DEVELOPING VERIFICATION ENVIRONMENT FOR THE USB 2.0 AND WIRELESS USB PHYSICAL LAYER

A DISSERTATION

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

MASTER OF TECHNOLOGY
in
ELECTRONICS AND COMMUNICATION ENGINEERING
(With Specialization in Communication Systems)

By

SRINIVAS KUMAR KATA

G13918
GHEERAD MEMORY
Acc. No. G13918
Date 6/11/08
I.I.T. ROORKEE
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
February, 2008

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in this dissertation, entitled **"DEVELOPING VERIFICATION ENVIRONMENT FOR THE USB 2.0 AND WIRELESS USB PHYSICAL LAYER"**, being submitted in partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** with specialization in **COMMUNICATION SYSTEMS**, in the Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee is an authentic record of my own work carried out from January 2007 to July 2007 on "USB 2.0" only, under guidance and supervision of, Mr. Jatinder Verma, Project Manager, Freescale Semiconductors Pvt Ltd INDIA, and continued on Wireless USB from July 2007 to February 2008, the total duration from January 2007 to February 2008 under guidance and supervision of, Dr. Arun Kumar, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee.

The results embodied in this dissertation have not submitted for the award of any other Degree or Diploma.

Date: February, 2008
Place: Roorkee

x. Srinivas
SRINIVAS KUMAR KATA

CERTIFICATE

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

Jatinder Verma

(External Guide)
Mr. Jatinder Verma
Project Manager
Freescale Semiconductors Pvt Ltd
Noida -201301 UP (INDIA)

(Internal Guide)
Dr. Arun Kumar
Professor, E&C Department
Indian Institute of technology, Roorkee
Roorkee – 247 667, (INDIA)

Freescale Semiconductor India Pvt. Ltd. Express Trade Towers, Floor Nos. 5, 6 & 7, Plot # 15 & 16, Sector-16 A,
Noida-201 301 U.P. (India)
Tel: +91-120-4395000 Fax: +91-120-4395199
Regd. Office : F-40, South Extension Part-I, New Delhi-110 049

freescale™
semiconductor

July 13 ,2007

TRAINING CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Srinivas Kumar Kata** was working with our organization as **Project-Trainee** during the period **15-Jan-07** to **13-July-07** and has successfully completed his Project in the area of "**Developing Verification Environment for the USB 2.0 & OTG**".

We wish his all the best for future endeavors.

For **Freescale Semiconductor India Private Limited.**

Jatinder Verma

Jatinder Verma
Design Manager

ACKNOWLEDGEMENTS

It gives me immense pleasure to acknowledge the help and cooperation that I received during this final semester project work.

I take this opportunity to extend my sincere thanks to my project manager, *Mr. Jatinder Verma,* who has the attitude and the substance of a genius, and leads his innovative team most effectively. I thank him for giving me an opportunity of working in this interesting field with his outstanding team.

I would like to place on record my earnest thanks to *Mr. Narendra Reddy,* who continually and convincingly conveyed a spirit of adventure in regard to my internship. He showed me different ways to approach a problem. A special thanks goes to him for his insightful comments & encouragement. Without his guidance and persistent help this dissertation would not have been possible.

I thank my project internal guide, *Dr. Arun Kumar*, for giving me the opportunity to work in a very interesting area, for always being patient and cooperative during this project work and for his support and guidance throughout my post graduate studies at IIT Roorkee.

I also express my heartfelt thanks to *Dr. D.K.Mehra*, Professor and Head, Department of E & C Engineering, IIT Roorkee for his support and encouragement during the course of my project work.

SRINIVAS KUMAR KATA

ABSTRACT

KEYWORDS: Universal Serial Bus Protocol (USB), OTG, UTMI, UTMI+, ULPI, VCS, SystemVerilog.

Freescale Semiconductors is working upon devising an integrated circuit for cellular subscriber device platforms, which aims at satisfying the requirements of high-tier products in the 2008-2009 time frame. Our focus is on one of its modules i.e., the USB 2.0 with OTG Support.

The USB2.0 protocol is a Serial protocol used at the interface for communication between the Host and the Device. It exploits differential data transmit advantage by just having 2 data lines. It is used to connect the external devices to the processor peripheral. The processor peripheral has IP bus interface to the other. Device peripheral will be USB compliant. There is OTG support which is capable of interchanging the roles of Host and Device. UTMI, UTMI+, ULPI interfaces which are introduced for the ease of modeling the USB protocol implementation for the IP vendors are also supported. All the simulations are compiled and run by using the tool VCS.

This project work aims at creation and simulation of a SystemVerilog based USB Verification IP for verifying the USB 2.0 OTG design. SystemVerilog provides a set of hardware-oriented modeling constructs within the context of C++ and Verilog and is emerging as a standard for high-level design, modeling & verification. The USB VIP supports USB2.0 functionality with OTG support for all the Interfaces Serial/UTM+/ULPI and all the speeds Low-/Full-/High-speed.

The Wireless USB is a Wireless protocol which is compatible with the existing USB2.0 which requires no wire connectors. The Physical Layer of the Wireless USB is implemented and verified in MATLAB which verifies all the modules present in the Physical Layer of the Wireless USB. The Program in MATLAB is done using ordinary OFDM instead of Multi Band OFDM as mentioned in the Standard.

Contents

ABBREVIATIONS

USB 2.0	- Universal Serial Bus with High-speed support
OTG	- On-The-Go Support
UTMI	- USB 2.0 Transceiver Macrocell Interface
UTMI+	- UTMI with OTG support
ULPI	- UTMI+ Low Pin Interface
IP	- Intellectual Property
VIP	- Verification IP
SRP	- Session Request Protocol
HNP	- Host Negotiation Protocol
PID	- Packet Identifier
ENDP	- END Point number
SOF	- Start of Frame
EOP	- End of Packet
AES	- Advanced Encryption Standard
CCM	- Counter Mode Encryption and Cipher Block Chaining Message Authentication Code
PLCP	- Physical Layer Convergence Protocol
PLME	- Physical Layer Management Entity
PMD	- Physical Medium Dependent
PPDU	- PLCP Protocol Data Unit
PSDU	- PHY Service Data Unit
UWB	- Ultra Wideband

1. INTRODUCTION

1.1 Motivation

The original motivation for the Universal Serial Bus (USB) [1] came from three interrelated considerations:

1. **Connection of the PC to the telephone:** It is well understood that the merge of computing and communication will be the basis for the next generation of productivity applications. The USB provides a ubiquitous link that can be used across a wide range of PC-to-telephone interconnects.
2. **Ease-of-use:** The combination of user-friendly graphical interfaces and the hardware and software mechanisms associated with new-generation bus architectures have made computers less confrontational and easier to reconfigure. However, from the end user's point of view, the PCs I/O interfaces, such as serial/parallel ports, keyboard /mouse /joystick interfaces, etc., do not have the attributes of plug-and-play.
3. **Port expansion:** The addition of external peripherals continues to be constrained by port availability. The lack of a bidirectional, low-cost, low-to-mid speed peripheral bus has held back the creative proliferation of peripherals such as telephone/fax/modem adapters, answering machines, scanners, PDAs, keyboards, mice, etc. Existing interconnects are optimized for one or two point products. As each new function or capability is added to the PC, a new interface has been defined to address this need.

The more recent motivation for USB 2.0 stems from the fact that PCs have increasingly higher performance and are capable of processing vast amounts of data. At the same time, PC peripherals have added more performance and functionality. User applications such as digital imaging demand a high performance connection between the PC and these increasingly sophisticated peripherals. USB 2.0 addresses this need by adding a third transfer rate of 480 Mb/s to the 12 Mb/s and 1.5 Mb/s originally defined for USB. USB 2.0 is a natural evolution of USB, delivering the desired bandwidth increase while preserving the original motivations for USB and maintaining full compatibility with existing peripherals.

Wireless USB: As technology innovation marches forward, Wireless technologies are becoming more and more capable and cost effective. Ultra-Wideband (UWB) radio

technology, in particular, has characteristics that much traditional USB usage models very well. UWB supports high bandwidth (480Mb/s) but only at limited range (~3 mts). Applying this wireless technology to USB frees the user from worrying about cables, it makes USB even easier to use. Because no physical ports are required, port expansion, or even finding the USB port, is no longer a problem. Losing the cable also means losing a source of power for peripheral which are powered by bus but not for self-powered.

Thus USB (wired or wireless) continues to be the answer to connectivity for the PC architecture. It is a fast, bi-directional, isochronous, low-cost, dynamically attachable interface.

1.2 Statement of the Problem

This work is aimed for the Verification of the USB 2.0 OTG Support as it is required before producing the design into a chip. To verify the design, a Verification IP is required to do all the verifications supported by USB 2.0 Protocol. There are 3 types of speeds supported by the protocol. There are 3 types of interfaces supported by the protocol. The Vendor has to verify all the interfaces in all speeds for USB2.0 protocol and OTG supplement.

This work also aims at the verification Wireless USB protocol at physical layer using the MATLAB.

1.3 Organization of Report

The report is organized in six chapters including this chapter. Chapter one gives an overview of the introduction and motivation, summarizes the problem statement for the thesis work. Chapter two gives overview of USB architecture and communication flow of the USB 2.0. Chapter three gives description of protocol types of packets, transactions and transfers and also explains the On-The-Go supplement, UTMI, UTMI+, ULPI specifications along with results obtained from the simulation. Chapter four deals with the introduction of the Wireless USB protocol from the Physical layer point of view. The fifth chapter deals with the simulation done, results obtained and code for wireless USB Physical Layer in MATLAB.

2. UNIVERSAL SERIAL BUS

2.1 Goals for the Universal Serial Bus

The USB is specified to be an industry-standard extension to the PC architecture with a focus on PC peripherals that enable consumer and business applications. The following criteria were applied in defining the architecture for the USB [1]:

1. Ease-of-use for PC peripheral expansion.
2. Low-cost solution that supports transfer rates up to 480 Mb/s.
3. Full support for real-time data for voice, audio, and video.
4. Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging.
5. Provision of a standard interface capable of quick diffusion into product.
6. Enabling new classes of devices that augment the PCs capability.

2.2 Feature List

The USB Specification provides a selection of attributes that can achieve multiple price/performance integration points and can enable functions that allow differentiation at the system and component level. Features are categorized by the following benefits:

1. Easy to use for end user with self-identifying peripherals, automatic mapping of function to driver and configuration and dynamically attachable and reconfigurable peripherals.
2. Wide range of bandwidths ranging from a few kb/s to several hundred Mb/s, capable of supporting 127 physical devices, supports transfer of multiple data and message streams between the host and devices with Low protocol overhead.
3. Guaranteed bandwidth, low latencies appropriate for telephony, audio, video, etc.
4. Flexibility in packet sizes, data rates and flow control for buffer handling.
5. Robustness for Error handling/fault recovery mechanism, Dynamic insertion and removal of devices.
6. Protocol is simple to implement with plug-and-play architecture and leverages existing operating system interfaces.

7. Low-cost implementation.
8. Architecture upgradeable to support multiple USB Host Controllers in a system.

2.3 Architecture overview of the USB

The USB is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host scheduled, token-based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals are in operation.

2.3.1 USB System

A USB system is described by three definitional areas:

1. USB Interconnect.
2. USB Host.
3. USB Devices.

USB Interconnect

The USB interconnect is the manner in which USB devices are connected to and communicate with the host. This includes the following:

- Bus Topology: Connection model between USB devices and the host.
- Inter-layer Relationships: In terms of a capability stack, the USB tasks that are performed at each layer in the system.
- Data Flow Models: The manner in which data moves in the system over the USB between producers and consumers.
- USB Schedule: The USB provides a shared interconnect. Access to interconnect is scheduled in order to support isochronous data transfers and to eliminate arbitration overhead.

USB Host

There is only one host in any USB system. The USB interface to the host computer system is referred to as the Host Controller. The Host Controller may be implemented in a combination of hardware, firmware, or software. A root hub is integrated within the host system to provide one or more attachment points.

USB Devices

USB devices are one of the following:

1. Hubs, which provide additional attachment points to the USB

2. Functions, which provide capabilities to the system, such as an ISDN connection, a digital joystick, or speakers

USB devices present a standard USB interface in terms of the following:

1. Their comprehension of the USB protocol.
2. Their response to standard USB operations, such as configuration and reset.
3. Their standard capability descriptive information.

2.3.2 Physical Interface

The USB transfers signal and power over a four-wire cable. The signaling occur over two wires (D+/D-) on each point-to-point segment. The power is transmitted over one wire Vbus.

2.3.3 Power

There are two aspects of power:

1. Power distribution over the USB deals with the issues of how USB devices consume power provided by the host over the USB.USB devices that rely totally on power from the cable are called bus-powered devices. In contrast, those that have an alternate source of power are called self-powered devices.
2. Power management deals with how the USB System Software and devices fit into the host-based power management system. The USB System Software interacts with the host's power management system to handle system power events such as suspend or resume.

2.3.4 Bus Protocol

The USB is a polled bus. The Host Controller initiates all data transfers. Most bus transactions involve the transmission of up to three packets. Each transaction begins when the Host Controller, on a scheduled basis, sends a USB packet describing the type and direction of transaction, the USB device address, and endpoint number. The USB data transfer model between a source or destination on the host and an endpoint on a device is referred to as a pipe. Pipes come into existence when a USB device is configured. One message pipe, the Default Control Pipe, always exists once a device is powered, in order to provide access to the devices configuration, status, and control information.

2.3.4 Robustness

There are several attributes of the USB that contribute to its robustness:

1. Signal integrity using differential drivers, receivers, and shielding.
2. CRC protection over control and data fields.

3. Detection of attach and detach and system-level configuration of resources.
4. Self-recovery in protocol, using timeouts for lost or corrupted packets.
5. Flow control for streaming data to ensure isochrony and hardware buffer management.
6. Data and control pipe constructs for ensuring independence from adverse interactions between functions.

2.3.5 System Configuration

The USB supports USB devices attaching to and detaching from the USB at any time. Consequently, system software must accommodate dynamic changes in the physical bus topology.

2.4 USB Communication Flow

The USB provides a communication services between software on the host and its USB function. Functions can have different communication flow requirements for different client-to-function interactions. The USB provides better overall bus utilization by allowing the separation of the different communication flows to a USB function. Each communication flow makes use of some bus access to accomplish communication between client and function. Each communication flow is terminated at an endpoint on a device. Device endpoints are used to identify aspects of each communication flow.

The Functions performed by the blocks in the below diagram are

1. Client Software: Software that executes on the host, corresponding to a USB device. This client software is typically supplied with the operating system or provided along with the USB device.
2. USB System Software: Software that supports the USB in a particular operating system. The USB System Software is typically supplied with the operating system, independently of particular USB devices or client software.
3. USB Host Controller (Host Side Bus Interface): The hardware and software that allows USB devices to be attached to a host.
4. USB Logical Device: A USB logical device appears to the USB system as a collection of endpoints. Endpoints are grouped into endpoint sets that implement an interface. Interfaces are views to the function.
5. USB Physical Device: A piece of hardware on the end of a USB cable that performs some useful end user function.

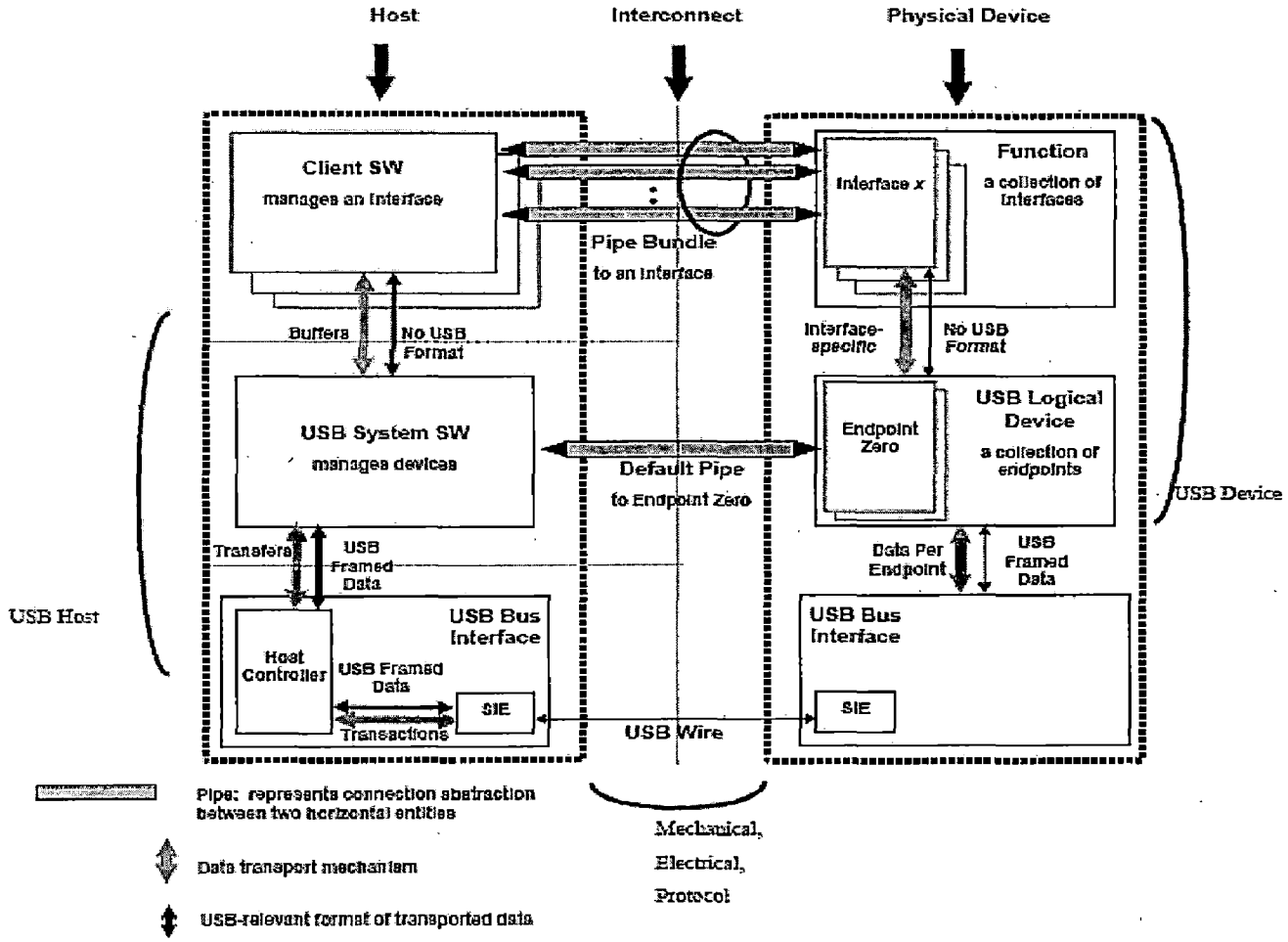


Fig 2.1: USB Host/Device Detailed View

The USB System Software manages the device using the Default Control Pipe. Client software manages an interface using pipe bundles (associated with an endpoint set). Client software requests that data be moved across the USB between a buffer on the host and an endpoint on the USB device. The Host Controller (or USB device, depending on transfer direction) packetizes the data to move it over the USB. The Host Controller also coordinates when bus access is used to move the packet of data over the USB.

Software on the host communicates with a logical device via a set of communication flows. The set of communication flows are selected by the device software/hardware designer(s) to efficiently match the communication requirements of the device to the transfer characteristics provided by the USB.

Device Endpoints

An endpoint is a uniquely identifiable portion of a USB device that is the terminus of a communication flow between the host and device. Each USB logical device is composed of a collection of independent endpoints. Each logical device has a unique address assigned by the system at device attachment time. Each endpoint on a device is given at design time a unique device-determined identifier called the endpoint number. Each endpoint has a device-determined direction of data flow. The combination of the device address, endpoint number, and direction allows each endpoint to be uniquely referenced. Each endpoint is a simplex connection that supports data flow in one direction: either input (from device to host) or output (from host to device).

Pipes

Pipes represent the ability to move data between software on the host via a memory buffer and an endpoint on a device. There are two types of pipes: stream and message. Stream data has no USB-defined structure, while message data does. Pipes have associations of data bandwidth, transfer service type, and endpoint characteristics like directionality and buffer sizes.

Frames and Microframes

USB establishes a 1 millisecond time base called a frame on a full-/low-speed bus and a 125 us time base called a microframe on a high-speed bus. A (micro) frame can contain several transactions. Each transfer type defines what transactions are allowed within a (micro) frame for an endpoint.

3. PROTOCOL ASPECTS

3.1 About USB2.0 Protocol

Byte/Bit Ordering

USB follows the little-endian order, i.e., LSB to MSB for both individual bits and fields.

SYNC Field [1]

All packets begin with a synchronization (SYNC) field, which is used by the input circuitry to align incoming data with the local clock. The SYNC pattern used for low-/full-speed transmission is required to be 3 KJ pairs followed by 2 K's for a total of eight symbols. The SYNC pattern used for high-speed transmission is required to be 15 KJ pairs followed by 2 K's, for a total of 32 symbols.

EOP Width

The width of the SE0 in the EOP is approximately 2 * TPERIOD i.e., for full-speed transmissions between 160 ns and 175 ns and for low-speed transmissions between 1.25 us and 1.50 us. In high-speed signaling, a bit stuff error is intentionally generated to indicate EOP. A receiver is required to interpret any bit stuff error as an EOP.

Packet Field Formats

Packet bit definitions are displayed in unencoded data format. All packets have distinct Start-of- Packet (SOP) delimiter which is part of the SYNC field and the End-of-Packet (EOP) delimiter.

Packet Identifier Field (PID): A PID consists of a four-bit packet type field which gives type of packet followed by a four-bit check field which is one's complement of packet type field which ensures reliable decoding of the PID so that the remainder of the packet is interpreted correctly.

- The understanding between the host and device is taken in packet form.

1. Token

FEILD	PID	ADDRESS	ENDP	CRC5
BITS	8	7	4	5

Fig 3.1: Token packet format

2. Start of Frame

FEILD	PID	Frame number	CRC5
BITS	8	11	5

Fig 3.2: SOF packet format

3. Data

FEILD	PID	DATA	CRC16
BITS	8	0-8192	16

Fig 3.3: Data packet format

Address Fields:

Function endpoints are addressed using two fields: the function address field and the endpoint field. A function needs to fully decode both address and endpoint fields.

- Address Field: Address Field <6:0> - 128 possible addresses of function ports. Function address zero is reserved as the default address and may not be assigned to any other use.
- Endpoint Field: Endpoint Field<3:0> - 16 possible varieties of combinations of transfers. All functions must support a control pipe at endpoint number zero (the Default Control Pipe). Low-speed devices support a maximum of three pipes per function. Full-speed and high-speed functions may support up to a maximum of 16 IN and OUT endpoints.

Frame Number Field: The frame number field is an 11-bit field that is incremented by the host on a per-frame basis. The frame number field rolls over upon reaching its maximum value of 7FFH and is sent only in SOF tokens at the start of each (micro) frame.

Data Field: The data field may range from zero to 1,024 bytes and must be an integral number of bytes. Data bits within each byte are shifted out LSB first.

- Low speed data field is of 0-8 bytes
- Full speed data field is of 0-511 bytes
- High speed data field is of 0-1023 bytes

Cyclic Redundancy Checks: Token and data packet CRCs provide 100% coverage for all single- and double-bit errors.

- Token CRC - A five-bit CRC field is provided for tokens and covers the ADDR and ENDP fields of IN, SETUP, and OUT tokens or the time stamp

field of an SOF token. The PING and SPLIT special tokens also include a five-bit CRC field. The generator polynomial is:

$$G(X) = X^5 + X^2 + 1$$

The binary bit pattern that represents this polynomial is 00101B. If all token bits are received without error, the five-bit residual at the receiver will be 01100B.

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

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

The binary bit pattern that represents this polynomial is 1000000000000101B. If all data and CRC bits are received without error, the 16-bit residual will be 1000000000001101B.

Handshake Packets

FEILD	PID
BITS	8

Fig 3.4: Handshake packet format

Handshake packets consist of only a PID. Handshake packets are used to report the status of a data transaction and can return values indicating successful reception of data, command acceptance or rejection, flow control, and halt conditions. There are four types of handshake packets and one special handshake packet:

- **ACK** may be issued either when sequence bits match and the receiver accepted data correctly or when sequence bits mismatch and the sender and receiver must resynchronize to each other
- **NAK** can only be returned by functions in the data phase of IN transactions or the handshake phase of OUT or PING transactions.
- **STALL** indicates that a function is unable to transmit or receive data, or that a control pipe request is not supported. Functional STALL is when the *Halt* feature associated with the endpoint is set. Protocol STALL is returned during the Data or Status stage of a control transfer, and the STALL condition terminates at the beginning of the next control transfer (setup).
- **NYET** is returned by a hub in response to a split-transaction when the full-/low-speed transaction has not yet been completed or the hub is otherwise not

able to handle the split-transaction. It is returned by a high-speed endpoint as part of the PING protocol.

- **ERR** is a high-speed only handshake that is returned to allow a high-speed hub to report an error on a full-/low-speed bus.

SPLIT

Host controllers and hubs support one additional transaction type called split transactions. This transaction type allows full- and low-speed devices to be attached to hubs operating at high-speed. These transactions involve only host controllers and hubs and are not visible to devices.

PING

The host controller queries the high-speed device endpoint with a PING special token. The endpoint either responds to the PING with a NAK or an ACK handshake. A NAK handshake indicates that the endpoint does not have space for a *wMaxPacketSize* data payload.

Transaction [1]

The mutual transfer of packets (token, data and acknowledgement) between the host and the device is called transaction. It may be of 2 levels or 3 levels.

The protocol overhead for each transaction includes:

- A SYNC field (packet): either 8 bits (full-/low-speed) or 32 bits (high-speed).
- A PID byte (packet): includes PID and PID invert (check) bits.
- An EOP (packet): 3 bits (full-/low-speed) or 8 bits (high-speed).
- In a token packet, the endpoint number, device address, and CRC5 fields (16 bits total).
- In a data packet, CRC16 fields (16 bits total).
- For transaction with multiple packets, the inter packet gap or bus turnaround time required.

For these calculations, there is assumed to be no bit-stuffing required.

Responses for the transactions from host and function for particular tokens

Table 3.1 Function and Host Responses to IN Transactions

Host	SENDS IN TOKEN				
Function Responses in one of way	Receives correctly and sends data packet.		Receives incorrectly and gives no response	Received correctly but has no data to transmit gives NAK.	Endpoint is not supporting the function to transmit the data so gives STALL
How the host reacts	If data packet is received correctly gives ACK	If data packet is not correct discards data	Retries later	Retries later.	Sets the Endpoint Configuration and Retries by sending IN token

Table 3.2 Function Responses to OUT Transactions

Host	SENDS OUT and DATA PACKETS			
Function Responses in one of way	Receives correctly and gives ACK	Receives incorrectly and gives no response	Received correctly but cannot accept data so gives NAK.	Endpoint is not supporting the function to transmit the data so gives STALL
How the Host reacts	-	Retries later	Retries later.	Sets the Endpoint Configuration and Retries by sending OUT and DATA token

Function Response to a SETUP Transaction

SETUP defines a special type of host-to-function data transaction that permits the host to initialize an endpoint's synchronization bits to those of the host. Upon receiving a SETUP token, a function must accept the data. A function may not respond to a SETUP token with either STALL or NAK, and the receiving function must accept the data packet that follows the SETUP token. If a non-control endpoint receives a SETUP token, it must ignore the transaction and return no response.

3.2 Transfer Types

The USB transports data through a pipe between a memory buffer associated with a software client on the host and an endpoint on the USB device. Data transported by message pipes is carried in a USB-defined structure, but the USB allows device-specific structured data to be transported within the USB-defined message data payload. The USB also defines that data moved over the bus is packetized for any pipe (stream or message), but ultimately the formatting and interpretation of the data transported in the data payload of a bus transaction is the responsibility of the client software and function using the pipe.

Each transfer type determines various characteristics of the communication flow including the following:

1. Data format imposed by the USB.
2. Direction of communication flow.
3. Packet size constraints.
4. Bus access constraints.
5. Latency constraints.
6. Required data sequences.
7. Error handling

The USB defines four transfer types:

Control Transfers [1]

Control transfers minimally have two transaction stages: Setup and Status.

- Bursty, non-periodic, host software-initiated request/response communication, typically used for command/status operations.
- A control transfer may optionally contain a Data stage between the Setup and Status stages. The amount of data to be sent during the data stage and its direction are specified during the Setup stage. The Status stage of a control transfer is the last transaction in the sequence.

Table3.3 Status Stage Responses

Status Response	Control Write Transfer (sent during data phase)	Control Read Transfer(sent during handshake phase)
Function completes	Zero-length data packet	ACK handshake
Function has an error	STALL handshake	STALL handshake
Function is busy	NAK handshake	NAK handshake

- The protocol STALL condition lasts until the receipt of the next SETUP transaction, and the function will return STALL in response to any IN or OUT transaction on the pipe until the SETUP transaction is received.

Bulk Transaction

The bulk transfer type is designed to support devices that need to communicate relatively large amounts of data at highly variable times that can use any available bandwidth.

Requesting a pipe with a bulk transfer type provides the requester with the following:

1. Access to the USB on a bandwidth-available basis,
2. Retry of transfers, in the case of delivery failure due to errors on the bus,
3. Guaranteed delivery of data but no guarantee of bandwidth or latency.

The following are features of the Bulk Transactions

- Non-periodic, large-packet bursty communication, typically used for data that can use any available bandwidth and can also be delayed until bandwidth is available.
- Bulk transactions use a three-phase transaction consisting of token, data and handshake packets are characterized by the ability to guarantee error-free delivery of data between the host and a function by means of error detection and retry.
- If the data is received without error by the function, it will return one of three (or four including NYET, for a high-speed device) handshakes: ACK, NAK, STALL.

Interrupt Transaction

The interrupt transfer type is designed to support those devices that need to send or receive data infrequently but with bounded service periods. Requesting a pipe with an interrupt transfer type provides the requester with the following:

1. Guaranteed maximum service period for the pipe.
2. Retry of transfer attempts at the next period, in the case of occasional delivery failure due to error on the bus

The following are features of the Interrupt Transactions

- Low-frequency, bounded-latency communication.
- If the endpoint has no new interrupt information to return the function returns a NAK handshake during the data phase.

- If the Halt feature is set for the interrupt endpoint, the function will return a STALL handshake.

Isochronous Transaction

In the USB environment, requesting an isochronous transfer type provides the requester with the following:

- Guaranteed access to USB bandwidth with bounded latency.
- Guaranteed constant data rate through the pipe as long as data is provided.
- No retry in the case of a delivery failure due to error.

The USB isochronous transfer type is designed to support isochronous sources and destinations.

- Periodic, continuous communication between host and device, typically used for time-relevant information.
- Isochronous transactions have a token and data phase, but no handshake phase.
- A high-speed Host Controller must be able to accept and send DATA0, DATA1, DATA2, or MDATA PIDs in data packets.
- High bandwidth, high-speed isochronous transactions support data PID sequencing.

There are 3 different speeds and 4 different types of transfers that USB supports and each with different payload sizes depending on the need of the endpoints only required types of transfers are supported by the protocol and each in appropriate proportion of available bandwidth. Here are the types of speeds, transfers and payload sizes (Packet Size constraints).

Table 3.4 Transfers, Speeds with Max Payload

Packet Constraints	Control	Interrupt	Bulk	Isochronous
Low-speed Payload(max)	8	8	N/A	N/A
Full-speed Payload(max)	64	64	8/16/32/64	1023
High-speed Payload(max)	64	1024	512	1024

Data Format	USB-defined structure	no data content structure	no data content structure	no data content structure
Direction	Bi-directional	Stream pipe, uni-directional	stream pipe, uni-directional	uni-directional
Bus Access Constraints	part of each (micro)frame is reserved	bus frequency and (micro) frame	On bandwidth-available basis.	Required bus access period.
Endpoint Used	In ,Out	In or Out	In or Out	In or Out

3.3 Device States

There are few possible states where the device works. Bus Enumeration is the process that the device goes into for participating in the transfers along with the Host.

1. **Attached [1]**

A USB device may be attached or detached from the USB. The current level on the resistor rises indicating the new connection. The Speed of the device will be detected at the time of attach.

- If the pull-up resistor of USB device is on D- line then the host will move to Low-speed clock after detecting the device
- If the pull-up resistor of USB device is on D+ line then the host will move to Full-speed clock and at the time of reset changes to High-speed depending on negotiation.

The device will permitted to data transfer only after the reset state

2. **Powered**

USB devices may obtain power from an external source and/or from the USB through the host to which they are attached. Externally powered USB devices are termed self- powered. If the device gets power through the Host it is termed as bus-powered. A device may support both or either of self-powered and bus-powered configurations.

3. **Default Address**

Host assigns a default address to the device initially after powered it gives the default address and zero-endpoint address. In this state the transfers between the host and device will take place through the default control pipe.

RESET PROCESS:

A device that is capable of high-speed operation determines whether it will operate at high-speed as a part of the reset process. After detecting SE0 for 2.5us the device recognizes that host is resetting. The reset for Low-speed is for waiting 10ms in SE0 state.

- The HS Detection Handshake is Set along with the reset signaling which is of atleast 10ms.
- A device has up to 6 ms after the reset process starting to assert a minimum of a 1 ms Chirp K.
- If the host is of HS capable then it has to respond for this Chirp K state within 100 us of its completion.
- Then the host has to assert with Chirp K and Chirp J states alternately for at least 3 times and each of the state having an interval of 40-60 us.
- As soon as the device detects the 3 KJ sequences it sets into HS mode by removing the D+ Pull-up resistor and asserts HS terminations, reverts to HS default state and waits for end of reset.
- The channel has to be again set back to SE0 for atleast of 100 us upto a maximum of 500 us. Reset must wake a device from the Suspend state.
- The reset signaling is compatible with low-/full-speed reset.

After the device is successfully reset, the device must also respond successfully to device and configuration descriptor requests and return appropriate information.

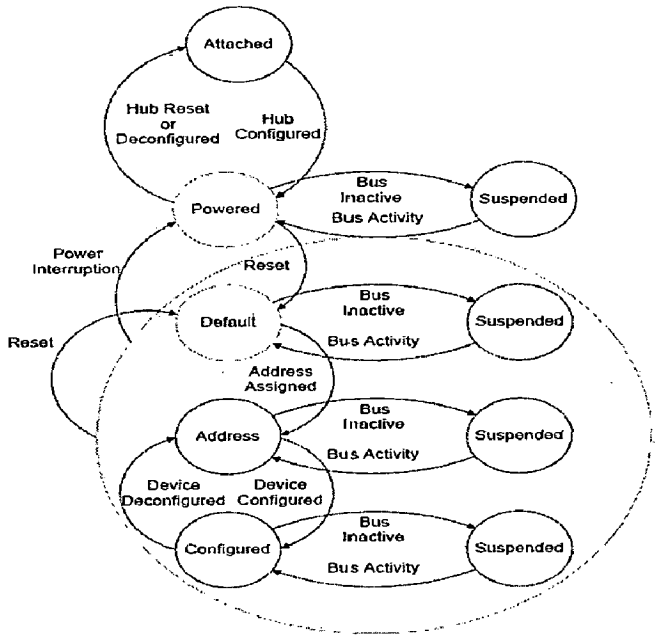


Fig 3.5: VISIBLE DEVICE STATES

4. Address

All USB devices use the default address when initially powered or after the device has been reset. Each USB device is assigned a unique address by the host after attachment or after reset. A USB device responds to requests on its default pipe whether the device is currently assigned a unique address or is using the default address. The host assigns a unique address depending on the hub to which the device is attached.

5. Configured

Before a USB device's function may be used, the device must be configured. From the device's perspective, configuration involves correctly processing the request with a non-zero configuration value. After reading the device descriptors the host configures the device. Configuring a device or changing an alternate setting causes all of the status and configuration values associated with endpoints in the affected interfaces to be set to their default values.

Descriptor

- Descriptors are also defined that can be used to contain different information on the device. Control transfers provide the transport mechanism to access device descriptors and make requests of a device to manipulate its behavior.
- In order to determine the maximum packet size for the Default Control Pipe, the USB System Software reads the device descriptor. The host will read the first eight bytes of the device descriptor. The device always responds with at least these initial bytes in a single packet. After the host reads the initial part of the device descriptor, it is guaranteed to have read this default pipe's wMaxPacketSize field (byte 7 of the device descriptor). The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.

6. Suspended

In order to conserve power, USB devices automatically enter the suspended state. When the device has observed no bus traffic for a specified period attached device must be prepared to suspend at any time they are powered. A USB device exits suspend mode when there is bus activity. A USB device may also request the host to exit suspend mode or selective suspend by using electrical signaling to indicate

remote wakeup. The SOF token will occur once per (micro) frame to keep full-/high-speed devices from suspending.

While in the Suspend state, a device must continue to provide power to its D+ (full-/high-speed) or D- (low-speed) pull-up resistor to maintain an idle so that the upstream hub can maintain the correct connectivity status for the device. When a device operating in high-speed mode detects that the data lines have been in the high-speed idle state for at least 3.0 ms, it must revert to the full-speed configuration no later than 3.125 ms after the start of the idle state. No earlier than 100us and no later than 875 us after reverting to full-speed, the device must sample the state of the line. If the state is a full-speed J, the device continues with the suspend process. SE0 would have indicated that the downstream facing port was driving reset, and the device would have gone into the High-speed Detection Handshake. When the resume occurs, the device or host transceiver must revert to high-speed without the need for a reset.

Resume

Resume signaling is used by the host or a device to bring a suspended bus segment back to the active condition. Hubs play an important role in the propagation and generation of resume signaling. It must send the resume signaling for at least 20 ms and then end the resume signaling in one of two ways, depending on the speed at which its port was operating when it was suspended. If the port was in low-/full-speed when suspended, the resume signaling must be ended with a standard, low-speed EOP (two low-speed bit times of SE0 followed by a J). If the port was operating in high-speed when it was suspended, the resume signaling must be ended with a transition to the high-speed idle state.

Remote wakeup

Remote wakeup allows a suspended USB device to signal a host that may also be suspended. This notifies the host that it should resume from its suspended mode, if necessary, and service the external event that triggered the suspended USB device to signal the host. If a device supports remote wakeup, it must also allow the capability to be enabled and disabled using the standard USB requests. The device can signal the system to resume operation if its remote wakeup capability has been enabled by the USB System Software. The remote wakeup device must hold the resume signaling for at least 1ms but for no more than 15ms.

3.4 OTG Specification

3.4.1 Introduction [2]

This section is used to describe the feature requires for an ordinary device to become an On-The-Go device which is capable of initiating a new session from a session end state and also negotiate with the host to become host. An ordinary device can be connected to an OTG capable device and data can be transferred as the ordinary USB 2.0 except with low-speed transactions. In addition to being fully compliant USB2.0 device, an OTG device must include the following features and characteristics

- A limited Host capability
- Full-speed operation as a device
- Full-speed support as a Host
- Session Request Protocol
- Host Negotiation Protocol
- One, and only one connection, a Mini-AB receptacle
- Minimum current output on Vbus.

The USB 2.0 specification defines the following connector pairs:

- Standard-A plug and receptacle for the host;
- Standard-B plug and receptacle for the peripheral; and
- Min-B plug and receptacle as alternative connectors for the peripheral.

The OTG supplement defines the following connector components:

- Mini-A plug,
- Mini-A receptacle, and
- Mini-AB receptacle.

The Mini-AB receptacle accepts either a Mini-A plug or a Mini-B plug.

3.4.2 Session Request Protocol

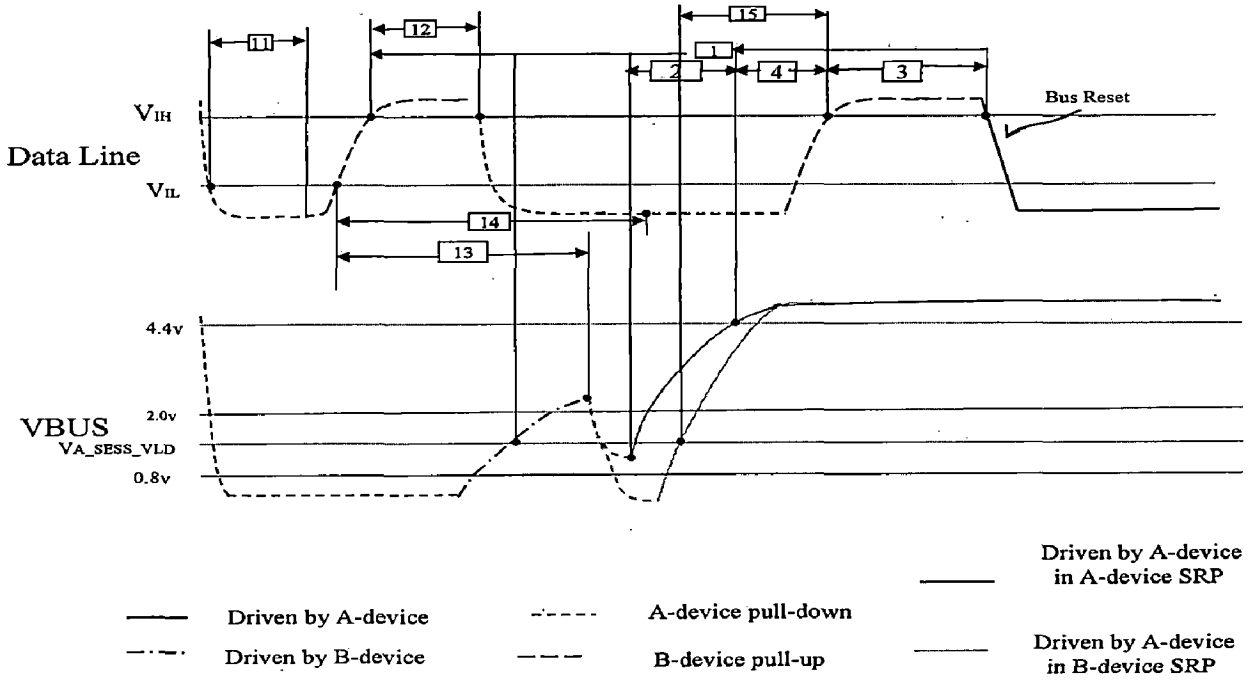


Fig 3.6 SRP Timing

Purpose: In order to conserve power, an A-device is allowed to leave the VBUS turned off when the bus is not being used. If the B-device wants to use the bus when VBUS is turned off, then it requires some way of requesting the A-device to supply power on VBUS.

SRP is to be used as follows: An OTG device is required to respond to SRP if it ever turns off VBUS while an A-plug is inserted. An OTG device that keeps VBUS turned on whenever an A-plug is inserted will never have a need to respond to SRP.

Methods used by an OTG device: There are 2 methods called "data-line passing" and "Vbus pulsing" which a B-device uses to request the A-device begin session which is called Session Request Protocol.

Initial Conditions:

1. The B-device may not attempt to start a new session until it has determined that the A-device should have detected the end of the previous session. The A-device detects the end of a session by sensing that Vbus has dropped below its session valid threshold.

2. The B-device must detect that both the D+ and D- data lines must have been low (SE0) for atleast Tb_se0_srp min. This ensures that the A-device has detected a disconnect condition from the device.

These initial conditions define the period after which the A-device will properly recognize SRP from the B-device. When the above conditions satisfy any previous session on the A-device is over and a new session may start.

Data-line Pulsing: To indicate a request for a new session using the data-line pulsing SRP, the B-device waits until the initial conditions are met and then turns on its data line pull-up resistor D+ for a period within the range specified by Tb_data_pls. The duration of such a data line pulse is sufficient to allow the A-device to reject spurious voltage transients on the data lines.

Vbus Pulsing: The B-device drives Vbus anytime after the initial conditions are met and data line pulsing has concluded. Vbus is driven for a period that is long enough, max to be charged to Vb_otg_out min and will not be charged above Vb_hst_out max. There are 2 scenarios that a B-device could encounter when pulsing Vbus to initiate SRP. In one scenario, the B-device is connected to an A-device that responds to the Vbus pulsing SRP, where the B-device can drive Vbus above the A-device session valid threshold in order to wake up the A-device. When driving such an A-device, the B-device shall ensure that Vbus goes above Vb_otg_out min, but does not exceed Vb_otg_out max. In second scenario, the B-device is attached to a standard host; the B-device shall not drive Vbus above Vb_hst_out max. This insures that no damage is done to standard hosts that are not designed to withstand a voltage externally applied to Vbus.

Duration of SRP: The maximum time allowed for the B-device to complete all of its SRP initiation activities is Tb_srp_init max. The SRP activities include all those activities that transpire while the B-device is not monitoring the state of the Vbus.

Response time of A-device: The A-device may be designed to respond to either of the methods of SRP. After initiating SRP, the B-device is required to wait at least Tb_srp_fail min for the A-device to respond, before informing the user that the communication attempt has failed. For this reason, it is recommended that the A-device is to turn on Vbus and generate a bus reset.

3.4.3 Host Negotiation Protocol [2]

HNP is used to transfer control of a connection from the default host (A-device) to default peripheral (B-device). This is accomplished by having the A-device condition the B-device to be able to take control of the bus, and then having the A-device present an opportunity for the B-device to take control.

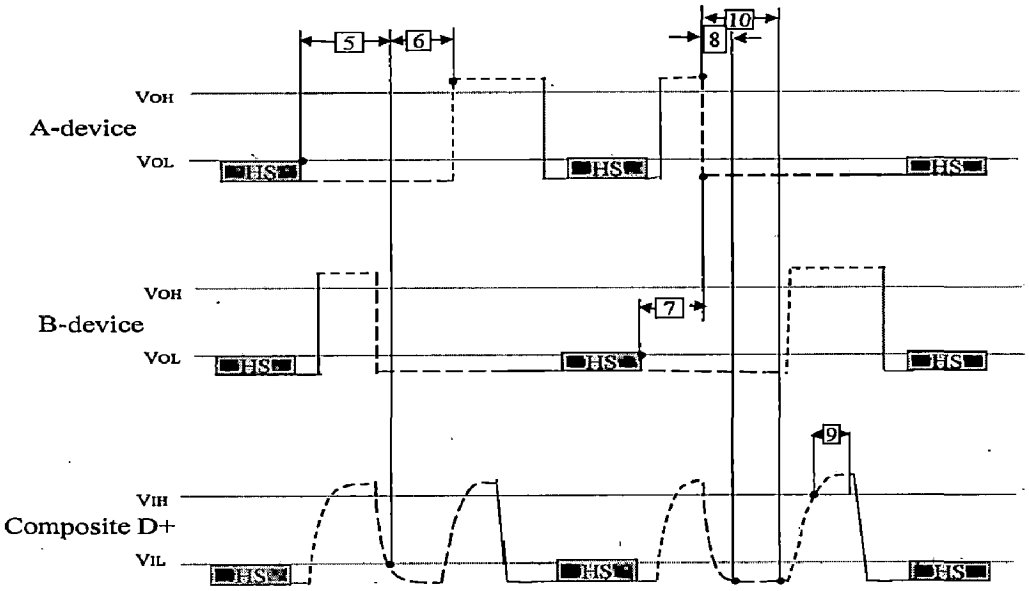


Fig 3.7 Timing Diagram of the HNP for HS devices

The sequence of events for HNP as observed on the USB, are as shown in above figure

- A-device finishes using bus and stops all bus activity (i.e., suspends the bus).
- B-device detects that bus is idle for more than Ta_aidl_bdis min and begins HNP by turning off pull-up on D+. This allows the bus to discharge to the SE0 state. If the bus was operating in HS mode, the B-device will first enter the full-speed mode and turn on its D+ pull-up resistor for atleast Tb_fs_bdis min before turning off its pull up to start the HNP sequence.
- The A-device detects the SE0 on the bus and recognizes this as a request from the B-device to become Host. The A-device responds by turning on its D+ pull-up within Ta_bdis_acon max of first detecting the SE0 on the bus.
- After waiting long enough to insure that the D+ line cannot be high due to the residual effect of the B-device pull-up, the B-device sees that the D+ line is high and D- line is low. This indicates that the A-device has recognized the HNP request from the B-device. The B-device becomes Host and asserts bus reset within Tb_acon_bse0 max to start using the bus.
- When the B-device completes using the bus, it stops all the bus activity.
- A-device detects lack of bus activity for more than Ta_bidl_adis min and turns off its D+ pull-up. If the A-device has no further need to communicate with the B-device, the A-device may turn off Vbus and end the session.
- B-device turns on its pull-up.

- After waiting long enough to insure that the D+ line cannot be high due to the residual effect of the A-device pull-up, the A-device sees that the D+ line is high(and D- line is low) indicating that the B-device is signaling a connect and is ready to respond as a peripheral. The A-device becomes host and asserts bus reset to start using the bus.

The timers in this section are defined in appendix.

3.5 USB 2.0 Transceiver Macrocell Interface (UTMI) Specification

3.5.1 Introduction [3]

High volume USB 2.0 devices will be designed using ASIC technology with embedded USB 2.0 support. For full-speed USB devices the operating frequency was low enough to allow data recovery to be handled in a vendors VHDL code, with the ASIC vendor providing only a simple level translator to meet the USB signaling requirements. Today's gate arrays operate comfortably between 30 and 60MHz. With USB 2.0 signaling running at hundreds of MHz, the existing design methodology must change. As operating frequencies go up it becomes more difficult to compile VHDL code without modification. This section defines the USB 2.0 Transceiver Macrocell Interface (UTMI) and many operational aspects of the USB 2.0 Transceiver Macrocell (UTM). The intent of the UTMI is to accelerate USB 2.0 peripheral development. ASIC vendors and foundries will implement the UTM and add it to their device libraries. Peripheral and IP vendors will be able to develop their designs, insulated from the high-speed and analog circuitry issues associated with the USB 2.0 interface, thus minimizing the time and risk of their development cycles. There are assumed to be three major functional blocks in a USB 2.0 peripheral ASIC design: the UTM, the Serial Interface Engine (SIE), and the Device specific logic.

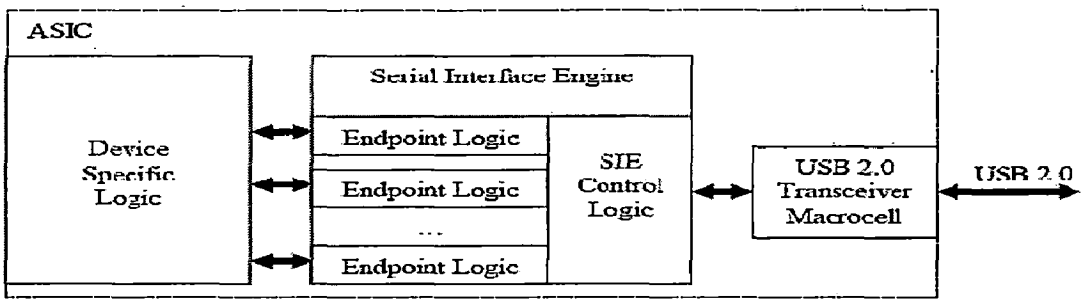


Fig 3.8 ASIC Functional Blocks

3.5.2 USB 2.0 Transceiver Macrocell (UTM):

This block handles the low level USB protocol and signaling. This includes features such as data serialization and deserialization, bit stuffing and clock recovery and synchronization. The primary focus of this block is to shift the clock domain of the data from the USB 2.0 rate to one that is compatible with the general logic in the ASIC.

Some key features of the USB 2.0 Transceiver are:

- Eliminates high speed USB 2.0 logic design for peripheral developers.
- Standard Transceiver interface enables multiple IP sources for USB 2.0 SIE VHDL.

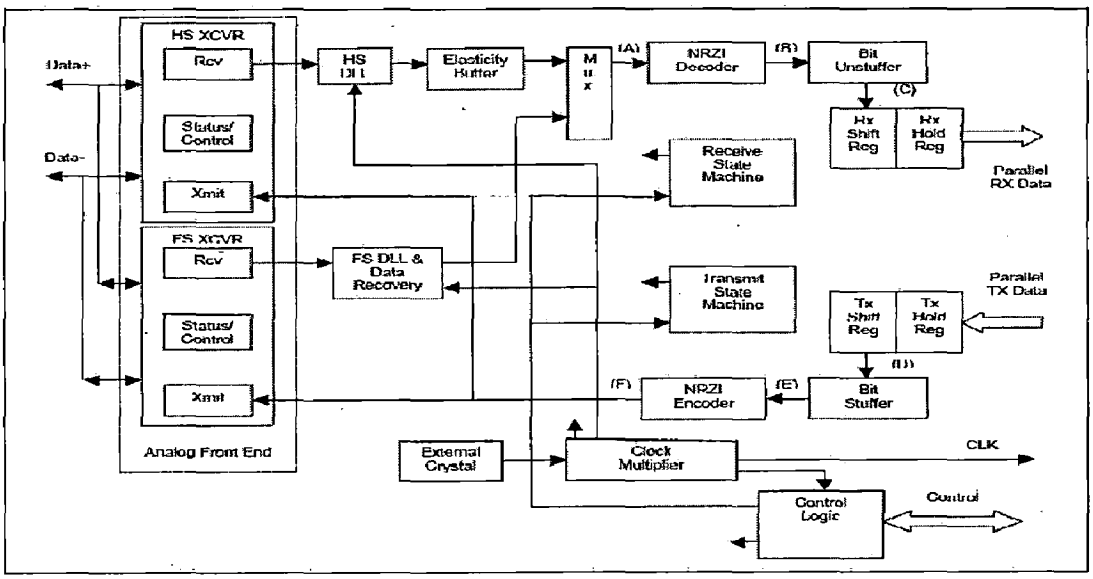


Fig 3.9: UTM Functional Block Diagram

- Supports 480 Mbps "High Speed" (HS)/ 12 Mbps "Full Speed" (FS), FS Only and "Low Speed" (LS) Only 1.5 Mbps serial data transmission rates.
- Utilizes 8-bit parallel interface to transmit and receive USB 2.0 cable data.
- SYNC/EOP generation and checking.
- High Speed and Full Speed operation to support the development of "Dual Mode" devices.
- Data and clock recovery from serial stream on the USB.
- Bit-stuffing/unstuffing, bit stuff error detection.
- Holding registers to stage transmit and receive data.
- Logic to facilitate resume signaling.
- Logic to facilitate wake up and suspend detection.

- Ability to switch between FS and HS terminations/signaling.
- Single parallel data clock output with on-chip PLL to generate higher speed serial data clocks.

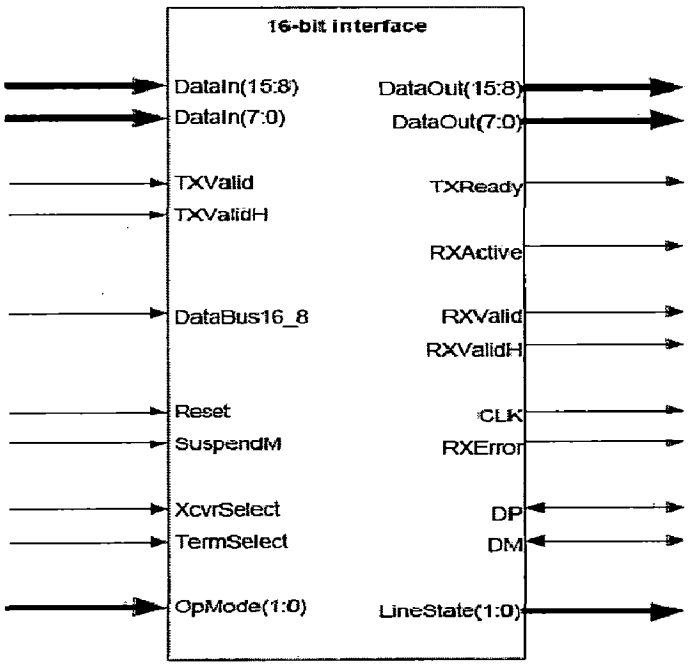


Fig 3.10: UTMI INTERFACE SIGNALS with respect to PHY

3.5.3 System Interface Signals with Description

Table 3.7 UTMI Signals

Name	Direction	Active Level	Description
CLK	Output	Rising-Edge	This output is used for clocking receive and transmit parallel data. 60 MHz HS/FS, with 8-bit interface 30 MHz HS/FS, with 16-bit interface 48 MHz FS Only, with 8-bit interface 6 MHz LS Only, with 8-bit interface
Reset	Input	High	Reset all state machines in the UTM.
XcvrSelect	Input	N/A	This signal selects between the FS and HS transceivers 0: HS transceiver enabled 1: FS transceiver enabled This signal is not provided in FS Only and LS Only transceiver implementations.

TermSelect	Input	N/A	Signal selects between the FS and HS terminations 0: HS termination enabled 1: FS termination enabled This signal is not provided in FS Only and LS Only transceiver implementations.
SuspendM	Input	Low	Places the Macrocell in a mode that draws minimal power from supplies. 0: Macrocell circuitry drawing suspend current 1: Macrocell circuitry drawing normal current
LineState (0-1)	Output	N/A	These signals reflect the current state of the single ended receivers. DM[1] DP[0] Description 0 0 0: SE0 0 1 1: 'J' State 1 0 2: 'K' State 1 1 3: SE1
OpMode (0-1)	Input	N/A	These signals select between various 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
Databus16_8	Input	High	Set for 16- bit data bus.
DP	Bidir	N/A	USB data pin Data+
DM	Bidir	N/A	USB data pin Data-
DataIn0-7	Input	N/A	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.
DataIn8-15	Input	N/A	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.
TXValid	Input	High	Indicates that the DataIn bus is valid. The assertion of TXValid initiates SYNC on the USB. The negation of TXValid initiates EOP on the USB.
TXValidH	Input	High	When DataBus16_8 = 1, this signal indicates that the DataIn (8-15) bus contains valid transmit data.
TXReady	Output	High	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.
DataOut0-7	Output	N/A	8-bit parallel USB data output bus. When DataBus16_8 = 1 this bus transfers the low byte of 16-bit receive data. When DataBus16_8 = 0 all receive data is transferred over this bus.
DataOut8-15	Output	N/A	An 8-bit parallel USB data output bus that transfers the high byte of 16-bit receives data. These signals are only valid when DataBus16_8 = 1.
RXValid	Output	High	Indicates that the DataOut bus has valid data. The Receive Data Holding Register is full and ready to be unloaded. The SIE is expected to latch the DataOut bus on the clock edge.
RXValidH	Output	High	When DataBus16_8 = 1 this signals indicates that the DataOut (8-15) bus is presenting valid receive data.
RXActive	Output	High	Indicates that receive state machine has detected SYNC and is active. RXActive is negated after a Bit Stuff Error or an EOP is detected.
RXError	Output	High	0 Indicates no error. 1 Indicates that a receive error has been detected. If asserted, it will force the negation of RXValid on the next rising edge of CLK

3.5.4 Data Receiving and Transmitting

- RXActive and RXValid are sampled on the rising edge of CLK.
- The receiver is always looking for SYNC.
- The Macrocell asserts RXActive when SYNC is detected (Strip SYNC state).
- The Macrocell negates RXActive when an EOP is detected (Strip EOP state).
- When RXActive is asserted, RXValid will be asserted if the RX Holding Register is full.
- RXValid will be negated if the RX Holding Register was not loaded during the previous byte time.
- This will occur if 8 stuffed bits have been accumulated.
- The SIE must be ready to consume a data byte if RXActive and RXValid are asserted (RX Data state).
- In FS mode, if a bit stuff error is detected then Receive State Machine will negate RXActive and RXValid.

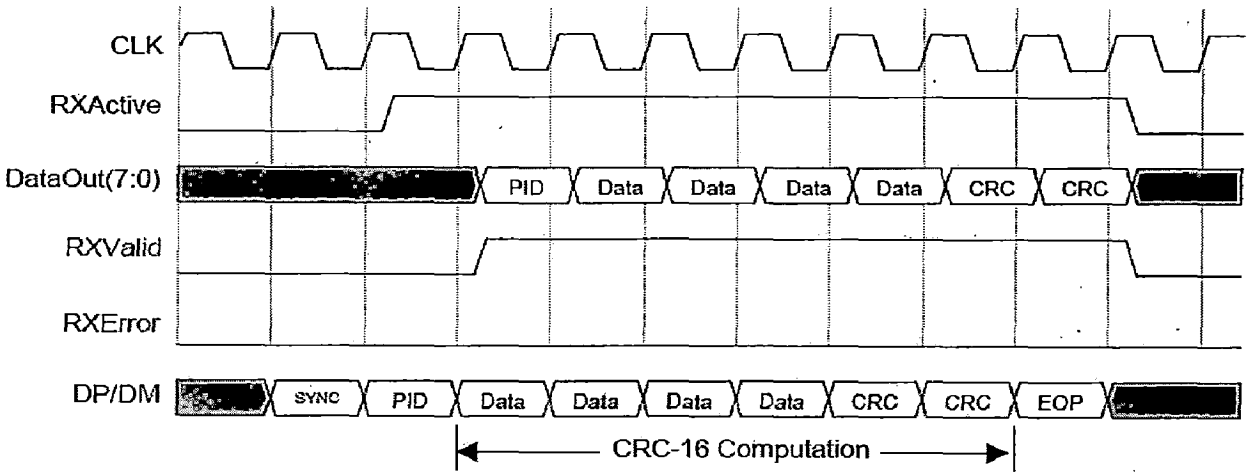


Fig 3.11: Receive Timing for Data Packet (with CRC-16)

Transmit must be asserted to enable any transmissions.

- The SIE asserts TXValid to begin a transmission.
- The SIE negates TXValid to end a transmission.
- After the SIE asserts TXValid it can assume that the transmission has started when it detects TXReady asserted.
- The SIE assumes that the UTM has consumed a data byte if TXReady and TXValid are asserted.
- The SIE must have valid packet information (PID) asserted on the DataIn bus coincident with the assertion of TXValid. Depending on the UTM

implementation, TXReady may be asserted by the Transmit State Machine as soon as one CLK after the assertion of TXValid.

- TXValid and TXReady are sampled on the rising edge of CLK.
- The Transmit State Machine does NOT automatically generate Packet ID's (PIDs) or CRC. When transmitting, the SIE is always expected to present a PID as the first byte of the data stream and if appropriate, CRC as the last bytes of the data stream.
- The SIE must use LineState to verify a Bus Idle condition before asserting TXValid state.

The SIE negates TXValid to complete a packet. Once negated, the Transmit State Machine will never reassert TXReady until after the EOP has been loaded into the Transmit Shift Register. Note that the UTM Transmit State Machine can be ready to start another packet immediately, however the SIE must conform to the minimum inter-packet delays identified in the USB 2.0 Specification.

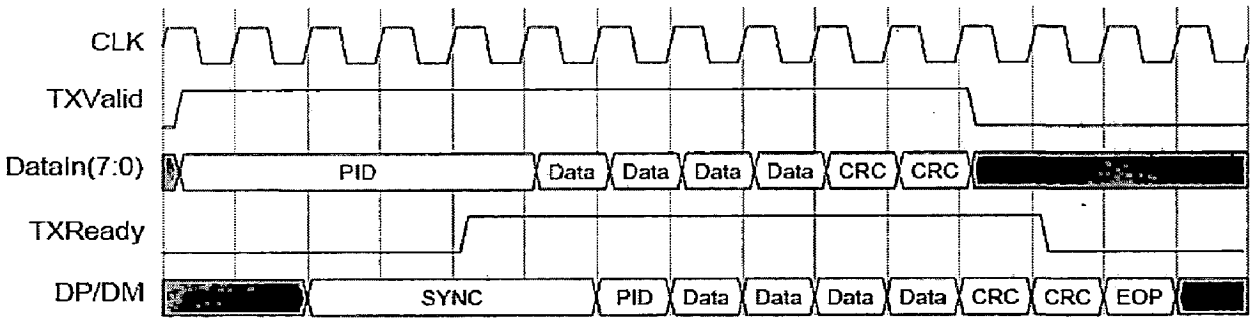


Fig 3.12: Transmit Timing for a Data packet

3.6 UTMI+ Specification

3.6.1 Introduction [4]

Purpose: The existing UTMI specification describes an interface only for USB2.0 peripherals. The intention of this UTMI+ specification is to extend the UTMI specification to standardize the interface for USB2.0 hosts and USB2.0 OTG peripherals. The UTMI+ specification defines and standardizes the interoperability characteristics with existing USB2.0 hosts and peripherals. Any transceiver core that has an interface compliant with UTMI+, has all signals compliant with UTMI. A transceiver core with UTMI+ interface can be used for USB2.0 peripheral, host or On-the-Go device designs that support LS, FS and HS traffic.

Additional signals for UTMI+: USB OTG peripherals have some additional capabilities and therefore some new signals need to be implemented.

1. A USB OTG dual role peripheral needs to be capable to distinguish between a mini-A and mini-B plug.

2. A USB OTG peripheral has to know if Vbus is below or above a certain voltage level.

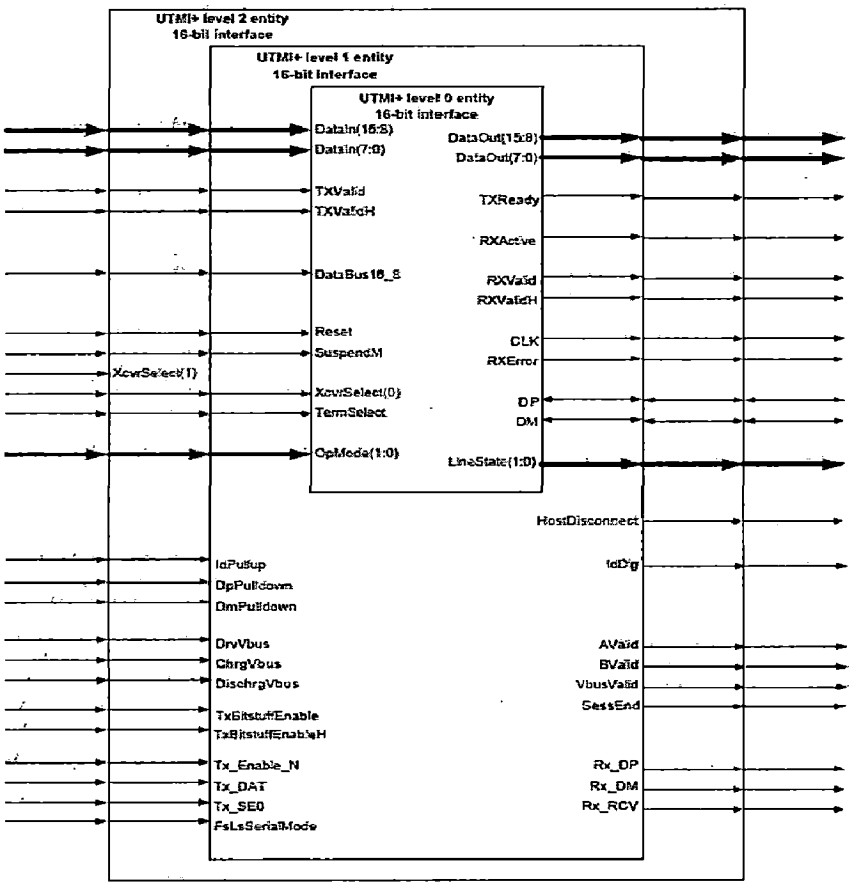


Fig 3.13: UTMI+ Interface with respect to PHY

3. A USB OTG peripheral must be able to drive Vbus and charge or discharge Vbus.

4. A USB OTG dual role peripheral needs to be able to switch the pull-up resistor on DP and the pull-down resistor on both DP and DM.

5. The downstream facing port of a host controller must have 15 k Ohm pull-down resistors on both DP and DM lines.

6. The host controller must be able to detect disconnect of a peripheral. This is possible for a FS peripheral by using LineState.

Therefore an additional signal needs to be implemented. To make the design of the digital SIE easier, this new disconnect signal will be used in both speeds (HS/FS) to indicate if there is a device connected or not.

3.6.2 UTMI + signals other than UTMI

Table 3.9 UTMI+ Signals

NAME	DIRECTION	ACTIVE STATE	DESCRIPTION
The id signal is indicating the state of the ID pin on the USB mini receptacle			
IdPullup	Input	Low	Signal that enables the sampling of the analog Id line. 0b: Sampling of Id pin is disabled. 1b: Sampling of Id pin is enabled.
IdDig	Output		Indicates whether the connected plug is a mini-A or B. This is valid when IdPullup is set to 1b. 0b : connected plug is a mini-A 1b : connected plug is a mini-B
These are the signals which indicate the Link layer about the Strength of the Vbus.			
AValid	Output	High	Indicates if the session for an A-peripheral is valid (0.8V < Vth < 2V). 0b : Vbus < 0.8V 1b : Vbus > 2V
BValid	Output	High	Indicates if the session for a B-peripheral is valid (0.8V < Vth < 4V). 0b : Vbus < 0.8V 1b : Vbus > 4V
VbusValid	Output	High	Indicates if the voltage on Vbus is at a valid level for operation (4.4V < Vth < 4.75V). 0b : Vbus < 4.4V 1b : Vbus > 4.75V
SessEnd	Output	Low	Indicates if the voltage on Vbus (0.2V < Vth < 0.8V). 1b : Vbus < 0.2V 0b : Vbus > 0.8V
When the charge pump is not integrated within the transceiver macrocell then the optional DrvVbus signal may be omitted from the macrocell.			
DrvVbus	Input	High	This signal enables to drive 5V on Vbus 0b : do not drive Vbus 1b : drive 5V on Vbus
ChrgVbu	Input	Low	The signal enables charging Vbus.

s			1b : charge Vbus through a resistor 0b : do not charge Vbus through a resistor
DischrgVbus	Input	Low	The signal enables discharging Vbus. 1b : discharge Vbus through a resistor 0b : do not discharge Vbus through a resistor
These two signals are used to switch on the 15k Ohm pull-down resistors on both DP and DM for a host. For a peripheral both signals should been set to 0b. For a host controller both signals should been set to 1b.			
DpPulldown	Input		0b : Pull-down resistor not connected to DP 1b : Pull-down resistor connected to DP
DmPulldown	Input		0b : Pull-down resistor not connected to DM 1b : Pull-down resistor connected to DM
This signal is used for all types of peripherals connected to it. It is only valid when DpPulldown and DmPulldown are 1b. If DpPulldown and DmPulldown are not 1b then the behavior of HostDisconnect is undefined			
HostDisconnect	Output	Low	If a device is connected, then the value of this signal will be 0b else 1b
These signals are only used when OpMode is set to 11b. While OpMode is set to 11b the automatic generation of SYNC and EOP is disabled. These signals make it also possible to transmit high-speed USB packets while the transceiver is put into OpMode = 11b.			
TxBitstuffEnable	Input	High	0b : Bit stuffing is disabled 1b : Bit stuffing is enabled
TxBitstuffEnableH	Input	High	0b : Bit stuffing is disabled 1b : Bit stuffing is enabled
The FsLsSerialMode signal indicates how the digital core signals the FS and LS packets to the transceiver. The reason to add FsLsSerialMode to the interface is to make it possible to reuse existing FS/LS host controller IP without changing its interface.			
FsLsSerialMode	Input	High	0b: The parallel interface. 1b: FS and LS packets are sent using the serial interface.
Tx_Enable_N	Input		Active low output enable signal.
Tx_DAT	Input		Differential data at D+/D- output

Tx_SE0	Input		Force Single-Ended Zero
Rx_DP	Output		Single-ended receive data, positive terminal. The data is valid if FsLsSerialMode is set to 1b
Rx_DM	Output		Single-ended receive data, negative terminal The data is only valid if FsLsSerialMode is set to 1b
Rx_RCV	Output		Receive data, The data is only valid if FsLsSerialMode is set to 1b

3.7 UTMI+ Low Pin Interface Specification

3.7.1 Introduction [5]

Purpose: ULPI defines a PHY to Link interface of 8 or 12 signals that allows 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.

If a ULPI PHY design is based on an internal UTMI+ core, then that core must implement the following UTMI+ features.

- LineState must accurately reflect D+/D- to within 2-3 clocks.
- Filtering to prevent spurious SE0/SE1 states appearing on LineState due to skew between D+ and D- .
- The PHY must internally block the USB receive path during transmit. The receive path can be unblocked when the EOP is detected internally.
- TXReady must be used for all types of data transmitted, including chirp.

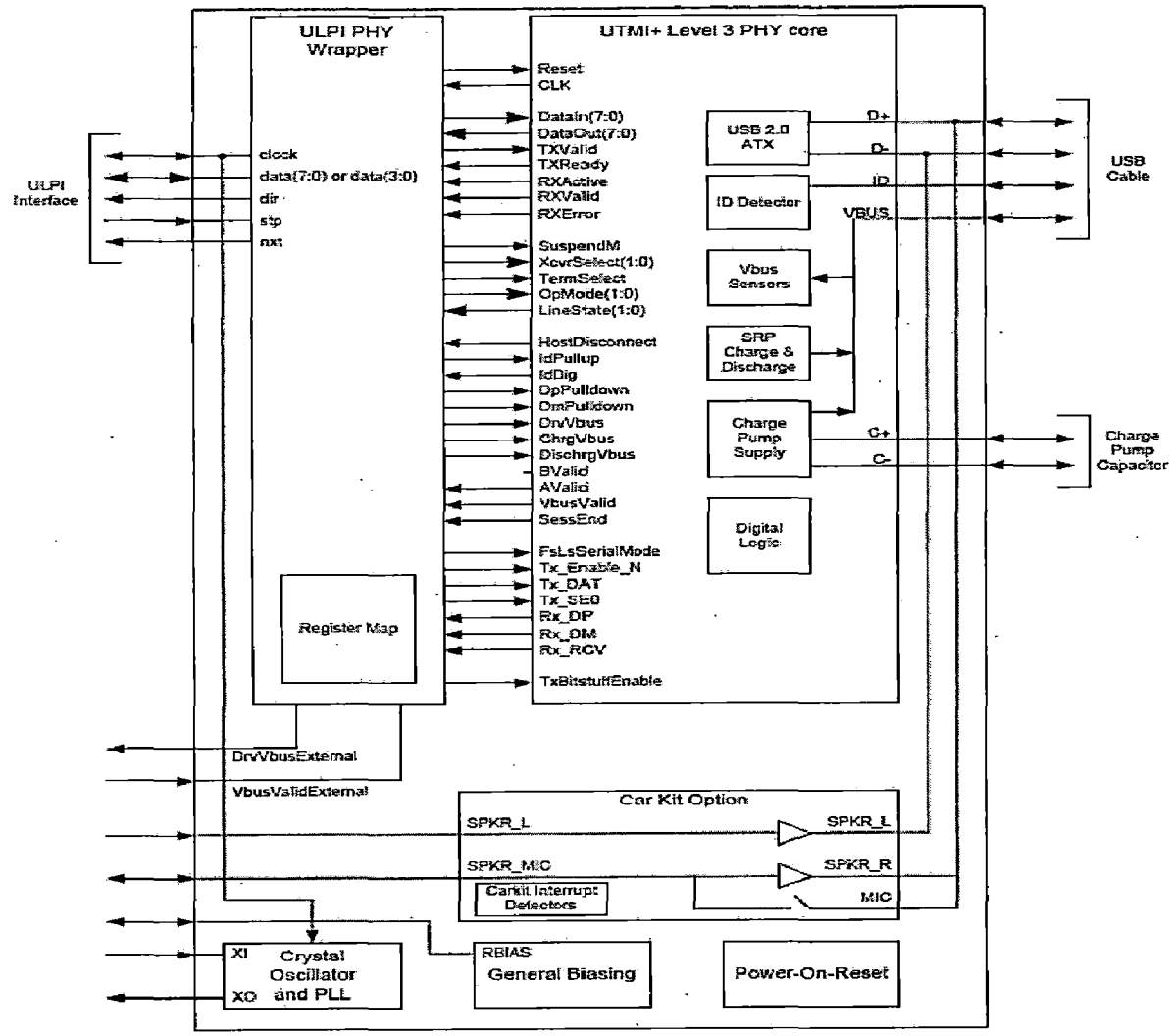


Fig 3.14: ULPI Interface with respect to PHY

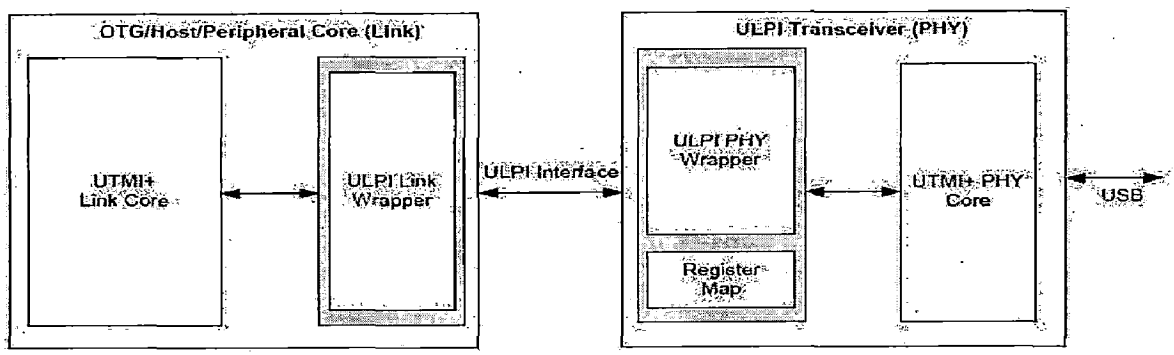


Fig 3.15: ULPI system from UTMI+ with wrappers

3.7.2 Interface signals of ULPI

Table 3.10 ULPI Signals

Signal	Direction	Description
clock	I/O	Interface clock. The PHY must be capable of providing a 60Mhz output clock. All interface signals are synchronous to clock.
data	I/O	Bi-directional data buses, driven low by link during idle. Bus ownership is determined by dir. Link and PHY initiate data transfers by driving a non-zero pattern onto the data bus. Driven 00h by the Link when the ULPI bus is idle. Bus widths allowed are • 8-bit data timed on rising-edge of the clock. • 4-bit data timed on rising and falling edges of the clock.
dir	OUT	Controls the direction of the data bus. When the PHY has data to transfer to the Link, it drives dir high to take ownership of the bus. When the PHY has no data to transfer it drives dir low and monitors the bus for Link activity. The PHY pulls dir high whenever the interface cannot accept data from the link.
stp	IN	If the Link is sending data to the PHY, stp indicates the last byte was on the bus in the previous cycle. If the PHY is sending data to the link, stp forces the PHY to end its transfer.
nxt	OUT	The PHY asserts the signal to throttle the data, except register read command and RX CMD. Identical to RXValid during USB receive, and TXReady during USB transmit. When the Link is sending data to the PHY, nxt indicates when the current byte has been accepted by the PHY. The PHY is not allowed to assert nxt during the first cycle of the TX CMD driven by Link. The link places the next byte on the data bus in the following clock cycle. When the PHY is sending data to the Link, nxt indicates when a new byte is available for the Link to consume.

3.7.3 ULPI Command Bytes [5]

ULPI modifies the original UMTI+ data stream so that it can fit more data types. Redundancy in the PID byte during transmit is overloaded with the ULPI transmit commands (TX CMD). Unused data bytes in the receive stream are overloaded with receive commands (RX CMD) ULPI defines a Transmit Command byte that is sent by the Link and a Receive Command byte that is sent the PHY.

Transmit command Byte (TX CMD): The TXCMD byte consists of a 2-bit command code and a 6-bit payload. The data which is transmitted from Link to PHY is send with the TX CMD byte as the first byte. The PHY will decode the TX CMD and transmit the data to Device in the USB structure only.

Table 3.11 Transmit command (TX CMD) byte format

Byte Name	Command Code data(7:6)	Command Payload data(5:0)	Command Description
Special	00b	000000b (NOOP)	No operation. 00h is the idle value of the data bus. The Link drives NOOP by default.
		XXXXXXb (RSVD)	Reserved Command Space. Values other than those above will give undefined behavior.
Transmit	01b	000000b (NOPID)	Transmit USB data that does not have a PID, such as chirp and resume signaling. The PHY starts transmitting on the USB beginning with the next data byte.
		00XXXXb (PID)	Transmit USB packet. data(3:0) indicates USB packet identifier PID(3:0)
		XXXXXXb (RSVD)	Reserved Command Space. Values other than those above will give undefined behavior.
RegWrite	10b	101111b (EXTW)	Extended register write command. 8-bit address available in the next cycle.
		XXXXXXb (REGW)	Register write command with 6-bit immediate address.
RegRead	11b	101111b (EXTR)	Extended register read command. 8-bit address available in the next cycle.
		XXXXXXb (REGR)	Register read command with 6-bit immediate address.

Receive Command Byte (RX CMD): The Receive Command byte is sent by the PHY to update the Link with LineState, USB receive, disconnect and OTG status information.

Table 3.12 RX CMD Byte format

<table>
<tr><th>data</th><th>Name</th><th>Description and Value</th></tr>
<tr><td>1:0</td><td>LineState</td><td>UTMI+ LineState signals
data(0) = LineState(0)
data(1) = LineState(1)</td></tr>
<tr><td>3:2</td><td>VbusState</td><td>Encoded Vbus Voltage state.
<table>
<tr><td>Value</td><td>Vbus Voltage</td><td>SessEnd</td><td>SessValid</td><td>VbusValid</td></tr>
<tr><td>00</td><td>Vbus</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>01</td><td></td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>10</td><td></td><td>X</td><td>1</td><td>0</td></tr>
<tr><td>11</td><td></td><td>X</td><td>X</td><td>1</td></tr>
</table></td></tr>
<tr><td>5:4</td><td>RxEvent</td><td>Encoded UTMI event signals
<table>
<tr><td>Value</td><td>RxActive</td><td>RxError</td><td>HostDisconnect</td></tr>
<tr><td>00</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>01</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>10</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>11</td><td>X</td><td>X</td><td>1</td></tr>
</table></td></tr>
<tr><td>6</td><td>ID</td><td>Set to the value of IdGnd (UTMI+ IdDig) valid 50ms after IdPullup is set to 1b.</td></tr>
<tr><td>7</td><td>alt_int</td><td>Asserted when a non-USB interrupt occurs. This bit must be set when an unmasked event occurs on any bit in the Carkit interrupt Latch register. The Link must read the Carkit Interrupt Latch register to determine the source of the interrupt.</td></tr>
</table>

When to send an RX CMD: An RX CMD is sent only in Synchronous Mode, and conveys two types of information to the Link. The first is USB receive information. The Second is interrupt events. All information is encoded into the single RX CMD byte.

USB Receive information includes LineState, RXActive and RXError. The PHY gains the ownership of the data bus by asserting dir and waiting 1 cycle as turnaround time and de-assert nxt and drive an RX CMD with respective event observed. After a USB transmit, the PHY must send an RX CMD with LineState indicating EOP of the Link.

Interrupt events include HostDisconnect, Vbus, IdGnd, and alternative sources such as Carkit interrupts. An RX CMD is sent to the Link whenever these events are detected, and the corresponding USB interrupt Enable Rising or USB interrupt Enable Falling registers are set.

3.7.4 Register Operations

ULPI provides a set of registers in the PHY that are accessed by the Link to control PHY functionality. The Link can read or write register bytes, set and clear register bits as needed using the TX CMD byte by RegRead and RegWrite operations.

Immediate RegWrite and Extended RegWrite Operation

For the Immediate RegWrite the Link sends the register write command (8'b10XXXXXX) and waits for nxt to assert. In the cycle after nxt asserts, the link sends the register write data and waits for nxt to assert again. When nxt asserts the second time, the Link asserts stp in the following cycle to complete the operation.

For Extended RegWrite it takes one more cycle as TX CMD is 8'hAF fixed first byte and when nxt is asserted it puts the 8-bit extended address on the data bus and then puts the write data on the bus.

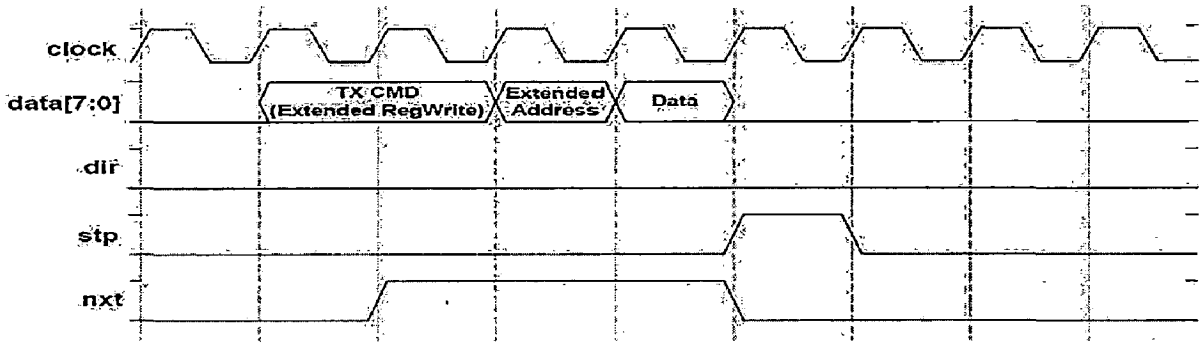


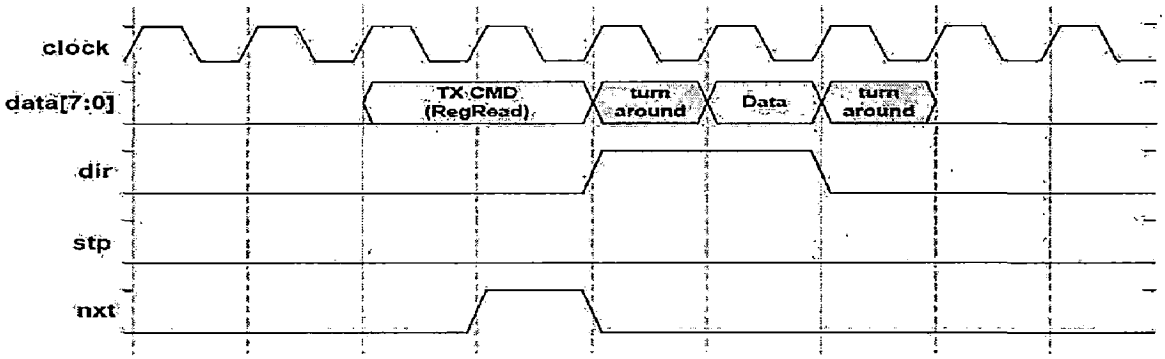
Fig 3.16 Extended RegWrite operation

Immediate RegRead and Extended RegRead Operation

For the Immediate RegRead the Link sends the register read command (8'b11XXXXXX) and waits for nxt to assert. In the cycle after nxt asserts, the PHY will take over the bus and waits for 1 cycle turnaround and then puts the required data

from the Registers and then turns down the dir giving the control to link. Then the Link reads the data on bus continues its operation.

For Extended RegRead it takes one more cycle as TX CMD is 8'hEF fixed first byte and when nxt is asserted it puts the 8-bit extended address on the data bus and then reads the data on the bus a cycle later.



3.17 Immediate RegRead operation

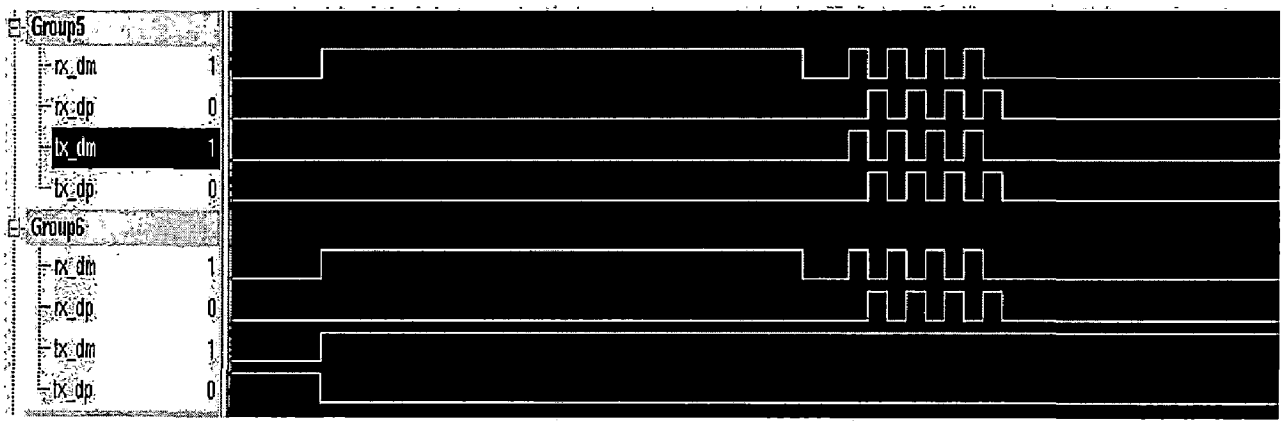
The important Registers used in the Normal operation and the On-The-Go Supplement are Function Control Register, OTG Control Register and USB Interrupt Status Register.

3.8 Simulation Results

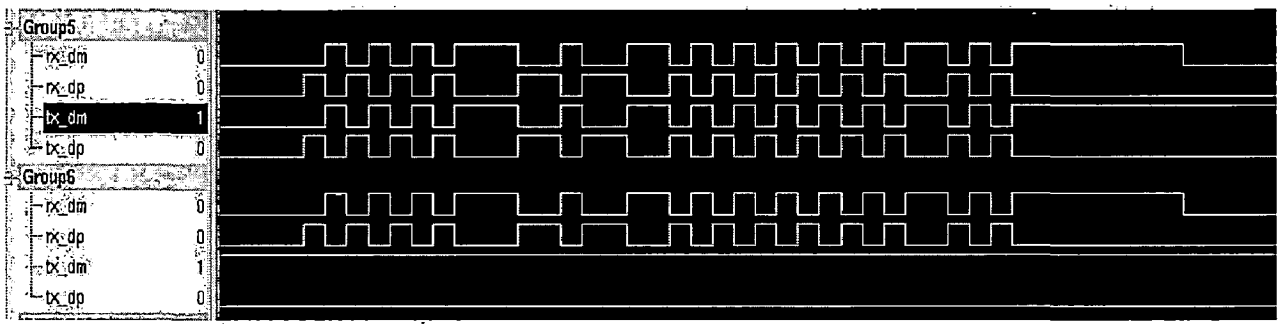
The snapshots of the simulations done for USB 2.0 for three interfaces in several scenarios are shown below.

3.8.1 Serial Interface

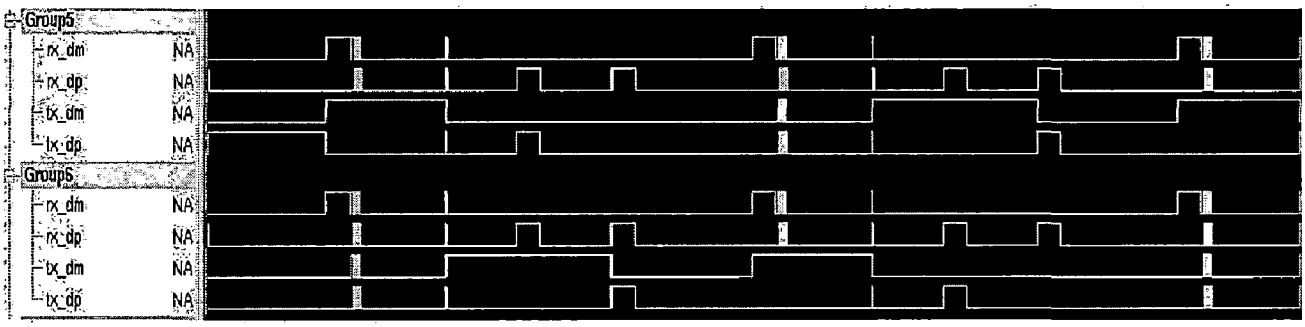
The below snapshot shows the Device Chirp K and Host Chirp KJ Sequence during high-speed reset with successful speed negotiation.



The SOF packets send preceded by the SYNC and followed by EOP.

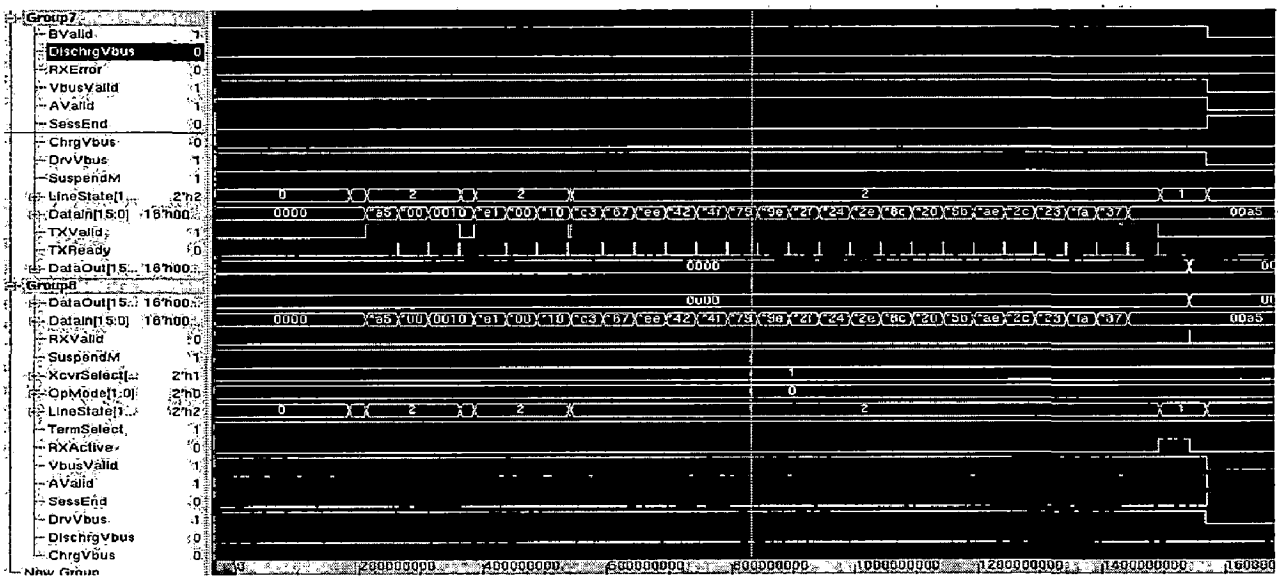


The Serial Interface where the device is getting attached at the beginning and goes into Reset and going into High speed. Then transactions taking place and then the device goes suspended state and into disconnected state and then connects as Host using HNP in High-speed and do the transactions and then give back the original host its state.

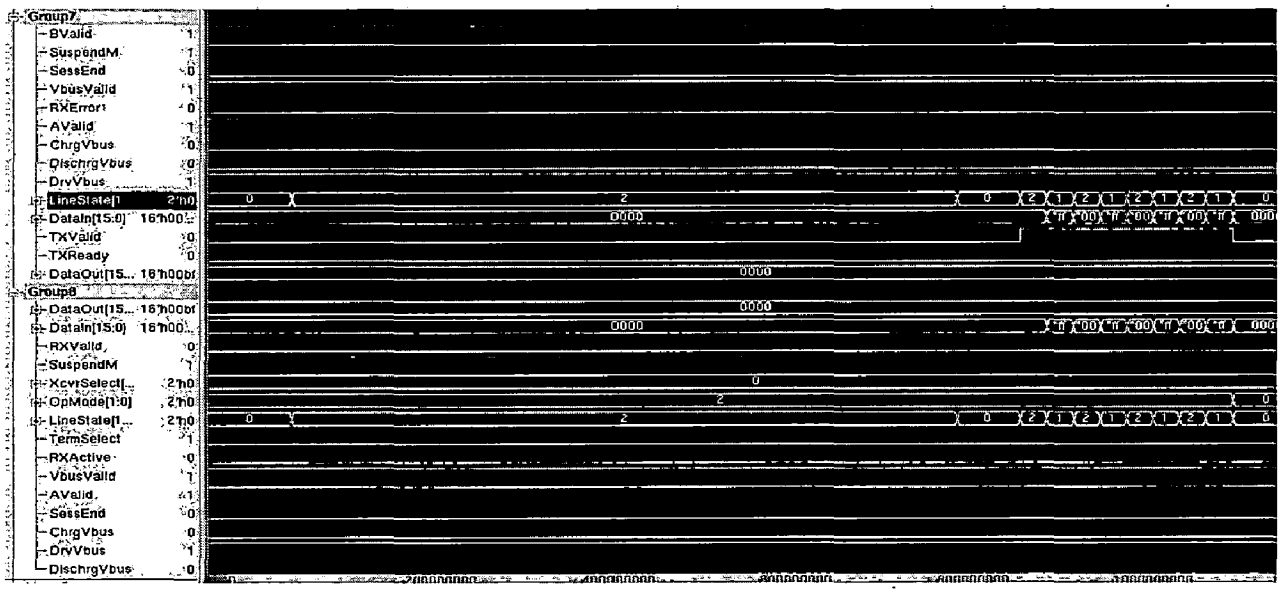


3.8.2 UTMI+ Interface

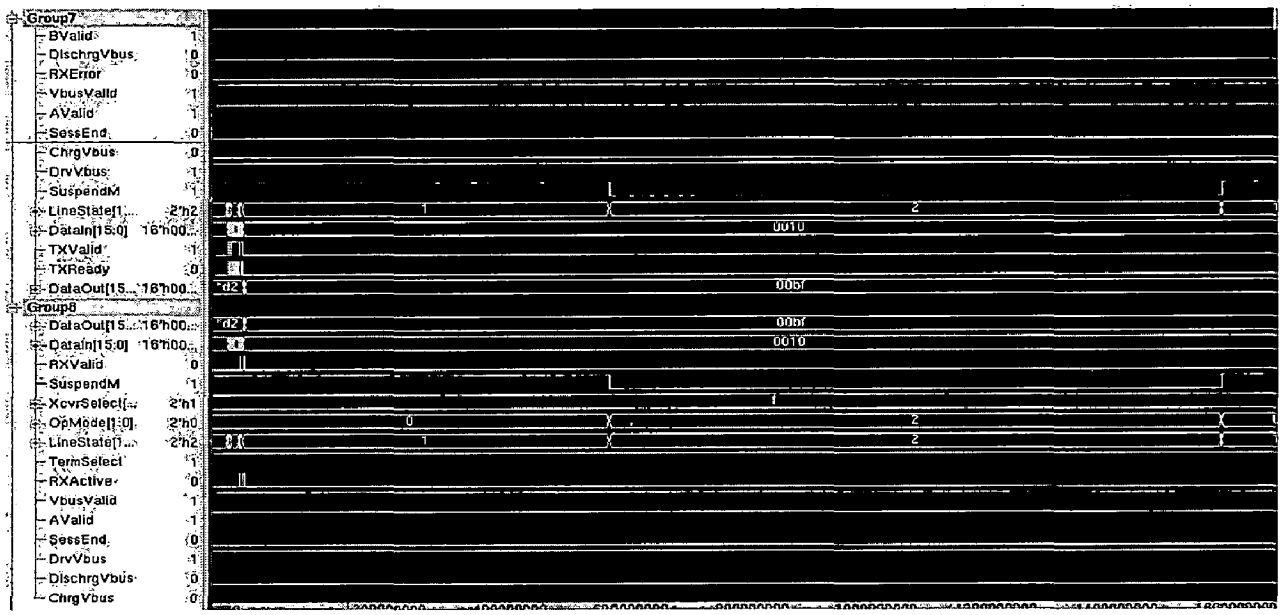
Transactions in the UTMI+ the SOF starting after the reset and then OUT transaction on TX lines and received ACK packet from the PHY on RX Lines.



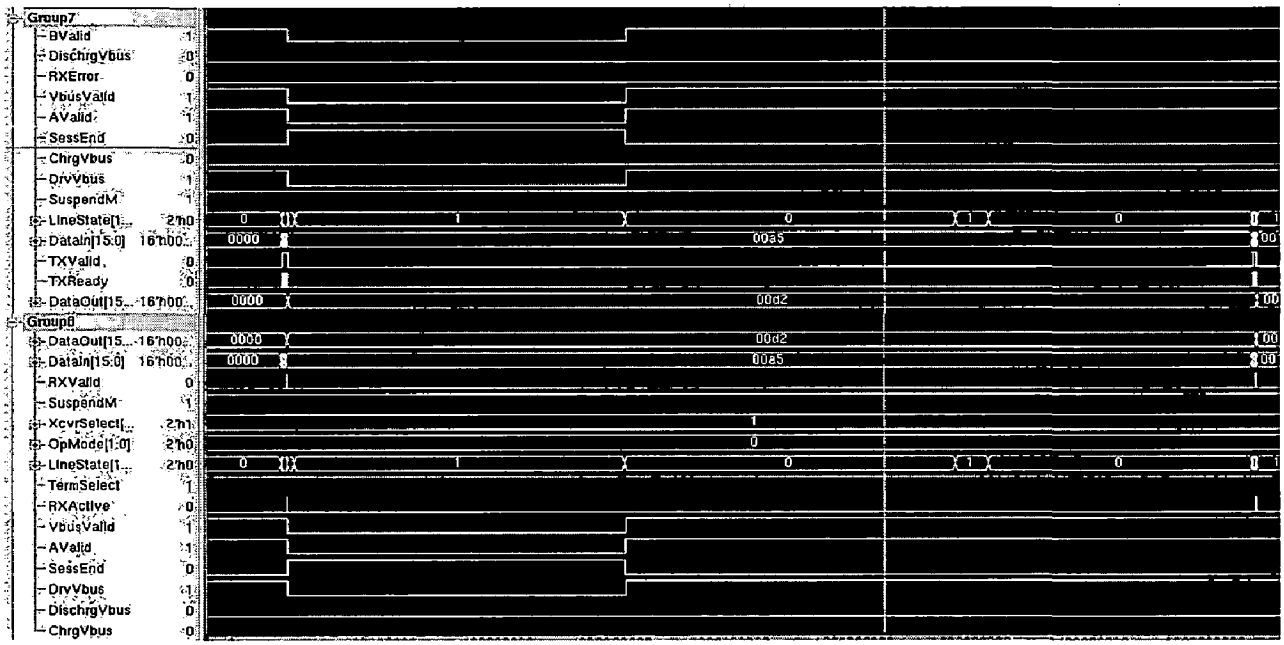
The reset task where the A-device is Link and the B-device is PHY which keeps LineState as 2 for chirp K and Link sends chirp KJ sequence in response on the TX lines.



Suspend and Resume in UTMI+ are seen in the figure which shows SuspendM signal is made Low at the time of suspend and made high after resume.

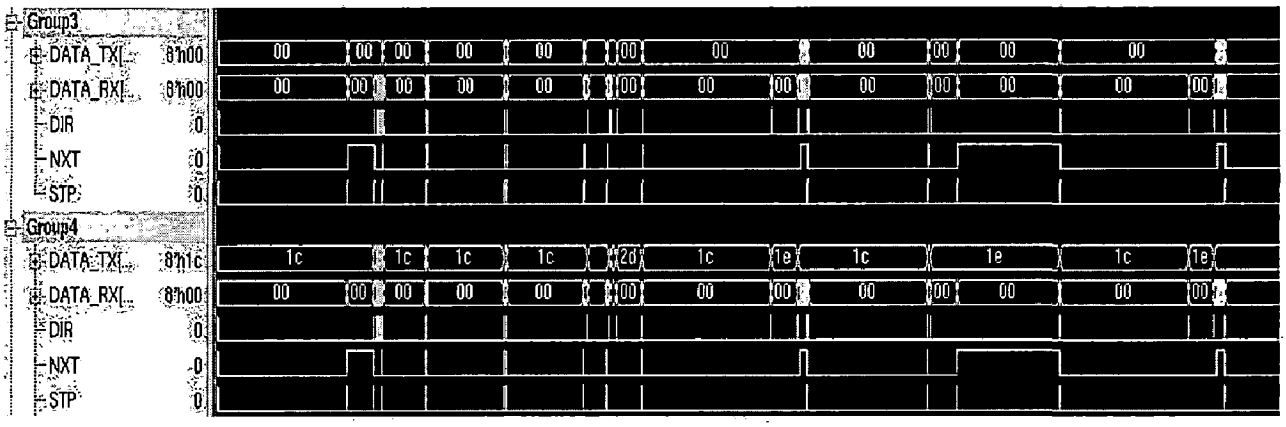


SRP in UTMI+ is shown in the below figure which shows all the 4 stages Initial Condition 1, 2 , data line pulsing and Vbus pulsing with respect to UTMI+ signals.

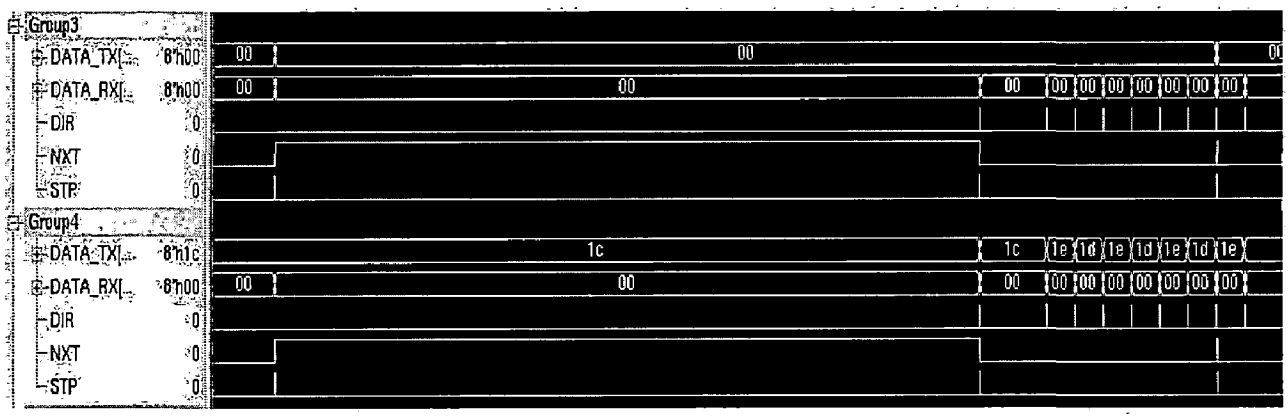


3.8.3 ULPI Interface

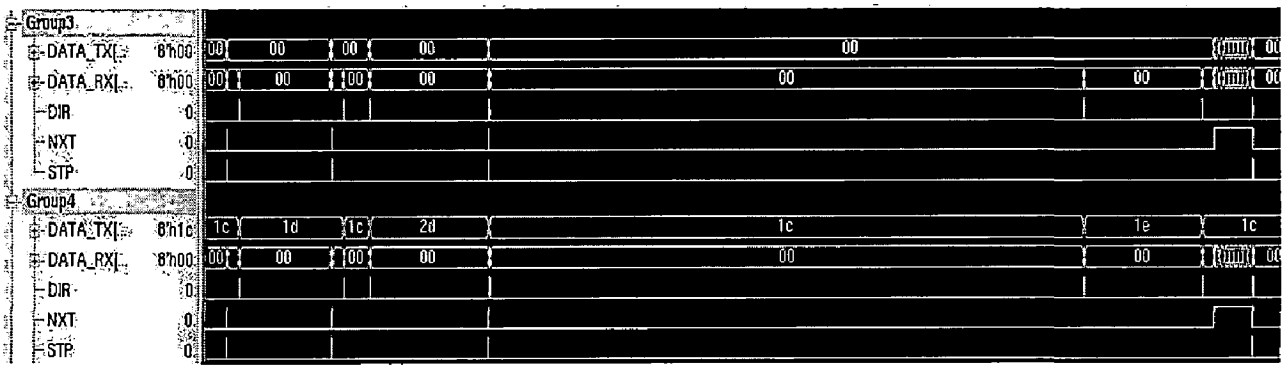
The Transactions in the ULPI Interface can be seen where the PHY sends the RX CMD whenever the change in the LineState in PHY is seen or on completion of Data or ACK packets.



HS Reset in ULPI is seen here the Link is B-device and PHY is the A-device. The Link will be sending the Chirp K for becoming the HS device then PHY after receiving the TX CMD (NOPID) will wait till the end of Chirp and then gives chirp KJ sequence in the form of RX CMD's to the Link so that both will come High-speed after reset.



HNP in ULPI is shown in the figure below where the A-device will end the session and B-device will get disconnected state and then A-device will attach to the B-device as device so the B-device (original) will reset the A-device (original) as Host.



4. WIRELESS USB PHYSICAL LAYER

4.1 General Introduction

Wireless USB is logical evolution of USB. There are several key design areas to be met, they are Leverage the existing USB infrastructure, Preserve the USB model of smart host and simple device, Provide effective power management mechanisms, Provide security, Ease of use and Wire Adapter to upgrade wired USB devices.

4.1.1 Architecture Overview [6]

Wireless USB is a logical bus that supports data exchange between a host device and a wide range of simultaneously accessible peripherals. The attached peripherals share bandwidth through a host-scheduled, TDMA Protocol. The bus allows peripherals to be attached configured, used, and detached while the host and other peripherals are in operation. Security definitions are provided to assure secure associations between hosts and devices, and to assure private communication.

Wireless USB connects USB devices with the USB Host using a 'hub and spoke' model. The Wireless USB host is the 'hub' at the center, and each device sits at the end of a 'spoke' having a point-to-point connection with the host which have a maximum of 127 devices.

4.1.2 Bus Protocol

Wireless USB is a polled, TDMA based protocol, similar to wired USB. The Host controller initiates all data transfers. To increase the efficiency of the physical layer by eliminating costly transitions between sending and receiving, hosts combine multiple token information into a single packet, in that packet, the host indicates the specific time when the appropriate devices should either listen for an OUT data packet, or transmit an IN data packet or handshake. Wireless USB defines new maximum packet sizes for some endpoint types to enhance channel efficiency. Similarly, some new flow control mechanisms are defined to enhance channel efficiency and to allow more power-friendly designs. New mechanisms are defined for isochronous pipes to deal with the lower reliability of the wireless medium.

4.1.3 Robustness

There are several attributes of wireless USB that contribute to its robustness [6]:

- The physical layer is designed for reliable communication and robust error detection and correction.

- Detection of attach and detach and system-level configuration of resources.
- Self-recovery in protocol, using timeouts for lost or corrupted packets.
- Flow control, buffering and retries ensure isochrony and hardware buffer management.

4.1.4 Security

All hosts and all devices must support Wireless USB security. The security mechanisms ensure that both hosts and devices are able to authenticate their communication partner and that communications between host and devices are private. The security mechanisms are based on AES-128/CCM encryption, providing integrity checking as well as encryption. Communications between host and device are encrypted using 'keys' that only the authenticated host and authenticated device possess.

4.1.5 System Configuration

Unlike wired USB, Wireless USB devices 'attach' to a host by sending the host a message at a well defined time. The host and device then authenticate each other using their unique IDs and the appropriate security keys.

4.2 PHY General Description

The Ecma Standard specifies the Ultra Wideband (UWB) physical layer (PHY) for a wireless personal area network (PAN), utilizing the unlicensed 3,100–10,600 MHz frequency band [7], supports data rates of 53.3 Mb/s, 80 Mb/s, 106.7 Mb/s, 160 Mb/s, 200 Mb/s, 320 Mb/s, 400 Mb/s, and 480 Mb/s. Support for transmitting and receiving data rates of 53.3, 106.7, and 200 Mb/s shall be mandatory.

The UWB spectrum is divided into 14 bands, each with a bandwidth of 528 MHz. The first 12 bands are then grouped into 4 band groups consisting of 3 bands. The last two bands are grouped into a fifth band group. A sixth band group is also defined within the spectrum of the first four, consistent with usage within worldwide spectrum regulations. At least one of the band groups shall be supported. The Ecma Standard specifies a Multi Band Orthogonal Frequency Division Modulation (MBOFDM) scheme to transmit information. A total of 110 sub-carriers (100 data carriers and 10 guard carriers) are used per band to transmit the information. In addition, 12 pilot subcarriers allow for coherent detection. Frequency-domain spreading, time-domain

spreading, and forward error correction (FEC) coding are used to vary the data rates. The FEC used is a convolutional code with coding rates of 1/3, 1/2, 5/8 and 3/4.

4.3 MAC General Description

The MAC is a sublayer of the Data Link Layer defined in the OSI basic reference model. The MAC service is provided by means of the MAC service access point (MAC SAP) to a single MAC service client, usually a higher layer protocol or adaptation layer. The MAC sublayer is represented by a device address. The MAC sublayer in turn relies on the service provided by the PHY layer via the PHY service access point (PHY SAP). The MAC protocol applies between MAC sublayer peers.
Device address: Individual MAC sublayers are addressed and are associated with a volatile abbreviated address called a DevAddr. DevAddrs are 16-bit values, generated locally, without central coordination. Consequently, it is possible for a single value to ambiguously identify two or more MAC entities. The MAC addressing scheme includes multicast and broadcast address values. A multicast address identifies a group of MAC entities. The broadcast address identifies all MAC entities.

4.4 Features Assumed from the PHY

A MAC sublayer is associated with a single PHY layer via the PHY SAP. The MAC sublayer requires the following features provided by the PHY:

- Frame transmission in both single frame and burst mode.
- Frame reception for both single frame and burst mode transmission.
- PLCP header error indication for both PHY and MAC header structures.
- Clear channel assessment for estimation of medium activity.
- Range measurement timestamps if MAC range measurement is supported.

Frames are transmitted by the PHY from the source device and delivered to the destination device in identical bit order. The start of a frame refers to the leading edge of the first symbol of the PHY frame at the local antenna and the end of a frame refers to the trailing edge of the last symbol of the PHY frame. Frame transmission and reception are supported by the exchange of parameters between the MAC sublayer and the PHY layer. These parameters allow the MAC sublayer to control, and be informed of, the frame transmission mode, the frame payload data rate and length, the frame preamble, the PHY channel and other PHY-related parameters. In single frame

transmission, the MAC sublayer has full control of frame timing. In burst mode transmission, the MAC sublayer has control of the first frame timing and the PHY provides accurate timing for the remaining frames in the burst.

The MAC service provides:

- Communication between cooperating devices within radio range on a single channel using the PHY.
- A distributed, reservation-based channel access mechanism.
- A prioritized, contention-based channel access mechanism.
- A synchronization facility for coordinated applications.
- Mechanisms for handling mobility and interference situations.
- Device power management by scheduling of frame transmission and reception.
- Secure communication with data authentication and encryption using cryptographic algorithms.
- A mechanism for measuring the distance between two devices.

The architecture of this MAC service is fully distributed. All devices provide all required MAC functions and optional functions as determined by the application. No device acts as a central coordinator.

4.5 PHY Layer Partitioning

The following section describes the PHY services provided to the MAC. The PHY layer consists of two protocol functions:

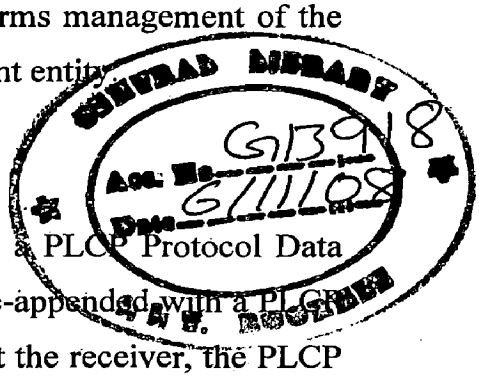
- A PHY convergence function, which adapts the capabilities of the physical medium dependent (PMD) device to the PHY service. This function is supported by the physical layer convergence protocol (PLCP), which defines a method of mapping the PLCP service data units (PSDU) into a framing format suitable for sending and receiving user data and management information between two or more stations using the associated PMD device.
- A PMD device whose function defines the characteristics and method of transmitting and receiving data through a wireless medium between two or more stations, each using the Ecma PHY.

PHY Function: The PHY contains three functional entities: the PMD function, the PHY convergence function, and the layer management function. The PHY service is provided to the MAC through the PHY service primitives.

PLCP sublayer: In order to allow the MAC to operate with minimum dependence on the PMD sublayer, the PHY convergence sublayer is defined. This function simplifies the PHY service interface to the MAC services.

PMD sublayer: The PMD sublayer provides a means to send and receive data between two or more stations.

PHY layer management entity (PLME): The PLME performs management of the local PHY functions in conjunction with the MAC management entity.



4.6 PLCP Sublayer

This section provides a method for converting a PSDU into a PLCP Protocol Data Unit (PPDU). During the transmission, the PSDU shall be pre-appended with a PLCP preamble and a PLCP header in order to create the PPDU. At the receiver, the PLCP preamble and PLCP header serve as aids in the demodulation, decoding, and delivery of the PSDU.

4.6.1 PPDU [7]

The format for the PPDU, which is composed of three components: the PLCP preamble, the PLCP header, and the PSDU. The components are listed in the order of transmission. The PLCP preamble is the first component of the PPDU and can be further decomposed into a packet/frame synchronization sequence, and a channel estimation sequence. The goal of the PLCP preamble is to aid the receiver in timing synchronization, carrier-offset recovery, and channel estimation.

The PLCP header is the second component of the PPDU. The goal of this component is to convey necessary information about both the PHY and the MAC to aid in decoding of the PSDU at the receiver. The PLCP header can be further decomposed into a PHY header, MAC header, Header Check Sequence (HCS), Tail bits, and Reed-Solomon parity bits. Tail bits are added between the PHY header and MAC header, HCS and Reed-Solomon parity bits and at the end of the PLCP header in order to return the convolutional encoder to the "zero state". The Reed-Solomon parity bits are added in order to improve the robustness of the PLCP header.

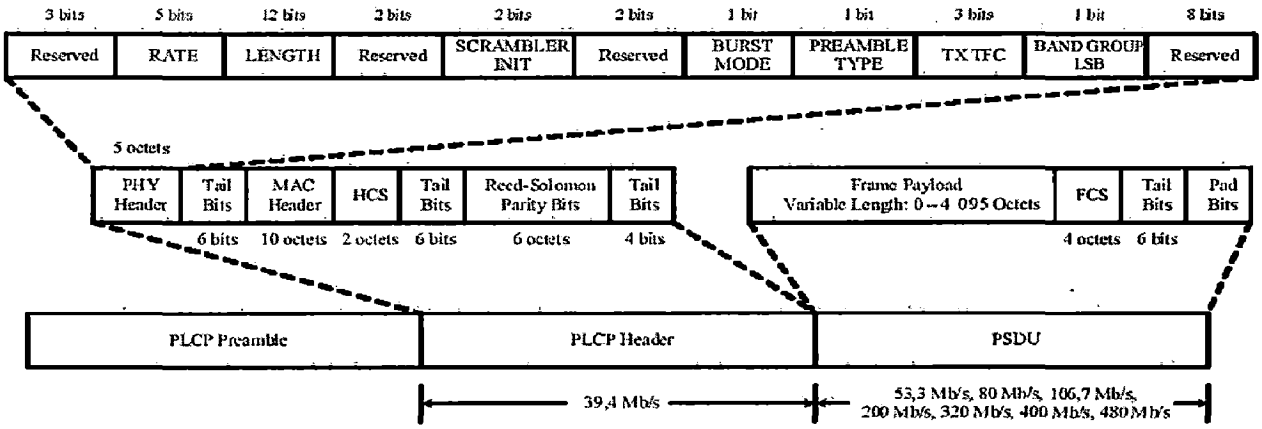


Fig 4.1: Standard PPDU Structure

The PSDU is the last component of the PPDU. This component is formed by concatenating the frame payload with the frame check sequence (FCS), tail bits, and finally pad bits, which are inserted in order to align the data stream on the boundary of the symbol interleaver.

When transmitting the packet, the PLCP preamble is sent first, followed by the PLCP header, and finally by the PSDU. The PLCP header is a codeword of a systematic Reed-Solomon code, appended with tail bits as explained above. The systematic part of the PLCP header is always sent at a data rate of 39.4 Mb/s. The PSDU is sent at the desired data rate of 53.3 Mb/s, 80 Mb/s, 106.7 Mb/s, 160 Mb/s, 200 Mb/s, 320 Mb/s, 400 Mb/s or 480 Mb/s. The least-significant bit (LSB) of an octet shall be the first bit transmitted.

4.6.2 PLCP Preamble

A PLCP preamble shall be added prior to the PLCP header to aid the receiver in timing synchronization, carrier-offset recovery, and channel estimation. There are two types of preambles: standard PLCP preamble and a burst PLCP preamble. A unique preamble sequence shall be assigned to each time-frequency code (TFC). The preamble is defined to be a real baseband signal, which shall be inserted into the real portion of the complex baseband signal. The PLCP preamble consists of two portions: a time-domain portion (packet / frame synchronization sequence) followed by a frequency-domain portion (channel estimation sequence). The burst preamble shall only be used in the burst mode when a burst of packets is transmitted, separated by a minimum inter-frame separation time (pMIFS). For data rates of 200 Mb/s and lower, all the packets in the burst shall use the Standard PLCP preamble.

However, for data rates higher than 200 Mb/s, the first packet shall use the Standard PLCP preamble, while the remaining packets may use either the Standard PLCP

preamble or the burst PLCP preamble. Support for transmission and reception of burst PLCP preamble is mandatory for all supported data rates above 200Mbps. The preamble type (PT) bit in the PHY header describes the type of preamble that shall be used in the next packet.

4.6.3 PLCP Header

A PLCP header shall be added after the PLCP preamble to convey information about both the PHY and the MAC that is needed at the receiver in order to successfully decode the PSDU.

The scrambled and Reed-Solomon encoded PLCP header shall be formed as defined in:

1. Format the PHY header based on information provided by the MAC.
2. Calculate the HCS value (2 octets) over the combined PHY and MAC headers.
3. The resulting HCS value is appended to the MAC header. The resulting combination (MAC Header + HCS) is scrambled.
4. Apply a shortened Reed-Solomon code (23, 17) to the concatenation of the PHY header (5 octets), scrambled MAC header and HCS (12 octets).
5. Insert 6 tail bits after the PHY header, 6 tail bits after the scrambled MAC header and HCS, and append the 6 parity octets and 4 tail bits at the end to form the scrambled and Reed-Solomon encoded PLCP header.

The resulting scrambled and Reed-Solomon encoded PLCP header is encoded, using a R=1/3, K=7 convolutional code, interleaved using a bit interleaver, mapped onto a QPSK constellation, and finally, the resulting complex values are loaded onto the data subcarriers for the IDFT in order to create the baseband signal.

(1) PHY Header

The PHY header contains information about the data rate of the MAC frame body, the length of the frame payload (which does not include the FCS), the seed identifier for the data scrambler, and information about the next packet – whether it is being sent in burst mode and whether it employs a burst preamble or not.

The PHY header field shall be composed of 40 bits, numbered from 0 to 39. Bits 3-7 shall encode the RATE field, which conveys the information about the type of modulation, the coding rate, and the spreading factor used to transmit the MAC frame body. Bits 8-19 shall encode the LENGTH field, with the least-significant bit being transmitted first. Bits 22-23 shall encode the seed value for the initial state of the scrambler, which is used to synchronize the descrambler of the receiver. Bit 26 shall

encode whether or not the packet is being transmitted in burst mode. Bit 27 shall encode the preamble type (Standard or burst preamble) used in the next packet if in burst mode. Bits 28-30 shall be used to indicate the lower 3 LSBs of the TFC (T1 - T3) used at the transmitter. Bit 31 shall be used to indicate the LSB of the band group used at the transmitter. Bit 34 shall be used to indicate the MSB of the TFC (T4) used at the transmitter. All other bits which are not defined in this sectione shall be understood to be reserved for future use and shall be set to ZERO.

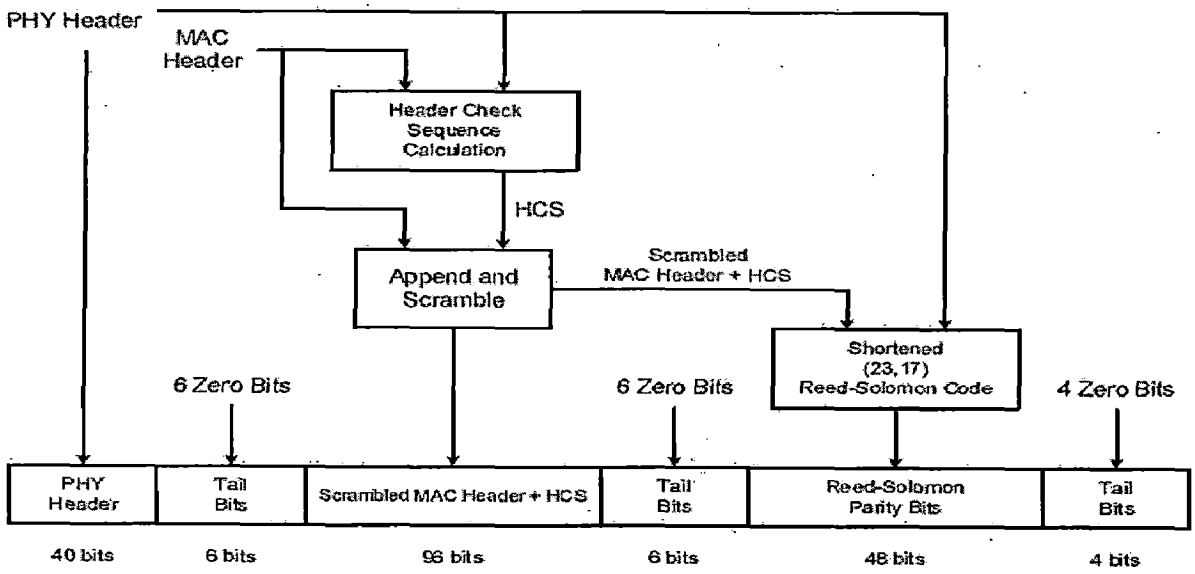


Fig 4.2 Scrambled and RS Encoded PLCP Header

(2) RS Outer Code for PLCP Header

The PLCP header shall use a systematic (23, 17) Reed-Solomon outer code to improve upon the robustness of the $R = 1/3$, $K = 7$ inner convolutional code. The Reed-Solomon code is defined over GF (2^8) with a primitive polynomial

$p(z) = z^8 + z^4 + z^3 + z^2 + 1$, where α is the root of the polynomial $p(z)$.

The generator polynomial is obtained by shortening a systematic (255, 249) Reed-Solomon code, which is specified by the generator polynomial

$g(x) = \prod_{i=1}^{6}(x - \alpha^i) = x^6 + 126x^5 + 4x^4 + 158x^3 + 58x^2 + 49x + 117$, where $g(x)$ is the generator polynomial over F, $x \in F$ and the coefficients are given in decimal notation.

(3) Header Check Sequence

The combination of PHY header and the MAC header shall be protected with a 2 octet CCITT CRC-16 header check sequence (HCS). The CCITT CRC-16 HCS shall be the ones complement of the remainder generated by the modulo-2 division of the combined PHY and MAC headers by the polynomial: $x^{16} + x^{12} + x^5 + 1$.

4.6.4 PSDU

The PSDU is the last major component of the PPDU and shall be constructed as:

1. Form the non-scrambled PSDU by appending the frame payload with the 4 octet FCS, six tail bits, and a sufficient number of pad bits in order to ensure that the PSDU is aligned on the interleaver boundary.
2. The resulting combination is scrambled according to 4.6.5
3. The six tail bits in the PSDU shall be produced by replacing the six scrambled zero bits with six non-scrambled zero bits.

The resulting scrambled PSDU is encoded, using a R = 1/3, K = 7 convolutional code and punctured to achieve the appropriate coding rate, interleaved using a bit interleaver, mapped onto either a QPSK or DCM constellation, and finally, the resulting complex values are loaded onto the data subcarriers of the OFDM symbol in order to create the real or complex baseband signal, depending on the desired data rate.

Pad bits: Pad bits shall be appended after the 6 tail bits prior to scrambling and encoding in order to ensure that the resulting PSDU is aligned with the boundaries of the bit interleaver.

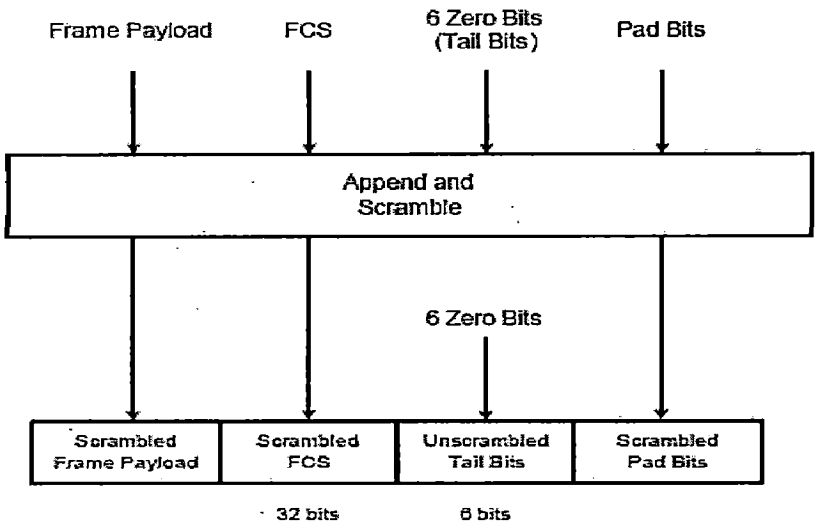


Fig 4.3 Scrambled PSDU

4.6.5 Data Scrambler

A side-stream scrambler shall be used to whiten only portions of the PLCP header, i.e., the MAC header and HCS, and the entire PSDU. In addition, the scrambler shall be initialized to a seed value specified by the MAC at the beginning of the MAC header and then re-initialized to the same seed value at the beginning of the PSDU.

The polynomial generator, $g(D)$, for the pseudo-random binary sequence (PRBS) generator shall be:

$g(D) = 1 + D^{14} + D^{15}$, where D is a single bit delay element.

Using this generator polynomial, the corresponding PRBS, $x[n]$, is generated as

$x[n] = x[n - 14] \oplus x[n - 15]$, $n = 0, 1, 2, \ldots$ where "⊕" denotes modulo-2 addition.

4.6.6 Tail Bits

The tail bit fields are required to return the convolutional encoder to the "zero state". This procedure improves the error probability of the convolutional decoder, which relies on the future bits when decoding the message stream. The tail bit fields after the PHY header and the HCS shall consist of six non-scrambled zeros, and the tail bit field after the Reed- Solomon parity bit field shall be a punctured tail bit sequence consisting of four non-scrambled zeros. The tail bit field following the scrambled frame check sequence shall be produced by replacing the six scrambled zero bits with six non-scrambled zero bits.

4.6.7 Convolutional Encoder

The convolutional encoder shall use the rate R = 1/3 code with generator polynomials, g0 = 133_8, g1 = 165_8, and g2 = 171_8. The bit denoted as "A" shall be the first bit generated by the encoder, followed by the bit denoted as "B", and finally, by the bit denoted as "C". Additional coding rates are derived from the "mother" rate R = 1/3 convolutional code by employing "puncturing". Puncturing is a procedure for omitting some of the encoded bits at the transmitter (thus reducing the number of transmitted bits and increasing the coding rate) and inserting a dummy "zero" metric into the decoder at the receiver in place of the omitted bits. In each of these cases, the tables shall be filled in with encoder output bits from left to right. For the last block of bits, the process shall be stopped at the point at which encoder output bits are exhausted, and the puncturing pattern applied to the partially filled block. The PLCP header shall be encoded using a coding rate of R = 1/3. The encoder shall start from the all-zero state. After the encoding process for the PLCP header has been completed, the encoder shall be reset to the all-zero state before the encoding starts for the PSDU; in other words, the encoding of the PSDU shall also start from the all-zero state. The PSDU shall be encoded using the appropriate coding rate of R = 1/3, 1/2, 5/8, or 3/4.

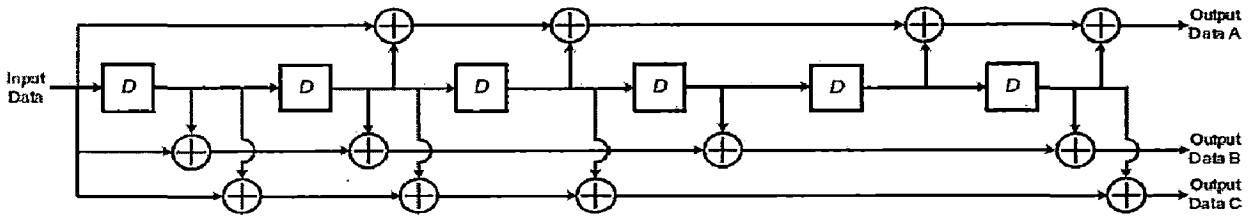


Fig 4.4: Convolutional encoder: rate R = 1/3, constraint length K = 7

Decoding is done by the Viterbi algorithm.

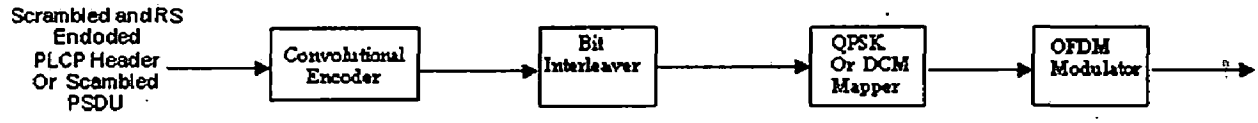


Fig 4.5 Encoding process for the scrambled, Reed-Solomon encoded PLCP header or Scrambled PSDU

4.6.8 Bit interleaving

The coded and padded bit stream shall be interleaved prior to modulation to provide robustness against burst errors. The bit interleaving operation is performed in three distinct stages:

(1) Symbol interleaving, which permutes the bits across 6 consecutive OFDM symbols, enables the PHY to exploit frequency diversity within a band group.

(2) Intra-symbol tone interleaving, which permutes the bits across the data subcarriers within an OFDM symbol, exploits frequency diversity across subcarriers and provides robustness against narrow-band interferers.

(3) Intra-symbol cyclic shifts, which cyclically shift the bits in successive OFDM symbols by deterministic amounts, enables modes that employ time-domain spreading and the fixed frequency interleaving (FFI) modes to better exploit frequency diversity.

Fig 4.6 Various Stages of Bit interleaver

4.6.9 Constellation mapping

This section describes the techniques for mapping the coded and interleaved binary data sequence onto a complex constellation. For data rates 200 Mb/s and lower the binary data shall be mapped onto a QPSK constellation. For data rates 320 Mb/s and above the binary data shall be mapped onto a multi-dimensional constellation using a dual-carrier modulation (DCM) technique.

(1) QPSK

The coded and interleaved binary serial input data, $b[i]$ where $i = 0, 1, 2, \ldots$, shall be divided into groups of two bits and converted into a complex number representing one of the four QPSK constellation points. The conversion shall be performed according to the Gray-coded constellation mapping, with the input bit, $b[2k]$ where $k = 0, 1, 2, \ldots$, being the earliest of the two in the stream. The output values, $d[k]$ where $k = 0, 1, 2, \ldots$, are formed by multiplying $(2\times b[2k]-1) + j(2\times b[2k+1]-1)$ value by a normalization factor of *KMOD*, as described in the following equation:

$d[k] = KMOD \times [(2\times b[2k]-1) + j(2\times b[2k+1]-1)]$, where $k = 0, 1, 2, \ldots$,

The normalization factor $KMOD = 1/\sqrt{2}$ for a QPSK constellation.

(2) Dual-carrier modulation (DCM)

The coded and interleaved binary serial input data, $b[i]$ where $i = 0, 1, 2, \ldots$ shall be divided into groups of 200 bits and converted into 100 complex numbers using a technique called dual-carrier modulation. The conversion shall be performed as follows:

- The 200 coded bits are grouped into 50 groups of 4 bits. Each group is represented as $(b[g(k)], b[g(k)+1], b[g(k) + 50], b[g(k) + 51])$, where $k \in [0, 49]$ and

$$g(k) = \begin{cases} 2k & k \in [0,24] \\ 2k + 50 & k \in [25,49] \end{cases}.$$

- Each group of 4 bits $(b[g(k)], b[g(k)+1], b[g(k) + 50], b[g(k) + 51])$ shall be mapped onto a four-dimensional constellation, as defined in Figure 25, and converted into two complex numbers $(d[k], d[k + 50])$.
- The complex numbers shall be normalized using a normalization factor *KMOD*.

The normalization factor $KMOD = 1/\sqrt{10}$ is used for the dual-carrier modulation.

4.6.10 OFDM modulation [8]

The Ecma Standard specifies a Multi Band Orthogonal Frequency Division Modulation (MBOFDM) scheme to transmit information. A 128-pt (I)FFT is used to create the ODFM symbols at a fixed rate of 242.42ns irrespective of requested data-rate. Each OFDM symbol has a 37-sample zero value guard appended to the end of the symbol giving a total OFDM symbol time (termed interval time) of 312.5ns. Each OFDM symbol is made from 100 data sub-carriers, 12 pilot subcarriers, 10 guard sub-

carriers (copy of the outer 5 data subcarriers to each symbol) and 6 null-valued pilots (5 high frequency and the DC term).

Data subcarriers

The mapping between the stream of complex values and the data subcarriers is dependent on the portion of the PPDU and the data rate. The instantaneous data-rates available are 53.3, 80, 106.7, 160, 200, 320, 400 and 480 Mbit/sec formed by a mixture of various puncturing, time and frequency spreading (diversity). For data rates of 200 Mbit/sec and below, a Time-Domain Spreading (TDS) scheme is employed to send an OFDM symbol on a carrier in one channel of the selected BG and then to send the same OFDM symbol. For data rates of 53.3, 80 Mbit/sec (and the PLCP Header), a Frequency-Domain Spreading (FDS) scheme is employed to QPSK map a complex-value onto 2 IFFT sub-carriers thus improve performance in Frequency Selective Fading.

Guard subcarriers

For each OFDM symbol, starting with the channel estimation sequence within the PLCP preamble, there shall be ten subcarriers, 5 on each edge of the occupied frequency band, allocated as guard subcarriers. The relationship between the power levels of the guard subcarriers and that of the data subcarriers shall be implementation dependent. This relationship shall remain constant within a packet, i.e., from the start of the channel estimation sequence to the end of the packet. The 10 guard subcarriers are located on either edge of the OFDM symbol; at logical frequency subcarriers -61, -60, ..., -57, and 57, 58, ..., 61.

Pilot subcarriers

In all of the OFDM symbols following the PLCP preamble, twelve of the subcarriers shall be dedicated to pilot signals in order to allow for coherent detection and to provide robustness against frequency offsets and phase noise. These pilot signals shall be placed in logical frequency subcarriers -55, -45, -35, -25, -15, -5, 5, 15, 25, 35, 45, and 55.

5. SIMULATION OF WIRELESS USB PROTOCOL PHYSICAL LAYER

5.1 Simulation Environment

The Simulation is done mainly for 2 components of PLCP Protocol Data Unit (PPDU) they PLCP Header and PHY Service Data Unit (PSDU). The PLCP header and PSDU data are transmitted as per the standard Ecma, apart from the Multiband OFDM implementation every other aspect of the Physical layer is verified as mentioned in the specification. The receiver part is also implemented and verified at every stage. Results obtained for different SNR values are shown in results section. The final verification is done to check the required sequence of the FCS decoded in the receiver.

5.2 Design Flow of Simulation

5.2.1 Transmitter

PLCP Header

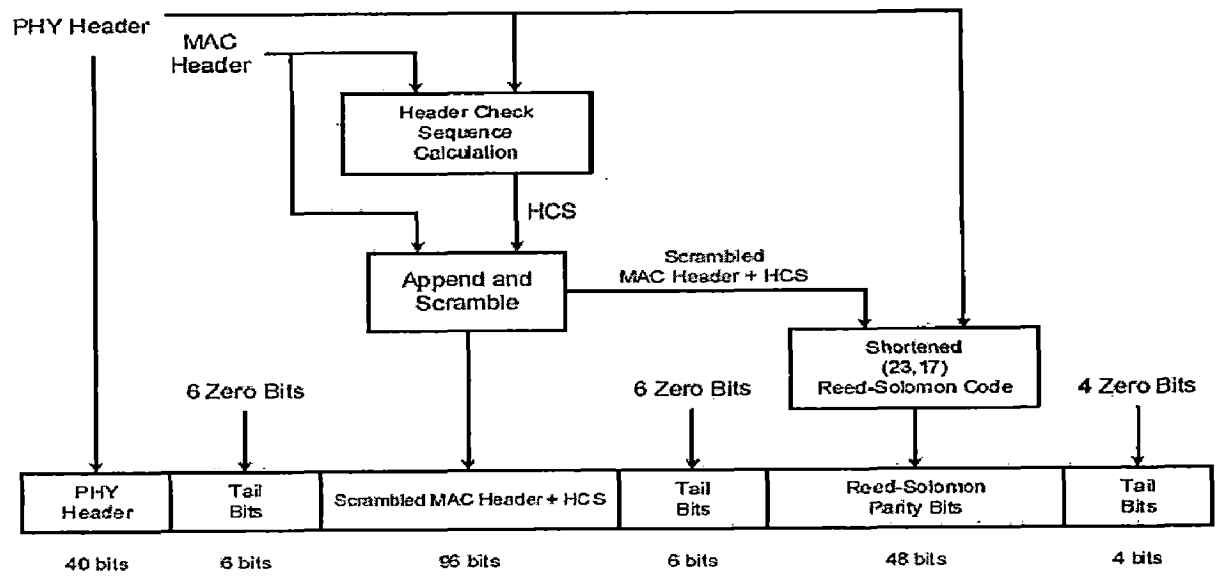


Fig 5.1 Scrambled and RS Encoded PLCP Header

Steps:

1) Getting the input PHY Header of 40 bits and MAC Header of 80 bits.
2) Calculating the 16 bit Header Check Sequence for the Combined PHY and MAC Header with 2 octet CCITT CRC-16 polynomial $x^{16}+x^{12}+x^5+1$.
3) Appending the obtained HCS to the MAC Header.
4) Scrambling the resultant value using the data scrambler

$x[n] = x[n-14] \oplus x[n-15]$, $n = 0, 1, 2, \ldots$ where "⊕" denotes modulo-2 addition.

Calculating the RS Parity bits for the Scrambled MAC Header combined with HCS and PHY Header.

RS Encoder: Reed-Solomon code over GF (2^8) with a primitive polynomial p (z) = z8 + z4 + z3 + z2 + 1, where α is the root of the polynomial p (z) is used. The shortened (255,249) RS code is used for calculating the parity bits. Shortening operation pre-appends 232 zero elements to the incoming 17 octet message as follows

message = 0, i = 17, ..., 248, where the 17 octet message is formed by concatenating the 5 octets from the PHY header to the 12 octets from the scrambled MAC header and HCS.

5) Adding appropriate tail bits like 6 zero tail bits after PHY Header and 6 zero tail bits after Appended and Scrambled MAC Header with HCS and 4 zero tail bits at the end of the RS parity bits.

Steps 1 to 6 are specific to PLCP Header and PSDU have only a few steps, after which both the PLCP Header and PSDU will have a common implementation of Convolutional Encoder, Bit Interleaving, QPSK mapper and OFDM modulation before transmitting.

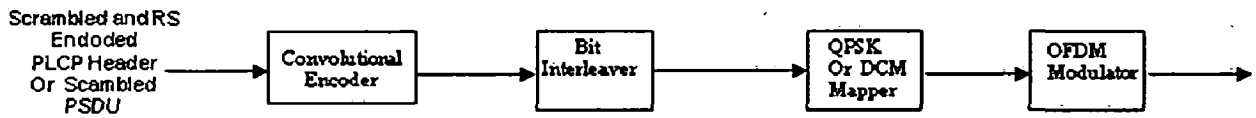


Fig 5.2 Encoding process of scrambled and RS Encoded PLCP Header or Scrambled PSDU

6) Convolutional Encoder of length = 7 and Rate =1/3 with the trellis polynomial formed with coefficients [133 165 171].
7) Bit Interleaving has 3 stages internally:
   - Inter-Symbol Interleaver which permutes the bits across 6 consecutive OFDM symbols, enables the PHY to exploit frequency diversity within a band group with the following equation

$$a_s[i] = a\left[\left\lfloor \frac{i}{N_{CBPS}} \right\rfloor + \left(\frac{6}{N_{TDS}}\right) \times mod(i, N_{CBPS})\right]$$

Where a[i] is the previous sequence from the Convolutional encoder

And $a_S[i]$ is the interleaved sequence after stage 1.

- Intra-symbol tone interleaving, which permutes the bits across the data subcarriers within an OFDM symbol, exploits frequency diversity across subcarriers and provides robustness against narrow-band interferers with the following equation

$$a_T[i] = a_S\left[\left\lfloor\frac{j}{N_{Tint}}\right\rfloor + 10 \times mod(j, N_{Tint})\right]$$

where $a_S[i]$ is the input to the Tone interleaver and $a_T[i]$ is the output of intra-symbol tone interleaving.

- Intra-symbol cyclic shifts, which cyclically shift the bits in successive OFDM symbols by deterministic amounts, enables modes that employ time-domain spreading and the fixed frequency interleaving (FFI) modes to better exploit frequency diversity with the following equation

$$b[i] = a_T[m(i) \times N_{CBPS} + mod(i + m(i) \times N_{CYC}, N_{CBPS})]$$

Where b[i] is the output of the cyclic shifter and $a_T[i]$ is the input to the cyclic shift.

8) QPSK mapper first converts the binary values to two different columns and then transposes them and append element wise to form symbols, and again they are converted into complex QPSK values and send as input to OFDM modulator.

9) OFDM modulation is done by implementing the 50 point IFFT which are obtained from the QPSK mapper. As the part of the channel noise White noise is added to check with conditions that occur in wireless medium.

PSDU

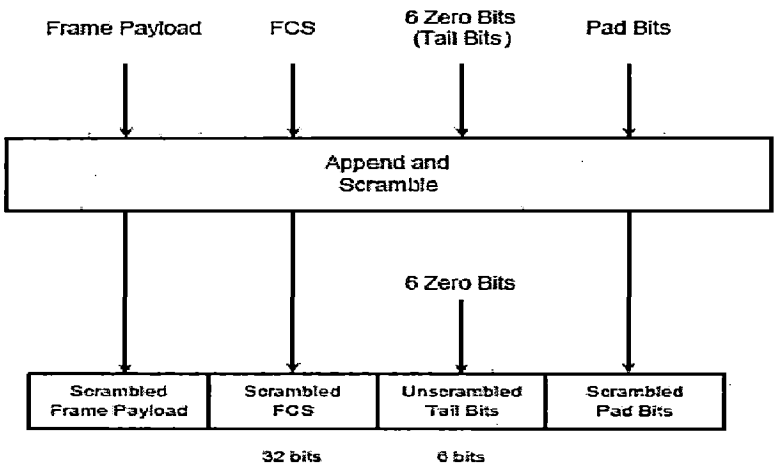


Fig 5.3: Scrambled PSDU

Steps:

1) Append the Frame Payload which is the same data as PHY level data of USB 2.0 and FCS calculated using the following Standard generator polynomial of degree 32:
   $G(x)=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$
2) 6 zero bits and the Pad bits which are calculated from the payload bytes are appended to make the PSDU align with boundary of OFDM symbols.
3) Scramble the sequence obtained as per data scrambler defined above step 4.
4) The resultant scrambled data of Frame Payload, FCS and Pad bits are taken from scrambled output but the 6 zero bits appended are again appended to scrambled data to form scrambled PSDU.

The obtained scrambled PSDU follow the same process from steps 7 to 10 as above.

5.2.2 Receiver

At the Receiver the reverse steps are followed.

PLCP Header

First the PLCP Header is transmitted so when it is received, the receiver demodulates the OFDM symbols by using FFT into QPSK symbols and the obtained symbols are converted to the serialized data is obtained. The obtained serialized data is deinterleaved and then decoded with Viterbi decoder. The obtained data is compared with the data which is send to the Convolutional encoder. Then the data obtained is RS decoded and then compared with the data which is send to the RS encoder. After that the data is descrambled with the same seed to obtain the PHY and MAC Header along with the HCS.

PSDU

For PSDU the data received is demodulated with FFT into QPSK symbols and the obtained symbols are converted to the serialized data is obtained. The obtained serialized data is deinterleaved and then decoded with Viterbi decoder. The length of the Frame Payload is calculated from the PHY header obtained in the above process and the data and FCS are separated from PSDU. Thus separated data is processed with the FCS calculation by appending the 32 bit zeros and calculating the FCS over the actual Frame payload appended with FCS calculated in transmitter. As per the specification of Wireless USB protocol the result should be a particular standard

sequence. The sequence is [1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1].

5.3 Results

The results of the simulation with various SNR values are shown below:

1. SNR = 2

   Comparison is wrong, not received correct demodulated PLCP data

   Comparison is wrong, not received correct demodulated PSDU data

   Comparison is wrong, deinterleaved wrong PLCP data

   Comparison is wrong, deinterleaved wrong PSDU Data

   Comparison is wrong, required PLCP data not recovered by viterbi decoder

   Comparison is wrong, required PSDU data not recovered by viterbi decoder

   Comparison is correct, required PLCP data headers

   Comparison is correct, required descrambled PLCP data

   Comparison is wrong, required descrambled PSDU data not obtained

   Comparison is correct, required FCS decoded, Frame payload received is correct

The simulation result is not constant all the time and varies for SNR = 2.

2. SNR = 5

   Comparison is wrong, not received correct demodulated PLCP data

   Comparison is wrong, not received correct demodulated PSDU data

   Comparison is wrong, deinterleaved wrong PLCP data

   Comparison is wrong, deinterleaved wrong PSDU Data

   Comparison is correct, required PLCP data recovered by viterbi decoder

   Comparison is correct, required PSDU data recovered by viterbi decoder

   Comparison is correct, required PLCP data headers

   Comparison is correct, required descrambled PLCP data

   Comparison is correct, required descrambled PSDU data

   Comparison is correct, required FCS decoded, Frame payload received is correct

3. SNR = 15

   Comparison is correct, received demodulated PLCP data

   Comparison is correct, received demodulated PSDU data

Comparison is correct, required PLCP data deinterleaved

Comparison is correct, required deinterleaved PSDU data

Comparison is correct, required PLCP data recovered by viterbi decoder

Comparison is correct, required PSDU data recovered by viterbi decoder

Comparison is correct, required PLCP data headers

Comparison is correct, required descrambled PLCP data

Comparison is correct, required descrambled PSDU data

Comparison is correct, required FCS decoded, Frame payload received is correct

5.4 MATLAB Code

The code for Wireless USB Physical layer Protocol in MATLAB is given below.

```
% PLCP HEADER AND PSDU
clc;
% PHY Header
PHYH = [0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0];

% TAIL BITS
TAIL1 = [0 0 0 0 0 0];

% MACH HEADER
%MACH = zeros (1,80);
MACH = [ 1 1 0 0 1 0 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1
0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 ];

% HCS CALCULATION
% GENERATOR POLYNOMIAL OF POWER 16
gx=[1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1];

% MAC Header Appended to PHY Header for calculation of Header Check
% Sequence
MACHPHYH = [ PHYH MACH ];

% APPENDING 16 zeros for calculating the HCS.
MACHPHYH_ZERO =[MACHPHYH zeros(1,16)];

% The first 16 bits of the input bit sequence are complemented.
hcs = bitcmp(MACHPHYH_ZERO(1:16),1);

% Mathematically, the CRC value corresponding to a given frame
% is defined by the fllowing procedure.
% The bit sequence is divided by G(x), producing a remainder
% R(x) of degree <= 15.
for i=1:length(MACHPHYH)
  nm=[hcs MACHPHYH_ZERO(i+16)];
  if nm(1)==1
    hcs=xor(nm(2:17),gx(2:17));
```

```
    else
      hcs=nm(2:17);
    end
end

% Taking Absolute value to complement
hcs = abs(hcs);

%Bit Complementing to get the actual HCS Value.
hcs = bitcmp(hcs,1);

% APPEND THE OBTAINED HCS VALUE TO MACH HEADER
APPEND_SCRAMBLE_BFR = [ MACH hcs ];

% SCRAMBLER
% TO SCRAMBLE THE APPENDED DATA.
% poly g[D] = 1+D^14+D^15;
g = [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1];

% FIRST SEED IS TAKEN AND INITIAL VALUE IS TAKEN
s =[0 0];

% FINDING THE SCRAMBLED DATA IN ANOTHER METHOD
% THE SEED VALUE WHICH IS USED TO SCRAMBLE THE DATA
seed_value = [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1];

% DATA SCRAMBLING
for i=1:size(APPEND_SCRAMBLE_BFR,2)
  % XORing of bit X15 and bit X14
  xor_out= bitxor(seed_value(15), seed_value(14));
  % randomized data value
  APPEND_SCRAMBLE_AFTER(i)= bitxor(xor_out, APPEND_SCRAMBLE_BFR(i));
  % new seed value
  seed_value=[xor_out seed_value(1:14)];
end

%APPEND_SCRAMBLE_AFTER = randomized_data;

% TAIL BITS TO BE APPENDED.
TAIL2 = [ 0 0 0 0 0 0];

% RS ENCODER
% Number of bits per symbol
 m = 8;
% THE TOTAL BYTES AFTER THE ENCODER
 n = 255;
% NUMBER OF BYTES INPUT TO ENCODER
 k = 249;

% BITS INSERTED TO THE ACTUAL DATA TO GET THE TOTAL OF 249 BYTES AS INPUT
% TO ENCODER
Insert_bits = zeros(1,1856);

% DATA APPENDING TO GET THE INPUT.
DATA_RS = [ Insert_bits PHYH APPEND_SCRAMBLE_AFTER ];

%LENGTH OF DATA IN BYTES
length(DATA_RS)/8;
```

```
% CONVERTING TO BYTES
DATA_RS = reshape(DATA_RS,8,length(DATA_RS)/8);

% BYTES IN COLUMN MATRIX
DATA_RS = DATA_RS';

% CONVERTING TO CLOUMN MATRIX WITH ROWS OF BITS
DATA_RS = bi2de(DATA_RS,'left-msb');

Tx = 1;
if Tx==1
   % TAKE THE TRANSPOOSE OF THE VECTOR TO PERFORM THE RS ENCODING
    TRAN_DATA_RS=[DATA_RS'];
   % Genero el vector de Galois, el polinomio generador del codigo y, por
   % último codifico los símbolos con el algoritmo de Reed-Salomon:
   msg = gf([TRAN_DATA_RS],m) ;
   if n==k
      codeRS = msg;
   elseif n~=k
      codeRS = rsenc(msg,n,k);
   end
   DATA_AFR_RS = codeRS.x;
elseif Tx==0
   yk = DATABFR3;
   msg = gf([yk],m);
   if n==k
      codeRS = yk;
   elseif n~=k
      codeRS = rsdec(msg',n,k);
      codeRS = codeRS.x;
   end
   DATA_AFR_RS = codeRS(1:end-1);
end

%Rounding off the decimal values(BYTE)
DATA_AFR_RS = double(DATA_AFR_RS);

%Converting back to binary form from the decimal(BYTE)
DATA_AFR_RS = de2bi(DATA_AFR_RS,8,'left-msb');

%Reshaping of the bits after performing RS Encoding
DATA_AFR_RS = reshape(DATA_AFR_RS', 1, length(DATA_AFR_RS)*8);

% RS ENCODER PARITY BITS CALCULATED
RSPBITS = DATA_AFR_RS(1993:2040);

% TAIL ADDED AT THE END TO KEEP FROM THE INTERFERENCE OF NEXT SYMBOL
TAIL3 = [0 0 0 0];

%PLCP DATA READY FOR CONVOLUTIONAL ENCODER
PLCP_DATA = [ PHYH TAIL1 APPEND_SCRAMBLE_AFTER TAIL2 RSPBITS TAIL3];


% AFTER SENDING THE PLCP DATA, PSDU DATA ISA GIVEN TO CONVOLUTIONAL
DECODER
% FRAME PAYLOAD WHICH IS COMPATIBLE WITH USB2.0 PROTOCOL AFTER
CALCULATING
```

```
% CRC
FRAME_PAYLOAD = [ 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];

% DATA TO APPEND AT THE END OF THE PAYLOAD TO GET RIGTH SHIFT
FCS1 = zeros(1,32);

% GENERATOR POLYNOMIAL OF POWER 32 TO GET THE 32 BIT FRAME CHECK
SEQUENCE
gx = [1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 1];

% APPENEDED VALUE TO CALCULATE THE FCS
TOTAL_BFR_FCS = [ FRAME_PAYLOAD FCS1 ];

% COMPLEMENTING THE FIRST 32 BITS BEFORE CALCULATING THE FCS
FCS = bitcmp(TOTAL_BFR_FCS(1:32),1);

%  The bit sequence is divided by G(x), producing a remainder
%  R(x) of degree <= 31.
for i=1:length(FRAME_PAYLOAD)
   nm = [FCS TOTAL_BFR_FCS(i+32)];
   if nm(1)==1
      FCS = xor(nm(2:33),gx(2:33));
   else
      FCS = nm(2:33);
   end
end

% TAKING ABSOLUTE VALUE BEFORE COMPLEMENTING
FCS = abs(FCS);

% COMPLEMENT VALUE OBTAINED TO GET THE ACTUAL FCS VALUE.
FCS_COMP = bitcmp(FCS,1);

% TAIL BITS
TAIL = [0 0 0 0 0 0];

% TOTAL INFORMATION BITS PER ^ SYSMBOLS
NIBP6S = 200;

% PAD BITS FOR ALAIGNING DATA TO RQUIRED LENGTH
PAD_BITS = zeros(1,(NIBP6S*ceil( (8*(length(FRAME_PAYLOAD)/8) + 38)/NIBP6S) -
(8*(length(FRAME_PAYLOAD)/8) + 38)));

% APPENDED PSDU DATA BEFORE SCRAMBLING
APP_DATA_BFR_SCR = [FRAME_PAYLOAD FCS_COMP TAIL PAD_BITS];

% THE SEED VALUE WHICH IS USED TO SCRAMBLE THE DATA
seed_value = [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1];

% data randomization
for i=1:size(APP_DATA_BFR_SCR,2)
   % XORing of bit X15 and bit X14
   xor_out= bitxor(seed_value(15), seed_value(14));
   %randomized data value
   APP_DATA_AFR_SCR(i)= bitxor(xor_out, APP_DATA_BFR_SCR(i));
   %new seed value
   seed_value=[xor_out seed_value(1:14)];
end
```

```
% SCRAMBLED FRAME PAYLOAD DATA
SCR_FRAME_PAYLOAD = APP_DATA_AFR_SCR(1:length(FRAME_PAYLOAD));

% SCRAMBLED FCS CALCULATED
SCR_FCS = APP_DATA_AFR_SCR(length(FRAME_PAYLOAD)+1 :
length(FRAME_PAYLOAD)+ 32);

% SCRAMBLED PAD BITS
SCR_PAD_BITS = APP_DATA_AFR_SCR(length(FRAME_PAYLOAD)+ 39 : end);

% MAKING SURE THAT TAIL BITS ARE ADDED AT CORRECT PLACES
% SCRAMBLED PSDU DATA TO CONVOLUTIONAL ENCODER
PSDU_DATA = [SCR_FRAME_PAYLOAD SCR_FCS TAIL SCR_PAD_BITS];


% COMMON FOR PLCP & PSDU
% CONVLUTIONAL ENCODER k = 7 R = 1/3
Conslen =7;

% Polynomials with which the convolutional Encoder is constructed.
GenPol = [ 133 165 171 ];

% Trellis obtained from the length and generator polynomial
Tr = poly2trellis(Conslen, GenPol);

% PLCP CNVOLUTIOANLLY ENCODED DATA
PLCP_CONV_DATA = convenc(PLCP_DATA, Tr);

% PSDU CNVOLUTIOANLLY ENCODED DATA
PSDU_CONV_DATA = convenc(PSDU_DATA, Tr);

% INTERLEAVER(data): interleave all encoded data with a block size
% corresponding to the number of coded bits per the allocated subchannels
% per OFDM symbol (Ncbps)
Ncbp6s = 600;
Ncbps = 100;
s= ceil( 2/2 );

% SYMBOL INTERLEAVER
for k = 0:Ncbp6s-1
   mk = 6*mod(k,Ncbps)+floor(k/Ncbps);
   firstPerm_interleaved_plcp_data(k+1) = PLCP_CONV_DATA(mk+1);
   firstPerm_interleaved_psdu_data(k+1) = PSDU_CONV_DATA(mk+1);
end
clear k;
clear mk;

% INTRA SYMBOL TONE INTRELEAVING
NTint = 10;
for l = 1 : Ncbp6s/Ncbps
   % second permutation according to eqn. 72
   for k=0:Ncbps-1
      jk=floor(k/10)+ 10 * mod(k,10) ;
      Secondperm_interleaved_plcp_data(((l-1)*Ncbps)+ k+1)=firstPerm_interleaved_plcp_data(((l-
1)*Ncbps)+ jk+1);
      Secondperm_interleaved_psdu_data(((l-1)*Ncbps)+ k+1)=firstPerm_interleaved_psdu_data(((l-
1)*Ncbps)+ jk+1);
```

```
  end
end

% INTRA -SYMBOL CYCLIC INTERLEAVER
Ncyc = 33;
for p = 0 : Ncbp6s-1
  kp = floor(p/Ncbps)*Ncbps + mod((p + floor(p/Ncbps)*Ncyc), Ncbps);
  Final_plcp_data (p+1) = Secondperm_interleaved_plcp_data(kp+1);
  Final_psdu_data (p+1) = Secondperm_interleaved_psdu_data(kp+1);
end


% MAPPER AND OFDM TRANSMITTER & RECEIVER AND DEMAPPER
Fd = 1;                 % symbol rate (1Hz)
Fs = 1*Fd;              % number of sample per symbol
M  = 4;                 % kind(range) of symbol (0,1,2,3)

Ndata = 300;            % all transmitted data symbol
Sdata = 50;             % 50 data symbol per frame to ifft
Slen  = 50;             % 50 length symbol for IFFT
Nsym  = Ndata/Sdata;    % number of frame -> Nsym frame

%vector initialization
plcp_z0=zeros(Slen,2);
plcp_z1=zeros(Ndata/Sdata*Slen,1);

Kmod = 1/sqrt(2);

% Convertint eh binary values to QPSK symbols
for pl = 1 : Ncbp6s/2
  I(pl) = 2*(Final_plcp_data(2*pl-1))-1;
  Q(pl) = 2*(Final_plcp_data(2*pl))-1;
end
I = I';
Q = Q';
QpskVal = [I  Q];
plcp_data = Kmod * QpskVal ;

%covert to complex number
plcp_Y2 = amodce(plcp_data,1,'qam');
clear plcp_data;

for j=1:Nsym;

  for i=1:Sdata;
    plcp_Y3(i+Slen/2-Sdata/2,1) = plcp_Y2(i+(j-1)*Sdata,1);
  end

  for i=1:Slen;
    plcp_Y4(((j-1)*Slen)+i) = plcp_Y3(i,1);
  end

  plcp_z0=ifft(plcp_Y3);

  for i=1:Slen;
    plcp_z1(((j-1)*Slen)+i)=plcp_z0(i,1);
  end
```

```
end

% Adding White Noise.
plcp_z2 = awgn(plcp_z1,15,'measured');

for j=1:Nsym;

   for i=1:Sdata;
      plcp_Y5(i+Slen/2-Sdata/2,1) = plcp_z2(i+(j-1)*Sdata,1);
   end

   plcp_Y6=fft(plcp_Y5);

   for i=1:Slen;
      plcp_Y7(((j-1)*Slen)+i)=plcp_Y6(i,1);
   end

end

plcp_Y7;

plcp_Demoddata = plcp_Y7/Kmod ;

% Converting the Qpsk sysmbols to binary values
plcp_Demoddata = plcp_Demoddata';
plcp_demodulated_symbol = qamdemod(plcp_Demoddata,4);
plcp_demodulated_bin_symbol = de2bi(plcp_demodulated_symbol,2);
C1 = plcp_demodulated_bin_symbol( : ,1);
C2 = plcp_demodulated_bin_symbol( : ,2);
plcp_order_changed_demod_data = [ C2 C1];
plcp_demod_data = plcp_order_changed_demod_data';


for  ij = 1 : Ncbp6s
   plcp_final_demod_data(ij) = plcp_demod_data(ij);
end

if(plcp_final_demod_data == Final_plcp_data)
   display('Comparison is correct, received demodulated PLCP data');
else
   display('Comparison is wrong, not received correct demodulated PLCP data');
end

% MAPPER AND OFDM TRANSMITTER & RECEIVER AND DEMAPPER

Fd = 1;                % symbol rate (1Hz)
Fs = 1*Fd;             % number of sample per symbol
M  = 4;                % kind(range) of symbol (0,1,2,3)

Ndata = 300;           % all transmitted data symbol
Sdata = 50;            % 50 data symbol per frame to ifft
Slen  = 50;            % 50 length symbol for IFFT
Nsym  = Ndata/Sdata;       % number of frame -> Nsym frame

% vector initialization
psdu_z0=zeros(Slen,2);
```

```
psdu_z1=zeros(Ndata/Sdata*Slen,1);

Kmod = 1/sqrt(2);

%Converting into the QPSK modulated sysmbols.
for pl = 1 : Ncbp6s/2
   psdu_I(pl) = 2*(Final_psdu_data(2*pl-1))-1;
   psdu_Q(pl) = 2*(Final_psdu_data(2*pl))-1;
end
psdu_I = psdu_I';
psdu_Q = psdu_Q';
psdu_QpskVal = [psdu_I  psdu_Q];
psdu_data = Kmod * psdu_QpskVal ;

%covert to complex number
psdu_Y2 = amodce(psdu_data,1,'qam');

for j=1:Nsym;

   for i=1:Sdata;
      psdu_Y3(i+Slen/2-Sdata/2,1) = psdu_Y2(i+(j-1)*Sdata,1);
   end

   for i=1:Slen;
      psdu_Y4(((j-1)*Slen)+i) = psdu_Y3(i,1);
   end

   psdu_z0=ifft(psdu_Y3);

   for i=1:Slen;
      psdu_z1(((j-1)*Slen)+i)=psdu_z0(i,1);
   end

end

% Adding the White Noise in the channel.
psdu_z2 = awgn(psdu_z1,15,'measured');

for j=1:Nsym;

   for i=1:Sdata;
      psdu_Y5(i+Slen/2-Sdata/2,1) = psdu_z2(i+(j-1)*Sdata,1);
   end

   psdu_Y6=fft(psdu_Y5);

   for i=1:Slen;
      psdu_Y7(((j-1)*Slen)+i) = psdu_Y6(i,1);
   end

end

% The FFT demodulated data of the Receiver
psdu_Y7;

% Normalising the Demodulated data
psdu_Demoddata = psdu_Y7/Kmod ;
```

```
% Converting the QPSK modulated symbols back to binary sysmbols.
psdu_Demoddata = psdu_Demoddata';
psdu_demodulated_symbol = qamdemod(psdu_Demoddata,4);
psdu_demodulated_bin_symbol = de2bi(psdu_demodulated_symbol,2);
C1 = psdu_demodulated_bin_symbol( : ,1);
C2 = psdu_demodulated_bin_symbol( : ,2);
psdu_order_changed_demod_data = [ C2 C1];

% taking transpose to get demodulated data
psdu_demod_data = psdu_order_changed_demod_data';

% Convert to serial data
for  ij = 1 : Ncbp6s
   psdu_final_demod_data(ij) = psdu_demod_data(ij);
end

% PSDU DATA COMPARISON
if(psdu_final_demod_data == Final_psdu_data)
   display('Comparison is correct, received demodulated PSDU data');
else
   display('Comparison is wrong, not received correct demodulated PSDU data');
end

% DE Interleaver
Ncyc = 33;
for p = 0 : Ncbp6s-1
  kp = floor(p/Ncbps)*Ncbps + mod((p + floor(p/Ncbps)*Ncyc), Ncbps);
  Secondperm_deinterleaved_plcp_data(kp+1) = plcp_final_demod_data(p+1) ;
  Secondperm_deinterleaved_psdu_data(kp+1) = psdu_final_demod_data(p+1) ;
end

%second permutation
NTint = 10;
for l = 1 : Ncbp6s/Ncbps
  for k=0:Ncbps-1
     jk = floor(k/10)+ 10 * mod(k,10);
     firstPerm_deinterleaved_plcp_data(((l-1)*Ncbps)+ jk+1) =
Secondperm_deinterleaved_plcp_data(((l-1)*Ncbps)+ k+1);
     firstPerm_deinterleaved_psdu_data(((l-1)*Ncbps)+ jk+1) =
Secondperm_deinterleaved_psdu_data(((l-1)*Ncbps)+ k+1);
  end
end

% first permutation
for k = 0:Ncbp6s-1
   mk = 6*mod(k,Ncbps)+floor(k/Ncbps);
   deinter_plcp_data(mk+1)=firstPerm_deinterleaved_plcp_data(k+1);
   deinter_psdu_data(mk+1)=firstPerm_deinterleaved_psdu_data(k+1);
end

% Comparision of PLCP deinterleaved data with data in transmiter before
% sending to interleaver.
if(deinter_plcp_data == PLCP_CONV_DATA)
   display('Comparison is correct, required PLCP data deinterleaved');
else
   display('Comparison is wrong, deinterleaved wrong PLCP data');
end
```

```
if(deinter_psdu_data == PSDU_CONV_DATA)
   display('Comparison is correct, required deinterleaved PSDU data');
else
   display('Comparison is wrong, deinterleaved wrong PSDU Data');
end

% VITERBI DECODER

%VITERBI DECODER FOR PLCP DATA
convdecod_plcp_data = vitdec(deinter_plcp_data,Tr,7,'trunc','hard');

%VITERBI DECODER FOR PSDU DATA
convdecod_psdu_data = vitdec(deinter_psdu_data,Tr,7,'trunc','hard');

% Comparison of the PLCP DATA before convolutional Encoder
if(convdecod_plcp_data == PLCP_DATA)
   display('Comparison is correct, required PLCP data recovered by viterbi decoder');
else
   display('Comparison is wrong, required PLCP data not recovered by viterbi decoder');
end

% Comparison of the PSDU DATA before convolutional Encoder
if(convdecod_psdu_data == PSDU_DATA)
   display('Comparison is correct, required PSDU data recovered by viterbi decoder');
else
   display('Comparison is wrong, required PSDU data not recovered by viterbi decoder');
end


% Extracting PHY HEADER from viterbi decoded PLCP Data
data_phy = convdecod_plcp_data(1:40);

% Finding the length of Data from PCLP packet Header
data_len = data_phy(9:20);

% Measuring the Length in Bytes
frame_len_bytes = data_len(1) + 2*data_len(2) + 4*data_len(3)+ 8*data_len(4) + 16*data_len(5) +
32*data_len(6) + 64*data_len(7) + 128*data_len(8) + 256*data_len(9) + 512*data_len(10)
+1024*data_len(11) + 2048*data_len(12);

% Length in bits
frame_len_bits = 8*frame_len_bytes;

% Mac Header from viterbi decoded PLCP Header
data_apnd_mac = convdecod_plcp_data(47:142);

% Buffer filled with zeros to decode the RS parity bits.
buff = zeros(1,1856);

% RS PARITY bits
rs_parity = convdecod_plcp_data(149:196);

% PLCP data which is to be RS decoded
plcp_req_redecode = [buff data_phy data_apnd_mac rs_parity];

% RS DECODER
```

```
Tx = 0;
m1 = 8;
n = 255;
k = 249;
length(plcp_req_redecode);
plcp_req_redecode = reshape(plcp_req_redecode,8,length(plcp_req_redecode)/8)';
plcp_req_redecode = bi2de(plcp_req_redecode,'left-msb');

% Decoder
if Tx == 0
 plcp_yk2 = plcp_req_redecode;
   msg = gf([plcp_yk2],m);
   if n==k
      codeRS = yk2;
   elseif n~=k
      codeRS = rsdec(msg',n,k);
      codeRS = codeRS.x;
   end
   PLCP_DATA_DECODED_RS = codeRS(1:end);
end

% Converting the data to double
PLCP_DATA_DECODED_RS = double (PLCP_DATA_DECODED_RS);

% converting the PLCP decoded data to binary data
PLCP_DATA_DECODED_RS = de2bi (PLCP_DATA_DECODED_RS,8,'left-msb');

% Reshaping to the serial mode from the parallel
PLCP_DATA_DECODED_RS = reshape (PLCP_DATA_DECODED_RS', 1,
length(PLCP_DATA_DECODED_RS)*8);

% PHY MAC Headers.
plcp_req_redecod_data = PLCP_DATA_DECODED_RS(end-135:end);

% PHY HEADER RECOVERED
PHY_recovered = plcp_req_redecod_data(1:40);

% MAC HEADER RECOVERED
APPANDSCR_DEC = plcp_req_redecod_data(41:136);

% Comparison of the Combined MAC HEADER & SCRAMBLED
if(APPANDSCR_DEC == APPEND_SCRAMBLE_AFTER)
   display('Comparison is correct, required PLCP data headers');
else
   display('Comparison is wrong, required PLCP data not obtained');
end

% DESCRAMBLER
seed_value1 = [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1];

% data randomization
for i=1:size(APPANDSCR_DEC,2)

   % XORing of bit X15 and bit X14
   xor_out = bitxor(seed_value1(15), seed_value1(14));

   %randomized data value
   randomized_data1(i) = bitxor(xor_out, APPANDSCR_DEC(i));
```

```
    %new seed value
    seed_value1 = [xor_out seed_value1(1:14)];
end

% Comparing Descrambled PLCP data FinalHeader.
if(randomized_data1 == APPEND_SCRAMBLE_BFR)
    display('Comparison is correct, required descrambled PLCP data');
else
    display('Comparison is wrong, required descambled PLCP data not obtained');
end


% The seed value which has to be taken
seed_value1 = [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1];

% data randomization
for i=1:size(convdecod_psdu_data,2)

    % XORing of bit X15 and bit X14
    xor_out = bitxor(seed_value1(15), seed_value1(14));

    %randomized data value
    randomized_data1(i) = bitxor(xor_out, convdecod_psdu_data(i));

    %new seed value
    seed_value1 = [xor_out seed_value1(1:14)];
end

% The Received Frame payload data with the 32 bits FCS appeneded
FRAME_PAYLOAD_DEC = randomized_data1(1:frame_len_bits + 32);

% Tail Bits
TAIL = [0 0 0 0 0 0];

% The pad bits extracted from the De-randomized convolutional decoded data
PADBITS_REC = randomized_data1( frame_len_bits + 39 : end);

% data obtained from descrambler
randomized_data2 = [ FRAME_PAYLOAD_DEC TAIL PADBITS_REC];

% Comparing the descambled PSDU data
if(randomized_data2 == APP_DATA_BFR_SCR)
    display('Comparison is correct, required descrambled PSDU data ');
else
    display('Comparison is wrong, required descrambled PSDU data not obtained');
end

% The 32 zeros which has to appended to calculate the FCS
FCS_append_REC = zeros(1,32);

% Data appended with zero FCS
temp_frame_demod_data =  [FRAME_PAYLOAD_DEC FCS_append_REC];

% Complementing the first 32 bits of the data appended
FCS_DEC = bitcmp(temp_frame_demod_data(1:32),1);
```

```
% The FCS calculation
for i=1:length(FRAME_PAYLOAD_DEC)
   nm2 = [FCS_DEC temp_frame_demod_data(i+32)];
   if nm2(1)==1
      FCS_DEC = xor(nm2(2:33),gx(2:33));
   else
      FCS_DEC = nm2(2:33);
   end
end

% The Absolute value of the FCS obtained
FCS_DEC = abs(FCS_DEC);

% The standard FCS sequence as per specification
FCS_REQ = [ 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 ];

% Comparing the FCS obtained with the above sequence.
if(FCS_DEC == FCS_REQ)
   display('Comparison is correct, required FCS decoded, Frame payload received is correct');
else
   display('Comparison is wrong, required FCS not obtained, error in the Frame payload');
end

clear all;
```

6. CONCLUSION

Thus Verification Environment for the USB 2.0 protocol with OTG support is developed as a standalone IP. The VIP id developed to support all the Speeds, Interfaces and Tasks with USB2.0 and OTG support. The VIP is developed for the directed environment. The VIP is working in the test bench environment for the UTMI+ interface. The Wireless USB is implemented as per the Ecma standard and the protocol is verified for a specific packet following the normal OFDM approach which is backward compatible to the wired USB.

Scope for Future Work

There are few tasks yet to be implemented like Carkit, FsLsSerialmode and LowPowerMode which are given in ULPI Specification. The OFDM implemented for the Wireless USB is ordinary and can be enhanced to Multiband OFDM, the standard called MBOA. The Wireless USB protocol is verified only for a particular packet which can be randomized and can be implemented for a transaction.

7. References

1) "USB 2.0 Specification", Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, Revision 2.0, April 27th, 2000, (www.usb.org/developers/docs.html).
2) "On-The-Go Supplement to the USB 2.0 Specification", Revision 1.2, April 4th, 2006, (www.usb.org/developers/onthego).
3) Steve McGowan, "USB 2.0 Transceiver Macrocell Interface Specification", version 1.05, March 29th, 2001, (www.intel.com/technology/usb/download/2_0_Xcvr_Macrocell_1_05.pdf).
4) "UTMI+ Specification", Revision 1.0, February 25th, 2004, (www.ulpi.org/documents.html), (ULPI and UTMI+ specifications (Zip File)).
5) "UTMI+ Low Pin Interface (ULPI) Specification", Revision 1.1, October 20th, 2004, (www.ulpi.org/documents.html), (ULPI and UTMI+ specifications (Zip File)).
6) ECMA-368, "High Rate Ultra Wideband PHY and MAC Standard", December 2007, http://www.ecma-iternational.org/publications/standards/Ecma-368.htm.
7) "Wireless Universal Serial Bus Specification", by Agere, Hewlett-Packard, Intel, Microsoft, NEC, Philips, Samsung, Revision 1.0, May 12th, 2005.
8) R. S. Sherratt, "Design Considerations for the Multiband OFDM Physical Layer in Consumer Electronic Products", *IEEE Tenth International Symposium on Consumer Electronics,* Page(s):1 – 5, ISCE '06.

APPENDIX

A. OTG VOLTAGE AND TIMINGS

VOLTAGE LEVELS:

Parameter	Symbol	Min	Max	Units (volts)
A-device Vbus Valid	Va_vbus_vld	4.4	4.75	V
A-device Session Valid	Va_sess_vld	0.8	2.0	V
B-device Session Valid	Vb_sess_vld	0.8	4.0	V
B-device Session End	Vb_sess_end	0.2	0.8	V
B-device (SRP capable) to Host Output Voltage	Vb_hst_out	0	2.0	V
B-device (SRP capable) to OTG device Output Voltage	Vb_otg_out	2.1	5.25	V

TIMINGS: A-DEVICE

Parameter	Num	Symbol	State	Min[1]	Max[1]	Units
SRP Response Time	1	TA_SRP_RSPNS	a_idle		4.9	sec
Wait for VBus Rise[2]	2	TA_WAIT_VRISE	a_wait_vrise		100	ms
B-Connect Long Debounce	3	TA_BCON_LDB	a_wait_bcon	100		ms
B-connect to A-reset	3	TA_BCON_ARST	a_wait_bcon		30	sec
Wait for B-Connect	4	TA_WAIT_BCON	a_wait_bcon	1		sec
A-Idle to B-Disconnect	5	TA_AIDL_BDIS	a_suspend	200		ms
B-Disconnect to A-Connect	6	TA_BDIS_ACON	a_suspend		3	ms
B-Idle to A-Disconnect	7	TA_BIDL_ADIS	a_peripheral	3	200	ms
Local Disconnect to Data Line Discharge	8	TLDIS_DSCHG	a_wait_bcon	25		µs
B-Connect Short Debounce	9	TA_BCON_SDB	a_wait_bcon	2.5		µs
B-Connect Short Debounce Window	10	TA_BCON_SDB_WIN	a_wait_bcon		100	ms

B-DEVICE

Parameter	Num	Symbol	State	Min[1]	Max[1]	Units
SE0 Time Before SRP	11	TB_SE0_SRP	b_idle	2		ms
Data-Line Pulse Time	12	TB_DATA_PLS	b_srp_init	5	10	ms
SRP Initiate Time	13	TB_SRP_INIT	b_srp_init		100	ms
SRP Fail Time	14	TB_SRP_FAIL	b_srp_init	5	6	sec
Session Valid to B-Connect	15	TB_SVLD_BCON	b_idle		1	sec
A-Idle to B-Disconnect	16	TB_AIDL_BDIS	b_peripheral	4	150	ms
Time between B-device HS to FS transition during suspend, and B-device disconnect	17	TB_FS_BDIS	b_peripheral	1	146.875	ms
Local Disconnect to Data Line Discharge	18	TLDISC_DSCHG	b_wait_acon	25		µs
A-SE0 to B-Reset	19	TB_ASE0_BRST	b_wait_acon	3.125	as per USB 2.0	ms
A-Connect Debounce	20	TB_ACON_DBNC	b_wait_acon	2.5		µs
A-Connect to B-SE0	21	TB_ACON_BSE0	b_wait_acon		1	ms

B. ULPI REGISTERS:

Function Control Register Fields

Field name	Bit	Access	Reset	Description
XcvrSelect	1:0	rd/wr/s/c	01b	Selects the required transceiver speed 00b: Enable HS transceiver 01b: Enable FS transceiver 10b: Enable LS transceiver 11b: Enable FS transceiver for LS packets
TermSelect	2	rd/wr/s/c	0b	Controls the internal 1.5K ohm pull-up resistor and 45 ohm Hs terminations. Control over the bus resistors changes depending on XcvrSelect, OpMode, DpPulldown and DmPulldown.
OpMode	4:3	rd/wr/s/c	00b	Selects the required bit encoding style during transmit 00b: Normal operation 01b: Non-driving 10b: disable bit-stuff and NRZI encoding

				11b: Optional. Do not automatically add SYNC and EOP when transmitting. Must be used only for HS packets.
Reset	5	rd/wr/s/c	0b	Resets the State machine running the PHY.
SuspendM	6	rd/wr/s/c	1b	Puts PHY in Low Power Mode when bit is set Low and PHY automatically sets this bit to 1 when exited from Low Power Mode 0b: Low Power Mode 1b: Powered
Reserved	7	rd/wr/s/c	0b	Reserved

OTG Control Register Fields

Field name	Bit	Access	Reset	Description
IdPullup	0	rd/wr/s/c	0b	Connects a pull-up to the ID line and enables sampling of the signal level 0b: Disable sampling of ID line 1b: Enable sampling of ID line.
DpPulldown	1	rd/wr/s/c	1b	Enables the 15k ohm pull-down resistor on D+. 0b: Pull-down resistor not connected to D+. 1b: Pull-down resistor connected to D+.
DmPulldown	2	rd/wr/s/c	1b	Enables the 15k ohm pull-down resistor on D-. 0b: Pull-down resistor not connected to D-. 1b: Pull-down resistor connected to D-.
DischrgVbus	3	rd/wr/s/c	0b	Discharge Vbus through a resistor. 0b: do not discharge Vbus. 1b: discharge Vbus.
ChrgVbus	4	rd/wr/s/c	0b	Charge Vbus through Resistor 0b: do not charge Vbus 1b: charge Vbus.

DrvVbus	5	rd/wr/s/c	0b	Signals the internal charge pump or external supply to drive 5v on Vbus 0b: do not drive Vbus 1b: drive 5V on Vbus.
DrvVbus External	6	rd/wr/s/c	0b	Selects between the internal and the external 5V Vbus supply. 0b: Drive Vbus using the internal charge pump. 1b: Drive Vbus using external supply.
UseExternal VbusIndicator	7	rd/wr/s/c	0b	Tells the PHY to use an external Vbus over-current indicator. 0b: Use the internal OTG comparator. 1b: Use external Vbus valid indicator signal.