FPGA BASED IMPLEMENTATION OF BIT ERROR RATE FOR HIGH SPEED LINKS

Thesis submitted in partial fulfillment of the requirement for
the award of degree of

Master of Technology

in

VLSI Design & CAD

Submitted by

Deep Mittal
(60761003)

Under the supervision of

Ms. Alpana Agarwal
Assistant Professor
Thapar University

Mr. Raj Kumar Nagpal
Section Manager
ST Microelectronics

Department of Electronics & Communication Engineering
Thapar University
Patiala-147004 (Punjab), India
July 2009

DECLARATION

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in VLSI and CAD at Thapar University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Deep

Deep Mittal

Certificate

This is to certify that the thesis work entitled "FPGA Based Implementation of BER for High Speed Links" has been carried out by Deep Mittal (60761003) for the degree of Master of Technology in VLSI and CAD at STMicroelectronics Pvt. Ltd., Greater Noida, India, under our supervision.

Raj Kumar Nagpal

Mr. Raj Kumar Nagpal
Section Manager (Central Laboratory),
STMicroelectronics Pvt. Ltd.,
Greater Noida

Alpana 14/7/09

Ms. Alpana Agarwal
Assistant Professor
Thapar University,
Patiala

Counter Signed By

Dr. A.K Chatterjee
Professor & Head of ECED,
Thapar University,
Patiala.

R K Sharma 21/7

Dr. R.K Sharma
Dean of Academic Affairs,
Thapar University,
Patiala.

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to Ms. Alpana Agarwal, Assistant Professor, ECED, Thapar University, Patiala and Mr. Raj Kumar Nagpal, Section Manager, CEC Department, ST Microelectronics, Greater Noida for their patient guidance and support throughout this report. I was truly very fortunate to have the opportunity to work under them. It was both an honour and a privilege for me to work with them. They also provide help in technical writing and presentation style and I found this guidance to be extremely valuable.

I am also thankful to Dr. A.K.Chatterjee, Head of Electronics and Communication Engineering Department and also to Mrs. Manu Bansal (Senior Lecturer), Mr. B.K Hemant (Project Faculty), Mr. Arun Chatterjee (Lecturer), Mr. Mohd. Iliyas (Project Faculty), Mr. Parul Bansal (Design Engineer) and Mr. Parteek Damle (FPGA Engineer) for their motivation and inspiration that triggered me for my thesis work.

I am also thankful to all my friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I do not find enough words with which I can express my feeling of thanks to the entire faculty and staff of ECED, Thapar University, Patiala and entire team of ST Microelectronics, for their help, inspiration and moral support which went a long way in successful completion of my work. I thank all those who have contributed directly or indirectly to this work.

Lastly and more importantly, I would like to thank my parents for their years of unyielding love and encouragement. They have always wanted the best for me and I admire my parent determination and sacrifice to put me through college.

Deep

Deep Mittal
60761003

ABSTRACT

Today's serial data communication rates are continue to increase at the fast pace due to the bandwidth hungry applications like image processing, video conferencing and many medical applications. As data rates are increasing, latency room gets still tighter. It in turn results in increased quality requirement on the interface timing and parasitic capacitance interconnect environment. Bit Error Rate is method to evaluate the quality figure of the communication link. ULPI Interface links are used more and more rather than UTMI interface links due to lesser skew and lower pin counts. USB 2.0 is used for this thesis work but the analysis is generic for all the links.

This thesis deals with counting the errors occurs during the High speed USB transmission. Bit errors results due to the improper design and implementation of the link or external noisy environment. Bit error rate is one of the parameter required to analyze the performance of the system. The fundamental purpose of BER is to verify procurement specifications what was specified and what was delivered. It verifies that the device meets the performance, design and implementation requirements identified in the procurement specifications without degradation in performance in any environment. In a nut shell, it demonstrates the quality figure of the communication standard USB2.0 and its performance in noisy environment where jitters and skew affect the performance of the system. The purpose of my thesis is to check the quality figure of the ULPI Interface of the PHY using FPGA based platform.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATION

BER	Bit Error Rate
BRAM	Block Random Access Memory
CRC	Cyclic Redundancy Check
ECG	Electrocardiogram
EOP	End of Packet
FPGA	Field Programmable Gate Array
FS	Full Speed Devices
HS	High Speed Devices
JTAG	Joint Test Action Group
LS	Low Speed Devices
NRZI	Non Return to Zero, Inverted
OTG	On The Go Devices
PC	Personal Computer
PHY	Physical Layer Board
PID	Packet Identifier
SE0	Single Ended Zero
SOP	Start of Packet
ULPI	UTMI+ Low Pin Interface
USB	Universal Serial Bus
UTMI	USB2.0 Transceiver Macrocell Interface

CHAPTER 1
INTRODUCTION

As the new Technologies emerge, USB leaders allow us to design smaller and faster testing circuits to test the chip and find the bugs. ULPI is designed by group of USB industry leader to address the need for high speed mode, low pin count and OTG PHY's. Using existing UTMI specification, ULPI is developed with reduced pin interface to 12 pins. Bit Error Rate setup is developed to analyze the different mode data transmission without burning the FPGA again and again. It provides the robust design to analyze the PHY board under different conditions.

This thesis deals with the development of bit error rate design to extract the quality figure of the PHY board. This design is to be implemented on the Virtex-II FPGA board. It has to work at 60Mhz clock coming from the external environment. Large chunk of memory is required to store the data packets. Even a single LUT in FPGA has the gate delay of 4ns. Routing delays are more prominent than the gate delays at higher technologies. So, It is very critical to meet the constraints for the robust design. Hence, benefits of FPGA resources are to be extracted to develop the design working at high frequency.

In Chapter2, Basic USB Communication protocol is defined. This chapter discusses the primary design goals of USB 2.0 like low power mode, hot pluggable etc. Chapter 3 briefly describes the USB2.0 Transceiver Macrocell Interface features and its interfacing signals to send data at low, full and high speed mode. Chapter 4 briefly describes the UTMI+ Low Pin Interface which lowers the pin count of UTMI at the interface level with the use of register set. In Chapter 5, UTMI to ULPI converter is described which can be used to test the PHY Board in different modes on the fly without burning the code again and again on the FPGA. In last Chapter, Bit Error Rate setup is defined which counts the number of error in the data transmission and calculate the Bit error rate to check the performance of the system.

CHAPTER 2
USB 2.0

2.1 Introduction

This chapter describes the design goals of USB [1] and introduces the concept of USB communication with different transmission rates. It supports low, full and high speed devices. USB employs NRZI encoding and differential signaling to transfer information across USB cables. Bit stuffing is also introduced in USB communication. It enables the receiver to maintain synchronization with the incoming data in case of long transmission of consecutive 1's. The USB environment supports a wide range of other signal-related functions such as: detecting device attachment and removal, suspending and resuming operation, resetting a device and others which are briefly introduces in this chapter.

2.2 Key Features of USB:

1. Low Cost:

The USB provides a low-cost solution for attaching peripheral devices to PCs. USB support attachment of 127 devices using Hubs.

2. Hot Pluggable:

USB supports hot plug and play feature. Device attachment is automatically detected by the USB and software automatically configures the device for immediate use without any need of user intervention.

3. Single Connector Type:

It has a single connector that can be used to attach any USB device. Additional connectors can be added to single USB port by using the USB hubs.

4. 127 Devices:

Each USB port supports the attachment of 127 USB devices.

5. Low Speed, Full Speed and High Speed Devices:

The USB 2.0 supports three device speeds:

a. Low Speed (1.5 Mbps)

b. Full Speed (12 Mbps)

c. High Speed (480 Mbps)

6. Cable Power:

Cable power is mainly 5.0Vdc. Peripherals can be powered directly from the cable. The current available can vary from 100mA-500mA depending on the hub port.

7. System Resource Requirement Eliminated:

USB devices do not require any memory or I/O address space and need no IRQ lines.

8. Error Detection and Recovery:

USB transactions include error detection mechanisms that are used to ensure that data is delivered without error. CRC and other error checking are performed to verify data delivery and if errors occur, the failed transmissions are retransmitted from the transmitter side. Data toggle mechanism is used to ensure that the transmitter and receiver remain synchronized throughout a long transfer requiring a large number of individual transactions. Data toggle is used to detect the corrupted handshake packets.

9. Power Conservation:

USB devices automatically enter the suspend state after 3ms of no bus activity. During suspend state [2], USB devices can consume no more than 500μA of current. There are other powers saving modes which are represented in paper [2].

2.3 USB Supported Transmission Data Rates

USB support three transmission data rates which are

• Low Speed: 1.5 Mbps

• Full Speed: 12 Mbps

• High Speed: 480 Mbps

The 1.0 and 1.1 (1.x) versions of USB Hub support only the Low Speed (1.5 Mb/s) and Full Speed (12Mb/s) while the 2.0 version of the USB specification support High Speed (480Mb/s) devices along with Low and Full Speed devices.

2.4 Applications

The USB devices for Low speed, Full speed and High speed have wide applications which are illustrated in table 2.1.

Table 2.1 Application of USB device data rates

Performance	Application
Low Speed : Interactive Devices	Keyboard, Mouse and Game Peripheral
Full Speed : Phone and Audio	ISDN, PBX, Printer, Scanner and Digital Audio
High Speed : Video, Disk and LAN	Mass Storage, Video Conferencing, Imaging and Broadband

2.5 Detecting Device Attachment and Speed Detect

Whenever any USB device is attached with the USB port, Host controller firstly detect the device attachment. USB Host Controller is designed to detect the attachment of USB devices to the USB port. After the attachment, transactions may be initiated by host software to configure the USB device for normal operation. USB hubs monitor each port to observe connect or disconnect event. Device attachment can only be detected when power has been applied to the port. Figure 2.1 illustrates a connection of Full or High speed device is attached to the Hub. Hub interface D+ and D- lines are connected to ground using pull down resistors of 15kΩ as shown in figure 2.1. When no device is attached, the single-ended receivers detect an electrical low on both data lines. USB devices must include a pull-up resistor on either D+ or D- (depending on its speed) to

enable connect detection. Full and High speed devices have pull up at its D+ line while Low speed devices has pull up at its D- line.

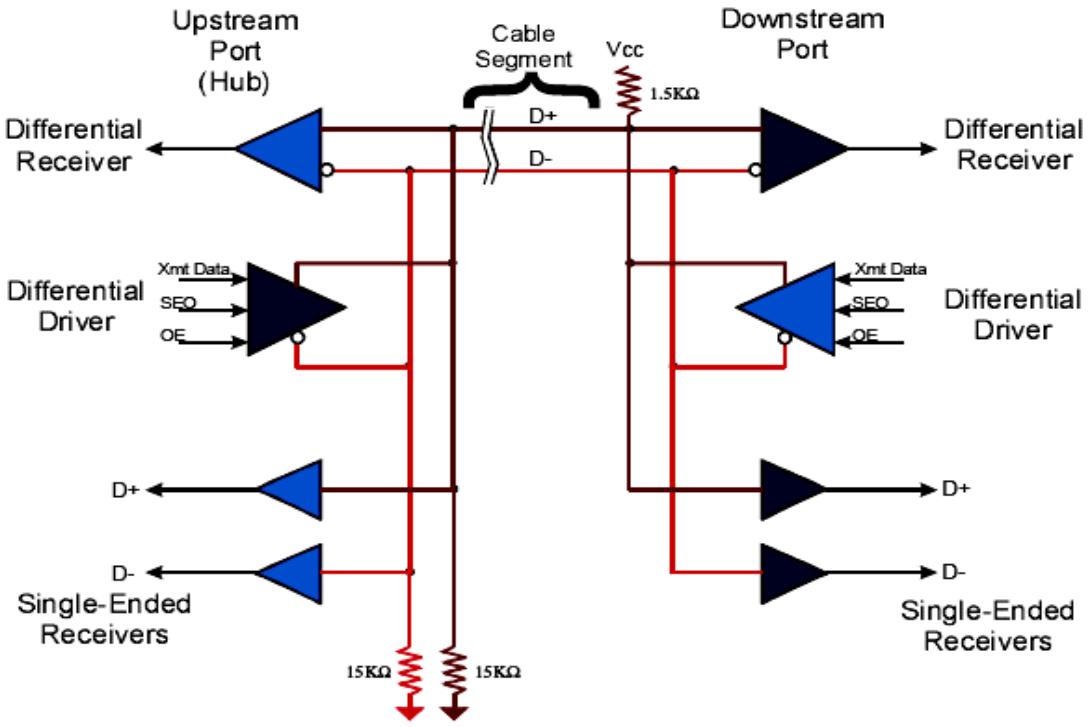

Figure 2.1 USB Hub Signaling Interface and attached USB Full-Speed Device [1]

When Hub detects any device attachment at its USB port then following timing events occurs as shown in a figure 2.2:

1. Δt1**:** It specifies the time required for a hub to apply valid power to a port once a device is attached to the Hub.
2. Δt2: It specifies the delay from port power valid till D+ (D-) is pulled above V_{IH} at the hub port receiver.
3. Δt3: This interval ensures that the signals are de-bounced. This de-bounce interval is enforced by software after which software can safely check hub status. It is used to determine whether a device is attached to the port just powered. The data lines will make and break the contact due to the connectors scraping together as the plug is inserted, thus causing the signals to bounce.

4. Δt4: After D+ or D- line has settled (during Δt3), the bus assumes the bus idle state until de-bounce interval get expired. During this time the device enters the suspended state after 3ms of bus idle.
5. Δt5: Software issues a Reset command to the hub which in turn signals RESET by driving D+ and D- on low for >10 ms and <20ms. This forces the device into its default state.
6. Δt6**:** This interval is called reset recovery time. The hub sets status when it is ready for access following reset. During this interval, the device will enter it's suspend state.

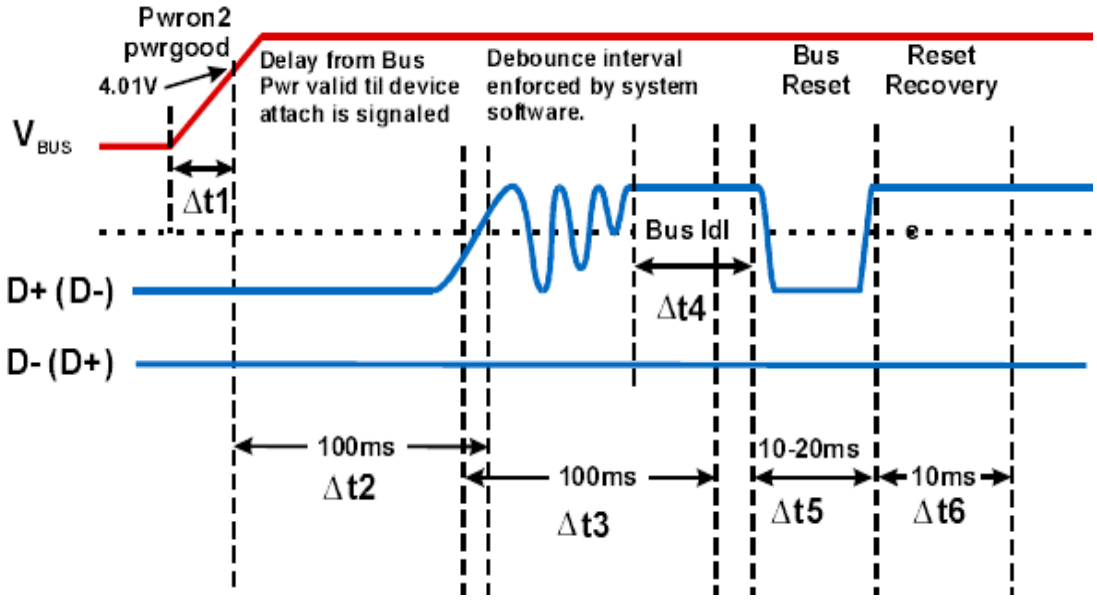

Figure 2.2 Connect Sequence from Port Power through Device Reset [1]

2.5.1 Full-Speed Device Connect

Figure 2.3 illustrates a FS device connected to a hub port.

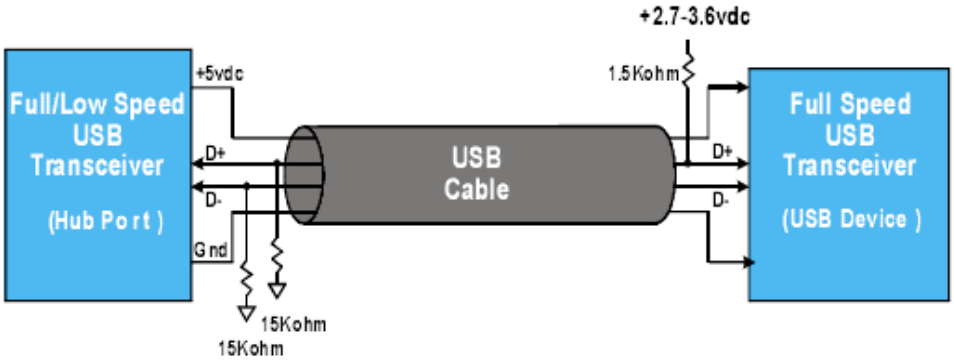

Figure 2.3 Full Speed Device Detection [1]

When a FS device is attached to the port, the current flows on D+ line from USB Device to the Hub. The voltage divider is created by the hub's pull-down resistor of 15KΩ and the device's pull-up resistor of 1.5KΩ on D+ line. Since the Hub's pull-down resistor value is 15KΩ and the USB device pull-up resistor value is 1.5KΩ, D+ will raise to approximately 90% of Vcc. When the hub detects that D+ approaches Vcc while the other line D- remains near ground, it knows that a full speed device has been attached as shown in a figure 2.4.

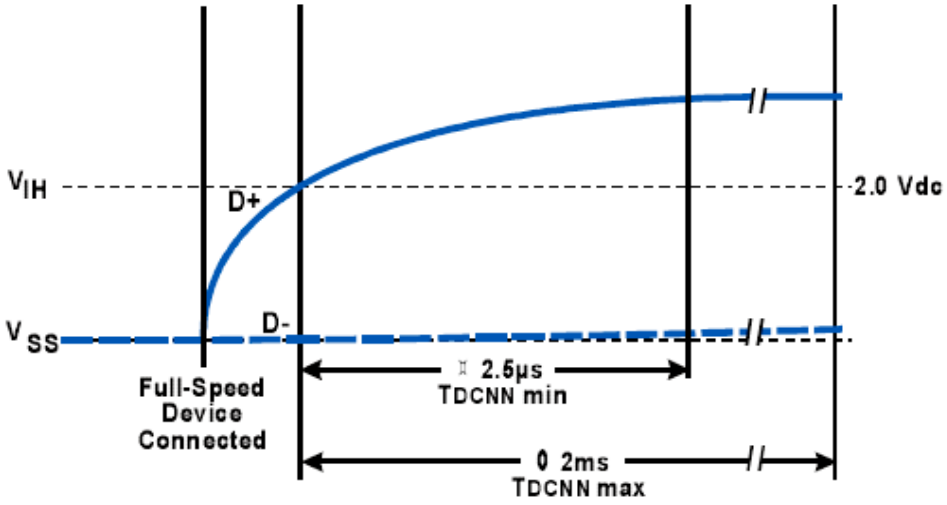

Figure 2.4 Signal States during FS Device Attachment [1]

2.5.2 Low-Speed Device Connect

Figure 2.5 illustrates a LS device connected to a hub port.

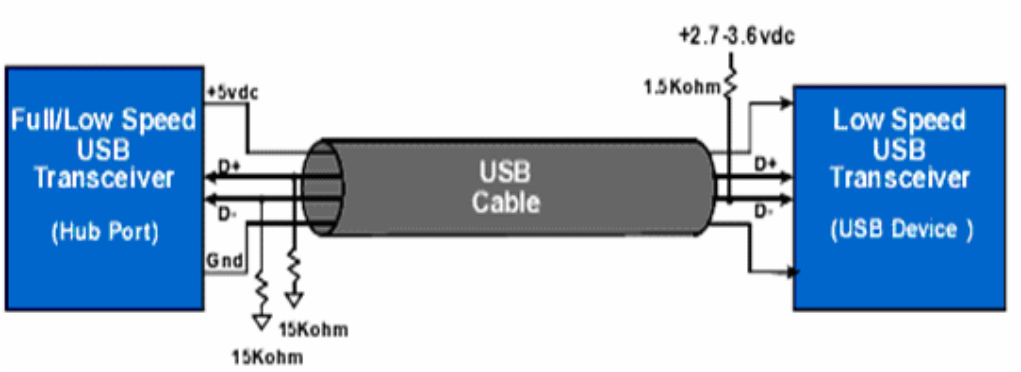

Figure 2.5 Low Speed Device Detection [1]

When LS device is attached to the port, the current flows on D- line from USB Device to the Hub. The voltage divider is created by the hub's pull-down resistor of 15KΩ and the device's pull-up resistor of 1.5KΩ on D- line. Since the Hub's pull-down resistor value is 15KΩ and the USB device pull-up resistor value is 1.5KΩ, D+ will raise to approximately 90% of Vcc. When the hub detects that D- approaches Vcc while the other line D+ remains near ground, it knows that low speed device has been attached as shown in a figure 2.6.

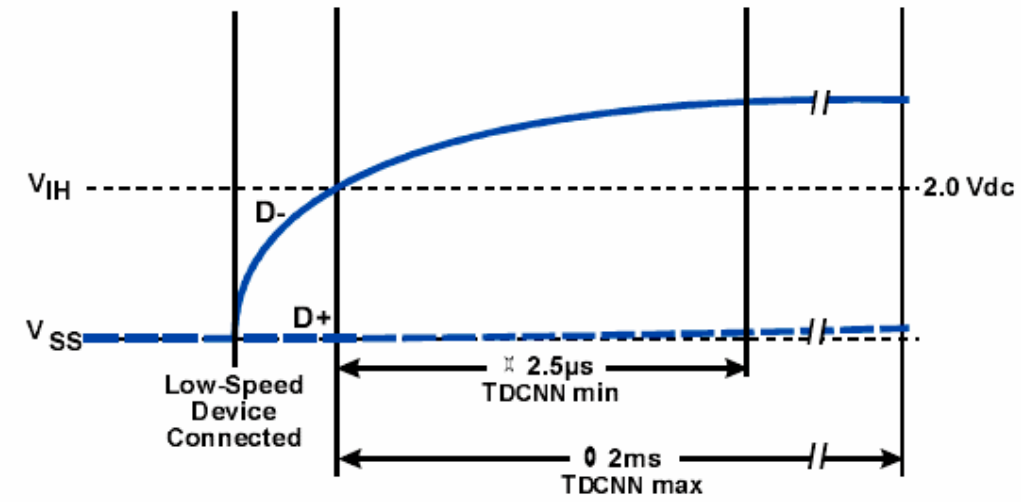

Figure 2.6 Line States during Low Speed Device Connection [1]

2.6 Detecting Device Disconnect

The hub always monitors its each port for the possibility of device being attached or removed. The device, that is currently connected, shows the transition on D+ and D- line on being disconnected. The transition that will be seen by the hub when a device is removed is shown in figure 2.7**.** A hub detects disconnect when it observes a single-ended zero (when both D+ and D- lines fall below V_{IL}).

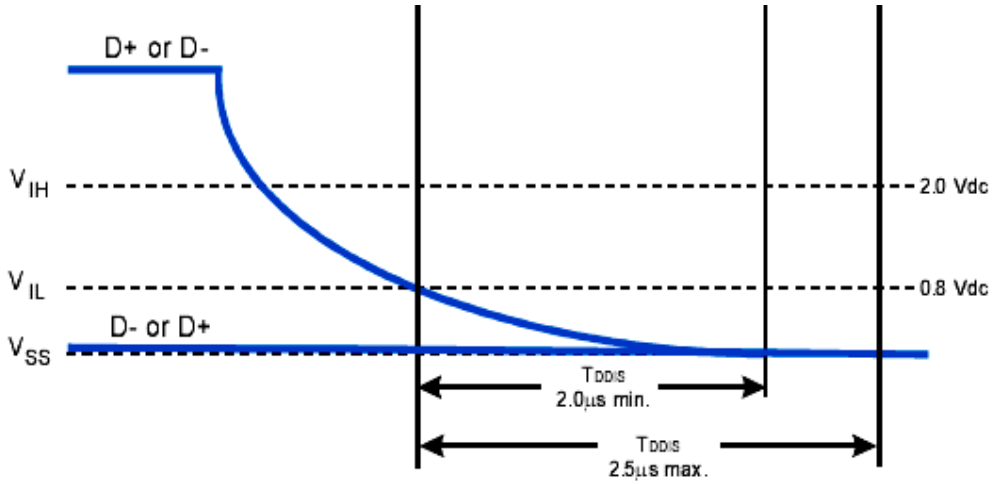

Figure 2.7 Signaling State during Device Disconnect [1]

2.7 Device RESET

When Host controller detects the device then it will reset the device firstly. This forces a device into its default state which is required prior to configuration. RESET is signaled by the hub port interface by driving a single-ended zero (SE0) as shown in figure 2.8. USB devices have two signaling states which are opposite for low- and full speed devices:

- J state
- K state

When a USB device is initially attached to the USB, one of its data lines i.e. D+ or D- is near to the Vcc while another one is near to the ground. This state of the USB device is known as the "J" signaling state and is also known as the idle state for the device. When a signal changes from high to low or vice versa, the two data lines changes to opposite values. This state of USB Device is known as "K" signaling state.

For High and Full speed devices, D+ is high and D- is low for J signaling state while D- is high and D+ is low for K signaling state. For Low speed device, D- is high while D+ is low for J signaling state and D+ is high and D- is low for K signaling state.

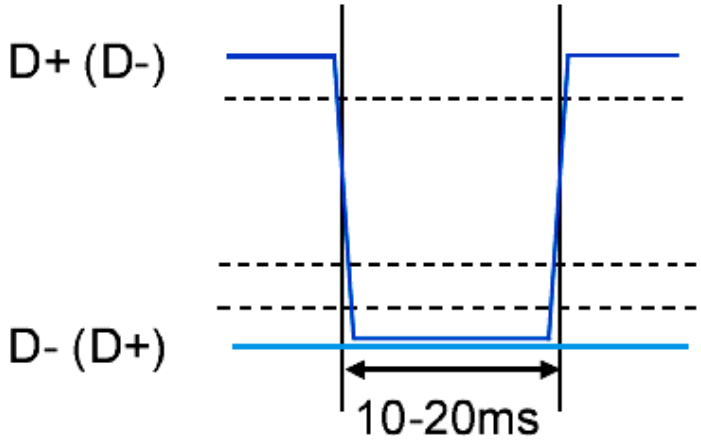

Figure 2.8 Reset Signaling States [1]

2.8 Start of Packet

When a USB Device is attached to any Hub, it will send the information in form of packets. Each packet begins with a synchronization sequence of 8 bits which is known

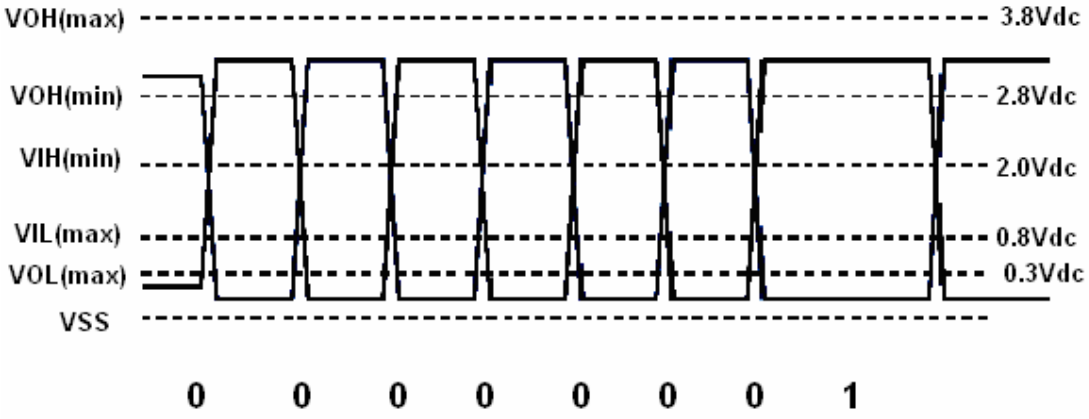

Figure 2.9 SOP is recognized at the start of the Synchronization Sequence [1]

as Start of Packet (SOP). This synchronization sequence is used to synchronize receiver clock with the transmitter clock. It allows a Phase Lock Loop (PLL) or a Delay Lock Loop (DLL) at receiver side to establish the synchronization with the incoming packet. When USB Device is attached to the Hub, USB device and Hub interface signals are in their idle state (J state). When the first packet arrives, the interface signal will changes from J state to the K state which is the indication of an incoming packet. Figure 2.9 illustrates the synchronization sequence and the SOP.

2.9 End of Packet

End of packet is signaled by a single ended zero (SE0) i.e. both D+ and D- lines are at low level. At the end of packet, the transmitter drives SE0 for 2 bit times (low or full speed) to signal EOP and the single ended receivers detect EOP. Figure 2.10 illustrates EOP signaling. At end of packet, both D+ and D- lines are zero for 2 bit times only in low and full speed USB devices.

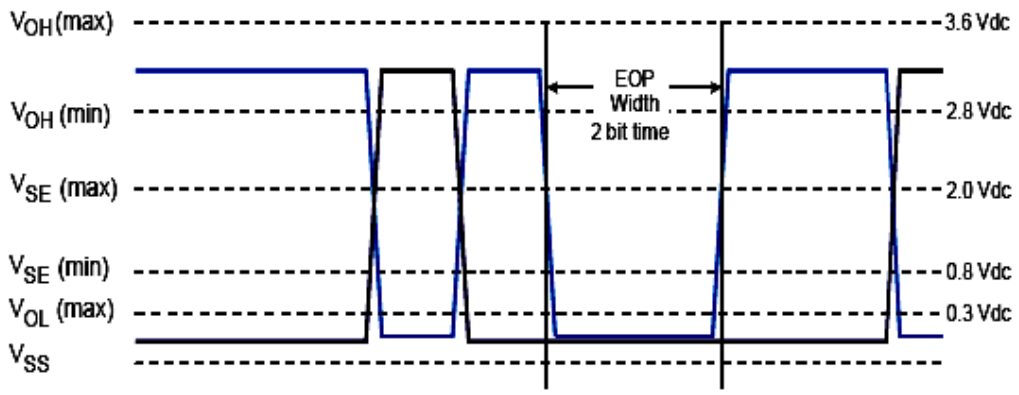

Figure 2.10 Full or Low Speed EOP signaling [1]

2.10 NRZI Encoding

Data packets are encoded first before sending to the receiver side. USB data packets are encoded using NRZI (Non-Return to Zero, Inverted). In NRZI encoding, 0's in the NRZI data stream are represented by transitions while 1's are represented by the absence of a transition. Transitions in the data stream used to maintain synchronization at receiver decoder with the incoming data. So it eliminates the need for a separate clock signal in the USB cable. However, a long string of consecutive 1s results in no transitions

which causes lost of synchronization at receiver end. The solution of this problem is to combine bit stuffing with the NRZI encoding.

2.11 Bit Stuffing

Bit stuffing is used to forcibly send the transition when it met six consecutive 1's in the USB data transmission as illustrated in figure 2.11. This is used to ensure that the receiver detects a transition in the USB data stream. This helps the receiver to maintain synchronization with the incoming data as at least one transition will occur at seventh bit.

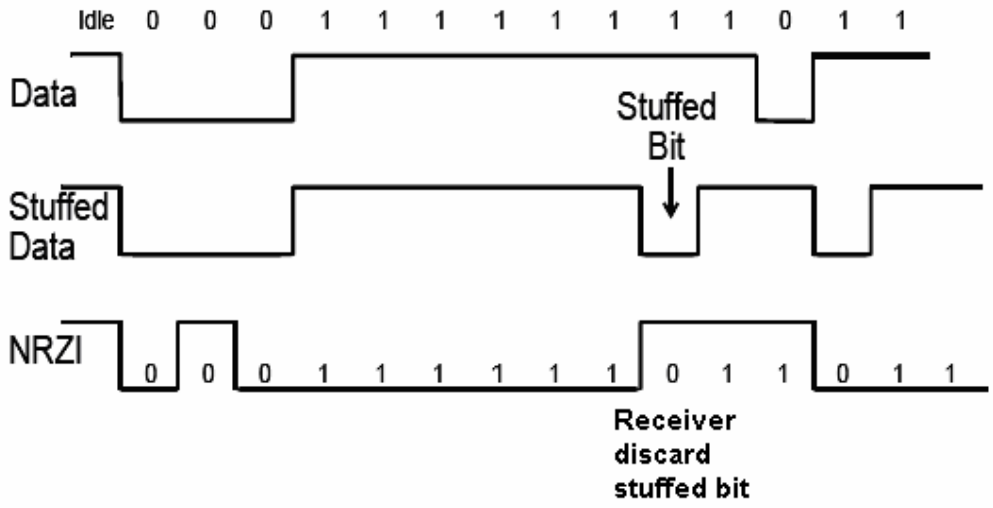

Figure 2.11 Stuffed bit in NRZI encoded signal [1]

Bit stuffing is done at the transmitter side and receiver is designed to automatically discard the seventh stuffed bit (0) that immediately follows the sixth consecutive 1. If there are seven consecutive 1's in bit stuffing mode then there is an error and no acknowledgment is send by the receiver. Figure 2.11 shows how to insert the stuffed bit in the data stream. Here data stream contains the string of eight consecutive ones. In stuffed data, stuffed bit (0) is inserted after consecutive six ones. After that NRZI encoder encodes the stuffed data. Receiver ignores the stuffed bit as it already knows that bit following the six consecutive 1's will be a stuffed bit (0). It may cause delay in delivery of data by one bit time at receiver end. If there are six consecutive ones followed by zero in the data stream, even then stuffed bit (0) is inserted after six ones which results in two consecutive zeros in the stuffed data stream.

2.12 Detecting High Speed Device Attachment

The 1.0 and 1.1 (1.x) versions of USB Hub can support only the Low Speed (1.5 Mb/s) and Full Speed(12Mb/s) while high speed hubs 2.0 support low speed, full speed as well as high speed devices. When high speed device is connected to the high speed hub then handshaking is performed.

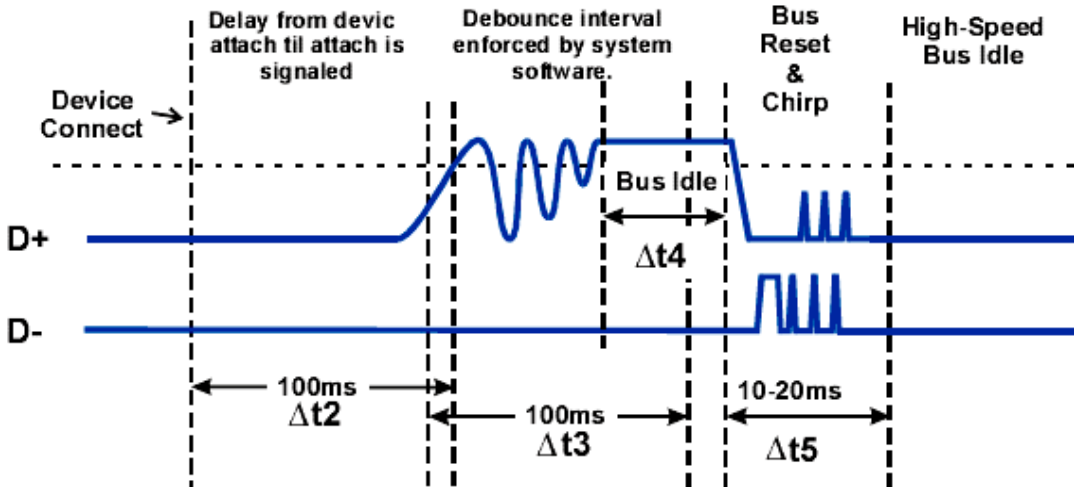

Figure 2.12 Sequence of Events from Device Connect to High Speed Operation [1]

High-speed hub ports must be able to detect when a low-speed, full-speed or high-speed device has been connected to the port. High-speed devices initially appear as full-speed devices when attached to a high-speed port. The high speed port and the high-speed devices must then perform a handshake to identify the high-speed device. If the handshake fails, the high-speed device works in full-speed operation. The sequence of events is illustrated in figure 2.12. This handshaking process is same as when a full-speed device is attached except the chirp sequence that occurs during device RESET. This chirp signal is transmitted by device to inform the controller that device attached to the hub is high speed device. Thus, chirp sequence is used to help controller to differentiate between full and high speed devices.

2.13 Device Reset and the Chirp Sequence

When high speed device is attached with the host controller, the host controller firstly identifies high speed device as a full-speed device (i.e. D+ signaling line goes high

while D- line is low). Host controller then issues RESET to the hub via the Reset command which causes the hub to drive a single ended zero (i.e. both D+ and D- lines are low) for >10ms. If a high-speed capable device is attached, the chirp sequence is transmitted by the device. Figure 2.13 shows the chirp sequence. Each step in the chirp sequence is explained below:

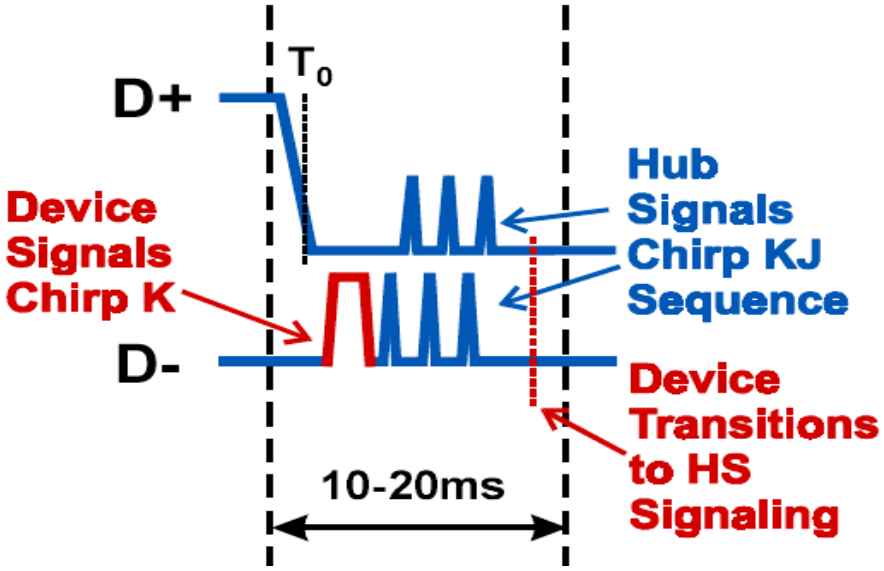

Figure 2.13 Chirp Sequence used to detect High Speed Capable Device [1]

1. Firstly Hub drives RESET on the USB signaling lines which causes single ended zero on both D+ and D- lines. It is shown as T0 in the figure 2.13.
2. The high-speed device detects single ended zero and transmits a Chirp K. Chirp K is signaled by the device via the high-speed current driver by sourcing the current into the D- signaling line. This Chirp K is signaled 1ms to 7ms after T0.
3. During RESET, the hub's high-speed receiver is activated and waits for the detection of Chirp K. The high-speed hub must detect a valid Chirp K. If Hub does not detect chirp from the device during RESET period, it completes the full-speed RESET and work in full speed signaling mode.
4. If Hub detects the chirp K from the device during RESET period, the hub will return an alternating sequence of Chirp K's and Chirp J's.
5. When USB device detects six chirps (3 KJ pairs), it must work as a high speed device within 500µs. This transition requires:
   - disconnecting the pull-up resistor from D+.

- enabling the high-speed terminations.
- entering the high-speed default state.

6. After RESET completes, Hub continues to drive single ended zero on D+ and D- signaling lines via its full-speed drivers.

2.14 High Speed Start of Packet

A synchronization sequence is transmitted at the beginning of each packet so that the receiver can synchronize its clock according to the incoming packet synchronization sequence. This synchronization sequence starts with the K transition and consists of a series of K to J and J to K transitions as illustrated in figure 2.14. The duration of this synchronization sequence is 32 bit times. It is originated by the host or responding device.

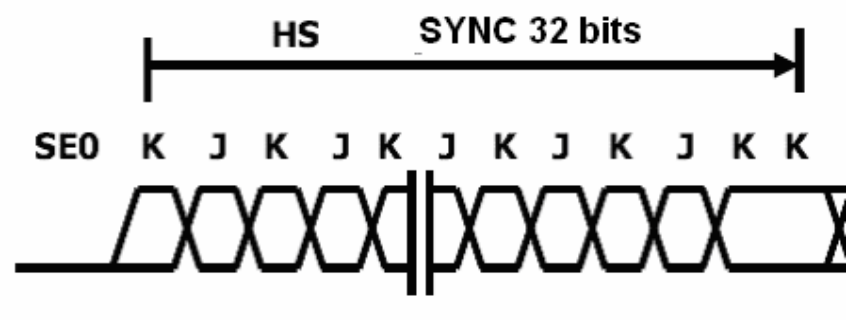

Figure 2.14 High Speed Start of Packet & Synchronization Sequence [1]

When packet is received by device, the synchronization sequence may be shorter than 32-bit. This can occur as packets come across a hub. Each hub drops up to 4 bits from the synchronization pattern before retransmitting the packet ahead. These 4 bits are dropped as there is a delay between detection of packet by the envelope detector and enabling the receiver. Due to this, maximum number of hubs between the host and device will be 5. Thus, a maximum of 5 hub crossings may result in a sync pattern containing only 12 bits at the receiving device.

2.15 High Speed End of Packet

Every data packet ends with an EOP sequence as illustrated in Figure 2.15. The EOP sequence is mainly 8 bits long except after the Start of Frame packet where the EOP

is extended up to 40 bits. EOP consists of an NRZI encoded bit pattern of 01111111 without bit stuffing. When receiver detects the intentional bit stuff error, it comes to know about the end of packet.

In High speed mode, zero indicates toggling from the K state to J state and J state lasts for seven clock cycles. After end of packet, D+ and D- lines returns to SE0 and receiver is squelched as shown in figure 2.15. Electrical specification of low speed, full speed and high speed modes during its various stages are provided in USB specification [3].

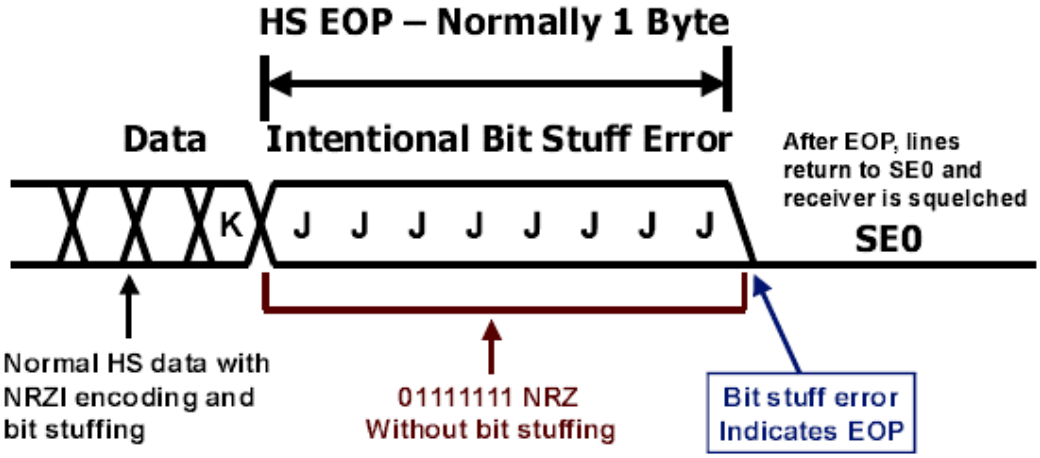

Figure 2.15 High speed EOP Detection [1]

CHAPTER 3
UTMI

3.1 Introduction

UTMI [4] stands for USB2.0 Transceiver Macrocell Interface. High speed USB2.0 devices are designed using ASIC technology. USB 2.0 support High speed devices which work at 480 Mbps. With USB2.0 running at high speed, the existing methodology has to be changed. UTMI defines an interface to which ASIC and peripheral vendor can develop their codes to support high speeds. ASIC vendors can develop their device libraries and designs to support high speed, thus reducing the time and risk of their development cycles. It can be implemented on the FPGA and enhance its performance by optimization [5].

3.2 UTMI Features

UTMI block handles low level protocol and signaling. This interface is used for data serialization and de-serialization, bit stuffing, clock recovery and synchronization. This block primary focus on shifting of clock domain of data from USB2.0 to one that is compatible with the ASIC vendor logic design.

Some key features of UTMI [4] are:

1. Eliminate high speed USB2.0 logic design from the peripheral developer.
2. It is a standard transceiver interface helps to interact with the multiple IP source.
3. It supports High speed (480Mbps), Full speed (12Mbps) and Low speed (1.5Mbps) serial data transmission.
4. Use 8 bit interface to transmit and receive USB 2.0 device data.
5. Synchronization and End of packet pattern generation and checking.
6. Data and Clock recovery from serial data from the USB device.
7. Bit stuffing and un-stuffing is executed in this block.
8. Holding registers are used to convert serial USB data into parallel UTMI data and vice versa.
9. Logic to facilitate Resume signaling

10. Ability to switch between Full speed and High speed signaling.

The UTMI supports High speed, Full speed and low speed mode. A vendor can choose the transceiver performance according to their need. The USB2.0 Transceiver can be placed in low power mode with SuspendM signal.

The functional block of USB2.0 Transceiver is shown below [4]:

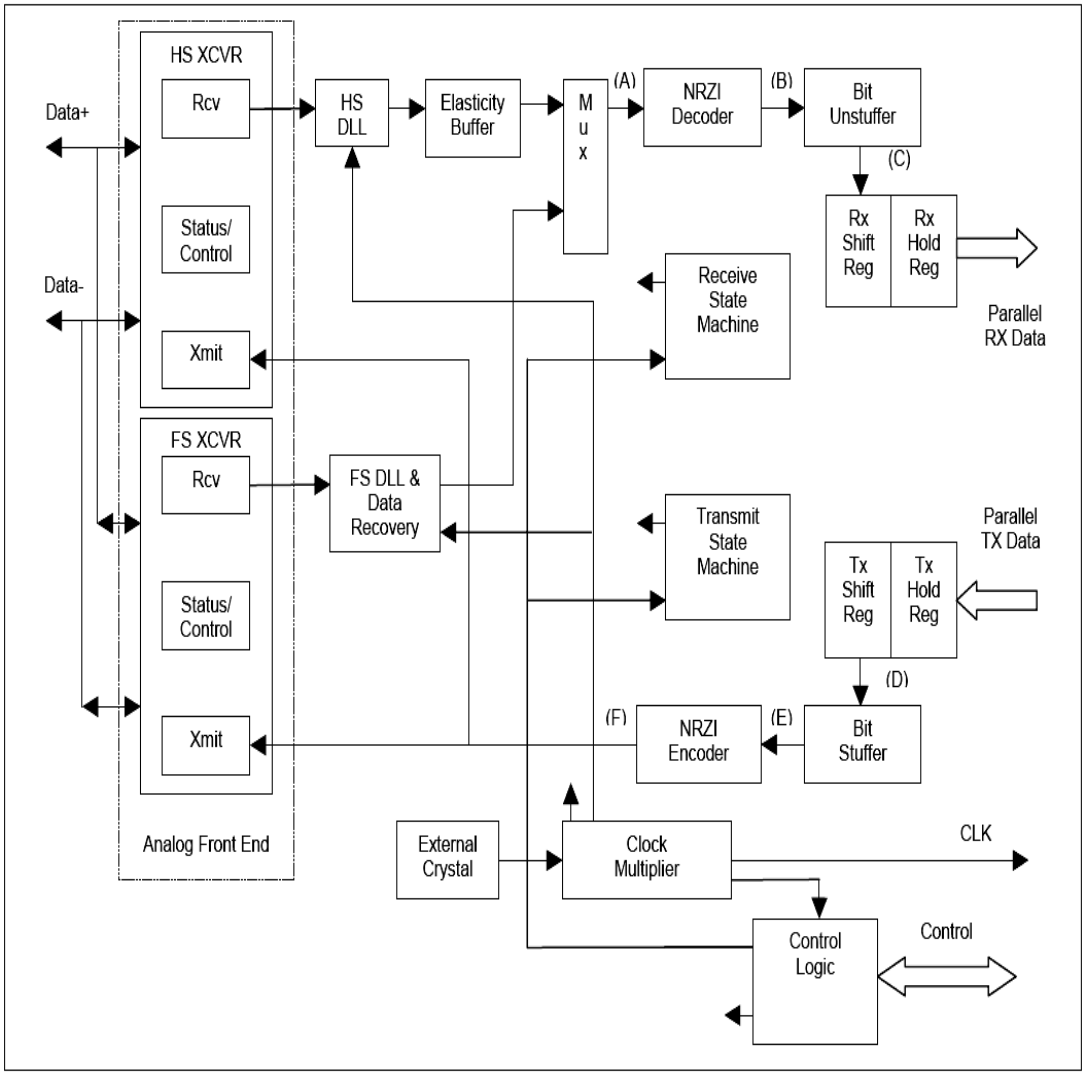

Figure 3.1 UTMI Functional Block Diagram [4]

UTMI has hundred interface signals. Many of these signals are static and remain constant for one mode of operation. Important signals of UTMI used for transmission of the data from ULPI interface to USB port are:

1. **CLK :**

   This output signal is used for clocking transmit and receive parallel data.

   60 MHz for HS/FS with 8 bit interface

   30 MHz for HS/FS with 16 bit interface

   48 MHz for FS only with 8 bit interface

   6 MHz for LS only with 8 bit interface

2. **RESET:**

   It resets all state machines in the UTMI block.

3. **XcvrSelect:**

   This signal selects between the FS and HS transceivers.

   0: HS transceiver enabled

   1: FS transceiver enabled

4. **Term Select:**

   This signal selects between the FS and HS terminations.

   0: HS termination enabled

   1: FS termination enabled

5. **SuspendM:**

   It places the Macrocell in a mode that draws minimal power from supplies. It shuts down all blocks not necessary for Suspend/Resume operation. While suspended, TermSelect must always be in FS mode to ensure that the 1.5K pull-up on DP remains powered.

   0: Macrocell circuitry drawing suspend current

   1: Macrocell circuitry drawing normal current

6. **LineState(0-1):**

   These signals reflect the current state of the single ended receivers. They are combinatorial until a "usable" CLK is available then they are synchronized to CLK. They directly reflect the current state of the DP (LineState[0]) and DM (LineState[1]) signals:

DM	DP	Description
0	0	0: Single Ended Zero
0	1	1: 'J' State
1	0	2: 'K' State
1	1	3: Single Ended One

7. OpMode(0-1):

These signals select between various operational modes.

Opmode[1]	Opmode[0]	Description
0	0	0: Normal Operation
0	1	1: Non-Driving
1	0	2: Disable Bit Stuffing and NRZI encoding
1	1	3: Reserved

8. DataIn(0-7) :

This is an 8-bit parallel USB data input bus.

When DataBus16_8 = 1, this bus transfers the lower bytes of 16-bit transmit data.

When DataBus16_8 = 0, this bus transfers all the data.

9. DataIn(8-15):

This is an 8-bit parallel USB data input bus that transfers the high byte of 16-bit transmit data. These signals are only valid when DataBus16_8 = 1.

10. TxValid (Transmit Valid):

This is an active high input signal. TxValid indicates that the DataIn bus is valid. The assertion of Transmit Valid initiates SYNC pattern on the USB. The negation of Transmit Valid initiates End of Packet on the USB. In High speed (XcvrSelect = 0) mode, the SYNC pattern must be asserted on the USB between 8 and 16 bit times after the assertion of TxValid is detected by the PHY. In FS (XcvrSelect = 1), FS Only or LS Only modes, the SYNC pattern must be asserted on the USB not less than 1 clock cycle and not more than 5 to 10 clock cycles after the assertion of TxValid is detected by the PHY.

11. TxValidH:

This is an active high input signal. When DataBus16_8 = 1, this signal indicates that the DataIn (8-15) bus contains valid transmit data. This signal is ignored when DataBus16_8 = 0. This signal is not provided in 8-bit transceiver implementations.

12. TxReady:

This is an active high output signal. If TxValid is asserted, the Link must always have data available for clocking in to the Transmit Hold Register on the rising edge of clock. If TxValid is valid and TxReady is asserted at the rising edge of CLK, the UTMI will load the data on the DataIn bus into the Transmit Hold Register on the next rising edge of CLK. At that time, Link should immediately present the data for next transfer on the DataIn bus. If TxValid is asserted and TxReady is negated, the Link must hold the previously asserted data on the DataIn bus.

These are the important signals required for the High speed data transmission. There are many other UTMI signals available which are assigned for different tasks are given in the UTMI specification [4].

CHAPTER 4
ULPI

4.1 Introduction

This section delineates a UTMI+ low pin interface (ULPI) between a Link and a PHY. Its Interface signals are illustrated in figure 4.1 and the basic communication protocol is described in this chapter [6]. The block diagram of ULPI PHY is shown below. ULPI transceiver contains the UTMI interface.

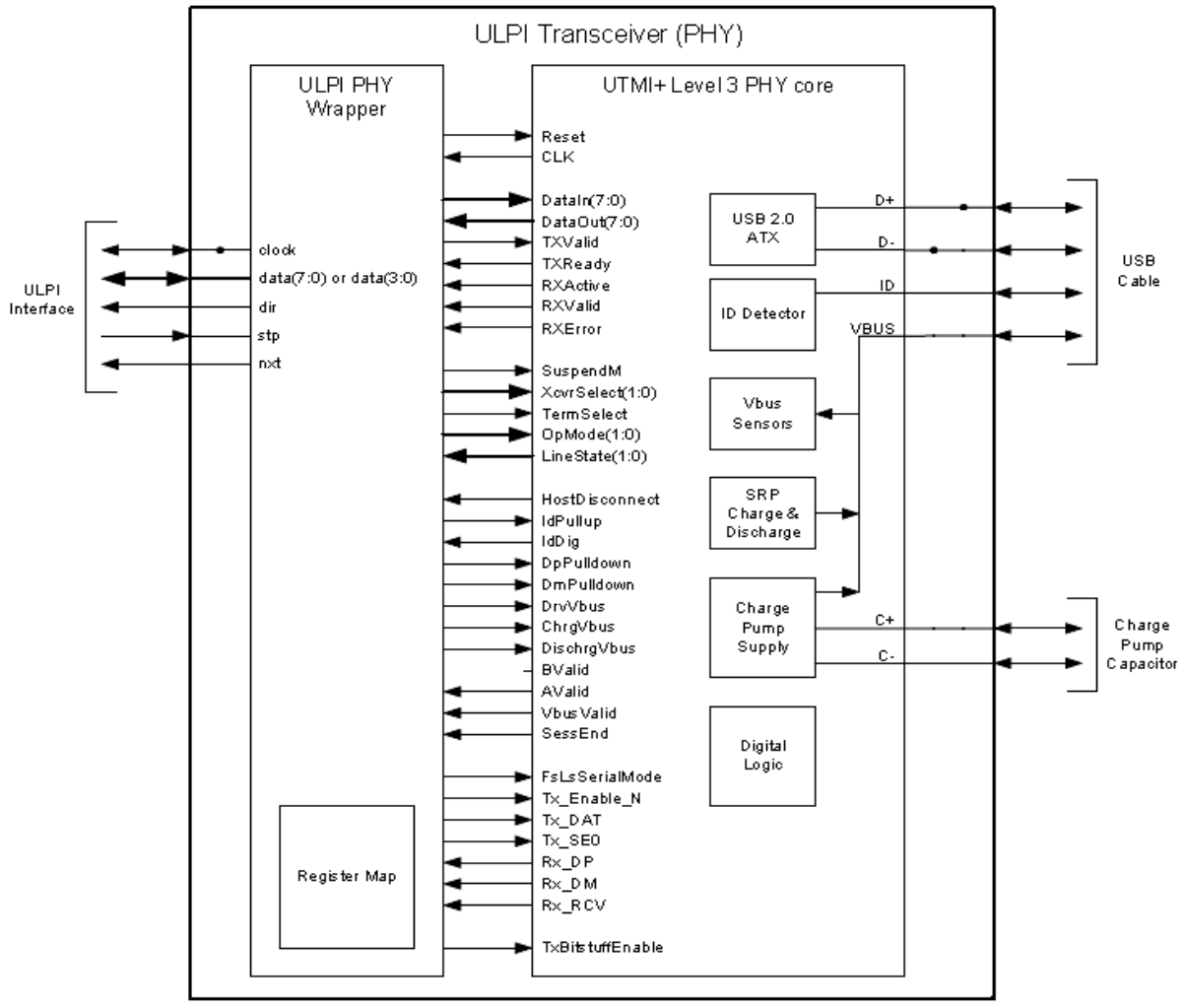

Figure 4.1 Block diagram of ULPI PHY [6]

4.2 UTMI vs ULPI Interface

This section describes how any UTMI + core is converted to a smaller LPI interface which significantly reduces pin count. ULPI is an interface of 8 or 12 signals that may be used to connect to an external transceiver that may be based on UTMI + specification. This lower pin count is achieved by driving all static signals by registers and by providing a bi-direction data bus that carries USB data and provides a means of accessing register data on the ULPI transceiver.

Infect, a ULPI PHY interface is based on the UTMI + core. As shown in a figure 4.2, Link and PHY based on this concept are implemented as a transparent wrapper around the existing UTMI + core. ULPI only reduces the pin count leaving all the functionality intact. This is one of the implementation. This does not imply that other implementations are not possible.

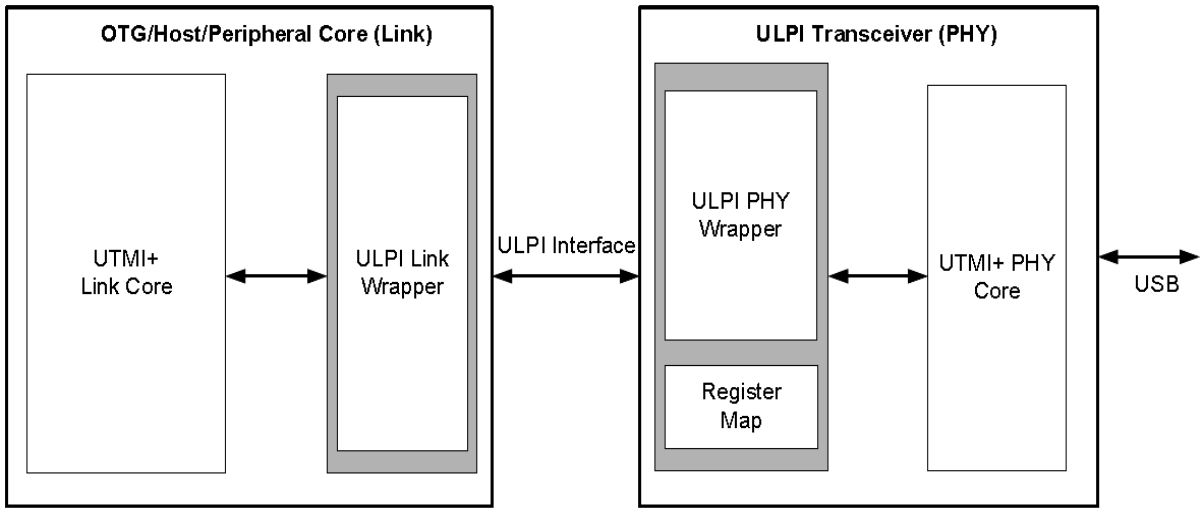

Figure 4.2 Creating a ULPI system using wrappers [6]

4.3 Interface Signals

Its transceiver interface signals are described briefly in Table 4.1. Its control signals DIR, NXT and STP are specified with the assumption that PHY is the master of the interface but not the Link. An implementation can be done with Link as the master. If Link is working as the master of the interface then control signal direction and

protocol must be reversed. Depending upon the application, data lines can be used to transmit or receive the packets.

Table 4.1 ULPI Interface Signals [6]

ULPI Interface		
Signal	**Direction**	**Description**
clock	I/O	All interface signals are synchronous to clock.
data	I/O	It is a bi-directional data bus which is driven low by the link during the idle state. Bus ownership is determined by DIR.
DIR	Out	It controls the direction of the data bus. When PHY has data to transfer to the Link, it drives DIR high to take the ownership of the bus. When PHY has no data to transfer, it drives the DIR low and monitors the bus for the Link activity.
STP	In	The Link asserts STP for one clock cycle to stop the data stream currently on the bus. If Link is sending the data to the PHY, STP indicates last byte of the data was on the bus in the previous cycle.
NXT	Out	PHY asserts this signal to throttle the data when Link is sending the data to the PHY. NXT indicates when the current byte has been accepted by the PHY.

4.4 ULPI Protocol

This protocol is enhanced version of UTMI. ULPI is developed from the UTMI interface with advantage of low pin count. All the static signals of UTMI are controlled with the help of ULPI register set.

4.4.1 ULPI Bus ownership

PHY is the master of the LPI bidirectional data bus. Ownership of the data bus is determined by the DIR control signal from PHY as shown in figure 4.3. If DIR is high, PHY takes the control of bus and can transmit data. If DIR is low, Link takes the control of the bus and can transmit the data. A change in DIR causes the turnaround cycle, during that neither PHY nor Link can drive the bus. Both PHY and Link should ignore any data on the bus because data present on the bus is always undefined. The DIR signal can be used to directly control the data output buffers of both PHY and Link.

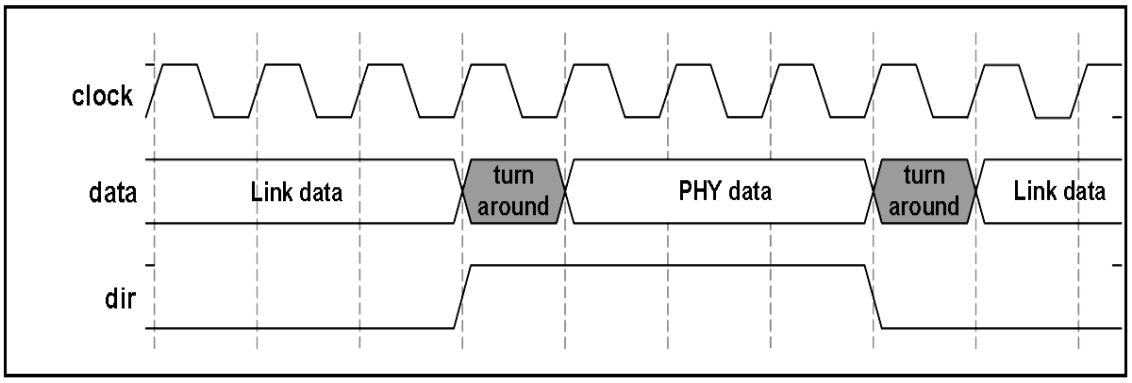

Figure 4.3 ULPI data bus ownership [6]

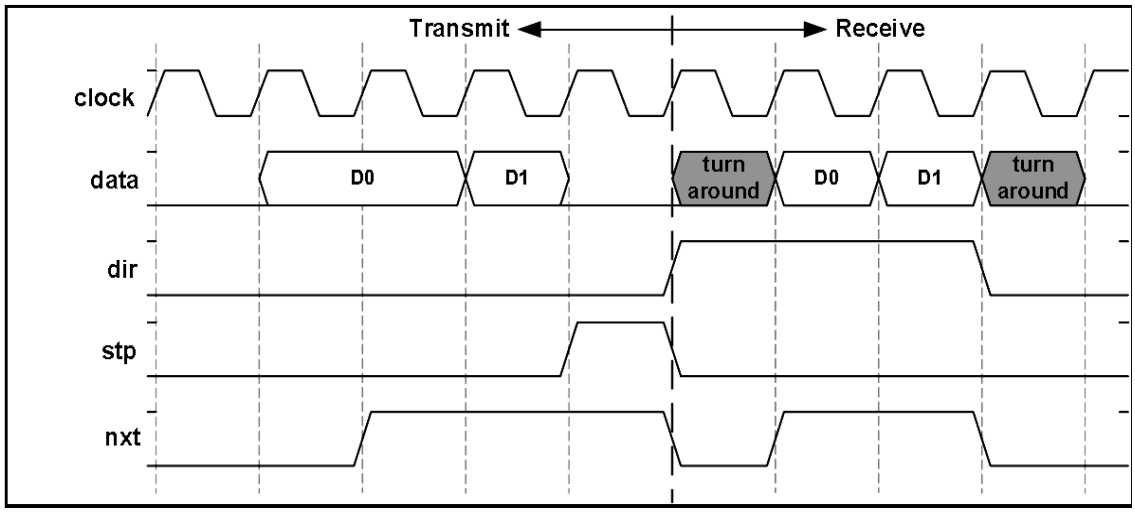

Figure 4.4 ULPI data transmit followed by data receive [6]

4.4.2 Transferring Data

During the first half of the figure 4.4, Link continuously drives the data bus. At the end of the data transmission, Link asserts STP in the cycle followed by last data byte indicating stop of the transmission. During the second half of the figure 4.4, PHY asserts DIR and take control of the data bus. It can transmit data only after the turn around cycle. PHY asserts DIR high when it has some data to be transmitted. PHY use control signals NXT to throttle the data during transmit as well as during receive. NXT may be asserted in the same cycle that the STP is asserted by Link.

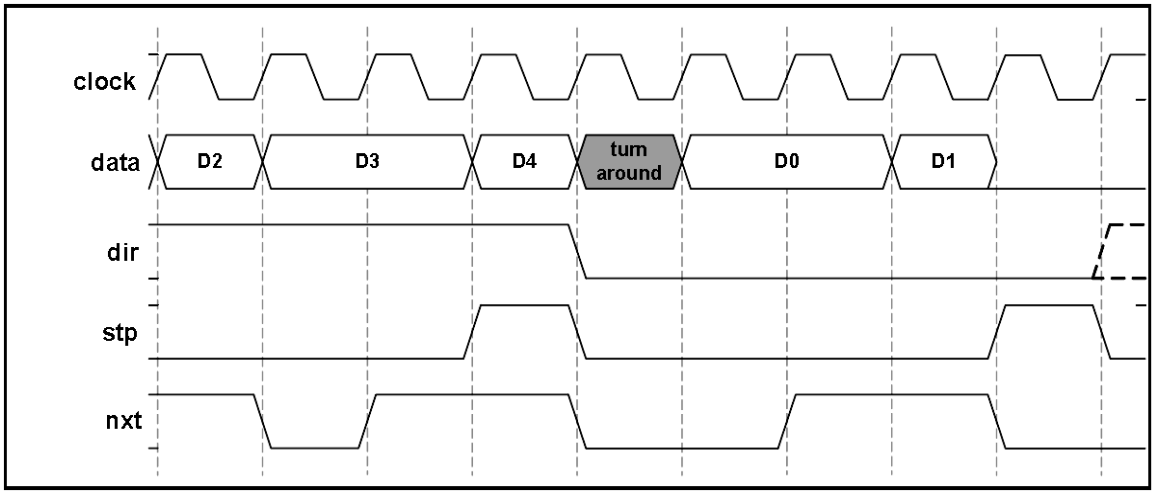

Figure 4.5 Link asserts STP [6]

4.4.3 Aborting Data

The PHY may assert DIR high to stop any data being transmitted by the Link. If Link wants to stop data receive from PHY, it asserts STP as shown in figure 4.5. So, PHY unconditionally negate DIR and start receiving data from Link. PHY may re-assert DIR to again take control of the data bus when Link finishes data transmission. Hence PHY can abort the data by asserting DIR while Link can abort the data by asserting STP. There is no transmission of packet data during these conditions.

4.5 ULPI Registers

ULPI provides an immediate register set with a 6-bit address that forms part of the Transmit Command Byte. An extended register set is also provided with an 8-bit address that requires an extra clock cycle to complete. The immediate register set is mirrored into the lower end of the extended address space. That is, reading or writing to extended address "00XXXXXX" will in fact operate on the immediate register set. The PHY must support both immediate and extended register operations. The register set is used to configure the PHY in low, full or high speed operation.

4.5.1 Function Control Register

It controls UTMI functions settings of the PHY. It is used to enable High speed, Full speed or Low speed transceiver and its termination point. It is also used to select the operation modes of the USB communication. It is explained in table 4.2.

Table 4.2 Function Control Register

Field name	Bit	Reset	Description
XcvrSelect	1:0	01b	Selects the required the transceiver speed 00b : enables HS transceiver 01b : enables FS transceiver 10b : enable the LS transceiver 11b: enables the FS transceiver for LS packets.
TermSelect	2	0b	This signal selects between the FS and HS terminations: 0: HS termination enabled 1: FS termination enabled
OpMode	4::3	00b	These signals select between various operational modes: [1] [0] Description 0 0 0: Normal Operation 0 1 1: Non-Driving 1 0 2: Disable Bit Stuffing and NRZI encoding 1 1 3: Reserved

Reset	5	0b	Reset all state machines in the UTMI + core.
SuspendM	6	1b	Places the Macrocell in a mode that draws minimal power from supplies. Shuts down all blocks not necessary for Suspend/Resume operation. While suspended, TermSelect must always be in FS mode to ensure that the 1.5K pull-up on DP remains powered. 0: Macrocell circuitry drawing suspend current 1: Macrocell circuitry drawing normal current
Reserved	7	0b	Reserved

4.5.2 Interface Control Register

It enables the alternative interfaces and PHY features as illustrated in table 4.3. All bits in this register are optional features of the PHY. It is mainly used to select the interface at the ULPI. FsLsSerial mode, Low power mode and Carkit mode are selected by this register. It is explained below:

Table 4.3 Interface Control Register

Field name	Bit	Reset	Description
6 pin FsLsSerial Mode	0	01b	0b: FSLS packets are sent using parallel interface 1b: FSLS packets are sent 6 pin using serial interface
3 pin FsLsSerial Mode	1	0b	0b: FS/LS packets are sent using parallel interface 1b: FS/LS packets are sent using 4 pin serial interface
Carkit Mode	2	00b	0b: Disable serial carkit mode 1b: Enable serial carkit mode
ClockSuspendM	3	0b	0b: Clock will not be powered in serial and carkit modes 1b: Clock will be powered in serial and carkit modes
AutoResume	4	1b	Enables the PHY to automatically transmit the resume signaling
Indicator Complement	5	0b	0b: PHY will not invert ExternalVbusIndicator 1b: PHY will invert ExternalVbusIndicator

Indicator Pass Through	6	0b	0b: Complement output signal is qualified with the internal Vbus valid comparator 1b: Complement output signal is not qualified with the internal Vbus valid comparator
Interface Protect Disable	7	0b	0b: Enables the interface protect circuit 1b: Disable the interface protect circuit

4.5.3 OTG Control Register

OTG Register is used to controls UTMI+ OTG functions of the PHY as illustrated in table 4.4. Signals defined by this register are explained below:

Table 4.4 OTG Control Register

Field name	Bits	Reset	Description
Idpullup	0	0b	0b: Disable sampling of ID line 1b: Enable sampling of ID line
Dppulldown	1	1b	0b: Pull down resister not connected to D+ 1b: Pull down resistors connected to D+
Dmpulldown	2	1b	0b: Pull down resister not connected to D- 1b: Pull down resistors connected to D-
Dischrgvbus	3	0b	0b: Do not discharge Vbus 1b: Discharge Vbus
Chrgvbus	4	0b	0b: Do not charge Vbus 1b: Charge Vbus
drvvbus	5	0b	0b: Do not drive Vbus 1b: Drive 5V on Vbus
Drvvbusexternal	6	0b	0b: Drive Vbus using internal charge pump 1b: Drive Vbus using external supply
Useexternalvbus indicator	7	0b	0b: Use internal Vbus valid indicator 1b: Use external Vbus valid indicator.

4.6 Transmit Command

The link always initiates the data transfer to the PHY by sending the Transmit Command (Txcmd). Txcmd contains the information about operation to be performed (Transfer data, Register write or Register read) and type of data to be send (Register data, Register address or packet payload). It is explained in table 4.5 shown below:

Table 4.5 Transmit Command

Byte name	Command code[7:6]	Command payload data [5:0]	Command description
Special	00b	000000b (NOOP)	No operation
		XXXXXXb (Reserved)	Reserved command space
Transmit	01b	000000b (NOPID)	Transmit USB data that does not have PID data such as chirp and resume signaling
		00XXXXb (PID)	Transmit USB packet data indicates USB packet identifier
		XXXXXXb (Reserved)	Reserved command space
RegWrite	10b	101111b	Extended register write command 8-bit address available in next cycle
		XXXXXXb	Register write command with 6 bit immediate address
RegRead	11b	101111b	Extended register read command 8-bit address available in next cycle
		XXXXXXb	Register read command with 6 bit immediate address

4.7 Receive Command

Receive Command (Rxcmd) is send by the PHY to update the Link with the information of Linestate, USB receive, disconnect and OTG status information. Rxcmd is explained in table 4.6 given below:

Table 4.6 Receive Command

<table>
<tr><th>Data</th><th>Name</th><th colspan="5">Description and value</th></tr>
<tr><td>1:0</td><td>Line State</td><td colspan="5">UTMI + Line State signals
Data(0) = LineState(0)
Data(1) = LineState(1)</td></tr>
<tr><td rowspan="6">3:2</td><td rowspan="6">Vbusstate</td><td colspan="5">Encoded Vbus voltage state</td></tr>
<tr><td>Value</td><td>Vbus voltage</td><td>SessEnd</td><td>SessValid</td><td>VbusValid</td></tr>
<tr><td>00</td><td>Vbus <Vb_sess_end</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>01</td><td>Vb_sess_end<Vbus< Vsess_vld</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>10</td><td>Vsess_vld<Vbus< Va_vbus_vld</td><td>X</td><td>1</td><td>0</td></tr>
<tr><td>11</td><td>Va_vbus_vld<Vbus</td><td>X</td><td>X</td><td>1</td></tr>
<tr><td rowspan="6">5:4</td><td rowspan="6">RxEvent</td><td colspan="4">Encoded UTMI event signals</td></tr>
<tr><td>Value</td><td>RxActive</td><td>RxError</td><td>HostDisconnect</td></tr>
<tr><td>00</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>01</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>11</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>10</td><td>X</td><td>X</td><td>1</td></tr>
<tr><td>6</td><td>ID</td><td colspan="5">Set to value ID ground</td></tr>
<tr><td>7</td><td>Alt_int</td><td colspan="5">Used for carkit interrupt latch</td></tr>
</table>

CHAPTER 5
TIMING ISSUES

5.1 Introduction

The Universal Serial Bus (USB) Standard has become, in recent years, an important product for use in digital communications as a direct replacement for different communication interfaces previously used with relevant advantages. Moreover, its extended use in very critical and sensitive applications like medical science equipments and ballistic missiles require high level of reliability. Such a high level of reliability requires a very accurate and robust communication platform.

Timing is a most critical task in the designing phase. Detecting timing problems early in the design process not only saves time but also permits much easier implementation of design alternatives. To ensure accurate data transfer for memory interfaces that operate at 200 MHz and beyond, timing analysis needs to play a more prominent role in the identification and resolution of system operation issues. At these frequencies, the margins for setup and hold times are tight, leaving minimal room to secure an accurate data capture and presentation window. Faster edge rates also magnify physical design effects which cause signal integrity issues that require additional settling time, shrinking timing margins. Few steps taken to take care of timing are:

1. Use of FPGA resources like Input output pad, Digital clock manager and Clock buffers efficiently and effectively.
2. LUT are used as a memory for storing small data while large chunk of data should be stored at Block Select RAM (BRAM).
3. Use Flip Flop to store the data temporarily and avoid formation of latches.
4. Use D Flip flop, not JK or T Flip Flop.
5. Avoid formation of combinational loops.
6. Case statement is used to form large input multiplexer rather than if statement.
7. Put flip flops on primary inputs and outputs of a chip to avoid signal integrity problem.
8. In a state machine, illegal and unreachable states are transition to the reset state.
9. If state machine has less than 16 states, use a one-hot encoding and many others.

These major implementation steps are explained in coming section. These are very critical for interfacing with high speed links.

5.2 Bidirectional Port

Any inout (bidirectional) port in the design interface is implemented as shown in figure 5.1. There is tristate buffer for outgoing signal while incoming signal should not be tristated. If incoming signal is tristated then there will be glitches during change of incoming signal from tristate to stable state which may alter the functioning of the chip.

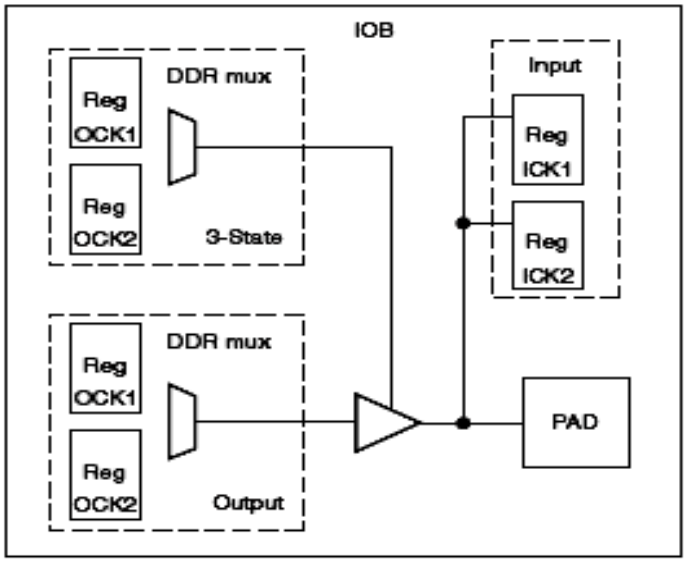

Figure 5.1 Input Output Pad [7]

All input and output signals can be routed from IO Block of FPGA by instantiating its module in the design. Usages of register near the input and output nets helps in resolving interfacing timing problems. On the input and output port, two registers can be used for double data rate interface. Data is transmitted at both positive and negative edge of the clock in double data rate transmission. Double data rate is directly accomplished by the two registers on each path, clocked by the rising edges from two different clock nets. These two clock signals are generated by the DCM and must be 180 degrees out of phase. Clock signals are routed through the clock buffers present in the FPGA to prevent any loading effect. A clock buffer reduces the clock transition time and provides the total clock period for the operation. Digital clock manager helps in providing

the clock to each of the flip flops without any skew. The DCM generates new system clocks which are phase-aligned to the input clock thus eliminating clock distribution delays.

5.3 Look-Up Table

Virtex-II function generators are implemented as 4-input look-up tables (LUTs). Four independent inputs are provided to each of the two function generators in a slice. These function generators are each capable of implementing any arbitrarily defined boolean function of four inputs. The propagation delay is therefore independent of the function implemented. Signals from the function feed the D input of the storage element. Each function generator (LUT) can implement a 16 x 1-bit synchronous RAM resource called a distributed Select RAM element. Distributed Select RAM memory modules are synchronous (write) resources. The combinatorial read access time is extremely fast while the synchronous write simplifies high-speed designs. A synchronous read can be implemented with a storage element in the same slice. The distributed Select RAM memory and the storage element share the same clock input.

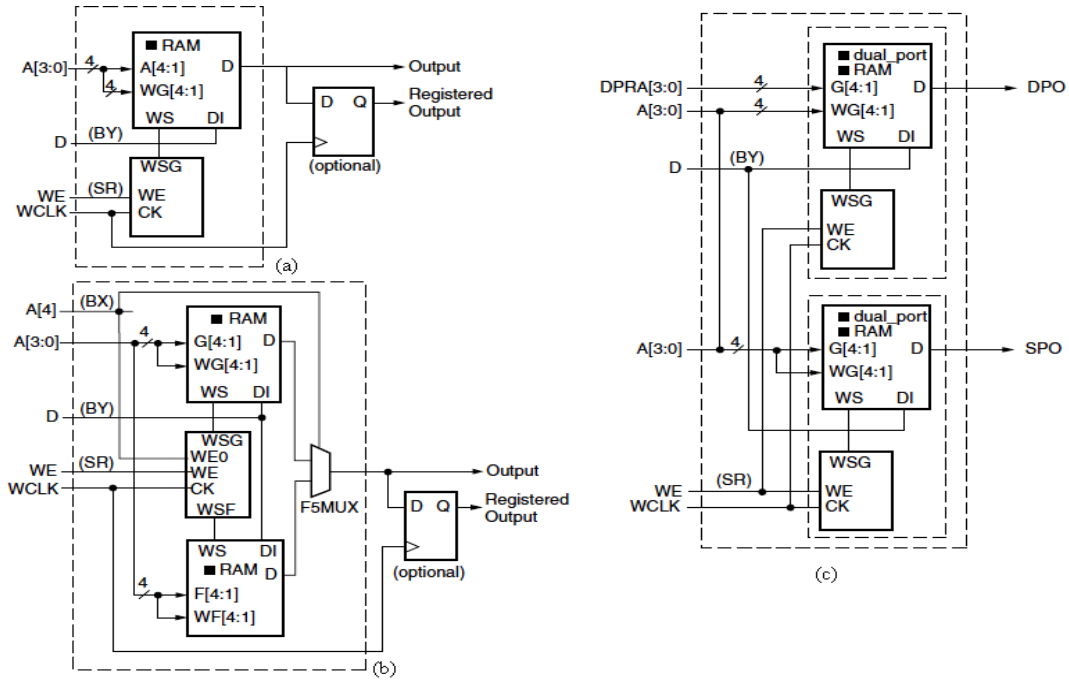

Figure 5.2 (a) Single Port Distributed Select RAM 16 × 1 (b) Single Port Distributed Select RAM 32 × 1 made from two 16 × 1 Distributed Select RAM (c) Dual Port Distributed Select RAM 32 × 1 [7]

For single-port configurations, distributed Select RAM memory has one address port for synchronous writes and asynchronous reads as shown in figure 5.2(a). For dual-port configurations, distributed Select RAM memory has one port for synchronous writes and asynchronous reads and another port for asynchronous reads as shown in figure 5.2(b). In single-port mode, read and write addresses share the same address bus. In dual-port mode, one function generator (R/W port) is connected with shared read and write address. The second function generator has the G inputs (read) connected to the second read only port address and the WG inputs (write) shared with the first read/write port address. These memories are used to store small chunks of data and are much faster than Block Select RAM.

5.4 Block Select RAM

Virtex-II devices incorporate large amounts of 18 Kbit block Select RAM. These complement the distributed Select RAM resources that provide shallow RAM structures implemented in CLBs. Each Virtex-II block Select RAM is an 18 Kbit true dual-port RAM with two independently clocked and independently controlled synchronous ports that access a common storage area. Both ports are functionally identical. Operations are synchronous for both read and write operation. The block Select RAM behaves like a register. Control, address and data inputs must be valid during the set-up time window prior to a rising (or falling, a configuration option) clock edge. Data outputs change as a result of the same clock edge.

5.4.1 Read/Write Operations

The Virtex-II block Select RAM read operation is fully synchronous. An address is presented and the read operation is enabled by control signals in addition to enable signals. Then, depending on clock polarity, a rising or falling clock edge causes the stored data to be loaded into output registers. The write operation is also fully synchronous. Data and address are presented and the write operation is enabled by control signals in addition to enable signals. Then, again depending on the clock input mode, a rising or falling clock edge causes the data to be loaded into the memory cell addressed. A write operation performs a simultaneous read operation. Three different options are available which are selected by configuration. These configurations are:

1. WRITE_FIRST: The “WRITE_FIRST” option is a transparent mode. The same clock edge that writes the data input (DI) into the memory also transfers DI into the output registers DO as shown in figure 5.3.

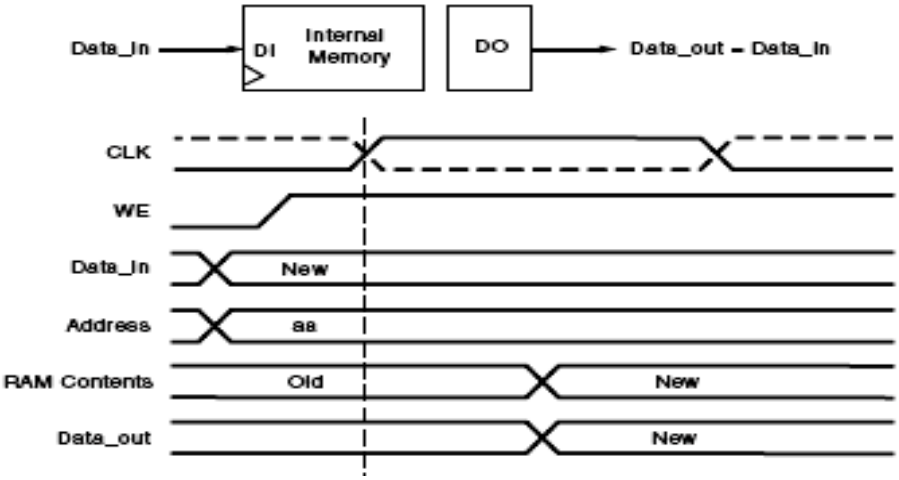

Figure 5.3 Write First Operation [7]

2. READ_FIRST: The “READ_FIRST” option is a read-before-write mode. The same clock edge that writes data input (DI) into the memory also transfers the prior content of the memory cell addressed into the data output registers DO as shown in Figure 5.4.

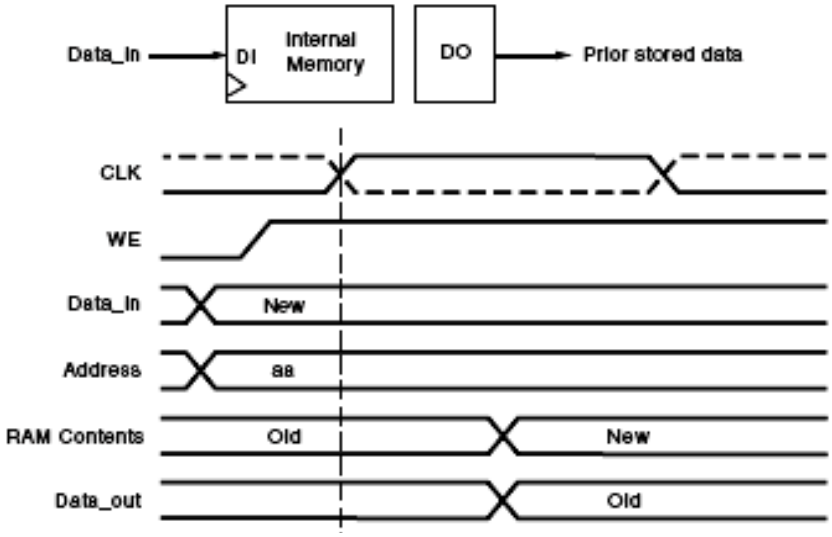

Figure 5.4 Read First Operation [7]

3. NO_CHANGE: The “NO_CHANGE” option maintains the content of the output registers regardless of the write operation. The clock edge during the write mode has no effect on the content of the data output register DO. When the port is configured as “NO_CHANGE”, only a read operation loads a new value in the output register DO as shown in figure 5.5.

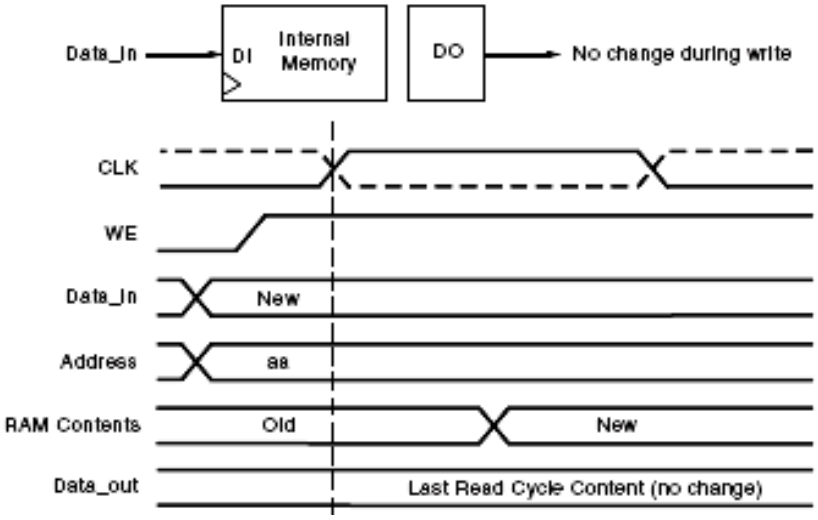

Figure 5.5 No Change Operation [7]

5.4.2 Advantage of Block RAM

A Synchronous Distributed RAM cannot perform read-modify-write operations in a single clock cycle but the Dual port Synchronous Block RAM in all Xilinx FPGAs can pipeline the write operation and achieve a throughput of one read-modify-write operation per clock cycle. To do so, the designer uses Port A as the read port, uses Port B as the write port and uses one common clock for both ports. The read address is routed to Port A. A copy of the read address is delayed by one clock and routed to Port B. The data from Port A is modified and used as the data input to Port B. Virtex-II Select RAM memory blocks are located in either four or six columns. The number of blocks per column depends of the device array size and is equivalent to the number of CLBs in a column divided by four.

5.5 Digital Clock Manager (DCM)

Clock signals are routed through the Digital Clock Manager present in the FPGA to prevent any loading and transition delays. Digital Clock Manager helps in providing the clock to each of the flip flops without any skew. It generates new system clocks which are phase-aligned to the input clock thus eliminating clock distribution delays. Hence, DCM offers a wide range of powerful clock management features like

1. Clock De-skew: The DCM generates new system clocks (either internally or externally to the FPGA) which are phase-aligned to the input clock. Thus, it eliminates clock distribution delays.
2. Frequency Synthesis: The DCM generates a wide range of output clock frequencies, thus, performing very flexible clock multiplication and division.
3. Phase Shifting: The DCM provides both coarse phase shifting and fine-grained phase shifting with dynamic phase shift control.

The DCM utilizes fully digital delay lines allowing robust high precision control of clock phase and frequency. It also utilizes fully digital feedback systems operating dynamically to compensate for temperature and voltage variations during operation. It is highly recommended to route the clock signal through DCM for robust system operation. Virtex-II devices have 16 clock input pins that can also be used as regular user I/Os. Eight clock pads are on the top edge of the device in the middle of the array and eight are on the bottom edge as illustrated in figure 5.6.

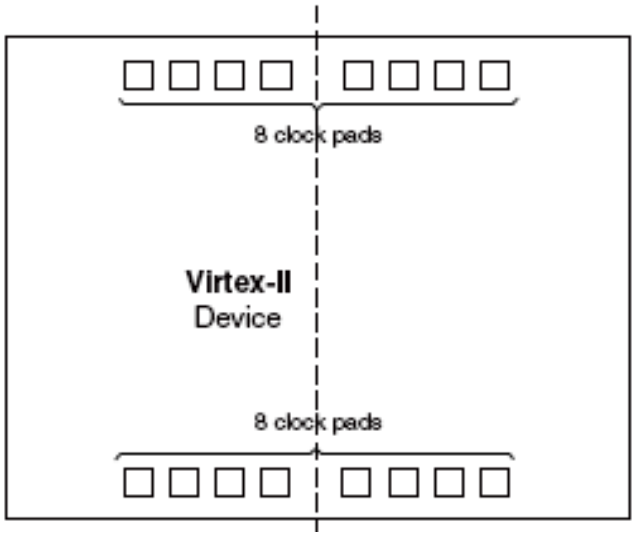

Figure 5.6 Clock Pads [7]

The global clock multiplexer buffer represents the input to dedicated low-skew clock tree distribution in Virtex-II devices. Like the clock pads, eight global clock multiplexer buffers are on the top edge of the device and eight are on the bottom edge. Each global clock buffer can either be driven by the clock pad to distribute a clock directly to the device or driven by the Digital Clock Manager (DCM). The DCM has clock outputs that are connected to global clock buffer inputs. Global clock buffers are used to distribute the clock to some or all synchronous logic elements (such as registers in CLBs, IOBs and Select RAM blocks). Eight global clocks can be used in each quadrant of the Virtex-II device. Designers should consider the clock distribution detail of the device prior to pin locking and floorplanning for more robust design.

5.6 Xilinx ChipScope

The Xilinx ChipScope tools package has several modules that designer can add to the design to capture input and output directly from the FPGA hardware. These are:

1. ICON (Integrated Controller): A controller module that provides communication between the ChipScope host PC and ChipScope modules in the design (such as VIO and ILA).
2. VIO (Virtual Input/Output): A module that can monitor and drive signals in the design in real-time. It is function as a virtual push-buttons (for input) and LEDs (for output). These are used for debugging purposes or can incorporate into the design as a permanent I/O interface.
3. ILA (Integrated Logic Analyzer): A module that lets designer view and trigger on signals in the hardware design. It works as a digital oscilloscope (like ModelSim's waveform viewer) that can be placed in the design to aid in debugging.

These ChipScope modules are extremely useful because they allow designer to view and manipulate signals directly from hardware during run-time. Since they are real verilog modules and netlists, they get incorporated, synthesized and implemented into the design just like any other Verilog code is incorporated. Using ChipScope, designer can capture almost any signal in the system including top-level signals. If the problem with the design lies at the top-level or is fundamentally hardware related, ChipScope modules are probably the best way to debug them. Checking the design in simulation cannot catch errors that are made in the synthesizable but not simulatable parts of the design. The hardware related timing bugs are easily captured by the Chipscope module.

5.7 CODING STYLE

This section gives guidelines for building robust, portable and synthesizable HDL code. Portability is both for different simulation, synthesis tools and for different implementation technologies. There is a world of difference between getting a design to work in simulation and getting it to work on a real FPGA.

5.7.1 Unsynthesizable Code

Synthesis is done by matching HDL code against templates or patterns available in the FPGA. It's important to use coding style that a synthesis tool recognizes. If designer is not careful then he could write the code that result in inefficient or incorrect hardware.

Bad code for FPGAs produce circuits with the following features:

1. Latches
2. Combinational loops
3. Multiple drivers for a signal
4. Tri-state buffers

Bad practice to code means produces undesirable hardware. Coding styles that lead to inefficient hardware might be useful in the early stages of the design process when the focus is on functionality and not optimality. The Unsynthesizable code contains:

1. Assign initial values on signals: In most implementation technologies, when a circuit powers up, the values on signals are completely random. Some FPGAs are an exception to this. For some FPGAs, when a chip is powered up, all flip flops will be '0'.
2. Wait for length of time: Delays through circuits are dependent upon both the circuit and their operating environments particularly supply voltage and temperature.
3. Multiple if rising edge statements in a process: The synthesis tools generally expect just a single if rising edge statement in each process. The simpler the VHDL code is, the easier it is to synthesize hardware.
4. The if-statement has a rising edge condition and an else clause: Generally, an if-then-else statement synthesizes to a multiplexer. The condition that is tested in the if-then-else becomes the select signal for the multiplexer. In the if-statement has a

rising edge with else, the select signal would need to detect a rising edge on clk which is not feasible to synthesize.

5.7.2 Synthesizable but Bad Coding Practice

There are many HDL codes which are synthesizable but results in different hardware for different FPGA Boards. It mainly generates latches and produce undesirable results. These codes contain:

1. Combinational “if-then” Without “else”: This code synthesizes to be a latch and latches are undesirable.
2 Bad Form of Nested If’s: This design results in a level sensitive latch whose input is a flip flop. It results in unpredictable output.
3. Missing signals in sensitivity list: It produces undesirable results.

5.7.3 Synthesizable VHDL Coding Guidelines

It is a better practice to use synthesizable codes to impact on the FPGA. It produces expected hardware on the FPGA. These are:

1. Use signals, do not use variables: The intention of the creators of HDL was for signals to be wires and variables to be just for simulation.
2. Use std_logic signals, do not use bit or Boolean: std_logic is the most commonly used signal type across synthesis tools, simulation tools and cell libraries.
3. Use in or out, do not use inout inside the block: inout signals are tri-state.
4. Use flops, not latches.
5. Use D-flops, not T, JK etc.
6. Do not assign a signal to itself: If the signal is a flop, use a chip enable to cause the signal to hold its value. If the signal is combinational then assigning a signal to itself will cause combinational loop which is a bad design.
7. Put flip flops on primary inputs and outputs of a chip: It creates more robust implementations. Signal delays between chips are unpredictable. Signal integrity can be a problem. Putting flip flops on inputs and outputs of chip provides clean boundaries between circuits. This only applies to primary inputs and outputs of a chip (the signals in the top-level entity). Within a chip, designer should adopt a

standard of putting flip-flops on either inputs or outputs of modules according to the design requirement.

8. Use multiplexers, not tri-state buffers
9. For a combinational process, the sensitivity list should contain all of the signals that are read in the process.
10. For a combinational process, every signal must be assigned to another in every branch of if-then and case statements: If a signal is not assigned a value in a path through a combinational process then that signal will be a latch.
11. Each signal should be assigned to in only one process: Multiple processes driving the same signal is the same as having multiple gates driving the same wire. This can cause contention, short circuits and other bad things.
12. Separate unrelated signals into different processes: Grouping assignments to unrelated signals into a single process can complicate the control circuitry for that process. Each branch in a case statement or if-then-else adds a multiplexer or chip enable circuitry. Synthesis tools generally optimize each process individually. The larger a process is, the longer it will take the synthesis program to optimize the process.
13. Illegal and unreachable states should transition to the reset state: In a state machine, Reset state creates more robust implementations. When design interacts with external environment then the design will be subjected to illegal inputs, voltage spikes, temperature fluctuations, clock speed variations etc. At any time, something weird will happen that will cause it to jump into an illegal state. Reset state prevents undesirable results.
14. If the state machine has less than 16 states, use a one-hot encoding: For n states, a one-hot encoding uses n flip-flops while a binary encoding uses $\log_2 n$ flip-flops. One-hot signals are simpler to decode because only one bit must be checked to determine if the circuit is in a particular state. For small values of n, a one-hot signal results in a smaller and faster circuit.
15. Include a reset signal in all clocked circuits: For most implementation technologies, when circuit is power-up, state is not confirmed in which it will start. It needs a reset signal to get the circuit into a known state. If something goes wrong while the circuit is running, it needs a way to get it into a known state.

5.8 TIMING

Take an example of design made up of 4 combinational blocks with definite block delay shown in figure 5.7. It is interacting with another chip which is working at 10MHz clock. Let assume that there is no delay between chip to chip. If there is no flip flop in the design then it will show timing violation as maximum register to register delay is 120ns (exceeds the clock time period).

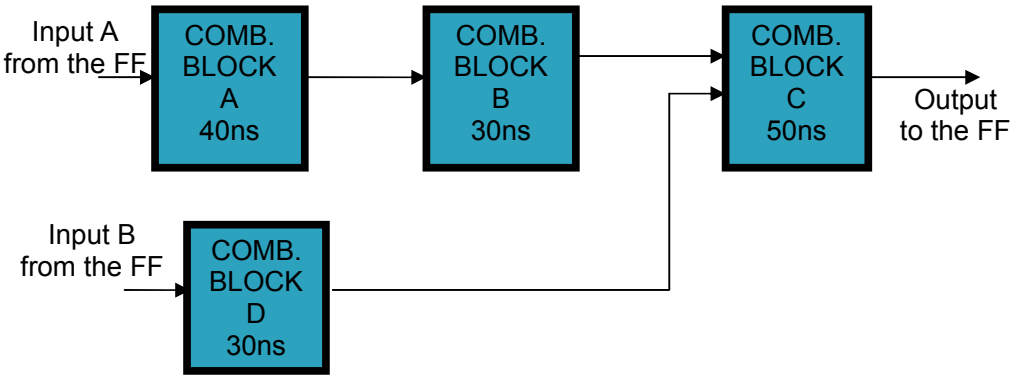

Figure 5.7 Design without internal flip flops

If flip flops are inserted between each block as shown in figure 5.8, then maximum register to register delay incur is 50ns. There is no timing violation and design will work on 10MHz clock. It is a bad design as Hardware latency is increased without any need and circuit is idle for half a clock period. It is necessary to reimplement the design.

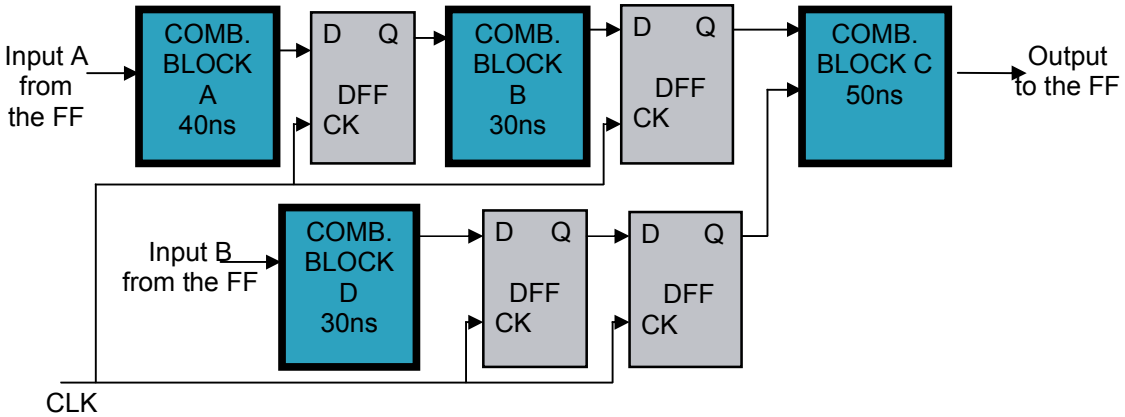

Figure 5.8 Design with large number of internal flip flops

If flip flops are inserted between each block as shown in figure 5.9, then maximum register to register delay incur is 80ns. There is no timing violation and design will work on 10MHz clock. Here circuit performance is increased but it is a bad design as inputs to the Block C are coming at different time interval. One input is coming at the clock edge while second input is coming 30ns after the clock edge. It results in formation of glitches at the output.

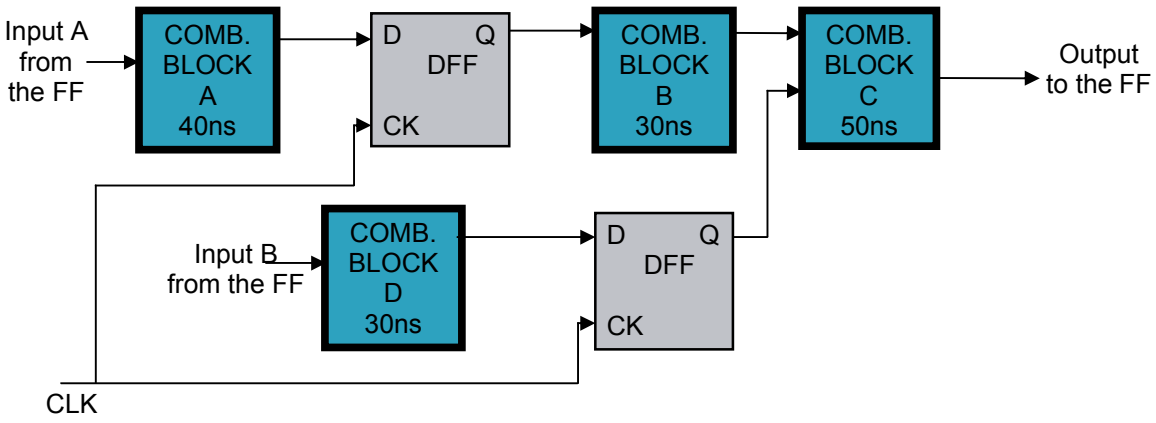

Figure 5.9 Design with improperly placed internal flip flops

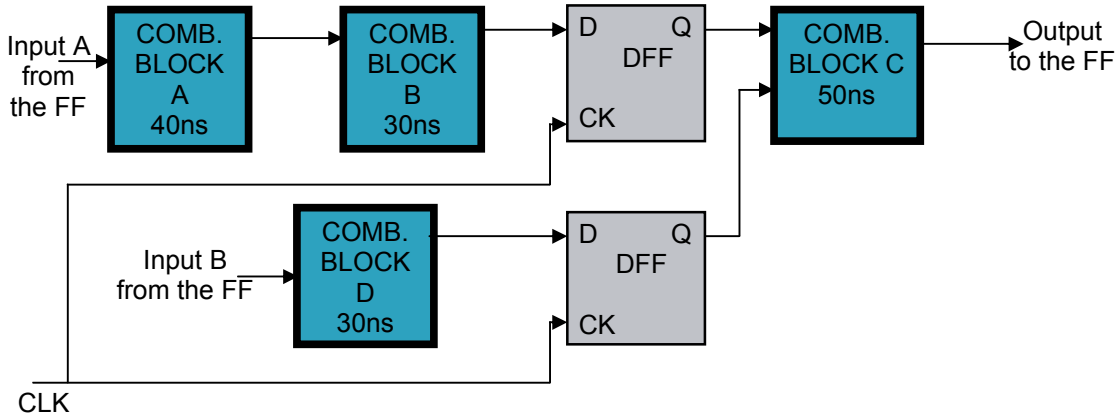

Figure 5.10 Design with properly placed internal flip flops

If flip flops are inserted between each block as shown in figure 5.10 then maximum register to register delay incur is 70ns. There is no timing violation and design will work on 10MHz clock. Here circuit performance is increased. Timing issues of the block design can be easily achieved by using flip flop but interface signal timings are more critical. Within the block, one operation is performed in one full clock period but at interfacing, operation is performed in part of the clock period as signal take time for

propagation from output port of a chip to input port of another chip. Below table differentiate between slack coming in design without modification and with modification at the interfacing section.

DESIGN WITHOUT MODIFICATION REPORT:

Constraint	Check	WorstCase Slack	Best Case Achievable	Timing Errors	Timing Score
ULPIDIR OFFSET = IN 4 ns AFTER ULPICLK HIGH	SETUP	-2.642ns	6.642ns	15	22702
WIRE_ULPINXT OFFSET = IN 4 ns AFTER ULPICLK HIGH	SETUP	-2.460ns	6.460ns	15	19972
ULPI_DATA OFFSET = IN 4 ns AFTER ULPICLK HIGH	SETUP	-2.280ns	6.280ns	14	17076
ULPI_DATA OFFSET = OUT 8 ns BEFORE ULPICLK HIGH	MAXDELAY	-0.157ns	8.157ns	2	161
ULPISTP OFFSET = OUT 8 ns BEFORE ULPICLK HIGH	MAXDELAY	-0.133ns	8.133ns	1	133

Constraints not met.

DESIGN WITH MODIFICATION REPORT:

Constraint	Check	WorstCase Slack	Best Case Achievable	Timing Errors	Timing Score
ULPIDIR OFFSET = IN 4 ns AFTER ULPICLK HIGH	SETUP	3.274ns	0.726ns	0	0
WIRE_ULPINXT OFFSET = IN 4 ns AFTER ULPICLK HIGH	SETUP	3.316ns	0.684ns	0	0
ULPI_DATA OFFSET = IN 4 ns AFTER ULPICLK HIGH	SETUP	3.085ns	0.915ns	0	0
ULPI_DATA OFFSET = OUT 8 ns BEFORE ULPICLK HIGH	MAXDELAY	0.065ns	7.935ns	0	0
ULPISTP OFFSET = OUT 8 ns BEFORE ULPICLK HIGH	MAXDELAY	0.053ns	7.947ns	0	0

All constraints were met.

CHAPTER 6
UTMI TO ULPI INTERFACE CONVERTER

In the USB Communication, there are two types of standard interfaces which are:

1. UTMI (USB2.0 Transceiver Macrocell Interface)
2. ULPI (UTMI+ Low Pin Interface)

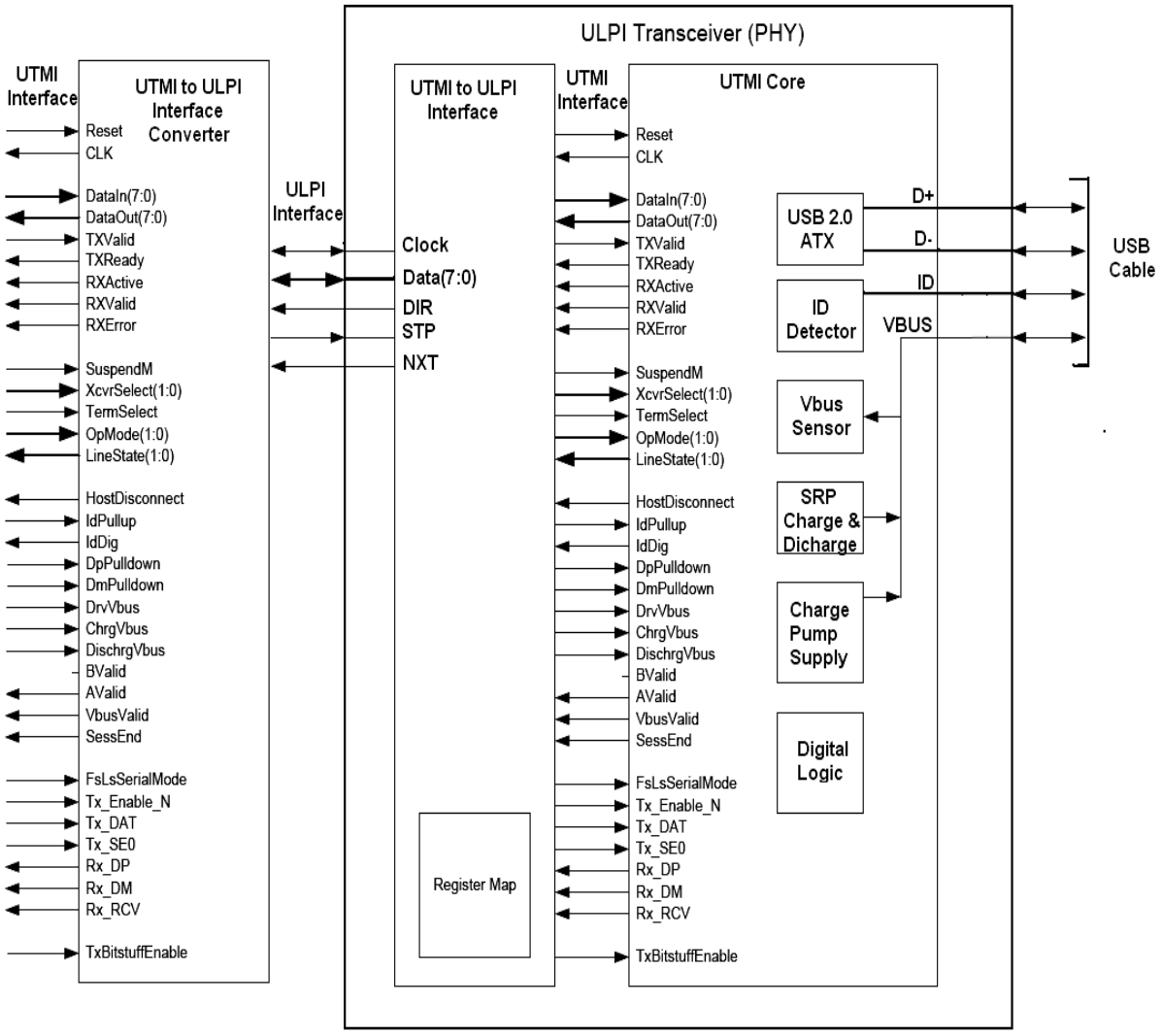

Figure 6.1 UTMI and ULPI Interface in ULPI Transceiver

6.1 Hardware Setup

UTMI is used to handle the USB signaling protocol to communicate between link and device. This includes features like data serialization, de-serialization, bit stuffing and clock synchronization. UTMI has 100 pins and contain many static pins. ULPI interface reduces the pin count for discrete USB transceiver implementations supporting On The Go implementation. ULPI use the register set to define the static signals of the UTMI interface. Depending upon the application, data lines are used to transmit or receive the packet and activate the static registers through the register write operation. The hardware set up for the UTMI to ULPI converter is shown below in figure 6.2.

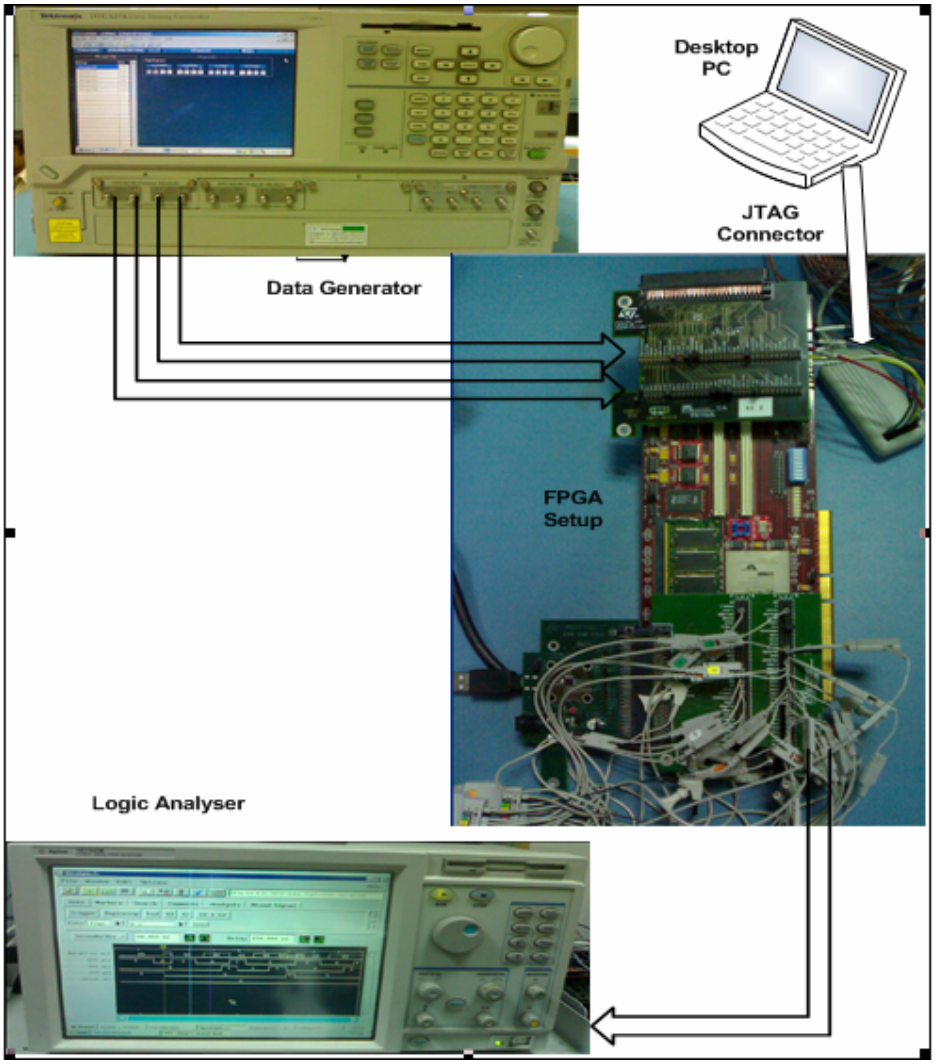

Figure 6.2 Setup for the UTMI to ULPI converter

This converter is used to validate the functionality of the PHY Board by setting the UTMI static pins manually. These static pins can be controlled by using data generator or by using jumpers on the adaptor board. This UTMI to ULPI converter is validated in Full speed and high speed mode. This code transmits the Txcmd for the register write operation whenever it detects the change in the static signals on the adapter board. It has benefit of reducing the burning of the FPGA again and again for different mode of operation. Static signals, used to set operation mode, are taken out on the pins and can be altered manually without changing the code. A low cost FPGA based device core had been developed [8]. It can enhance this converter by embedding both device core and converter on same FPGA and connecting them in top module. It eliminates the need of extra circuitry like PHY board and adaptor card needed for handshaking.

6.2 Register Write

Register write operation is done before transmission of the data which is shown in figure 6.3. Txcmd is sent to write the register. For different register read write, different Txcmd is to be send as explained in previous chapter. Txcmd explains which operation is to be performed (register read/write or data transmission with NOPID/PID) and on which register (Function control, Interface control and OTG control register).

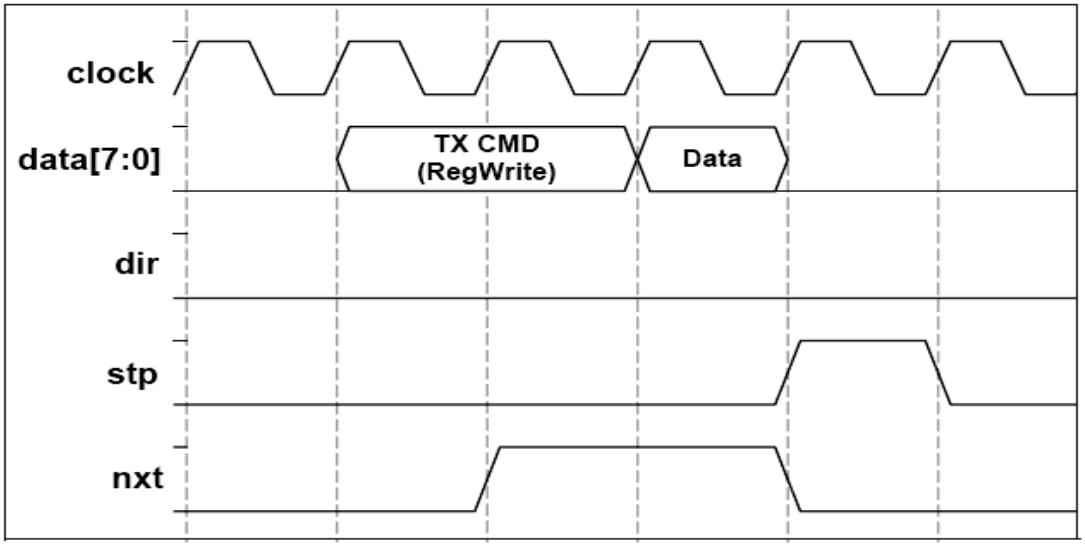

Figure 6.3 Register Write Operation [6]

After the configuration of the PHY, data is transmitted from the UTMI interface toward the ULPI interface after asserting the Txvalid as shown in figure 6.4. When Txvalid is asserted then PHY will assert the Txready to signal the link that it is ready to accept the data.

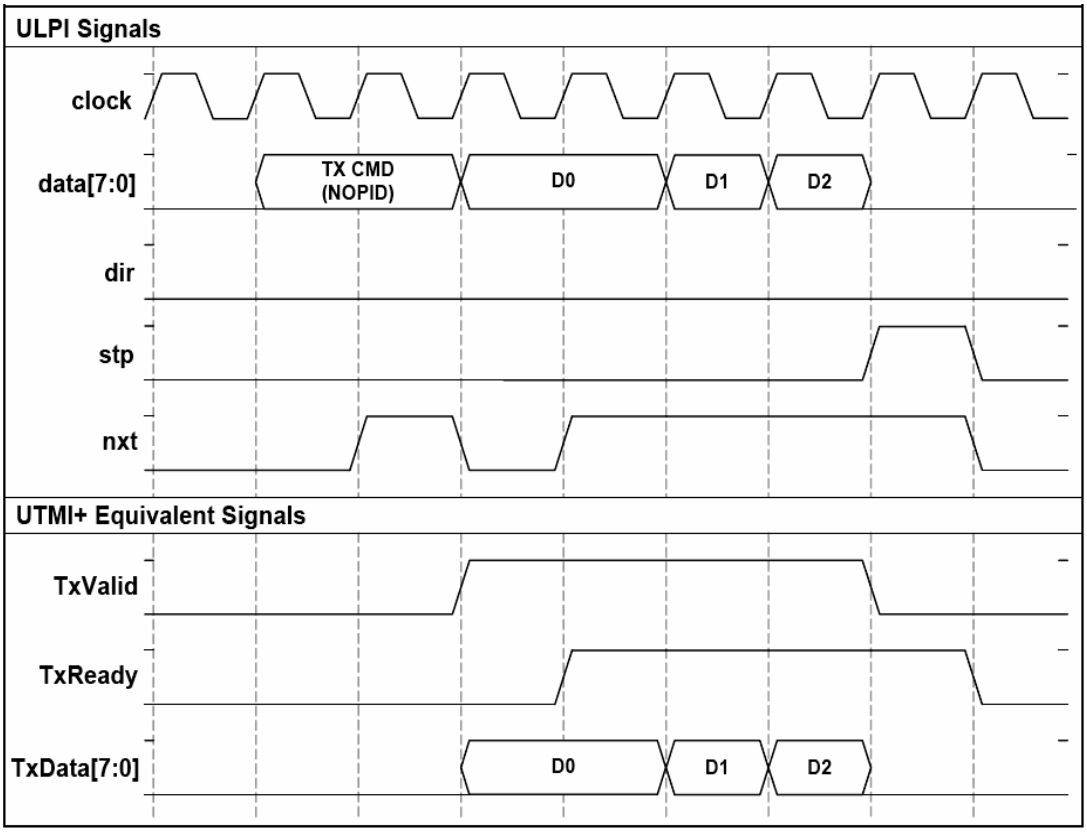

Figure 6.4 USB Data Transmit [6]

6.3 Full Speed Operation

Full speed mode operation is shown in figure 6.5. Steps to configure the device in Full speed mode are:

1. Full speed mode is activated by asserting terminal select signal. Transceiver select signal is assigned the value 01H.
2. Link will send Txcmd (84H) whose lower 6 bits contain the address of the Function control register.
3. When NXT get asserted from the PHY side, Link will transmit the data (65H) to be written in the function register. This data is due to the asserted SuspendM bit in

the function register for powered mode. Opmode is set at 00H for the normal mode while terminal select is asserted for selecting the full speed mode.

4. Link will assert the STP signal for one cycle after receiving the asserted NXT signal from the PHY side. Configuration of mode of operation of PHY board is done by register writing. Below is the figure 6.5 replicate configuration of the PHY through register write operation and then transmission of the data from UTMI data signal to the ULPI data signal.

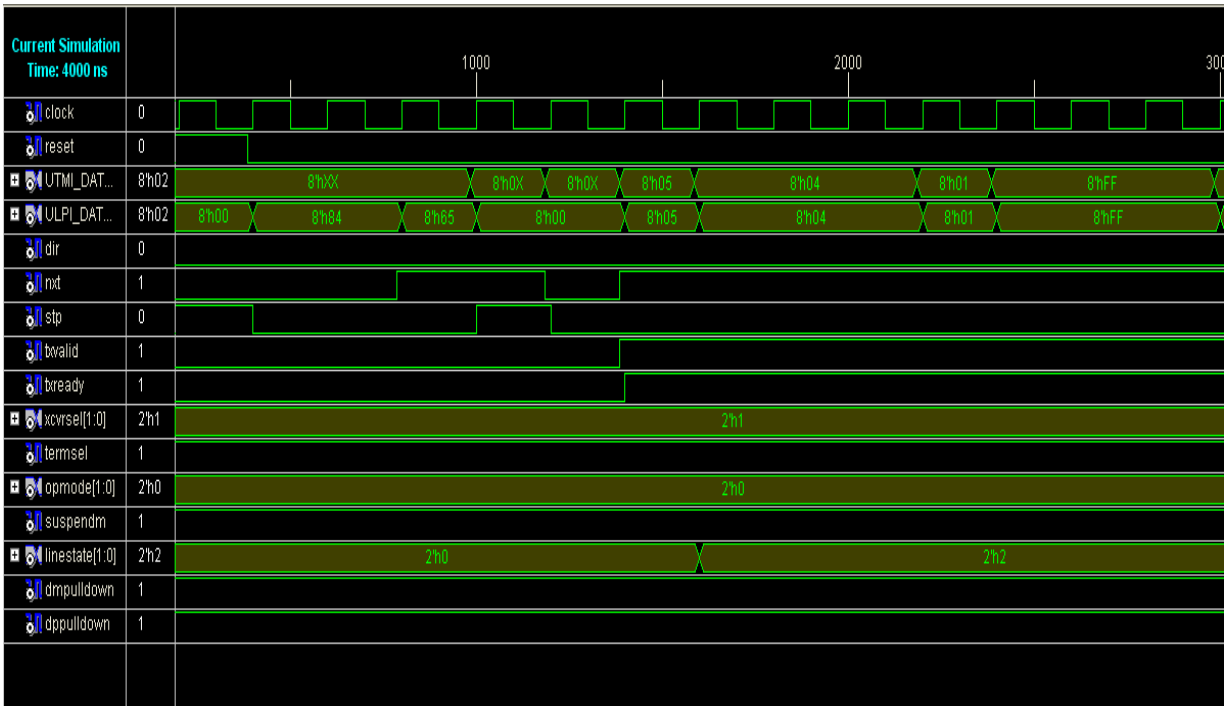

Figure 6.5 Result of USB Data Transmit on Xilinx ISE

5. After the configuration of the PHY in the full speed mode, Txvalid is asserted by the link. Txvalid is used to signal the PHY that link is ready to send the data.
6. If PHY is ready to receive the data, it will assert the Txready signal. If PHY is busy in doing another work, it will not assert the Txready signal.
7. Data from the UTMI data signal will signal at the ULPI data signal on assertion of Txready signal. When data is received by the PHY, DIR should be low. On transmission of the data, Linestate reflects the status of the D+ and D- lines. As seen in figure 6.5, Linestate signal value is 02H i.e. 10b. It means that current state of the receiver is ‘K’ state that shows the start of the data transmission.

Some set of figures captured on the Logic analyzer for the full speed configuration and transmission of the data afterwards.

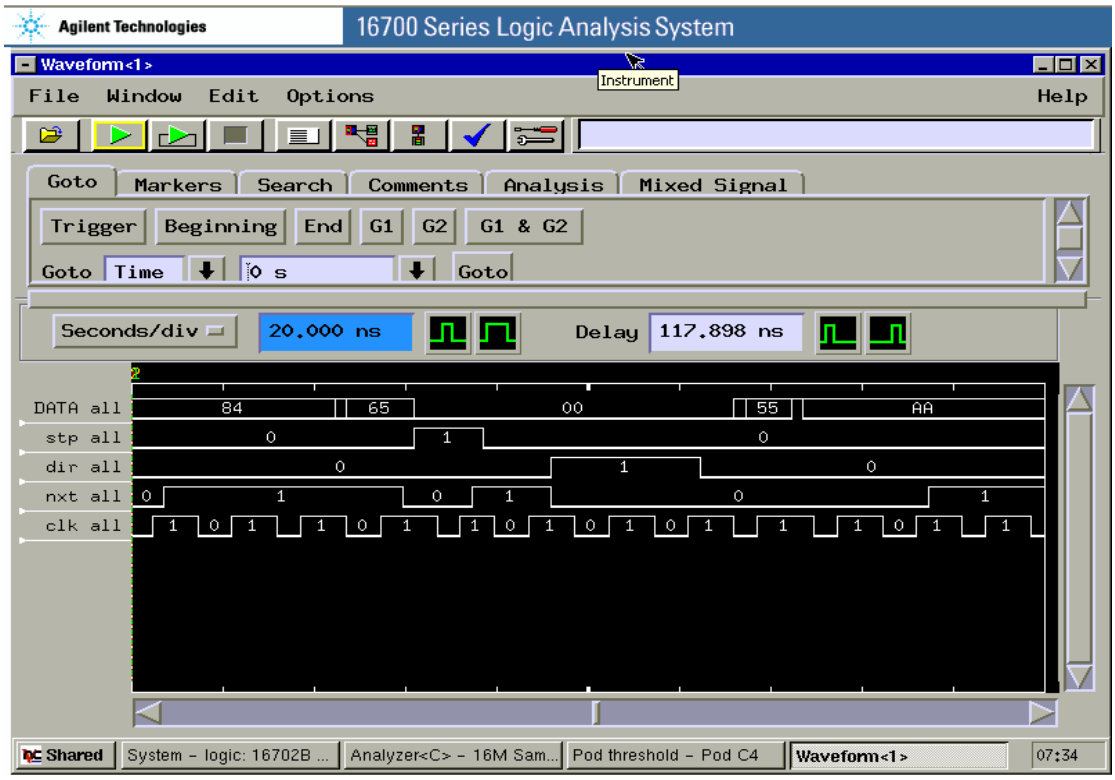

Figure 6.6 Data Transmission in Full speed mode on Logic Analyzer

6.4 High Speed Operation

It can also work in High speed mode. Steps to configure the device in High speed mode are:

1. High speed mode is activated by de-asserting terminal select signal. Transceiver select signal is assigned the value 00H.
2. Link will send Txcmd (84H) whose lower 6 bits contain the address of the Function control register.
3. When NXT get asserted from the PHY side, Link will transmit the data (60H) to be written in the function register. This data is due to the asserted SuspendM bit in the function register for powered mode. Opmode is set at 00H for the normal mode while terminal select is not asserted for selecting the full speed mode.

4. Link will assert the STP signal for one cycle after receiving the asserted NXT signal from the PHY side. Configuration of mode of operation of PHY board is done by register writing. Below is the figure 6.7 replicate configuration of the PHY through register write operation. Transmission of the data from UTMI data signal to the ULPI data signal begins upon assertion of the Txready signal by the PHY.

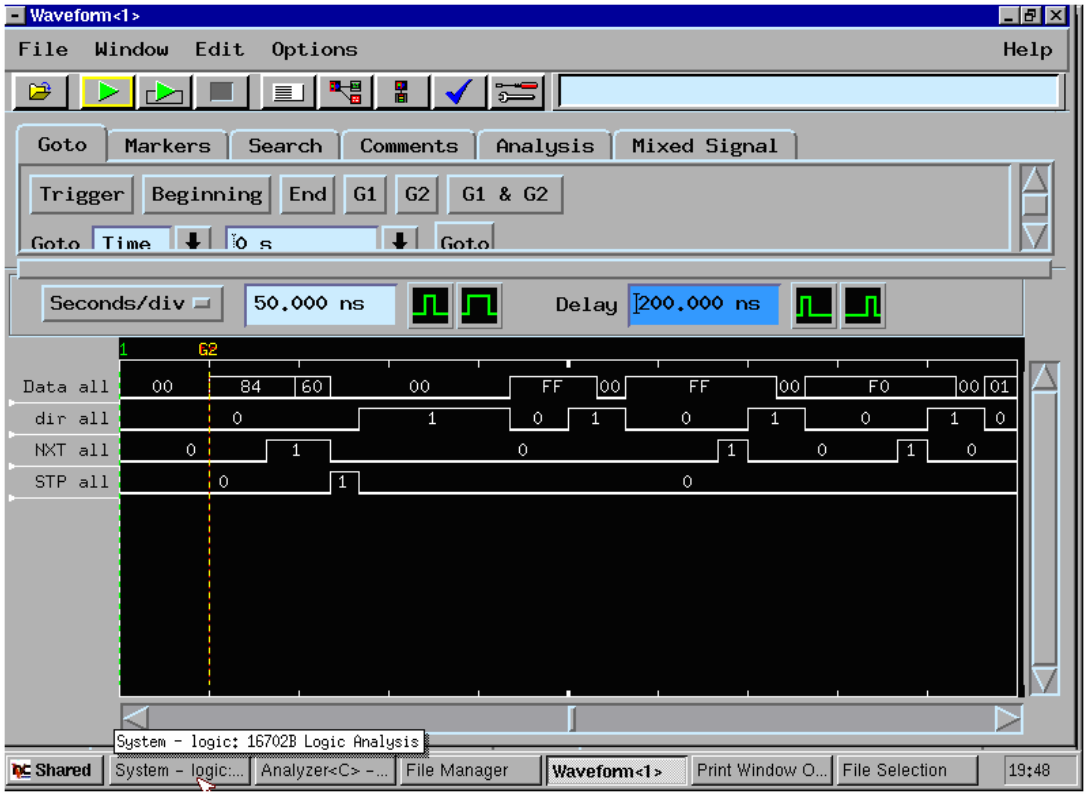

Figure 6.7 Data Transmission in High Speed Mode on Logic Analyzer

6.5 OTG Register Write

This converter can write both Function and OTG register. Procedure to write both OTG and Function register is:

1. Apply high or low to the static signals. For full speed device, Opmode will be 10b, Termselect is asserted, Transceiver select is 01b and Dppulldown is negated while other signals are set to their default values.
2. Link will send the Txcmd (84H) for the function register write.

3. After NXT is asserted from the PHY, Link will transmit data to be written in the function register.
4. After NXT is asserted from the PHY, Link will assert STP for one clock cycle.
5. Link will send the Txcmd (8AH) for the OTG register write.
6. After NXT is asserted from the PHY, Link will transmit data to be written in the OTG register.
7. After NXT is asserted from the PHY, Link will assert STP for one clock cycle.

After completion of function register write, data packets are transmitted on assertion of Txready. Here as shown in the figure 6.8.

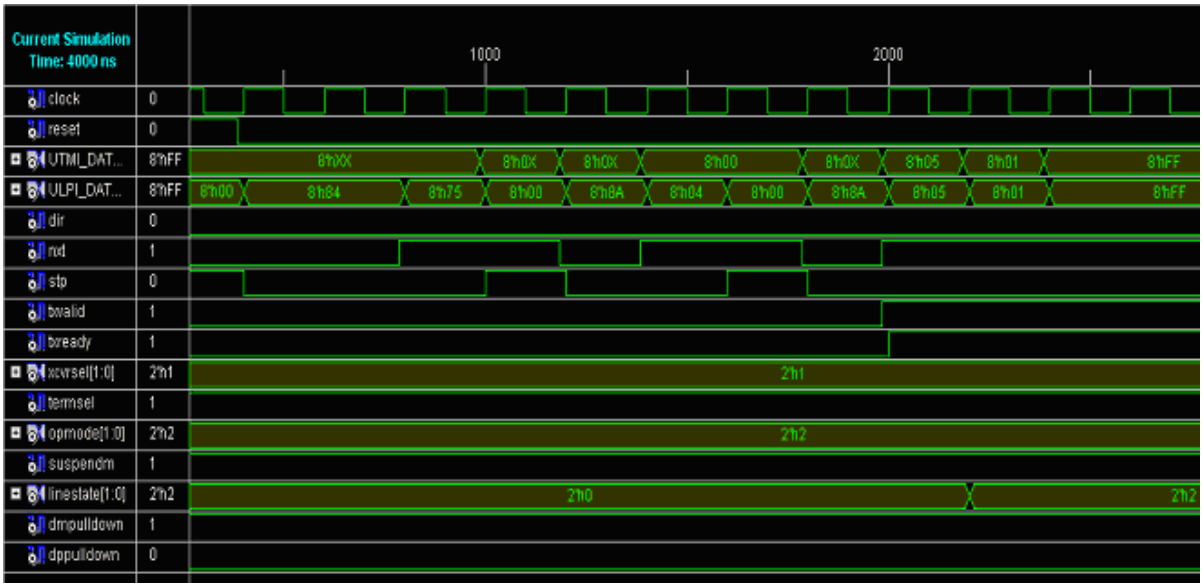

Figure 6.8 Function and OTG register write

CHAPTER 7
BIT ERROR RATE

Bit error rate (BER) is a key parameter that is used to measure effect of noise in data transmission from device to host. When data is transmitted over a data link, there is possibility of error introduction in the channel. If errors are introduced into the data then performance of the system degrades. So, it is necessary to measure the performance of the system and BER provide the ideal way to measure the performance of the system.

Unlike many others measurement methods, BER measure the full end to end performance of the system including transmitter, receiver and the communication channel between them. In this way, BER enables the actual performance of a system in the operation to be tested rather than other testing methods where each component parts are tested and hoping that they will operate satisfactory when placed in the system.

7.1 Bit Error Rate Basics

As the name applies, Bit error rate (BER) is defined as the rate at which error occurs in a transmission system. This is directly measured from number of errors occurred when a string of known data bits are transmitted. The bit error rate can be measured by simple formula:

$$\text{BER} = \frac{\text{Number of errors counted}}{\text{Total numbers of bits send}}$$

If the medium between the transmitter and receiver is good and signal to noise ratio is high then the bit error rate will be very small and is not noticeable effect on the system performance. However, if environment is noisy then there is great importance of measurement of the bit error rate. It measures the performance of the system in the noisy environment.

For the USB data transmission, Bit error rate mainly results from imperfection in the component used and mismatch in impedance matching of transmitter with the channel or receiver with the channel. Another main factor responsible for the bit error in the USB

communication is jitter in the clock that can alter the sampling of the data and cause timing mismatch. This results in the increase in the error counter.

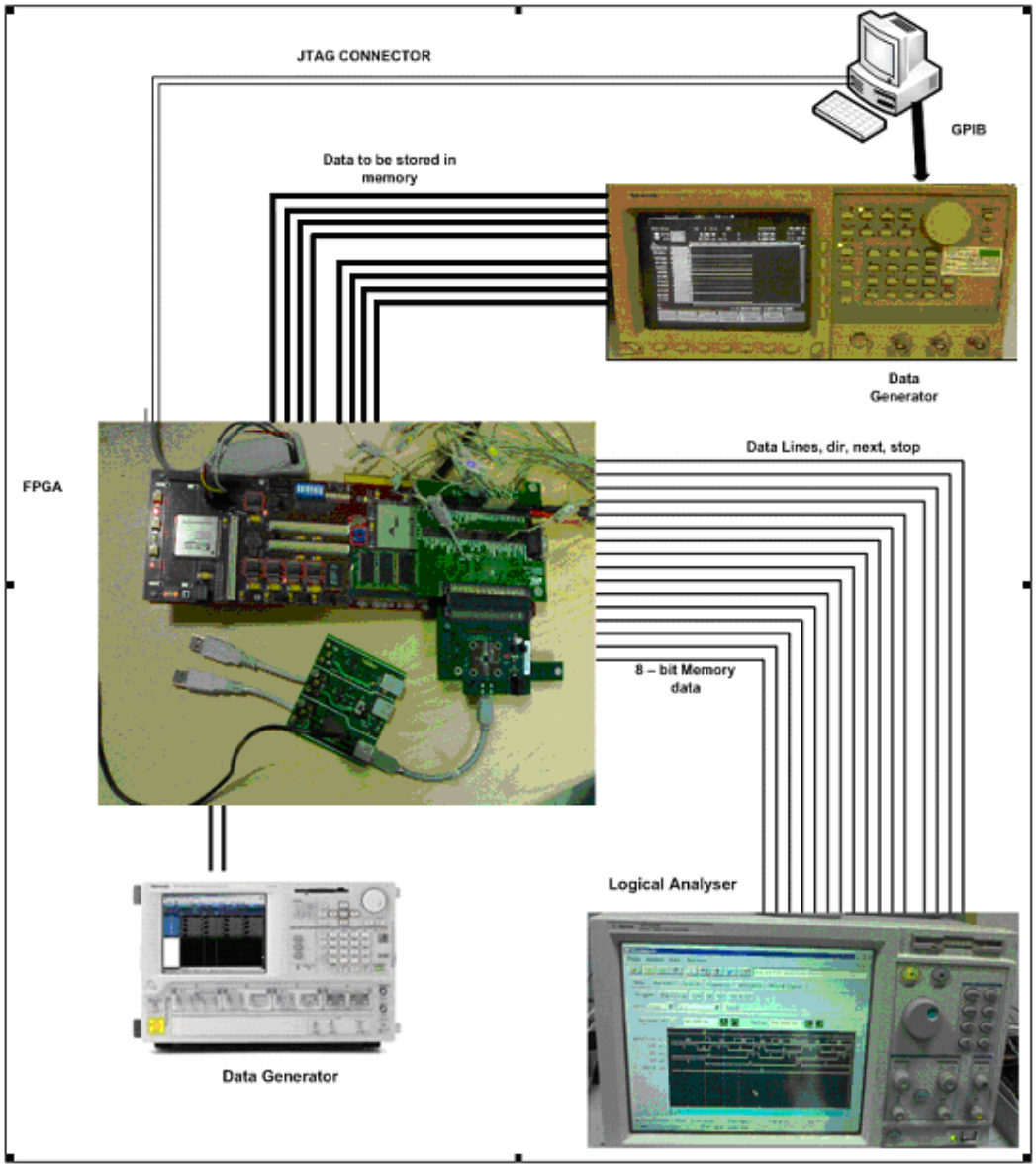

Figure 7.1 Hardware set up for BER

7.2 Measurement of BER through BER setup

According to the figure 7.1, Verilog code is burn in the FPGA through Xilinx ISE through JTAG connection. First step of the code burnt in FPGA is to configure the PHY board through the ULPI interface. Configuration of the PHY board is done through its register writing. Lab view is used to configure the data generator through the personal

computer. This software is used to generate NRZI encoded data required to send from the data generator to D+ and D- lines of the USB port as well as data with clock, read and write enable signal required to send from the data generator to store data in the FPGA BRAM. Script is written in lab view to automatically generate these sequences. Counter is developed in the FPGA which firstly compare the Data from the USB device with the data stored in the BRAM and increment the counter if both are unequal.

A remote diagnostic tool had been developed for USB device kernel resources [9]. This tool provides a method that device driver developer accesses the devices and manipulating their kernel resources directly without writing any code for kernel. This tool reduces the burden in developing USB device drivers. It provides the method that device driver developer accesses the devices and examines the hardware operations without any kernel programming efforts. Boundary-scan technology allows the debugger to talk via a JTAG port directly to the IC's core [10]. This will help in checking the interfacing input output pads quality used to develop this setup. The boundary scan test architecture provides a means to test interconnects between integrated circuits on a board without using physical test probes. It is portable and inexpensive.

7.3 ULPI Register

According to the ULPI interface, Function register is written such that High speed mode is selected with the High speed termination. High speed mode work at 480 Mbps. Opmode used for this setup is mainly Normal mode where NRZI encoder is enabled and Bit stuffing is enabled. This means that data coming on the D+ and D- line will be NRZI encoded with bit stuffed. Before sending the data, there are three registers which is written first to configure the PHY board. These registers are:

1. Function Register: It is used to control the UTMI function setting of PHY like select the bit encoding style, select the transceiver and its speed, transceiver reset and control the suspend mode.
2. Interface Register: It is used to change the interface of the PHY board from ULPI interface to 6 bit or 3 bit Full speed/Low speed Serial mode, switch to Car kit interface, Clock suspend mode and enable PHY to transmit the Auto resume signaling.

3. OTG Register: It is used to control the UTMI+ OTG function of PHY. It can enable sampling of ID line, enable 15Kohm pull down on D+ or D- line, charge or discharge or drive Vbus.

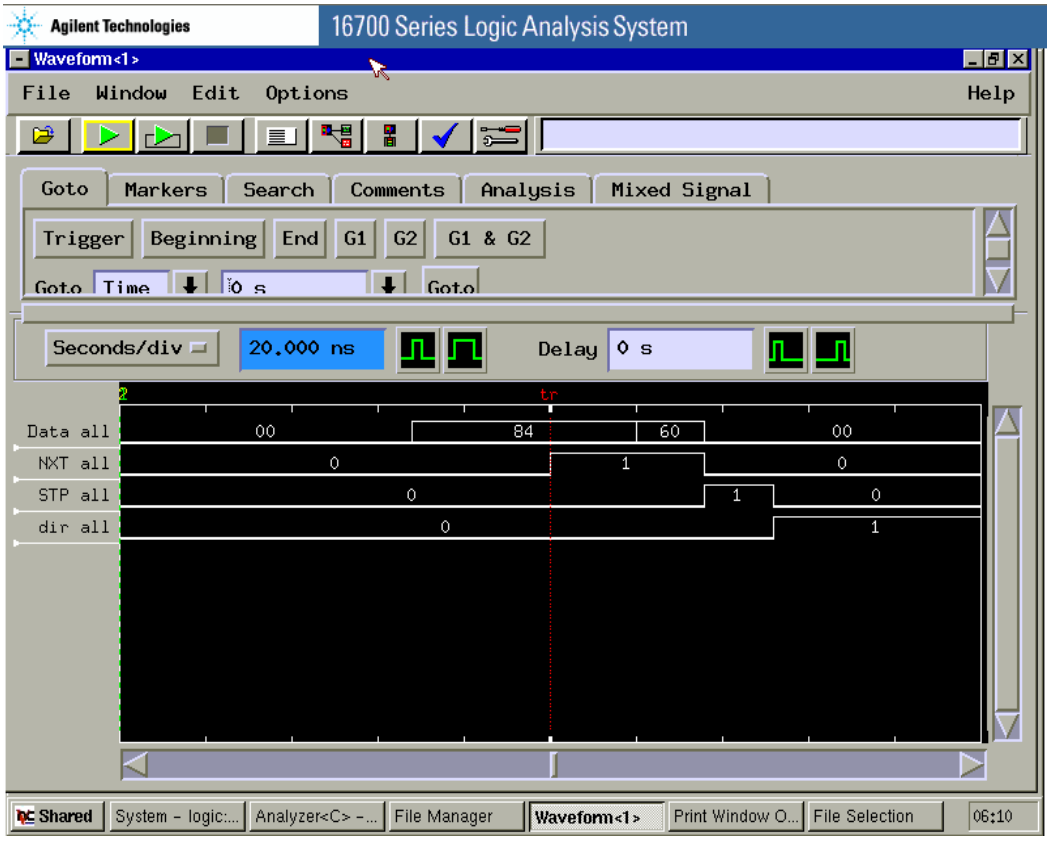

Figure 7.2 Function Register Write

To write into the register, Transmit command (Txcmd) is send. Txcmd upper two msb bits are command code which tells what operation is to be executed while rest bits are data payload. Register write operation is executed by sending command code 10 while data payload reflects the 6 bit immediate register address. Data transmitted after Txcmd is written in the register.

According to the figure 7.2, Txcmd is eighty four. Here command code is 01 which means register write operation is performed. Rest bits refer to the 6 bit register immediate address. Function register 6 bit immediate address is 04H. When link send the Txcmd, it will wait in this state till it get handshake from the PHY. PHY acknowledge the handshake by asserting the NXT high. After getting the NXT high, Link will send the data to be written in the function register. After getting the NXT high again at next clock

edge, Link will send the STP high for one clock cycle. The Link asserts the STP high to signal the end of register write operation. This is the procedure to update the different ULPI registers.

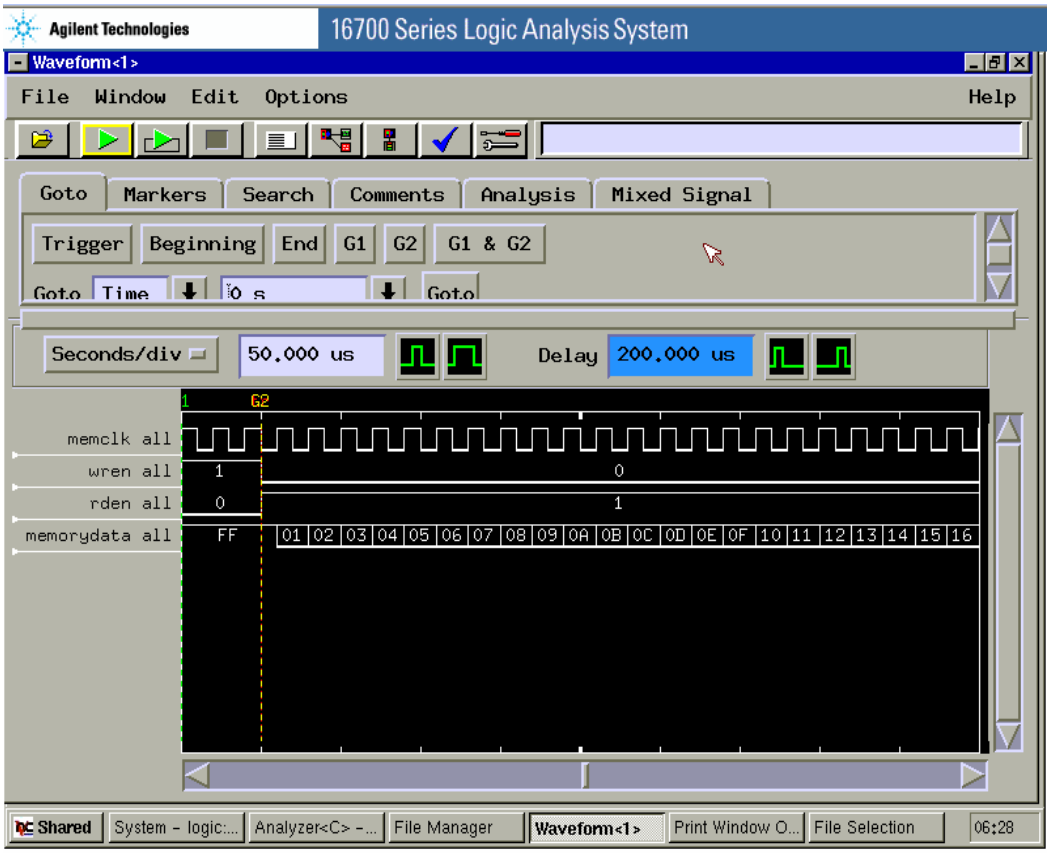

Figure 7.3 BRAM Data Reading

7.4 BRAM Data Storage

BRAM in FPGA is used to store the data which is to be send by the USB device from the USB port. These data is stored in FPGA BRAM through the data generator. Lab view software is used to generate the data to be stored in the BRAM. Data is stored at the clock edge stimulated by the data generator. To store the correct data, data value should be constant at the positive edge of the clock i.e. data at clock equal to zero and data when clock changes to one should be constant. Two pins are used to write and read from the BRAM generated by the data generator. These pins are:

1. Write enable
2. Read enable

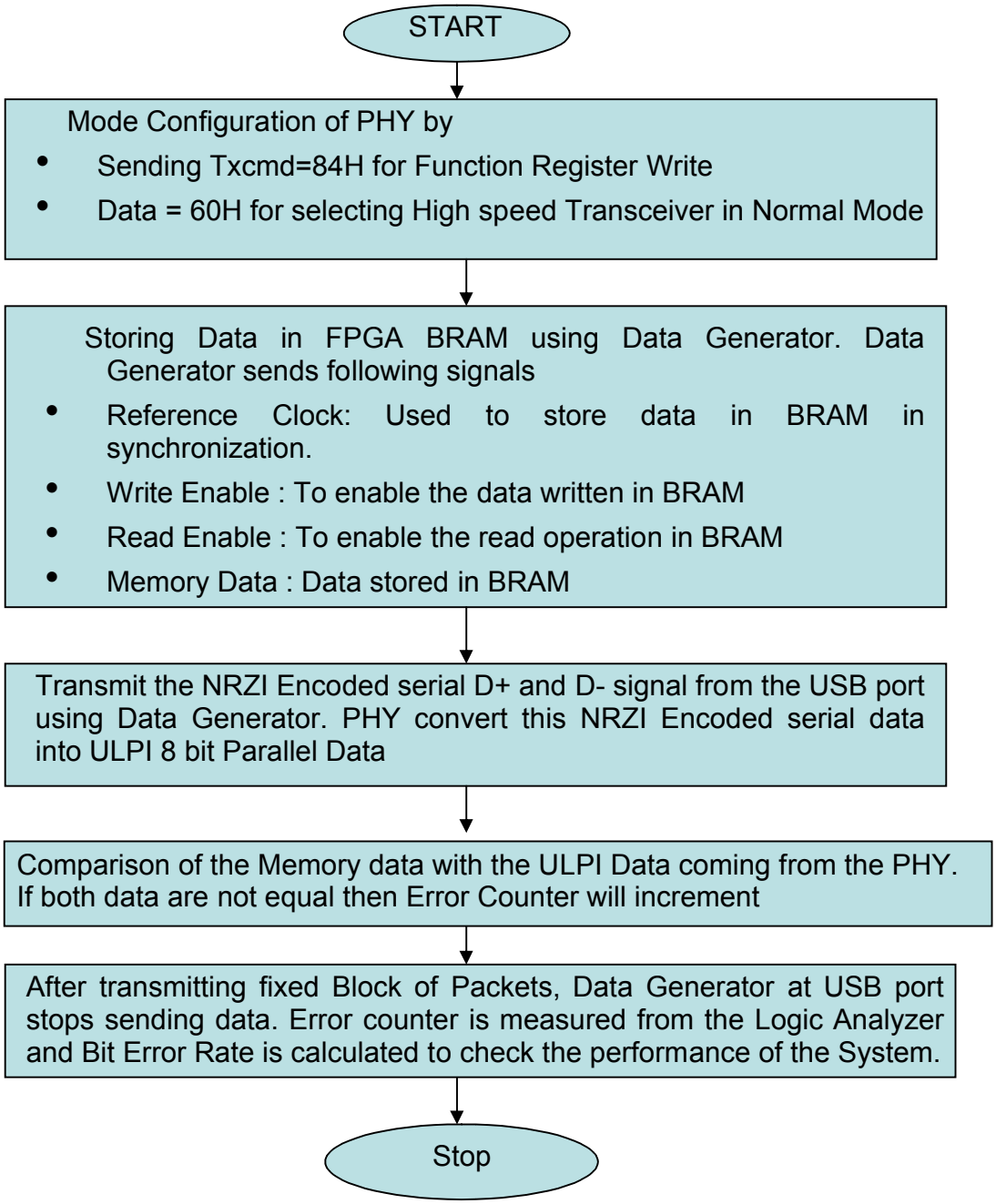

Figure 7.4 Flow Chart for the Bit Error Rate Calculation

Logic Analyzer is used to capture these waveforms from the adaptor card. Write enable is asserted to write the data in BRAM. Data generator is used to send the data to FPGA BRAM. It is controlled by the Lab view software. Data written in the BRAM is accessed through asserting read enable signal through the data generator. Data read from the BRAM is captured at Logic analyzer and can be analyzed that data is correctly written. At each clock edge, data is accessed from the BRAM as shown in figure 7.3.

7.5 Bit Error Rate Calculation

When data is properly written into the BRAM, another data generator is used to send the NRZI encoded serial data from the D+ and D- lines of the USB port. PHY Board has circuitry to remove the stuffed bit as well as decode the NRZI encoded serial data. It also contains shift and hold registers to convert serial data into parallel 8 bit ULPI data. Other control signals selected by the register write operation is used to select the High speed terminal as well as high speed transceiver. Packet transmission starts after configuration of ULPI registers are shown in figure 7.5.

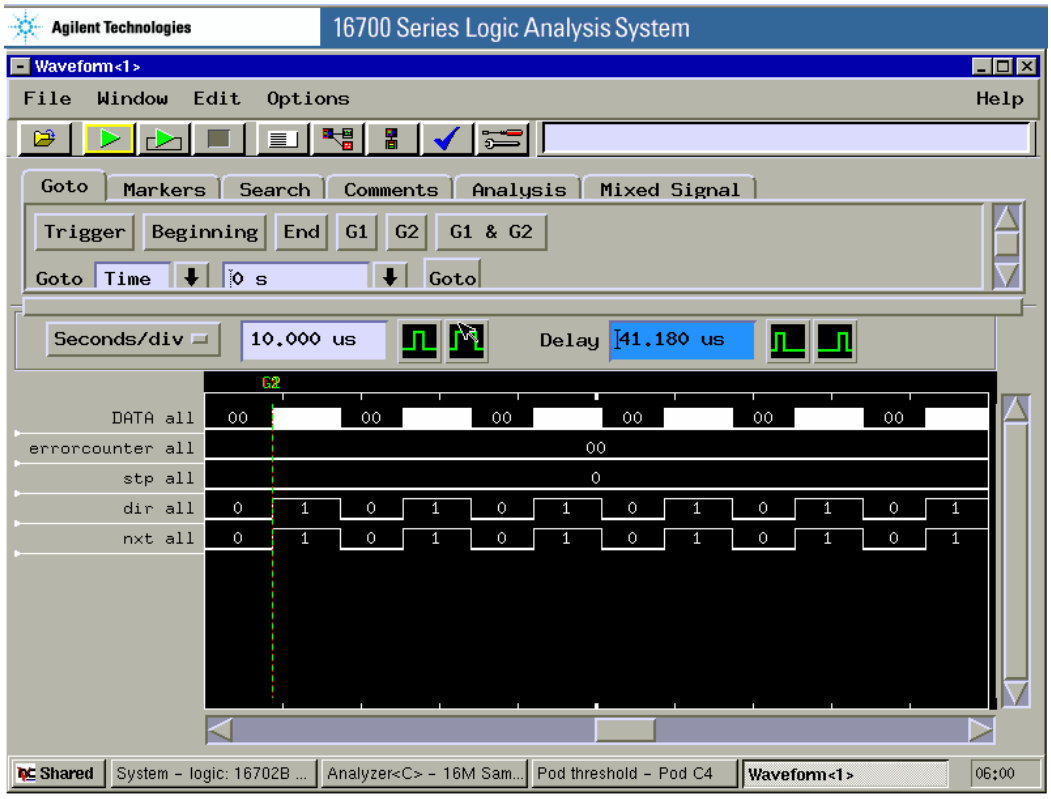

Figure 7.5 Packet Transmission

PHY board converts the NRZI encoded data into the parallel 8 bit ULPI data. Link captures ULPI data at the clock edge and access the BRAM data through the data pointer. Data pointer is automatically incremented after accessing the BRAM data. Comparator in the link compares both captured data from the ULPI interface with the BRAM accessed data. Error counter is incremented depending upon the number of bits of ULPI data remain unmatched from the BRAM data. If ULPI data matches with the BRAM data then Error counter remains there. After the long run of data packets from the USB device, Error counter is displayed and Bit error rate is calculated from the available data.

PHY must assert DIR and NXT high to receive the ULPI data from the ULPI interface. During this period, Link compares ULPI data with the memory data. When DIR switches from low to high then there is a turn around time for one clock cycle. During turn around time, neither link nor PHY can send the data. Data during the turn around time is undefined. Hence it should be ignored.

When PHY is receiving data from USB device then it gains the control of data bus by asserting the DIR. There are two conditions for receiving the correct data from ULPI interface that are:

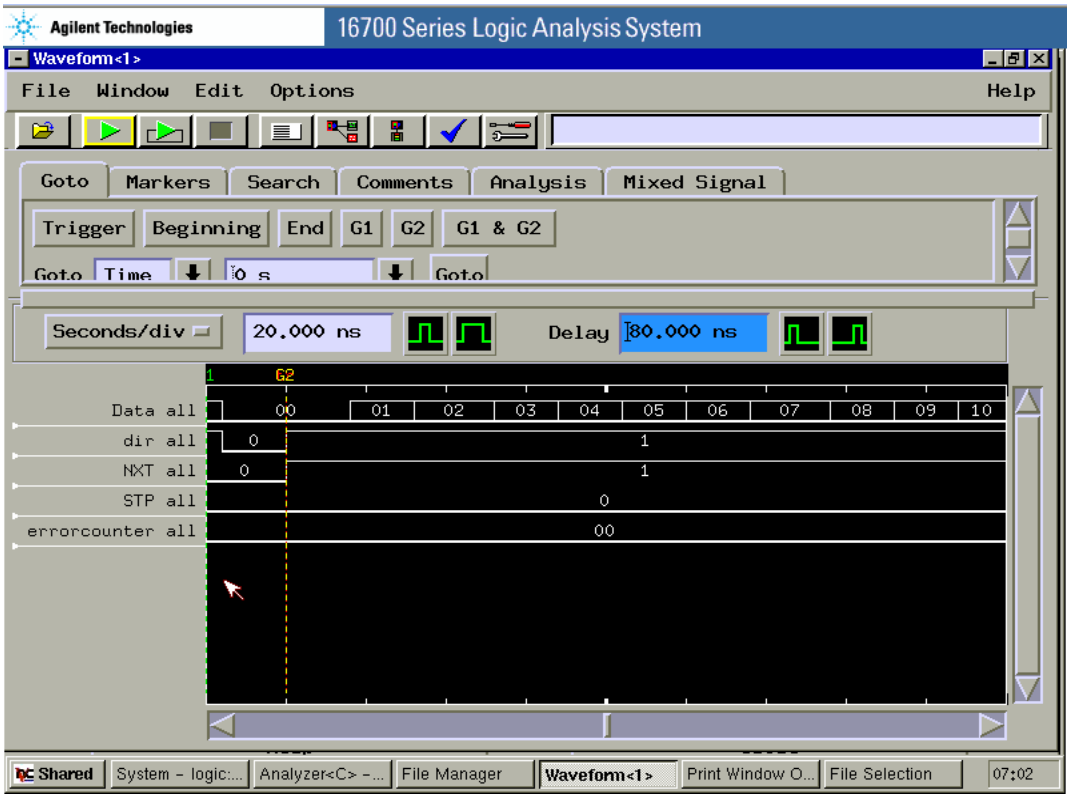

Figure 7.6 USB Receive When DIR is previously Low on Logic Analyzer

1. If DIR is previously low, the PHY must assert both DIR and NXT so that link knows that this is a USB receive data. When DIR and NXT signal assert simultaneously then USB data is received to ULPI interface at next clock edge as shown in figure 7.6.
2. If DIR is previously high, PHY will send the Rxcmd till NXT is negated. PHY will drive the data on ULPI data bus as soon as NXT goes high as shown in figure 7.7.

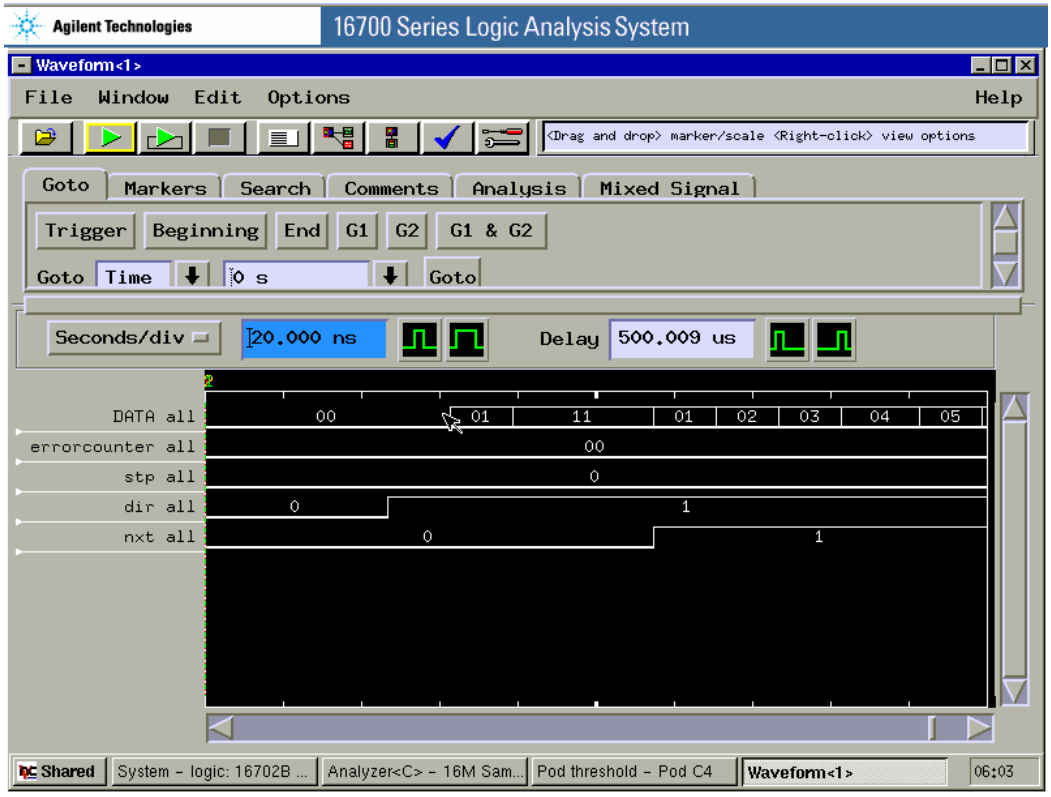

Figure 7.7 USB Receive When DIR is previously High on Logic Analyzer

Rxcmd (Receive command) is sent by the PHY to update the Link with the D+ and D- signal, Vbus voltage status, Rxactive and Rxerror status, ID line status and OTG status information.

When DIR and NXT get asserted simultaneously, PHY will take turn around time for one clock cycle as shown in figure 7.6. Whenever DIR get asserted and negated, turn around time occurs. In the turn around time, PHY gains the control of the data bus after asserting the DIR or losses the control of the data bus after de-asserting the DIR.

In the next clock cycle, NXT goes low to receive the Rxcmd. Rxcmd contains the update status information of USB receive information and interrupt events. If back to back changes are detected on the USB device status, the PHY keeps DIR asserted with NXT negated and drives back to back Rxcmd.

If ULPI bus is busy with the USB data then Rxcmd is queued in the PHY register and send when ULPI bus is available. When ULPI bus is available, PHY send the current Rxcmd (not old ones). On next clock cycle, Packet Identifier (PID) is available on the ULPI bus with NXT asserted. Packet identifier defines the type of the packet. Packet ID consists of a 4 bit identifier field followed by a 4 bit check field. The check field contains the inverted value (1's complement) of the packet identifier field value. Packet data will immediately follows the 8 bit Packet identifier with asserted DIR and NXT as shown in figure 7.8.

Packet identifiers define the purpose of the packet. It defines the format and content of the packet. Packets are divided into three categories:

- Token Packets — Token packets are sent at the beginning of a USB transaction. It is used to define the endpoint target address with the direction of transfer.

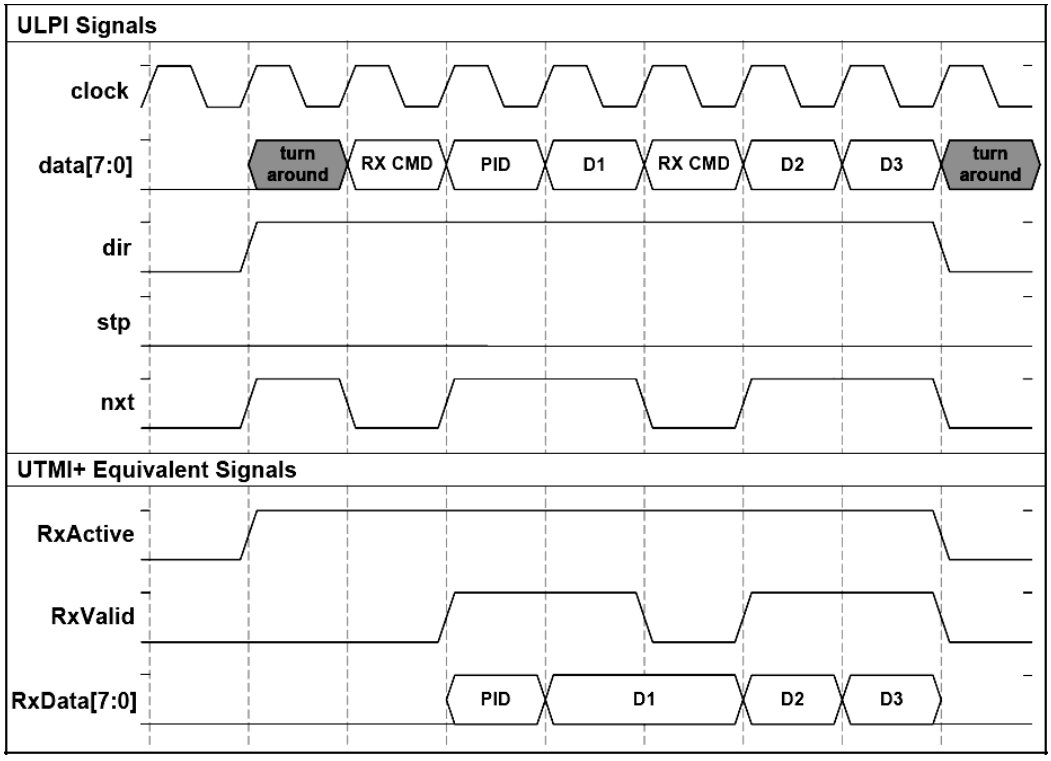

Figure 7.8 USB Receive When DIR is previously Low [6]

- Data Packets — Data packets follow the token packets during the data transactions. It contains data payloads that are to be transferred to or from the USB devices.
- Handshake Packets — Handshake packets are typically send by the USB device when it receives the data. It provides mechanism to provide the feedback to the sender to verify the successful reception of the data packet. In some cases, the USB device may send a handshake packet when system requests the data packet from the USB device. This handshake packet is used to indicate the system that device currently does not have data to send.

When DIR is asserted, PHY takes the control over the ULPI bus and it will continuously send the updated Rxcmd till NXT is low as shown in the figure 7.9. When NXT get asserted, USB device starts sending the packet identifier followed by the data to the link. It will send data till DIR and NXT is asserted. At the end of the packet, DIR and NXT get negated.

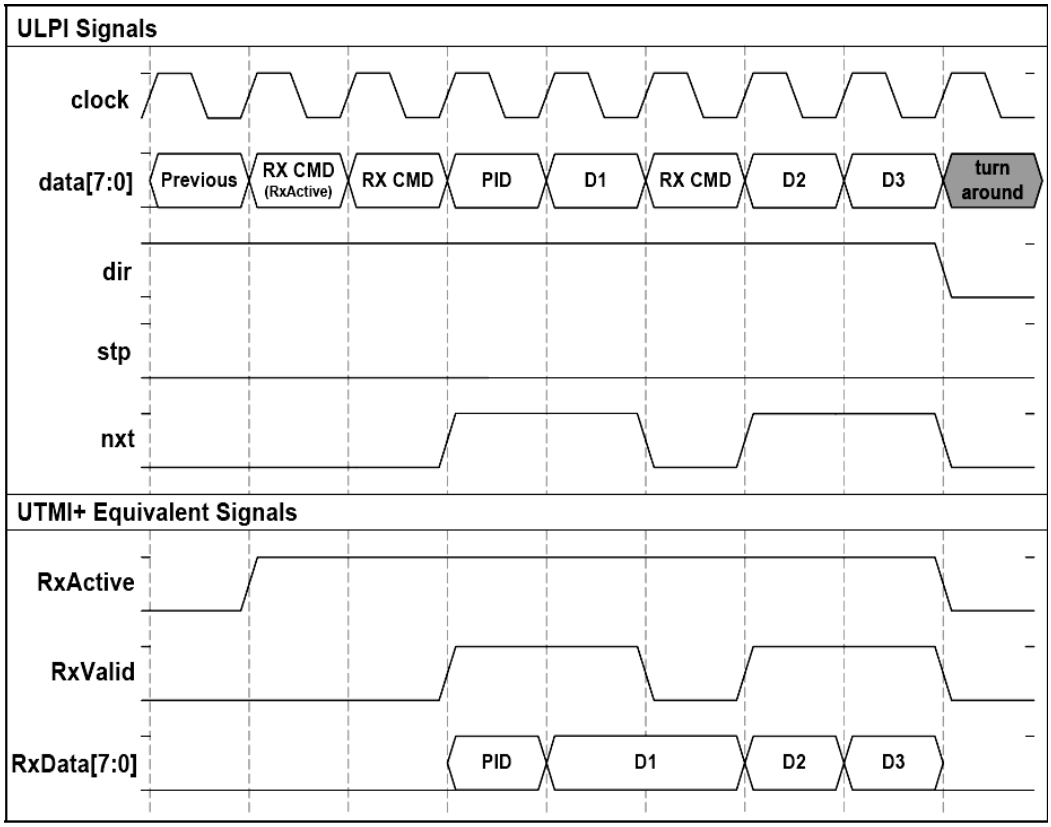

Figure 7.9 USB Receive When DIR is previously High [6]

Once data start coming from the USB device, data stored in the BRAM is accessed and compared with the ULPI data. After DIR and NXT get high simultaneously, turn around time occurs and no data is compared in that cycle. In next cycle, ULPI data is compared with the BRAM data and error counter will not increment if both data are equal as shown in figure 7.10.

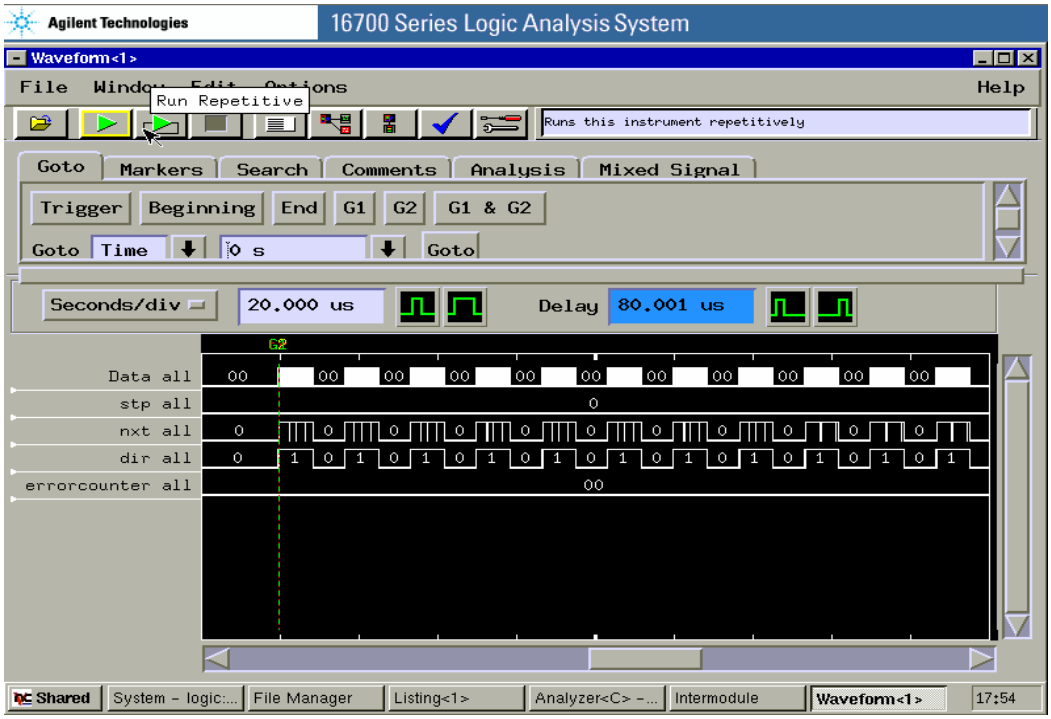

Figure 7.10 Data Packet Receive with removal of Stuffed bits

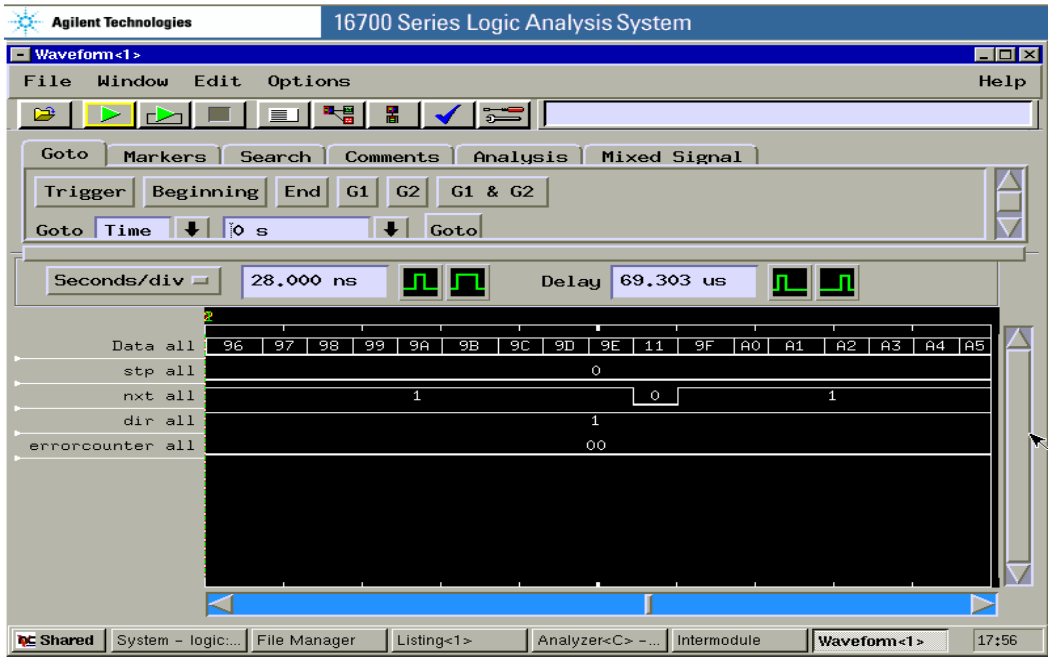

Figure 7.11 Data Packet Receive with NXT low for one cycle

During the packet comparison, DIR and NXT should remain high. As bit stuffing is done in the NRZI encoded data from the USB device, NXT will go low for one clock cycle within the packet when 8 stuffed bits are removed as shown in figure 7.11. During that clock cycle, PHY will send the Rxcmd on the ULPI data bus.

As shown in figure 7.3, Data stored in the BRAM is 01 to FF. When this data is compared with the ULPI data, error counter will start increment. As shown in figure 7.12, data 06 stored in the BRAM does not match with the 86 coming on the ULPI data bus and error counter is incremented in the next clock edge depending upon the number of unmatched bits. In the similar way, data 07 stored in the BRAM does not match with the 06 coming on the ULPI data bus and error counter is incremented from 01 to 02. In this way, error counter is measured by sending the predefined number of packets from the data generator attached at the USB port. BER is measured by dividing the number of errors reflected by the error counter by the multiplication of the number of packets sends by the data generator and number of bytes sends in one packet.

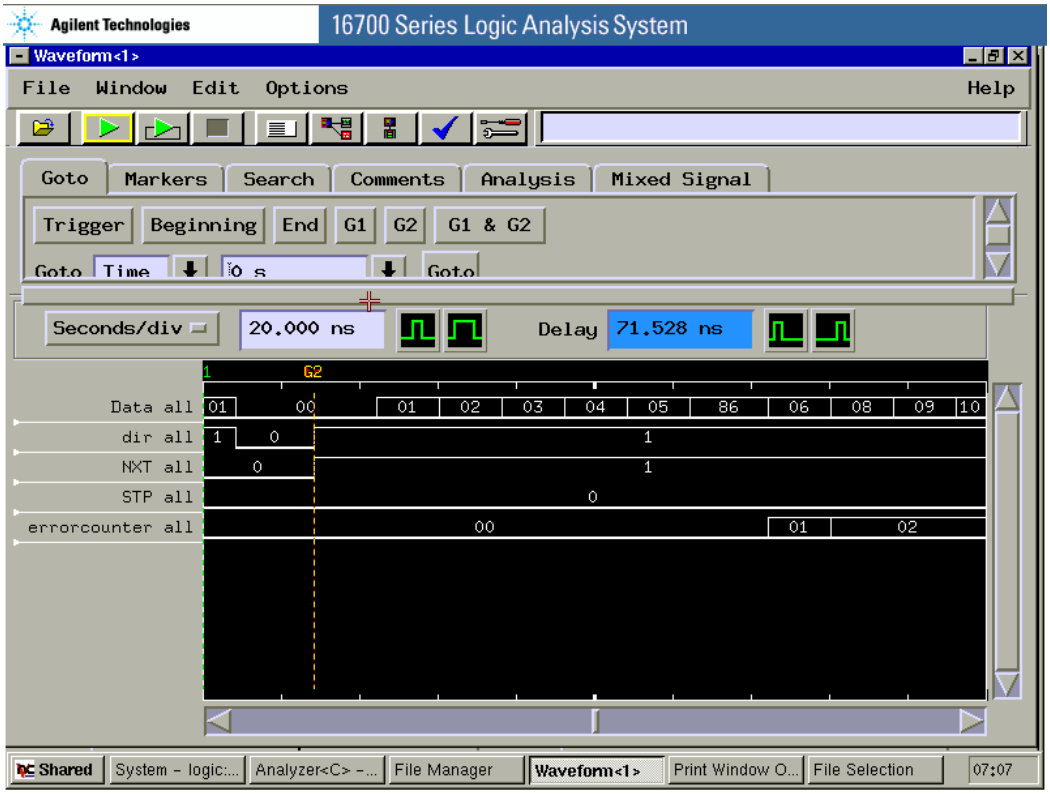

Figure 7.12 Data Receive with Error

Bit error rate is a parameter which gives an excellent indication of the performance of the USB Data communication channel. As one of the key parameter used to define USB channel performance is Bit error rate, so the knowledge of the Bit error rate enables the alteration in the other features of the USB communication to enhance the system performance. The USB is used in many applications like Holter ECG System [11]. This setup can be implemented to identify the quality figure of Holter ECG System.

CHAPTER 8
CONCLUSION AND FUTURE SCOPE

Bit Error Rate is measured by performing the test for the long duration and calculating the number of errors encountered in the known number of transmitted bits. This test is executed for 40 hours. Time required for sending a packet is measured from the Logic analyzer. Time elapsed between two packets is 16.56μs. One packet contains 510 bytes i.e. 4080 bits. The numbers of bits send in 40 hours are 3.55×10^{13}. Logic analyzer shows no error i.e. it meets the required BER (1 in 10^{13}). Hence, PHY can work as a high speed link due to high bit error rate.

Timing is a most critical task in the designing phase. Detecting timing problems early in the design process not only saves time but also permits much easier implementation of design alternatives. To ensure accurate data transfer for memory interfaces that operate at 200 MHz and beyond, timing analysis needs to play a more prominent role in the identification and resolution of system operation issues. At these frequencies, the margins for setup and hold times are tight, leaving minimal room to secure an accurate data capture and presentation window. Faster edge rates also magnify physical design effects which cause signal integrity issues that require additional settling time, shrinking timing margins.

Timing constraints are achieved by routing the clock through the clock tree available in the FPGA. Clock pads are inserted at the interface with the different environment. Cock Buffers are used to avoid loading effect of external environment on the design. It helps in reducing the transition of the input and the output signals and provide maximum period for performing the operation. Digital Clock Manager helps in providing the clock to each of the flip flops without any skew. It generates new system clocks which are phase-aligned to the input clock thus eliminating clock distribution delays. Hence, DCM offers a wide range of powerful clock management features like Clock De-skew, Frequency Synthesis and Phase Shifting.

FPGA devices incorporate large amounts of 18 Kbit block Select RAM. These are required to store large chunks of data. These complement the distributed Select RAM resources that provide shallow RAM structures implemented in CLBs. Each Virtex-II block Select RAM is an 18 Kbit true dual-port RAM with two independently clocked and independently controlled synchronous ports that access a common storage area.

Operations are synchronous for both read and write operation which is not available in the block distributed RAM. Block distributed RAM are used as a memory for storing small data and are much faster than block select RAM. While in case of accessing large chunk of data, block select RAM (BRAM) are used. In higher technologies, routing delays are more prominent than the gate delays. In case of large chunk of data stored in block distributed RAM, routing delays come in picture and it is difficult to meet tight constraints. In case of block select RAM, constraints are easily met as it is located at one place and uses only one switch matrix to route the data. Xilinx ChipScope allow the designer to view and manipulate signals directly from hardware during run time. It is very useful for debugging the design during run time.

USB data transmission in the high speed environment with good performance is a challenging field. Bit Error Rate helps in measuring the quality figure of the USB2.0 communication system. It shows its robustness in noisy environment. If Bit Error Rate of 1 in 10^{13} is achieved by the system then that system is very robust and can be used for critical projects where quality figure is the main requirement. Bit error rate finds the application in the field of medical imaging, High speed ballistic missile and others. There is a tradeoff between Bit Error Rate, power dissipation, area and speed of the system. It can be smaller (1 in 10^{8}) for low power devices like handheld devices (mobiles) and others.

This environment can be extended by transmitting the data from one link through ULPI Interface of the one PHY board and receiving the data at another link through the ULPI Interface of another PHY board. USB ports of two PHY boards are connected with each other. It can validate the two PHY board with minimum time requirement. If one of the PHY board is faulty then this setup can be used to identify the faulty PHY. Graphical Interface can be used to measure the Bit Error Rate. It can be developed using Lab view and accessed through the PCI slot of Computer motherboard. It will provide more ease and detailed report to check the quality figure of the system.

REFERENCE

[1] Don Anderson, "USB System Architecture (USB 2.0)", Addison-Wesley Developer's Press, March 2001.

[2] Kevin Lynn, "Universal Serial Bus (USB) Power Management", Conference Proceedings, pp. 434-441, November 1997.

[3] "Universal Serial Bus Specification", Revision 2.0, April 27, 2000.

[4] "USB 2.0 Transceiver Macrocell Interface Specification", Version 1.05, March 29, 2001.

[5] K. Babulu and K. Soundara Rajan, "FPGA Implementation of USB Transceiver Macrocell Interface with USB2.0 Specifications", First International Conference on Emerging Trends in Engineering and Technology, pp. 966-970, July 2008.

[6] "UTMI+ Low Pin Interface Specification", Revision 1.1, October 20, 2004.

[7] "Virtex-II Platform FPGAs: Complete Data Sheet", Product Specification, October 14, 2003.

[8] Elio A. A. De Maria, Edgardo Gho, Carlos E. Maidana, Fernando I Szklanny and Hugo R. Tantignone, "A Low Cost FPGA Based USB Device Core", 4th Southern Conference on Programmable Logic, pp. 149-154, March 2008.

[9] Jeong-Si Kim and ChaeDeok Lim, "A Remote Diagnostic Tool for USB Kernel Resources in the Embedded Linux Systems", The 8th International Conference on Advanced Communication Technology, Vol. 1, pp. 830, February 2006.

[10] Xuhui Chen, Dengyi Zhang and Hongyun Yang, "Design and Implementation of a Single-chip ARM-based USB Interface JTAG Emulator", Fifth IEEE International Symposium on Embedded Computing, pp. 272-275, October 2008.

[11] Hailong Jin and Bing Miao, "Design of Holter ECG System Based on MSP430 and USB Technology", The 1st International Conference on Bioinformatics and Biomedical Engineering, pp. 976-979, July 2007.