedini, Amir, and Paul M. Chau. "Quality of service prediction using neural networks." In *Military Communications Conference, 1996. MILCOM'96, Conference Proceedings, IEEE*, vol. 2, pp. 567-570. IEEE, 1996.

[8] Ouyang, Tung, and A. A. Nilsson. "A simulation study of an ATM service multiplexer in the North Carolina Information Highway." In *Southeastcon'96. Bringing Together Education, Science and Technology., Proceedings of the IEEE*, pp. 348-351. IEEE, 1996.

[9] Zheng, Lizhong, and David N. C. Tse. "Diversity and multiplexing: A fundamental tradeoff in multiple-antenna channels." *Information Theory, IEEE Transactions* on 49, no. 5 (2003): 1073-1096.

[10] Razouqi, Qutaiba, T. Lee, Seong-Soon Joo, and Sumit Ghosh. “Incremental, dynamic, virtual circuit connection (IVCC): a new paradigm for routing in future high-speed networks.” In *Communications, 2001. ICC 2001. IEEE International Conference on*, vol. 8, pp. 2578-2582. IEEE, 2001.

[11] Jackson, Leland B., J. Kaiser, and H. McDonald. “An approach to the implementation of digital filters.” *Audio and Electroacoustics, IEEE Transactions on* 16, no. 3 (1968): 413-421.

[12] Zhang, Dongli, and Dan Ionescu. “Reactive estimation of packet loss probability for IP-based video services.” *Broadcasting, IEEE Transactions* on 55, no. 2 (2009): 375-385.

[13] Nestle, Elliot, and Robert F. Schunneman. “TIME DIVISION MULTIPLEXING.” U.S. Patent 3,599,160, issued August 10, 1971.

[14] Shay, William A. “Time Division Multiplexing.” Handbook of Computer Networks: Key Concepts, Data Transmission, and Digital and Optical Networks, Volume 1: 568-578.

[15] Jiawen, Wang, Li Li, Zhang Yuang, Pan Hongbing, He Shuzhuan, and Zhang Rong. “Statistical time division multiplexing based local system architecture for multi-cluster NoC.” In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference* on, pp. 472-476. IEEE, 2011.

[16] Lo, Chi-Wen, Chia-Wen Lin, Yung-Chang Chen, and Jen-Yu Yu. “A packet loss estimation model and its application to reliable mesh-based P2P video streaming.” In *Multimedia and Expo (ICME), 2011 IEEE International Conference* on, pp. 1-6. IEEE, 2011.

[17] Rezgui, Jihene, Abdelhakim Hafid, and Michel Gendreau. “Distributed admission control in wireless mesh networks: models, algorithms, and evaluation.” *Vehicular Technology, IEEE Transactions* on 59, no. 3 (2010): 1459-1473.

[18] Huang, Zheng F. “Stochastic time division multiplexing.” U.S. Patent 4,700,341, issued October 13, 1987.

[19] Cao, Jin, William S. Cleveland, Dong Lin, and Don X. Sun. “The effect of statistical multiplexing on internet packet traffic: Theory and empirical study.” *Internet available at: http://cm. bell-labs. com/cm/ms/departments/sia/InternetTraffic/webpapers. html* (2001).

[20] Zhang, Z., and A. S. Acampora. “Effect of on/off distributions on the cell loss probability in ATM networks.” In *Global Telecommunications Conference, 1992. Conference Record., GLOBECOM’92. Communication for Global Users., IEEE*, pp. 1533-1539. IEEE, 1992.

[21] Hsu, Cheng-Hsin, and Mohamed Hefeeda. “Statistical multiplexing of variable-bit-rate videos streamed to mobile devices.” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)* 7, no. 2 (2011): 12.

[22] Wu, Guo-Liang, and Jon W. Mark. "Computational methods for performance evaluation of a statistical multiplexer supporting bursty traffic." *IEEE/ACM Transactions on Networking (TON)* 4, no. 3 (1996): 386-397.

[23] Chu, W. W. "A study of asynchronous time division multiplexing for time-sharing computer systems." In *Proceedings of the November 18-20, 1969, fall joint computer conference*, pp. 669-678. ACM, 1969.

[24] Chandra, Kavitha. "Statistical multiplexing." *Encyclopedia of Telecommunications* (2003).

[25] Maglaris, Basil, Dimitris Anastassiou, Prodip Sen, Gunnar Karlsson, and John D. Robbins. "Performance models of statistical multiplexing in packet video communications." *Communications, IEEE Transactions on 36*, no. 7 (1988): 834-844.

[26] Boorstyn, Robert, Almut Burchard, Jörg Liebeherr, and Chaiwat Oottamakorn. "Statistical multiplexing gain of link scheduling algorithms in QoS networks."(1999).

[27] Huang, Jianwei, Chee Wei Tan, Mung Chiang, and Raphael Cendrillon. "Statistical multiplexing over DSL networks." In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 571-579. IEEE, 2007.

[28] Sherif, M. R., I. W. Habib, M. Naghshineh, and P. Kermani. "A generic bandwidth allocation scheme for multimedia substreams in adaptive networks using genetic algorithms." In *Wireless Communications and Networking Conference, 1999. WCNC. 1999 IEEE*, pp. 1243-1247. IEEE, 1999.

[29] de Veciana, Gustavo, and Ross Baldick. "Resource allocation in multi-service networks via pricing: statistical multiplexing." *Computer Networks and ISDN Systems* 30, no. 9 (1998): 951-962.

[30] Hsu, Cheng-Hsin, and Mohamed Hefeeda. "On statistical multiplexing of variable-bit-rate video streams in mobile systems." In *Proceedings of the 17th ACM international conference on Multimedia*, pp. 411-420. ACM, 2009.

[31] Lahdili, Hassan, Hossein Najaf-Zadeh, and Louis Thibault. "Statistical multiplexing for digital audio broadcasting applications." *Broadcasting, IEEE Transactions* on 56, no. 1 (2010): 19-27.

[32] Cidon, Israel, Asad Khamisy, and Moshe Sidi. "Analysis of packet loss processes in high-speed networks." *Information Theory, IEEE Transactions on* 39, no. 1 (1993): 98-108.

[33] Afonso, George, Rabie Ben Atitallah, and Jean-Luc Dekeyser. "Software Implementation vs. Hardware Implementation: The Avionic Test System Case-Study." In *ASPLOS*. 2012.

[34] DeHon, André. "The density advantage of configurable computing." Computer 33, no. 4 (2000): 41-49.

[35] Iera, Antonio, Salvatore Marano, and Antonella Molinaro. “On the management of end-to-end multimedia and multibearer connections in a personal communications system.” In *Personal, Indoor and Mobile Radio Communications, 1996. PIMRC’96., Seventh IEEE International Symposium* on, vol. 3, pp. 1178-1182. IEEE, 1996.

[36] Lee, J. Y., and C. K. Un. “Performance analysis of an input and output queueing packet switch with a priority packet discarding scheme.” IEE Proceedings-Communications 142, no. 2 (1995): 67-74.

[37] Likhanov, Nikolay B., Ravi R. Mazumdar, and Francois Theberge. “Providing QoS in large networks: Statistical multiplexing and admission control.” In *Analysis, Control and Optimization of Complex Dynamic Systems*, pp. 137-167. Springer US, 2005.

[38] Spitler, Stephen L., and Daniel C. Lee. “Optimization of call admission control for a statistical multiplexer allocating link bandwidth.” *Automatic Control, IEEE Transactions on 48*, no. 10 (2003): 1830-1836.

[39] Zhao, Dongmei, Xuemin Shen, and Jon W. Mark. “Efficient call admission control for heterogeneous services in wireless mobile ATM networks.” *Communications Magazine, IEEE* 38, no. 10 (2000): 72-78.

[40] Oh, Hyun-Taek, and Seung-Woo Seo. “A new call admission control algorithm for multi-class services in wireless ATM.” In *Vehicular Technology Conference, 1999. VTC 1999-Fall. IEEE VTS 50th*, vol. 3, pp. 1740-1744. IEEE, 1999.

[41] Ogino, Nagao, and Yasushi Wakahara. “An ATM call admission control scheme using neural network.” In *Multimedia Communications, 1994. MULTIMEDIA’94., 5th IEEE COMSOC International Workshop on*, pp. 7-2. IEEE, 1994.

[42] Aussem, Alex. “Call admission control in ATM networks with the random neural network.” In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 4, pp. 2482-2487. IEEE, 1994.

[43] Ishizaki, Fumio, Tetsuya Takine, Hiroaki Terada, and Toshiharu Hasegawa. “Loss probability approximation of a statistical multiplexer and its application to call admission control in high- speed networks.” In *Global Telecommunications Conference, 1995. GLOBECOM’95., IEEE.*, vol. 1, pp. 417-421. IEEE, 1995.

[44] Whitt, Ward. “Tail probabilities with statistical multiplexing and effective bandwidths in multi-class queues.” *Telecommunication Systems* 2, no. 1 (1993): 71-107.

[45] Rizk, Amr, and Markus Fidler. “On Multiplexing Models for Independent Traffic Flows in Single-and Multi-Node Networks.” *Network and Service Management, IEEE Transactions on* 10, no. 1 (2013): 15-28.

[46] Matsuoka, Shirou, Sumiko Miyata, and Katsunori Yamaoka. “Relationship between packet loss probability and burst parameters for two types of traffic.” *Proc. of IEEE NoF2012* (2012): 1-5.

[47] Miyata, Sumiko, and Katsunori Yamaoka. “Flow-admission control based on equality of heterogeneous traffic (two-type flow model).” *IEICE Transactions on Communications* 93, no. 12 (2010): 3564-3576.

[48] Changuel, Nesrine, Bessem Sayadi, and Michel Kieffer. “Predictive Encoder and Buffer Control for Statistical Multiplexing of Multimedia Contents.” *Broadcasting, IEEE Transactions* on 58, no. 3 (2012): 401-416.

[49] Teixeira, Luís, and Luís Corte-Real. “Statistical multiplexing of H. 264 programs.” In *Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on*, pp. 621-625. IEEE, 2007.

[50] Kirk, David B., and W. Hwu Wen-mei. Programming massively parallel processors: a hands-on approach. Newnes, 2012.

[51] Omey, E. “Asymptotic properties of convolution products of functions.” Publ Inst Math (NS) 43, no. 57 (1988): 41-57.

[52] Ang, Alfredo HS, and Wilson H. Tang. “Probability concepts in engineering.” Planning 1, no. 4 (2004): 1-3.

[53] Parson, Dale. “Real-time resource allocators in network processors using FIFOs.” In *Proceedings of Advanced Networking and Communications Hardware Workshop (ANCHOR)* 2004. 2004.

[54] Cummings, Clifford E. “Simulation and synthesis techniques for asynchronous FIFO design.” In *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*. 2002.

[55] Chakrabarti, Indrajit. “Hardware Description Language.”

[56] Thomas, Donald E., and Philip R. Moorby. “The Verilog hardware description language”. Vol. 2. Accord Station, Hingham, MA,,, USA: Kluwer Academic Publishers, 2002.

[57] Palnitkar, Samir. “Verilog HDL: a guide to digital design and synthesis”. Vol. 1. Prentice Hall Professional, 2003.

[58] Hübner, Michael, and Jürgen Becker. “Exploiting dynamic and partial reconfiguration for FPGAs: toolflow, architecture and system integration.” In *Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pp. 1-4. ACM, 2006.

[59] Francis, Robert J., Jonathan Rose, and Zvonko G. Vranesic. “Field programmable gate arrays”. Vol. 180. Springer, 1992.

[60] Oldfield, John V., and Richard C. Dorf. “Field-programmable gate arrays”. John Wiley & Sons, 1995.

[61] Trimberger, Stephen, ed. “Field-programmable gate array technology”. Springer, 1994.