MULTI-PURPOSE USB INTERFACE USING FPGA

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

ELECTRICAL ENGINEERING

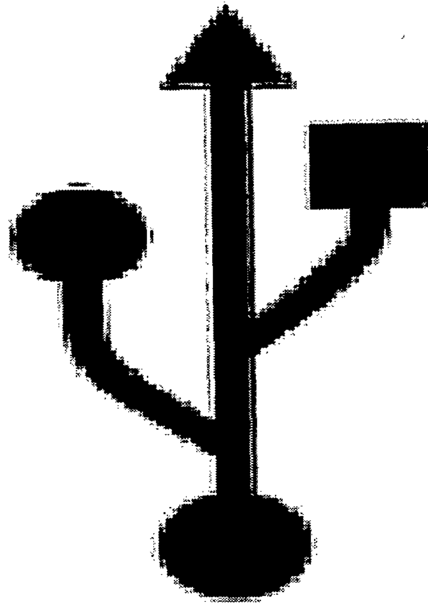
(With Specialization in System Engineering & Operations Research)

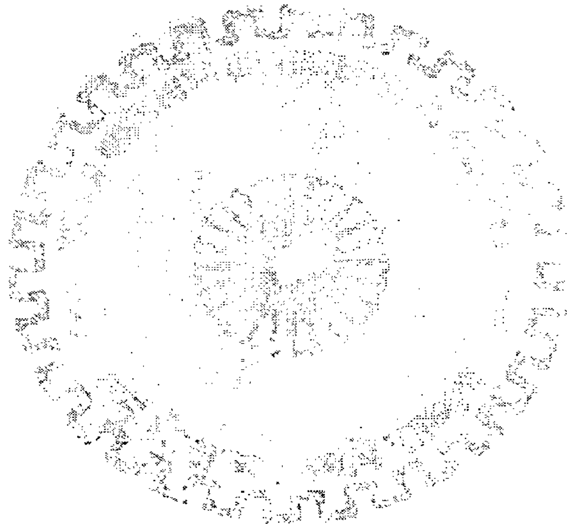
By

VAIBHAV JAIN

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)

JUNE, 2006





Dedicated to

My Parents

CANDIDATE'S DECLARATION

I hereby declare that the work which is being presented in this dissertation entitled, "**MULTI-PURPOSE USB INTERFACE USING FPGA**", submitted in the partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Electrical Engineering**, with specialization in **System Engineering and Operations Research**, I.I.T. Roorkee, India is an authentic record of my own work carried out from July 2005 to June 2006 under the guidance of **Dr. Rajendra Prasad**, Electrical Engineering Department, Indian Institute of Technology, Roorkee, India.

The matter embodied in this dissertation report has not been submitted by me for the award of any other degree or diploma.

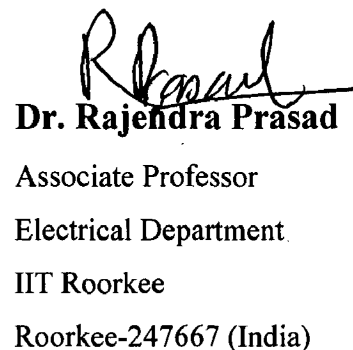
Place: Roorkee

Dated: 19 June 2006

VAIBHAV JAIN

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.



Dr. Rajendra Prasad
Associate Professor
Electrical Department
IIT Roorkee
Roorkee-247667 (India)

// ACKNOWLEDGEMENTS

*From the core of my heart, I express my gratitude to **Dr. Rajendra Prasad**, Department of Electrical Engineering, I.I.T. Roorkee, for giving me his valuable guidance, constant support and belief in me. He gave me the opportunity to work on a challenging topic. I cordially admire his greatness and fatigueless enthusiasm without which it is impossible to me completes this work up to this level. Working under him is an unforgettable experience for me I will remember him with great emotions and respect in my future.*

*Special thanks must be mentioned here for **Prof. M. K. Vasantha**, Department of Electrical Engineering, I.I.T. Roorkee for teaching me the fundamentals of which indirectly helped me to understand the USB system without which this thesis would have been incomplete.*

*I express my thanks to **Dr. (Ms.) Indra Gupta**, in charge Microprocessor and Computer Laboratory for providing me the all the facilities in the lab including a computer and Spartan II FPGA Kit required for this project as and when needed.*

*I am highly grateful to the department of Electrical Engineering, IIT Roorkee for providing me the all kinds of desired facilities for completing my work. My cordial thanks are for **Prof. S.P.Gupta**, Head of Electrical Engineering department and **Prof. H.O. Gupta**, Ex-head of the department, for creating the opportunities and facilities all the time. My respectful thanks are for all the faculty members of the department for the unforgettable help and support either direct or indirect always available to me during the course of the work.*

*I wish to convey my thanks to **Mr. Vishal Saxena** and **Mr. Vijender Singh, Mr. Rahul Dubey** Research Scholars, Department of Electrical Engineering, I.I.T. Roorkee for their help, guidance and support through out the year.*

*How can I forget my batch mates, hostel mates and friends, particularly, **Sanjay, Arun, & Rajesh** for helpful and fruitful discussions on various topics, and for the unforgettable experience of living with them.*

In the end, I would like to appreciate my parents and my brother and sister for their constant faith in me and for feeing confidence in me through their moral support.

Dated: June 2006 VAIBHAV JAIN

LIST OF FIGURES AND TABLES

Figure 3-1 Simple USB Host/Device View [29].
Figure 3-2 USB Hub
Figure 3-3 Bus Topology
Figure 3-4 Full Speed Device Attachment
Figure 3-5 Low Speed Device Attachment
Figure 3-6 NRZI Encoding
Figure 3-7 NRZI Encoding After Bit Stuffing.
Figure 3-8 Bit Stuffing Algorithm.
Figure 4-1 PID Field [29].
Figure 4-2 Address Field [29].
Figure 4-3 Endpoint Field [29].
Figure 4.4 Token Packet
Figure 4.5 Data Packet Format
Figure 4-6 Successful Data Transactions
Figure 4-7 Failed Data Transactions
Figure 4-8 Device Status
Figure 5-1 Logic Block
Figure 5-2 Logic Block Pin Locations
Figure 6-1 Receiver Block Diagram
Figure 6-2 CRC – 5 Algorithms
Table 6-1 Interpretation of PID Values
Figure 6-3 Transmitter Block Diagram
Figure 6-4 USB Interface Block Diagram
Figure 6-5 USB Controller State Machine
Figure 7-1 Sync_Token
Figure 7-2 Pid_Token
Figure 7-3 Final Bit Sequences For OUT Transaction

Figure 7-3 Final Bit Sequences For IN Transaction
Figure 8-1 Successful Bit Sequence For IN Transaction
Figure 8-2 Simulation Result For IN Transaction.
Figure 8-3 Successful Bit Sequence For OUT Transaction
Figure 8-4 Simulation Result For OUT Transaction.
Figure AA-1 Timing Simulation Waveform
Figure AB-1 RTL View Part -1
Figure AB-2 RTL View Part -2
Table AC-1 Types of PIDs and Their Description.
Table AC-2 Device Responses To IN Transactions
Table AC-3 Host Responses To IN Transactions
Table AC–4 Device Responses to an OUT Transaction
Table AD-1 Device Status
Figure AE-1 Basic Spartan-Ii Family FPGA Block Diagram
Figure AE-2 Spartan-Ii Input/Output Block (IOB)
Figure AE-3 Spartan-Ii CLB Slice (Two Identical Slices In Each CLB)

ABBREVIATIONS

ACK	Handshake packet indicating a positive acknowledgment
Bandwidth	The amount of data transmitted per unit of time
Bit	A unit of information used by digital computers
Bit Stuffing	Insertion of a '0' in the data stream for synchronization
Clint	Software resident on the host which interact with the USB
CRC	CYCLIC REDUNDANCY CHECK
EOF	End of Frame
EOP	End of Packet
FPGA	Field Programmable Gate Array
Floor planning	process of choosing best connectivity in a design
Frame	A series of transaction in 1ms time
Host	The computer where the USB host controller is installed
Hub	USB device to provide additional connection to USB
IEEE	Institute of Electrical and Electronics Engineering
IOB	Input/Output Buffer
IOE	Input/Output Element
IRP	I/O Request Packet

LAB	Logic Array Block
LE	Logic Element (For Altera FPGA)
LUT	Look Up Table
NAK	Handshake packet indicating a negative acknowledgment
NRZI	Not Return To Zero Invert
Packet	A bundle of data organized in a group of information
PID	Packet ID
PLD	Programmable Logic Device
PLL	Phase Locked Loops
SOF	Start of Frame
SOP	Start of Packet
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XST	Xilinx's Synthesis Tool

CONTENTS

COVER PAGE	ii
CANDIDATE'S DECLARATION	iv
ACKNOWLEDGEMENTS	v
LIST OF FIGURES & TABLES	vi
ABBREVIATION	viii
CONTENTS	x
ABSTRACT	xiv
CHAPTER 1: INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 PROBLEM WORKED OUT	1
1.3 SOLUTION ACHIEVED	1
1.4 SCOPE OF THE REPORT	2
1.5 ORGANIZATION OF THESIS	2
CHAPTER 2: INTRODUCTION TO UNIVERSAL SERIAL BUS	4
2.1 INTRODUCTION	4
2.2 NEED OF USB	4
2.3 WHY USB	5
2.4 GOALS OF THE USB	6
CHAPTER 3: THE USB SYSTEM	7
3.1 DATA FLOW MODEL	7
3.2 USB HARDWARE AND SOFTWARE	7
3.2.1 USB DEVICE DRIVER	8
3.2.2 USB DRIVER	8
3.2.3 USB HOST CONTROLLER DRIVER	8

3.2.4 USB HOST CONTROLLER/ROOT HUB 8
3.2.5 USB HUB 9
3.2.6 USB DEVICE 10
3.3 BUS TOPOLOGY 10
3.4 TRANSFER TYPES 11
3.4.1 CONTROL TRANSFERS 11
3.4.2 BULK TRANSFERS 12
3.4.3 INTERRUPT TRANSFERS 12
3.4.4 ISOCHRONOUS TRANSFERS 12
3.5 SIGNALING ENVIRONMENT 13
3.5.1 DIFFERENTIAL PAIR SIGNALING 13
3.5.2 DIFFERENTIAL DRIVERS 13
3.6 DEVICE SPEED IDENTIFICATION 13
3.7 DATA ENCODING AND DECODING 14
3.8 BIT STUFFING 15

CHAPTER 4: THE USB TRANSFER PROTOCOL 18
4.1 SYNC FIELD 18
4.2 FORMATS OF PACKET FIELD 18
4.2.1 PACKET IDENTIFIER FIELD 18
4.2.2 ADDRESS FIELDS 19
4.3 DATA FIELD 21
4.4 CYCLIC REDUNDANCY CHECKS 21
4.4.1 GENERATOR POLYNOMIAL 22
4.4.2 CRC'S FOR TOKEN 22
4.4.3 CRC'S FOR DATA 22
4.5 PACKET FORMATS 23
4.5.1 TOKEN PACKETS 23
4.5.2 DATA PACKETS 24

4.5.3 HANDSHAKE PACKETS 24
4.6 DATA TOGGLE ERRORS 25
4.6.1 SUCCESSFUL DATA TRANSACTIONS 25
4.6.2 DATA CORRUPTED OR NOT ACCEPTED 26
4.7 USB DEVICE STATES 27

CHAPTER 5: TECHNOLOGIES USED 29
5.1 FPGAS 29
5.1.1 APPLICATIONS 30
5.1.2 ARCHITECTURE 30
5.2 VHDL 31
5.3 DESIGN SYNTHESIS AND PROGRAMMING OF THE FPGA 31
5.4 INTRODUCTION TO THE SPARTAN II DEVELOPMENT BOARD 32

CHAPTER 6: BLOCK DIAGRAM AND STATE MACHINE 33
6.1 COMPONENTS 33
6.2 RECEIVER 33
6.3 TRANSMITTER 39
6.4 USB INTERFACE 43
6.4.1 TRANSACTION DECIDER 43
6.4.2 CONTROLLER 44

CHAPTER 7: IMPLEMETATION 48
7.1 VHDL IMPLEMENTATION 48
7.2 BIT SEQUENCE 48
7.3 FPGA IMPLEMMTATION 53

CHAPTER 8: **RESULTS AND DISCUSSION** **54**

8.1 TESTING PROCEDURE **54**

8.2 SIMULATION RESULTS **54**

8.2.1 IN TRANSACTION **55**

8.2.2 OUT TRANSACTION **57**

8.3 SYNTHESIS RESULTS **59**

8.4 DEVICE UTILIZATION SUMMARY **59**

CHAPTER 9: **CONCLUSION AND FUTURE SCOPE** **60**

9.1 CONCLUSIONS **60**

9.2 FUTURE SCOPE OF WORK **61**

REFERENCES **62**

APPENDIX A TIMING SIMULATION WAVEFORMS **67**

APPENDIX B RTL VIEW **69**

APPENDIX C PID TYPE AND HANDSHAKE RESPONSE **72**

APPENDIX D DEVICE STATUS **75**

APPENDIX E SPARTAN-II 2.5V FPGA FAMILY **77**

ABSTRACT

The USB stands for Universal Serial Bus it is a very popular interface in recent computer systems. USB replaces legacy interface, such as serial port, parallel port, and so on. Every recent PC and Macintosh computer includes USB ports that can connect to standard peripherals such as keyboards, mice, scanners, cameras, printers, and drives as well as custom hardware for just about any purpose. The aim of this work is to develop a USB interface that can be used to transfer data. The interface is the bridge which allows data transfer from one side to other one. Presented work is exploring an application area of FPGA to develop an interface design on FPGA. The interface is multi-purpose so that the data can be traveled in both the direction. This work describes the implementation of USB 1.1 low speed interface in FPGA chip, using VHDL programming at Xilinx ISE 7.1i platform supported by Aldec Active HDL simulation environment. The design is targeted to make a feel of an original USB device which is sending or receiving data from USB port. the design contains the data lose protection and synchronization mechanism, in USB the data travels in the form of packets , there are three type of packets Token, Data and Handshake , the interface takes care of all the three packets and there inner contents. Due to the limit of clock frequency the design can show only low speed transaction so only USB 1.1specifications are discussed here.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Nowadays extensive usage of FPGAs implies the need for communication that is faster than a simple serial line. The designer has to transmit large packets of data between his FPGA application and the PC. Hence the solution to the problem is USB interface; the universal serial bus (USB) is an expansion scheme that replaces the serial cards in a PC. It provides a high-speed serial connection to the Pc's bus and as many as 127 devices can be connected to the same bus at a time, The USB interface is very simple for using and universal usage, which comes from its widespread presence in today's PCs. The designer has to install the software tools that simplify the implementation of the USB transactions. These tools include a USB driver implementing the basic transfer methods, and a simple application demonstrating the way of transaction with the driver in the PC side. Well the FPGA portion consists of the USB differential driver and the USB interface design .The goal of this work, is to develop a universal software tools for simple usage of the USB 1.1 transactions. This design also having features for data protection by using FIFO and synchronization of the data transfer and the low power consumption by using PLL.

1.2 PROBLEM WORKED OUT

The problem statement in front of me through this thesis was to design an interface which allows the user to communicate through the USB port. And the design should be multi-purpose, that the user can transfer the direction data is both the direction. It should consume less power and there should be some mechanism to prevent the data loses. Data transfer should be exactly identical like in USB device or the data transfer should follow the data transfer protocol.

1.3 SOLUTION ACHIEVED

The USB interface has been designed for IN and OUT transactions. It allows the user to communicate through the USB port. For the low power consideration PLL is employed. This PLL also help the interface in maintaining the synchronization. For

preventing the data loss the FIFO is been implemented. Data transfer is exactly identical like in USB device and follow the data transfer protocol. Well the design only supports the low speed data transfer so full and high speed data transfer can not be seen through this design, hence why the report contains the specifications and data of the USB 1.1 only because the design is compatible to USB 1.1 only. Well with slight modifications the design will be able to support USB 2.0.

1.4 SCOPE OF THE REPORT

Scope of this report is limited to develop an interface design for USB 1.1 data transfer and implement it on FPGA. My interface design support two type of transactions and follow the USB transfer protocol.

In this report more stress is given on the simulation rather that the demonstration after implementation, although the design has been implemented on FPGA successfully because the clock frequency is so high that user can not see the data transfer. In simulation user can see that state of the controller at ant time. The simulation and synthesis results are placed in the report.

The design is limited to the USB 1.1 as described above hence on USB 1.1 is disused in this report.

1.5 ORGANIZATION OF THESIS

Chapter 2: - This chapter provides the necessary information to the USB, its need and benefits over the existing peripherals.

Chapter 3: - This chapter provides the basics of the USB system so that the reader can understand the USB1.1 data transfer protocol.

Chapter 4: - This chapter provides the detailed information of USB transfer protocol and its transfer mechanism that has been implemented on FPGA in this thesis.

Chapter 5: - This chapter provides the information regarding the tools and technology used to develop the interface design and the reasons for choosing them.

Chapter 6: - This chapter provides the strategy which has been used to develop the design with the help of the block diagram and the state machine and their explanation.

Chapter 7: - This chapter provides the method by which the data transfer can be seen or the testing procedure of the design, this chapter mainly stressed on simulation.

Chapter 8: - This chapter provides the various results and discussion on the results and the performance of the design.

Chapter 9: - This chapter concludes the whole work and gives some suggestion for its future developments.

CHAPTER 2

INTRODUCTION TO UNIVERSAL SERIAL BUS

This chapter provides the basic and necessary information about the USB, also gives brief description of the need of USB & why USB is the solution of the end user problems regarding peripheral devices and the serial ports.

2.1 INTRODUCTION

In today's era where personal computers are used in lots of applications, many new devices have come up since IBM's original two serial port personal computer some twenty tears ago. Scanners, portable hard drives, Zip drives, and force feedback joystick are just a few examples of device are present on the desktop. Although attempts have been made to present four to eight serial ports on a single PC, there was no real standard that gained huge acknowledgment. Therefore, designers began work on specification for a new interconnecting solution. Well USB is the solution to these all problems.

Universal Serial Bus (USB) is a solution touted by seven leaders of the PC and telecom industry: Compaq, DCE, IBM, Intel, Microsoft, NCE and Northern Telecom (now Nortel Networks). USB boasts a data rate of 12 Mbps (mega byte per second) (for USB 1.1) and allows you to connect up to 127 devices to your PC. It supports modems, keyboard, mice, CD ROM drives, joystick, tape, floppy, hard drives, scanners and printers. In addition, a new stream of peripherals such as telephone, digital speakers, digital snapshot and motion cameras are to take advantage of this existing and versatile new interface for communication with PC.

2.2 NEED OF USB

There are several problems or shortcomings present with the existing peripherals, some of them are listed below.

- **Interrupts:** - Perhaps the most critical system resource problem revolves around the allocation of the interrupts by myriad of devices; hence interrupt shortage can become a major problem.

- **Non Shareable Interface:** - Standard PC interface support the attachment of single device, hence the flexibility of such connection is reduced.

- **Cable Crazed:** - Dedicated cables are required for the mouse, keyboard etc. The variety of different connectors is required to connect particular peripheral.

- **No Hot Attachment of Peripheral:** - ever forgot to plug in your mouse before powering up your system? After the boot process your mouse won't work.

- **Cost:** - the cost of implementing system and peripheral devices based on the original PC design is fairly expensive.

USB is the solution of all the problems described.

2.3 WHY USB

A question comes in mind that why I am switching to USB? Well there is something which motivates user to switch to the USB. The motivation for the Universal Serial Bus (USB) comes from three interrelated considerations:

- **Connection of the PC to the Telephone:** - It is well understood that the merge of computing and communication will be the basis for the next generation of productivity applications. The movement of machine-oriented and human-oriented data types from one location or environment to another depends on ubiquitous and cheap connectivity, and fast data transfer also. The USB provides an easy link that can be used across a wide range of PC-to-telephone interconnects[9].

- **Ease-of-Use:** - The lack of flexibility in reconfiguring the PC has been acknowledged as the week point to its further deployment. The combination of user-friendly graphical interfaces and the hardware and software mechanisms associated with new-generation bus architectures have made

computers less hostile and easier to reconfigure. Moreover, from the end user's point of view, the PC's I/O interfaces, such as serial/parallel ports, keyboard/mouse/joystick interfaces, etc., do not have the attributes of plug-and-play.

- **Port Expansion:** - The addition of external peripherals continues to be constrained by port availability. The lack of a bidirectional, low-cost, low-to-mid speed peripheral bus has held back the creative reproduction of peripherals such as telephone/fax/modem adapters, answering machines, scanners, keyboards, mice, etc. Existing interconnects are optimized for one or two point products. As each new function or capability is added to the PC, a new interface has been defined to address this need. The USB is the solution to connectivity for the PC architecture. It is a fast, bi-directional, isochronous, low-cost, dynamically attachable serial interface that is consistent with the requirements of the PC Platform of today and tomorrow[9].

2.4 GOALS OF THE USB

The USB is designed to achieve some goals regarding its performance in comparison of existing peripherals; it should sole all the problems & limitations associated with the existing peripherals. The following criteria were applied in defining the architecture for the USB

- Ease-of-use for PC peripheral expansion
- Low-cost solution that supports transfer rates up to 12Mb/s
- Full support for real-time data for voice, audio, and compressed video
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
- Integration in commodity device technology.
- Provision of a standard interface capable of quick diffusion into product
- Enablement of new classes of devices that augment the PC's capability[13].

CHAPTER 3

THE USB SYSTEM

In this chapter the working of the USB hardware and software in the USB system is explained and also the signaling environment.

3.1 DATA FLOW MODEL

The USB provides communication channel between a host and attached USB devices. However, the general view to an end user sees of attaching one or more USB devices to a host, as in Figure 3-1, is in fact a little more difficult to implement than is indicated by the Figure. There are several main concepts, constraints, restrictions and features must be supported to provide the end user with the reliable and perfect operation demanded. The USB is presented in a layered fashion to ease explanation and allow developers of particular USB products to focus on the details related to their product.

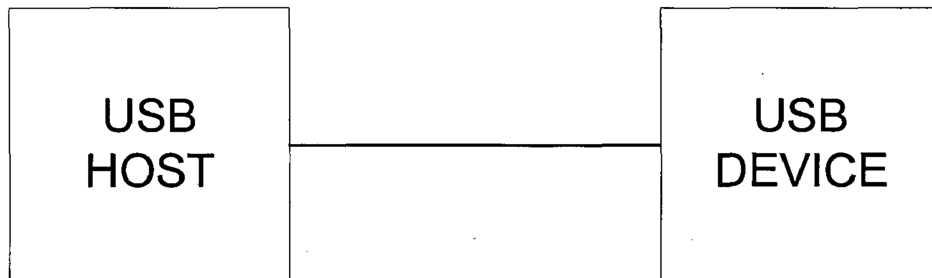


Figure 3 -1 Simple USB Host/Device View

3.2 USB HARDWARE AND SOFWARE

The USB system composed of USB hardware and USB software

USB software

- USB host controller driver
- USB device driver
- USB driver

USB hardware

- USB devices

- USB hot controller/root hub
- USB hub

3.2.1 USB DEVICE DRIVER: - Software that executes on the host, corresponding to a USB device. This client software is typically supplied with the operating system or provided along with the USB device. It issues request to the USB driver via IO request packet (IRPs). These IRPs initiate an interrupt transfer by establishing an IRP and supplying memory buffer into which data will be returned by the USB device.

3.2.2 USB DRIVER: - It knows the device characteristics and knows how to communicate with the device via USB eg. Some device require a specific amount of throughput during each frame, while others may only require access every nth frame. When an IRP received from the USB Clint driver, the USB driver organize the request into individual transaction that will be executed during the series of 1 ms frames. The USB driver sets up the transaction based on the knowledge of USB device requirements[13].

3.2.3 HOST CONTROLLER DRIVER (HCD): - It schedules the transaction to be broadcast over the USB by building series of transactions lists. Each list consists of pending transaction targeted for one or more USB devices attached to the hub. This list defines the sequence of transactions to be executed during the 1 ms frame. The actual scheduling depends upon the no. of factors like type of transaction, transfer requirements and transaction traffic of the other USB devices.

3.2.4 USB HOST CONTROLLER/ ROOT HUB (HOST SIDE BUS INTERFACE): - All communication on USB originates at the host under the software control the hardware consist of the host controller and the root hub.

- **USB HOST CONTROLLER**

The host is responsible for the following:

- Detecting the attachment and removal of USB devices
- Managing control flow between the host and USB devices

- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Providing power to attached USB devices.

The USB System Software on the host manages interactions between USB devices and host-based device software.

The USB host occupies a unique position as the coordinating entity for the USB. In addition to its special physical position, the host has specific responsibilities with regard to the USB and its attached devices. The host controls all access to the USB. A USB device gains access to the bus only by being granted access by the host. The host is also responsible for monitoring the topology of the USB.

- **THE USB ROOT HUB**

The root hub provides the common points for the USB devices and performs the following key operation:

- Controls the power to its USB ports
- Enable and disable ports
- Recognize device attached to each port
- Set and report status events associated with each port

It consists a hub controller and a repeater. The hub controller responds the accesses made to it, and the repeater forwards transmissions to from the host controller.

3.2.5 USB HUB

In addition to the root hub the USB system supports additional hub that permits extension of the USB devices. USB hub can be integrated into device such as keyboard. , hubs may be self powered or bus powered, bus powered hubs are limited by the amount of power available from the bus and can therefore support a maximum of four USB ports.[28]

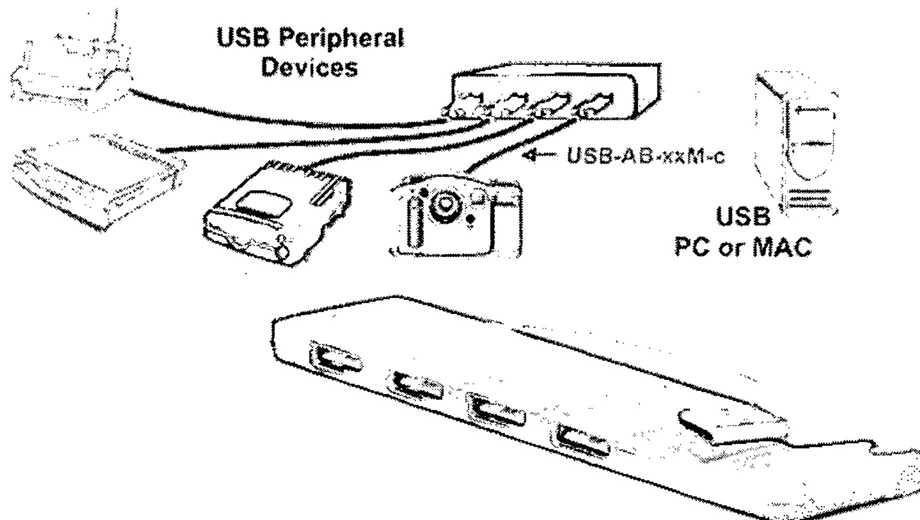


Figure 3.2 USB Hub

3.2.6 USB DEVICES

USB devices contain descriptor that specify a given device attribute and characteristics. There are mainly 2 types of devices

- High speed devices: - high speed devices see all the transaction broadcasted over the USB and can implemented as full feature device , the device accept communicate at a rate of 12 Mbps.
- Low speed device: - these devices are limited to the transaction which follow a special permeable packet , low speed hub port remains disable during full speed transaction permeable packet specifies that the next transaction will be a low speed one (1.5 mbps)[15].

3.3 BUS TOPOLOGY

Devices on the USB are physically connected to the host via a tiered star topology, as illustrated in Figure 3-4. USB ports points are provided by a hub. A host includes an embedded hub within itself called the root hub. The host provides one or more attachment points via the root hub. USB devices that provide additional functionality to the host are known as devices.

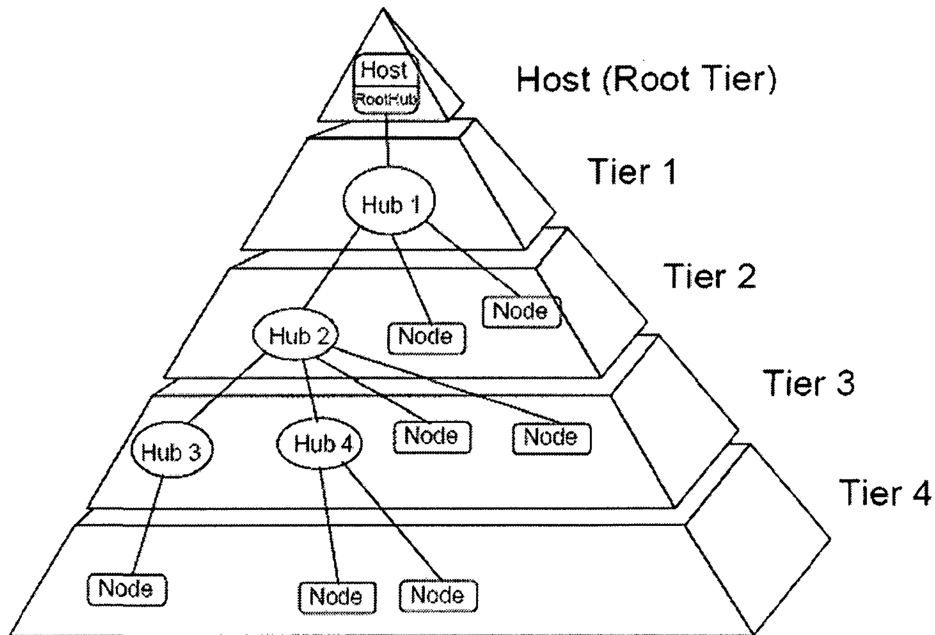


Figure 3 - 3 Bus Topology

As can be seen that a hub is at the center of each star and the wire segment is a point-to-point connection between the host and a hub or device, or a hub connected to another hub or device. Multiple devices may be packaged together so that they appear to be a single physical device. For example, a keyboard and a trackball might be combined in a single package. In inside view of the package, the individual devices are attached to a hub and it is the internal hub which is connected to the USB. When multiple devices are combined with a hub in a single package, they are referred to as a compound device. Figure 3-5 illustrates a compound device.

3.4 TRANSFER TYPES

The USB supports functional data and control exchange between the USB host and a USB device as a set of either unidirectional or bi-directional pipes. USB data transfers take place between host software and a particular endpoint on a USB device. The USB architecture supports four basic types of data transfers:

3.4.1 CONTROL TRANSFERS

To configure devices control data is used by the USB System Software when they are first attached. Other driver software can choose to use control transfers in

implementation-specific ways. Data delivery is lossless. Hence no error checking is done.

3.4.2 BULK TRANSFERS

Bulk data specially used of larger amounts of data, such as used for printers or scanners. Bulk data is sequential. Reliable exchange of data is ensured at the hardware level by using error detection in hardware by using CRC method. Also, the bandwidth taken up by bulk data can vary, depending on other bus activities. Its bandwidth allocation priority is lowest.

3.4.3 INTERRUPT TRANSFERS

This is limited-latency transfer to or from a device. Such data may be presented for transfer by a device or to the device at any time and is delivered by the USB at a same rate no specified by the device. Interrupt data must consists of event notification, characters, or coordinates that are organized as one or more bytes, hence the bandwidth allocation priority is highest for this. Data may have response time bounds that the USB must support; this is called bus time out.

3.4.4 ISOCHRONOUS TRANSFERS

Isochronous data is continuous and real-time in various fields like creation, delivery, and consumption. Timing-related information is implied by the steady rate at which isochronous data is received and transferred. Isochronous data must be delivered at the rate received to maintain its timing, hence it can be said that that isochronous data transfer, transfers the data at a constant rate, it is also be sensitive to delivery delays. For isochronous pipes, the bandwidth required is typically based upon the sampling characteristics of the associated function. A typical example of isochronous data is voice, because in voice transmission the constancy of the transfer speed is must. If the delivery rate of these data streams is not maintained, data loss in the data stream will occur due to buffer or frame under runs or overruns; hence error detection is not possible in this case[13].

3.5 SIGNALING ENVIRONMENT

The signaling specification for the USB is described in the following subsections.

3.5.1 DIFFERENTIAL PAIR SIGNALING

USB employs differential pair signaling to reduce the signal noise. Differential drivers and receivers are used to reduce the source of signal noise listed below;

- Amplifier noise: - the noise introduced when both the driver and receiver of the signal amplify a signal.
- Cable noise: - the noise picked up by the cable due to electromagnetic fields.

3.5.2 DIFFERENTIAL DRIVERS

The USB uses a differential output driver to drive the USB data signal onto the USB cable. It employs inverting and non inverting buffers. The input is applied to both the buffers yielding two outputs D+, D-The static output swing of the driver in its low state must be below VOL (max) of 0.3V with a 1.5k load to 3.6V and in its high state must be above the VOH (min) of 2.8V with a 15k load to ground. The output swings between the differential high and low state must be well-balanced to minimize signal skew. Slew rate control on the driver is required to minimize the radiated noise and cross talk.

3.6 DEVICE SPEED IDENTIFICATION

The USB is terminated at the hub and function ends as shown in Figure 3-4 and Figure 3 -5. Full-speed and low-speed devices are differentiated by the position of the pull-up resistor on the downstream end of the cable:

- Full-speed devices are terminated as shown in Figure 3-4 with the pull-up resistor on the D+ line.
- Low-speed devices are terminated as shown in Figure 3-5 with the pull-up resistor on the D- line.
- The pull-down terminators on downstream ports are resistors of 15K+-5% connected to ground.

  A USB hub detects that a device has been attached to one of its ports by monitoring the differential data lines after cable power has been applied to the ports , when no device is attached to port , pull – down resistor on the D+ and

D- lines ensure that both data lines are near ground . when device is attached the current flows across the voltage divider created by hub's pull down register on either D+ or D- . Since the pull down resistor value is 15K and device's pull – up resistor value is 1.5K a data line will rise to approximately 90 % of Vcc. When the hub detects that one of that data lines approaches Vcc while the other remains near ground, it knows that a device has been attached. Note that full speed devices have a pull – up on D+ and low speed devices have that pull – up on D_, permitting the identification of device speed [18].

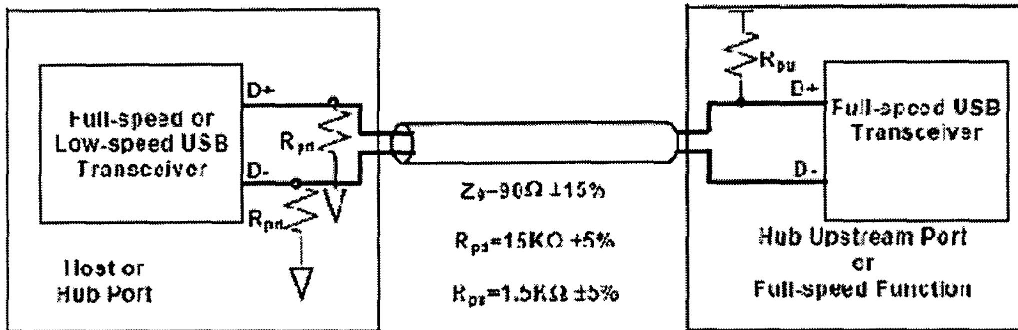


Figure 3-4 Full Speed Device Attachment

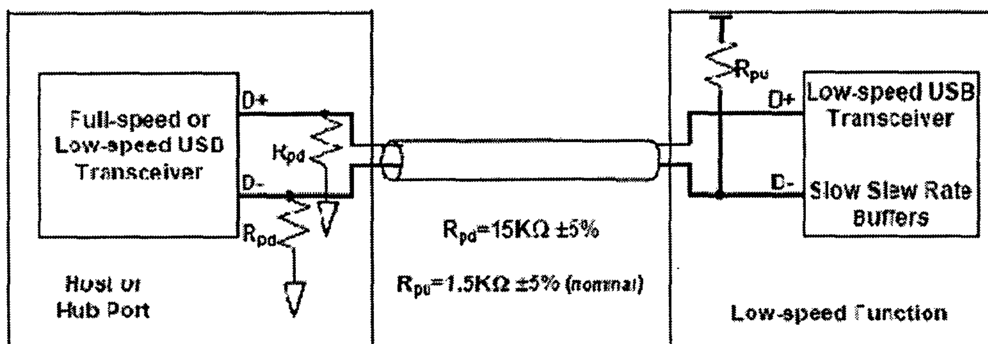


Figure 3-5 Low Speed Device Attachment

3.7 DATA ENCODING AND DECODING

The USB employs NRZI data encoding when transmitting packets. In NRZI encoding, a "1" is represented by no change in level and a "0" is represented by a

change in level. Figure 3 - 6 shows a data stream and the NRZI equivalent. The high level represents the J state on the data lines in this and subsequent figures showing NRZI encoding. A string of zeros causes the NRZI data to toggle each bit time. A string of ones causes long periods with no transitions in the data.

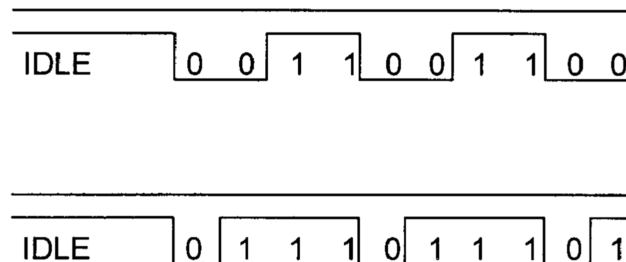


Figure 3-6 NRZI Encoding

3.8 BIT STUFFING

To ensure the proper signal transmission, bit stuffing is employed by the transmitting device when sending a packet. A zero is inserted after every six consecutive ones in the data stream before the data is NRZI encoded, to ensure at least a transition in the NRZI data stream. This gives the receiver logic a data transition at least once every seven bit times to guarantee the data and clock lock. Hence this all is done in order to clock lock. Bit stuffing by the transmitter is always enforced, without exception. If required by the bit stuffing rules, a zero bit will be inserted even if it is the last bit before the end-of-packet (EOP) signal. The receiver decodes the NRZI data, recognize the stuffed bits, and discard them before placing it to the o/p. If the receiver sees seven consecutive ones anywhere in the packet, then a bit stuffing error has occurred and the packet should be ignored.

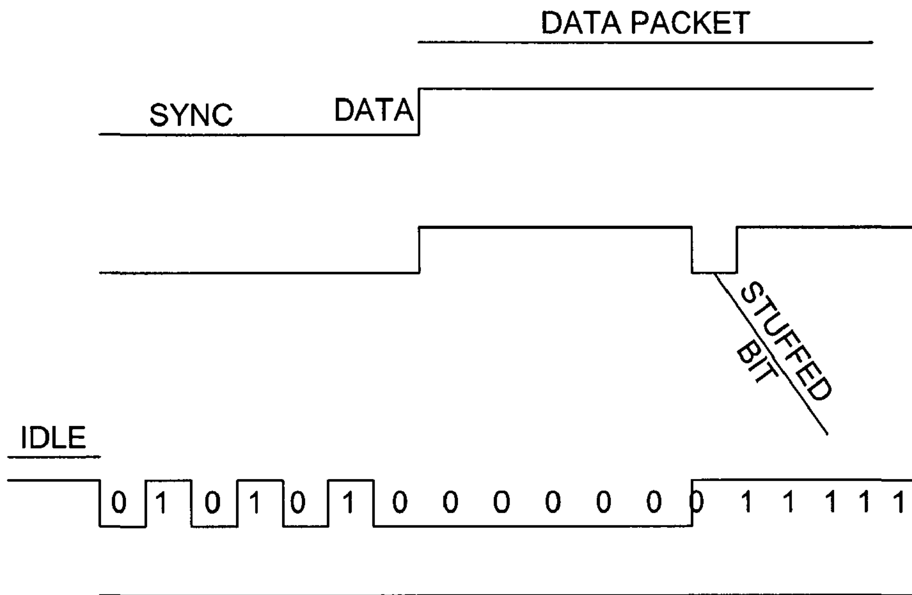


Figure 3-7 NRZI Encoding After Bit Stuffing.

A proper algorithm has to be followed to provide the bit stuffing in the USB interface. The flowchart of that algorithm is shown in the Figure 3- 8. In this algorithm there is counter which is initially set to zero and if the incoming bit is '1' then the counter is incremented else it is reset to zero , in this process if the counter reaches to 6 then a '0' is inserted in the data bit stream and the counter is reset to '0' . This process continues until the packet transfer is complete.

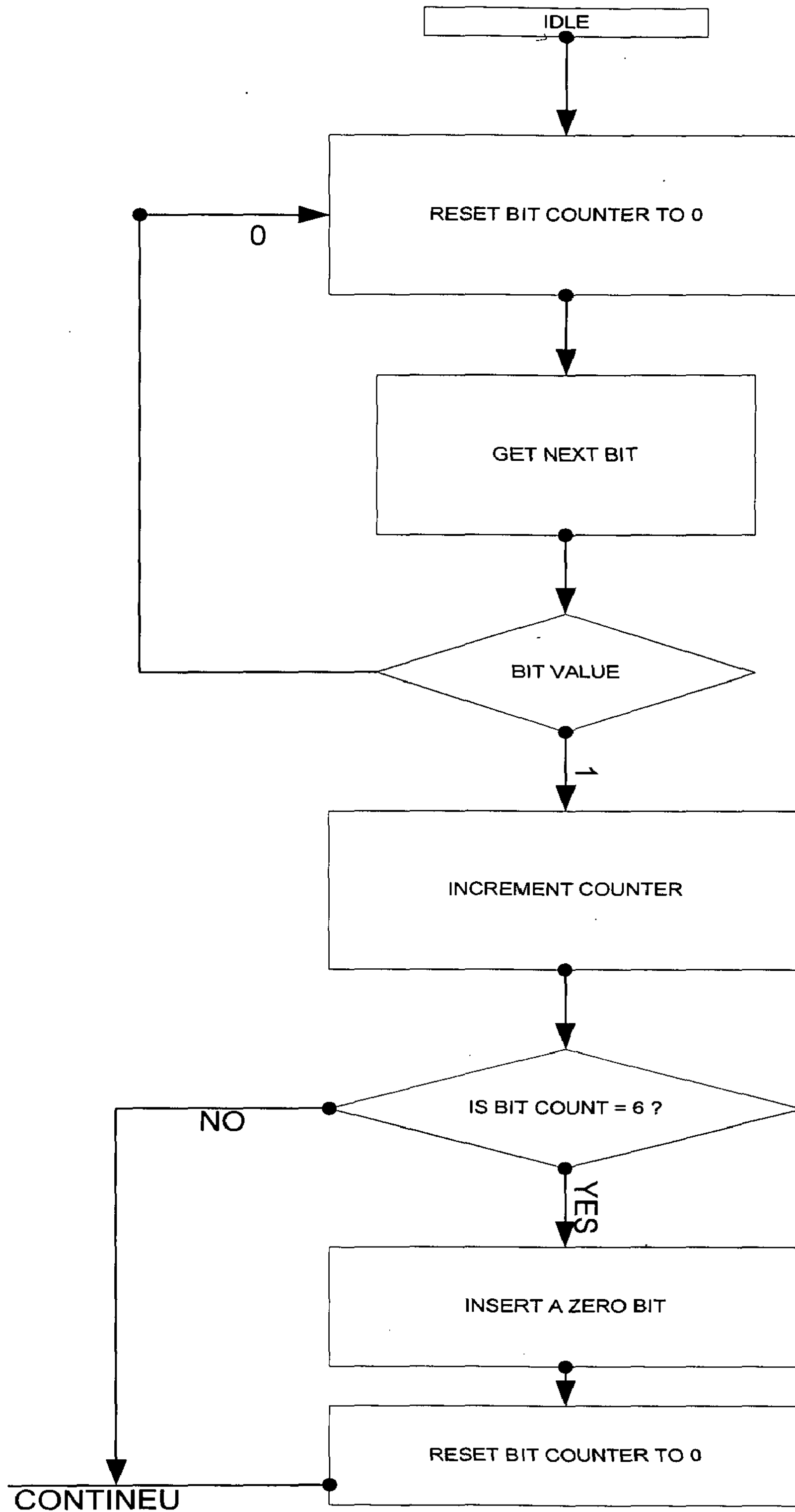


Figure 3-8 Bit Stuffing Algorithm.

CHAPTER 4
THE USB TRANSFER PROTOCOL

This chapter provides the information about the data transfer mechanism in USB. Each packet is started with a sync field so starting the discussion from the sync field itself.

4.1 SYNC FIELD

All packets begin with a synchronization (SYNC) field, which is a coded sequence that generates a maximum edge transition density. As I know that edge. The SYNC field appears on the bus as IDLE followed by the binary string "01010100,"here it is clear that in the sync field there is transition in each bit position in its NRZI encoding. It is used by the input circuitry to align incoming data with the local clock and is defined to be eight bits in length. The last two bits in the SYNC field are a symbol that is used to identify the end of the SYNC field and, by inference, the start of the PID.

4.2 FORMATS OF PACKET FIELD

Field formats for the token, data, and handshake packets are described here. All packets have distinct Start- and End-of-Packet delimiters. The Start of- Packet (SOP) delimiter is part of the SYNC field.

4.2.1 PACKET IDENTIFIER FIELD

A packet identifier (PID) immediately follows the SYNC field in every USB packet. A PID consists of a four-bit packet type field followed by a four-bit check field as shown in Figure 4-1. The PID indicates the type of packet and, the format of the packet and the type of error detection applied to the packet. The four-bit check field of the PID is used to ensure reliable decoding of the PID. The PID check field is generated by performing a one's complement of the packet type field. A PID error exists if the four PID check bits are not complements of their respective packet identifier bits.

PID 0	PID 1	PID 2	PID 3	$\overline{\text{PID 0}}$	$\overline{\text{PID 1}}$	$\overline{\text{PID 2}}$	$\overline{\text{PID 3}}$

Figure 4-1 PID Field

The PID must be completely decoded by the host and all functions .Any PID received with a failed check field or which decodes to a non-defined value is assumed to be corrupted and it, as well as the remainder of the packet, is ignored by the packet receiver. If a function receives an otherwise valid PID for a transaction type or direction that it does not support, the function must not respond. For example, an IN-only endpoint must ignore an OUT token. PID types.

4.2.2 ADDRESS FIELDS

The fields by the help of which the device endpoints are addressed are called address fields. There are two main address fields

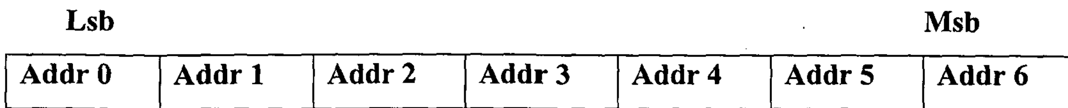
- The device address field
- The endpoint field.

A device needs to fully decode both address and endpoint fields. Address or endpoint aliasing is not permitted, and a mismatch on either field must cause the token to be ignored.

➤ Address Field

The device address field specifies the device, via its address, weather it is source or destination of a data packet, it depends upon the value of the token PID. As shown in Figure 4-2, a total of 128 addresses are specified as ADDR<6:0>. The ADDR field is specified for IN, SETUP, and OUT tokens. Upon reset and power-

up, a function's address defaults to a value of zero. Device address zero is reserved as the default address and may not be assigned to any other use.

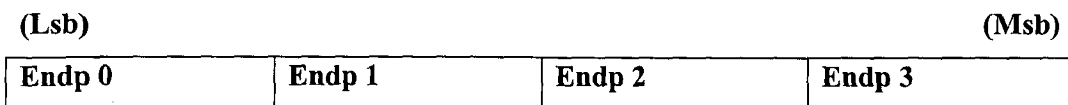


Lsb						Msb
Addr 0	Addr 1	Addr 2	Addr 3	Addr 4	Addr 5	Addr 6

Figure 4-2 Address Field

➢ Endpoint Field

The end point field is an additional four-bit (ENDP) field, as shown in Figure 4-3. This permits more flexible addressing of devices in which more than one endpoint registers are required. Endpoint numbers are device-specific except for endpoint address zero. This endpoint field is defined for IN, SETUP, and OUT token PIDs. All devices must support a control pipe at endpoint number zero (the Default Control Pipe), because endpoint 0 is reserved for control transfer. Low-speed devices support a maximum of three pipes per function: a control pipe at endpoint number zero plus two additional pipes (two control pipes, a control pipe and an interrupt endpoint, or two interrupt endpoints). Full-speed functions may support up to the maximum of 16 endpoint numbers of any type.



(Lsb)			(Msb)
Endp 0	Endp 1	Endp 2	Endp 3

Figure 4-3 Endpoint Field

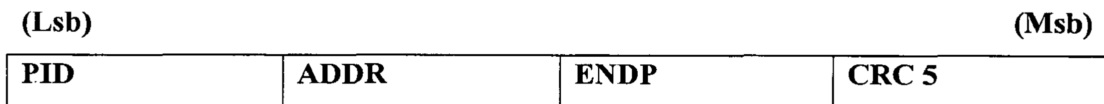
4.5 PACKET FORMATS

This section shows packet formats for token, data, and handshake packets. Fields within a packet are displayed in these figures in the order in which bits are shifted out onto the bus.

4.5.1 TOKEN PACKETS

Figure 4-4 shows the field formats for a token packet. A token consists of a PID, specifying either IN, OUT, or SETUP packet type; and ADDR and ENDP fields. For OUT and SETUP transactions, for identifying the endpoint that will receive the subsequent Data packet the address and endpoint are used. For IN transactions, these fields uniquely identify which endpoint should transmit a Data packet. Most important thing is only the host can issue token packets, because host initiates all transactions. IN PIDs define a Data transaction from a device to the host. OUT PIDs define Data transactions from the host to a device.

(Lsb) **(Msb)**

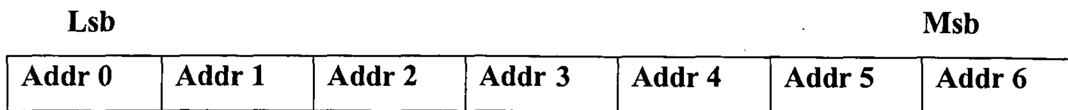


PID	ADDR	ENDP	CRC 5

Figure 4.4 Token Packet

The last part of the token packet is a five-bit CRC that covers the address and endpoint fields as shown above. The CRC does not cover the PID, because error in PID is checked by its own check field. Token packets are delimited by an EOP after three bytes of packet field data. In the end of the packet the EOP field is there this signifies that the packet has been ended. During the EOP the D+ and D- lines of the differential driver are both idle.

up, a function's address defaults to a value of zero. Device address zero is reserved as the default address and may not be assigned to any other use.



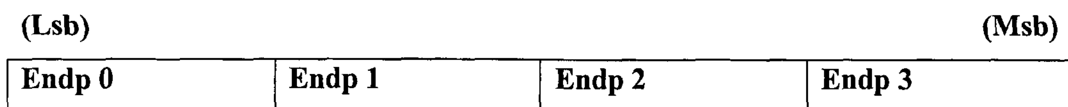
Lsb **Msb**

Addr 0	Addr 1	Addr 2	Addr 3	Addr 4	Addr 5	Addr 6

Figure 4-2 Address Field

➢ Endpoint Field

The end point field is an additional four-bit (ENDP) field, as shown in Figure 4-3. This permits more flexible addressing of devices in which more than one endpoint registers are required. Endpoint numbers are device-specific except for endpoint address zero. This endpoint field is defined for IN, SETUP, and OUT token PIDs. All devices must support a control pipe at endpoint number zero (the Default Control Pipe), because endpoint 0 is reserved for control transfer. Low-speed devices support a maximum of three pipes per function: a control pipe at endpoint number zero plus two additional pipes (two control pipes, a control pipe and an interrupt endpoint, or two interrupt endpoints). Full-speed functions may support up to the maximum of 16 endpoint numbers of any type.



(Lsb) **(Msb)**

Endp 0	Endp 1	Endp 2	Endp 3

Figure 4-3 Endpoint Field

4.3 DATA FIELD

The range of the data field may from zero to 1,023 bytes and must be an integral number of bytes. Data bits within each byte are shifted out LSb first. Data packet size varies with the transfer type.

4.4 CYCLIC REDUNDANCY CHECKS

The basic idea of CRC algorithms is simply to treat the message as an enormous binary number, to divide it by another fixed binary number, and to make the remainder from this division the checksum. Upon receipt of the message, the receiver can perform the same division and compare the remainder with the "checksum" (transmitted remainder). Cyclic redundancy checks (CRCs) are used to protect all non-PID fields in token and data packets. In this context, these fields are considered to be protected fields. The PID is not included in the CRC check of a packet containing a CRC. CRC generation is done before bit stuffing and consequently CRCs are decoded in the receiver after stuffed bits have been removed. Token and data packet CRCs provide 100% coverage errors. A failed CRC is considered to indicate that one or more of the protected fields is corrupted and causes the receiver to ignore those fields, and, in most cases, the entire packet.

Usually, the checksum is then appended to the message and the result transmitted.

At the other end, the receiver can do one of two things:

- Separate the message and checksum. Calculate the checksum for The message (after appending W zeros) and compare the two checksums.

- Checksum the whole lot (without appending zeros) and see if it comes out as zero!

These two options are equivalent. For CRC generation and checking, well the second one is applied in the interface design. The shift registers in the generator and checker are seeded with an all zeros pattern. For each data bit sent or received, the high order

bit of the current remainder is XORed with the data bit and then the remainder is shifted left one bit and the low-order bit set to zero. If the result of that XOR is one, then the remainder is XORed with the generator polynomial. When the last bit of the checked field is sent, the CRC in the generator is inverted and sent to the checker MSb first. When the last bit of the CRC is received by the checker and no errors have occurred, the remainder will be equal to the polynomial residual. A CRC error exists if the computed checksum remainder at the end of a packet reception does not match the residual. Bit stuffing requirements must be met for the CRC, and this includes the need to insert a zero at the end of a CRC if the preceding six bits were all ones [20 – 24].

4.4.1 GENERATOR POLYNOMIAL

For getting the better accuracy of the CRC algorithm the generator polynomial should be chosen wisely, the polynomial should be such that the reminder after the division is unique in every case. So the standard polynomials are used in the interface design to ensure the 100% accuracy of the CRC algorithm.

4.4.2 CRC's FOR TOKEN

A five-bit CRC field is provided for token CRC, it covers the ADDR and ENDP fields of all tokens. The generator polynomial is the standard one for 5 bit CRC is:

G(X) = X5 + X2 + 1

The binary bit pattern that represents this polynomial is 00101B. After appending the checksum to the message polynomial and re calculating CRC it comes out to be "00000B" with the same generator polynomial.

4.4.3 CRC's FOR DATA

The data CRC is a 16-bit polynomial applied over the data field of a data packet. The generating polynomial is:

G(X) = X16 + X15 + X2 + 1

The binary bit pattern that represents this polynomial is 1000000000000101B. After appending the checksum to the message polynomial and re calculating CRC it comes out to be "0000000000000000B" with the same generator polynomial [20 – 24].

4.5 PACKET FORMATS

This section shows packet formats for token, data, and handshake packets. Fields within a packet are displayed in these figures in the order in which bits are shifted out onto the bus.

4.5.1 TOKEN PACKETS

Figure 4-4 shows the field formats for a token packet. A token consists of a PID, specifying either IN, OUT, or SETUP packet type; and ADDR and ENDP fields. For OUT and SETUP transactions, for identifying the endpoint that will receive the subsequent Data packet the address and endpoint are used. For IN transactions, these fields uniquely identify which endpoint should transmit a Data packet. Most important thing is only the host can issue token packets, because host initiates all transactions. IN PIDs define a Data transaction from a device to the host. OUT PIDs define Data transactions from the host to a device.

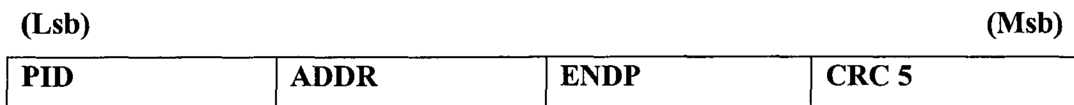


Figure 4.4 Token Packet

The last part of the token packet is a five-bit CRC that covers the address and endpoint fields as shown above. The CRC does not cover the PID, because error in PID is checked by its own check field. Token packets are delimited by an EOP after three bytes of packet field data. In the end of the packet the EOP field is there this signifies that the packet has been ended. During the EOP the D+ and D- lines of the differential driver are both idle.

4.5.2 DATA PACKETS

The next packet after the token is the data packet. A data packet consists of a PID which identifies that it is DATA 0 or DATA 1 packet, a data field containing zero or more bytes of data, and a CRC as shown in Figure 4.5. There are two types of data packets, identified by differing PIDs: DATA0 and DATA1. Two data packet PIDs are defined to support data toggle synchronization. That is defined later in this chapter.

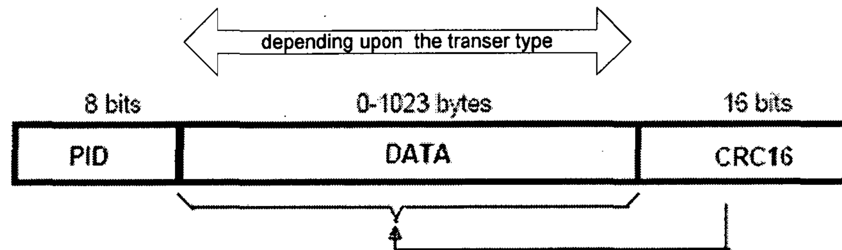


Figure 4.5 Data Packet Format

Data must always be sent in integral numbers of bytes. The data CRC is computed over only the data field in the packet and does not include the PID, which has its own check field. CRC and EOP field are the same as in taken packet.

4.5.3 HANDSHAKE PACKETS

The third packet in the transaction is handshake packet which tells that data been transmitted successfully or not. It consists of only a PID field. It reports the status of the data transmission. Only transaction types that support flow control can return handshakes. Handshakes are always returned in the handshake phase of a transaction and may be returned, instead of data, in the data phase. Handshake packets are delimited by an EOP after one byte of packet field.

There are mainly three types of handshake packets:

- ACK
- NAK
- STALL

ACK indicates that the data packet was received without any errors over the data field and that the data PID was received correctly means CRC check is passed. In short the data transmission is successful. An ACK handshake is applicable only in transactions

in which data has been transmitted and where a handshake is expected. ACK can be returned by the host for IN transactions and by a device for OUT transactions.

NAK indicates that a function was unable to accept data from the host (OUT) or that a function has no data to transmit to the host (IN). In short the data transmission is not successful. NAK can only be returned by device in the data phase of IN transactions or the handshake phase of OUT transactions. The host can never issue NAK. For showing the NAK the host does not issue ant command, it indicate that a device is temporarily unable to transmit or receive data hence the device recognize that host has rejected the data and it retransmit the data.

STALL is returned by a device in response to an IN token or after the data phase of an OUT transaction .STALL indicates that a device is unable to transmit or receive data, or that a control pipe request is not supported. The host is not permitted to return a STALL under any condition.

4.6 DATA TOGGLE ERRORS

The USB provides a method to ensure data sequence synchronization between data transmitter and receiver during various transactions. This strategy provides a means of ensuring that the handshake phase of a transaction was interpreted perfectly by both the transmitter and receiver. Synchronization is achieved by the use of the DATA0 and DATA1 PIDs and separate data toggle sequence bits for the data transmitter and receiver. Receiver sequence bits toggle only when the receiver is able to accept data and receives an error-free data packet with the correct data PID. Transmitter sequence bits toggle only when the data transmitter receives a valid ACK handshake. Hence the data transmitter and receiver must have their sequence bits synchronized at the start of a transaction. The synchronization mechanism used varies with the transaction type.

4.6.1 SUCCESSFUL DATA TRANSACTIONS

Figure 4 - 6 shows the case where two successful transactions have encountered. For the data transmitter, this means that it toggles its sequence bit upon receipt of ACK. Toggle in the receiver bits occurs if it receives a valid data packet and the packet's data PID matches the current value of its sequence bit. Toggle in the transmitter bits

occur after it receives and ACK to a data packet. If data cannot be accepted, the receiver must issue NAK and the sequence bits of both the transmitter and receiver remain as it is. If data can be accepted and the receiver's sequence bit matches the PID sequence bit, then data is accepted and the sequence bit is toggled. That's why it is called data toggle method. Two-phase transactions in which there is no data packet leave the transmitter and receiver sequence bits unchanged.

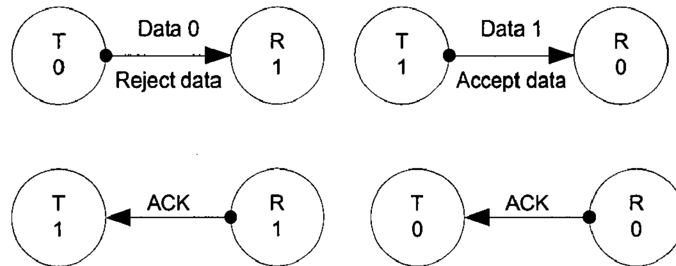


Figure 4-6 Successful data transactions

4.6.2 DATA CORRUPTED OR NOT ACCEPTED

If data cannot be accepted or the received data packet is corrupted, the receiver will issue a NAK or STALL handshake, and the receiver will not toggle its sequence bit. Figure 4 - 7 shows the case where a transaction is NAKed and then retried. Any non-ACK handshake will generate similar behavior. The transmitter, having not received an ACK handshake, will not toggle its sequence bit. Hence, a failed data packet transaction leaves the transmitter's and receiver's sequence bits synchronized and un-toggled. The transaction will then be retried and, if successful, will cause both transmitter and receiver sequence bits to toggle.

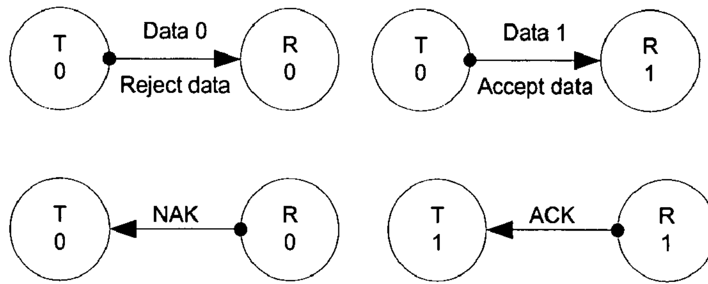


Figure 4-7 Failed data transactions

4.7 USB DEVICE STATES

A USB device has several possible states. Some of these states are visible to the USB and the host, while others are internal to the USB device. This section describes those states. The states of the device can be easily seen in the Figure 4 – 8. In each active state if there is no bus activity for bus time out then the device goes to the suspended state. The figure describes the entire process of device configuration. The conditions and the behavior of the states are self explanatory by their names and their presence in the Figure 4 – 8 so no description is being provided here[28].

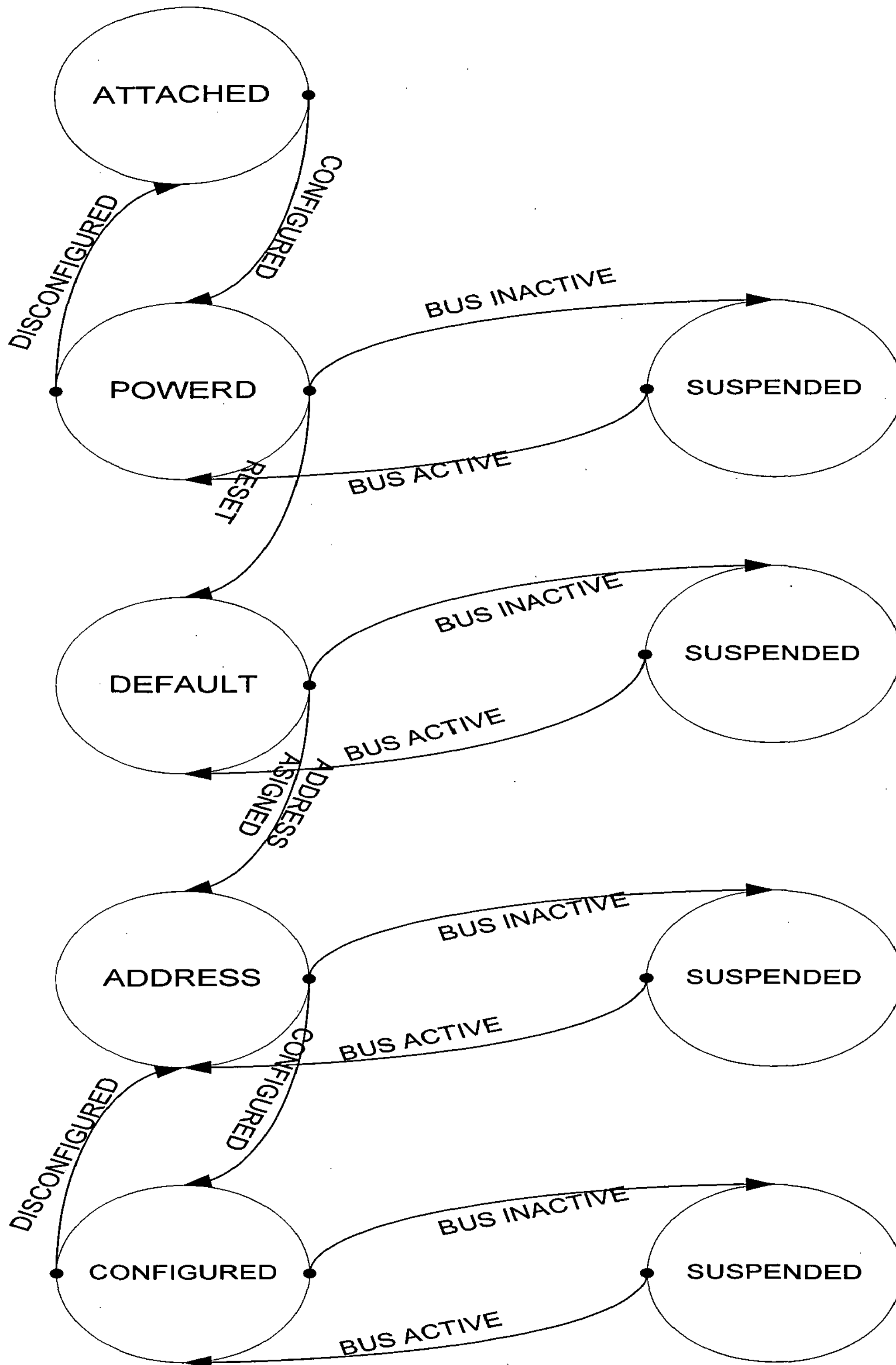


Figure 4-8 Device Status

CHAPTER 5

TECHNOLOGIES USED

The two main technologies which I had used in this work are

- FPGA
- VHDL.

The final USB interface has been developed on FPGA and VHDL is the language in which the design is developed.

5.1 FPGAs

FIELD PROGRAMMABLE GATE ARRAY (FPGA): FPGAs came into existence before 25 .CPLDs and FPGAs include a relatively large number of programmable logic elements. In CPLD logic gate densities range up to tens of thousands of logic gates, while FPGAs typically range from tens of thousands to many million. FPGA is a semiconductor device containing programmable logic components with programmable interconnects. These components are reprogrammable to alter the functionality of basic logic gates such as AND, OR, XOR, NOT and moreover up to the complex combinatorial functions simple mathematical functions. These logic components also include memory elements. The hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected per the requirement of system designer, these logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the term "field programmable") so that the FPGA can perform whatever logical function is needed. Well there is a drawback also that is generally FPGAs are slower than their counterparts' application-specific integrated circuit (ASIC), moreover they can't handle as complex a design, and draw more power. However, there are advantages also such as a shorter time to market, ability to re-program in the field to fix bugs, and lower engineering costs. The basic differences between CPLDs and FPGAs are in their architectures. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers. Hence it is less flexible, but the advantage is more predictable

timing delays and a higher logic-to-interconnect ratio. on the other hand, The FPGA architectures are dominated by interconnect. This makes them far more flexible but also far more complex to design for. Another difference is in most FPGAs the presence of higher-level embedded functions (such as adders and multipliers)[1].

5.1.1 APPLICATIONS

the applications of FPGAs are never ending which includes include DSP, aerospace and defense systems, ASIC prototyping, imaging in medical science , computer vision, speech recognition, bioinformatics, computer hardware emulation and a growing range of other areas. FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs.

5.1.2 ARCHITECTURE

The typical basic architecture consists of an array of configurable logic blocks (CLBs) and routing channels. Multiple I/O pads may fit into the height of one row or the width of one column. Generally, all the routing channels have the same number of wires.

An application circuit must be mapped into an FPGA with adequate resources.

The typical FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop, as shown below[30].

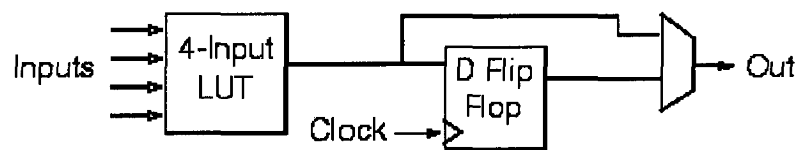


Figure 5 -1 Logic Block

There is only one output, which can be either the registered or the unregistered LUT output. The logic block has four inputs for the LUT and a clock input. The locations of the FPGA logic block pins are shown below.

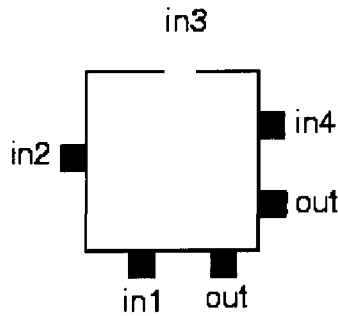


Figure 5 -2 Logic Block Pin Locations

5.2 VHDL

Very high speed integrated circuit (VHSIC) Hardware Description Language (HDL). VHDL is a language for describing digital electronic systems. It comes in the existence from United States Government's Very High Speed Integrated Circuits (VHSIC) program, initiated 25 years ago. At that time there was a need for a standard language for describing the structure and function of integrated circuits (ICs). Hence the VHSIC Hardware Description Language (VHDL) was developed, and subsequently adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) in the US. VHDL can fulfill a number of needs in the design process. Firstly, it allows description of the structure of a design that is how it is decomposed into sub-components, and how those sub- components are mapped. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping [3,4,31].

5.3 DESIGN SYNTHESIS AND PROGRAMMING OF THE FPGA

Once the VHDL code has been compiled it must be synthesized. Synthesis converts the VHDL description into a set of components these components can be assembled in the target FPGA. Xilinx Foundation series is used as synthesizer in this process. After the synthesis stage completes it comes to the implementation stage. The operations that are performed in the implementation tool are to map, place and route and "configure" the design. When the synthesis tool maps the design it takes the synthesized primitives and allocates them to available resources on the target FPGA. Constraints such as pin assignments can be set at this stage that is given by the user.

Once mapping is complete the implementation tool places and routes the design. The timing constraints must be satisfied hence the logic blocks and I/O blocks used to implement the design are chosen in order to meet any timing constraints. The final task implemented by the synthesizing software is to form a binary file that is called .bit file normally, which is used to program the FPGA. The program is finally implemented on FPGA and the user can verify his results now [2].

5.4 INTRODUCTION TO THE SPARTAN II DEVELOPMENT BOARD

The Spartan™-II 2.5V Field-Programmable Gate Array family gives high performance to the end user moreover abundant logic resources, and a rich feature set, all this is available at exceptionally low price. The six-member family offers densities of the system gates ranging from 15,000 to 200,000 .System performance is supported up to 200 MHz. Spartan-II devices deliver more gates, I/Os. It is having included the block RAM (to 56K bits); distributed RAM (to 75,264 bits), 16 selectable I/O standards, and four DLLs. Successive design iterations continue to meet timing requirements. It is a superior alternative to mask-programmed ASICs. The initial cost, lengthy development cycles, and inherent risk of conventional ASICs are avoided in the FPGA. Also the programmability of FPGA permits design upgrades in the field with no hardware replacement necessary which is impossible with ASICs. Details of features and architecture can be found in the Appendix E [30].

CHAPTER 6

BLOCK DIAGRAM AND STATE MACHINE

6.1 COMPONENTS

The USB interface I have designed is divided in 3 main components

- Receiver.
- Transmitter.
- USB interface.

These components work in conjunction for the correct operation of the interface. The main component is USB interface which controls the operation of the receiver and transmitter.

The further sections describe the details of the each component.

6.2 RECEIVER

The start of any transaction starts from the receiver because each transaction started by the host so the receiver receives the bits forms the host and identifies the type of transaction and all the necessary information for the successful transaction , hence you can say the receiver is an information extractor.

The receiver performs the following tasks

- Detecting the start of packet (SOP).
- Decoding the NRZI encoded data from the USB bus.
- Removing stuffed bits.
- Serial to Parallel Conversion.
- 5-bit and 16-bit CRC calculation for error checking.
- Informing controller of type of transaction being conducted.
- Extracting data from the transaction and sending it to the device specific logic.
- Informing device specific logic of the address and endpoint that are the targets of a USB transaction.

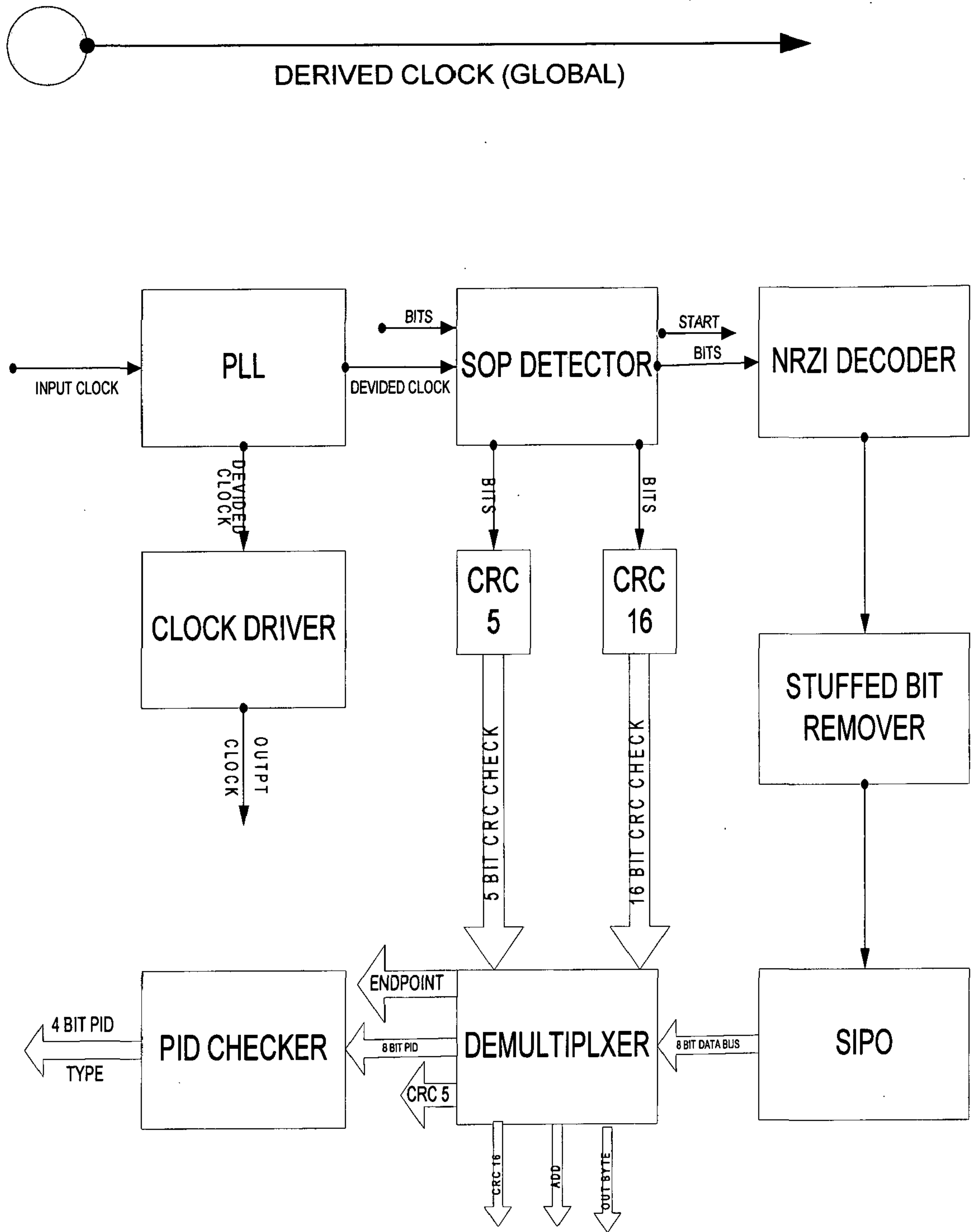


Figure 6 -1 Receiver Block diagram

The details of each block are described below.

➢ PLL

The first and the most important part of the receiver is a PLL (Phase Lock Loop). Basically PLL is used for clock synchronization, in the USB interface the clock signals of the transmitter and receiver may differ substantially so there is a lack of synchronization between transmitter and receiver The USB cable does not provide a separate clock signal. The receiver must somehow synchronize its clock signal with that of the transmitter in order to receive an entire packet without misinterpreting any information. Hence observing when signal transitions occur is employed so that the, the receiver can tell where one bit ends and another begins and synchronize the receiver to the incoming bit stream. This block is responsible for clock recovery. The work of PLL is to sample the incoming data at some sampling rate well the maximum clock frequency of the development board is 20MHz. Since data is received from the host at 1.5MHz the input can be run to the PLL process at 6MHz allowing the receiver to sample the incoming data at 6MHz. The PLL checks for a transition in the data being received by the receiver. If there were a transition the output clock signal would be toggled. If there were no transition, the PLL would detect if two input clock cycles had passed since the last toggle of the Output clock signal. If yes, the signal would be toggled. Hence a complete output wave is formed this way every incoming bit is sampled. For keep tracking of when to toggle the output clock signal a counter is used. The PLL can effectively only synchronize the USB Interface's clock phase with that of the host. It is important to note that it does not replicate the host's exact Frequency because the development board's clock is not fast enough. In order for the receiver not to lose synchronization a transition on the incoming bit stream must occur on a regular basis. NRZI encoding does not guarantee a regular Transition. (NRZI explained below)A bit stream of only '1's results in an encoded bit stream without transitions.

➢ CLOCK DRIVER

The clock driver gives the divided clock to the transmitter and the USB interface so that the all operation can be synchronized.

➢ START OF PACKET DETECTOR

In USB the data travels in the form of packets, each packet start with a particular start of packet sequence, so that the synchronization between the packets should not be loosed, the work of this block is to detect the sequence "01010100" in the received bits the actual sync is "00000001" but it is NRZI encoded as "01010100". I update the state of the SOP on the receiving of each bit and finally when the last state is achieved the SOP is detected. The sequence "00000001" is chosen for ensuring the maximum transition density in the bit sequence.

➢ UPDATER

Well as the name signifies this block just updates one signal in the component. The updater is the block which updates the state of the SOP detector o that SOP detector can reach to its final state.

➢ NRZI DECODER

As I have discussed in the USB system and USN transfer protocol the received bits must be NRZI decoded first before processing to the USB interface. The bits sended by the host of the transmitter to the receiver are NRZI encoded; the NRZI encoding is explained in chapter 4, so the receiver has to decode this encoded sequence. As in NRZI the '0' is represented by a transition and '1' is represented by no transition. In decoding if the coming bit is same as the previous one then it is '1' else '0'.

➢ STUFFED BIT REMOVER

As I have discussed in the USB system and USN transfer protocol before NRZI encoding the bit stuffing is done hence after NRZI decoding stuffed bit must be removed. Bit stuffing is used for recovering the synchronization loss due to no transition in the bits. At the receiving end the receiver has to remove the stuffed bit , the logic used to remove this bit is to find a sequence 6 consecutive '1' or '0' in the decoded sequence. If the sequence found the next bit is discarded because it is stuffed one.

➢ SERIAL IN PARALLAL OUT (SIPO)

As the incoming data was parallel format so that the output should be the same hence a SIPO is employed there for giving the parallel output. It accepts the data in serial form after the stuffed bit has been removed and gives output 8 bits.

➢ DEMULTIPLEXER

Definitely the most important part of the receiver is the demultiplexer it the output byte by the SIPO and the CRC are fed to the demultiplxer and depending upon the selecting signals the various bits of the data out and CRC are allotted to the address , endpoints , PID, CRC check fields.

➢ CRC 5

The 5-bit CRC calculator process calculates the CRC for incoming bits. If the calculated CRC matches that sent in the packet, the receiver knows that the packet has been received correctly. Figure 7-6 shows how the 5-bit CRC process calculates the CRC. If the input to the CRC calculator is 1 the current value of the CRC register is XORed with the Polynomial $X_5 + X_2 + 1$ before being shifter left. Other wise the CRC register is shifted left without being XORed. The schematic shown in the figure works because a bit XORed with 0 is equal to itself.

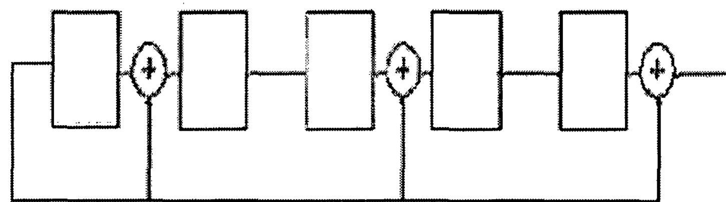


Figure 6 – 2 CRC – 5 Algorithms

➢ CRC 16

The 16-bit CRC calculator calculates the value of the 16-bit CRC for packets sent by the transmitter. The reader may surprise that why CRC 5 is not in the transmitter. Note that the transmitter does not have to calculate the 5-bit CRC since this is used as a check in TOKEN packets only. The transmitter does not send token packets. Though a 16-bit CRC will be mentioned here, for a more detailed description refer to the website "Understand Cyclic Redundancy Check". The 16-bit CRC checksum is applied to data packets. It works by shifting the bits left, in the same manner as CRC 5 has done. If the present input is a '1', then the value of the CRC register is XORed with the polynomial 11000000000000100. The CRC calculator works by shifting the data left and XORing the 15_{th}, 14_{th} and 3_{rd} digits with the input because 1000000000000101B is the standard generator polynomial. The value of the register only shifts left if the input is a 0. The data is only changed if the input is a 1 since any bit XORed with 0 is equal to itself. A 16-bit CRC is calculated as follows. A 16-bit value is used to hold the result of the CRC calculation. This value is initialized to '0'; it may also be initialized by '1'. If the input bit is a '0' the value is shifted left '1' bit. If the input is a '1' the value is shifted left 1-bit and XORed with the polynomial, $X_{16} + X_{15} + X_2 + 1$. The binary bit pattern that represents this polynomial is 1000000000000101B. Please see the USB Specifications, details of which are provided in the references, for more information on 16-bit CRC and USB.

➢ PID CHEKER

Well this is the block which is actually starting the transaction, because it tells that the transaction is IN or OUT. Hence it is an important block from the interface point of view. It tells the interface that which type of the packet has been received by the receiver it ells that weather the packet is a token packet, a data packet if data that of which type and weather it is a handshake packet. It checks the PID of the packet and compares it with the standard one there are two types of the fields in PID type field and the check field the check field in reverse of the PID field, it is also error checking method. The standard valued for the PID field for different packets are given below in the table 6 -1.

S.NO	PACKET TYPE	TYPE FEILD	CHECK FEILD
1	SOP	“1010”	“0101”
2	SETUP	“1011”	“0100”
3	OUT	“1000”	“0111”
4	IN	“1001”	“0110”
5	DATA 0	“1100”	“0011”
6	DATA 1	“1101”	“0010”
7	ACK	“0100”	“1011”
8	NAC	“0101”	“1010”
9	STALL	“0111”	“1000”

Table 6 -1 Interpretation of PID Values

6.3 TRANSMITTER

The next component in the interface is the transmitter the transmitter starts it work when there is an IN transaction then after the token packet the transmitter starts sending the data packet otherwise in OUT transaction the transmitter sends only the handshake packet , briefly I can summaries the work of the transmitter in following tasks.

- Parallel to serial conversion.
- Bit Stuffing.
- Sending start of packet (sync) sequence.
- Calculating and sending 16-bit CRC of packet so host can do error checking.
- Generate the correct PID as specified by the controller.
- Allowing controller to decide what type of byte to send.

Figure 6 - 3 shows a block diagram of the key components of the transmitter.

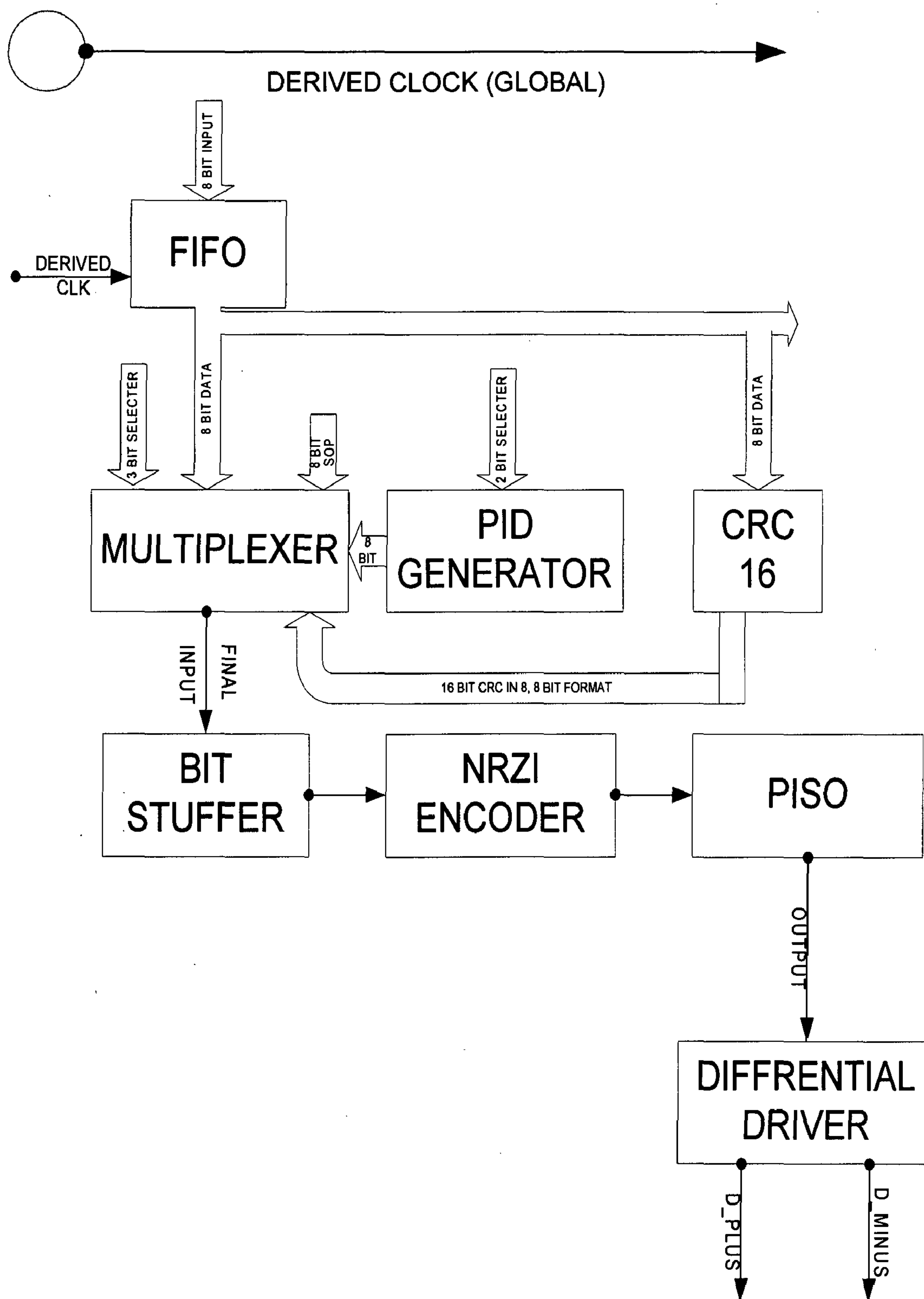


Figure 6 – 3 Transmitter Block Diagram

➢ NRZI ENCODER

The incoming bits are to be NRZI encoded as per the requirement of the transfer protocol hence I had implemented an NRZI encoder. In NRZI encoding the '0' is represented as a transition and '1'is represented as no transition.

➢ CRC 16 CALCULATOR

The 16-bit CRC calculator calculates the value of the 16-bit CRC for packets sent by the transmitter. Note that the transmitter does not have to calculate the 5-bit CRC since this is used as a check in TOKEN packets only. The transmitter does not send token packets. Though a 16-bit CRC will be mentioned here, The 16-bit CRC checksum is applied to data packets. It works by shifting the bits left. If the present input is a 1, then the value of the CRC register is XORed with the standard polynomial 110000000000000100. The CRC calculator works by shifting the data left and XORing the 15_{th}, 14_{th} and 3_{rd} digits with the input. The value of the register only shifts left if the input is a 0. The data is only changed if the input is a 1 since any bit XORed with 0 is equal to itself. A 16-bit CRC is calculated as follows. A 16-bit value is used to hold the result of the CRC calculation. This value is initialized to 0. If the input bit is a 0 the value is shifted left 1 bit. If the input is a 1 the value is shifted left 1-bit and XORed with the polynomial, $X_{16} + X_{15} + X_2 + 1$. The binary bit pattern that represents this polynomial is 1000000000000101B.

➢ PID GENERATOR

The PID Generator is told the type of PID to generate by the interface, through the pid_select port. The interface determines what type of PID to send after making sure that downstream data packets have been received correctly. If data has been received correctly, an acknowledgment (ACK) packet is sent. Otherwise a negative acknowledgement (NAC) packet is sent. It selects the PID with the help of an input_ select signal which is a 2 bit signal. Depending upon the value of the signal it decides that PID is ACK or NAK.

➢ MULTIPLEXER

The controlling part of the transmitter from the interface point of view the incoming data is first of all latched in the multiplexer and the depending the 3 bit mult_select signal it select signal which is given by the interface the multiplexer gives the byte out for the processing through the transmitter, it can be said that multiplexer is a controller inside the transmitter itself.

➢ FIRST IN FIRST OUT (FIFO)

This block is not included in the USB transfer protocol this block is added for data protection. The incoming data is first of all latched in a FIFO so that if the data changes before going out of the transmitter then there will be no data lose hence the transmitter work start with the byte coming out of the FIFO. Any type of FIFO can be used synchronous and asynchronous; in the USB interface I have designed I used a synchronous FIFO which is having width of 8 bits and depth of 16.

6.4 USB INTERFACE

The transmitter and receiver are the supporting parts to the interface the actual component in my design is USB interface .The interface forms the bridge between the host and the controller in controls the operation of both receiver and transmitter. It decides that what will be the data coming out and through the input to the transaction decider it decides that what will be the type of the transaction it controls the transmitter and receiver data paths. The controller must be able to handle the following packet types:

- Out and In.
- Data 0 and Data 1.
- ACK, NAK.

There are two sub parts in the interface which are.

- Transaction decider.
- Controller.

6.4.1 TRANSACTION DECIDER

This is the simple block which only decides the type of the transaction will be by simple if else logic, actually it dose not decides the transaction type transaction type is

decided by the PID type it actually decides the bit sequence to the input of the receiver and according to the correct bit stream the PID of the token packet is decided correctly weather it is an IN transaction or it is an OUT transaction.

6.4.2 CONTROLLER

The controller is the main part of this interface design, as the name signifies it controls the entire process of the transaction, the transmitter and receiver works only like a component of this controller the controller is actually a state machine. This state machine is controlled by various signals it is a Mealy machine the next state of the controller is dependent on the present state and the input. The data travels in the packet form in each packet there are several stages these stages are the states of the state machine. In each state there are some signal are set and some are reset as described below in the description. In the block diagram shown in the Figure 6 – 4 of the controller this can be easily seen that the controller is the block which is connected to the every lock in the interface.

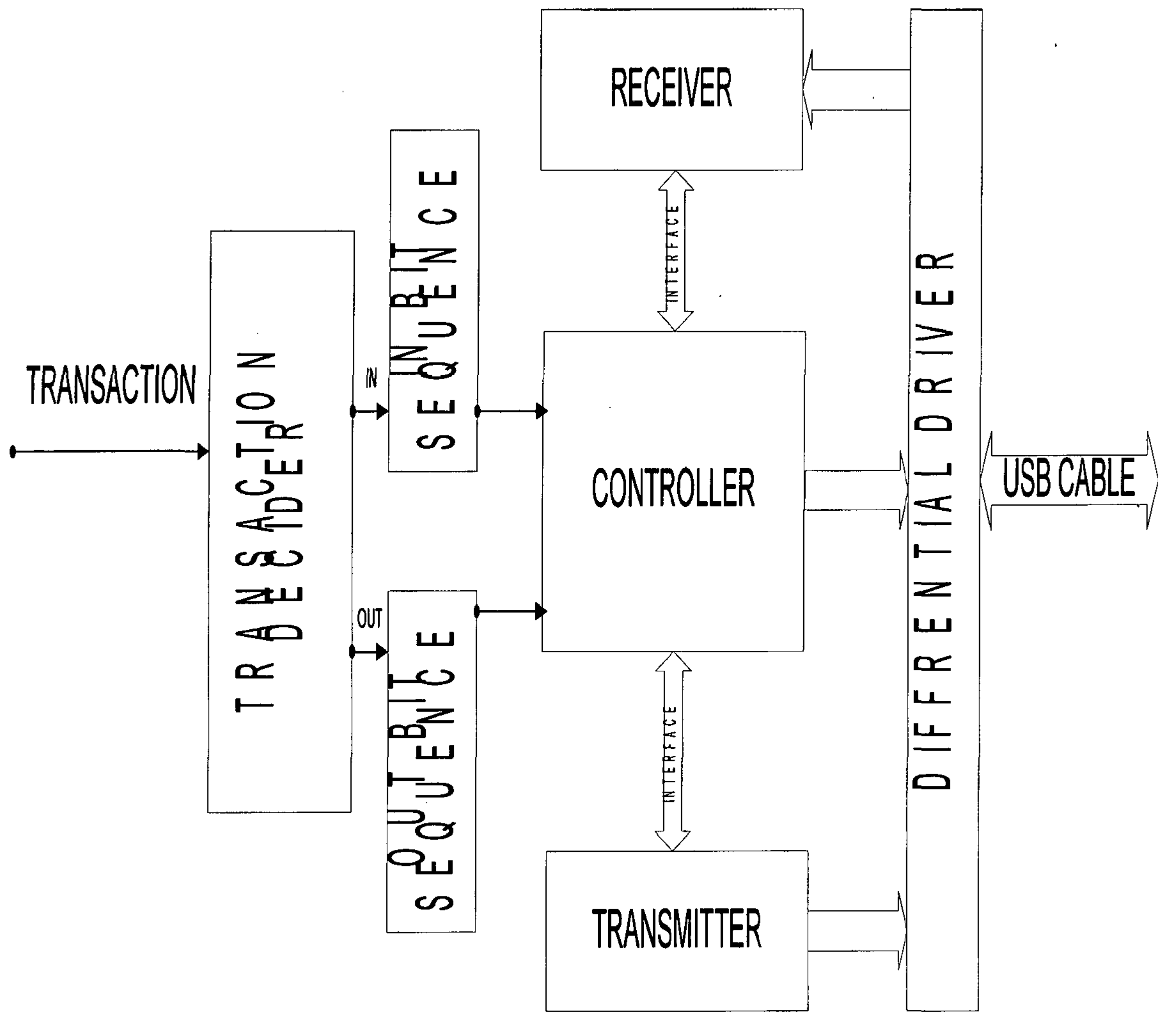


Figure 6 – 4 USB Interface Block Diagram

The state machine is as follows.

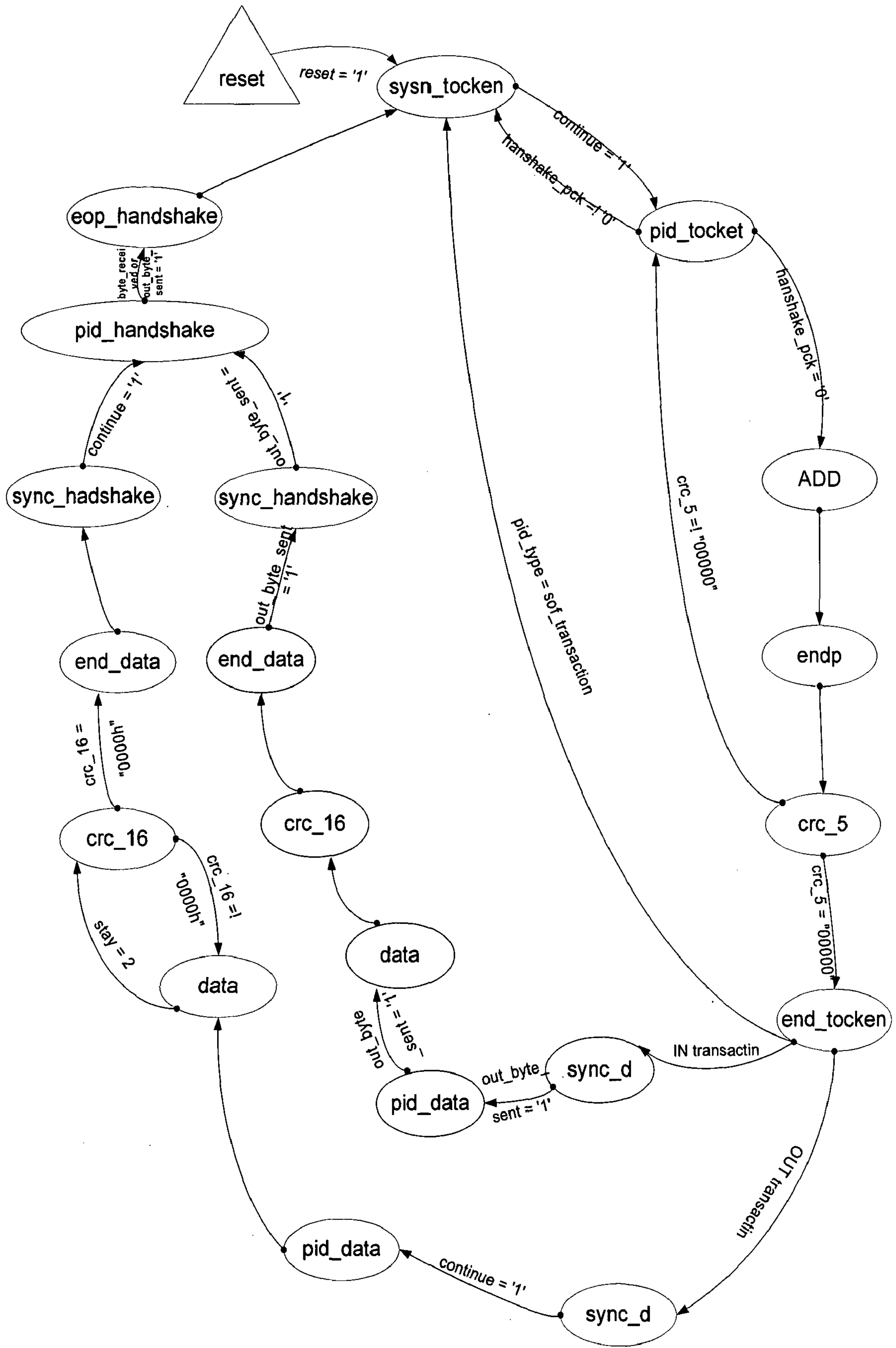


Figure 6 – 5 USB Controller State Machine

- **Sync_Tocken:** - This is the reset state and after the successful transmission the state machine returns to this state.
- **Pid_tocken**: - In this state machine checks the packet id of the token packet and identifies that it is an IN transaction or OUT transaction.
- **ADD**: - In this the address of the USB device is been calculated and shown as output of the interface.
- **Endp:** - In this state the register number in the USB device is been calculated and shown as the output of the interface.
- **Crc_tocken**: - In this the CRC_5 for the token packet is been checked if the final CRC 5 is "00000" then CRC check is passed else it goes pid_tocken state back.
- **Eop_tocken**: - In this state the eop pin is set hence the D_plus and D_minus both are low for some cycles. Signifies to the host that that the token packet is ended here.
- **Sync_data**: - Depending upon the type of transaction this state decides that the synchronization sequence to be send by the receiver or the transmitter.
- **Pid_data**: - In this state the PID of the data packet is checked weather it is a DATA 0 packets or DATA 1 packet.
- **Data**: - Depending upon the type of transaction the data byte is received from the host or the transmitter.
- **Crc_data**: - In this state if OUT transaction is there then CRC 16 is been checked if it comes out to be "0000000000000000" then CRC check is passed else the state machine returns to the data state. If IN transaction is there CRC is not checked because it is the work of the host.
- **Eop_data**: - In this state the eop pin is set hence the D_plus and D_minus both are low for some cycles. Signifies to the host that that the data packet is ended here.
- **Sync_handshake**:- Depending upon the type of transaction this state decides that the synchronization sequence to be send by the receiver or the transmitter.
- **Pid_handshake**: - In this state the PID of the handshake packet is checked weather it is an ACK or NAC packet.
- **Eop_handshake**: - In this state the eop pin is set hence the D_plus and D_minus both are low for some cycles. Signifies to the host that that the handshake packet is ended here.

CHAPTER 7
IMPLEMETATION

7.1 VHDL IMPLEMENTATION

In USB interface all the transaction should be started by the host itself but here only the interface part is implemented hence the start of the transaction is started by the user defined bits to the receiver (because each transaction is started by the host so the receiver has to take the bits from the host and decide further transaction). In the FPGA board implementation the clock frequency is so high that I can not give the bits sequence by hand so I have made a transaction decider in the interface which asks user to tell which type of transaction he wants depending that it feds the particular bit sequence in the receiver.

7.2 BIT SEQUENCE

The bit sequence I have made a series of 96 bits which is fed to the receiver depending upon the type of transaction. This bit sequence is designed with taking precaution that receiver works with NRZI decoded bits. The complete bit sequence can be divided in several parts; there is a special bit sequence for each state. That is described below.

➢ SYNC_TOKEN

The synchronization sequence for any packet is “00000001” but receiver works on NRZI decoded data so before feeding this synchronization sequence the sequence should be NRZI encoded hence it is “01010100”. As shown in the Figure 7 -1 below.

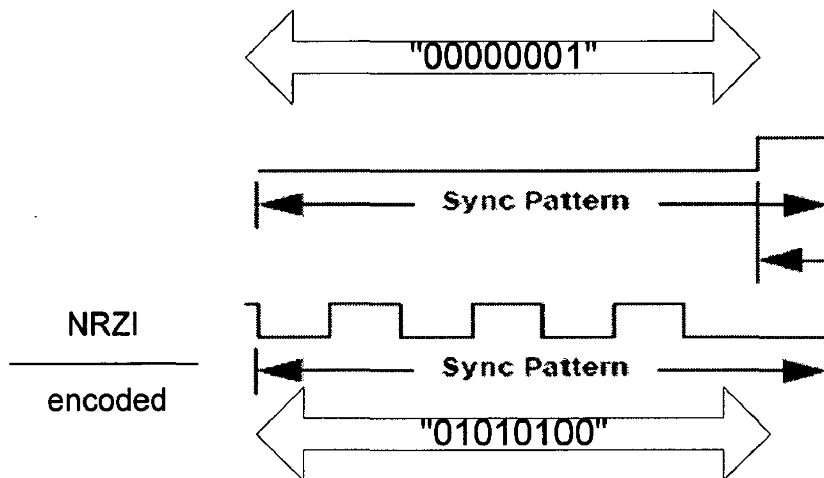


Figure 7 -1 Sync_Token

➢ PID_TOKEN

The next coming field id pid_tocken, it tells that which type of transaction it is IN or OUT. This is the field which identifies that what type of transaction host wants to do, for IN and OUT transaction the cases are shown below.

- For IN transaction the type field of pid_tocken is "1001" and check field is its invert that is "0110" hence the complete byte is "10010110" and after NRZI encoding it is "01001110" . As shown in the Figure 7 -2 below.
- For OUT transaction the type field of pid_tocken is "1000" and check field is its invert that is "0111" hence the complete byte is "10000111" and after NRZI encoding it is "01010000" . As shown in the Figure 7 -2 below

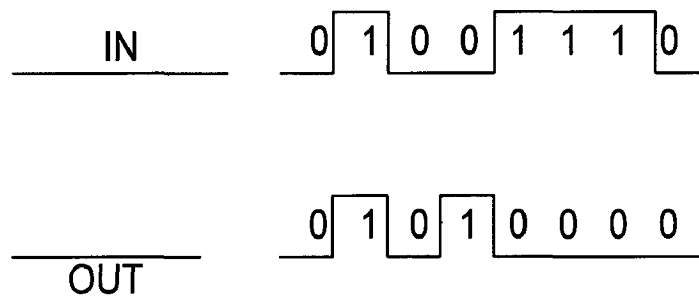


Figure 7 – 2 Pid_Token

➢ ADD

Address is defined as the last 7 bits of the output of the receiver. Let us select the address “0000000” for this the NRZI encoded bit sequence is “1010101”.

➢ ENDP

Endpoint is defined as the bits (0, 7, 6, and 5) of the output of the receiver, for selecting endpoint “0000” the output of the receiver should be “000XXXX0” hence input to the receiver in NRZI encoded form is “0101”.

➢ CRC_5

The 5 bit CRC is calculated on the 11 bit data (address and endpoint) with taking the generator polynomial $G(X) = X4 + X2 + 1$ the CRC for “10101010101” comes out to be “00101”.

➢ EOP_TOCKEN

In this state the eop bit is set hence no need to send any bit sequence, for maintaining the synchronization some arbitrary bits can be send.

➢ SYNC_DATA

Now data packet had been started so the first byte will be a sync byte, there may be two cases IN and OUT.

- For IN transaction the sync sequence is fed by the transmitter.
- For OUT transaction the sync sequence is same as “01010100”.

➢ PID_DATA

Depending upon the transaction there may be two cases.

- For IN transaction the pid_data field is given by the transmitter itself.
- For OUT transaction the pid_data field is (type field and check field) Let it be DATA 0 then type field is “1100” and the check field is “0011” so the complete byte is “11000011” and in NRZI encoded form “00101000”.

- **DATA**

Depending upon the transaction there may be two cases.

- For IN transaction the data is feed by the transmitter at the time of implementation hence let take the input "11101011" = EBh...
- For OUT transaction let the data is "01100001" so in NRZI encoded form it comes out to be "11101011".

- **CRC_DATA**

Depending upon the transaction there may be two cases.

- For IN transaction the crc_data is calculated by the transmitter on the data bits "11101011"with the help of the generator polynomial $X_{16} + X_{15} + X_2 + 1$, the CRC 16 comes out to be "0000001011111010".
- For OUT transaction the crc_data is been calculated on the data bits "11101011"with the help of the generator polynomial $X_{16} + X_{15} + X_2 + 1$, the CRC 16 comes out to be "0000001011111010".

- **EOP_DATA**

In this state the eop bit is set hence no need to send any bit sequence, for maintaining the synchronization some arbitrary bits can be send.

- **SYNC_HANDSHAKE**

Now handshake packet had been started so the first byte will be a sync byte, there may be two cases IN and OUT.

- For IN transaction the sync sequence is same as "01010100".
- For OUT transaction the sync sequence is fed by the transmitter.

- **PID_HANDSHAKE**

Depending upon the transaction there may be two cases.

- For IN transaction the pid_handshake field is (type field and check field)

Let it be ACK then type field is "0100" and the check field is "1011" so the complete byte is "01001011" and in NRZI encoded form "11011000".

- For OUT transaction the pid_data field is given by the transmitter itself.

➢ EOP_HANDSAKE

In this state the eop bit is set hence no need to send any bit sequence, for maintaining the synchronization some arbitrary bits can be send.

➢ FINAL BIT SEQEUNCES

Hence the final bit sequence for the OUT transaction is. Shown in Figure 7 – 3 and Figure 7 – 4 below.

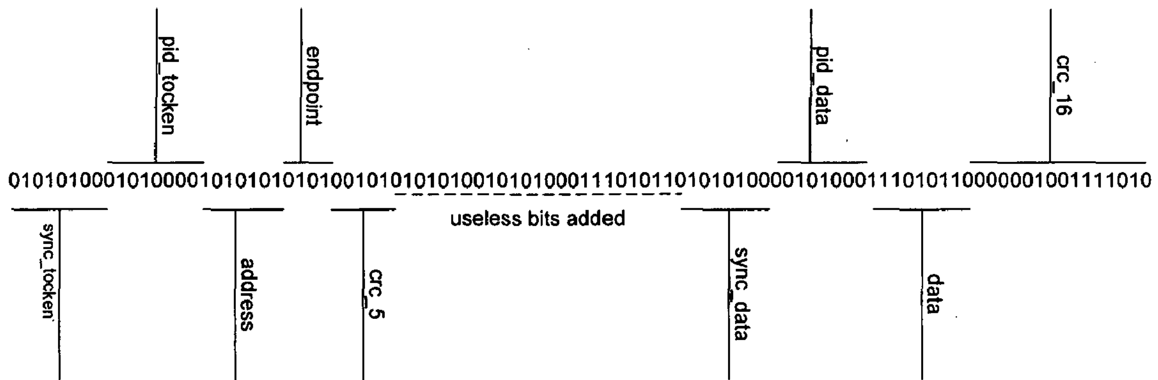


Figure 7 – 3 Final Bit Sequences For OUT Transaction

And for the IN transaction.

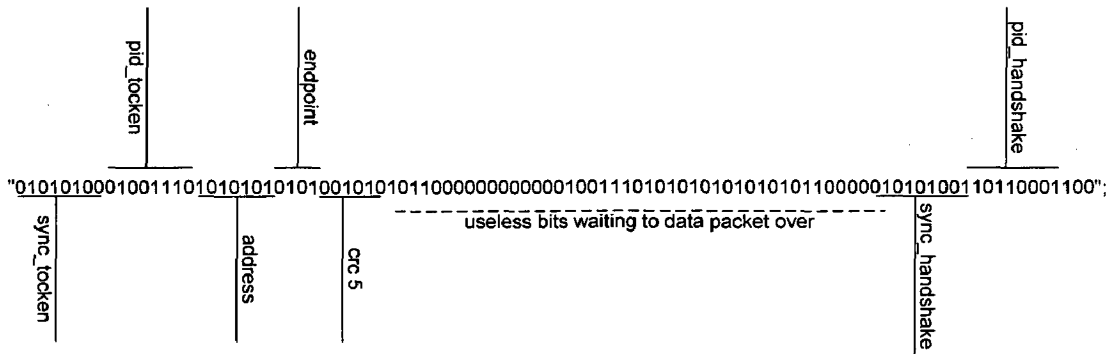


Figure 7 – 3 Final Bit Sequences For IN Transaction

7.3 FPGA IMPLEMMTATION

The program has been successfully loaded on the Xilix SpartanII FPGA development board. The done bit set on the FPGA board gives the guarantee that the results of the VHDL program will be as it is. But the clock frequency is too high that the user can not see the data transmission with the rapid glowing of the Leeds. So I prefer to show the demonstration by the help of simulation because in simulation there is a facility to see the outcome of the program at any moment, but in FPGA development board I can not see the transaction because the transaction has been completed in milliseconds. For seeing the output on the FPGA kit the clock dividing is a bad idea although if I reduce the clock below 6 MHz than the interface can not be used in actual USB communication and for seeing the LEDs 6 MHz is still too high.

CHAPTER 8
RESULTS AND DISCUSSION

For the verification of my design I have tested the output of the program and the results at different test inputs, out of them I m showing one of the test results.

8.1 TESTING PROCEDURE

For testing the interface design as I have described in the Chapter - 7 that user has to choose that which type of transaction he wants, he can choose either IN or OUT transaction, initially a fast input clock has to be given to the system which will further divided by the software itself, the reset is a global signal hence it had to be given and the last the input for the IN transaction should be given.
Hence in start there are only 4 inputs

- Fast clock
- Reset
- Input
- Transaction type

For starting the transaction first of all reset pin should be set so that all the signals , outputs are set to be '0', the make reset '0' for starting the transaction , if user choose transaction type is '1' then it is OUT transaction, if user choose transaction type is '0' then it is IN transaction .
The results and discussions on them can be divided in two categories

- Simulation results
- Synthesis results

8.2 SIMULATION RESULTS

As I have described earlier that simulation is the best method to see the performance of the interface design the simulation results can be seen on the platform of Active HDL 6.1. the simulations for two types of transactions are given below.

- IN transaction
- OUT transaction

8.2.1 IN TRANSACTION

For IN transaction choose transaction type '0' as soon as the reset pin is '0' the bit sequence

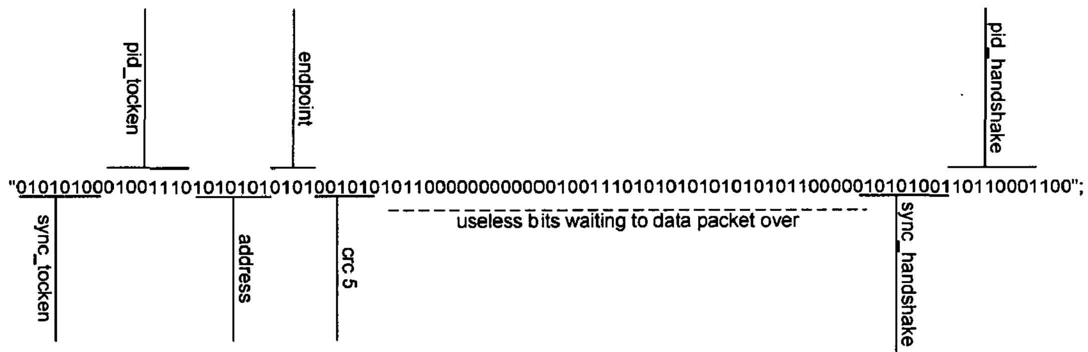


Figure 8 – 1 Successful Bit Sequence For IN Transaction

starts sending 1 bit in each clock to the controller this bit sequence is particularly designed for IN transaction let the input data be "11101011" = EBh then the output data will be NRZI encoded and it can be seen on the D_plus pin, as can be seen in the figure clearly as soon as the SOP is detected the continue bit is set and the state machine enters in the pid_token state. The address and endpoints are "0000000" and "0000" the crc_5 is "00000" that's indicate that CRC is matched and the CRC check is passed. The sync_data byte is sended by the transmitter hence it can be seen on the D_plus pin the data byte "11101011" is NRZI encoded as "00011000" which is seen at the D_plus pin, the synch_handshake is sended by the host that is also visible at D_plus pin .the data has been transfer successfully so the pid_handsake should be ACK hence pid_type should be 6 according to the program. After the successful transaction of the data the successful pin is set. Whenever the machine is in the state of eop then for some clock cycles both D_plus and D_minus are '0' else both are different. The result of the simulation are shown in the Figure 8 -2, reader can verify the performance of the interface design. The red circles are points where the desired results occur.

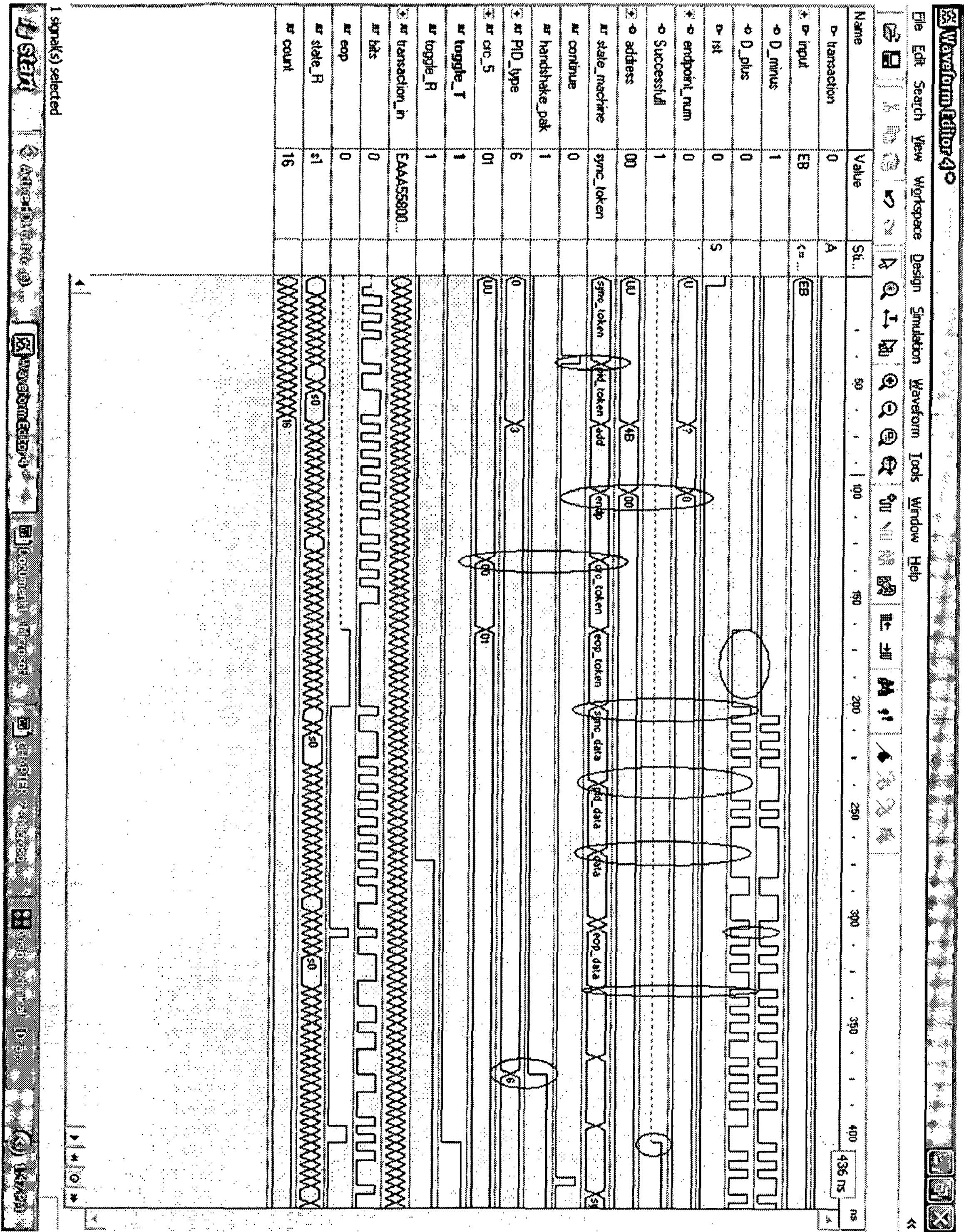


Figure 8 – 2 Simulation Result For IN Transaction.

8.2.2 OUT TRANSACTION

For OUT transaction choose transaction type '1' as soon as the reset pin is low the bit sequence

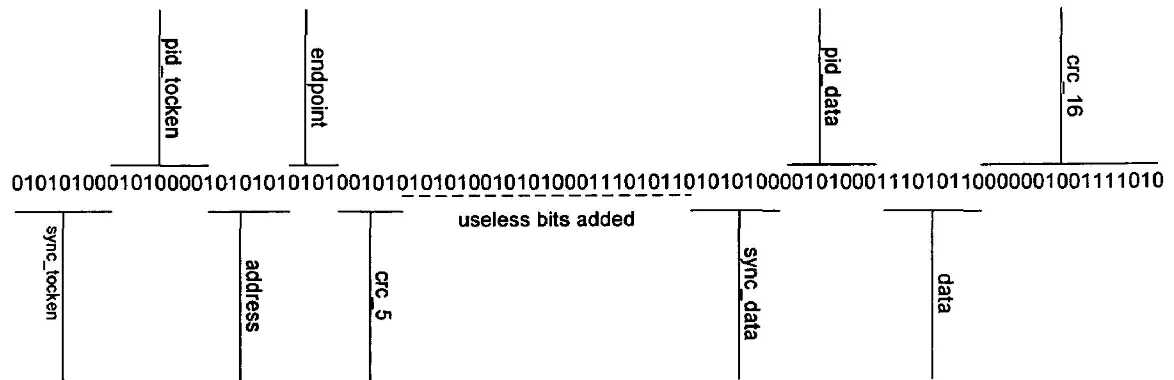


Figure 8 – 3 Successful Bit Sequence For OUT Transaction

starts sending 1 bit in each clock to the controller this bit sequence is particularly designed for OUT transaction let the input data be "01100001" = 61h , NRZI encoded "11101011"this data can be seen on the output pins in the data state as soon as the data state passes its 8rt bit . As can be seen in the figure clearly as soon as the SOP is detected the continue bit is set and the state machine enters in the pid_token state. The address and endpoints are "0000000" and "0000" the crc_5 is "00000" that's indicate that CRC is matched and the CRC check is passed. sync_data is been sended by the host the pid_data is "00101000" which is NRZI decoded as "11000011" = C3h which can be seen on pid_t pin. The CRC 16 is sended by the host for "11101011" it comes out to be "0000001001111010" the CRC 16 is "0000h" hence CRC check is passed .after that the sync_handshake, and pid_handshake signals been sended by the transmitter that can seen on the D_plus pin . after the successful transaction of the data the successful pin is set. whenever the machine is in the state of eop then for some clock cycles both D_plus and D_minus are '0' else both are different. The result of the simulation are shown in the Figure 8 -4, reader can verify the performance of the interface design. The red circles are points where the desired results occur.

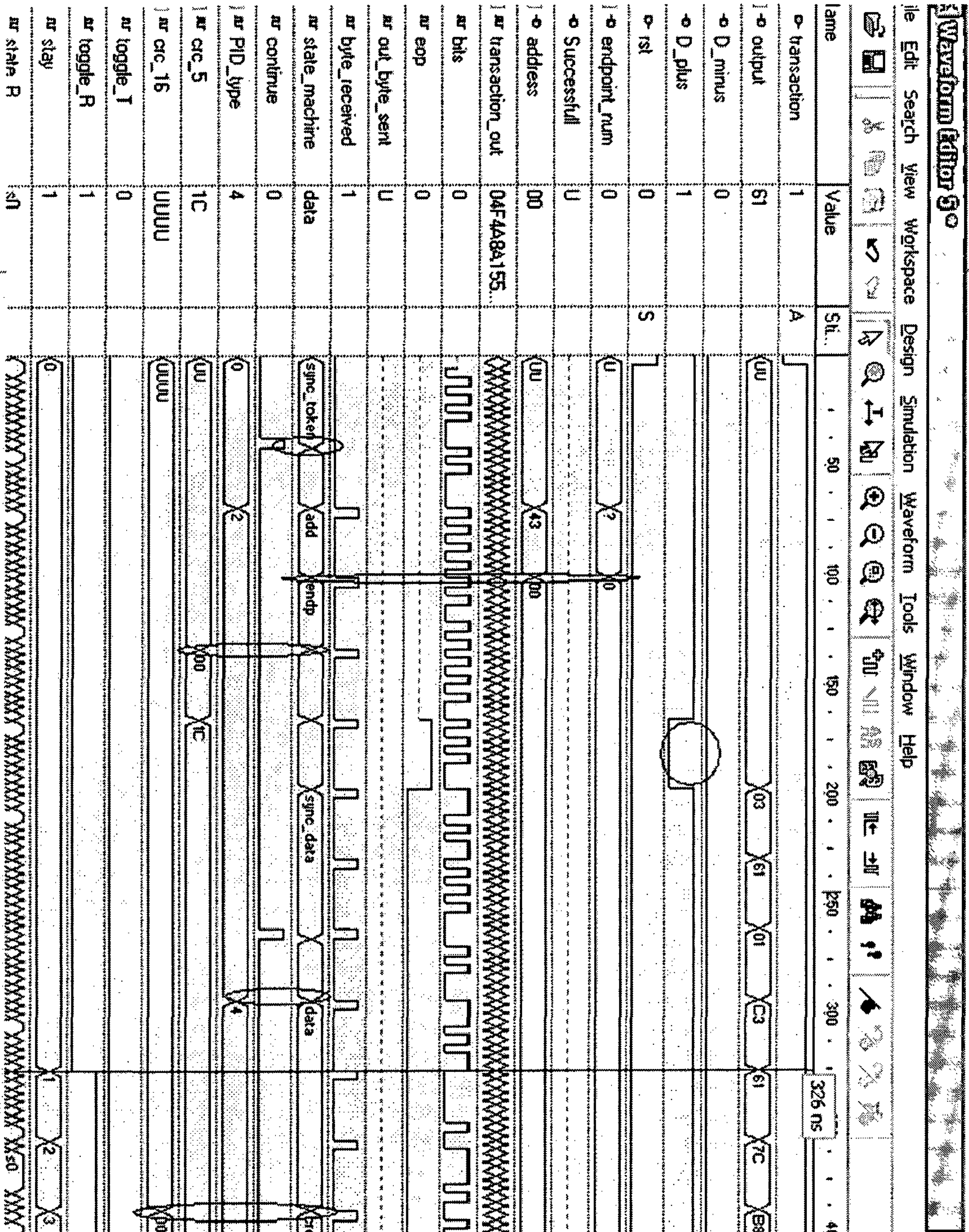


Figure 8 – 4 Simulation result for OUT transaction.

8.3 SYNTHESIS RESULTS

the program has been successfully synthesized and the then it had been translated, mapped, placed and routed also successfully on Xilinx FPGA Spartan II development Board. The RTL view of the design can be seen in the Appendix B. In reset condition it displays the reset and in IN or OUT transaction condition the transfer is so fast that one can not the blinking of the LEDs so only D_plus, D_minus and successful pin values can be seen in the board. LEDs.

8.4 DEVICE UTILIZATION SUMMARY

Selected Device: 2s200pq208-5

Number of Slices:	468	out of	2352	19%
Number of Slice Flip Flops:	568	out of	4704	12%
Number of 4 input LUTs:	597	out of	4704	12%
Number of bonded IOBs:	34	out of	144	23%
Number of GCLKs:	2	out of	4	50%

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

9.1 CONCLUSIONS

After seeing the simulation results and the implementation on the development board it can be concluded that the USB interface has been implemented successfully and designs provides an interface for "system on a chip" designers to connect to a USB bus. This will save their designing time and the time required debugging and testing a USB Controller if they were to implement the functionality of the interface themselves. This design can be used for the develop board like Altera or Xilinx. The electronic industries race to reduce the size of devices has led to the design of entire systems on a chip. The USB standard has made attaching peripherals to PCs incredibly simple. These trends of "System on a Chip" and the rising popularity of USB is the basis of this thesis. Due to the limited clock frequency of the FPGA development board used for this thesis only low speed USB was implemented. Low speed USB has a maximum bandwidth of 1.5 MHz. although a high speed USB interface can easily be developed with slight alteration in the design.

The word multi-purpose signifies itself because this interface can be used to transfer data in both IN and OUT direction. Moreover any peripheral can be attached to transfer the data just after parallel port interface is ready .The three components in the interface design which are

- Transmitter
- Receiver
- USB Interface

works in synchronization so that no bit is loosed hence the interface work for data protection also, more over one FIFO is also implemented in the transmitter component so that if the data changes before the data packet starts then the previous data should not be loosed hence the FIFO implementation approach is also helpful in data protection, this FIFO approach is not present USB transfer protocol hence it may be considered as an added advantage in the interface design. A PLL has been implemented in the receiver component which synchronize the all parts of the

interface moreover due to the dynamic clock management in the PLL it also helps in power saving.

For the testing and verification of the design the simulation results and the synthesis results are placed in the report user can verify the design for its own data.

This thesis helped tremendously for understanding the USB protocol and its working, and my VHDL knowledge is greatly enhanced because of writing the code for the interface design, near about all major tools of VHDL has been used in the design.

9.2 FUTURE SCOPE OF WORK

Well nothing in this world is finally completed and modifications and developments are available to every work, so my work also, hence here listed few modifications and some more developing works which can be added to this thesis in future.

- In this interface the design asks user to tell the type of transaction initially well if the parallel port interface is implemented then through the parallel port user can send his data with the bit sequence in this case the user don't want to tell that what type of transaction he wants, his bit sequence will automatically detect the type of transaction.
- After the data is present on the D_plus and D_minus pins the data can be placed to USB port through the USB cable. So that actual data flow occurs.
- For seeing the actual data transfer to some computer a device driver should be written , it can be written Linux, actually the USB device drivers are freely available with windows software but this design has some limitations and modification so a different device driver has to be written. Well this modification is beyond the scope of this topic.
- This designs is limited to only low speed data transfer because the clock frequency is limited so it can not use Isochronous data transfer , using the Altera development board the full speed interface can be made .
- An ADC can be implemented instead a parallel port interface so that any peripheral can be attached to design to transfer the data.

REFRENCES

1. FPGA Research at the University of Toronto, performs research in FPGA ASIC and CAD, HTTP:
http://www.eecg.toronto.edu/EECG /RESEARCH/FPGA.html

2. Mentor Graphics ,"FPGA ADVANTAGE, Design, simulation and systhesis"HTTP:
http://www.mentor.com/products/fpga_pld/fpga_advantage/
index.cfm?v=google&p=adwords_fpga&s=1x1&g=fpga&c=fpga_ocid_712b_cfp_581_ceid_20-1x1-&asid=260

3. Perry, D. L. 2004. VHDL: Programming By Example, 4th ed. New York: McGraw-Hill Companies inc.

4. Bhasker, J. 1997. A VHDL Primer. Allentown, PA: Star Galaxy Press

5. Wakerly, J. F. 2000. *Digital Design: Principles and Practices*, 3rd ed. Upper Saddle River, NJ: Printice Hall.

6. Mano, M. M. 2001. *Digital Design*, 3rd ed. Upper Saddle River, NJ: Printice Hall

7. FPGAWorld "Demos on Demand" [Online Demos are available by vendors of VLSI firm, HTTP:
http://www.demosondemand.com/dod/

8. CTU Prague, Faculty of Electrical Engineering Department of Computer Science and Engineering "FPGA IMPLEMENTATION OF USB 1.1 DEVICE CORE" Karlovo náměstí 13, Praha 2
http://service.felk.cvut.cz

9. USB on-the-go interface for portable devices
Remple, T.B.;
Consumer Electronics, 2003. ICCE. 2003 IEEE International Conference on
17-19 June 2003 Page(s):8 - 9

10. The design of a USB device controller IYOYOYO
Kouyama, T.; Nano, H.; Kon, C.; Shimizu, N.;
Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific
21-24 Jan. 2003 Page(s):573 – 574

11. USB on-the-go interface for portable devices
Remple, T.B.;

Consumer Electronics, 2003. ICCE. 2003 IEEE International Conference on
17-19 June 2003 Page(s):8 - 9

12. Universal serial bus implementation in an integrated access chip for ISDN systems
Cruickshank, H.; Sun, Z.; Fan, Z.;
Communications, IEE Proceedings-
Volume 148, Issue 4, Aug. 2001 Page(s):207 – 211

13. Universal serial bus (USB) to universal interface using field programmable gate arrays (FPGA) to mimic traditional hardware [military aircraft testing applications]
Stong, N.;
AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference. Proceedings
22-25 Sept. 2003 Page(s):386 – 391

14. USB printer driver development for handheld devices
Damodharan, T.K.; Rhymend Uthariaraj, V.;
Information Technology Interfaces, 2004. 26th International Conference on
2004 Page(s):599 - 602 Vol.1

15. A microcontroller based data acquisition system with USB interface
Popal, M.; Marcu, M.; Popa, A.S.;
Electrical, Electronic and Computer Engineering, 2004. ICEEC '04. 2004 International Conference on
5-7 Sept. 2004 Page(s):206 - 209

16. Advanced low power system design techniques using the universal serial bus microcontroller
Hathaway, T.; Verma, V.;
Northcon/96
4-6 Nov. 1996 Page(s):270 – 274

17. A cyclic-executive-based QoS guarantee over USB
Chih-Yuan Huang; Li-Pin Chang; Tei-Wei Kuo;
Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE
27-30 May 2003 Page(s):88 – 95

18. Universal serial bus (USB) power management
Lynn, K.;
WESCON/98
15-17 Sept. 1998 Page(s):194 – 201

19. Universal serial bus (USB) power management
Lynn, K.;
WESCON/97. Conference Proceedings
4-6 Nov. 1997 Page(s):434 - 441

20. Interleaved cyclic redundancy check (CRC) code
Jun Jin Kong; Parhi, K.K.;
Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on
Volume 2, 9-12 Nov. 2003 Page(s):2137 - 2141 Vol.2

21. Hardware design and VLSI implementation of a byte-wise CRC generator chip
Sait, S.M.; Hasan, W.;
Consumer Electronics, IEEE Transactions on
Volume 41, Issue 1, Feb. 1995 Page(s):195 – 200

22. Single bit error correction implementation in CRC-16 on FPGA
Shukla, S.; Bergmann, N.W.;
Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on
2004 Page(s):319 – 322

23. Cyclic redundancy code (CRC) polynomial selection for embedded networks
Koopman, P.; Chakravarty, T.;
Dependable Systems and Networks, 2004 International Conference on
28 June-1 July 2004 Page(s):145 – 154

24. A tutorial on CRC computations
Ramabadran, T.V.; Gaitonde, S.S.;
Micro, IEEE
Volume 8, Issue 4, Aug. 1988 Page(s):62 – 75

25. High-speed parallel CRC circuits in VLSI
Pei, T.-B.; Zukowski, C.;
Communications, IEEE Transactions on
Volume 40, Issue 4, April 1992 Page(s):653 – 657

26. Dynamic Clock Management for Low Power application in FPGAs
Ian Brynjolfson and Zeljko Zilic
Deratment of Electrical Engineering , McGill University
Montreal, quebec, Canada.

27. Dynamic Speed-Setting of a Low-Power CPU", *Proceedings of 1st ACM Intenwtionul Conference on Mobile Computing und Networking,* 1995.

28. USB specification 1.1, September 1998:
http://www.usb.org

29. ANDERSON, D.: 'Universal serial bus system architecture' (Mind-Share Inc., 1997) ISBN 0-201-46137-4

30. Xilinx application notes at
http://www.xilinx.com

31. Ashenden P.J. 1997 "The VHDL Cookbook" Morgan Kaufmann Publishers
http://www.cs.adelaide.edu.au/~petera.

APPENDIX

APPENDIX A
TIMING SIMULATION WAVEFORMS

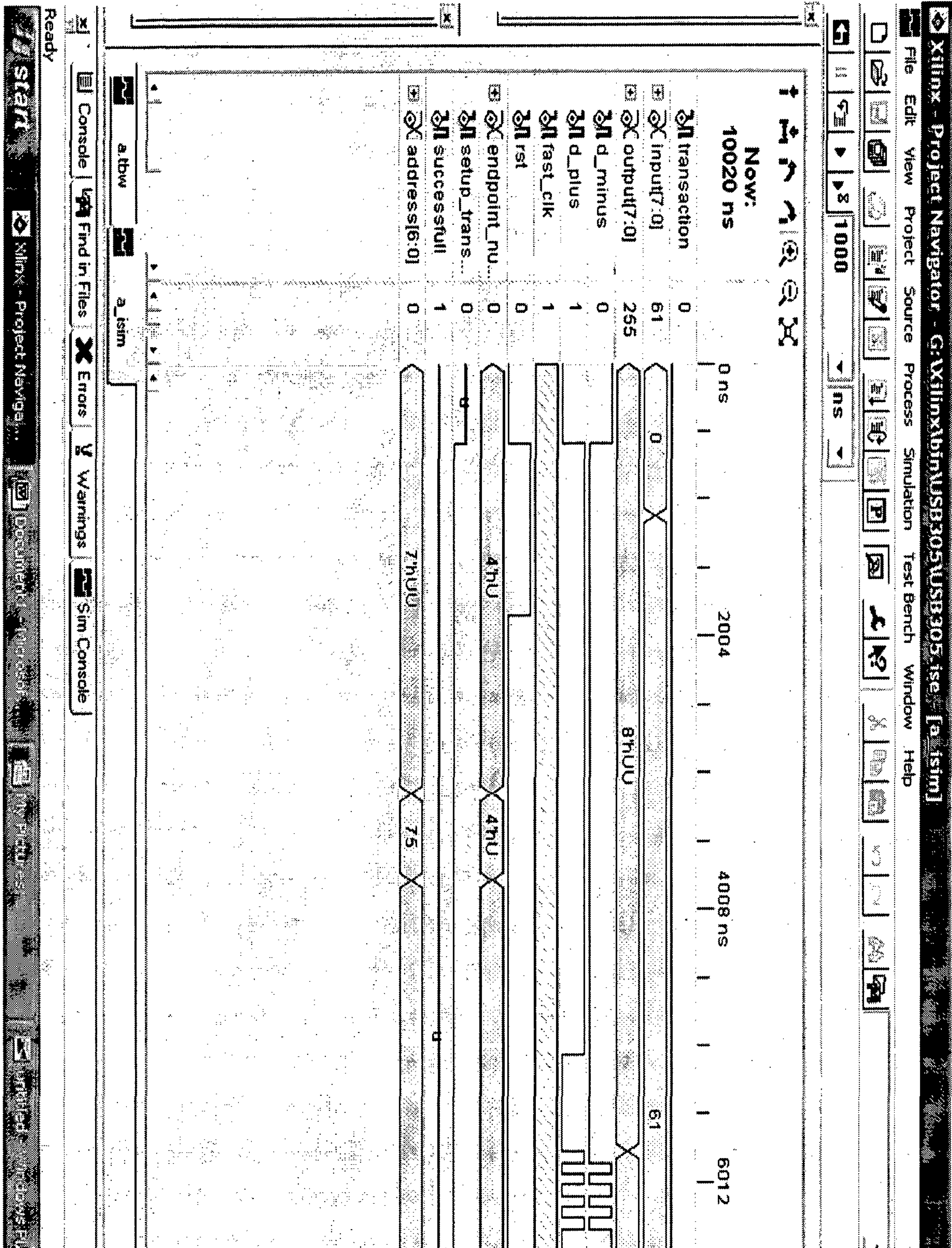


Figure AA -1 Timing Simulation Waveform

APPENDIX B
RTL VIEW

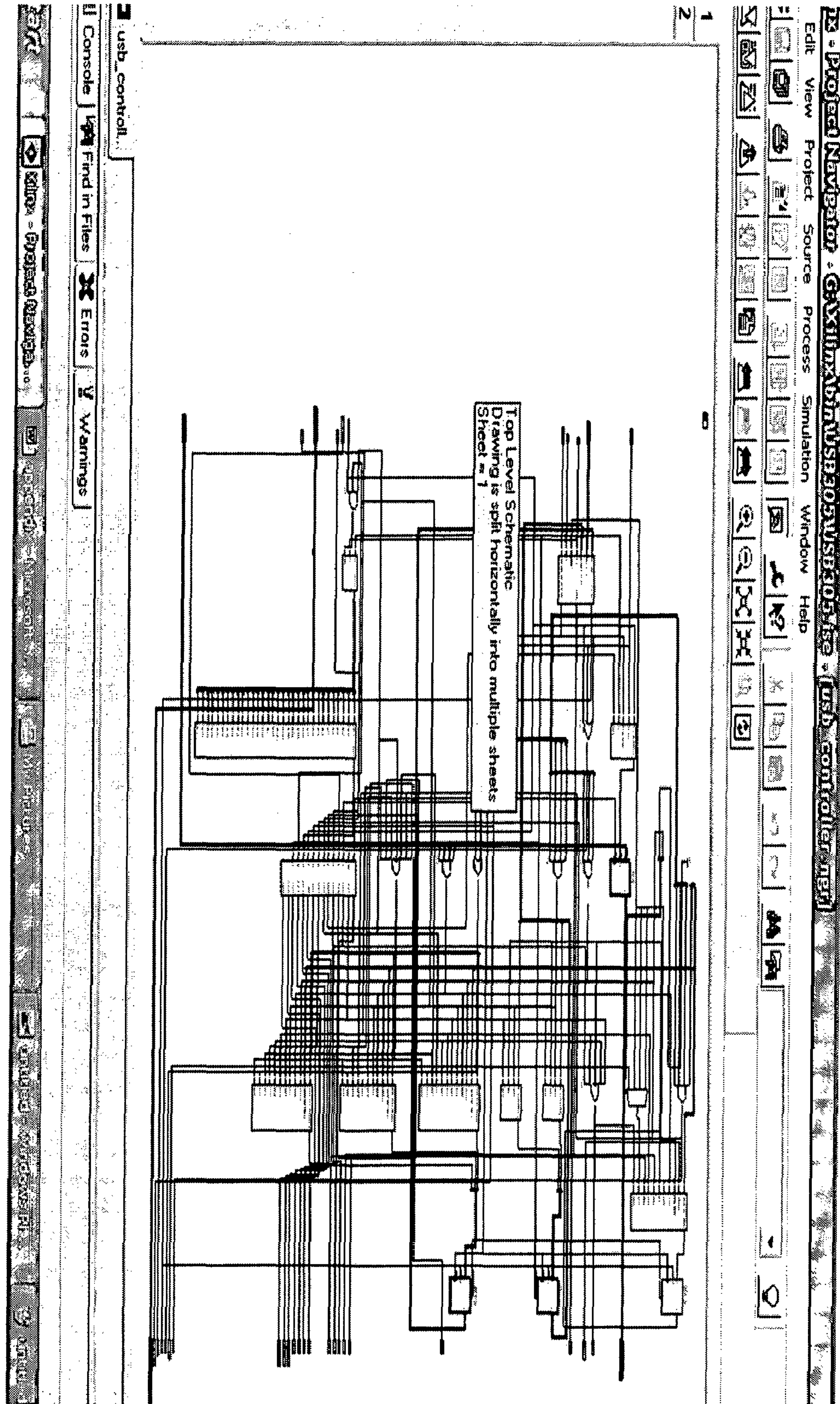


Figure AB -1 RTL VIEW PART -1

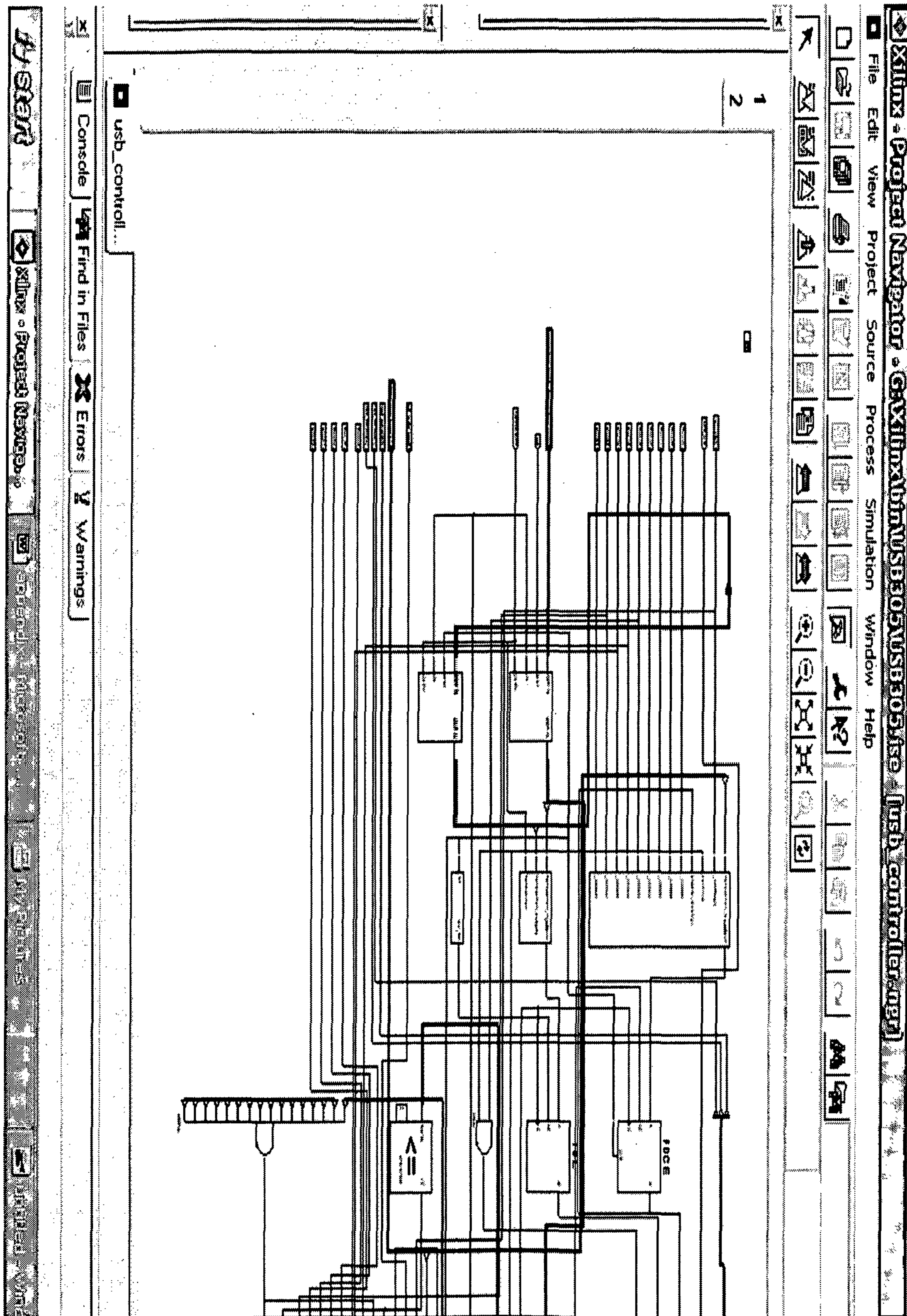


Figure AB -2 RTL VIEW PART -2

APPENDIX C
PID TYPE AND HANDSHAKE RESPONSE

PID Type	PID Name	PID[3:0]*	Description
Token	OUT	0001B	Address + endpoint number in host-to-function transaction
	IN	1001B	Address + endpoint number in function-to-host transaction
	SOF	0101B	Start-of-Frame marker and frame number
	SETUP	1101B	Address + endpoint number in host-to-function transaction for SETUP to a control pipe
Data	DATA0	0011B	Data packet PID even
	DATA1	1011B	Data packet PID odd
Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Rx device cannot accept data or Tx device cannot send data
	STALL	1110B	Endpoint is halted or a control pipe request is not supported.
Special	PRE	1100B	Host-issued preamble. Enables downstream bus traffic to low-speed devices.

Table AC - 1 Types of PIDs and their description.

Token Received Corrupted	Function Tx Endpoint Halt Feature	Function Can Transmit Data	Action Taken
Yes	Don't care	Don't care	Return no response
No	Set	Don't care	Issue STALL handshake
No	Not set	No	Issue NAK handshake
No	Not set	Yes	Issue data packet

Table AC - 2 Device Responses To IN Transactions

Data Packet Corrupted	Host Can Accept Data	Handshake Returned by Host
Yes	N/A	Discard data, return no response
No	No	Discard data, return no response
No	Yes	Accept data, issue ACK

Table AC - 3 Host Responses To IN Transactions

Data Packet Corrupted	Receiver Halt Feature	Sequence Bits Match	Function Can Accept Data	Handshake Returned by Function
Yes	N/A	N/A	N/A	None
No	Set	N/A	N/A	STALL
No	Not set	No	N/A	ACK
No	Not set	Yes	Yes	ACK
No	Not set	Yes	No	NAK

Table AC – 4 Device Responses to an OUT Transaction

APPENDIX D
DEVICE STATUS

Attached	Powered	Default	Address	Configured	Suspended	State
NO	-	-	-	-	-	Device is not attached to the USB. Other attributes are not significant.
YES	NO	-	-	-	-	Device is attached to the USB, but is not powered. Other attributes are not significant.
YES	YES	NO	-	-	-	Device is attached to the USB and powered, but has not been reset.
YES	YES	YES	NO	-	-	Device is attached to the USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.
YES	YES	YES	YES	NO	-	Device is attached to the USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.
YES	YES	YES	YES	YES	NO	Device is attached to the USB, powered, has been reset, has a unique address, is configured, and is not suspended. The host may now use the function provided by the device.
YES	YES	-	-	-	YES	Device is, at minimum, attached to the USB and is powered and has not seen bus activity for 3 ms. It may also have a unique address and be configured for use. However, because the device is suspended, the host may not use the device's function.

Table AD – 1 Device Status

APPENDIX E
SPARTAN-II 2.5V FPGA FAMILY

FEATURES

1. Second generation ASIC replacement technology.
   - Densities as high as 5,292 logic cells with up to 200,000 system gates.
   - Streamlined features based on Virtex architecture.
   - Unlimited reprogrammability.
   - Very low cost.
   - Advanced 0.18 micron process.
2. System level features
   - Select RAM+™ hierarchical memory:
   - 16 bits/LUT distributed RAM
   - Configurable 4K bit block RAM
   - Fast interfaces to external RAM
   - Fully PCI compliant
   - Low-power segmented routing architecture
   - Full read back ability for verification/observability
   - Dedicated carry logic for high-speed arithmetic
   - Efficient multiplier support
   - Cascade chain for wide-input functions
   - Abundant registers/latches with enable, set, reset
   - Four dedicated DLLs for advanced clock control
   - Four primary low-skew global clock distribution nets
   - IEEE 1149.1 compatible boundary scan logic
3. Versatile I/O and packaging
   - Pb-free package options
   - Low-cost packages available in all densities
   - Family footprint compatibility in common packages
   - 16 high-performance interface standards
   - Hot swap Compact PCI friendly
   - Zero hold time simplifies system timing
4. Fully supported by powerful Xilinx development system
   - Foundation ISE Series: Fully integrated software
   - Alliance Series: For use with third-party tools
   - Fully automatic mapping, placement, and routing

GENERAL OVERVIEW

The Spartan-II family of FPGAs have a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs). There are four Delay-Locked Loops (DLLs), one at each corner of the die. Two columns of block RAM lie on opposite sides of the die, between the CLBs and the IOB columns. These functional elements are interconnected by a powerful hierarchy of versatile routing channels (see Figure AE - 1). Spartan-II FPGAs are customized by loading configuration data into internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions and interconnections implemented in the FPGA. Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or Boundary Scan modes. Spartan-II FPGAs are typically used in high-volume applications where the versatility of a fast programmable solution adds benefits. Spartan-II FPGAs are ideal for shortening product development cycles while offering a cost-effective solution for high volume production. Spartan-II FPGAs achieve high-performance, low-cost operation through advanced architecture and semiconductor technology. Spartan-II devices provide system clock rates up to 200 MHz. Spartan-II FPGAs offer the most cost-effective solution while maintaining leading edge performance. In addition to the conventional benefits of high-volume programmable logic solutions, Spartan-II FPGAs also offer on-chip synchronous single-port and dual-port RAM (block and distributed form), DLL clock drivers, programmable set and reset on all flip-flops, fast carry logic, and many other features.

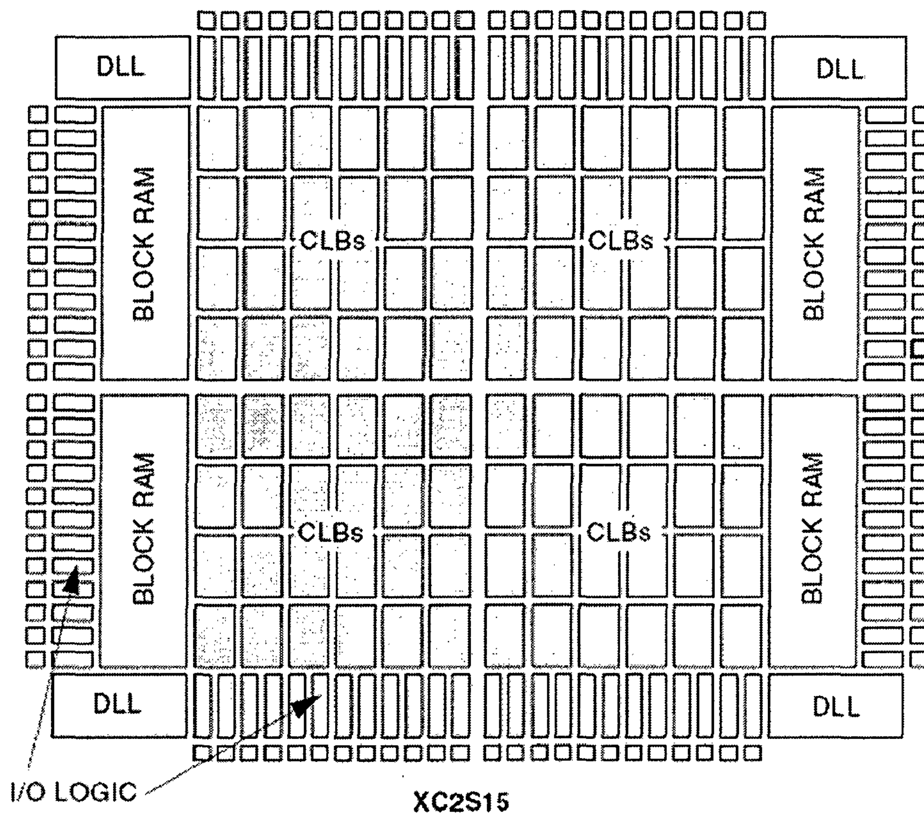


Figure AE -1: Basic Spartan-II Family FPGA Block Diagram

ARCHITECTURAL DESCRIPTION

Spartan-II Array

The Spartan-II user-programmable gate array, shown in Figure AE- 1, is composed of five major configurable elements:

- IOBs provide the interface between the package pins and the internal logic
- CLBs provide the functional elements for constructing most logic
- Dedicated block RAM memories of 4096 bits each
- Clock DLLs for clock-distribution delay compensation and clock domain control
- Versatile multi-level interconnects structure.

As can be seen in Figure AE -1, the CLBs form the central logic structure with easy access to all support and routing structures. The IOBs are located around all the logic

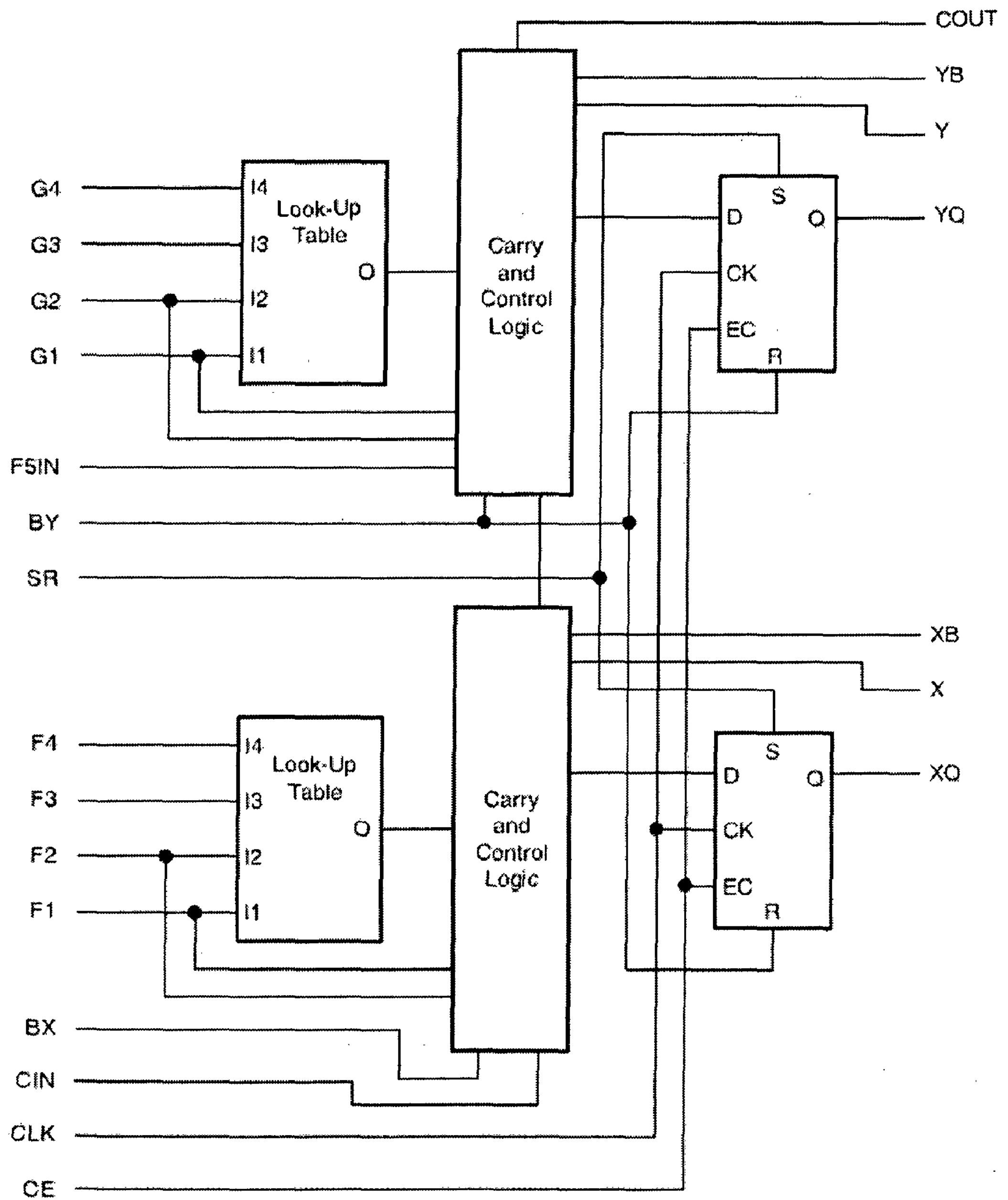


Figure AE -3: Spartan-II CLB Slice (two identical slices in each CLB)