International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 4 (2015) pp. 10623-10632
© Research India Publications
http://www.ripublication.com

Implementation of NRZI Encoding/Decoding in USB Host Controller

V. Elanangai
Department of E.E.E, Sathyabama University
OMR Road, Chennai-119.
elanangai123@gmail.com

Abstract

This paper introduces NRZI encoding and decoding of a USB device controller which accepts data from a client, and transfer it to a host computer through a 12Mb/s USB connection. The system design basically consist of serial interface engine which mainly consists of three parts, which are control unit, data path unit and error detection and correction block (CRC). After completing the packet generation of the given input fetched from the memory buffer then it was handled by the CRC generator and the bit stuffing is done. While receiving the data NRZI decoding, bit stuffing, SYNC and EOP bits are removed and CRC is calculated to verify the data. VHDL programming language is used for the coding and the final design will be downloaded to the Xilinx-Spartan3 FPGA toolkit.

Keywords: NRZI Encoding/Decoding, USB Host controller, and Xilinx Spartan3 FPGA tool kit.

Introduction

The personal computer has evolved into a powerful multimedia appliances over the last several years. Spurred by such advances as powerful Intel Microprocessors, advanced graphics subsystems, and highly capable software, the mainstream portion of the personal computer market segment has made impressive advances in multimedia capabilities. However, the PC has also continued to be plagued by an Achilles' heel—its unfriendly I/O subsystems. Users have continued to struggle with cryptic elements of the PC like IRQ, DMA, and I/O addresses. The universal Serial Bus (USB) should go a long way towards solving many of these problems and offers powerful new multimedia capabilities to help make the PC the ubiquitous multimedia appliances.

All communications on USB originates at the host under software control. The host hardware consists of the USB host controller, which initiates the transactions

over the USB system [2] and the root hub, which provides attachment points for USB device. The host controller is responsible for generating the transactions that have been scheduled by the host software. The host controller driver builds a linked list of data structures in memory that defines the transactions that are scheduled to be performed during a given frame. These data structures, called transfer descriptors, contain all of the information the host controller needs to generate the transactions. This information includes : USB Device Address, Type of Transfer[1], Direction of Transfer, Address of the Device Driver's Memory Buffer.

The host controller in Fig.1 performs writes to a target device by writing data from a memory buffer that is to be delivered to the target device. The host controller performs a parallel to serial conversion on the data, creates the USB transaction, and forwards it to the root hub to send over the bus. Similarly if a read transfer is required, the host controller builds the read transaction and sends it to the root hub. The hub transmits the read transaction over the USB. The target device recognizes that it is being addressed, the device then transmits data back to the root hub that forwards the data on to the host controller. The host controller performs the serial to parallel conversion on the data and transfer the data to the device driver memory buffer.

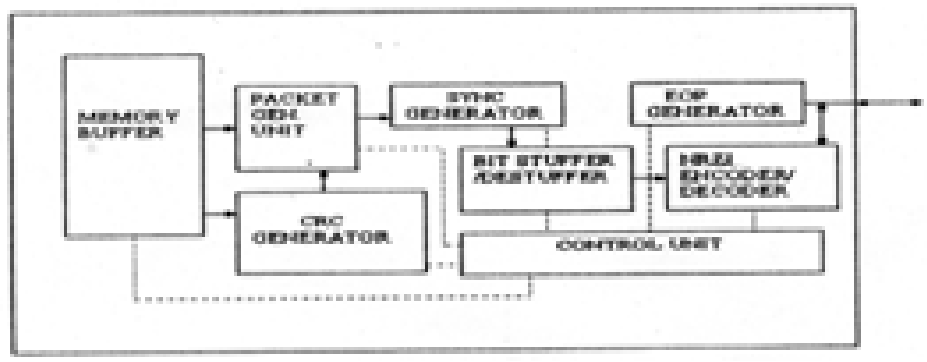


Figure 1: Host Controller

Design of Usb System

A. USB system

USB system includes host controller with hub interface. Fig.2 illustrates the above mentioned blocks.

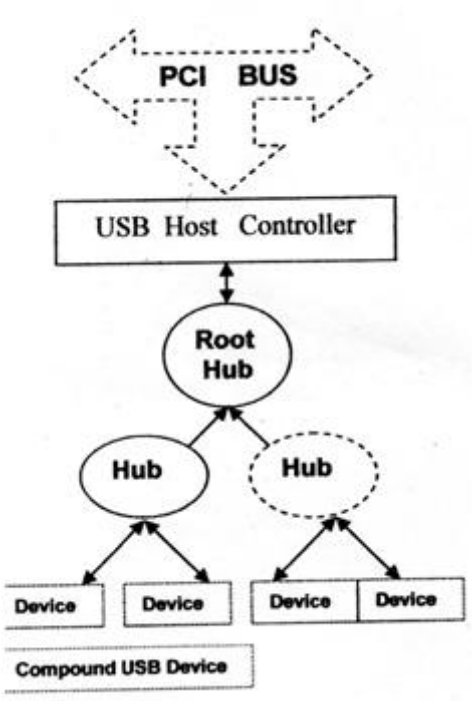


Figure 2: Block diagram of USB System

The SIE [5] shown in Fig.3 is the front end of this hardware and handles most of the protocols. The SIE typically comprehends signaling up to the transaction level.

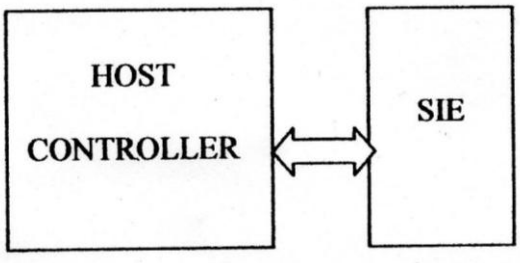


Figure 3: USB bus interface

B. Functioning of USB system

The functions that it handles could include, Packet recognition, transactions sequencing, SOP, EOP, RESET, RESUME signal detection/generation, Clock/ Data separation, NRZI Data Encoding/decoding and bit-stuffing[1], CRC generation and checking(Token and Data), Packet ID (PID) generation and checking/decoding, Serial-Parallel/Parallel-Serial conversion.

Error Handling

Isochronous transfers provide no data packet retries that is no handshakes are returned to a transmitter by a receiver. So that timeless of data delivery is not perturbed. However, it is still important for the agents responsible for data transport to know when an error occurs and how the error affects the communication flow. In particular, for a sequence of data packets [3] (A,B,C,D), USB allows sufficient information such that a missing packet (A,_,C, D) can be detected and will not unknowingly be turned into an incorrect data or time sequence (A,C,D or A,_,B,C,D). The protocol provides four mechanisms that support this, exactly one packet per frame, SOF, CRC, and bus transaction timeout.

Isochronous transfer requires exactly 1 data transaction every frame for normal operation. USB does not dictate what data is transmitted in each frame. The data transmitter/source determines specifically what data to provide. This regular data per frame provides a framework that is fundamental to detecting missing data errors. Any phase of a transaction can be damaged during transmission on the bus. Since every frame is preceded by an SOF packet and a receiver can see SOfs on the bus, a receiver can determine that its expected transaction did not occur between two SOFs. Additionally, since even an SOF packet can be damaged, a device must be able to reconstruct the existence of a missed SOF.

A data packet may be corrupted on the bus, therefore, CRC protection allows a receiver to determine that the data packet it received was corrupted. Finally, the protocol defines the details that allow a receiver to determine via bus transaction timeout that it is not going to receive its data packet after it has successfully seen its token packet. Once a receiver has determined that a data packet was not received, it may need to know the size of the data that was missed in order to recover from the error with regard to its functional behavior. If the communication flow is always the same data size per frame, then the size is always a known constant. However, in some cases the data size can vary from frame to frame. In this case, the receiver and transmitter have an implementation dependent mechanism to determine the size of the lost packet.

In summary, whether a transaction is actually moved successfully over the bus or not, the transmitter and receiver always advance their data/buffer streams one transaction per frame to keep data per time synchronization. The detailed mechanisms described above allow detection, tracking, and reporting of damaged transactions so that a function or its client software can react to the damage in a function appropriate fashion. The details of that function/application specific reaction are outside the scope of the USB specification.

Non Return To Zero-Invert (NRZI)

C. Encoding in USB

USB data packets are encoded using NRZI. Fig.4 illustrates the steps involved in information transfer. NRZI encoding is first done by the USB agent that is sending information. Next, the encoded data is driven onto the USB cable by the differential

driver. The receiver amplifies the incoming differential data and delivers the NRZI data to the decoder. Encoding and differential signaling are used to help ensure data integrity and eliminate noise problem, without requiring a separate clock signal be delivered with the data. NRZI [12] is by no mean a new encoding scheme. It has been used for decades in a wide variety of applications.

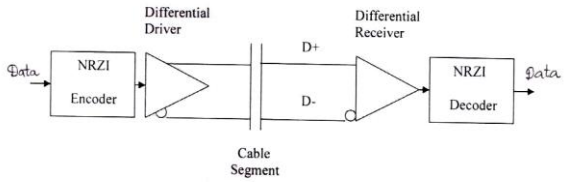


Figure 4: Transfers across USB cable employs NRZI Encoding and Differential Signalling

Fig.5 illustrates a serial data stream and the resulting NRZI data. Zero's in NRZI represented by transitions while 1s are represented by the absence of a transition.

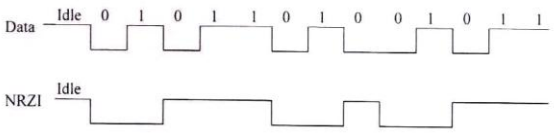


Figure 5: NRZI Encoded data

The NRZI encoder must maintain synchronization with the incoming data stream to correctly sample the data. The NRZI data stream must be sampled within a data window to detect whether a transition has occurred since the previous bit time. The decoder samples the data stream during each bit time to check for transition. Fig.6 shows data transition. First the data will be fetched from the memory. After fetching, the checking will take place bit wise. It will check for the binary bits 0 and 1. If the data fetched is 0, then transition of the data occurs. Otherwise, if it is the reverse that is if the data fetched is the bit 1, then transition does not occur. Likewise, all the bits in any particular packet from the memory will be checked for its transfer. If the above mentioned process is over for all the bits in a packet, then next packet will be fetched and the same process continues.

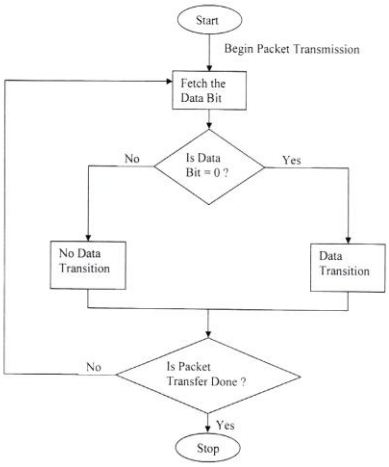


Figure 6: Flow diagram of NRZI

NRZI encoder must maintain synchronization with the incoming data stream to correctly sample the data. The NRZI data stream must be sampled within a data window to detect whether a transition has occurred since the previous bit time. The decoder samples the data stream during each bit time to check for transitions. Transitions in the data stream permit the decoder to maintain synchronization with the incoming data, thereby eliminating the need for a separate clock signal. However that a long string of consecutive ones result in no transitions, causing th receiver to eventually lose synchronization. The solution is to employ stuffing.

D. Decoding in USB

Decoder function is the inverse of the Encoder. Unlike encoding, in decoding the transition takes place only when the data bit is 1. No transition takes place when the data bit is 0. The encoded data is received by the receiver. The receiver ‘s function is the inverse of the transmitter. The receiver is more complicated than the transmitter and requires further functionality.

The receiver has the capability to calculate both 5-bit CRC and16-bit CRC. This is necessary since the receiver receives token packets, which contains a 5-bit CRC and data packets, which contains a 16-bit CRC. The transmitter compares the value calculated with that received from the packets. If there is any inconsistency, retransmission is requested from the host.

Execution of Code

Each and every component used are tested. This operation allowed the USB controller to receive and transmit packets without a problem. The controller also set the output enable signal of the transceiver high when transmitting data. It sets output enable to low when receiving data. The receiver decoded the NRZI code, from the USB bus, to binary data. The stuffed bit of this binary data was removed correctly. The SOP detector detects the start of packet promptly and advised the controller. The controller

then enabled the receiver in time to accept the next morning bit. The receiver executed the serial-to –parallel conversion of the binary data correctly.

The transmitter carried out conversion of parallel data into a bit stream to be sent on the USB bus. The bit stream had a stuffed bit added to it at the correct positions. The transmitter executed NRZI encoding correctly.

E. Flow charts

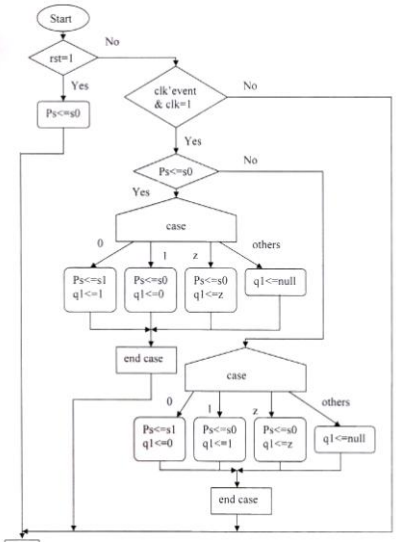


Figure 7: Flow chart of NRZI Encoding

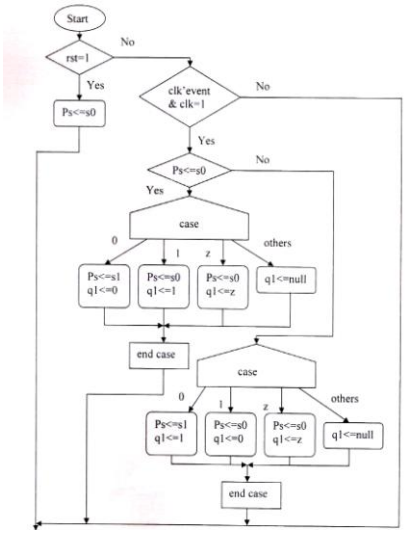
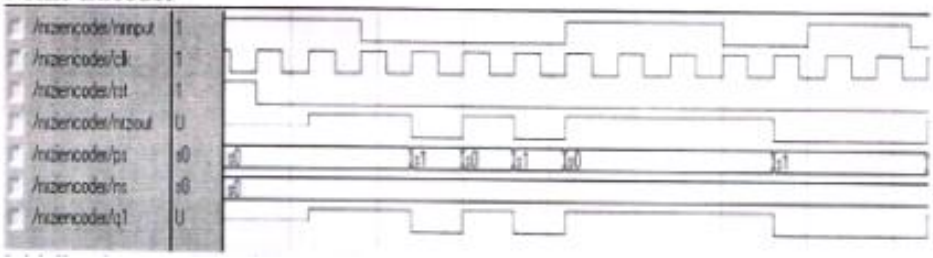


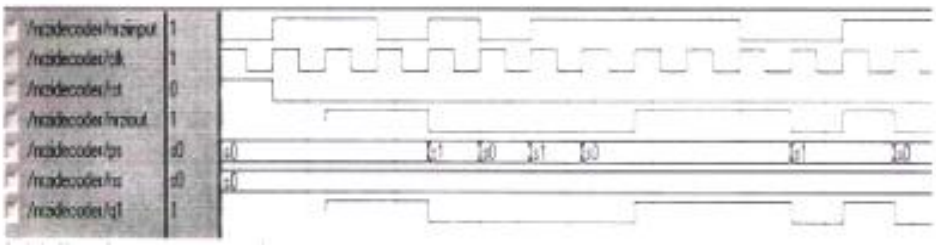
Figure: Flow chart of NRZI Decoding

Simulation Results

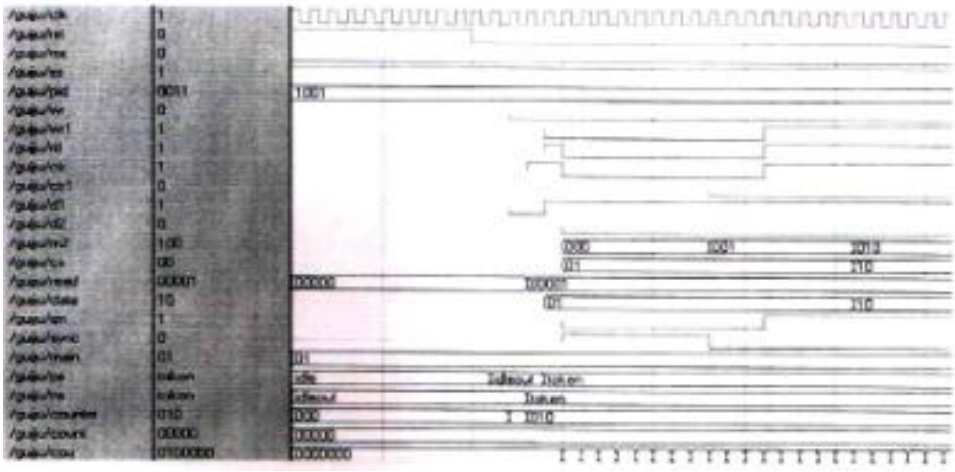
NRZI Encoder



NRZI Decoder



USB Host Controller



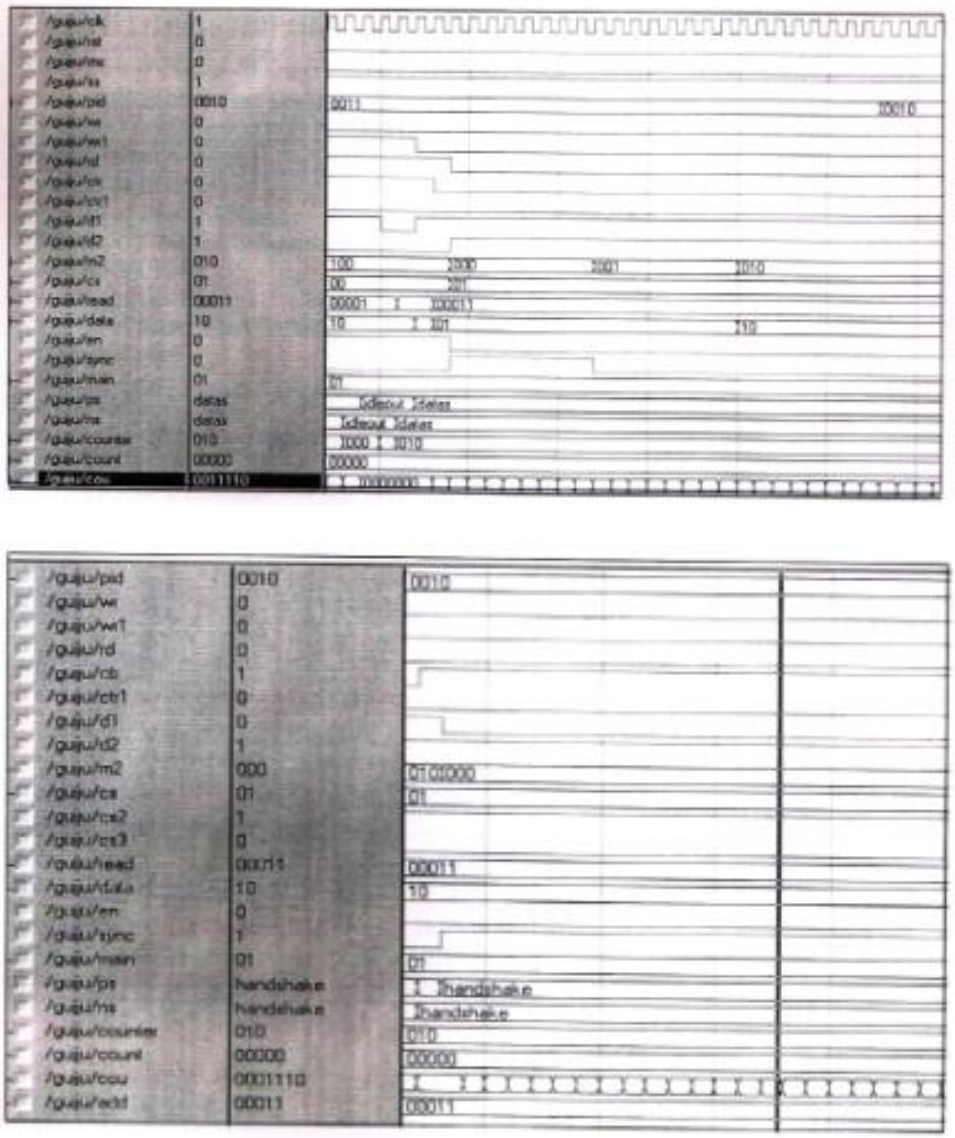


Table 1: Device Utilization Summary

RESOURCE	USED	AVAIL	UTILIZATION
IOs	4	140	2.86%
Function Generators	3	1536	0.20%
Slices	2	768	0.26%
Dffs or Latches	2	2082	0.10%

Conclusion

This was developed in VHDL and the target technology is FPGAs. Thus, NRZI Encoding and Decoding was developed and integrated with other modules to form the

complete USB. The top modules of host controller and the root hub were integrated. The resulting system works on simulation and synthesis.

References

[1] D. Anderson, D. Dzatko, "Universal Serial Bus System Architecture,". MidShare, Inc., 2001.

[2] J.Axelson, "USB COMPLETE Second Edition," Madison, WI: Lakeview Research LLC, 2004.

[3] Future Technology Devices International Ltd; "USB Data Packet Structure".

[4] John Hyde, "USB Design by Example", Published by John Wiley & Sons Inc.

[5] USB specification 'USB Serial interface engine (SIE)" from www.usb.org.

[6] Whats USB webpage, http://www.pulsewan.com/data101/usb_basics.htm

[7] LearnUSBbyDoing webpage,www.devasys.com/PD11x/JHWP.

[8] Dinah Ann Varughese, "Transmission and Distribution of data through USB using FPGA", IJRET, March 2014,vol 03, eISSN: 2319-1163 | pISSN: 2321-7308.

[9] S.Brown, Fundamental of Digital Logic with VHDL Design, Mc Graw-Hill,2000.

[10] Full Speed USB 1.1 Fuction Controller, Trenzelectronics,2000.

[11] Wikipedia, the Free Encyclopedia."NRZI" <http://en.wikipedia.org>.

[12] USB 2.0 Transceiver Macrocell Interface (UTML) Specification,2000.

[13] Signal Encoding, NRZI www.wildpackets.com.