A Real Time Implementation of Serial Communication between Graphical User Interface and Simulator Board Using RS-232

M. Srilatha,
Digital System Engineering (M.E),
ECE Department,
University College of Engineering (A),
Osmania University, Hyderabad,
Andhra Pradesh -500 007, India.

CH. Tejasree,
VLSI System Design (M.tech),
ECE Department,
CVSR College of Engineering,
JNTU-H, Hyderabad,
Andhra Pradesh- 501 301 India,

SV. Kishore,
Embedded System (M.tech),
ECE Department,
Gudlavalleru Engineering College
JNTU Kakinada
Andhra Pradesh-521 356 India,

P.Chandrasekhar,
Assistant Professor,
ECE Department,
University College of Engineering (A),
Osmania University, Hyderabad,
Andhra Pradesh -500 007, India.

S.R.Pankaj Kumar *Scientist.'F',*
Defence Electronics Research Laboratory (DLRL),
Ministry of Defence, Government of India
Chandrayangutta lines,
Hyderabad – 500 005, India.

Abstract— **Virtex Series FPGA are used for establishing communication to the other sub units of an system. In order to establish the link, RS232 is used. The objective of this work is to establish link between simulator board and RS232. The work is broadly segregated into two major steps. In step I, a Graphical User Interface (GUI) is designed and developed based on Visual C++/Visual Basic Platform which enables the selectivity of data. The selected data is encoded and coded data is sent to the simulator board using an RS232 serial link. Step I is processed using PC. In step II, the coded data is decoded to identify the corresponding parameter data and it finally generates the required data. This generation of data is carried out by developing a code using VHDL language. The Step II is processed by using Power processor. The two steps are finally integrated and the output is seen at the GPIO (General Purpose Input Output) pins.**

Key words— **Simulator board, GUI, XPS, Power processor, Digital Glue Logic, Intellectual property (IP), ACK, NACK, RS232, GPIO.**

I. INTRODUCTION

Virtex family provides most powerful features in the FPGA[1]. It contains distinct platforms. Each platform contains a different ratio of features. One of the functions of the Simulator board is to accept the data from Graphical User Interface (GUI) which will be in frame format according to protocol, which is sent on RS232 link. Simulator board receives the data and displays received data in GUI on RS232 link.

Fig 1.1 shows the connection between the GUI and simulator board. The Simulator board after reception of data will decode the data message, acknowledges the GUI, and sends the received data to the digital glue logic. Digital glue logic will process the data, according to the received data and converts into the required output format and finally to respected IO pins. The main aim of this paper is to establish a serial communication between simulator board and Graphical User Interface (GUI). The paper is segregated into two activities which will be carried out in two steps.

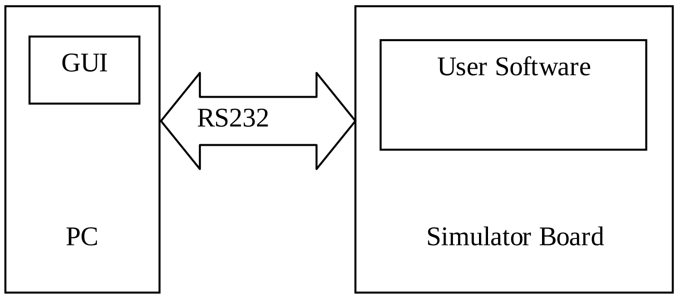

Fig: 1.1 Connection between the GUI and Simulator Board

STEP1: A GUI has to be designed and developed based on Microsoft Visual C# Platform [8] provides the user with selection of data and converts them into frames according to the protocol and sends to the developed user software on RS232 link.
STEP2: Development of User Software. This can be done by using Xilinx Platform Studio [5] for required simulator board.

GUI acts an Interface between the simulator board and user. It is developed in such a way that the data is processed and output will be shown and as desired.

II. GRAPHICAL USER INTERFACE

The function of Graphical User Interface is to generate the appropriate selected data to user software as done by the Simulator board. Every Selected data from GUI will be executed by the user software. Input of the GUI is selection of data by user, which is to be processed further. The output is a frame, which has to send on RS-232 link.

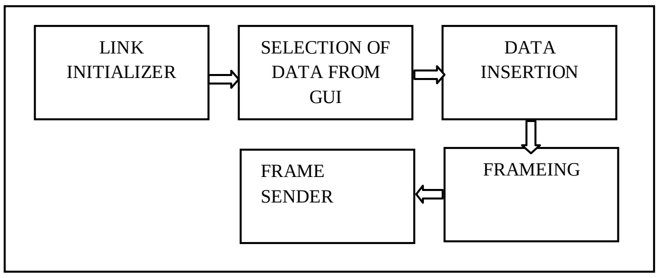

Fig. 2.1 Block diagram of Graphical user Interface

The above Fig.2.1 shows the block diagram of GUI, Link initialization process is implemented same as it implemented in the development process of user software. If the link is ok, then GUI will let the user to select data and then to insert data appropriate to the selected data.

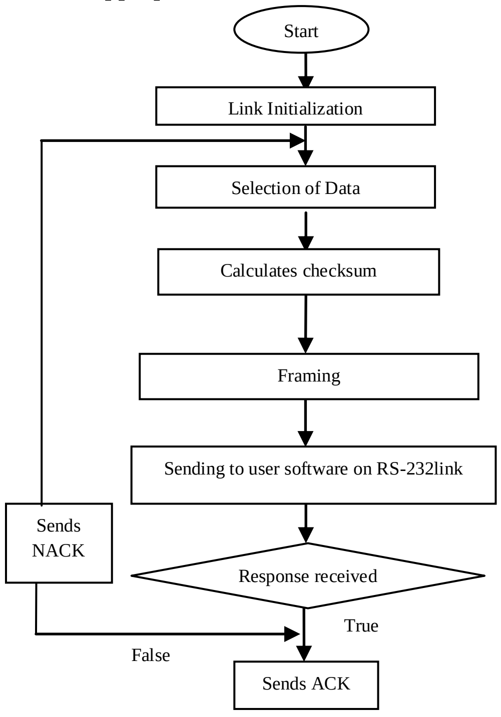

Fig. 2.2 GUI flow diagram

Fig.2.2 represents the Graphical User Interface flow diagram. Once the data is selected by user, then GUI sends the information to the user software in a frame format on RS-232 link. The frame structure will be according to the protocol. After data is provided by user at first it is encoded and then given to the framing block. Once encoding is done, and then selected code and data count fields are added to the data field. Next checksum is calculated by performing XOR operation performed in byte by byte process and next start of frame and end of frame bytes are added, which completes the framing process.

The framed data is sent to the Simulator board, and waits for the acknowledgement. GUI gets acknowledgement ACK or no acknowledgement NACK.
Design and development process of GUI is done by using Microsoft Visual C# tool.

III. USER SOFTWARE DEVELOPMENT

The selected data is input to this user software from GUI, and the output is generation of given parameter data on external IO pins.

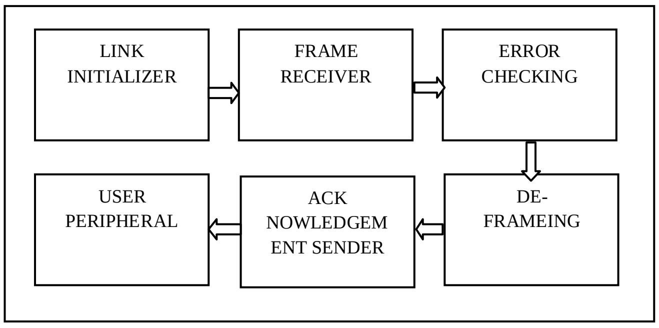

Fig. 3.1 Block diagram of user software.

The above Fig.3.1 shows block diagram of user software, initially it checks for Link initialization, if link is established, board looks for Start of Frame (SOF). Assembly of frames starts only after reception of a SOF character until End of Frame (EOF) is detected. After receiving the total frame error checking will be done by comparing received checksum by calculated checksum. Checksum is calculated by doing bitwise XOR of received frame data bytes (byte by byte) except SOF and EOF. After Calculation of checksum, an Acknowledgement ACK or NACK will be sent to GUI in response to the received data. De-framing is done by removing unnecessary bytes from frame and extracts data bytes. Actual data bytes are send to user peripheral for further processing. Data received in user peripheral is stored and converted to the required format. The practical implementation of this is done by using Xilinx XPS (Xilinx platform studio).

After successful link establishment, both entities enter into the Data Transfer phase. Then the two communication nodes can exchange messages vice versa. Once the user software enters into data transfer phase, it waits till the start of frame to be received, which triggers the user software into frame receiver mode. Once the user software enters into frame receiver mode it keeps on receiving the bytes till the end of frame is received. All the received bytes of frame are store as byte array in the order they receive.

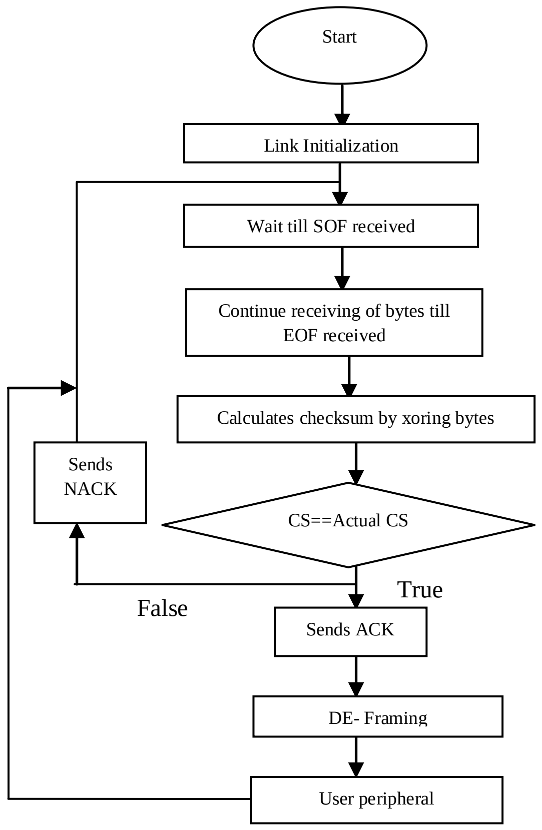

Fig. 3.2 User software flow diagram

Once the frame is received successfully, user software checks whether frame is error free or not. Error checking consists following two stages.

1. It calculates the check sum and compares it with the actual check sum field present in the frame.

2. It counts the number of actual data bytes in frame and compares with the data count field.

If the check sum doesn't matches with actual check sum field then it generates a negative acknowledgement frame and sends to the GUI and it doesn't go for further processing and simply looks for the new start of frame. Negative acknowledgement frame is generated according to the protocol. It contains the data of received frame and byte in the data field which represents the error in the frame to the GUI. If the frame is error free, then application software looks whether all the bytes are received or not. It does this by comparing total length of frame with message length value +5. If any one or both the cases fail then a negative acknowledgement is sent to the GUI.

Bit0 (SOF)	Data Field	Bit7(EOF)

Fig. 3.3 Frame Structure

The Fig. 3.3 represents the frame structure. Once if these two stages are succeeded then positive acknowledgement frame is sent to GUI which contains received data and positive acknowledgement byte. If the received frame is error free then it is send to the de-framing block. De-framing is done in order to extract the actual data field by removing overheads in the frame. Once the data bytes are extracted, then data is send to the newly created IP which contains actual application work. The peripheral stores the data in a particular register according to the received data and process it further and converts it into required output format specified by the received data.

IV. IMPLEMENTATION

Implementation of this is done by using XILINX XPS tool[2, 3, 6]. As user software has to be implemented in FPGA, implementation process is divided into two stages. First stage is implementation of hardware and next stage is software implementation.

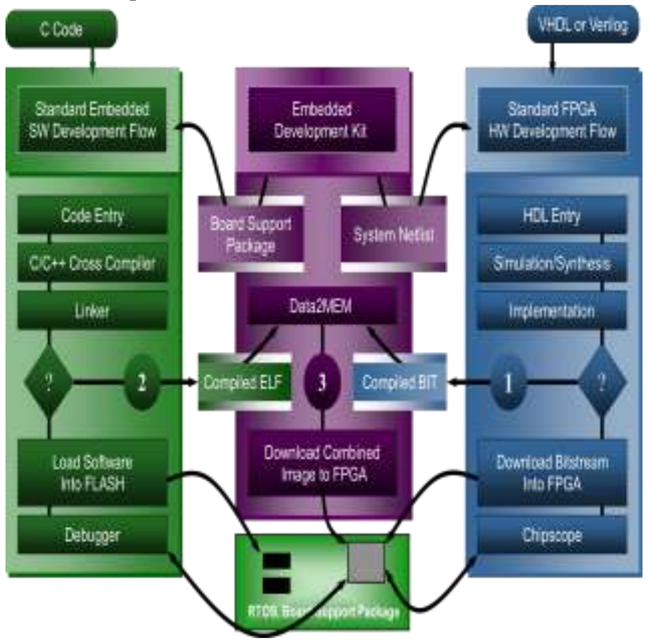

Fig. 4.1 Tool flow overview

The fig.4.1 represents the tool flow overview of XPS. The system design flow simply combines the standard hardware flow used to create FPGA bit streams and standard software flow used to create processor ELF files.

The first step is to create the 'System Netlist' using the Embedded Developers Kit and instantiate that Netlist into the design's HDL code. The hardware design is then synthesized, merged and implemented using the exact same flow as used with any other 'black box' core. While it is common to include a portion of the yet created software image inside the FPGA using block RAM, the 'Compiled BIT' file created during this phase of development only contains the systems hardware description.

The second step is to create the 'Board Support Package' (BSP) using the Embedded Developers Kit (EDK) and include the required drivers in the system's C code. The code is then compiled and linked with the various functions available in the BSP as same with any other processor system. Because the embedded system is built using the FPGA fabric, the BSP is customized for the particular set of peripherals included in the 'System Netlist'. Unlike an off shelf general purpose processor, every BSP is potentially unique and as such EDK is tasked with customizing a generic set of drivers as required to properly support the 'arbitrary processor system'.

Once the final set of peripherals and bus structures have been solidified, the software and hardware flows can be run independently. Even if part or the entire software image is stored using on-chip block RAM, the software flow does not require running the hardware flow from scratch, when making software changes. Only if a change is made to the instantiated 'System Netlist' does the hardware need to be implemented again before the new software image which relies on the architectural change can be loaded and run.

If the software image is completely stored externally, configuring the FPGA and loading the external storage device are performed in exactly the same way as a typical two chip solution. If part or all of the software image is stored using on-chip block RAM and as such is embedded within the FPGA's bit stream, an additional step is required before the FPGA can be configured. EDK provides a tool called Data2MEM which merges the appropriate sections of the 'Compiled ELF' file with the 'Compiled BIT' file. The resulting BIT file is created typically created in a few seconds and can then be used to configure the FPGA. When the entire software image is stored within the FPGA, only the BIT file is needed to configure the system and load the software image. If only portion of the software image, such as the bootstrap, is stored within the FPGA, then Data2MEM is run to create a combined BIT file and the system is once again configured/loaded as any two chip solution using the unmerged ELF sections and the combined BIT file. Debugging the software running on the system is performed in the same manner as would be on any general purpose processor.

Hardware Implementation:

Hardware Implementation can be done as follows. First Invoke the XPS, create a new project and select the Base System Builder Wizard. In the Base System Wizard, Create New XPS Project Using BSB Wizard window, browse to a directory in which to write the project file (system.xmp), choose I would like to create a new design, select Board window, select the board which suits. Next choose I would like to create a system. In the Board Architecture field, select FPGA details. In the Base System Builder wizard, in Processor window, verify that the PowerPC Processor is selected. Next Configure PowerPC Processor window make the selections as per your requirements. Under Processor Configuration, for Debug Interface, accept FPGA JTAG as the Debug interface. Next for Cache setup, uncheck Enable. Confirm that Enable Floating point unit (FPU) is unchecked. Next in the Base System Builder **-** Configure I/O Interfaces window, make the selections as given below

Check RS232_Uart_1 then select XPS UARTLITE in the Peripheral field. Set The Baud Rate to 9600, the Data bits to 8, the Parity to None. Check Use Interrupt.

Uncheck RS232_Uart_2. Next In the Base System Builder **-** Configure I/O Interfaces window- Select LEDs_8Bit. In the Peripheral field, select XPS_GPIO, and uncheck Use Interrupt**,** Add Internal Peripherals (1 of **1)** window, make the selections as below: For XPS BRAM IF CNTLR, select Memory of your requirement. Next In the Base System Builder **-** Software Setup window, make the selections as below. In the STDIN and STDOUT fields, select RS232_Uart_1.

In the Boot Memory field, select xps_bram_if_cntrl_1. Under Sample Application Selection, check Memory Test (default) and check Peripheral self test (default). In the Base System Builder **-** Configure Memory Test Application window, make the selections as: In the Instruction, Data and Stack/Heap fields, select xps_bram_if_cntlr_1.**b** In the Interrupt Vector field, select DDR_SDRAM (default).

In the Base System Builder **-** Configure Peripheral Test Application window, make the selections as below. In the Instruction, Data Stack/Heap, and Interrupt Vector fields, select DDR_SDRAM. In the Base System Builder **-** System

Created window, where the selections which have been made are summarized, click Generate to generate the system. In the Base System Builder - Finish window, check Save Settings File click Finish.

Software implementation:

Software implementation can be done as follows. Select Software menu, go to Software platform settings to select the software settings for the embedded system. In Software platform settings check Xilkernel, uncheck lwip and Xilmfs. In OS& Libraries settings configure the Xilkernel libraries according to requirements. Next in driver window default drivers are configured. Select OK. Again select Software menu, go to Generate Libraries and BSPs to generate libraries and board support packages. This will generate .MSS file. In Application area select Add application project option and type the name of the application in the Project Name area and click OK. A new application is created. In the created application add source files and header files which are used for implementing the application. After writing the user software, build the application using Software → Build all User Applications for checking the errors and debugging the developed application. After completing the above procedure .elf file for the application is generated it is used to download it onto the FPGA. Select Debug →XMD Debug options select connection type to Hardware, JTAG Cable type Auto, check Auto discover JTAG chain definition and click OK. Download bit stream to FPGA using JTAG using Device Configuration→ Download Bit stream to the FPGA. Here sytem.bit file and .elf file are merged in order generate download.bit file and configures the FPGA through JTAG. When .elf file is not included in download.bit file then Select Debug > Launch XMD from the XPS GUI to debug the user software. After launching XMD debugger download .elf file of the application using the command download <application name>/executable.elf. Run the application using the command run. Use stop command to stop the processor.

V. RS-232

The concept of serial communication is the process of sending data one bit at a time, sequentially, over a communication channel. RS-232(Recommended Standard 232) is the traditional name for a serial binary single-ended data and control signals connecting between a DTE (Data Terminal Equipment) and a DEC (Data Circuit terminating Equipment).

Rs-232 data usually is sent as a packet with 7 or 8bit words, start, stop, parity bits (may be varied). Start bit followed by data bits, parity bit and finished by stop bit. The start and stop bits are used in asynchronous communication.

Details of character format and transmission bit rate are controlled by the serial port hardware, often a single integrated circuit called a UART that converts data from parallel to asynchronous start-stop serial form. Details of voltage levels, slew rate, and short-circuit behavior are typically controlled by a line driver that converts from the UART's logic levels to RS-232 compatible signal levels, and a receiver that converts from RS-232 compatible signal levels to the UART's logic levels.

VI. TESTING

To Test the user software a GUI is developed. First the user software is downloaded in to the board using JTAG cable using download bit stream option in XPS menu. After downloading bit stream, now we download application using XMD debugger in XPS menu using “Download <application name>/executable. Elf”. Now the FPGA board is ready to communicate on RS-232 link.

VII. RESULTS

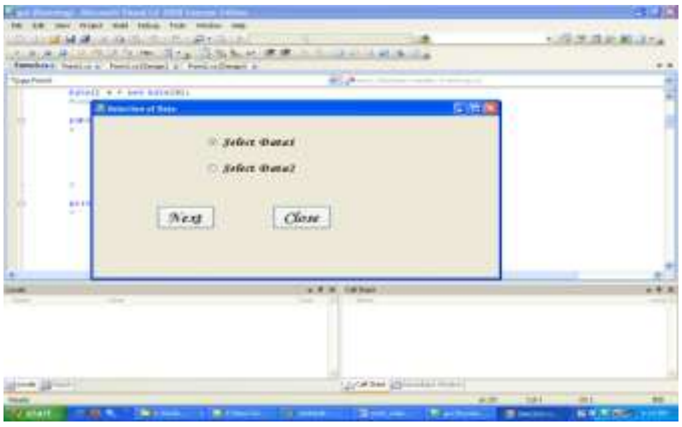

Fig. 8.1 Selection of Data from GUI

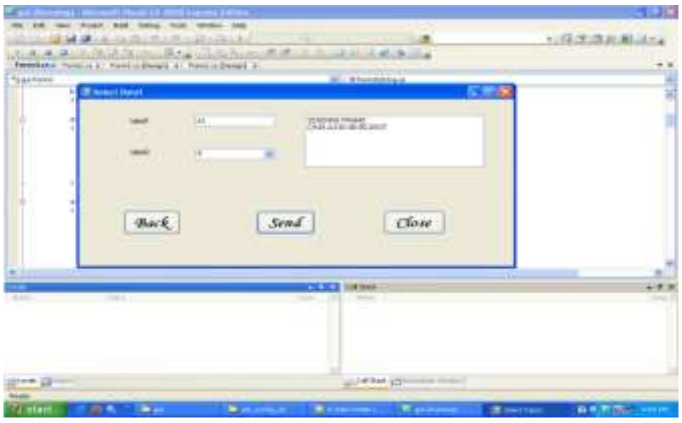

Fig. 8.2 Result after select the data from GUI

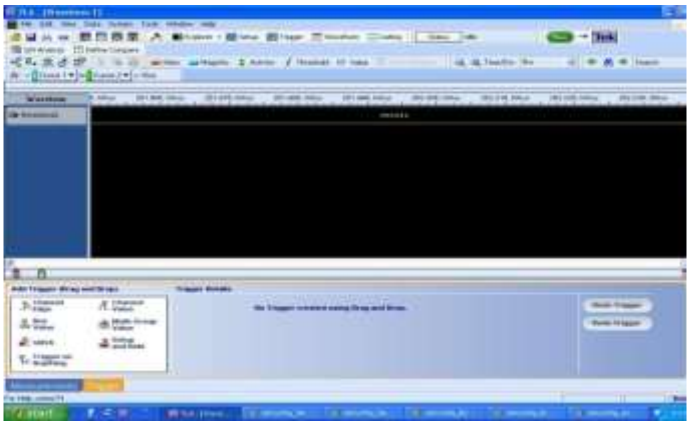
Fig. 8.3 Result in LSA (Logic Analyzer) after dumping the program into FPGA on GPIO pins.

VIII. CONCLUSION

Serial communication through RS-232 link has been successfully established between Graphical User Interface and Simulator board. Subsequently selected data have been communicated between the Graphical User Interface and Simulator board

The future scope of this work can be enhanced by suing Ethernet link.

ACKNOWLEDGMENT

The authors would like to thank P. Prashanth Application Engineer, Digi Logic Systems, Hyderabad for their support in implementation of the hardware and D. Krishnaveni Sc.'D', DLRL, Hyderabad for giving valuable suggestions .

REFERENCES

[1]. Clive "MAX" Maxfield, "*the design warrior's guide to FPGAs*", 2004.
[2]. Jonathon W. Donaldson, "*Xilinx EDK Tutorials and Notes* ".
[3]. Xilinx Website -*www.xilinx.com*
[4]. *www.xilinx.com/support/documentation/dt_edk_edk10-1.htm*.
[5]. Embedded System Tools -Reference Manual - Xilinx
[6]. Xilinx software Help tool – Xilinx
[7]. Douglas L. Perry "*VHDL Programming by Example*", 2002.
[8]. Microsoft visual C# platform studio-2010