

**MINISTRY OF EDUCATION AND TRAINING  
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION   
FCAULTY FOR HIGH QUALITY TRAINING**

Logo

Description automatically generated

**Report**

**Project: USB**

Subject: Integrated Circuit and System Design

Code subject: ICSD336774E\_21\_2\_01CLC

Instructor: TS. Do Duy Tan

Student: Tran Quang Nhat 20119148

Phan Nhu Khoi 20119189

***Ho Chi Minh City, May 2022***

# **CHAPTER 1: OVERVIEW**

* 1. **Question**

Today, science and technology are developing more and more and achieving many brilliant achievements in all fields in general, including electronics. The world's electronics has produced many technologies with high gray matter content, along with products that are versatile and superior in features. In recent years, electronics has also developed highest in Vietnam and is considered to have great able, so it is even more important to learn, learn and grasp information and knowledge about electronics. According to that trend, we would like to present the project implemented in the subject project called USB controller design for the purpose of learning and grasping important knowledge about standards USB.

* 1. **Objectives**

In this report, we will learn and understand clearly the USB communication standard, design a USB data transmitter, build a USB test model.

* 1. **Research content**

The research report on USB communication standard, how USB work, designing an RTL model of a USB receiver, designing a data transmission via USB.

Moreover, we also find out about kit Spartan 3E and use Xilinx ISE to code for transmitter and simulators.

* 1. **Outline**

Contents

[**CHAPTER 1: OVERVIEW** 2](#_Toc104625599)

[**1.1** **Question** 2](#_Toc104625600)

[**1.2** **Objectives** 2](#_Toc104625601)

[**1.3** **Research content** 2](#_Toc104625602)

[**1.4** **Outline** 2](#_Toc104625603)

[**CHAPTER 2: THEORETICAL BASIC** 4](#_Toc104625604)

[**2.1 USB Port** 4](#_Toc104625605)

[**2.2 USB Bus model** 4](#_Toc104625606)

[**2.3 USB transfer modes:** 5](#_Toc104625607)

[**a.** **Control Transfers:** 5](#_Toc104625608)

[**b.** **Interrupt Transfers:** 5](#_Toc104625609)

[**c.** **Bulk Transfers:** 5](#_Toc104625610)

[**d.** **Isochronous Transfers:** 5](#_Toc104625611)

[**2.4 Physical Interface** 5](#_Toc104625612)

[**2.5 Transmission Protocol** 6](#_Toc104625613)

[**2.6 Types of packets in USB communication:** 7](#_Toc104625614)

[**2.7 USB Transfer Processes** 8](#_Toc104625615)

[**a. Control transmission:** 8](#_Toc104625616)

[b. **Interrupt Transfer:** 10](#_Toc104625617)

[**c. Isochronous Transfers** 11](#_Toc104625618)

[**d.** **Bulk Transfers** 11](#_Toc104625619)

[**CHAPTER 3: INTRODUCING SPARTAN 3E KIT** 12](#_Toc104625620)

[**3.1 Components of the kit** 12](#_Toc104625621)

[**3.2 Resource** 13](#_Toc104625622)

[**3.3. Oscillator on the kit** 13](#_Toc104625623)

[**3.4. Power supply** 14](#_Toc104625624)

[**3.5. Configuration FPGA.** 14](#_Toc104625625)

[**3.6. SPARTAN Structure Overview - 3E** 14](#_Toc104625626)

[**3.7. Input/Output Blocks (IOBs):** 15](#_Toc104625627)

[**3.8. Configurable Logic Blocks (CLBs):** 16](#_Toc104625628)

[**3.9. Block RAM:** 16](#_Toc104625629)

[**3.10. Dedicated Multipliers:** 17](#_Toc104625630)

[**3.11. Digital Clock Managers (DCMs):** 18](#_Toc104625631)

[**CHAPTER 4: DESIGN USB DATA TRANSFER** 18](#_Toc104625632)

[**4.1 Introduction of USB transmission model** 18](#_Toc104625633)

[**4.1.1 USB Transmitter Unit (UTMI):** 19](#_Toc104625634)

[**4.1.2 Serial Interface Engine (Serial Interface Engine)** 19](#_Toc104625635)

[**4.2 USB Transmitter Interface Block (UTMI):** 19](#_Toc104625636)

[**4.2.1 Diagram of UTMI function blocks:** 19](#_Toc104625637)

[**4.2.2. Detailed description of function blocks:** 20](#_Toc104625638)

[**4.3. RTL level of the USB receiver** 21](#_Toc104625639)

[**4.4 Model of USB transmission block** 21](#_Toc104625640)

[**4.4.1 Description of USB TX block input and output signals** 22](#_Toc104625641)

[**4.4.2 Description of function blocks:** 23](#_Toc104625642)

[**CHAPTER 5 WRITING CODE FOR USB TRANSFER AND SIMULATION** 27](#_Toc104625643)

[**5.1 Design of data transmitter** 27](#_Toc104625644)

[**5.2 Code for transmitter and simulation** 28](#_Toc104625645)

[**CHAPTER 6 CONCLUSION** 35](#_Toc104625646)

# **CHAPTER 2: THEORETICAL BASIC**

## **2.1 USB Port**

USB port (Universal Serial Bus) is a serial standard developed by a group of many companies: Compaq, DEC, IBM, Intel, Microsoft, NEC and Nortel. The idea of ​​building a USB port was to develop a port that can The USB bus was developed in several versions:

* USB 1.0: introduced in 1996, with a maximum speed of 12Mbit/s.
* USB 1.1: introduced in 1997, inherited and developed from USB 1.0. Besides, it also supports 2 transfer rates: 12Mbit/s (Full speed) and 1.5 Mbit/s (low speed).
* USB 2.0: introduced in 200, with compatibility with earlier versions and optional high speed 480Mbit/s.
* USB 3.0: released in 2008, 10 times faster than USB 2.0.

A picture containing text, watch

Description automatically generated

Figure 1.1: Symbols of USB(a), connector(b) and cable(c)

## **2.2 USB Bus model**

The basic structure of the USB network is the star layers. A USB system consists of one or more **peripherals**, one or more **hubs**, and a **host controller**. The controller works closely with the **root hub** to extend the points. connected to the master, from the hub to the root will connect to other **devices or hubs**.

Diagram

Description automatically generated

4th floor

3rd floor

2nd floor

1st floor

Base floor

Figure 1.2: USB bus connection model

1st USB device

1st USB device

1st USB device

1st USB device

Hub USB

USB Porrt

Figure 1.3: Star connection model of a USB connection

## **2.3 USB transfer modes:**

Data exchanged between a USB device and a computer uses one of four transfer modes: control transfer, interrupt transfer, block transfer, and synchronous transfer.

1. **Control Transfers:**

Usually used to install hardware and issue device control commands. This type is operated at high priority with automatic error control.

1. **Interrupt Transfers:**

Used for devices that need to provide a small amount of data (only transfer in the input direction) and cyclically such as: mouse, keyboard, etc. Unlike interrupt mechanism, here no interrupt request is sent to the USB host, but the computer will periodically ask around. Usually, the device sends to the host 8 bytes of data in one transmission.

1. **Bulk Transfers:**

When there is a large amount of data to be transferred and transmission error control is required, but there is no time requirement, the data is usually transmitted in block mode. This type is often applied to devices such as printers, machines. to scan.

1. **Isochronous Transfers:**

When the amount of data to be transferred is large at a predetermined data rate, such as for a sound card. maintain, drop and minor bugs.

## **2.4 Physical Interface**

USB bus has two common types of connectors: type A and type B. USB bus uses a four-wire Cable to connect devices, where a pair of twisted-pair lines is used as the differential data (D+ and D-) with two wires left to feed 5V and ground (GND)

A picture containing text, weapon, screenshot

Description automatically generated

Figure 1.4 Physical Interface of USB

## **2.5 Transmission Protocol**

**a. Fields in USB communication:**

The bit order transmitted over the USB bus is the least significant bit (LSB) transmitted first up to the MSB bit, USB has 4 types of packets transmitted on the bus: identity packet (Token), data packet (Data), frame start packet (Start of Frame) and handshake packet (Handshake). In each packet, there are many fields:

* Synchronization field (SYNC): all all packets are prefixed with a SYNC field, which is the NRZI (Non-Return to Zero Inverted) binary code of the string “KJKJKJ”. All packets are limited between SOP (Start-Of-Packet) and EOP (End-Of-Packet).
* Identification (PID) field: Immediately following the SYNC field in each USB packet, the 8-bit packet length consists of the lower 4 bits being the packet identification bits, and the high 4 bits being the reverse of the lower 4 bits for error control when transmission. The PID field describes the type of packet to be transmitted after it. Both the USB controller and the devices recognize the specified identifier field codes.

|  |  |  |  |
| --- | --- | --- | --- |
| Type PID | Name PID | PID [3:0] | Description |
|  | Out | 0001B | USB host writes out |
|  | In | 1001B | The USB host reads from the device |
|  | SOF | 0101B | Frame start and frame number |
|  | SETUP | 1101B | The USB host installs the device |
| Data | DATA0 | 0011B | Data packet data0 |
|  | DATA1 | 1011B | Data packet data1 |
|  | ACK | 0010B | Confirm reply |
|  | NAK | 1010B | The external device does not accept the data  or the transmitted data is corrupted |
| Handshake | STALL | 1110B | Endpoint is suspended or not supported |
| Special | PRE | 1100B | The USB host provided allows the  bus output to be exchanged with low-speed devices |

Table 1.1: Types of packets transmitted in USB

* Address field (ADDR): is the address reserved for accessing the device. The device can receive data or transmit data to the host (depending on the PID packet value) through its address The address field uses 7 bits [A0: A6]. The address field is included in the IN, SETUP, OUT packets. Each address assigned to a single device when starting to power devices has a default address of zero and the USB host receives the device's descriptor through this default address.
* End point field: an endpoint field using 4 bits that allows the selection of an end point in the device. Except for the zero endpoint, the number of endpoints is separate for each functional device. An endpoint is the address of a buffer in the device, the address field defined in IN, SETUP, and OUT packets. From the endpoint to the USB host establishes a communication channel called a pipe. All functional devices support a default pipe with zero endpoint, the USB host will exchange with the newly attached device. through this pipe.
* Frame number field: The frame number field is 11 bits long; it is sent in SOF (Start of Frame Packets) packets. SOF is sent by the USB host at certain intervals.
* Data field: data field from 0 to 1023 bytes in length, the bytes are sent one after the other. In each data byte, the smallest data bit is sent first and the MSB bit last.

## **2.6 Types of packets in USB communication:**

* Token Packets: indicate only the type of transmission immediately following it. There are 3 types of identification packets:

o IN: Indicates the device that the USB host wants to read the information.

o OUT: Tell the device that the USB host wants to send information.

o SETUP: used to initiate control information transmission.

General format of identity packets: Table

Description automatically generatedFigure 1.5 Identity packets

Each packet has 4 PID fields (1 byte) indicating the packet type, 7 address bits (Address), 4 endpoint bits (End point), and finally 5 transmission error check (CRC) bits.

* + Start Of Frame: A frame start packet emitted from the USB controller, consisting of the following fields:

o 8-bit PID packet identification.

o 11 bits: frame number.

o 5 bits: CRC5 error checking.

Text

Description automatically generated with medium confidenceFigure 1.6 Packet SOF

* + Data Packets: A data packet consisting of a PID field (8 bits). The data field is 0 to 1023 bytes long and 16 bits CRC. Two types of data packets are identified by the field. PID is Data0 and Data 1, these two packet types are defined to aid in synchronous transmission.

Table

Description automatically generated

Figure1.7 Data package

* + Handshake Packets: There are three types of handshakes

o ACK: reply received data successfully.

o NAK: Indicates that the device does not respond to received data or that the received data is faulty. All handshake packets are transmitted in an interrupt to notify the host USB controller that data was not sent.

o STALL: Indicates that the USB device is not capable of transmitting or receiving data or the requested pipe is not supported.

Table

Description automatically generatedFigure 1.8 Handshake Packets

## **2.7 USB Transfer Processes**

### **a. Control transmission:**

Control transmission is done through 3 stages: Setup, Data and Status. For low-speed devices the packet size is 8 bytes, while for full-speed the packet size can be 8, 16, 32 or 64 bytes.

Stage 1(Setup): consists of 3 packets

* + Identity packet: sent by host USB controller to USB device. In this packet, USB host sends device's address and endpoint.
  + The data packet is sent immediately after the Setup packet ends, and the packet's PID field is always of type Data0.
  + Handshake packet: last packet answered by USB device: ACK if no error, NAK if error occurred.

A picture containing text, clock, device, gauge

Description automatically generated

installation error

Data error

Success

Handshake

Data

Identity

Host USB



* USB Device

Figure 1.9: Packets in phase 1 of control transmission.

Stage 2 (Data): This stage also contains 3 packets: identification, data and handshake:

* Data packet: indicates whether the data is in the input (IN) or output (OUT) direction.
* Data packet: contains data to be transferred
* In case the data has direction from the USB device to the USB host. If no error occurs, the data will be sent to the USB host. Where an error occurs depends on the error that the sending USB device replies with “STALL” or “NAK”.
  + - * In case the data is directed from the USB host to the USB device, the data packet will be sent to the device.
      * Handshake packet: when receiving data, the USB master will reply with an ACK packet to the device. In the case of output depending on the data that the USB device perceives as having an error or not, it sends a reply to the USB controller with one of the packets: ACK, NACK, STALL.

|  |
| --- |
| Success |
| Error |

Diagram

Description automatically generatedText

Description automatically generated with medium confidence

Figure 1.10: Packets in phase 2 of control transmission.

Stage 3 (Status Stage): returns the status of the whole process

* + - Input: depending on whether the exchange has an error or not, the USB device will reply to the USB host with one of the following packets: ACK, STALL or NACK (Figure a).
    - Outbound: master sends reply to packets to USB device

Diagram

Description automatically generated

A picture containing shape

Description automatically generated

Figure 1.11: Packets in phase 3 of the control process.

### b. **Interrupt Transfer:**

Unlike the message exchange method by interrupt program calls, USB interrupt transmission is completely polled by the USB host. If a USB device actively requests it, it must wait until the USB host asks. arrival and response. Interrupt transmission is relatively simple with only 3 packets per transmission. Interrupt transmission is commonly used in asynchronous communication devices, small and simple devices.

Interrupt transmission stipulates the size of packets depending on the device speed:

* For Low-speed devices: maximum size is 8 bytes
* For Full-Speed ​​devices: up to 64 bytes.
* With High-Speed ​​devices: up to 1024 bytes max

Diagram

Description automatically generatedA picture containing diagram

Description automatically generated



Figure 1.12: Interrupt transmission stages

b.1) Interrupt IN:

The USB host will ask the interrupt endpoints periodically. The specific polling rate set in the packet describes each device's endpoint. Each poll requires the master to send an IN Token. If the IN Token is corrupted, the device ignores this packet and continues to monitor the bus for new IN Tokens.

If there is a request the device sends a data packet to the USB host and waits for a reply. If there is no interrupt the device replies to the host with NACK or STALL.

b.2) Interrupt OUT:

When the master wants to send data to the interrupt device, it issues an OUT token followed by an interrupt data packet. If the OUT Token or data packet (Data Packet) is damaged, the USB device will ignore this packet. If the device's interrupt end buffer is empty, it will return an ACK packet to the USB host. In case the endpoint buffer is empty due to previously received data, the USB device will return a NAK. However, if an error occurs out with an endpoint, it will return STALL.

### **c. Isochronous Transfers**

Used when the data to be transmitted is large and the data rate is specified such as: audio, video stream. In this transmission a data rate value is maintained and no error checking. Synchronous transmission is also supported. Supports both directions: synchronous transmission in the direction of input (Isochronous IN) and synchronous transmission in the direction of output (Isochronous OUT). The maximum size of the data packet in this type of transmission.

* Full-Speed ​​Devices: 1023 bytes
* High-Speed ​​Devices: 1024 bytes

Diagram, schematic

Description automatically generated

 Figure 1.13 Isochronous Transfers

1. **Bulk Transfers**

Used when there is a large amount of data to be transmitted and it is necessary to control transmission errors, but there are no constraints on transmission time. This type of transmission is often applied to devices such as printers, scanners, etc. Block transfer is only supported for Full-Speed ​​and High-Speed ​​devices. Maximum packet size that the device supports

* Full-Speed: 8, 16, 32 or 64 bytes.
* High-Speed: 512 bytes

Block transfer is also relatively simple, consisting of only three packets: an identity packet, a data packet, and a handshake.

Diagram

Description automatically generated Diagram

Description automatically generated



Figure 1.14: Packets in block transmission

d.1) Input: When the USB host is ready to receive data, it issues an IN Token. If the device receives an IN Token with an error, it will ignore the packet. If the IN Token is received without error, the USB device returns the data packet to be transmitted. In the event of an error, the device returns NACK or STALL.

d.2) Outbound: when the USB host wants to send a data packet to the functional device, it sends to the OUT-Token device, the data is corrupted or the device buffer is full, it will reply with a NACK packet. If no error occurs, when data is received, the USB device responds with an ACK packet. In case the endpoint fails, it responds with STALL.

# **CHAPTER 3: INTRODUCING SPARTAN 3E KIT**

## **3.1 Components of the kit**

1. Xilinx XC3S500E Spartan-3E FPGA: main KIT

2. Xilinx 4 Mbit Platform Flash configuration PROM

3. Xilinx 64-macrocell XC2C64A Cool Runner CPLD

4. 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz

5. 16 MByte (128 Mbit) of parallel NOR Flash (Intel Strata Flash)

6. 16 Mbits of SPI serial Flash (Micro)

7. 2-line, 16-character LCD screen

8. PS/2 mouse or keyboard port

9. VGA display port

10. 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)

11. Two 9-pin RS-232 ports (DTE- and DCE-style)

12. On-board USB-based FPGA/CPLDdownload/debug interface

13. 50 MHz clock oscillator

14. SHA-1 1-wire serial EEPROM for bitstream copy protection

15. Hirose FX2 expansion connector

16. Three Diligent 6-pin expansion connectors

17. Four-output, SPI-based Digital-to-Analog Converter (DAC)

18. Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain

19. pre-amplifier

20. Chip Scope™ Soft Touch debugging port

21. Rotary-encoder with push-button shaft

22. Eight discrete LEDs

23. Four slide switches

## **3.2 Resource**

10000 logic cells, with 232 user defined I/O pins.

## **3.3. Oscillator on the kit**

There are 3 sources of clk clock for the FPGA:

a. Use the available 50 MHz quartz on the board.

The output of the quartz is connected to the chip's C9 pin.

b. Supply the clk pulse through the 8-pin pin on the board

. The output of the quartz is connected to the chip's B8 pin.

c. Level the clk pulse through the SMA pin, this pin only receives clk from the clk generator, cannot use quartz directly.

Output clk connected to pin A10 of chip.

Map

Description automatically generated with medium confidence

Figure 3.1 Available Clock Inputs

## **3.4. Power supply**

All the pins of the FPGA have 2 power options of 2.5v or 3.3v through the JP9 jumper. (Left plug 2.5V, right plug 3.3V).

## **3.5. Configuration FPGA.**

There are four ways to program the FPGA (Figure 4-1).

1. Directly feed into the FPGA through the JTAG port or the USB port.

2. Load Platform Flash PROMXCF04S (4Mbit), then configure the FPGA in Master Serial Mode.

3. Load the PROMST Microelectronics serial Flash (16 Mbit), then configure the FPGA in SPI Mode.

4. Load Strata Flash parallel PROM Intel (128 Mbit), then configure the FPGA in BPI Up Mode or BPI Down Mode.

## **3.6. SPARTAN Structure Overview - 3E**

Components:

- Input/Output Blocks (IOBs): I/O blocks

- Configurable Logic Blocks (CLBs) : made up of Look-Up Tables (LUTs).

- Block RAM: Supports 16 Kb RAM per RAM Block, the number of RAM Blocks depends on each chip, with XC3S500E having 20 RAM Blocks.

- Multiplier Blocks: 18-bit input multiplier block.

- Digital Clock Manager (DCM) Blocks: clk pulse control block.

- Interconnect: Connections.

## **3.7. Input/Output Blocks (IOBs):**

I/O blocks Including 2-way I/O I/O blocks and input path blocks (cannot be output). The input is a function block that delays the input signal before it is fed Z into the internal blocks. It is also possible to set the delay for the I/O inputs. We may not use this delay function.

After configuring the FPGA, unused pins are by default set to input with an internal pull-down resistor.

Power supply for IOBs:

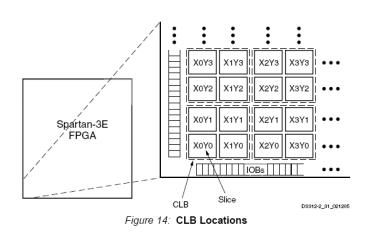
VCCO: Supply power to the output, ie the output voltage will be equal to VCCO. About 3.3 V

VCCINT: Power the blocks inside the FPGA. Approx. 1.2 V.

VCCAUX: Power supply. Less important. About 2.5 V. Diagram

Description automatically generated

## **3.8. Configurable Logic Blocks (CLBs):**

This block performs logic and storage functions. Each CLBs has 8 look-up tables LUTs (Look-Up Table) and 8 memory elements. LUTs are blocks that execute logic expressions, in addition each LUTs can be configured as a 16-bit RAM or a 16-bit register. A memory element is a block that can be specified to become a Flip. -Flop or latch. 

The XC3S500E has 1164 CLBs. Corresponding to 9312 LUTs and 9312 Flip-Flops.

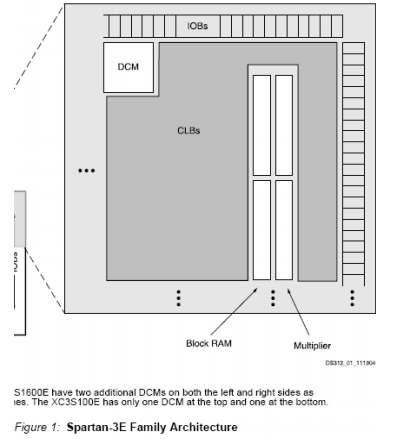
Logic Cell: Each LUTs associated with a memory element is called a "Logic Cell".

XC3S500E has 10476 Logic Cell

## **3.9. Block RAM:**

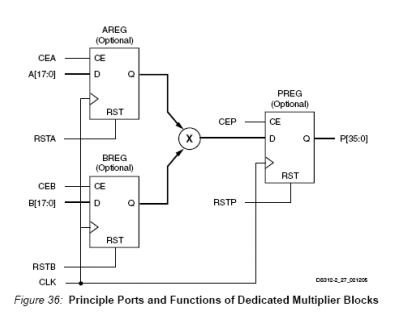
The LUTs in the CLBs (VII.2) section can be configured as RAM, but this RAM is often used as a data buffer, while the RAM Blocks are often used for storing larger data.

XC3S500E contains 20 Blocks of RAM. Each block of RAM can hold 18 Kbit data



**3.10. Dedicated Multipliers:**

The multipliers are placed together with the RAM Blocks into one block. The multiplier supports two 18-bit inputs and 36-bit outputs. Multipliers can be cascaded to increase the number of inputs.



**3.11. Digital Clock Managers (DCMs):**

The clock manager is a block with three main functions (Figure 40):

- Clock-skew Elimination: Helps increase switching speed.

- Frequency Synthesis: Helps to change the frequency at the clock output.

- Phase Shifting: Helps change the clock phase.

The XC3S500E has:

- 16 global clock inputs GCLK0 - GCLK 15 located at the top and bottom of the FPGA.

- 8 clock inputs on the right side RHCLK0 – RHCLK7.

- 8 left clock inputs RLCLK0 – RLCLK7.

**CHAPTER 4: DESIGN USB DATA TRANSFER**

**4.1 Introduction of USB transmission model**

The USB Transceiver or UTMI (USB Transceiver Macrocell Interface) acts as an intermediary between the serial device interface and the physical transmission line. The following model shows the relationship between the components in the block. function of an ASIC device. The system is divided into 3 main components: UTMI, serial control interface and communication device.

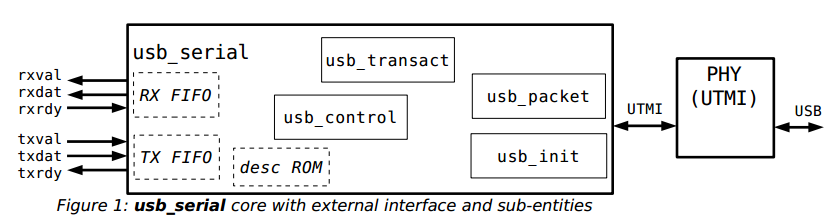
Diagram

Description automatically generated

**4.1.1 USB Transmitter Unit (UTMI):**

This block is responsible for controlling the underlying low-level USB protocols as well as control signals. It includes functions such as: serial and parallel data conversion, bit stuff handling, synchronization. as well as error handling.

**4.1.2 Serial Interface Engine (Serial Interface Engine)**

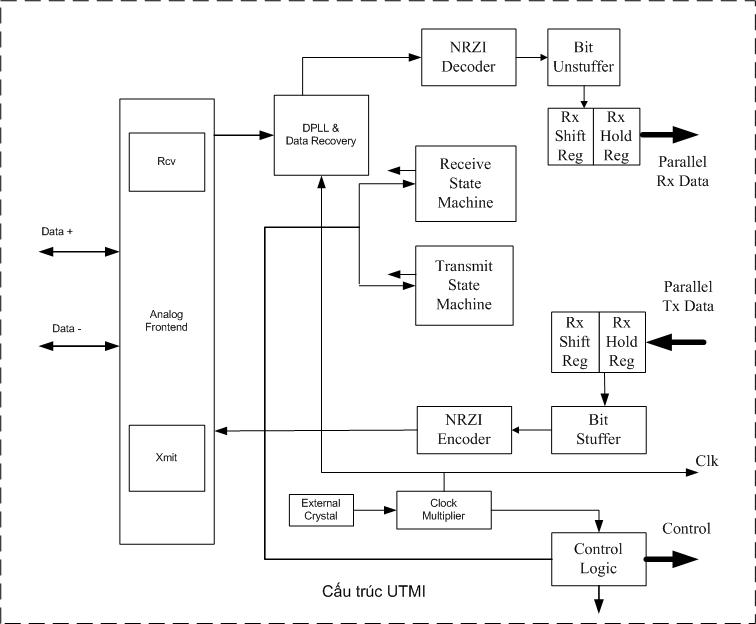
****

This block can be divided into two smaller blocks: the SIE logic control block (Serial Interface Endpoint) and the logic terminal block (Logical Endpoint). The SIE logic control block contains the USB PID, and the logical identifier addresses along with the machines. State to control USB packets as well as communication and handshake processes. Logical Endpoints contain special logic units such as endpoint identifiers, FIFOs, and controllers. FIFO. In which the SIE logic controller is the most important part, necessary for all USB-related designs, the remaining endpoints are customized with many functions depending on the application and requirements.

## **4.2 USB Transmitter Interface Block (UTMI):**

### **4.2.1 Diagram of UTMI function blocks:**

The following figure shows the function blocks of the USB transmitter and receiver. The blocks will be detailed below:



Structure UTMI

Figure 2.2: Function blocks in the USB transmitter

### **4.2.2. Detailed description of function blocks:**

a, Analog Frontend:

Is the analog interface of the transmission and reception system, converting logic levels to the corresponding line voltage signals? Input of the block is taken from the NRZI encoded signal and processed by the Xmit unit, similarly the input analog signal is processed by the Rcv block before being sent to the next blocks.

b, State Machine block:

Coordinating all low-level transmission protocols, generating control signals for the entire system operation.

c, Control Logic block:

Generates state machine control signals as well as line control signals.

d, Data synchronization block

Generates a sampling pulse to synchronize transmit and receive.

e, Stuff bit generation or detection block

Generates the stuff bit for the transmitter data stream before encoding or knowing to remove the stuff bit from the data at the receiver.

f, NRZI encoder/decoder block

Data is encoded as NRZI before being sent to the physical transmission field in the transmitter block. The reverse is done at the receiver side to recover the received signal.

g, Shift Register Block

An interface that converts between serial and parallel.

h, Input and output data storage block

Includes registers containing data for output or input. There are also other control signals.

## **4.3. RTL level of the USB receiver**

The following figure depicts the functional blocks of a complete USB transceiver block with the connection signals between the blocks.

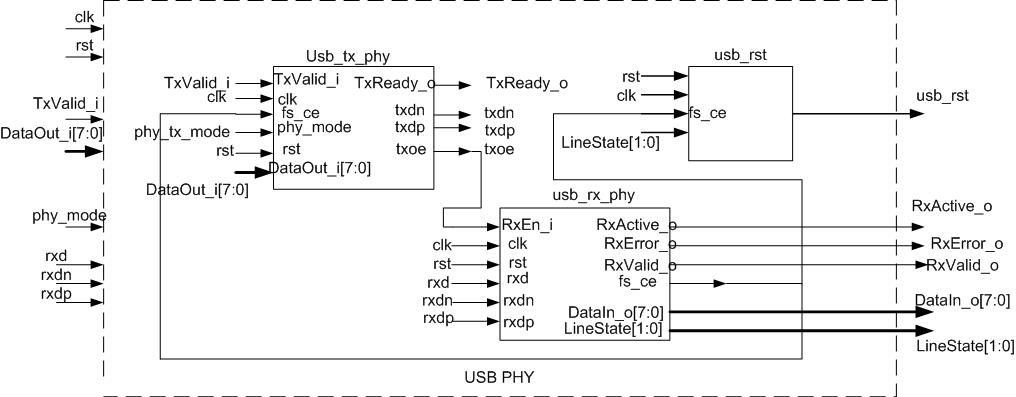


Figure 4.3: RTL model of the USB receive

The physical USB receiver at the physical level, in addition to the two basic transmission and reception blocks, can also have units for generating state control signals, as well as other signaling. Details of these signals will be discussed in detail in the next section. the next part.

## **4.4 Model of USB transmission block**

The figure below shows a specific USB PHY block at RTL. Level

Table

Description automatically generated

Figure 2.4 : Model of USB PHY transmission block with input and output signals

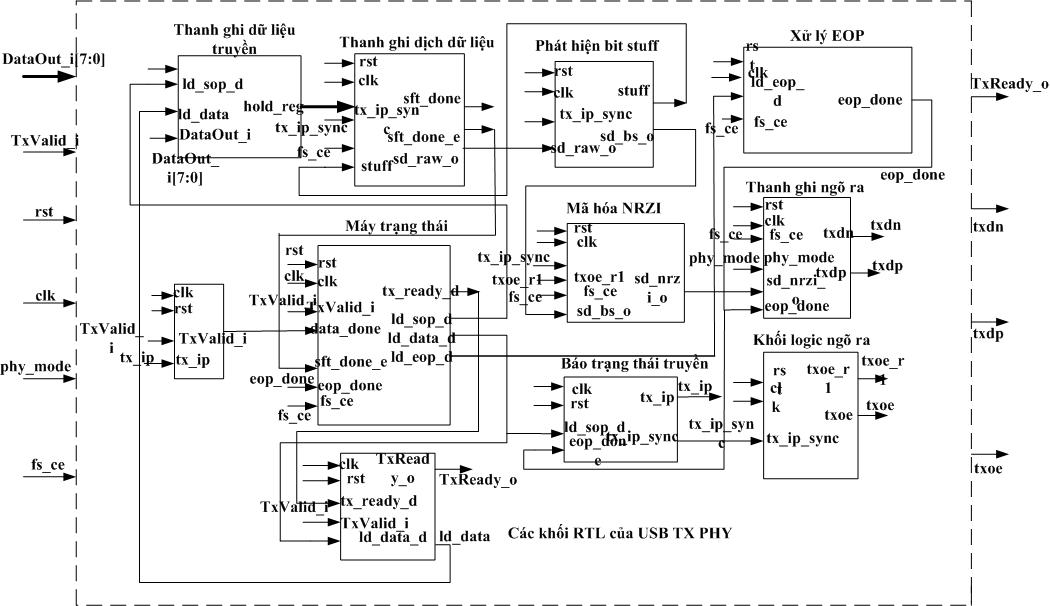


Figure 2.5: Specific function blocks of USB TX PHY

### **4.4.1 Description of USB TX block input and output signals**

a. Input signals

|  |  |  |
| --- | --- | --- |
| Name | Impact level | Decription |
| CLK | Edge up | A clock is supplied from the system, used for parallel data transmission and to create a synchronous clock for the internal system. |
| RST | High Level | Reset all state machines and registers. |
| PHY\_MODE | High Level | Select the operating mode: test or physical mode |
| FS\_CE | High Level | The sampling signal is synchronized between the two sides, controlled from the receiver side. |
| TxValid | High Level | Transmit Validation: Informs that the Data In data on the bus is valid. When this signal is active, the synchronization process will be initiated by the SYNC signal. When not enabled, the EOP signal will be transmitted. , signaling the end of the packet. |
| DataOut\_i[7:0] | N/A | Data In: 8-bit parallel USB input data. |

b. Output signals:

|  |  |  |
| --- | --- | --- |
| Tx\_Ready\_o | High level | Data transmission is ready. If TXValid = 1 and TxReady is active at the rising edge of CLK, the UTM will put DataIn from the bus into the Tx storage register at the next clock pulse. Otherwise if TxReady is not active, data will be saved on the bus. |
| txdp | N/A | Positive differential output |
| txdn | N/A | Negative differential output |
| txoe | High level | Signal that enables differential signal output to the transmission line. |

### **4.4.2 Description of function blocks:**

a.Transfer data register and shift block:

This block has the function of reading parallel data from the application side and translating to create serial data for USB communication. This block includes an 8-bit shift register for parallel/serial conversion and an 8-bit storage register to buffer the next data byte to be converted.

Diagram

Description automatically generated

Figure 2.6: Operation model of the data store/shift register block

The signals in the block are detailed below:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Direction | Impact level | Decriptions |
| Ld\_sop\_d | In | High level | Synchronous sequence register load enable signal to initiate transmission for data transfer |
| Ld\_data | In | High level | Register data read enable signal to initiate USB transmission |
| Tx\_ip\_sync | In | High level | Transmission in progress signal bit signaling current is the residual bit stuff |
| stuff | In | High level | Signal bit signaling current is the residual bit stuff |
| Sft\_done | Out | High level | The end of conversion flag. |
| Sft\_done\_e | Out | High level | Enable flag, recognize rising edge of sft\_done signal |
| Sd\_raw\_o | Out | N/A | Serial data output has been converted. |

b.Bit stuff processing block:

To ensure correct signal changes, the stuff bit is added when transmitting data over USB. An extra '0' bit is added to the stream when 6 bits are detected. '1' is continuous in the data stream before it is NRZI encoded, to ensure that there is a signal state change in the transmitted NRZI data stream.

c. NRZI Encryption Block:

Data after parallel / serial conversion and Stuff processing will be encoded as NRZI before transmission.

Features of NRZI encoding:

• Encountering bit '1' will reverse the previous voltage polarity.

• Encountering bit '0' will not reverse the previous voltage polarity

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Direction | Impact level | Decriptions |
| Txoe\_r1 | In | High level | Enables encoded signal to be sent out of the line |
| Sd\_bs\_o | In | High level | Aware of state changes in input data bit stream |
| Sd\_nrzi\_o | Out | N/A | Encoded serial output |

d. EOP processing block:

After the data has been transmitted, the end signal will be transmitted. The main EOP signal is a 2-bit delay for the SE0 signal (both D+ and differential outputs). D- can be reduced to low), and add 1 bit period of J state (D+ is high and D- is low) to synchronize the two sides at the end of transmission.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Direction | Impact level | Description |
| Ld\_eop\_d | In | High level | EOP Transmit Generation Enable |
| Eop\_done | In | High level | EOP Transmit Completion Flag. |

e.Output register block:

Output operates in two modes:

• Normal physical transmission mode: outputs encoded bits to the transmission line.

• Testing mode: Signaling the end of transmission of a packet.

g.Output logic block:

This block is responsible for generating the enable signal to output the encoded signal. Output 'toe' is active only high when the busy flag 'tx\_ip\_sync ' is not enabled. Output 'toe\_r1' is the signal. The delay signal of 'toe' is the synchronized signal.

Diagram

Description automatically generated

Figure 2.8: State machine model of a USB transmitte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initial state | Relay state | Impact signal | Out | Description |
| IDLE | SOP | TxValid\_i | ld\_sop\_d = 1 | When the TxValid signal indicates valid data on the bus, the ld\_sop\_d output is set high to start transmitting the SYNC synchronous packet. |
| SOP | DATA | sft\_done\_e | tx\_ready\_d = 1  ld\_data\_d = 1 | After the parallel/serial conversion is completed with the sft\_done\_e flag enabled, the tx\_ready\_d and ld\_data\_d outputs are raised high to start reading and putting data on the bus. |
| DATA | EOP1 | Data\_done = 0 và  Sft\_done\_e = 1 | Ld\_eop\_d = 1 | • If the data has been transmitted, the EOP signal will be transmitted in the next EOP1 state with the ld\_eop\_d signal  • If the data has not been transmitted, this state continues until completion. |
| EOP1 | EOP2 |  |  | The EOP signal undergoes 2 bit cycles to generate the SE0 . signal |
| EOP2 | WAIT | eop\_done &&  fs\_ce |  | The EOP signal terminates with one bit period with state J . |

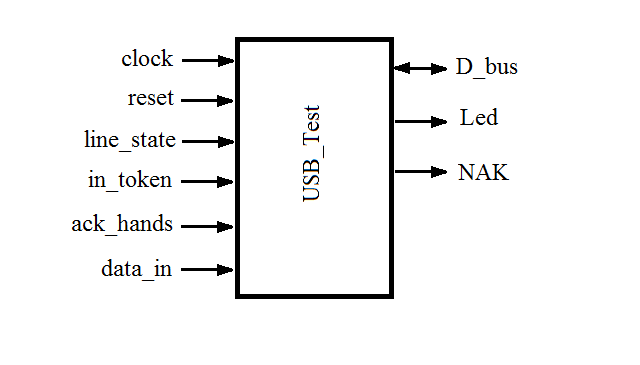
i.Transmission status message block:

During transmission, flags tx\_ip and tx\_ip\_sync will always be on. This signal is controlled by 2 signals starting ld\_sop\_d and ending eop\_done.

# **CHAPTER 5 WRITING CODE FOR USB TRANSFER AND SIMULATION**

## **5.1 Design of data transmitter**

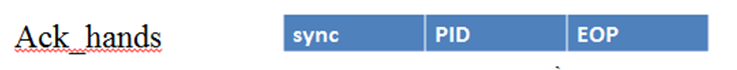
Model :

****

A screenshot of a computer

Description automatically generated with medium confidence

check



Text

Description automatically generated with medium confidence

## **5.2 Code for transmitter and simulation**

*Verilog Code:*

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: HCMUTE

// Engineer: TRAN QUANG NHAT, PHAN NHU KHOI

// Lecturer: DO DUY TAN

// Create Date: 11:42:09 05/24/2022

// Design Name: USB Transmission

// Module Name: USB1

// Project Name:

// Target Devices:

// Tool versions:

// Description:

// Dependencies:

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//////////////////////////////////////////////////////////////////////////////////

module USB1(

input CLOCK,

input RESET,

input [32:0] IN\_TOKEN,

input [16:0] ACK\_HANDS,

input [15:0] DATA\_IN,

output reg [15:0] D\_BUS,

output reg LED,

output reg [3:0] NAK

);

always @(posedge CLOCK)

begin

if(RESET == 1'b0)

begin

D\_BUS = 16'b0;

LED = 1'b0;

NAK = 4'b0;

end

else

if((IN\_TOKEN[24:17] == 8'b0001\_1110 )&&( ACK\_HANDS[8:1] == 8'b0010\_1101))

begin

D\_BUS = DATA\_IN;

LED = 1'b1;

NAK = 4'b0010;

end

else

begin

D\_BUS = 16'b0;

LED = 1'b0;

NAK = 4'b1010;

end

end

endmodule

*Test Bench Code:*

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////

// Company: HCMUTE

// Engineer: TRAN QUANG NHAT, PHAN NHU KHOI

// Lecturer: DO DUY TAN

// Create Date: 11:49:02 05/24/2022

// Design Name: USB1

//ModuleName: C:/Users/Mike/OneDrive/Documents/Xilinx\_ISE\_DS\_Win\_14.7\_1015\_1/projects/USB1/USB1\_tb.v

// Project Name: USB1

// Target Device: USB Transmission

// Tool versions:

// Description:

// Verilog Test Fixture created by ISE for module: USB1

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

////////////////////////////////////////////////////////////////////////////////

module USB1\_tb;

// Inputs

reg CLOCK;

reg RESET;

reg [32:0] IN\_TOKEN;

reg [16:0] ACK\_HANDS;

reg [15:0] DATA\_IN;

// Outputs

wire [15:0] D\_BUS;

wire LED;

wire [3:0] NAK;

// Instantiate the Unit Under Test (UUT)

USB1 uut (

.CLOCK(CLOCK),

.RESET(RESET),

.IN\_TOKEN(IN\_TOKEN),

.ACK\_HANDS(ACK\_HANDS),

.DATA\_IN(DATA\_IN),

.D\_BUS(D\_BUS),

.LED(LED),

.NAK(NAK)

);

initial begin

// Initialize Inputs

CLOCK = 0;

RESET = 0;

IN\_TOKEN = 0;

ACK\_HANDS = 0;

DATA\_IN = 0;

// Wait 100 ns for global reset to finish

#100;

RESET = 1'b1;

#100;

// Add stimulus here

IN\_TOKEN = 33'b00000001\_00011110\_00000010\_00110101\_0; // correct token, we want to read data, but this token is write data

ACK\_HANDS = 17'b00000001\_00101101\_1; // correct ACK feedback from USB peripherals

DATA\_IN = 16'b1001000111111111;

#100;

IN\_TOKEN = 33'b00000001\_10011111\_00000010\_00110101\_0; // Wrong token, we want to read data, but this token is write data

ACK\_HANDS = 17'b00000001001011011; // correct ACK feedback from USB peripherals

DATA\_IN = 16'b1001000101110110;

#100;

IN\_TOKEN = 33'b00000001\_00011110\_00000010\_00110101\_0; // correct token, we want to read data, but this token is write data

ACK\_HANDS = 17'b00000001\_00101101\_1; // correct ACK feedback from USB peripherals

DATA\_IN = 16'b1010010010010010;

#100;

RESET = 1'b0;

#100

RESET = 1'b1;

IN\_TOKEN = 33'b00000001\_00011110\_00000010\_00110101\_0; // correct token, we want to read data, but this token is write data

ACK\_HANDS = 17'b00000001\_10100101\_1; // incorrect ACK feedback from USB peripherals

DATA\_IN = 16'b1001000111111111;

end

always

begin

CLOCK = ~ CLOCK;

#10;

end

endmodule

Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated

# **CHAPTER 6 CONCLUSION**

The project has carried out research on USB communication, handle the topic requirements with a simple transmitter. However, the theory of USB standard is very large, it’s very difficult for my team’s current capabilities to complete most of USB communication standard. Therefore, the project only designs a small part or USB transmitter and stop at simulation level.

From these theories, we can make a competed USB, compliant with all USB communication standards. The project team would like to thank teacher for their guidance, especially Mr. Do Duy Tan directly guided the team to implement this project

Reference document:

[1]researchpaper-Embedding-Soft-processor-based-USB-device-driver-on-FPGA

USB\_Complete\_3rdEdition, Jan Axelson

[2]Introducing the Spartan 3E FPGA and VHDL, Author Mike Fields