INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

Design And Implementation Of RS232 To Universal Serial Bus Protocol Converter Using FPGA

V. Madhurima
Assistant Professor in the Dept. ECE, S.V. College of Engineering, A.P, India
N.Suguna
Assistant Professor in the Dept. ECE, S.V. College of Engineering, A.P, India

Abstract:
Universal Serial Bus (USB) is a new personal computer interconnection protocol, developed to make the connection of peripheral devices to a computer easier and more efficient. It reduces the cost for the end user, improves communication speed and supports simultaneous attachment of multiple devices (up to127)RS232, in another hand, was designed to single device connection, but is one of the most used communication protocols. An embedded converter from RS232 to USB is very interesting, since it would allow serial-based devices to experience USB advantages without major changes. This work describes the specification and development of such converter and it is also a useful guide for implementing other USB devices. The main blocks in the implementation are USB device, UART (RS232 protocol engine) and interface FIFO logic. The USB device block has to know how to detect and respond to events at a USB port and it has to provide a way for the device to store data to be sent and retrieve data that have been received UART consists of different blocks which handle the serial communication through RS232 protocol. There are a set of control registers to control the data transfer. The interface FIFO logic has FIFO to bridge the data rate differences between USB and RS232 protocols.

Key words: *First-In-First-Out, RS-232, Universal Asynchronous Receive Transmit, Universal Serial Bus.*

1.Inroduction

This paper describes the specification and implementation of a converter from RS232 to USB (Universal Serial Bus). This converter is responsible for receiving data from a peripheral device’s serial interface and sending it to a computer’s USB interface. In the same way, it must be able to send data from the PC’s USB interface to the device. The problems faced with the old standards stimulated the development of a new communication protocol, which should be easier to use, faster, and more efficient. RS232 is a definition for serial communication on a 1:1 base. RS232 defines the interface layer, but not the application layer. To use RS232 in a specific situation, application specific software must be written on devices on both ends of the connecting RS232 cable. RS232 ports can be either accessed directly by an application, or via a device driver in the operating system.USB is a new personal computer interconnection standard developed by industry and telecommunication leaders, which implements the Plug and Play technology. It allows multiple devices connection (up to 127) ranges. The use of a the devices attachment to PCs. USB is a low cost and easing solution comprehending the low-speed and mid-speed data converter from a serial interface to USB would free a serial communication port to other applications, allowing a device that uses a serial interface to communicate using an USB interface. USB on the other hand is a bus system which allows more than one peripheral to be connected to a host computer via one USB port. Hubs can be used in the USB chain to extend the cable length and allow for even more devices to connect to the same USB port. The standard not only describes the physical properties of the interface, but also the protocols to be used. Because of the complex USB protocol requirements, communication with USB ports on a computer is always performed via a device driver. This way, we are not limited to the availability of a serial port and we can experience the USB advantages. Using a converter allows us to have the device unchanged, making the converter responsible for treating the differences between the protocols. This work was based on protocol engine which can be managed by exchanging data with a PC across a serial interface. Most of the times, this communication is not done constantly, since it is necessary to have a serial port available just for it. This paper presents the converter implementation, focusing on the development process, which comprehends the device itself and the PC-side software that will communicate with it. This methodology can be extended to other devices. We first present some important USB standard concepts. Then, we define the system specification, divided on host and device requirements. After, we describe the hardware

(UART) features and software design and implementation. Finally, we discuss about achieved results and future work

2.Problem Description

The USB specification describes bus attributes, protocol definition, programming interface and other features required to design and build systems and peripherals compliant with the USB standard. We briefly explain features used in our project. USB devices can be functional (displays, mice, etc) or hubs, used to connect Other devices in the bus. They can be implemented as low or high-speed devices. Low-speed devices are limited to maximum 1.5 Mb/s rate.Each device has a number of individual registers - known as Endpoints which are indirectly accessed by the device drivers for data exchange. Each endpoint supports particular transfer characteristic has a unique address and direction. A special case is Endpoint 0, which is used for control operations and can do bi-directional transfers. It must be present in all devices. According to the device's characteristics, other types of endpoints can be defined. USB Host verifies the attachment and detachment of new devices, initiating the enumeration process and managing all the following transactions. It is responsible to install device driver (based on information provided by device descriptors), to automatically reconfigure the system (hot attachment) and to collect statistics and status of each device. USB on the other hand is a bus system which allows more than one peripheral to be connected to a host computer via one USB port. Hubs can be used in the USB chain to extend the cable length and allow for even more devices to connect to the same USB port. The standard not only describes the physical properties of the interface, but also the protocols to be used. Because of the complex USB protocol requirements, communication with USB ports on a computer is always performed via a device driver. Device's descriptors specify USB devices attributes and characteristics and describe device communication requirements (Endpoint Descriptors). The USB host uses this information to configure the device, to find its driver, and to access it. Devices with similar functions are grouped into classes [1, 2] in order to share common features and even use the same device drivers. Each class can define their own descriptors (class-specific descriptors), as for example, HID (Human Interface Device) Class Descriptors and Report Descriptors. The HID class consists of devices used by people to control computer systems. It defines a structure that describes a HID device, with specific communication requirements. According to the converter characteristics, it can be implemented as a HID device, using already developed HID drivers. A HID device's descriptors must support an Interrupt IN endpoint and the firmware must also contain a report descriptor that defines the format for transmitted and received device data.

2.1.Requests

The USB protocol is based on requests sent by the host and processed by the USB devices. These requests can be directed to a device or a specific endpoint in it. Standard requests must be implemented by all devices and are used for configuring a device and controlling the state of its USB interface, among other features. Two HID-specific requests must be supported by the converter: Set Report and Get Report.

These requests enable the device to receive and send generic device information to the host. Set Report request is the only way the host can send data to a HID device, once it does not have an Interrupt OUT endpoint

2.2.Communication Flow

USB is a shared bus and many devices might use it at the same time. The devices share the bandwidth using a protocol based on tokens and commanded by the host. USB communication is based on transferring data at regular intervals called frames. A frame is composed by one or more transactions. USB data transfers are typically originated by a USB Device Driver when it needs to communicate with its device. It supplies a memory buffer used to store the data in transfers to or from the USB device. The USB Driver provides the interface between USB Device Driver and USB Host Controller, translating transfer requests into USB transactions. Some of these transfers consist of a large block of data, which need to be splitted into several transactions. The Host Controller generates the transaction based on the Transfer Descriptor, which describes the frame sharing among the several devices requests. When a transaction is sent to the bus, all devices see it. Each transaction begins with a packet that determines its type and the endpoint address. The USB driver controls this addressing scheme. Inside the device, the USB Device Layer comprehends the actual USB communication mechanism and transfer characteristics. USB Logical Device implements a collection of endpoints that comprise a given functional interface, which can be manipulated by its respective USB client.

2.3.Transfer Types

The USB specification defines four transfer types: Control, Interrupt, Isochronous and Bulk. Control transfers send requests and data relating to the device's abilities and configuration. They can also be used to transfer blocks of information for any other purpose. Control transfers consist of a Setup stage, followed by a Data stage, which is composed of one or more Data transactions, and a Status stage. All data transactions in a Data Stage must be in the same direction (In or out). Interrupt transfers are typically used for devices that need to transfer data at regular period of time, and consequently must be polled periodically. The polling interval is defined in the end point Descriptor. The data deployed for this kind of transfer for low-speed devices is 8 bytes. Error correction is done in this kind of transfer. Two other transfer types are Isochronous and Bulk**,** which are used for devices that need a guaranteed transfer rate or for large blocks of data transfers. They are not used in this work.

3.Procedure/Algorithm

3.1.System Specification

To develop a USB peripheral we need all the following: A host that supports USB. Driver software on the host to communicate with the peripheral. An application executing in the host that communicates with the peripheral device. A UART with a USB interface. Code implementation on the USB controller to carry Out the USB communication. Code implementation on the USB controller to carry out the peripheral functions. The UART& FIFO used to store sent and received data in the USB communication process.

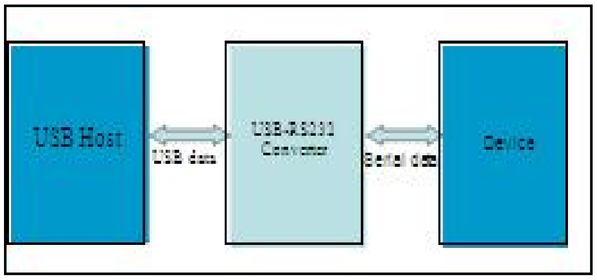

Figure 1: RS232 to USB Converter

3.2.Host Requirements

The choice of the Operating System used by the host was done in 1999, based on the USB support it provides. It should provide the entire drivers infrastructure and support the protocol characteristics, as for example, Plug and Play. The host must be able to receive USB data using its device drivers and make them available to the applications that have done the request. It is essential that we have a driver in the host to process USB transfers, recognizing the device, receiving and sending data to a USB device.

3.3.Device Requirements

Some communication requirements, such as transmission speed, frequency and amount of data to be transferred, were essential in communication the process of defining the UART be used. Considering the speeds available for USB devices, it was clear that the converter could be implemented as a low speed device, Considering the amount of data transferred and the transmission frequency, the converter was defined to use Interrupt transfers, a transfer type where considerable amounts of data must be transferred in pre defined amounts of time. The host is responsible for verifying if the device needs to transmit data from time to time. Interrupt transfers can be done in both directions, but needs to transmit data from time to time. Interrupt transfers can be done in both directions, but not at the same time. For the converter, they could be used to send and receive data from the PC. The Operating System provides HID drivers that allow us to use this transfer type. The maximum packet size for one transaction is 8 bytes for low speed devices. If we are sending larger amounts of data, they need to be splitted into many transactions, once USB is a shared bus. Another feature defined for the converter was the number of endpoints needed. As explained before, endpoints are buffers

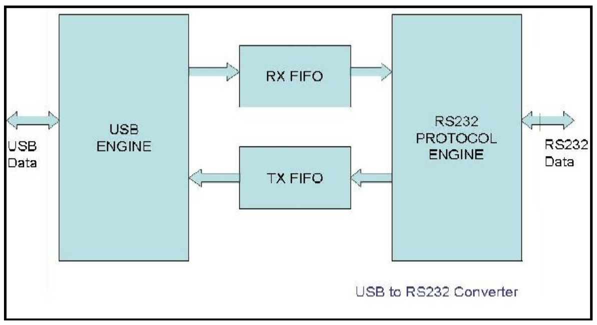

Figure 2: RS232 to USB Interface Diagram

3.Hardware Description

It is a low-cost solution for low-speed applications with high I/O requirements. RS232 ports which are physically mounted in a computer are often powered by three power sources: +5 Volts for the UART logic, and -12 Volts and +12 Volts for the output drivers. USB however only provides a +5 Volt power source. Some USB to RS232 converters use integrated DC/DC converters to create the appropriate voltage levels for the RS232 signals, implementations, the +5 Volt voltages is directly used to drive the output. The UART has serial interface to the RS232 driver. The operation of UART is controlled by an external host processor. There is an 8-bit data interface to host along with read and write control signals. Clock is fed from external cryatal.

The choice of a UART with three endpoint was done in order to allow us to have, beyond the Interrupt IN, an Interrupt OUT endpoint for receiving data from the host (OUT). Its definition requires we have an odd endpoint number besides Endpoint 0. This configuration could not be implemented at the time the project was being developed once the Operating System did not offer support for Interrupt OUT endpoints, which were defined in a later version of the specification. The instruction set has been optimized specifically for USB operations, USB controller provides one USB device address with three endpoints. The USB device address is assigned to the device and saved in the USB Device Address Register during the USB enumeration process. The USB controller communicates with the host using dedicated FIFO.

4.Software Design And Implementation

The development of the converter was divided in to modules:

USB host module, USB data exchange module, UART & FIFO module, interfacing UART and FIFO with USB for data exchange.

4.1.The Process Of Sending And Receiving Data

The process of sending data to the UPS is done through Control Transfers using SET REPORT on Endpoint 0. The host sends a request to the USB device, indicating it wants to send data. An interrupt informs the device when new data have arrived on Endpoint 0 and the corresponding Interrupt Service Routine copies it into a data buffer, which is used in the serial communication process.. The maximum packet size that is received from the host was defined according to the largest command that must be sent to the function must be changed to allow receiving an arbitrary number of bytes. These routines are called after the Host or the controller sends a packet to the bus. Endpoint 0 ISR receives. Using hardware flow control implies that more lines must be present between the sender and the receiver, leading to a thicker and more expensive cable. Therefore, software flow control is a good alternative if it is not needed to gain maximum performance in communications. Software flow control makes use of the data channel between the two devices which reduces the bandwidth. The reduce of bandwidth is in most cases however not so astonishing that it is a reason to not use it. First, the computer sets its RTS line to signal the device that some information is present. The device checks if there is room to receive the information and if so, it sets the CTS line to start the transfer. When using a null modem connection, this is somewhat different. There are two ways to handle this type of handshaking in that situation. One is, where the RTS of each side is connected with the CTS side of the other. In that way, the communication protocol differs somewhat from the original one. The RTS output of computer A signals computer B that A is capable of receiving information, rather than a request for sending information as in the original configuration. This type of communication can be performed with a null modem cable for full handshaking. Although using this cable is not completely compatible with the original way hardware flow control was designed, if software is properly designed for it it can achieve the highest possible speed because no overhead is present for requesting on the RTS line and answering on the CTS line. In the second situation of null modem communication with hardware flow control, the software side looks quite similar to the original use of the handshaking lines. The CTS and RTS lines of one device are connected directly to each other. This means, that the request to send query answers itself. As soon as the RTS output is set, the CTS input will detect a high logical value indicating that sending of information is allowed. This implies that information will always be sent as soon as sending is requested by a device if no further checking is present. To prevent this from happening, two other pins on the connector are used, the data set ready DSR and the data terminal ready DTR. These two lines indicate if the device attached is working properly and willing to accept data. When these lines are cross-connected (as in most null modem cables) flow control can be performed using these lines. A DTR output is set, if that computer accepts incoming characters.

5.Result Analysis

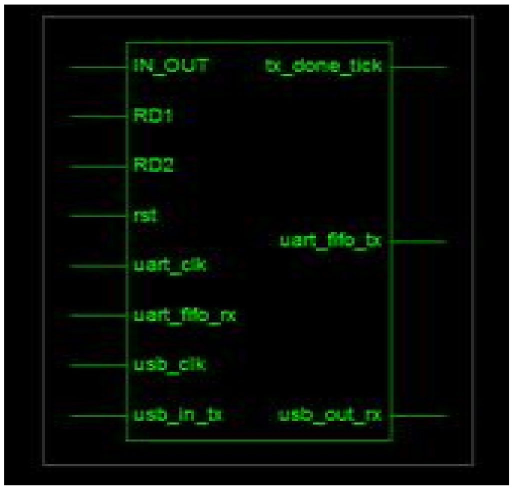

Figure 3:Schematic Result

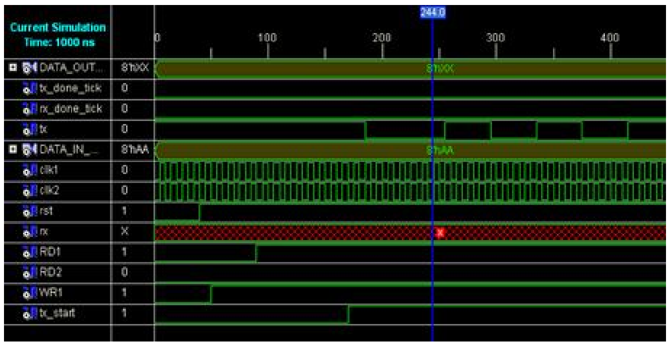

Figure 4: Shows the Waveforms of RS232USBconverter

6.Conclusion

An embedded converter from RS232 to USB is designed in this project. Verilog will be used for implementing all these blocks. Xilinx ISE 9.2i software will be used for functional simulation of the design. This converter will reduce the cost for the end user, improves communication speed and supports simultaneous attachment of multiple devices (up to 127). FPGA implementation of the design is done on Spartan 3E FPGA (XC3S500E).

7.Reference

1. Ana Luiza de Almeida Pereira Zuquim, Claudionor **JosC** Nunes Coelho Jr, Antanio Ot6vio Fernández, Marcos **PCgo** de Oliveira, AndrCa Iabrudi Tavares, “An Embedded Converter from RS232 to Universal Serial Bus”, IEEE
2. Jan axelson, “USB Complete, Everything you need to develop custom USB peripherals”, Penram Intl. Publishing(India), 1999
3. Universal Serial Bus Specification Revision 2.0
4. http://www.usb.org
5. Charles H.Roth, Jr, “Digital Systems Design using VHDL”, PWS publishing company, 1996.
6. ZainalabediNavabi,“VHDL Analysis and Modelling of Digital Systems”, McGraw – Hill, Second Edition.
7. http://www.lvr.com
8. http://www.usbstuff.com
9. Douglas L. Perry ,”VHDL”, Second Edition, McGraw-Hill, Inc, 1993
10. http://www.mrgadget.com.au/catalog/targus-usb-to-parallel-adapter-p-1160.html
11. USB Complete: The Developer's Guide, 4th Edition
12. USB Mass Storage: Designing and Programming Devices and Embedded Hosts14. FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version. Pong P.Chu