

FPGA BASED IMPLEMENTATION OF TEST AND DEBUG ENVIRONMENT FOR HIGH SPEED SERIAL LINKS

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

Master of Technology

in

VLSI Design & CAD

Submitted by

Tejinder Kumar
(60761023)

Under the supervision of

Dr. Kulbir Singh
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.

Tejinder Kumar

Tejinder Kumar

Certificate

This is to certify that the thesis work entitled "FPGA based implementation of test and debug environment for high speed serial links" has been carried out by Tejinder Kumar (60761023) 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

Kulbir Singh

Dr. Kulbir Singh

Assistant Professor

Thapar University,

Patiala

A.K. Chatterjee

Dr. A.K Chatterjee

Professor & Head of ECED,

Thapar University,

Patiala.

R.K. Sharma

Dr. R.K Sharma

Dean of Academic Affairs,

Thapar University,

Patiala.

Acknowledgement

Words are often too less to reveal one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped through the duration of my thesis.

First of all, I render my gratitude to the almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down in the time of crisis and showing me the silver lining in the dark clouds. Without his grace, this would not have been possible.

I acknowledge with gratitude and humility my indebtedness to **Dr. Kulbir Singh, Assistance Professor, ECED**, Thapar University, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards him for providing individual guidance and support throughout thesis work.

I express my heartfelt thanks towards **Mr. Raj Kumar Nagpal**, ST Microelectronics, Greater Noida for their valuable guidance, encouragement, inspiration and the enthusiasm with which he solved my difficulties.

I convey my sincere thanks to **Dr. A.K Chatterjee, Professor and Head of Electronics & Communication Department**, Thapar University, Patiala for his encouragement and cooperation.

I would like to thank **Mr. Parul Bansal, Mr. Gaurav mathur and Mr. Pratik Damle** (STMicroelectronics pvt. Ltd.) for their effortful support and sharing their deep rooted experiences.

I would also like to thank all staff members and my other colleagues who were always there at the need of the hour and provided with all the help and facilitates, which I required for the completion of my thesis.

My greatest thanks are to all who wished me success especially my parents. I do not find enough words which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

Tejinder

Testing and debugging are two emerging domains in the era of VLSI design. The accelerating upward trend in operating frequencies for chips used in almost all the application domains has tremendous implications for both chip design and manufacturing. The performance problems may also arise from the increasing scale of tasks performed on the chip, process complexity, more complex fabrication steps, various dielectrics and multiple threshold devices. So it becomes quite important to test and debug a chip.

At-speed testing is very much crucial for reducing test time. This report presents the “FPGA based Implementation of test and debug environment” for at speed functional testing and validation of ULPI PHY.

The fundamental purpose of this “test and debug” platform is to verify procurement ULPI specifications, what was specified is what was delivered: it verifies that the device meets the functional, performance, design, and implementation requirements identified in the procurement specifications.

This FPGA based test and debug platform also ensures the bit error rate (BER) for end to end performance and quantifies the reliability of the entire system from “bits in” to “bits out”.

Jitter is a dominant factor in affecting the BER of a USB data communication system. It may alter the sampling of the data and cause timing mismatch. So BER must be calculated under the worst case incorporating both transmitter and receiver clock jitter into the USB communication system.

Declaration Certificate	i
Acknowledgement	ii
Abstract	iii
Table of Contents	iv
List of Figures	ix
List of Tables	xii
Chapter 1 Introduction	1
1.1 Need for Testing and Debugging	1
1.2 Functional Testing and BER Measurement	1
1.3 Thesis Organization	2
Chapter 2 Literature Survey	4
2.1 Design Goals of USB 2.0	4
2.2 UTMI & ULPI Specifications	4
2.3 FPGA Based Test and Debug Environment	5
2.4 Concept of Automated and GUI Based Test and Debug Environment	5
Chapter 3 Universal Serial Bus (USB 2.0)	6
3.1 Introduction	6
3.2 Key Features of USB	6
3.3 USB Supported Transmission Data Rates	8
3.4 Detecting Device Attachment and Speed Detect	8

3.4.1	Full-Speed Device Connect	10
3.4.2	Low-Speed Device Connect	12
3.5	Detecting Device Disconnect	13
3.6	Device RESET	13
3.7	Start of Packet (SOP)	14
3.8	End of Packet (EOP)	14
3.9	NRZI Encoding	15
3.10	Bit Stuffing	15
3.11	Detecting HS Device Attachment	16
3.12	Device Reset and the Chirp Sequence	17
3.13	HS SOP & Synchronization Sequence	19
3.14	HS End of Packet	19
3.15	ON THE GO	20
3.15.1	Session Request Protocol	21
3.15.2	Host Negotiation Protocol	21
Chapter 4	USB Transceiver Macrocell Interface (UTMI)	22
4.1	Introduction	22
4.2	UTMI Features	22
4.3	UTMI Difficulty Levels	23
4.3.1	UTMI + Level 0	24
4.3.2	UTMI + Level 1	25
4.3.3	UTMI + Level 2	26
4.3.4	UTMI + Level 3	27

4.4	UTMI Interface Signals	27
Chapter 5	UTMI + Low Pin Interface (ULPI)	32
5.1	Introduction	32
5.2	Interface Signals	32
5.3	Protocol	33
5.3.1	LPI Bus Ownership	33
5.3.2	Transferring Data	33
5.3.3	Aborting Data	34
5.4	Block Diagram	35
5.5	UTMI vs. ULPI Interface	36
5.6	Registers	37
5.6.1	Register Map	37
5.6.1.1	Function Control	38
5.6.1.2	Interface Control	39
5.6.1.3	OTG Control	41
5.7	Transmit Command Byte (TX CMD)	41
5.8	Receive Command Byte (RX CMD)	42
Chapter 6	Un-wrapper	44
6.1	Introduction	44
6.2	Basic Set up utilizing Un-Wrapper	45
6.3	Observations	46
6.3.1	High Speed Mode	46
6.3.2	FS Speed Mode	47

Chapter 7	FPGA Based Implementation of Test and Debug Environment for High Speed Serial Links	50
7.1	Introduction	50
7.2	FPGA Implemented Test and Debug Environment	50
7.3	Functional Validation	51
7.4	Register Write and Read Operations	51
7.5	Register Write	51
7.6	Register Read	52
7.7	Simulation Results for Register Write and Read	53
7.8	FS/HS/LS Data Transmission	55
7.8.1	HS Data Transmission	55
7.8.2	FS Data Transmission	56
7.8.3	LS Data Transmission	57
7.9	Data Packet Transmission	58
7.10	Full Speed / Low Speed Serial Mode	59
7.10.1	Data Line Definition for FsLsSerialMode	59
7.10.2	Steps to Implement FsLsSerialMode Transmission	60
7.10.3	Steps to Implement FsLsSerialMode Reception	61
7.11	Low Power Mode	62
7.12	Bit Error Rate (BER	64
7.12.1	Measurement of BER	64
7.12.1.1	PHY Configuration	64
7.12.1.2	Memory Write	65

7.12.1.3	Data Comparison	67
7.13	Block Diagram Showing BER Code Description	70
7.14	Flow Chart of the Code	71
Chapter 8	Conclusions and Scope for Future Work	74
8.1	Conclusions	74
8.2	Scope for Future Work	75
Chapter 9	References	76

List of Figures

Figure 3.1	Signaling Interface USB Hub and Attached USB Full-Speed Device	9
Figure 3.2	Connect Sequence from Port Power through Device Reset	10
Figure 3.3	Full-Speed Device Detection	11
Figure 3.4	Signal States during FS Device Attachment	11
Figure 3.5	Low-Speed Device Detection	12
Figure 3.6	Line States during Low-Speed Device Connection	12
Figure 3.7	Signaling State during Device Disconnect	13
Figure 3.8	Reset Signaling States	13
Figure 3.9	SOP Is Recognized at the Start of the Synchronization Sequence	14
Figure 3.10	Full- or Low-Speed EOP Signaling	15
Figure 3.11	Stuffed Bits	16
Figure 3.12	Sequence of Events from Device Connect to High-Speed Operation	17
Figure 3.13	Chirp Sequence Used to Detect High-Speed Capable Device	18
Figure 3.14	High-speed Synchronization Sequence and SOP	19
Figure 3.15	High-speed EOP Detection	20
Figure 4.1	USB2.0 Transceiver	24
Figure 4.2	UTMI + Levels	25
Figure 4.3	UTMI+ Level 0 entity diagram	26
Figure 4.4	UTMI+ Level 1 entity diagram	27
Figure 4.5	UTMI+ Level 2 entity diagram	28

Figure 5.1	LPI Generic Data Bus Ownership	34
Figure 5.2	LPI Generic Data Transmit Followed by Data Receive	35
Figure 5.3	Link Asserts stp to Receive Data	36
Figure 5.4	Block Diagram of PHY	36
Figure 5.5	Creating a ULPI System Using Wrappers	37
Figure 6.1	Un-Wrapper from ULPI to UTMI + Core	45
Figure 6.2	Basic Set up Utilizing Un-Wrapper	46
Figure 6.3	HS Data Transmission (ULPI Interface)	48
Figure 6.4	HS Data Transmission (Dp, Dn)	48
Figure 6.5	FS Data Transmission (ULPI Interface)	49
Figure 6.6	FS Data Transmission (Dp, Dn)	50
Figure 7.1	FPGA Implemented Test and Debug Environment	52
Figure 7.2	Register Write	53
Figure 7.3	Register Read	54
Figure 7.4	Waveform for Function Control Register Write and Read Together	55
Figure 7.5	Waveform for Interface Control Register Write and Read Together	55
Figure 7.6	Waveform for OTG Control Register Write and Read Together	56
Figure 7.7	ULPI PHY Configuration in HS Mode	57
Figure 7.8	HS Data Transmission	57
Figure 7.9	ULPI PHY Configuration in FS Mode	58
Figure 7.10	FS Data Transmission	58
Figure 7.11	LS Data Transmission	59
Figure 7.12	Data Packet Transmission	59

Figure 7.13	Data Transmission in FsLsSerialMode (NCSIM)	61
Figure 7.14	Data Transmission in FsLsSerialMode (Logic Analyzer)	62
Figure 7.15	Data Receive in FsLsSerialMode (NCSIM)	63
Figure 7.16	Data Receive in FsLsSerialMode (Logic Analyzer)	63
Figure 7.17	Clock Suspended in Low Power Mode (NCSIM)	64
Figure 7.18	Clock Suspended in Low Power Mode (Logic Analyzer)	64
Figure 7.19	Function Control register Configuration for HS mode	66
Figure 7.20	Labview Based GUI Interface to Automate the Data Generator	67
Figure 7.21	Memory Read	68
Figure 7.22	USB Receive While dir was Previously Low	69
Figure 7.23	Error Counter at Value 0	70
Figure 7.24	Error Counter at Non-Zero Value	70
Figure 7.25	Block Diagram Showing BER Code Description	71
Figure 7.26	Flow Chart Depicting Working of FPGA Implemented Test and Debug Platform	72

List of Tables

Table 3.1	Applications and Relative Performance Required	8
Table 4.1	Clock Description	29
Table 4.2	XcvtSelect Description	30
Table 4.3	TermSelect Description	30
Table 4.4	OpMode Signal Description	31
Table 4.5	Line State Signals Description	32
Table 4.6	SuspendM Signal Description	32
Table 5.1	LPI Generic Interface Signals	33
Table 5.2	Function Control Register	39
Table 5.3	Interface Control Register	40
Table 5.4	OTG Control Register	42
Table 5.5	Transmit Command Byte Format	43
Table 5.6	Receive Command Byte Format	44
Table 7.1	Serial Mode Signal Mapping for 6-Pin FsLsSerialMode	60
Table 7.2	Serial Mode Signal Mapping for 3-Pin FsLsSerialMode	61

Abbreviations

ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
USB	Universal Serial Bus
UTMI	USB Transceiver Macrocell Interface
ULPI	UTMI + Low Pin Interface
HS	High Speed
FS	Full Speed
LS	Low Speed
NRZI	Non Return to Zero
TX CMD	Transmission Command
RX CMD	Receive Command
SRP	Session Request Protocol
HNP	Host Negotiation Protocol
LPI	Low Pin Interface
BER	Bit Error Rate

1.1 Need for Testing and Debugging

The Universal Serial Bus (USB) Standard [1] 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 [2]. 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 test and debug platform.

This report presents the “FPGA based implementation of test and debug platform” for at speed functional testing and validation of ULPI PHY. This test and debug platform incorporates the un-wrapper [3] that converts the ULPI interface to UTMI core to facilitate the testing and debugging of ULPI PHY by avoiding repeated burning of FPGA.

1.2 Functional Testing and BER Measurement

The fundamental purpose of this test and debug platform is to verify procurement ULPI specifications [3] what was specified is what was delivered: it verifies that the device meets the functional, performance, design, and implementation requirements identified in the procurement specifications.

One of the changes that modern digital communications systems have brought is the need for end-to-end performance measurements. The measure of that performance is usually bit-error rate (BER) [4], which quantifies the reliability of the entire system from “bits in” to “bits out”.

Bit errors can be caused in serial communication systems due to improper design or due to random events. Intrinsic and extrinsic noise factors may play a vital role for causing bit errors.

Intrinsic errors are those errors due to the design, components and implementation of a Link. These errors can also be caused by internal noise sources (e.g. thermal noise), poor electrical

connections, and (with some systems) receiver sampling errors. The extrinsic sources of errors are caused by external sources and outside influences. These include power supply noise/ripple; electrostatic discharge; electromagnetic interference; cable/connector vibrations.

Jitter is a third dominant factor after intrinsic and extrinsic noise sources in affecting the BER of a USB data communication system. It may alter the sampling of the data and cause timing mismatch. So BER must be calculated under the worst case incorporating both transmitter and receiver clock jitter into the USB communication system [5].

This FPGA implemented test and debug platform also ensures the bit error rate (BER) for end to end performance and quantifies the reliability of the entire system from “bits in” to “bits out”.

1.3 Thesis Organization

This thesis report has been organized into seven chapters. A brief introduction of each chapter is as follows:

Chapter 2 deals with the review of literature.

Chapter 3 describes the design goal of USB 2.0 and the basic USB 2.0 communication protocol. It also delineates various USB 2.0 features and capability to support Low speed (LS), full speed (FS) and High speed (HS) data transmission rates.

Chapter 4 explains USB2.0 Transceiver Macrocell Interface [6,7] and its basic features. It includes different complexity levels of UTMI core and explains its various Interface signals.

Chapter 5 explains a UTMI + low pin interface (ULPI) between a Link and a PHY. Its Interface signals and registers are defined and the basic communication protocol is described.

Chapter 6 describes a un-wrapper to un-wrap the ULPI interface to convert it to the UTMI core. The purpose of un-wrapping is to facilitate the debugging process.

Chapter 7 signifies the importance of testing and debugging of the semiconductor devices and describes the “FPGA implemented test and debug environment” for verification of procurement ULPI specifications. This chapter also signifies the importance of quantifying end to end performance of the entire digital communication system through bit error rate (BER) calculation to accurately define the reliability of the system for critical and sensitive applications like medical science equipments and ballistic missiles and lists the various

causes which degrade the end to end performance of a system. Practical results verifying the procurement ULPI specifications and results for calculation of bit error rate (BER) have been described in this chapter.

2.1 Design Goals of USB 2.0

Don Anderson [8] reported that interfaces used in the original IBM PC designs due to its numerous shortcomings have been completely replaced by USB interfaces (ULPI and UTMI) due to its wide range of applications and various attractive features like low cost, hot pluggable, single connector type, cable power, 127 devices, system resource requirement eliminated, error detection and recovery, power conservation.

As per the Universal serial bus specifications [1], USB 1.0 and 1.1 (i.e., 1.x) systems can support only 1.5Mb/s (low-speed) and 12Mb/s (full-speed) transactions. Systems based on USB 2.0 are designed to support 12 Mb/s (high-speed), 1.5Mb/s (low-speed) and 12Mb/s (full-speed). This provides backward compatibility to 1.x devices, while significantly extending USB performance, and consequently, increasing the number of devices that can be supported.

As mentioned in On-The-Go Supplement to the USB 2.0 Specification [9, 10], many of the portable devices would benefit from being able to communicate to each other over the USB interface because USB 2.0 allows a portable device (OTG device) to take on the role of a limited USB host, without the burden of supporting all the above functions of a PC.

K.Lynn[11] explained that Power management and distribution is a major factor in correctly designing USB (Universal Serial Bus) peripherals. Proper methods of designing USB peripheral power distribution are crucial to ensure full compliance with the USB specification.

2.2 UTMI & ULPI Specifications

As per the UTMI specifications [6, 7] and ULPI specifications [3], USB controller utilizes two types of interfaces ULPI and UTMI + core to interface with the external world. UTMI specifications [7] describe that UTMI + core have different complexity levels. Any level can be utilized depending upon the complexity of the application.

As per the UTMI specifications [6], has 100 interface signals which requires large number of pins out. But according to ULPI specifications [3], in contrast to UTMI + core, ULPI interface requires only 12 interface signals that allow a lower pin count option for connecting to an external transceiver that may be based on the UTMI+ specification. The pin count reduction is achieved by having relatively static UTMI+ signals be accessed through registers and by providing a bi-directional data bus that carries USB data and provides a means of accessing register data on the ULPI transceiver.

2.3 FPGA Based Test and Debug Environment

Elio A. A. et al [2] suggests a very unique solution for system implementations to include the USB communication standard in a variety of applications and introduces a low cost USB core, developed to address a variety of applications using FPGA integrated circuits. This suggestion gives the idea to implement “FPGA based test and debug platform” for functional testing and validation of the ULPI PHY.

One of the very important aspects of testing and debugging is bit error rate (BER). Bit error rate (BER) ensures the end to end performance and quantifies the reliability of the entire system. In other words, it is a rate at which the errors may occur in a transmission system.

L. Schooley [5] *et al* explained that, Carrier phase jitter degrades the performance of digital communication systems. So bit error rate must be calculated in the adverse conditions of Jitter which assures the stability of digital communication system in all conditions.

2.4 Concept of Automated and GUI Based Test and Debug Environment

The techniques used so far are not so much user friendly. Lot of human intervention is required for their proper operation and performance. Now as suggested by Zhang Xinjun [12], integrated testing and debugging system that integrates program testing and debugging processes into a series of consecutive processes and supports automation of each of these processes.

Moreover, GUI based automated testing and debugging platforms are on the shelf for some systems. But GUI based automated testing and debugging platforms are still under development.

3.1 Introduction

This chapter describes the design goals of USB [8] 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 [13, 14, 15, 16].

3.2 Key Features of USB:

This section lists the key features that comprise the USB implementation [8].

1. Low Cost:

USB can reduce the overall cost of peripheral design and system support rather 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. 127 Devices:

Each USB port supports the attachment of 127 USB devices. Hub devices provide additional ports for attaching other USB devices. Hubs can be stand-alone devices, or can be integrated into other USB peripherals such as printers and key-

boards etc.

4. 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.

5. 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.

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

The USB 2.0 supports three device speeds:

- ❖ Low Speed (1.5 Mbps),
- ❖ Full Speed (12 Mbps) and
- ❖ High Speed (480 Mbps)

7. System Resource Requirement Eliminated:

I/O address conflicts are also quite common in the PC environment. In contrast to it USB devices do not require any memory or I/O address space and need no IRQ lines.

8. Power Conservation:

USB devices automatically enter the suspend state after 3ms of no bus activity. Devices are placed into the suspend state under software control. During suspend state, USB devices can consume no more than 500µa of current.

9. 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.

3.3 USB Supported Transmission Data Rates

- ❖ 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. Table 3-1 categorizes the types of devices that fall into these performance ranges.

Table 3.1 - Applications and Relative Performance Required [8]

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

3.4 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 which 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 3.1 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 $15k\Omega$ as shown in figure 3.1. 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.

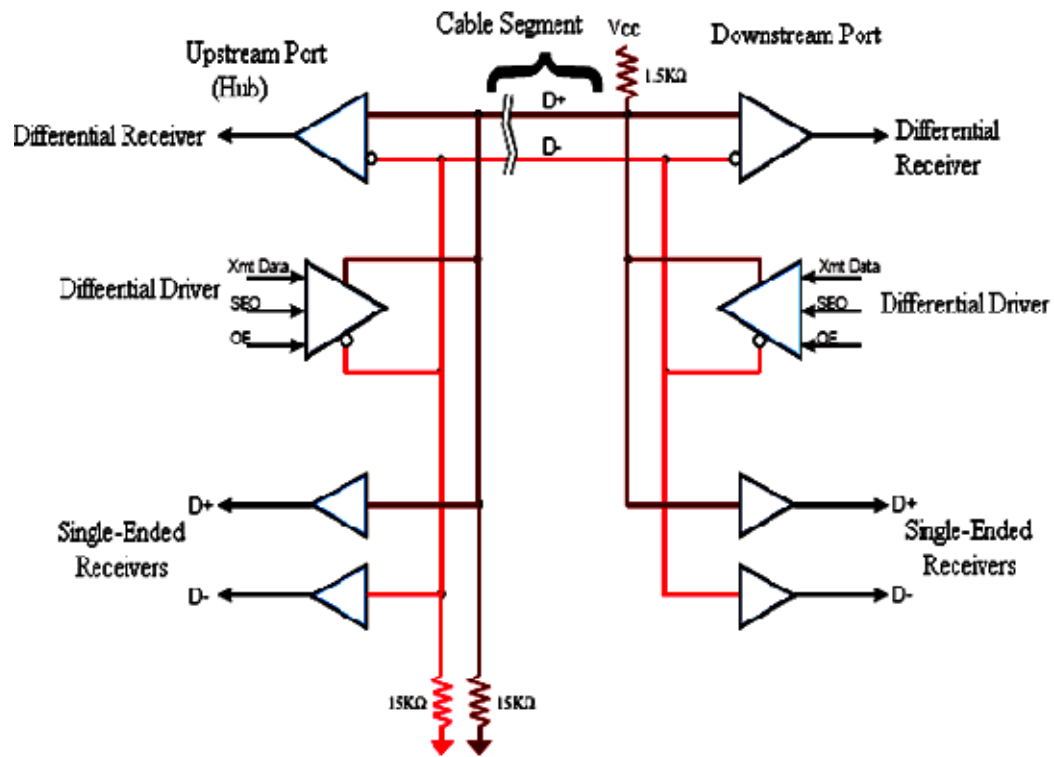


Figure 3.1 - Signaling Interface USB Hub and Attached USB Full-Speed Device [8]

When Hub detects any device attachment at its USB port then following timing events occurs as shown in a figure 3.2:

1. Δt_1 : It specifies the time required for a hub to apply valid power to a port once a device is attached to the Hub.
2. Δt_2 : It specifies the delay from port power valid till D+ (D-) is pulled above V_{IH} at the hub port receiver.

3. Δt_3 : This interval ensures that the signals are de-bounced. This de-bounce intervals enforced by software, after which software can safely check hub status to determine if 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.

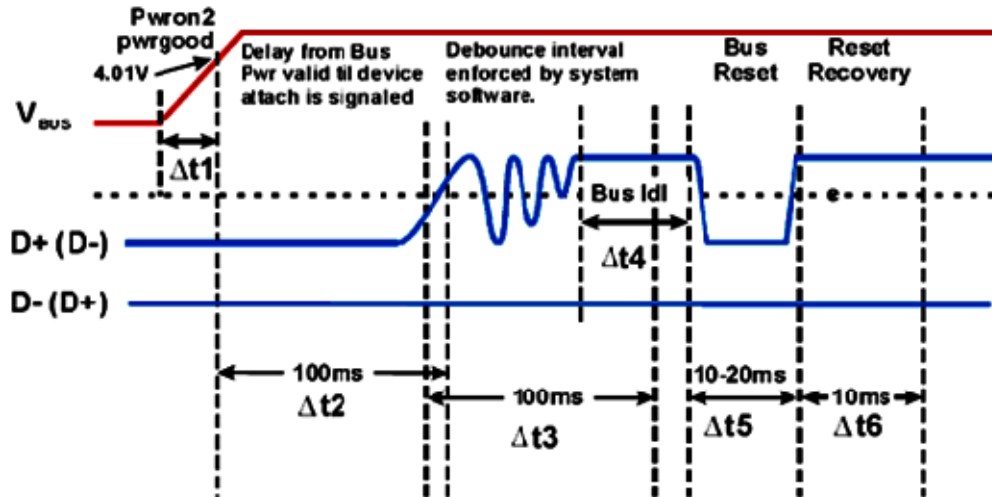


Figure 3.2 - Connect Sequence from Port Power through Device Reset [8]

4. Δt_4 : After D+ or D- line has settled (during Δt_3), 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. Δt_5 : 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. Δt_6 : 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.

3.4.1 Full-Speed Device Connect

Figure 3.3 illustrates a FS device connected to a hub port. When 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 $15\text{K}\Omega$ and the device's pull-up resistor of $1.5\text{K}\Omega$ on D+ line. Since the Hub's pull-down resistor value is $15\text{K}\Omega$ and the USB device pull-up resistor value is $1.5\text{K}\Omega$, D+ will raise to approximately 90% of V_{cc} . When the hub detects that D+ approaches V_{cc} while the other line D- remains near ground, it knows that a full speed device has been attached as shown in a figure 3.4.

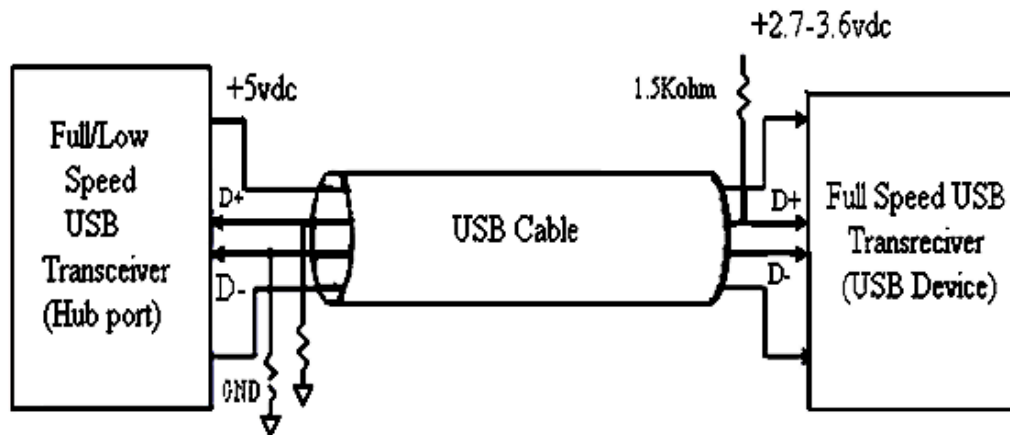


Figure 3.3 - Full-Speed Device Detection [8]

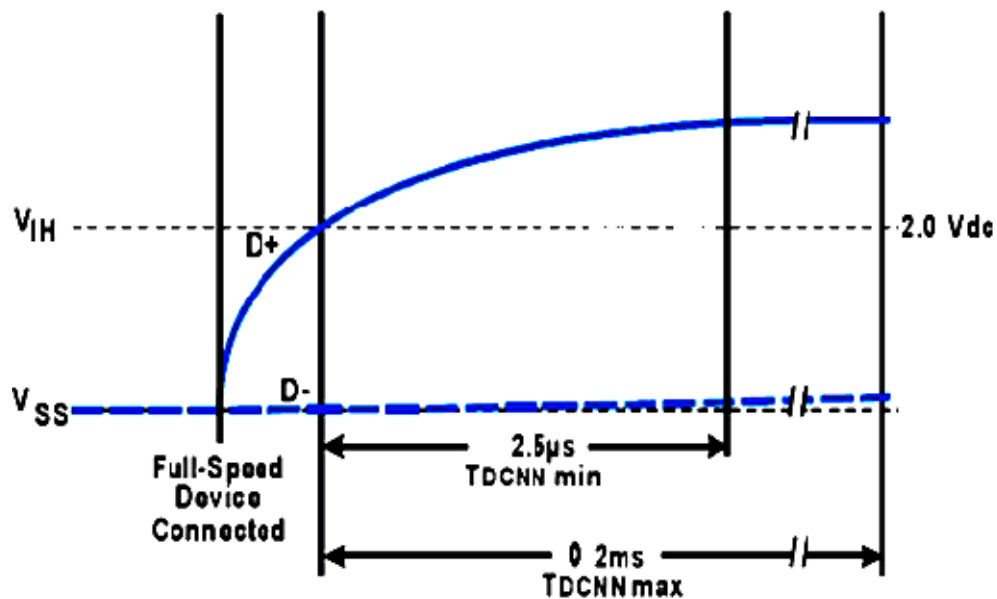


Figure 3.4 - Signal States during FS Device Attachment [8]

3.4.2 Low-Speed Device Connect

Figure 3.5 illustrates a LS device connected to a hub port. 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 $15K\Omega$ and the device's pull-up resistor of $1.5K\Omega$ on D- line. Since the Hub's pull-down resistor value is $15K\Omega$ and the USB device pull-up resistor value is $1.5K\Omega$, 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 3.6.

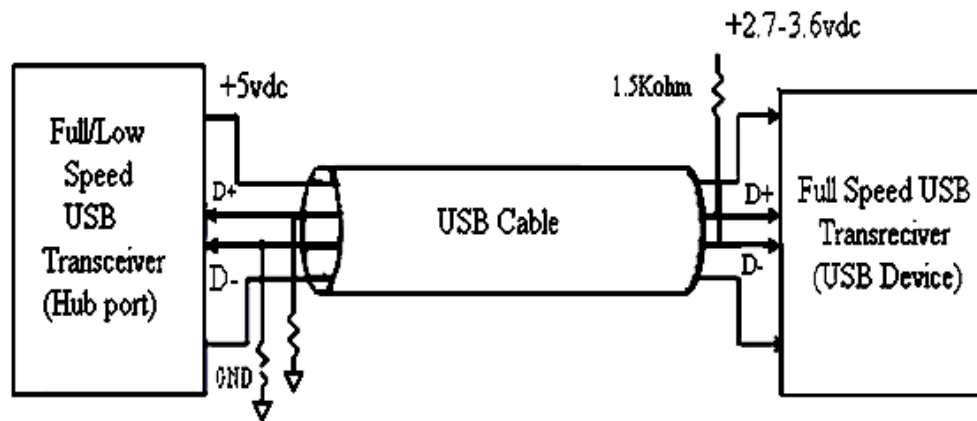


Figure 3.5 - Low-Speed Device Detection [8]

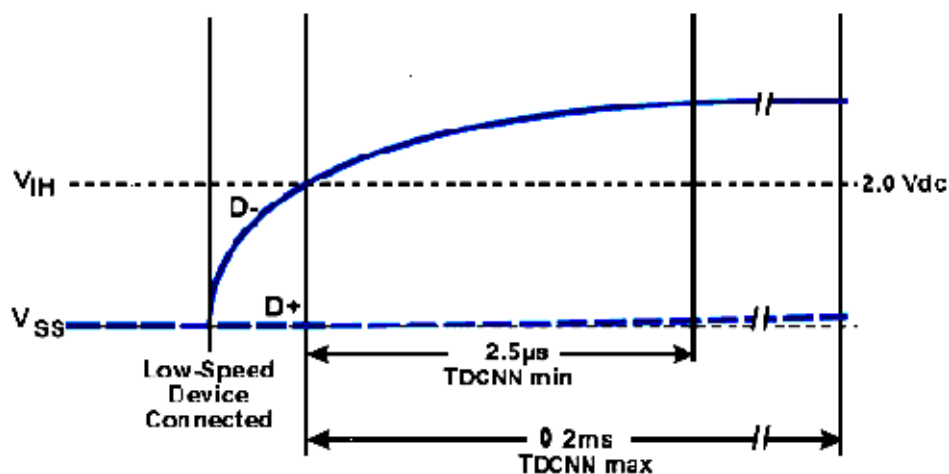


Figure 3.6 - Line States during Low-Speed Device Connection [8]

3.5 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 3.7. A hub detects disconnect when it observes a single-ended zero (when both D+ and D- lines fall below VIL).

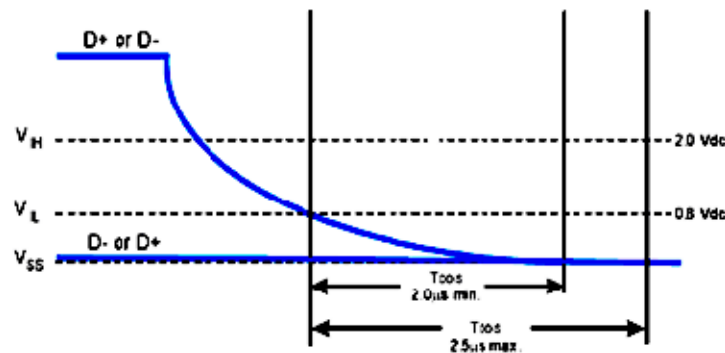


Figure 3.7 - Signaling State during Device Disconnect [8]

3.6 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).

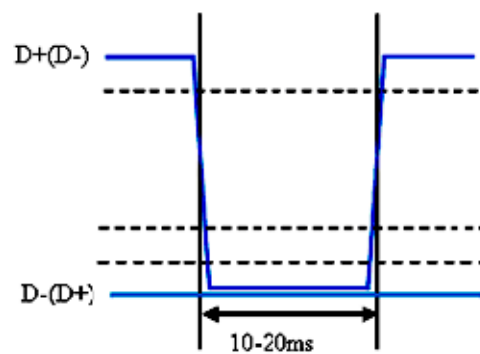


Figure 3.8 - Reset Signaling States [8]

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.

3.7 Start of Packet (SOP)

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 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.

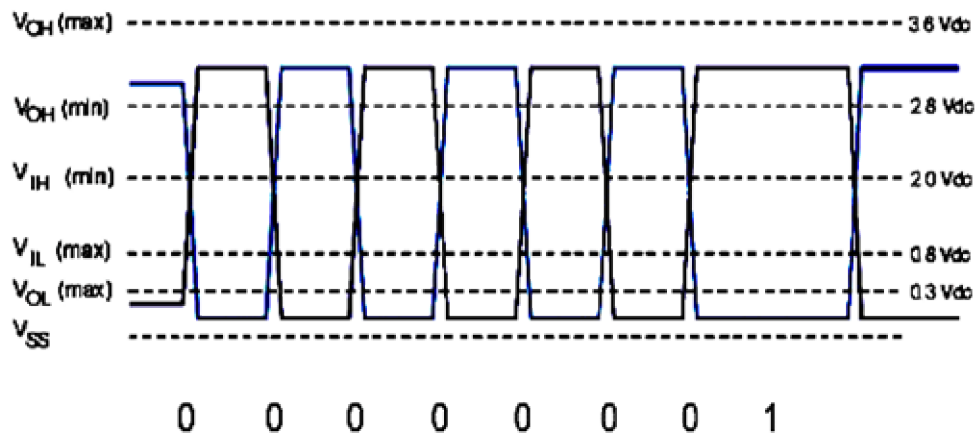


Figure 3.9 - SOP Is Recognized at the Start of the Synchronization Sequence [8]

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 change from J state to the K state which is the indication of an incoming packet. Figure 3.9 illustrates the synchronization sequence and the SOP.

3.8 End of Packet (EOP)

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 3.10 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.

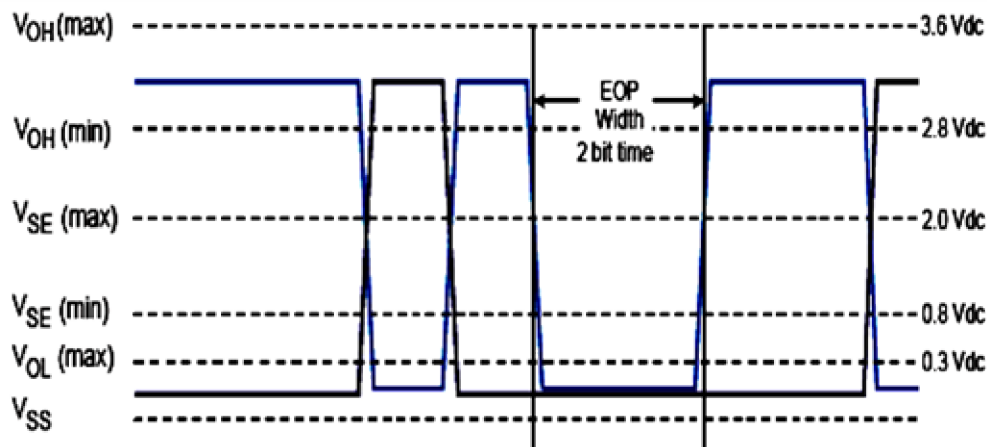


Figure 3.10 - Full- or Low-Speed EOP Signaling [8]

3.9 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 loss of synchronization at receiver end. The solution of this problem is to combine

bit stuffing with the NRZI encoding.

3.10 Bit Stuffing

Bit stuffing is used to forcibly send the transition when it met six consecutive 1's in the USB data transmission. 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.

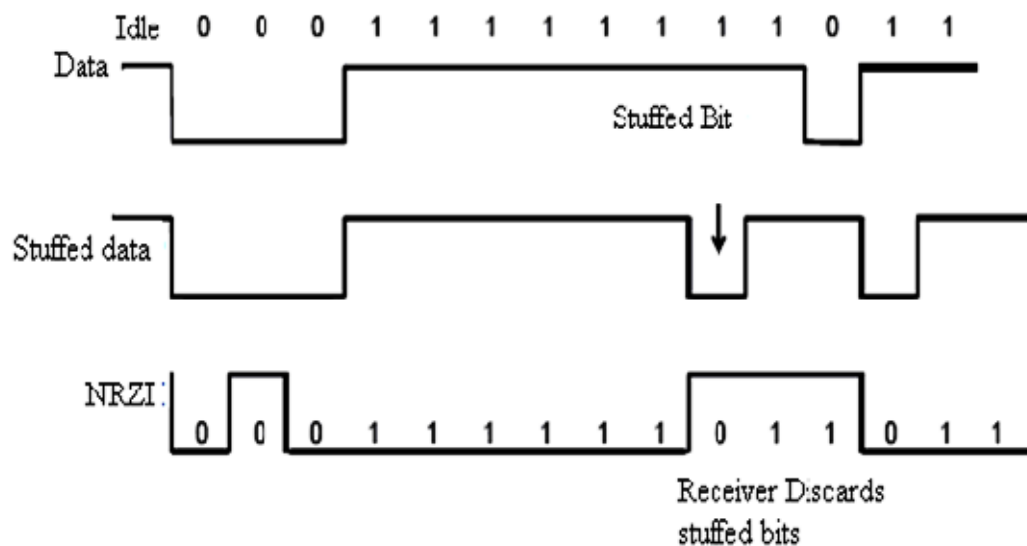


Figure 3.11 - Stuffed Bit [8]

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 sent by the receiver. Following figure 3.11 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 results in two

consecutive zeros in the stuffed data stream.

3.11 Detecting HS 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 (12 Mb/s) while high speed hubs 2.0 support low speed, full speed as well as high speed devices. When we connect high speed device to the high speed hub then handshaking is performed.

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 3.12.

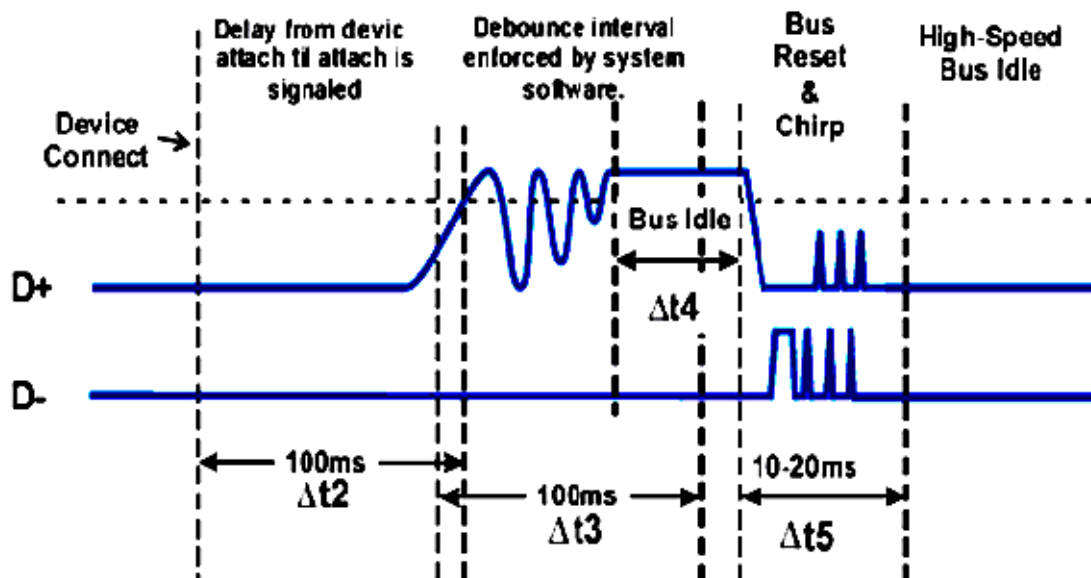


Figure 3.12 - Sequence of Events from Device Connect to High-Speed Operation [8]

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 seq-

uence is used to help controller to differentiate between full and high speed devices.

3.12 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 3.13 shows the chirp sequence.

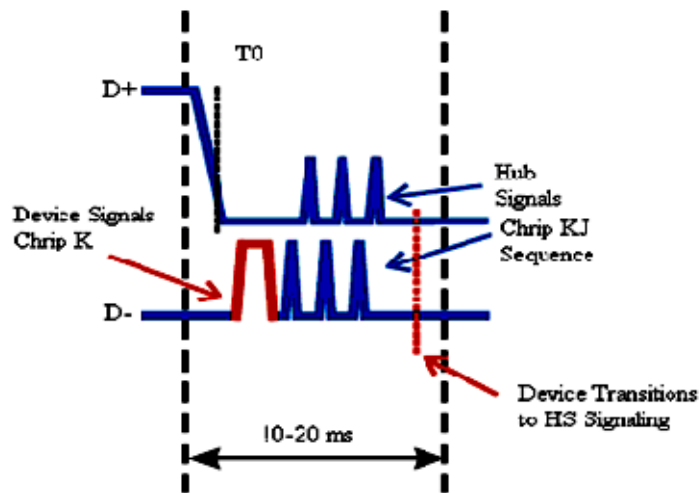


Figure 3.13 - Chirp Sequence Used to Detect High-Speed Capable Device [8]

Each step in the chirp sequence is explained below:

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 3.13.
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 Ks and Chirp Js.
5. When USB device detects six chirps (3 KJ pairs), it must work as a high speed device within 500 μ s. 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.

3.13 HS SOP & Synchronization Sequence

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

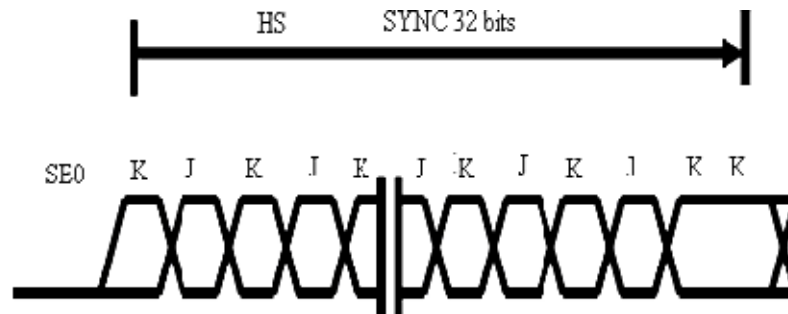


Figure 3.14 - High-Speed Synchronization Sequence and SOP [8]

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.

3.14 HS End of Packet

Every data packet ends with an EOP sequence as illustrated in figure 3.15. 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.

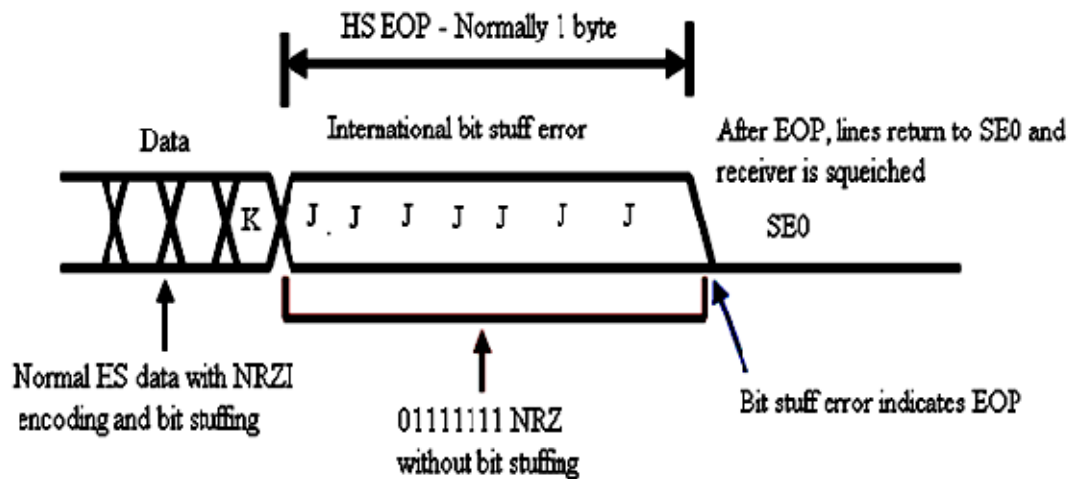


Figure 3.15 – High-Speed EOP Detection [8]

3.15 On the Go

USB has become a popular interface for exchanging data between PC and peripherals. An increasing number of portable peripherals are using the USB interface to communicate with the PC. USB communication can only take place between a host and a peripheral. However, in order to qualify as a PC host, a device must have several characteristics as:

1. Storage for a large number of device drivers.
2. The ability to source a large current.
3. Series “A” host connector receptacle.

It is not practical for many portable devices to have all of the above characteristics. An On-The-Go device must include the following features and characteristics:

1. A limited Host capability
2. Full-speed operation as a peripheral (high-speed optional)
3. Full-speed support as a host (low-speed and high-speed optional)
4. Targeted Peripheral List
5. Session Request Protocol
6. Host Negotiation Protocol
7. One and only one connection: a Micro-AB receptacle.
8. Minimum IA_VBUS_OUT output on VBUS
9. Means for communicating messages to the user

When acting as Host, an On-The-Go device is not required to support operation with all types of USB peripherals. It is up to the manufacturer of each On-The-Go device to determine what peripherals the On-The-Go device will support and provide a list of those peripherals. This is called the On-The-Go device’s “Targeted Peripheral List”.

3.15.1 Session Request Protocol

The Session Request Protocol (SRP) allows a B-device to request the A-device to turn on VBUS and start a session. This protocol allows the A-device, which may be battery powered, to conserve power by turning VBUS off when there is no bus activity while still providing a means for the B-device to initiate bus activity. At the start of a session, the A-device defaults to having the role of Host [9, 10].

Any A-device, including a PC or laptop, is allowed to respond to SRP. Any B-device, including a standard USB peripheral, is allowed to initiate SRP. An On-The-Go device is

required to be able to initiate and respond to SRP. There are two methods that shall be used by the B-device to request that the A-device begin a session. They are called “data-line pulsing” and “VBUS pulsing”.

3.15.2 Host Negotiation Protocol

The Host Negotiation Protocol (HNP) allows the Host function to be transferred between two directly connected On-The-Go devices and eliminates the need for a user to switch the cable connections in order to allow a change in control of communications between the devices. HNP will typically be initiated in response to input from the user or an Application on the On-The-Go B-device. HNP may only be implemented through the Micro-AB receptacle on a device [9, 10].

USB 2.0 Transceiver Macrocell Interface (UTMI)

4.1 Introduction

UTMI stands for USB2.0 Transceiver Macrocell Interface [17]. 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 [6].

4.2 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 are [6]:

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 his needs. The USB2.0 Transceiver can be placed in low power mode with SuspendM signal. The functional block of USB2.0 Transceiver is shown in a figure 4.1below:

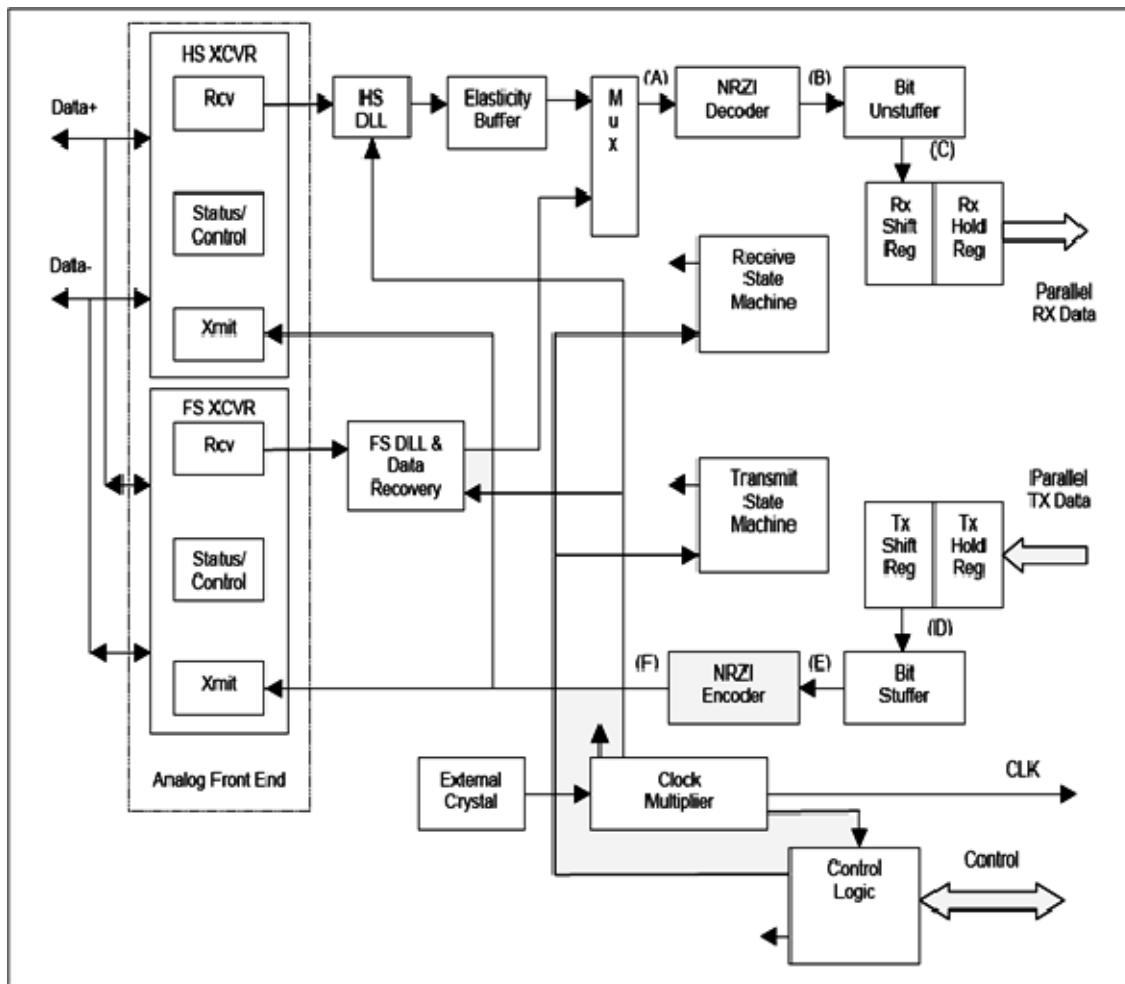


Figure 4.1- USB2.0 Transceiver [6]

4.3 UTMI Difficulty Levels

UTMI core has various levels. Each level has different complexity level as shown in a figure 4.2 below.

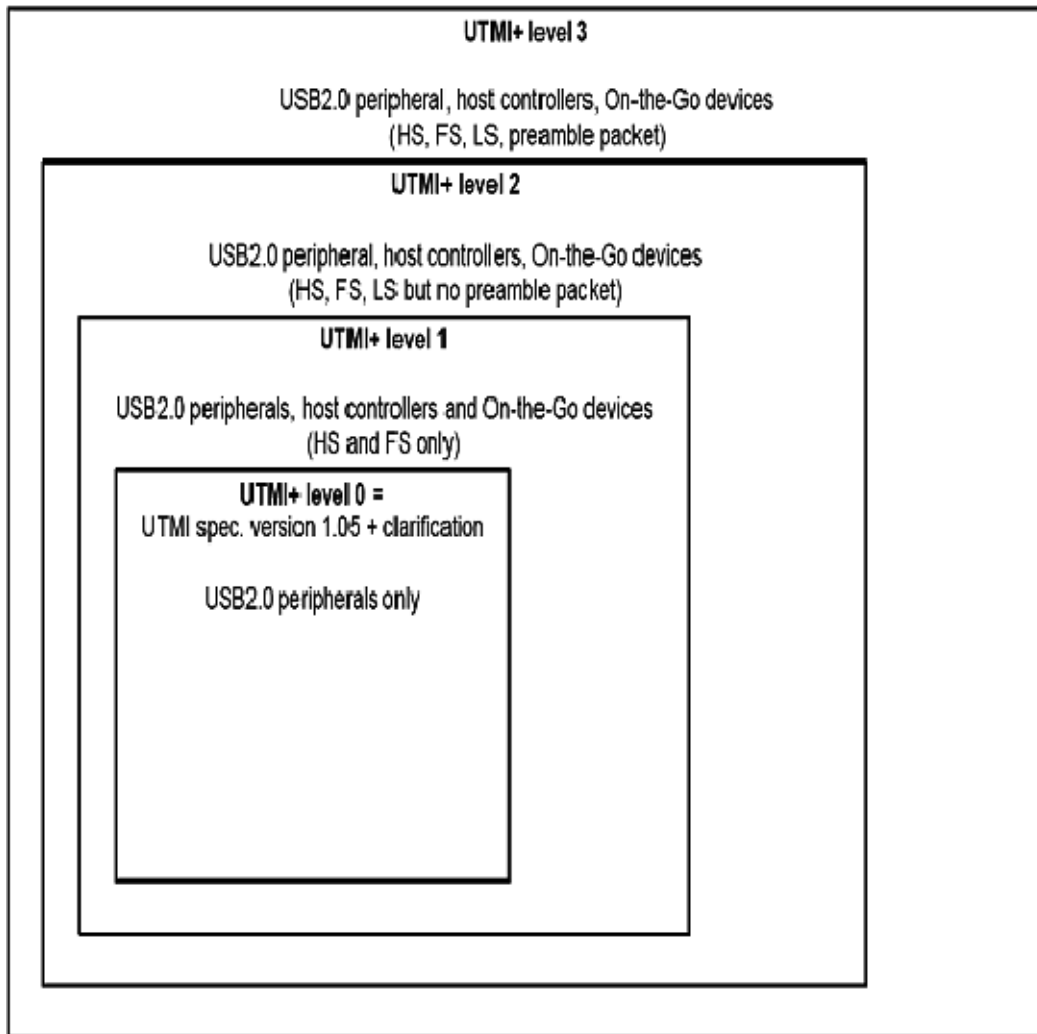


Figure 4.2 - UTMI + Levels [7]

4.3.1 UTMI + Level 0:

UTMI + level 0 transceiver core can be used in USB 2.0 peripheral designs. These cores are not viable for USB 2.0 hosts or On-The-Go peripherals without additional logic. For a

UTMI + Level 0 transceiver with 8 bit interface the TXValidH, RXValidH, Data Bus6_8, DataIn(15:8), Data out(15:8) signals are not used.

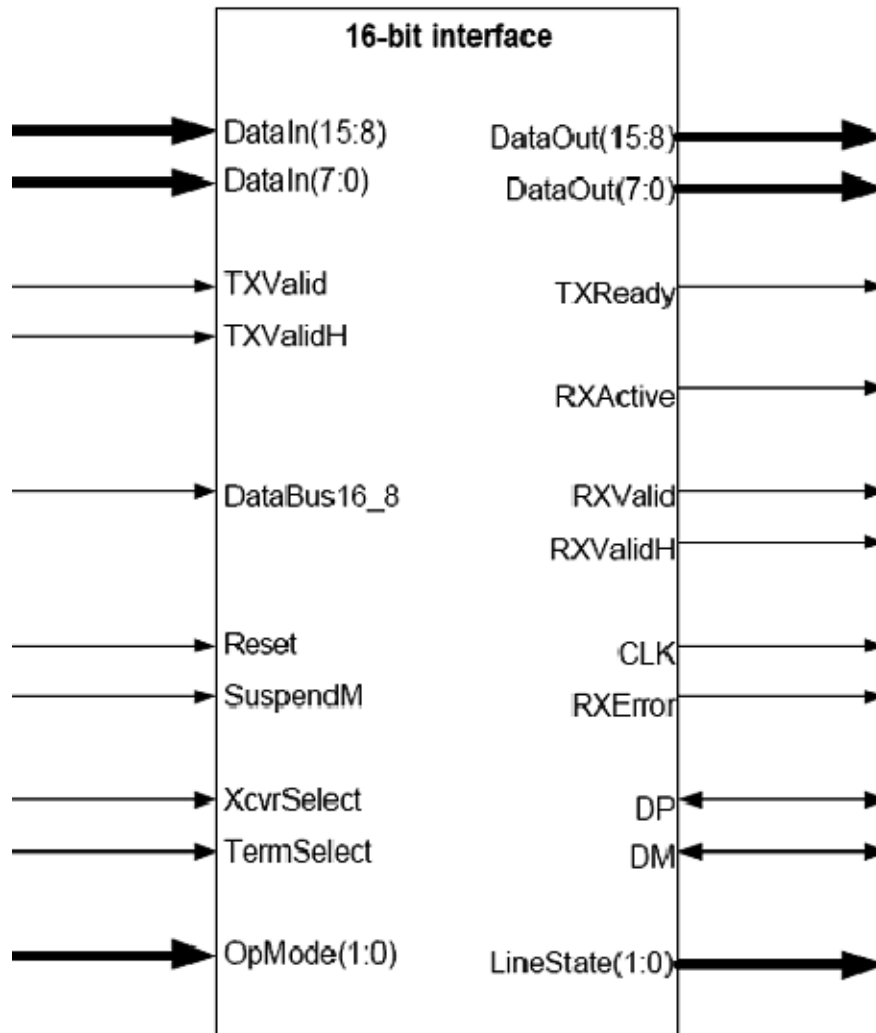


Figure 4.3 - UTMI+ Level 0 Entity Diagram [7]

4.3.2 UTMI + Level 1:

Any transceiver core that has compliance with UTMI + Level 1, has signal compliance with UTMI + Level 0. UTMI + Level 1 support USB 2.0 peripherals, host controllers and On-the-Go devices in HS and FS mode only. It can not implement for LS mode.

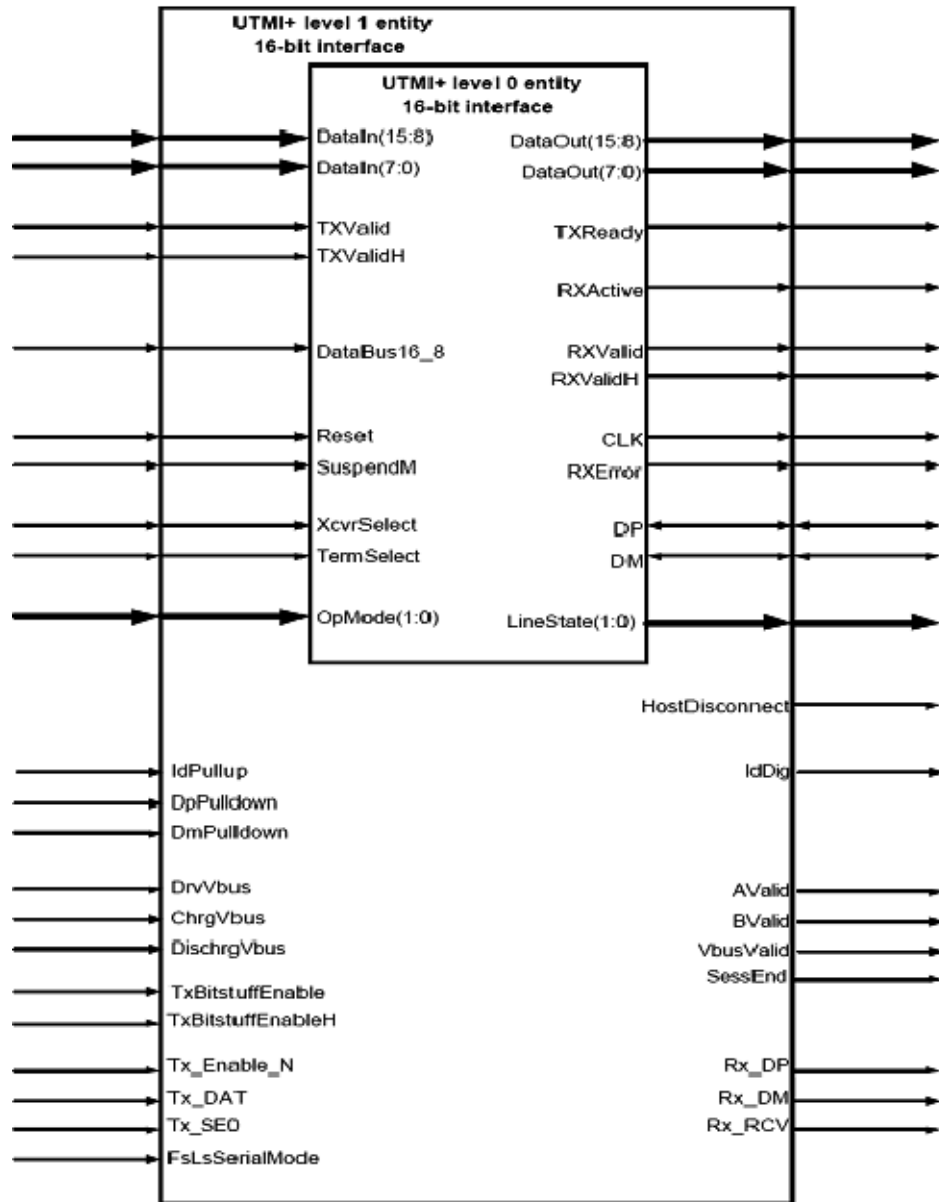


Figure 4.4 - UTMI+ Level 1 Entity Diagram [7]

4.3.3 UTMI+ Level 2:

Host controllers that support UTMI + Level 2 can communicate at LS except a host sending a packet to the LS device that is connected through FS hub because preamble packet is not allowed in UTMI + Level 2. That is covered in next Level. Host must be able to send LS keep alive packet on the LS bus that is equivalent to LS EOP.

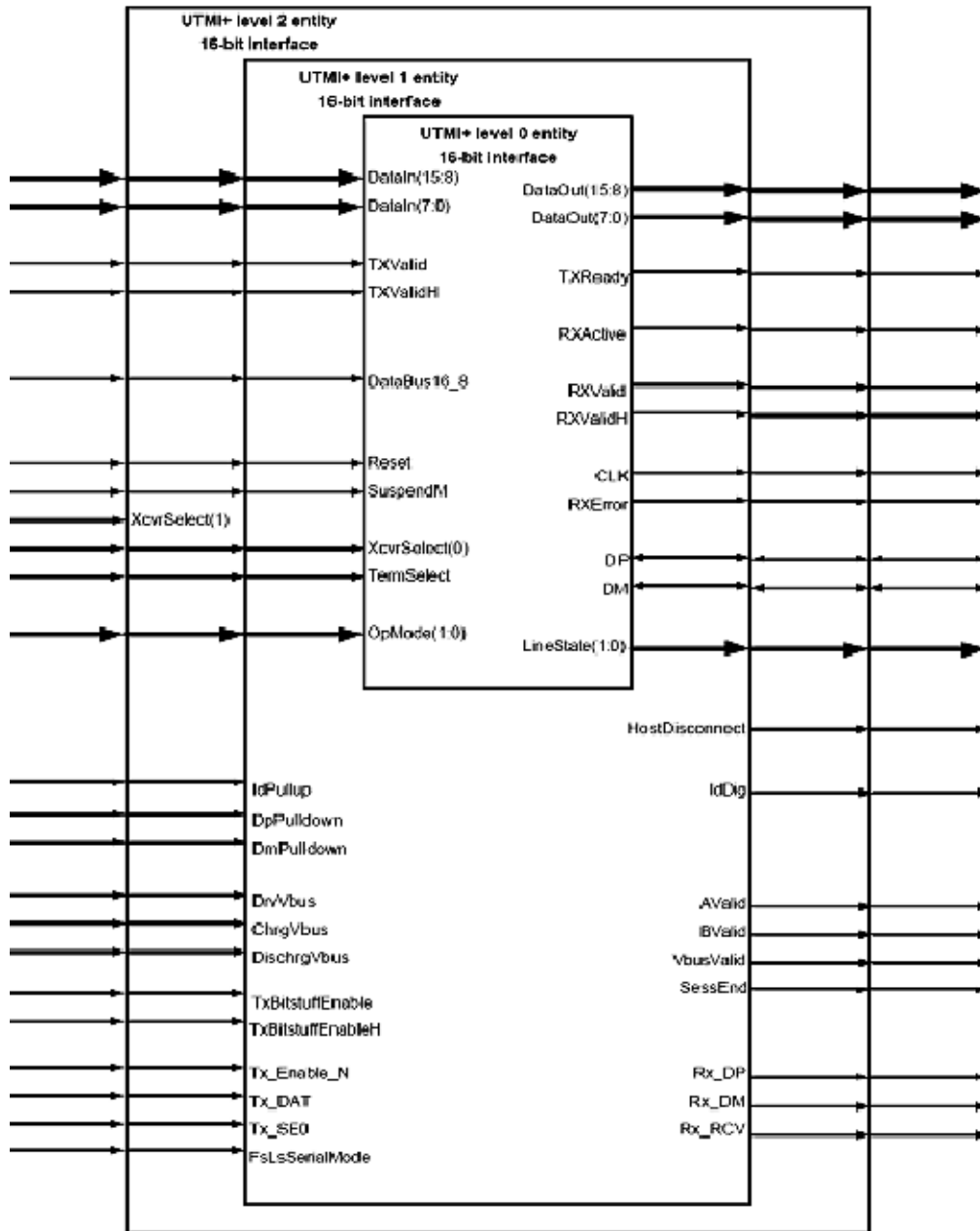


Figure 4.5 - UTMI+ Level 2 Entity Diagram [7]

4.3.4 UTMI+ Level 3:

In UTMI + Level 3 host has a capability to generate preamble packet that makes it feasible for host controller to communicate at LS with the device connected to Full speed hub. Signal interface is just same for UTMI + Level 2.

4.4 UTMI Signal Description

This section gives the basic UTMI core signals.

1. CLK

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

Table 4.1 – Clock Description

Clock Frequency (In MHz)	Description
60	HS/FS, with 8 bit interface
30	HS/FS with 16 bit interface
48	FS only, with 8 bit interface
6	LS only, with 8 bit interface

2. RESET

Reset all state machines in the UTM.

3. 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. If TXValid is asserted, the SIE must always have data available for clocking in to the TX Holding Register on the rising edge of CLK. If TXValid is TRUE and TXReady is asserted at the rising edge of CLK, the UTM will load the data on the DataIn bus into the TX Holding Register on the next rising edge of CLK, at that time, SIE should immediately present the data for next transfer on

the DataIn bus. If TXValid is asserted and TXReady is negated, the SIE must hold the previously asserted data on the DataIn bus. From the time TXValid is negated, TXReady is a don't care for the SIE.

4. XcvrSelect

This signal selects between the FS and HS transceivers.

Table 4.2 – XcvrSelect Description

XcvrSelect	Description
0	HS transceiver enabled
1	FS transceiver enabled

5. TermSelect

This signal selects between the FS and HS terminations.

Table 4.3 – TermSelect Description

TermSelect	Description
0	HS termination enabled
1	FS termination enabled

6. TXValid

This is an active high input signal. Transmit valid Indicates that the DataIn bus is valid. The assertion of Transmit Valid initiates SYNC on the USB. The negation of Transmit Valid initiates EOP on the USB. In HS (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 Transmit State Machine. In FS (XcvrSelect = 1), FS Only, or LS Only modes, the SYNC pattern must be asserted on the USB no less than 1 CLK and no more than 5 10 CLKs³ after the assertion of TXValid is detected by the Transmit State Machine.

7. DataIn0-7

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

When DataBus16_8 = 1 this bus transfers the low byte of 16-bit transmit data.

When DataBus16_8 = 0 all transmit data is transferred over this bus.

8. OpMode(0-1)

These signals select between various operational modes.

Table 4.4 – OpMode Signal Description

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

9. 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.

Table 4.5 – Line State Signals Description

DM	DP	Description
0	0	SE0
0	1	'J' State
1	0	'K' State
1	1	SE1

10. SuspendM

Places the Macrocell in a mode that draws minimal power from supplies. Shut 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.

Table 4.6 – SuspendM Signal Description

SuspendM	Description
0	Macrocell circuitry drawing suspend current
1	Macrocell circuitry drawing normal current

11. DataIn8-15

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.

5.1 Introduction

This section delineates a low pin interface (LPI) between a Link and a PHY. Its Interface signals are defined and the basic communication protocol is described.

5.2 Interface Signals

Its transceiver interface signals are described briefly in Table 5.1. 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 stream can be used to transmit or receive the packets.

Table 5.1 – LPI Generic Interface Signals [3]

Signal	Direction	Description
PHY Interface		
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 <i>dir</i> .
dir	Out	It controls the direction of the data bus. When PHY has data to transfer to the Link it drives <i>dir</i> high to take the ownership of the bus. When PHY has no data to transfer it drives the <i>dir</i> 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.

5.3 Protocol

5.3.1 LPI 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. 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.

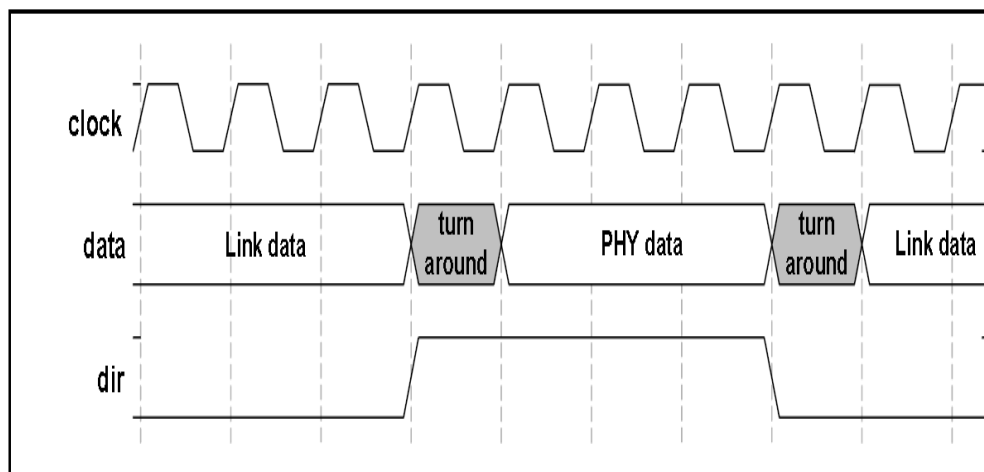


Figure 5.1 – LPI Generic Data Bus Ownership [3]

5.3.2 Transferring Data

During the first half of the figure 5.2, 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.

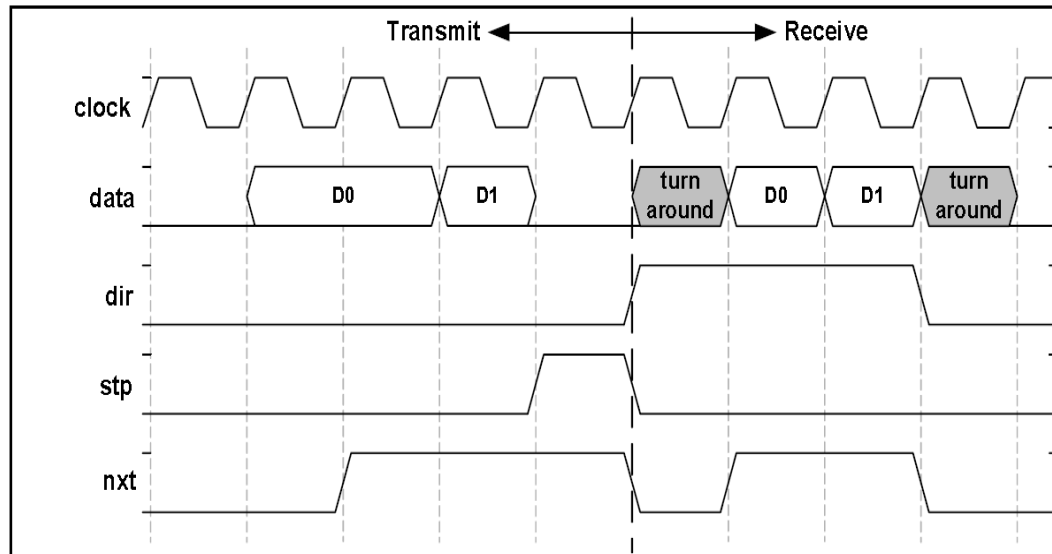


Figure 5.2 - LPI Generic Data Transmit Followed by Data Receive [3]

During the second half of the figure 5.2, 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 signal 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.

5.3.3 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. So PHY unconditionally de-assert dir and start receiving data from Link. PHY may re-assert dir to again take control of the data bus when Link finishes of data transmission.

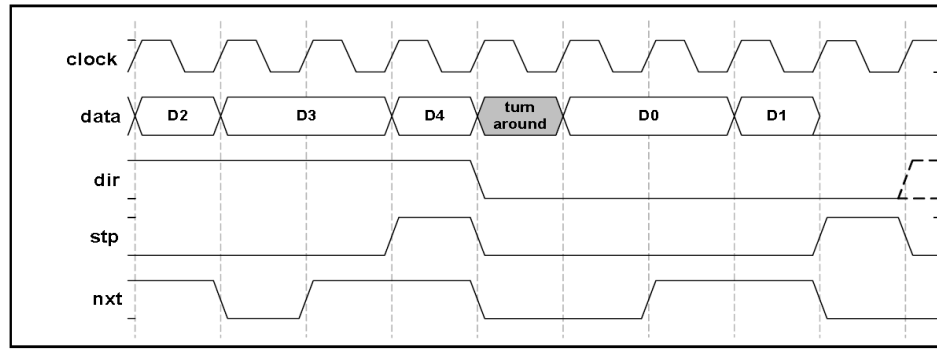


Figure 5.3 – Link Asserts stp to Receive Data [3]

5.4 Block Diagram

The block diagram of ULPI PHY is shown below.

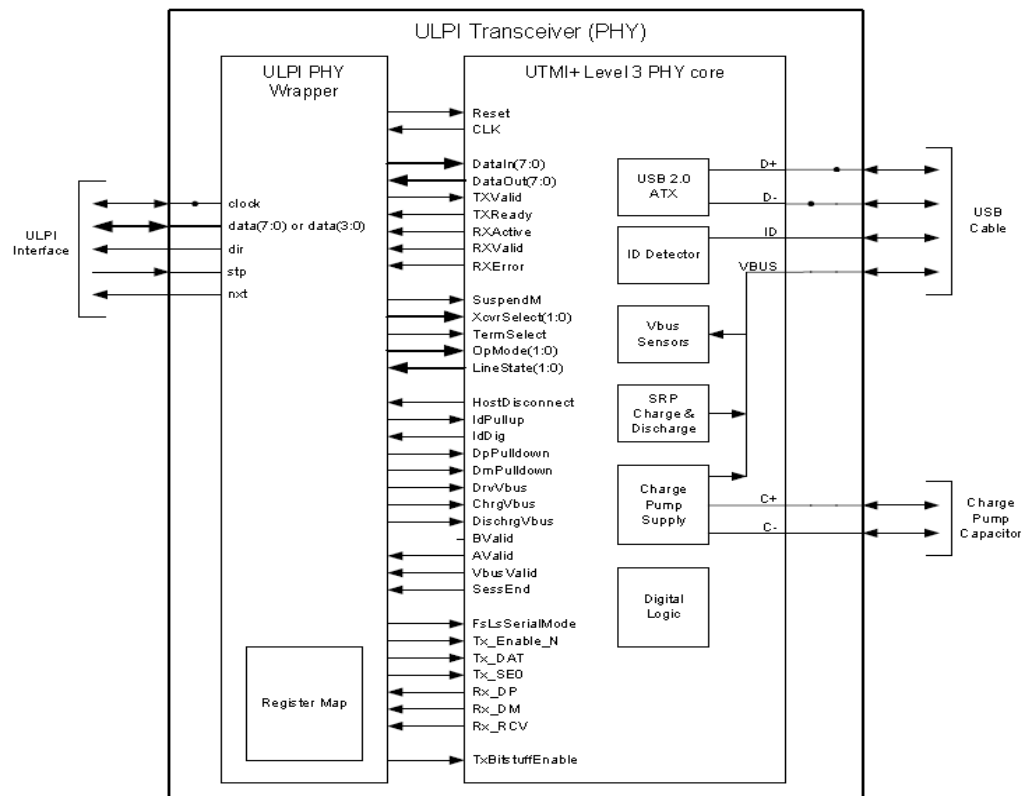


Figure 5.4 – Block Diagram of PHY

ULPI interface is developed from the UTMI interface with advantage of low pin count.

5.5 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 a 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 5.4. A 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.

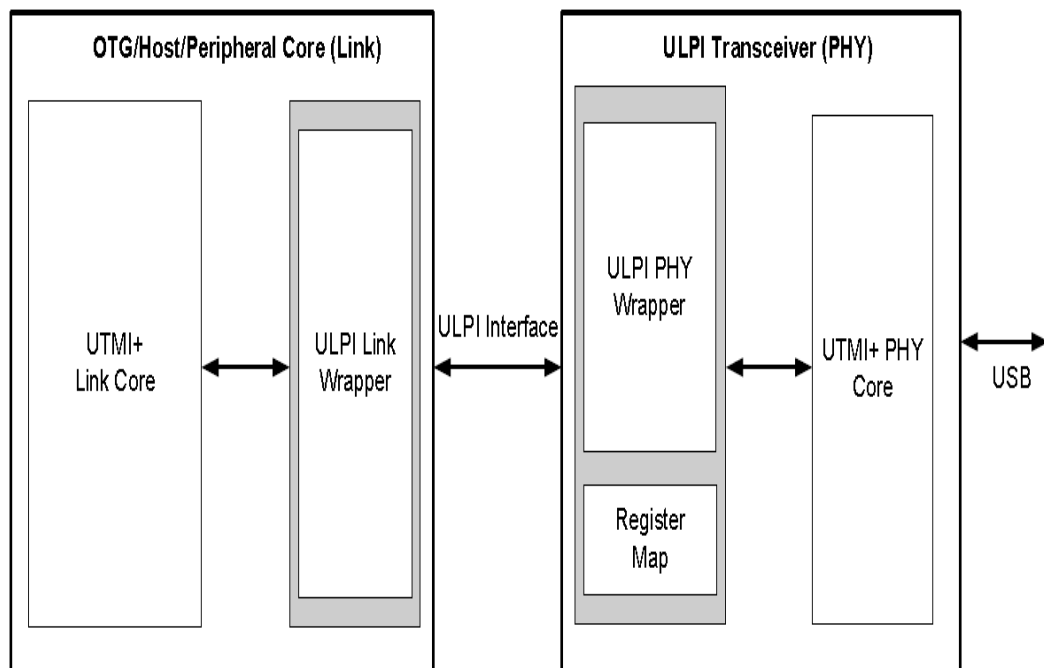


Figure 5.5 – Creating a ULPI System Using Wrappers [3]

5.6 Registers

5.6.1 Register Map

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.

5.6.1.1 Function Control

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.

Table 5.2 – Function Control Register [3]

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

5.6.1.2 Interface Control

It enables the alternative interfaces and PHY features. All bits in this register are optional features of the PHY. It is mainly used to select the interface at the ULPI. FsLsSerialMode Low power mode and CarKit mode are selected by this register. It is explained below:

Table 5.3 – Interface Control Register [3]

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 signalling
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

5.6.1.3 OTG Control Register

OTG Register is used to controls UTMI+ OTG functions of the PHY. Signals defined by this register is explained below

Table 5.4 – OTG Control Register [3]

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-
Dischrgybus	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 otg comparator or internal vbus valid indicator 1b: use external vbus valid indicator.

5.7 Transmit Command Byte (TX CMD)

The link always initiates the data transfer to the PHY by sending the 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 table 5.5 shown below:

Table 5.5 – Transmit Command Byte Format [3]

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

5.8 Receive Command Byte (RX CMD)

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 below:

Table 5.6 – Receive Command Byte Format [3]

Data	Name	Description and value
1:0	Line State	UTMI + Line State signals Data(0) = LineState(0) Data(1) = LineState(1)
3:2	Vbusstate	Encoded vbus voltage state

		Value	VBUS voltage	Sess. End	Sess. Valid	Vbus Valid
		00	VBUS <Vb_SESS_END	1	0	0
		01	Vb_sess_end<vbus<vsess_vld	0	0	0
		10	Vsess_vld<vbus<va_vbus_vld	X	1	0
		11	VA_VBUS_VLD<VBUS	X	X	1
5:4	RxEvent	Encoded UTMI event signals				
		Value	RxActive	RxError	HostDisconnect	
		00	0	0	0	
		01	1	0	0	
		11	1	1	0	
		10	X	X	1	
6	ID	Set to value ID ground				
7	Alt_int	Used for carkit interrupt latch				

6.1 Introduction

The function of un-wrapper is to un-wrap the ULPI interface to convert it to the UTMI core. The purpose of un-wrapping is to facilitate the debugging process [3].

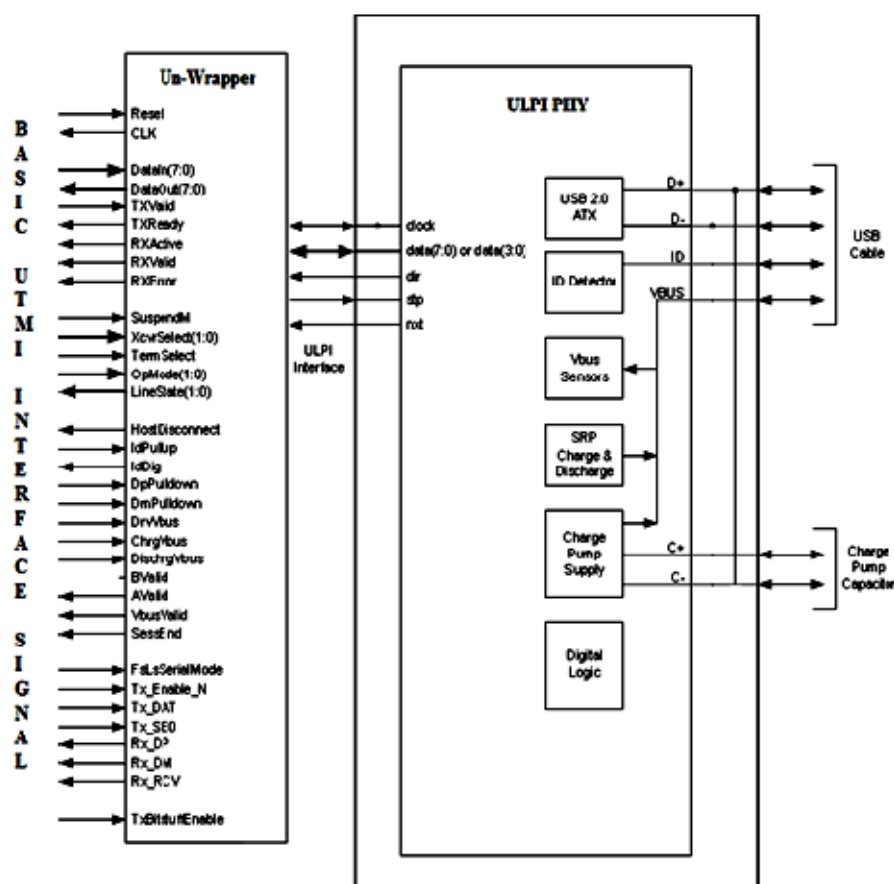


Figure 6.1 - Un-Wrapper from ULPI to UTMI + Core

During interaction with ULPI interface, Link transmits register address at the ULPI interface followed by data to be written in the specified register. This data configures the

various functions of PHY like data rate, enable or disable NRZI encoding and bit stuffing etc. In case Debugger needs to operate PHY in different mode. It is required to write different data in the same register each time mode is changed. For this, Debugger needs to alter the code and it leads to the repetition of various steps which includes synthesis, translate, mapping etc. Also again and again bit file need to be burnt in FPGA.

Un-wrapper replaces the registers with input pins. If Link needs to operate the PHY in different modes then there is no need to burn the code again and again, rather change the value at the input pins will reflect the same behavior as desired.

Figure 6.2 shows the basic set up of transmitter circuit using un-wrapper. Link transmits the parallel data to ULPI PHY at 60 MHz and PHY converts this parallel bit stream to serial bit stream at 480 MHz in HS mode and 12Mbps in case of FS mode. This serial bit stream can be precisely visualized at the digital timing Oscilloscope.

6.2 Basic Set up Utilizing Un-Wrapper

The basic set up incorporating un-wrapper to exercise various ULPI PHY modes include

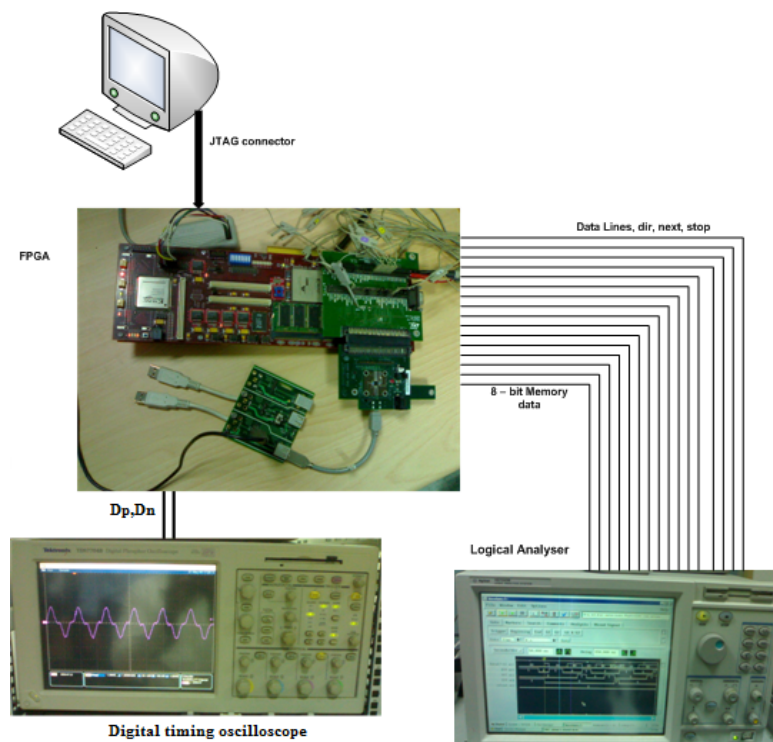


Figure 6.2 - Basic Set up Utilizing Un-Wrapper [18]

Avnet Vertex 2 development board [19], ULPI PHY [20], T&MT adapter card [21], squid board, logic analyzer and digital oscilloscope capable of capturing at least 480 Mbps data rate. JTAG connector is utilized to burn FPGA through Xilinx ISE 6.0. It also includes basic accessories like jumpers, probes etc. as shown in a figure 6.2.

6.3 Observations

Following are the observations made which gives stamp to approval of the correctness of experiment.

6.3.1 High Speed Mode

1. To configure ULPI PHY in a high speed data transmission rate, `xcvrsel` pins on the un-wrapper interface must be de-asserted as delineated in a figure 6.3.
2. To enable NRZI encoding [22] and bit stuffing, `opmode` pins on the un-wrapper interface must be 00 for normal operation as indicated in a figure 6.3 and to disable NRZI encoding these pins should be 10.
3. In high speed mode, peak to peak voltage observed at the differential port of Digital oscilloscope should be 400mv which assures the high speed mode.
4. In HS mode, if Link continuously transmits data to ULPI PHY, `nxt` will remain asserted.
5. In HS mode, if Link continuously transmits FF h at ULPI interface. `nxt` goes low after every 7th cycle if bit stuffing is enabled by de-asserting `opmode` pins of un-wrapper.
6. If `opmode` pins at the un-wrapper interface are made 10. It will disable the bit stuffing as well as NRZI encoding. So whatever data is applied at the ULPI interface will directly appear at the `dp,dn` lines of ULPI PHY.
7. Data 00h must be transmitted from Link to ULPI interface after enabling NRZI encoding to make `Dp, Dn` lines toggle continuously for frequency measurement. `Dp, Dn` lines toggle at frequency of 480 MHz in HS mode as shown in a figure 6.4.

Figure 6.3 indicates that value 60 is to be written in a function register which is equivale-

nt to writing 00 at the xcvsel and 00 at the opmode pins of un-wrapper. These values configure PHY in HS mode and enable both NRZI encoding and bit stuffing.

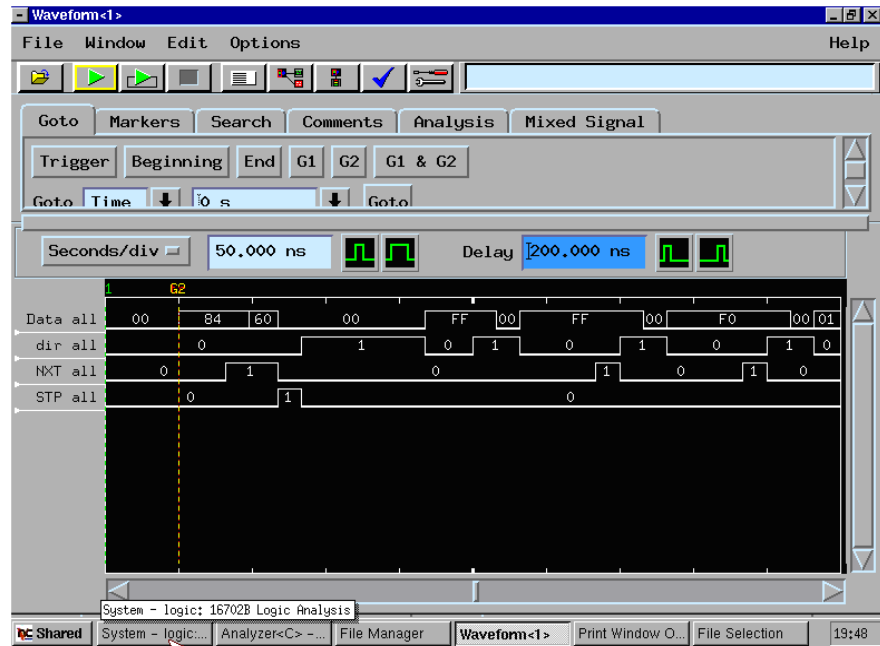


Figure 6.3 – High-Speed Data Transmission (ULPI Interface)

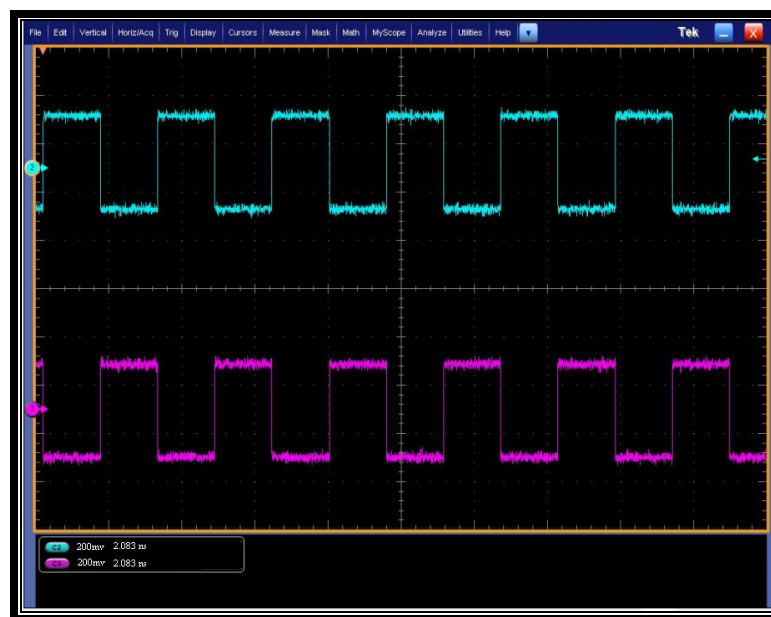


Figure 6.4 – HS Data Transmission (Dp, Dn)

6.3.2 FS-Speed Mode

1. To configure ULPI PHY in a full speed data transmission rate, xcvsel pins on the un-wrapper interface must be 01 as delineated in a figure 6.5.
2. To enable NRZI encoding and bit stuffing, opmode pins on the un-wrapper interface must be 00 for normal operation as indicated in a figure 6.5 and to disable NRZI encoding these pins should be 10.
3. In full speed mode, peak to peak voltage observed at the digital oscilloscope should be 3v which gives green light to FS mode.
4. In FS mode, if Link continuously transmits data to ULPI PHY, nxt will be asserted after every 40th cycle as shown in a figure 6.5.
5. If opmode pins at the un-wrapper interface are made 10. It will disable the bit stuffing as well as NRZI encoding. So whatever data is applied at the ULPI interface will directly appear at the dp,dn lines of ULPI PHY.
6. Data 00h must be transmitted from Link to ULPI interface after enabling NRZI encoding to make Dp, Dn lines toggle continuously for frequency measurement. Dp, Dn lines toggle at frequency of 12 MHz in FS mode as shown in a figure 6.6.

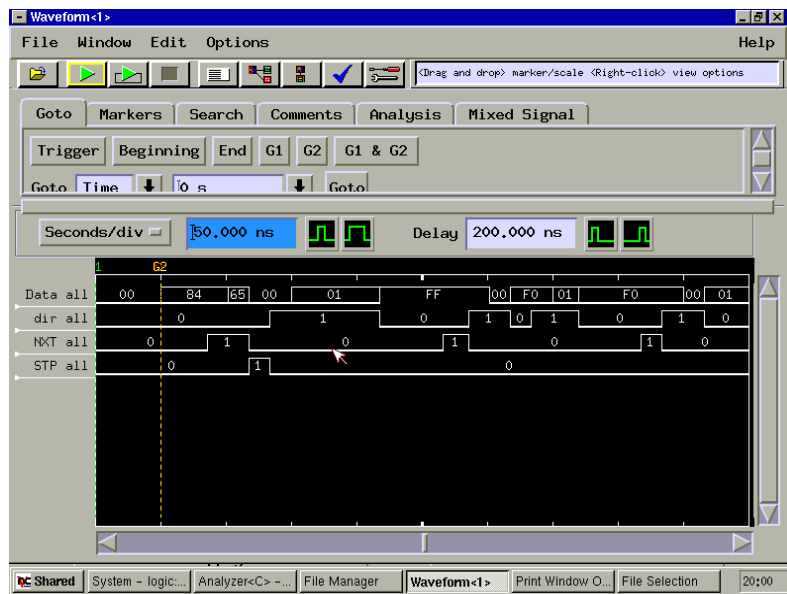


Figure 6.5 – FS Data Transmission (ULPI Interface)

Figure 6.5 indicates that value 65 is to be written in function register which is equivalent to writing 01 at the xcvsel and 00 at the opmode pins of un-wrapper. These values configure PHY in FS mode and enable both NRZI encoding and bit stuffing.

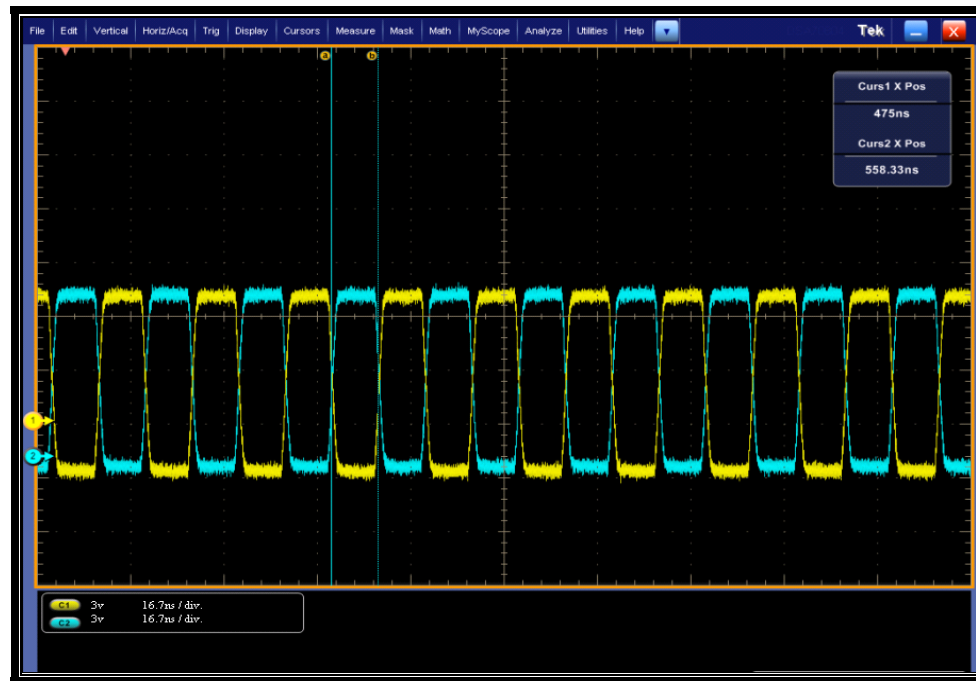


Figure 6.6 – Full-Speed Data Transmission (Dp, Dn)

FPGA Based Implementation of Test & Debug Environment for High Speed Serial Links

7.1 Introduction

Testing is the practice of making objective judgments regarding the extent to which the device meets, exceeds or fails to meet stated objectives.

The fundamental purpose of testing is to verify procurement specifications what was specified is what was delivered: it verifies that the device meets the functional, performance, design, and implementation requirements identified in the procurement specifications.

In a nut shell, testing demonstrate the presence of a bug in the design. After testing, there comes a term debugging means to identify what caused the bug.

The purpose of my thesis is functional testing and validation of the ULPI PHY using FPGA based platform [23]. The FPGA implemented test and debug environment facilitates the functional testing and debugging of ULPI PHY.

7.2 FPGA Based Test and Debug Environment

The basic set up for the FPGA based test and debug environment is shown in a figure 7.1. It includes Avnet Vertex 2 develop board, ULPI PHY, T&MT adapter card, Squid card, two data generator, Labview based setup in the PC, digital oscilloscope and logic analyser or chip scope[24].

JTAG connector is used to burn the verilog code [25, 26] into the FPGA board. Pods are attached with the data generator to drive parallel data bus. Nonetheless, accessories like jumpers, probes, SMA connectors etc are used in this platform.

One of the data generators [27] should support at least 480 MHz and other must support GPIB interface. Digital oscilloscope must be capable of capturing at least data rate of 480

MHz.

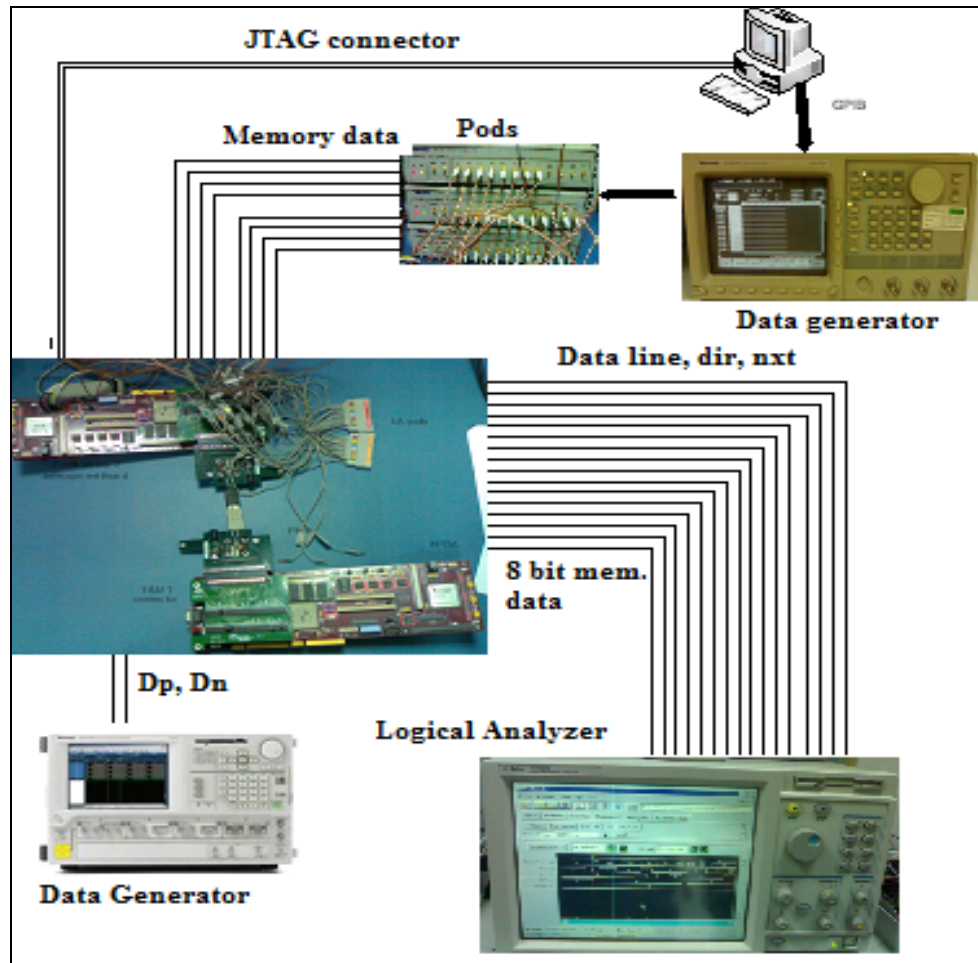


Figure 7.1 - FPGA Based Test and Debug Environment

7.3 Functional Validation

The following functional tests have been successfully tested on this platform [28].

- ❖ Register write and read operation
- ❖ FS, HS and LS data transmission
- ❖ FsLsSerialMode
- ❖ Low Power mode
- ❖ BER

7.4 Register Write and Read Operations

As discussed in chapter 4, there are three most important registers function control, interface control, OTG control inside a ULPI PHY. PHY supports various modes like LS, FS, HS, FsLsSerial and low power, Car kit mode [29] etc. In case of USB 2.0, PHY also supports OTG capabilities.

Before starting any communication with PHY, it becomes very important to configure PHY by writing the appropriate value in the appropriate register. Otherwise it may start working in a default mode because every ULPI register has got some default or reset value.

7.5 Register Write

To write any register in ULPI PHY the following steps are required:

1. Firstly TX CMD byte for register write is sent to the ULPI PHY with the required register address and waits for *nxt* to assert.

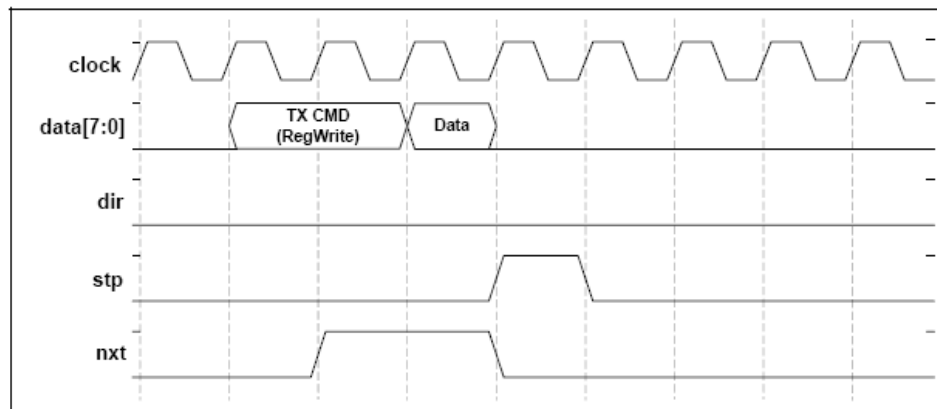


Figure 7.2 - Register Write

2. After the *nxt* is asserted Link sends register write data and again waits for *nxt* to assert.
3. When the *nxt* is asserted Link sends *stp* to complete the operation.

4. The PHY must detect the stp assertion before it can accept another transmit command.
5. If the PHY aborts the register write operation by asserting dir, then Link should retry the Register write operation when the bus is idle.

7.6 Register Read

To read any ULPI register following steps are required:

1. Firstly TX CMD byte for register read is sent to the ULPI PHY with the required register address.
2. As shown in a figure 7.3, Link sends a register read command and waits for nxt to assert.
3. After the nxt is asserted, PHY asserts dir to gain control of the data bus.
4. In the cycle after dir assert, PHY must return the register read data.
5. PHY does not assert nxt when dir is asserted during register read operation.
6. If the PHY aborts the register write operation by asserting dir earlier than shown in figure 7.3, then Link should retry the Register read operation when the bus is idle.

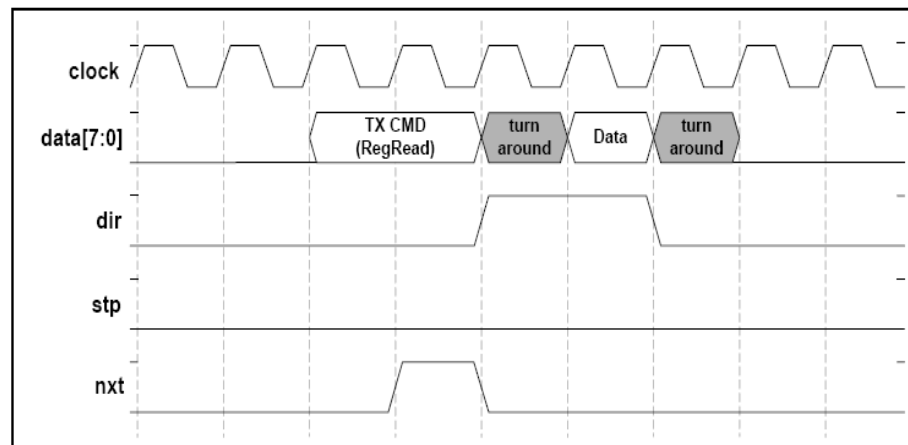


Figure 7.3 - Register Read

7.7 Simulation Results for Register Write and Read

Figure 7.4 shows the function control register write and read on NCSIM (Cadence).

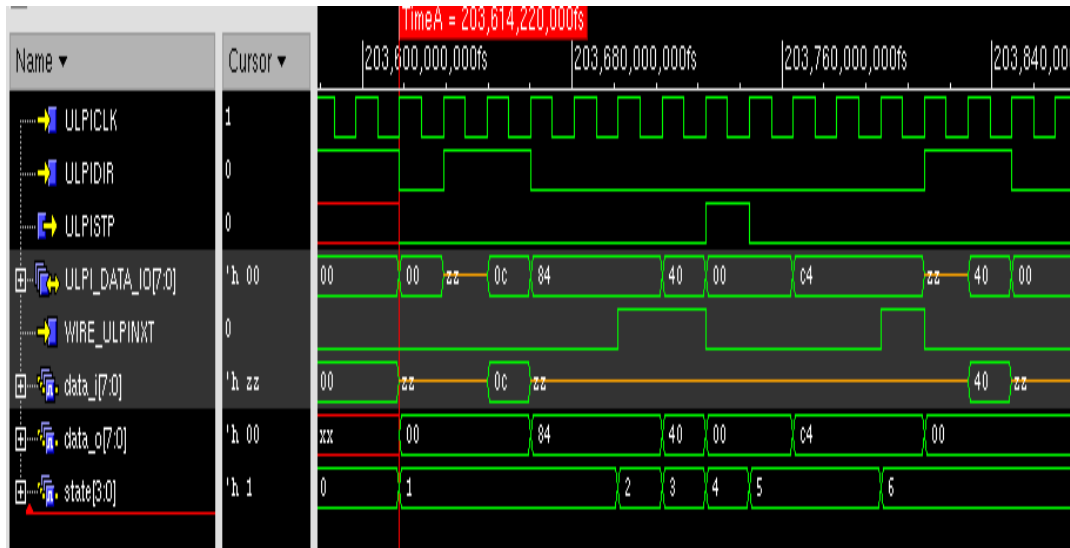


Figure 7.4 - Waveform for Function Control Register Write and Read Together

Figure 7.5 shows the interface control register write and read on NCSIM (Cadence).

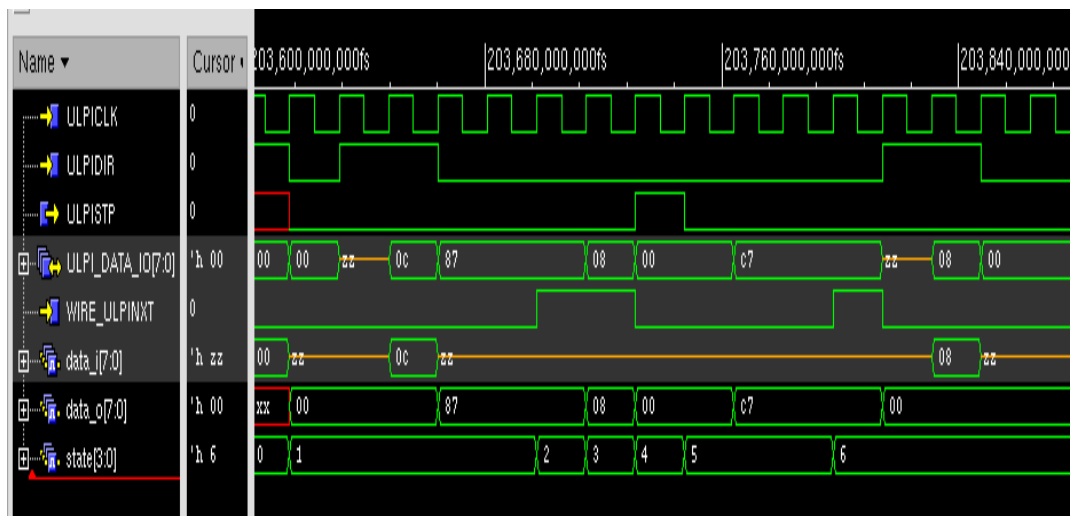


Figure 7.5 - Waveform for Interface Control Register Write and Read Together

Figure 7.6 shows the OTG control register write and read on NCSIM (Cadence).

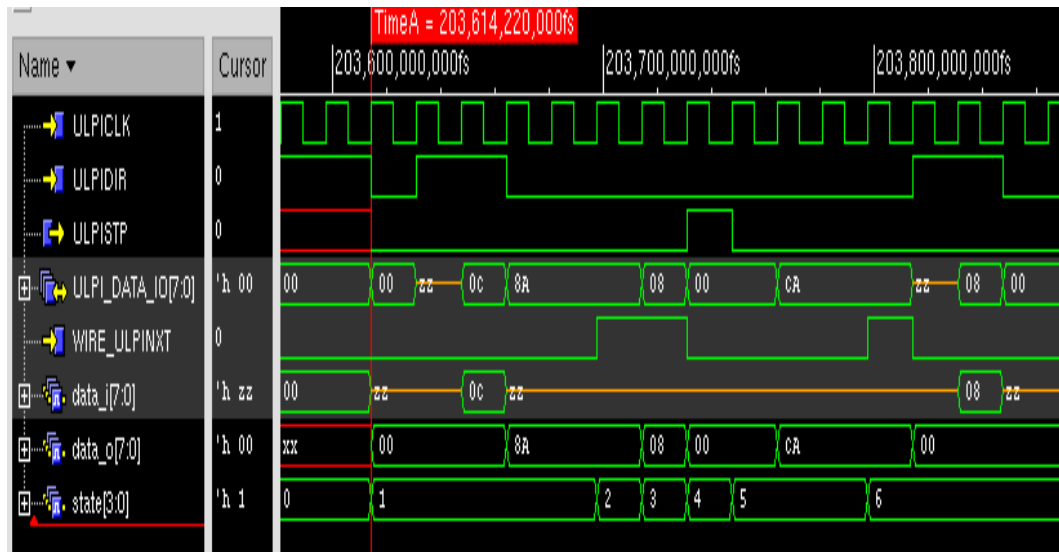


Figure 7.6 - Waveform for OTG Control Register Write and Read Together

7.8 FS/HS/LS Data Transmission

USB 2.0 specifications define three types of data transmission rates, those are Low Speed (1.5MHz) only (LS), full Speed (12MHz) only (FS) and High Speed (480MHz)/full speed (12MHz) (HS). So it is required to specify the data transmission rate. ULPI PHY may opt HS, FS or LS mode depending upon the function register configuration.

7.8.1 HS Data Transmission

USB 2.0 supports high speed data transmission. If high speed capable device is attached to the USB 1.x, it will be treated as full speed device only. High speed data transmission rate is 480 MHz.

xcversel bits of function register are made 00 as indicated in a figure 7.7. In high speed data transmission, ULPI PHY attaches synchronization sequence at the start of packet and EOP at the end of the packet.

Figure 7.7 shows PHY configuration in HS mode with NRZI and bit stuffing enabled.

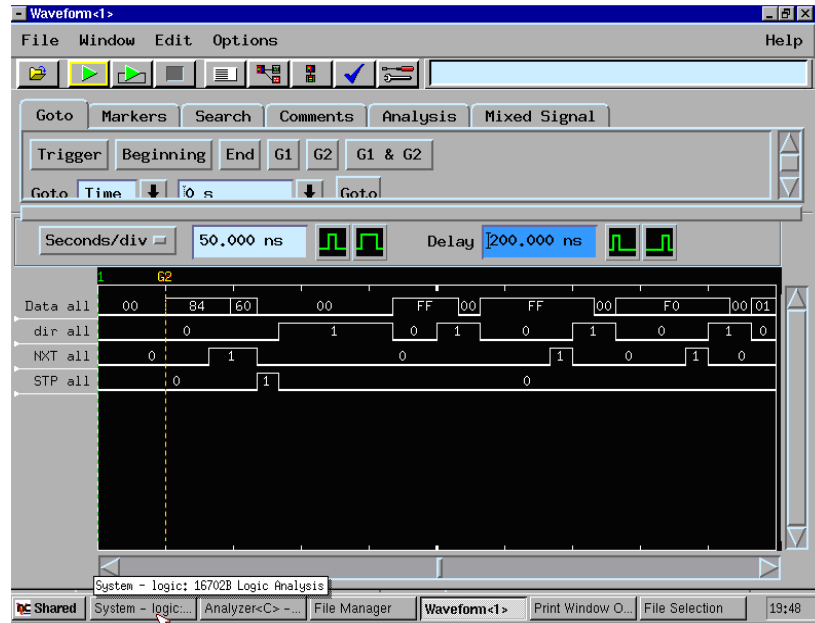


Figure 7.7 – ULPI PHY Configuration in HS Mode

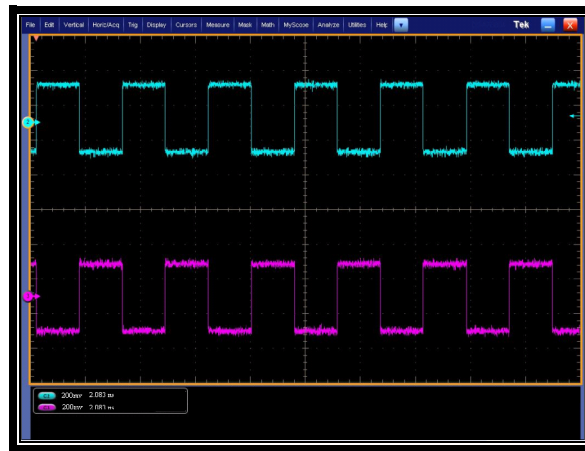


Figure 7.8 – HS Data Transmission

7.8.2 FS Data Transmission

USB 1.x and 2.0 protocols both supports full speed data transmission. Full speed data transmission rate is 12 MHz xcversel bits of function register are made 01. In full speed data transmission, ULPI PHY attaches synchronization sequence at the start of packet and

EOP at the end of the packet.

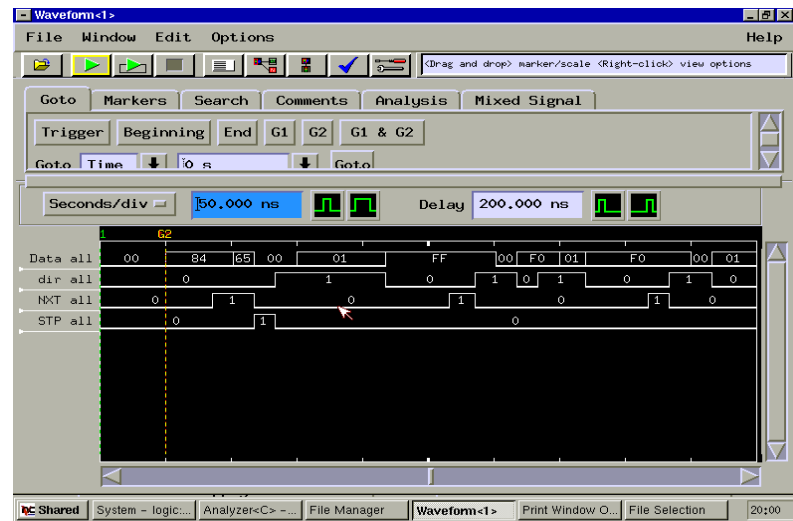


Figure 7.9 - FS Data Transmission

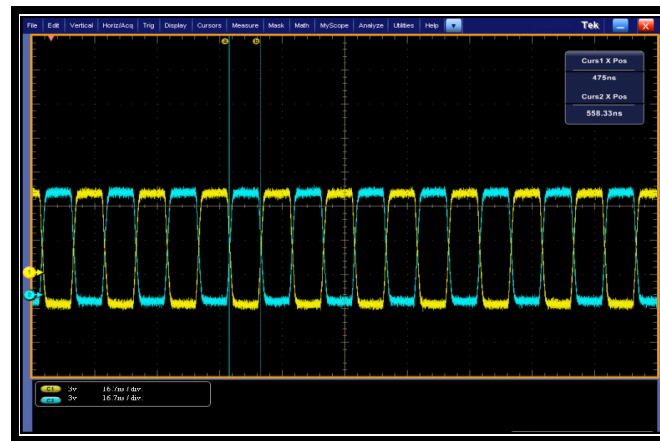


Figure 7.10 - FS Data Transmission

7.8.3 LS Data Transmission

USB 1.x and 2.0 protocols both support low speed data transmission. Low speed data transmission rate is 1.5 MHz. xcvrsel bits of function register are made 10. In low speed

data transmission, ULPI PHY attaches synchronization sequence at the start of packet and EOP at the end of the packet.

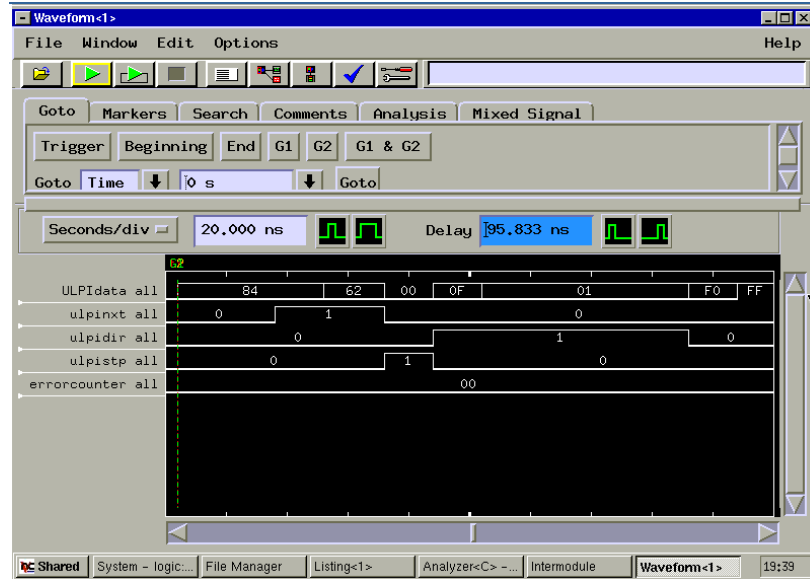


Figure 7.11 - LS Data Transmission

7.9 Data Packet Transmission

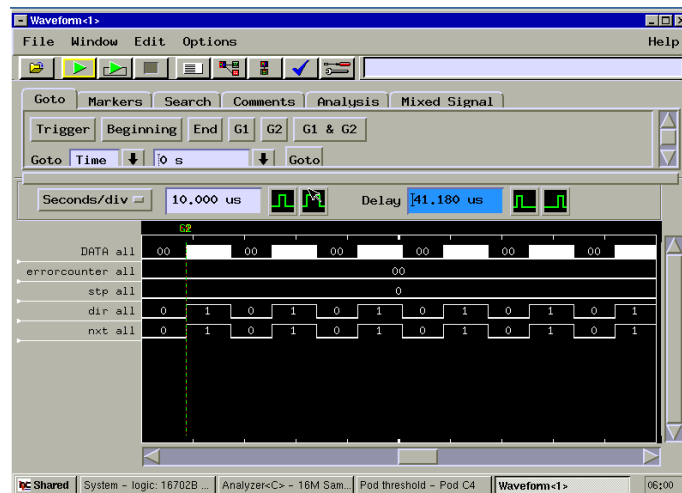


Figure 7.12 - Packet Transmission

Figure 7.12 shows the data packet transmission from Link to PHY.

7.10 FS/LS Speed Serial Mode

Full Speed / Low speed Serial Mode (FsLsSerialMode) gives the Link direct access to the FS/LS serial analog transmitter and receiver. Two types of Serial mode are defined in ULPI: 3-pin FsLsSerialMode and 6-pin FsLsSerialMode.

7.10.1 Data Line Definition for FsLsSerialMode

The 6-pin serial mode provides a separate data line for each serial signal defined in UTMI+. The 3-pin serial mode compasses the USB receive and transmit data onto bi-directional data lines and is provided so that 4-bit data bus PHY implementations can support serial signalling.

In both serial modes, an interrupt pin is provided. The int pin is asserted whenever an unmasked interrupt event occurs.

Table 7.1 – Serial Mode Signal Mapping for 6-Pin FsLsSerialMode

Signal	Maps to	Direction	Description
tx_enable	Data(0)	IN	Active high transmit enable.
tx_dat	Data(1)	IN	Transmit differential data on D+/D-
tx_sel	Data(2)	IN	Transmit single-ended zero on D+/D-.
Int	Data(3)	OUT	Active high interrupt indication. Must be asserted whenever any unmasked interrupt occur.
rx_dp	Data(4)	OUT	Single ended receive data from D+
rx_dm	Data(5)	OUT	Single ended receive data from D-
rx_rcv	Data(6)	OUT	Differential receive data from D+
reserved	Data(7)	OUT	Reserved. The PHY must drive this pin to low.

Table 7.2 – Serial Mode Signal Mapping for 3-Pin FsLsSerialMode

Signal	Maps to	Direction	Description
tx_enable	Data(0)	IN	Active high transmit enable.
dat	Data(1)	I/O	Transmit differential data on D+/D- when tx_enable is high. Receive differential data from D+/D- when tx_enable is low.
Se0	Data(2)	I/O	Transmit single-ended zero on D+/D- when tx_enable is high. Receive single-ended zero from data from D+/D- when tx_enable is low.
Int	Data(3)	I/O	Active high interrupt indication. Must be asserted whenever any unmasked interrupt occur.

7.10.2 Steps to Implement FsLsSerialMode Transmission

1. Firstly TX CMD byte for interface register write is sent to the ULPI PHY with the required register address (1000_0111) and waits for nxt to assert.
2. Link sends a register write command (0000_0001) for FsLsSerialMode and waits for nxt to assert.

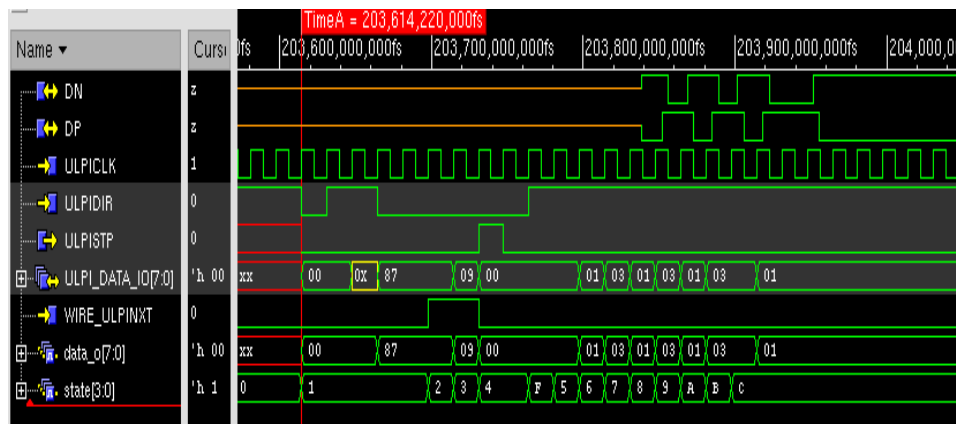


Figure 7.13 – Data Transmission in FsLsSerialMode (NCSIM)

3. When the `nxt` is asserted Link sends `stp` to complete register write operation.
4. Link de-asserts `stp` and wait for `dir` to assert because in `FsLsSerialMode` `dir` is always asserted.
5. When `dir` is asserted Link starts sending data (01,03 etc. having `tx_enable` 1) as shown in a figure 7.14
6. After few cycles' data starts coming at the `Dp` and `Dn`. Where `Dp` is data (1) and `Dn` is inverse of data(1).

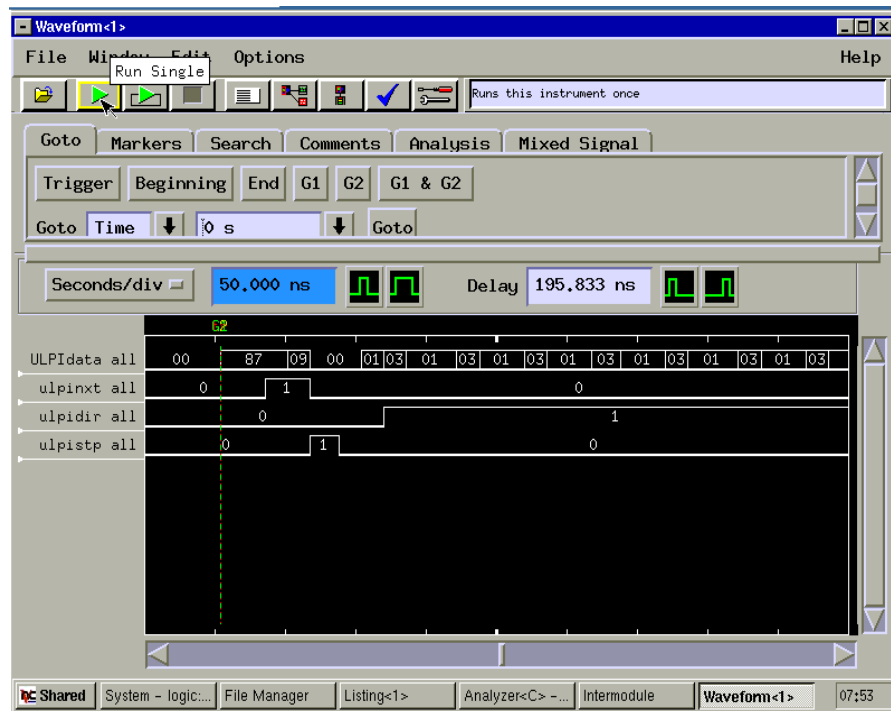


Figure 7.14 – Data Transmission in FsLsSerialMode (Logic Analyzer)

7.10.3 Steps to Implement FsLsSerialMode Reception

1. TX CMD byte for interface register write is sent to the ULPI PHY with the required register address (1000_0111) and waits for `nxt` to assert.
2. Link sends a register write command (0000_0001) for `FsLsSerialMode` and waits

for nxt to assert.

3. When the nxt is asserted Link sends stp to complete register write operation.
4. Link de-asserts stp and wait for dir to assert because in FsLsSerialMode dir is always asserted.
5. When dir is asserted Link starts receiving data (tx_enable 0).
6. Where Dp is data(1) and Dn is inverse of data(1).

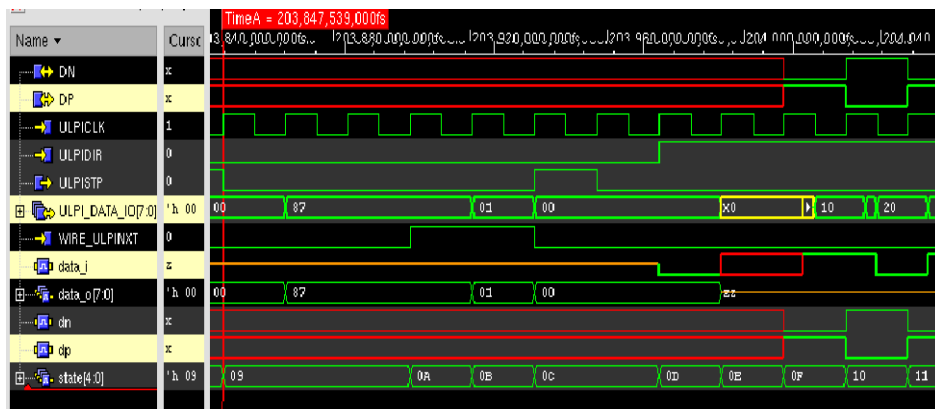


Figure 7.15 – Data Receive in FsLsSerialMode (NCSIM)

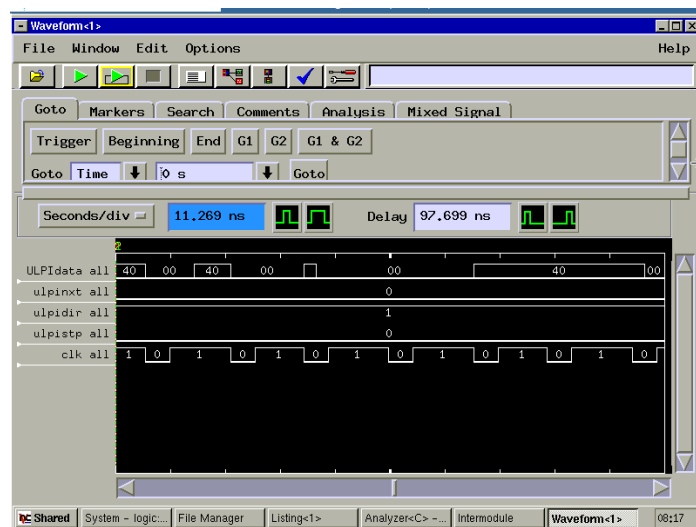


Figure 7.16 – Data Receive in FsLsSerialMode (Logic Analyzer)

7.11 Low Power Mode

The Link can optionally place the PHY into Low Power Mode when the USB bus is suspended. The PHY can power down all circuitry except the interface pins and full speed receiver. The bus resistors must also be powered if VBUS is present.

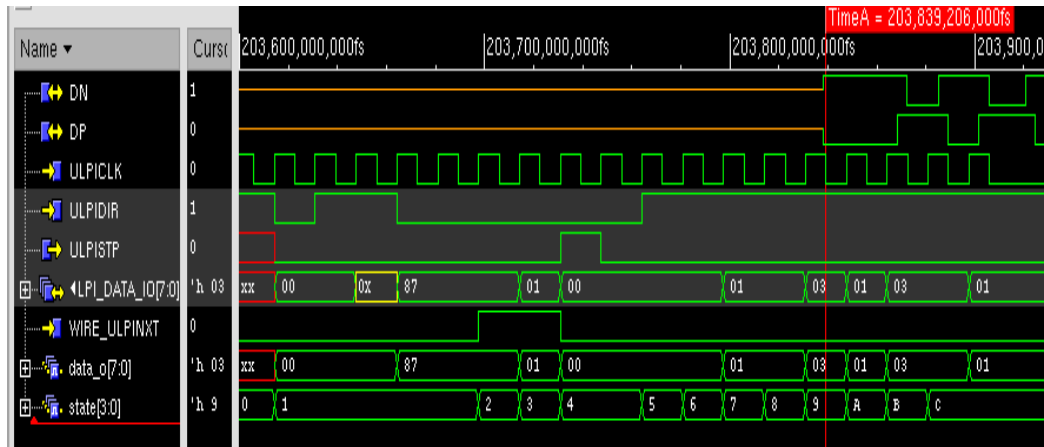


Figure 7.17 - Clock Suspended in Low Power Mode (NCSIM)

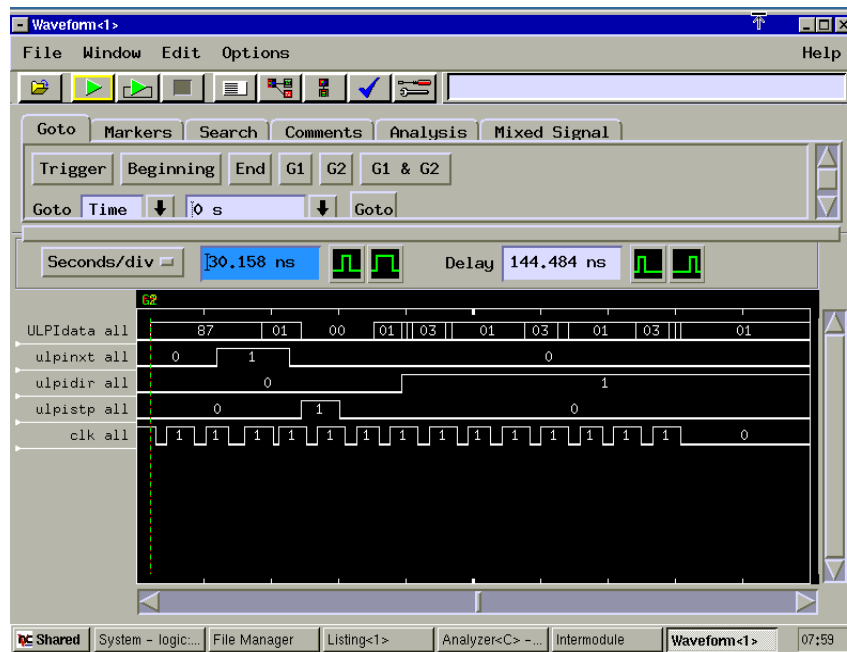


Figure 7.18 - Clock Suspended in Low Power Mode (Logic Analyzer)

Any function must be powered if its corresponding register bit is set, including interrupt sources and the charge pump. If the PLL is powered down, the clock must be stopped without glitches.

As shown in a figure 7.17 and figure 7.18 after the clock got down, data at ULPI_DATA_IO[7:0] cannot reach Dp and Dn because Link asserts the clock suspend pin of interface control register.

7.12 Bite Error Rate (BER)

Bite error rate (BER) [30, 31] ensures the end to end performance and quantifies the reliability of the entire system from “bits in” to “bits out”. In other words, it is a rate at which the errors may occur in a transmission system. BER is a simple concept and can be defined as

$$\text{BER} = \text{Errors} / \text{Total Number of Bits}$$

In USB data transmission system, Bit errors can be caused due to imperfections in the design and random events. Bit errors can also be caused due to intrinsic or extrinsic factors. Intrinsic factors cause errors due to the design, components and implementation of a Link. It includes internal noise sources (e.g. thermal noise), poor electrical conditions and receiver sampling errors. Extrinsic error sources include power supply noise/ripple; electrostatic discharge; electromagnetic interference; cable connector vibrations.

Jitter is a another dominant factor in affecting the BER of a USB data communication system, if all the intrinsic and extrinsic sources are controlled. It may alter the sampling of the data and cause timing mismatch.

7.12.1 Measurement of BER

FPGA based test and debug platform may be used to quantify the bit error rate. Procedure to calculate the bit error rate in ULPI PHY is as follows:

7.12.1.1 PHY Configuration

ULPI PHY is configured to get the desired data transmission rate and to enable or disable NRZI encoding and bit stuffing. USB 2.0 communication protocol supports three types of

data transmission rates LS (1.5 Mbps), FS (12 Mbps), HS (480 Mbps). PHY can also be configured in low power mode.

Function register of ULPI PHY gets configured, soon as the verilog code gets burn in FPGA, to the data rate specified in the verilog code. But data transmission rate can be changed on the fly with the help of un-wrapper pins present on the T&MT connector. Figure 7.19 shows the PHY configuration in HS (480 MHz) mode with enabled NRZI encoding.

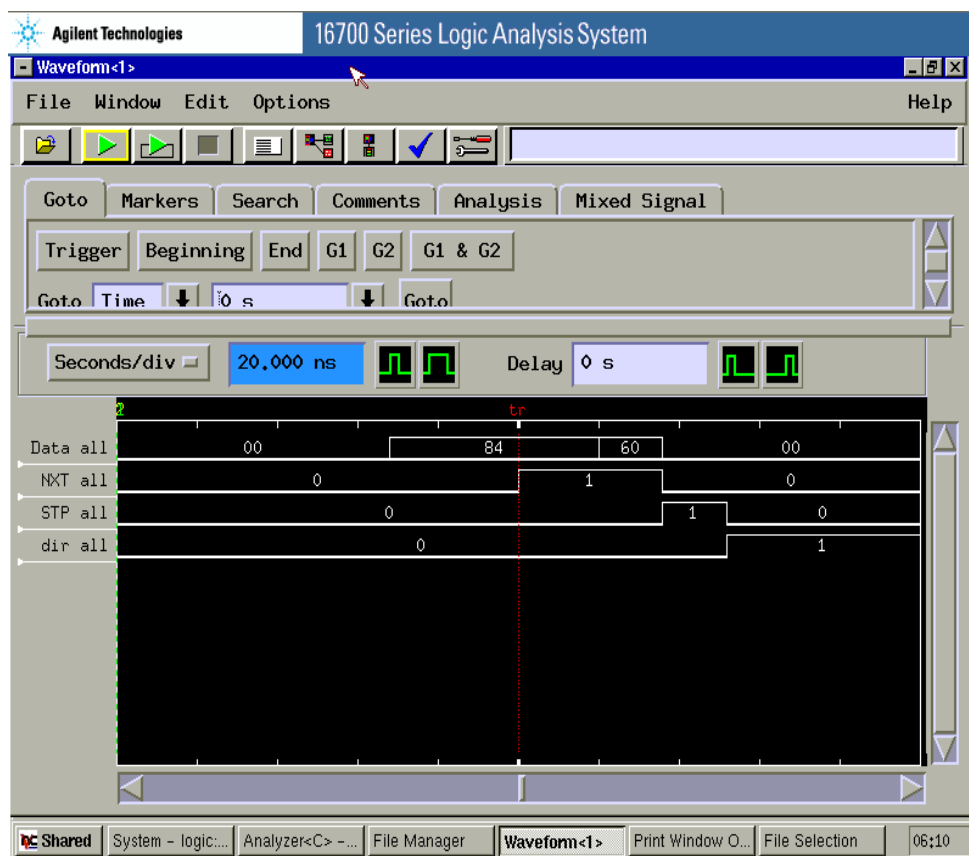


Figure 7.19 – Function Control register Configuration for HS mode

7.12.1.2 Memory Write

BRAM of FPGA is filled with the data to be compared with assist of data generator. Labview based GUI interface is used to automate the data generator that supports the Mi-

crosoft excel files containing bit streams.

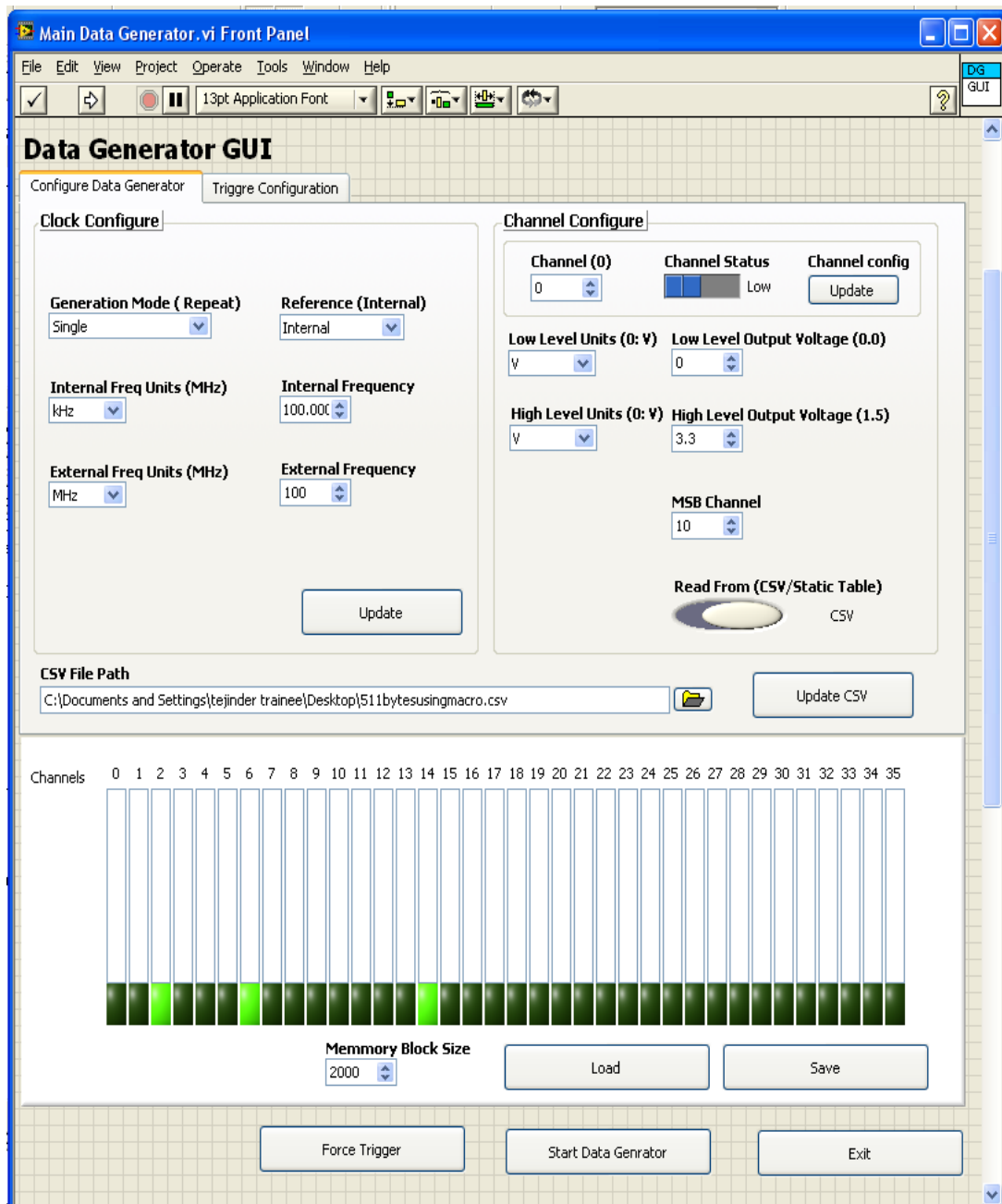


Figure 7.20 - Labview Based GUI Interface to Automate the Data Generator

Figure 7.21 shows memory read operation that confirms the memory write operation.

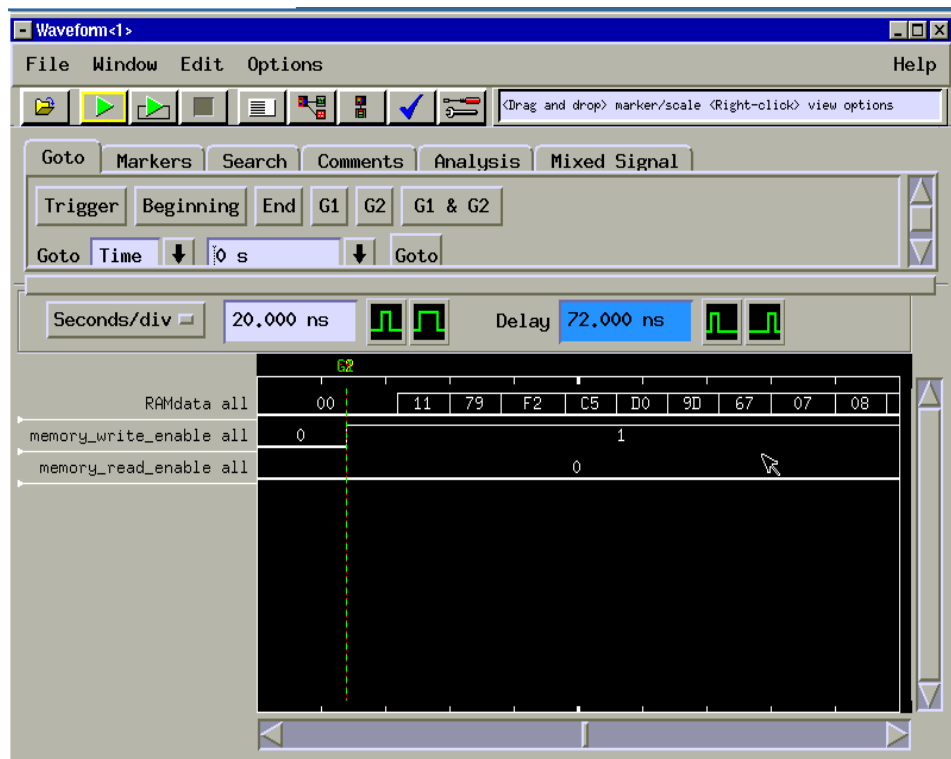


Figure 7.21 - Memory Read

7.12.1.3 Data Comparison

Just after the memory write operation is conformed. NRZI encoded data is applied at the Dp, Dn pins of the ULPI PHY. Labview script is used to encode data. As the verilog code for error counter has been burnt in the FPGA with proper constraints [32, 33, 34, 35]. It will start comparing incoming data with data stored in BRAM of FPGA and will start counting number of errors that encounter in the USB communication system only when the following conditions are met.

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 and error counter will start comparing incoming data with the stored data.
2. If DIR is previously high, PHY will send the Rxcmd till NXT is de-asserted. PHY will drive the data on ULPI data bus as soon as NXT goes high as shown in figure

7.22.

The above stated conditions validate the correct data at the ULPI interface.

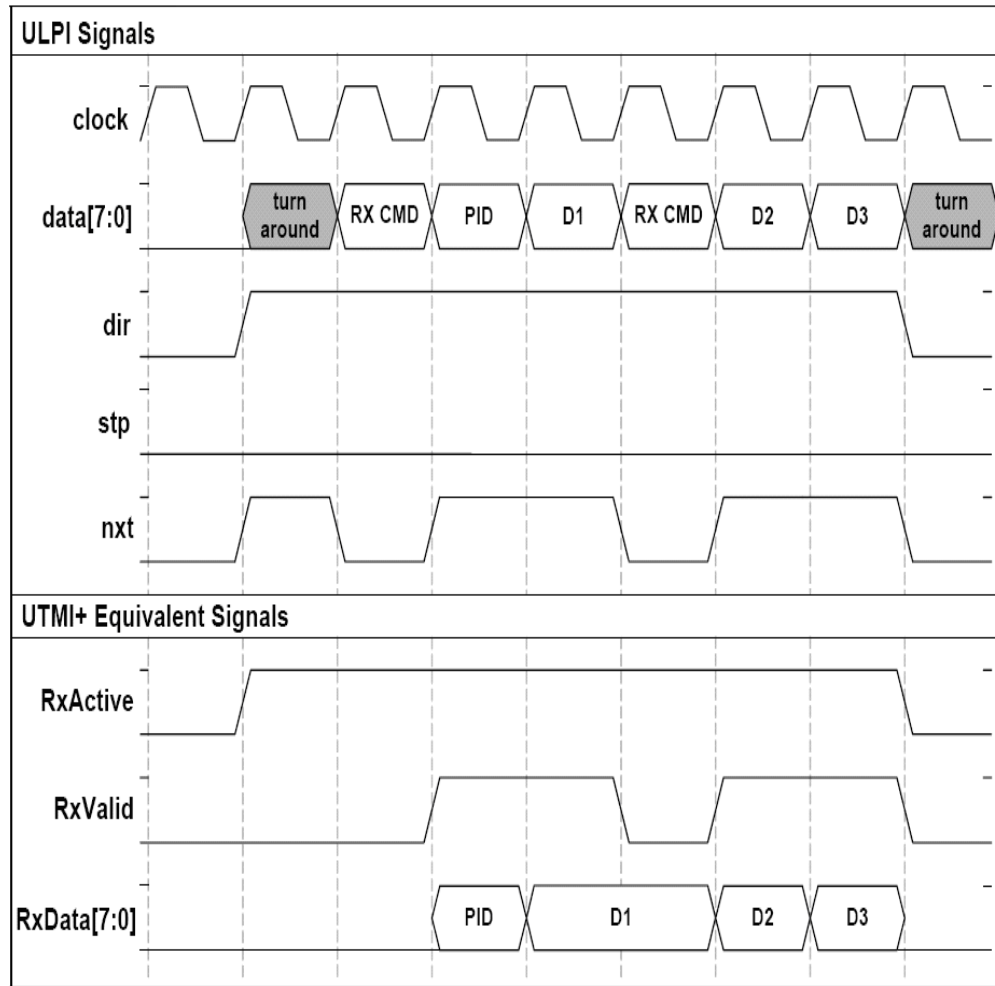


Figure 7.22 - USB Receive While dir was Previously Low

When DIR and NXT get asserted simultaneously, PHY will take turn around time for one clock cycle as shown in figure 7.22. Whenever DIR get asserted and de-asserted, 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.

During the data reception, if nxt goes low then PHY may take the control of data bus and transmit Rxcmd to Link. Figure 7.23 shows no increment in the error counter that simply means no data byte is corrupted.

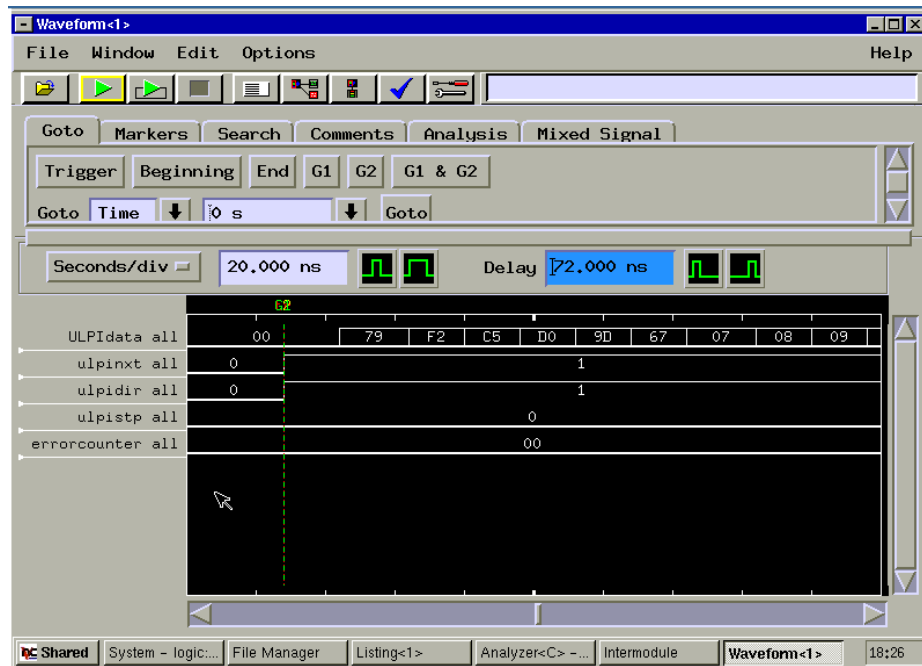


Figure 7.23 - Error Counter at Value 0

Figure 7.24 below shows that error counter starts incrementing because some of the data bytes have been corrupted.

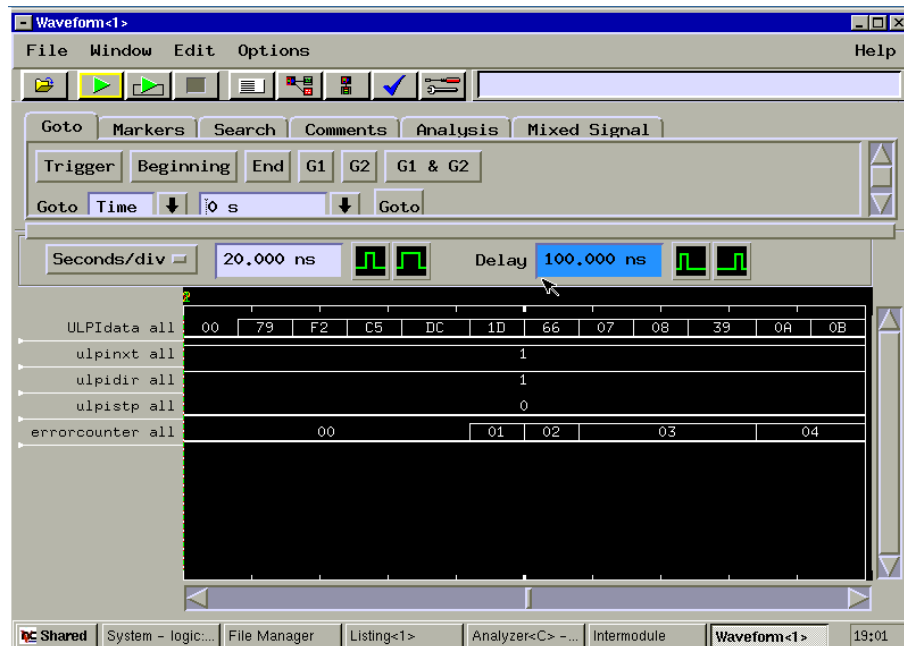


Figure 7.24 - Error Counter at Non-Zero Value

7.13 Block Diagram Showing BER Code Description

This block diagram represents the verilog code to be burnt in FPGA for calculation of bit error rate (BER). It includes the circuit which configures the ULPI PHY with the help of hand shake signals like dir, nxt and stp.

It uses FPGA's memory to store data to be compared with the received data. It also includes the comparator which compares data coming at the Dp, Dn with the stored data.

Error counter is the other important unit in this block diagram that gets incremented if comparator indicates any mismatch.

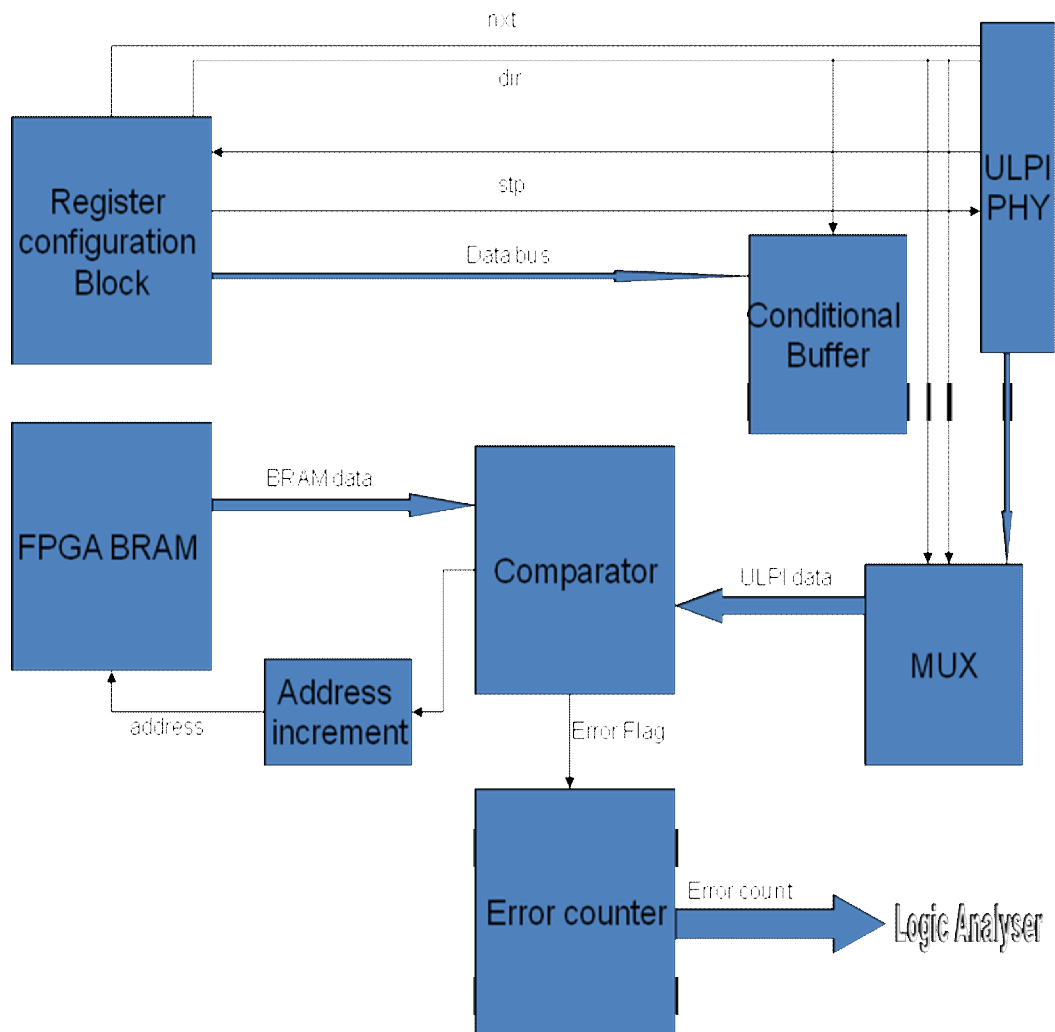


Figure 7.25 - Block Diagram Showing BER Code Description

7.14 Flow Chart Depicting Working of FPGA Based Test and Debug Platform

Below is the flow chart that explains the working of FPGA based test and debug platform.

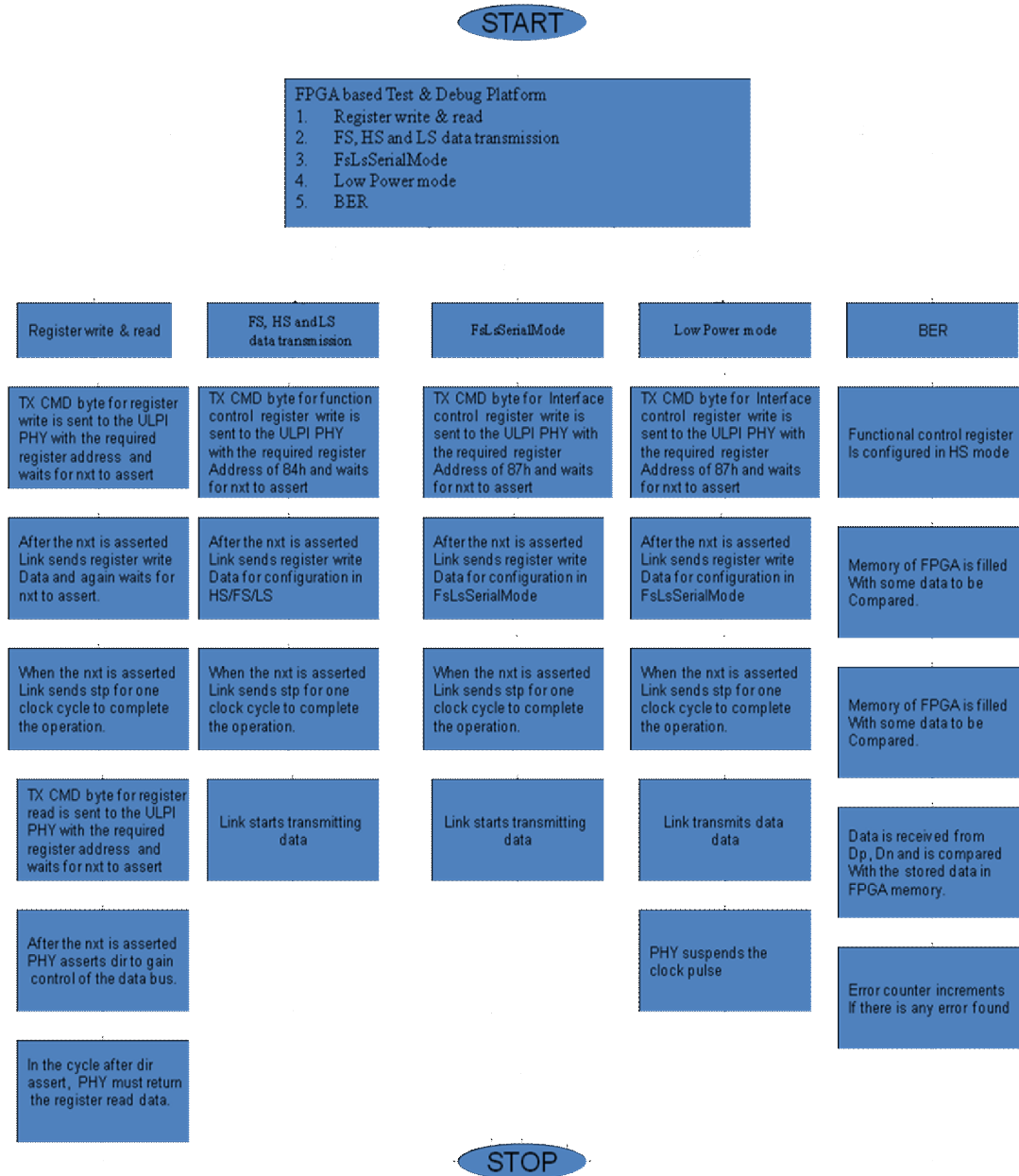


Figure 7.26 – Flow Chart for Working of FPGA Implemented Test and Debug Platform

Conclusions and Scope for Future Work

8.1 Conclusions

High speed testing and debugging is a challenging field in era of VLSI design. ULPI PHY can be validated by functional testing and BER measurement. Bit error rate ensures the reliability of the USB communication systems in a jitter affected environment.

Jitter is a dominant factor in affecting the BER of a USB data communication system. It may alter the sampling of the data and cause timing mismatch. So BER must be calculated under the worst case incorporating both transmitter and receiver clock jitter into the USB communication system.

ULPI interface replaced the UTMI + core by drastically reducing the pin count at the interface with the external transceiver. 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.

“FPGA based test and debug environment” is a low cost solution which confirms the reliability of the ULPI PHY through functional testing and also ensures bit error rate (BER) for end to end performance.

This test and debug platform incorporates the un-wrapper that converts the ULPI interface to UTMI core to facilitate the testing and debugging of ULPI PHY by avoiding repeated burning of FPGA.

BER becomes very much crucial for critical and sensitive applications like medical science equipments and ballistic missiles. USB communication for such type of applications can tolerate maximum BER of 10^{-14} .

There is a trade-off between bit error rate (BER) and various other important constraints like power dissipation, area and speed of the system. Recent tremendous up thrust in the operating frequencies of a chip leads to more critical problems regarding bit error rate.

In a nut shell, higher values of bit error rate can be accommodated for hand held devices like walkman, mobile phones etc. which require large battery life and perform at a very high speed.

8.2 Scope for Future Work

The testing and debugging techniques discussed so far are not so much user friendly Lot of human intervention is required for their proper operation and performance.

In this era of automated testing and debugging, integrated testing and debugging systems have been proposed that integrates program testing and debugging processes into a series of consecutive processes and supports automation of each of these processes.

Moreover, GUI based automated testing and debugging platforms are on the shelf for some systems. But GUI based automated testing and debugging platforms for USB system are still on the way to development.

References

- [1] Universal Serial Bus Specifications, Revision 2.0.
- [2] 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”, in Proc. *4th southern IEEE conf. Programmable Logic*, pp. 149-154, March 26-28, 2008.
- [3] UTMI+ Low Pin Interface (ULPI) Specification, Revision 1.1.
- [4] Gray Breed, “Bit Error Rate: Fundamental Concepts and Measurement Issues ”, *Summit Technical Media*, Jan. 2003
- [5] L. Schooley and G. Davis, “Jitter effects on bit error rate reexamined”, *IEEE Trans. Communications*, Vol. 29, no. 6, pp. 920-922, June 1981.
- [6] USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, v1.05.
- [7] UTMI+ Specification, Revision 1.0.
- [8] Don Anderson, “Universal Serial Bus System Architecture”, 2nd edition, MindShare Inc. 2001. ISBN-13: 978-0201309751
- [9] On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a.
- [10] CEA-2011, OTG Transceiver Specification.
- [11] K. Lynn, “Universal Serial Bus (USB) Power Management”, in Proc. *IEEE Conf. WESCON-97*, pp. 434-441, Nov. 4-6, 1997.
- [12] Xinjun Zhang, Y. Tashiro, H. Takata, T. Fujii, M. Yamamuro, D. Kasai and T. Shimomura, “An approach to automated program testing and debugging”, in Proc. *6th IEEE Conf. Software Engineering*, pp. 582-589, Dec. 7-10, 1999.
- [13] R.B. Anderson, M. Borowczak and P.A. Willsey, “The use of device simulation in development of USB storage devices”, in proc. *41st Annual IEEE Simulation Symp-*

- Osium – 2008*, pp. 220 – 226, April 13 – 16, 2008.
- [14] Hailong Jin and Bing Miao, “Design of Holter ECG System Based on MSP430 and USB Technology”, in *Proc. 1st Int. IEEE Conf. Bioinformatics and Biomedical Engineering*, pp. 976-979, July 6-8, 2007.
- [15] B. Xiaoping and Shenlei “Study on system verification of USB 2.0 interface protocol control chip hardware design”, in *Proc. Int. IEEE Conf. Electrical Engineering*, pp. 1-5, April 11-12, 2007.
- [16] Bai Xiaoping, Chen Guican and Wel Yuanfeng, “Study on the simulation method for universal serial bus 2.0 hard design based on smart model”, in *Proc. 2nd Int. IEEE Conf. Systems-2007*, pp. 22-26, April 22-28, 2007.
- [17] J.J. Nam, V. J. Kim, K. H. Choi and H. J. Park, “A UTMI - compatible physical-layer USB 2.0 transceiver chip”, in *Proc. Int. IEEE Conf. Systems on Chip*, pp. 309-312, Sept. 17-20, 2003.
- [18] “Low Pin Count Transceiver Interface to 8-bit UTMI+”, revision 1.21, Mentor Graphics Corporation.
- [19] Virtex-II FPGA Family Data Sheet. V. 3.0. product specification DS031. Xilinx Inc. August 2003.
- [20] SMSC Hi-Speed USB Host, Device or OTG PHY with ULPI Low Pin Interface Data Sheet. Product Specification EVB USB3300.
- [21] USB 2.0 transceiver and Macrocell Tester (T&MT) Interface Specification, Version 1.2.
- [22] Shimada, “Method and Apparatus for Encoding an NRZI Digital Signal With Low DC Component”, *United States Patent*, May 1985.
- [23] K. Babulu and K. Soundara Rajan, “FPGA implementation of USB transceiver macrocell interface with USB-2.0 specifications”, in *Proc. 1st IEEE Int. Conf. Emerging Trends in Engineering and Technology*, pp. 966-970, July 16 2008.

- [24] Chip Scope Pro 10.1 Software and Cores User Guide, UG029 (v10.1), Xilinx inc. March 24, 2008.
- [25] Samir Palnitkar, “Verilog HDL a guide to digital design and synthesis”, 2nd edition, Prentice Hall Publication.
- [26] Jayaram Bhasker, “A VHDL Primer”, 2nd edition, Prentice Hall Publications.
- [27] Data sheet and user guide, Tektronix data timing generator DTG 5274.
- [28] Reference manual-STM32F10x, ST Microelectronics, June 2007.
- [29] CEA-936A, Mini-USB Analog Car kit Interface Specification.
- [30] E.J. Dean, P. D. Dresseihaus, J. X. Przybysz, A. H. Miklich, A. Hodge Worsham and S. V. Polonsky, “Bit error rate measurements for GHz code generator circuits”, *IEEE Trans. Applied Superconductivity*, Vol. 9, no. 3, pp. 3598-3601, June 1999.
- [31] Palani Subbiah, “Bit-Error Rate (BER) For High-Speed Serial Data Communication”, *Cypress Semiconductor*, June 2004.
- [32] Lo Ko, C. Healy, E. Ratliff, R. Arnold, D. Whalley and M. Harmon, “Supporting the specification and analysis of timing constraints”, in Proc. *IEEE Symp. Real-Time Technology and Applications*, pp. 170-178, June 10-12, 1996.
- [33] Xilinx Timing Constraints User Guide UG612, Version-1.0.0. Xilinx inc. December 2008.
- [34] P. Eles, K. Kuchcinski, Z. Peng and A. Doboli, “Timing constraint specification and synthesis in behavioral VHDL”, in Proc. *IEEE Conf. Design Automation*, pp. 452-457, Sept. 18-22, 1995.
- [35] Constraints Guide UG625, Version 11.1.0. Xilinx inc. April 2009.