PROJECT 2 Sai Krishna Kandagatla
FINAL REPORT U15974415

Design and Implementation of Loopback System on Pico blaze Microcontroller to Serial Terminal and Smart Phone Using Bluetooth Interface

1. Abstract:

Pico Blaze is an 8-bit soft-core microprocessor created by Xilinx that can be done in some FPGA families. This paper exhibits a lot of peripherals that have been created to interface with Pico Blaze: loopback control, serial communication, UART and Bluetooth module. To exhibit its capacities, the project has been executed in an FPGA board and some run control and observing frameworks have been created. The design approach of the peripherals and details of the integration of the systems is shown in the paper.

This project reports on the use of a Field-Programmable Gate Array based embedded processor to control a loopback system using UART protocol and then Bluetooth module to send it onto the mobile phone. The project idea is to send the information serially using the microcontroller from the Assembler to the PC in the form of serial communication and then the information is sent to the mobile phone using the Bluetooth module. Therefore, this project mainly focuses on sending and receiving the data by the implementation of Loopback system-based Pico blaze microcontroller and sending to the mobile phone using the Bluetooth module.

Here loopback system using a Pico Blaze microcontroller, and to implement functionality using Assembly code is transmitted to a Serial terminal on monitor first and then to Smart Phone Using Bluetooth module (Pmod bt2) as interface between the Smart phone and FPGA and it is implemented using the UART Protocol.

Keywords: Loopback, Assembler, Pico Blaze, Serial Terminal, UART, Bluetooth Module.

2. Theory Background:

Multi-processor designs have picked up intrigue as of late because of their capacity to exploit programmable silicon parallelism at satisfactory power-productivity figures. Notwithstanding the potential advantage they offer over single-processor models, it is uncertain how one can compose reduced and proficient projects for different parallel centers. In this paper, we propose the utilization of a synchronous a hardware description language to program a system of Pico Blaze processor. The partitioning of a multiprocessor program over various centers is clear because the information is completely parallel. An efficient change process changes over the parallel info detail into simultaneous Pico Blaze programs. [1]

Pico Blaze is a fully embedded 8-bit RISC microcontroller core optimized for 7-series and older Xilinx FPGA architectures. This reference design is offered free to Xilinx users, and comes with an easy to-use code assembler KCPSM6 (or KCPSM3 for older FPGA families), VHDL and Verilog source code, simulation models, comprehensive documentation and reference designs. [10]

The Pico Blaze microcontroller core is totally embedded within the target FPGA and requires no external resources. The Pico Blaze microcontroller is extremely flexible. The basic functionality is easily extended and enhanced by connecting additional FPGA logic to the microcontroller's input and output ports. [11]

Xilinx has thought of a solution for plans that have modest necessities but require the presence of a microchip: Pico Blaze. Pico Blaze is an 8-bit microprocessor soft core planned to supplant vast limited state machines in low- to mid-multifaceted designs. In its current release, Pico Blaze comes prepared, among different highlights, with a basic interrupt handling module, generic input/output communication ports, and an equipment stack, allowing the chance to be completely agreeing by the designer's needs. [11]

UART is a device that has the capability to both receive and transmit serial data. UART exchanges text data in an American Standard Code for Information Interchange (ASCII) format in which each alphabetical character is encoded by 7 bits and transmitted as 8 data bits. For transmission the UART protocol wraps this 8-bit sub word with a start bit in the least significant bit (LSB) and a stop bit in the most significant bit (MSB) resulting in a 10-bit word format. UART transmitter controls transmission by fetching a data word in parallel format and directing the UART to transmit it in a serial format. Likewise, the Receiver must detect transmission, receive the data in serial format, strip of the start and stop bits, and store the data word in a parallel format. Since the UART is asynchronous in working, the receiver does not know when the data will come, so receiver generate local clock to synchronize to transmitter whenever start bit is received. [6]

Serial communication is widely used in the electronics industry. Serial communication system supports full-duplex communication on a serial link. Its fundamentally capacities are in the serial and parallel data stream change between one another, the transmission of serial data between devices and information exchange and is a regularly utilized information correspondence segment in SoC. In addition, elite FPGA devices costly, which will cause a sharp increment in configuration cost. In the interim, normal UART ASIC is modest and solid, however the baud rate it can bolster is excessively low. It is a trade-off between configuration expenses and serial transmission speeds. High system operating frequency gives adequate space to the expansion of baud rate under the rapid applications. [2]

The remote advancement has been connected with the phone utilities or controlling of different gadgets, PCs at a separation. A standout amongst the most widely recognized remote correspondence advancements is the Bluetooth innovation. The Bluetooth is an innovation for short separation remote correspondence utilizing 2.4GHz Industrial, Scientific and Medical (ISM) radio band. There are such numerous advances work in this band including IEEE 802.11b/g. Bluetooth utilizes continue skipping spread range improvement for all transmission. As it is outstanding, this advancement limits impedance and transmission goofs and gives transmission security. [12]

3. Description of Project:

The Pico Blaze architecture consists of an 8-bit datapath with a 64-byte scratchpad RAM, 16 registers and a 1K word (18 bits wide) Instruction PROM. A block diagram view of the Pico Blaze embedded microcontroller is shown in Figure 1. The latest version of the Pico Blaze microcontroller is called KCPSM3 which can be downloaded from the Xilinx Pico Blaze lounge. The Pico Blaze design suite of tools consists of "kcpsm.v" (the main Verilog microcontroller code) and an assembler folder "assemblycode.psm". The assembler is used to generate the Verilog code for the Block RAM from the psm code. The Pico Blaze microcontroller is resource efficient allowing multiple Pico Blaze microcontrollers to be implemented on a single FPGA. [11]

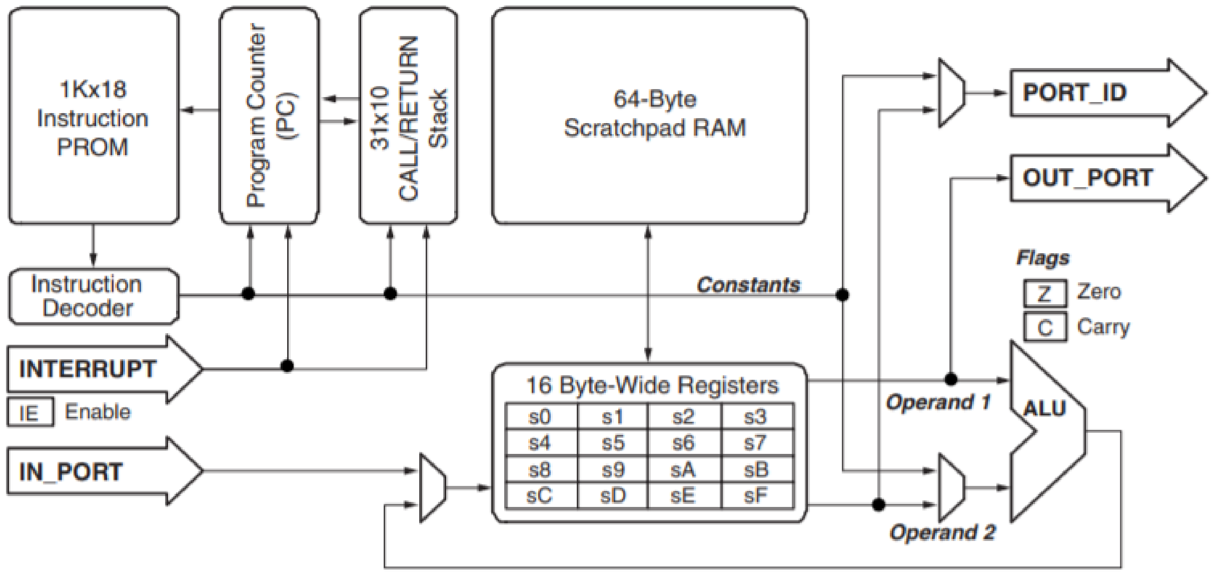


Fig. 1. Block Diagram of Pico Blaze Microcontroller

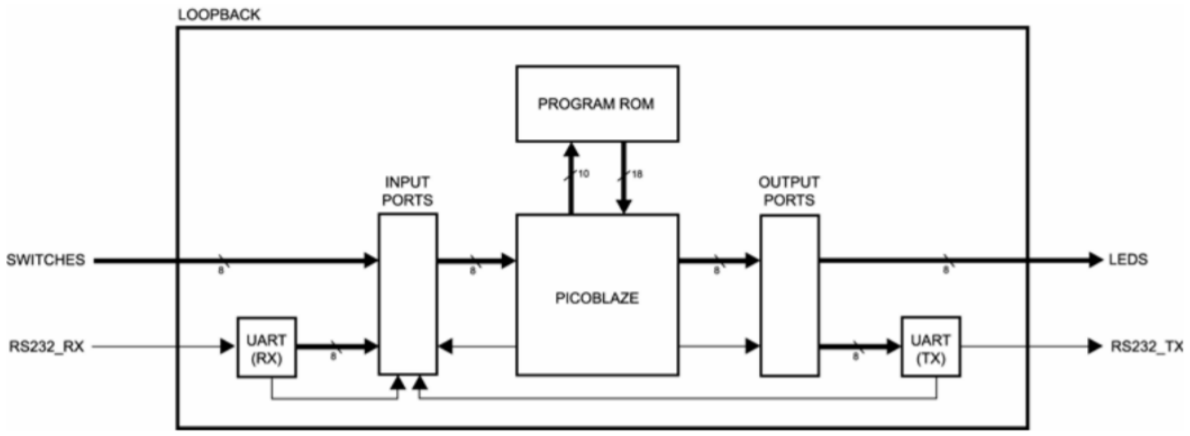


Fig. 2. Schematic Diagram

The main aim of the project is to write software in Pico Blaze assembly to implement a loopback test. A loopback test is a test in which a flag is sent to a device and came back from that same device to decide if the device is working properly. The first loopback test will reverberate reversed switch settings on LEDs. Here, you are transmitting with your fingers, and accepting with your eyes what is circled back by the system. The second loopback test will resound serially received data over a serial port. Here, a PC is transmitting with its serial port, and accepting with its serial port what is looped back by the system. Consider a straightforward loopback system as in Fig. 3. In such a setup, a PC can communicate a message which is returned (looped) back to PC by the processor. This is known as Loopback generally.

Pico Blaze is a compact 8-bit microcontroller. It comprises of two important blocks: the processor and an instruction ROM which holds the assembly program. A PC interfaces with Pico Blaze processor utilizing serial connection. PC acts as a transmitter. For this situation the information arrives serially on Receiver. Pico Blaze acknowledges only byte-size data. So, we need to change the serial data into parallel (8-bit) data. In such a case, a UART (Universal Asynchronous Receiver Transmitter) block can do as it sees the line transition from high to low, it knows that a UART data word is coming. This first transition indicates the start bit. Once the beginning of the start bit is found, the FPGA waits for one half of a bit period. This ensures that the middle of the data bit gets sampled. UART makes the serial-to-parallel conversion i.e., the information receiving on receiver line is buffered in a 16-byte buffer. The UART serial data stream is shown below. After Pico Blaze reads the information, it can ask the next data item by asserting read by uart signal.

Presently, let us consider the situation when Pico Blaze acts as a transmitter i.e., PC is the receiver. For this situation, Pico Blaze puts out 8-bit information while PC is expecting serial data. To check the serial port conduct, connect the FPGA board to the COM8 serial port on your PC computer using a USB cable. Utilize the terminal session to launch the Putty Serial application. If you need to use some other COM port, you will need to change the Putty Serial Terminal session options. After pushing of the button of the Pico Blaze system, you should receive the message on the Terminal window. See that the push button is not clicked sometimes when the button is released, you may receive a few garbage characters, and to the message, in the Putty Terminal window. Afterwards, in the same the message is received to smart phone using the Bluetooth module as in Fig. 4. PC sends information to Android phone by using Bluetooth transmitter containing FPGA board. The android application is intended to receive serial information from the Bluetooth module.

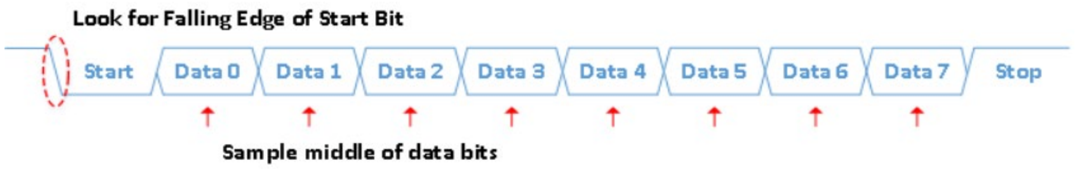

Look for Falling Edge of Start Bit
Start
Data 0
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7
Stop
Sample middle of data bits

Fig. UART Serial data stream

Design of Loopback system that I have implemented:

The design I implemented is Loopback system. So, the sending and receiving of data is same. The Loopback system I have implemented is shown in the Fig 3. The data generated by the Loopback system is shown in Fig 3(a) and Fig 3(b). The tabular representation of the data sent and received based on ascii values for that example is shown in the Table 1.

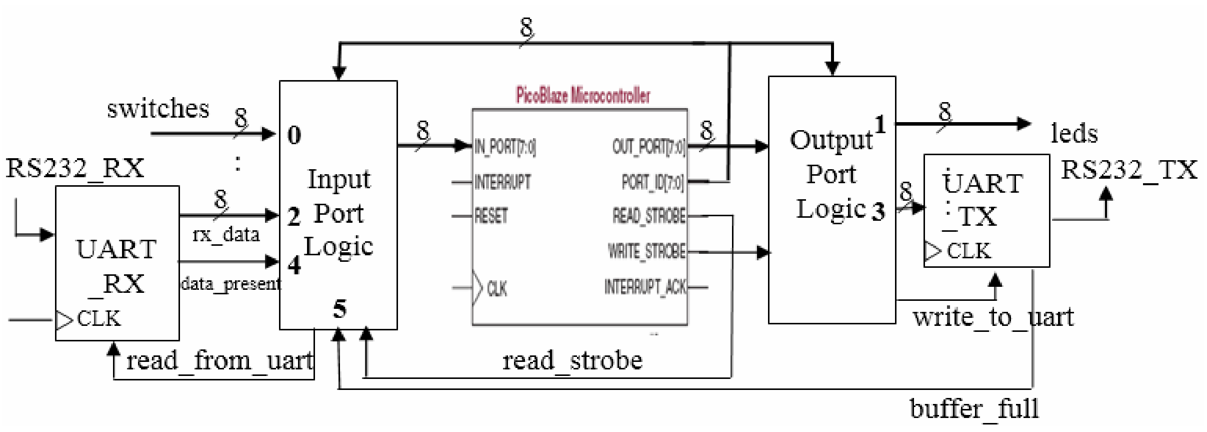
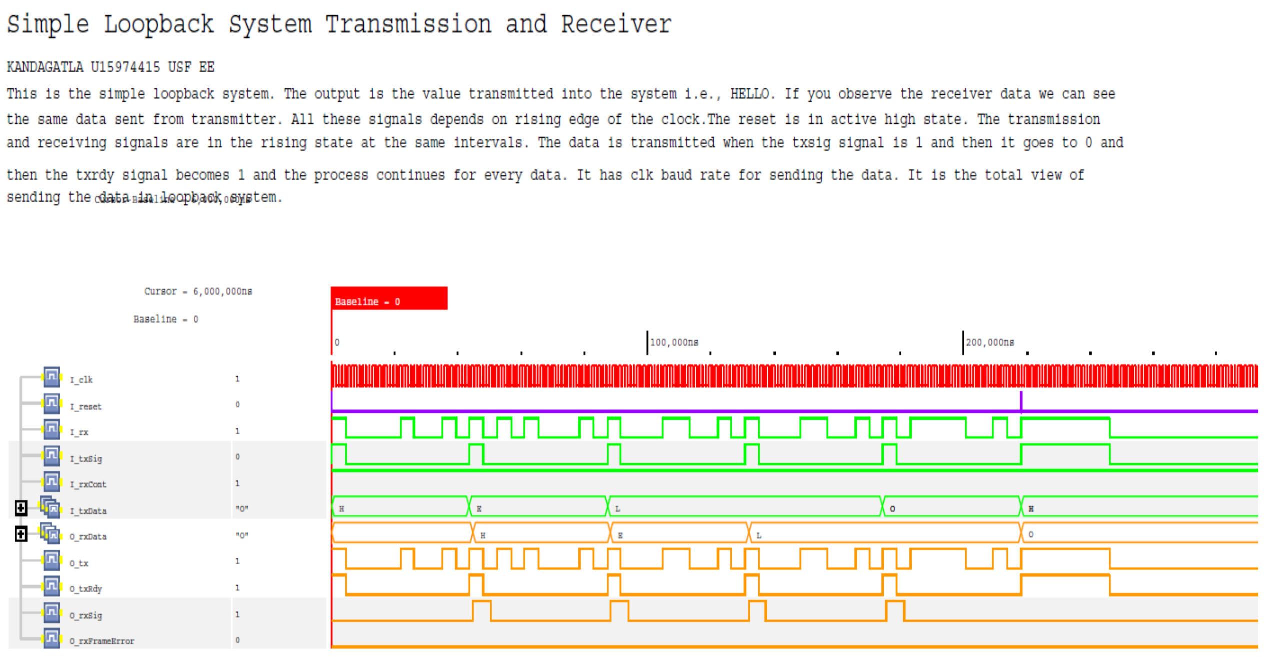


Fig. 3. A simple loopback system

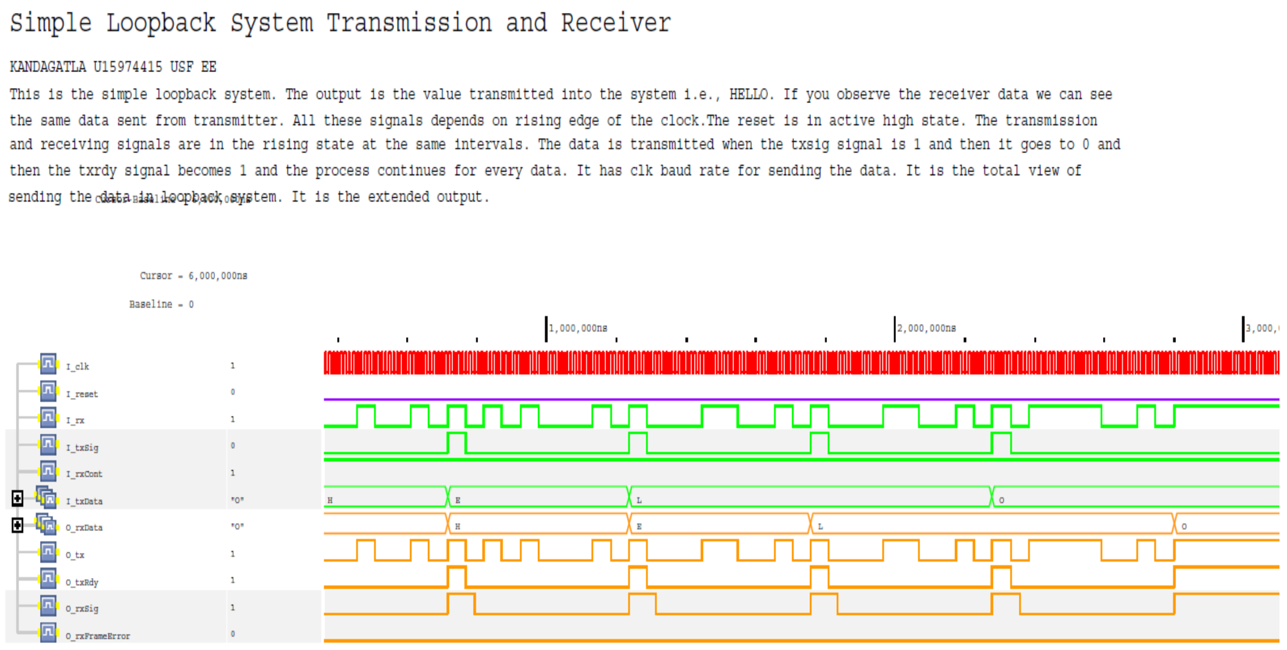


Simple Loopback System Transmission and Receiver

KANDAGATLA U15974415 USF EE

This is the simple loopback system. The output is the value transmitted into the system i.e., HELLO. If you observe the receiver data we can see the same data sent from transmitter. All these signals depends on rising edge of the clock.The reset is in active high state. The transmission and receiving signals are in the rising state at the same intervals. The data is transmitted when the txsig signal is 1 and then it goes to 0 and then the txrdy signal becomes 1 and the process continues for every data. It has clk baud rate for sending the data. It is the total view of sending the data in loopback system.

Fig. 3(a) Simulation of Loopback system



Simple Loopback System Transmission and Receiver

KANDAGATLA U15974415 USF EE

This is the simple loopback system. The output is the value transmitted into the system i.e., HELLO. If you observe the receiver data we can see the same data sent from transmitter. All these signals depends on rising edge of the clock.The reset is in active high state. The transmission and receiving signals are in the rising state at the same intervals. The data is transmitted when the txsig signal is 1 and then it goes to 0 and then the txrdy signal becomes 1 and the process continues for every data. It has clk baud rate for sending the data. It is the total view of sending the data in loopback system. It is the extended output.

```
Cursor - 6,000,000ns
Baseline - 0

I_clk           1
I_reset         0
I_rx            1
I_txSig         0
I_rxCont        1
I_txData        "O"
O_rxData        "O"
O_tx            1
O_txRdy         1
O_rxSig         1
O_rxFrameError  0
```

Fig. 3(b) Simulation of Loopback system extended

S.NO	Transmission	Receiving
1	H	H
2	E	E
3	L	L
4	L	L
5	O	O

Table 1: Sending and Receiving data from example

The data that is transmitted to a PC using the UART from loopback system is sent to the mobile phone using Bluetooth module. The Bluetooth module used in this project is Diligent pmod bt2 which is compatible with Spartan- 6 FPGA.

The Bluetooth module can be used with UART protocol and the software emulator/ Serial terminal is used to interface the Bluetooth module with Smart phone. The pmod bt2 is shown in the fig below.

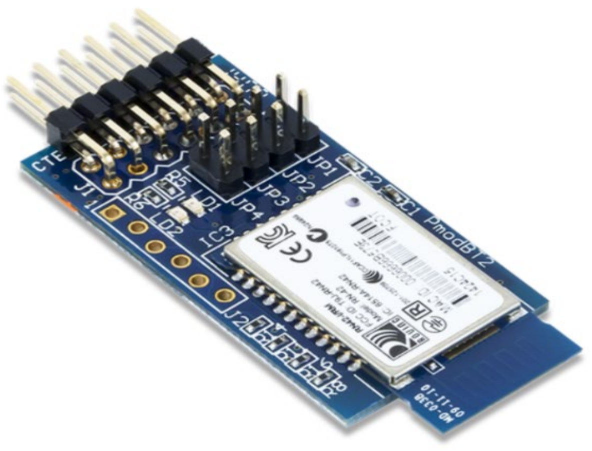


Fig. 4. Pmodbt2 Bluetooth Module

Functional Description of Pmodbt2:

The Pmod BT2 uses a 12-pin Pmod port and communicates via UART. There is a secondary SPI header on the board for updating the RN-42 firmware if needed, however this port is not used under normal operation.

The typical application for the Pmod BT2 is to replace a wired UART connection between two Bluetooth capable devices. When paired with an Android, Linux, Mac OS X, or Windows computer, the Pmod BT2 looks like a serial COM port like how a USB-UART bridge or RS-232 serial port behaves. The Pmod BT2 can be easily configured from the Bluetooth connected computer by entering a “Command Mode” that allows settings such as UART baud rate to be programmed into non-volatile configuration registers.

When used with another Pmod BT2, a wireless UART bridge can be achieved with no need for software configuration of the Bluetooth link. This is accomplished by using the on-board jumper settings to control an automated discovery process of the two Pmod BT2's. In this way you can communicate wirelessly between two embedded devices without complicated software like is required with Wi-Fi or 802.15.4 solutions. Communicate wirelessly with simple serial prints and reads!

Note that due to security restrictions built into iOS, the Pmod BT2 cannot be used with the iPhone, iPad, or other iOS devices. [12]

Interfacing with the Pmod:

UART/Pmod Interface:

The Pmod BT2 communicates with the host board via the UART protocol. By default, the UART interface uses a baud rate of 115.2 kbps, 8 data bits, no parity, and a single stop bit. The start-up baud rate may be customized to predefined rates or set to a specific user customized baud rate ranging from 1200 bps to 921 kbps.

The reset pin (RST) on J1 is active low. If the RST pin is toggled, the device will undergo a hard reset. This hard reset performs similarly to a power cycling of the device. The second interface besides the standard UART signals is the STATUS pin also on J1. The STATUS pin directly reflects the connection status of the device. STATUS is driven high by the device when connected and is driven low otherwise. [6]

Jumpers:

The Pmod BT2 has several settings that are configurable via jumper blocks JP1 through JP4. These jumpers are all sampled in the first 500 ms of operation and configure the RN-42 module's behaviour whether they are shorted are un-shorted when the Pmod BT2 is powered up.

JP1 restores the device to factory default settings after three transitions of the jumper setting (short-to-open or open-to-short). This only works if JP1 was initially shorted before the Pmod BT2 was powered on. Also, each transition must be separated by a 1 second pause. If done correctly, the LED on the Pmod BT2 will blink rapidly.

JP2 enables pairing with a special device class defined by the user in software. This may be used so that the Pmod BT2 operates as a substitute for an RS232 cable. JP3 enables auto connect to a stored address defined by the user. Finally, JP4 chooses whether to operate at the stored baud rate (115.2kbps default) or a baud rate of 9600 regardless of the software selected rate when shorted. [12]

Description of implementation of the project:

I have converted Loopback data output to single bit output using Parallel in serial out block as shown in the visio diagrams and the output of it is sent to the PC monitor using UART module. So, the output from the Loopback block is given to Key2ascii block and the output from that block is given as the input of UART. In any asynchronous interface, the first thing you need to know is when in time you should sample the data. If you do not sample the data at the right time, you might see the wrong data. To receive your data correctly, the transmitter and receiver must agree on the baud rate. The baud rate is the rate at which the data is transmitted. For example, 9600 baud means 9600 bits per second. The code below uses a generic in VHDL or a parameter in Verilog to determine how many clock cycles there are in each bit. This is how the baud rate gets determined. The UART module transfer the data to Serial Terminal based on its baud rate. The baud rate of UART can be maintained by changing in the code.

The UART block, Loopback block and complete schematic of project (top-level) are shown in the below figures.

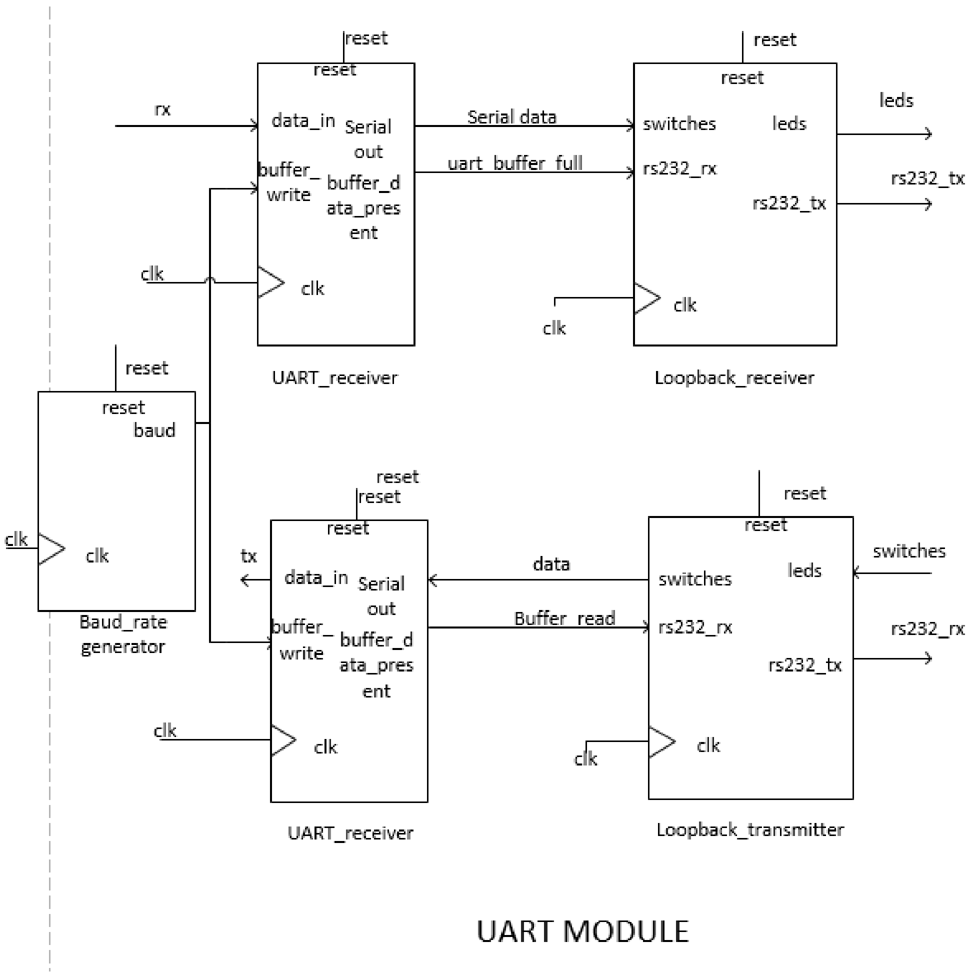
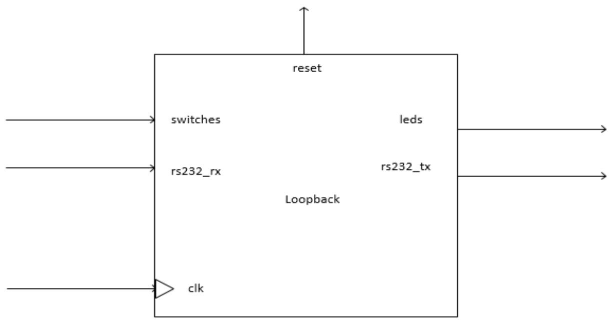


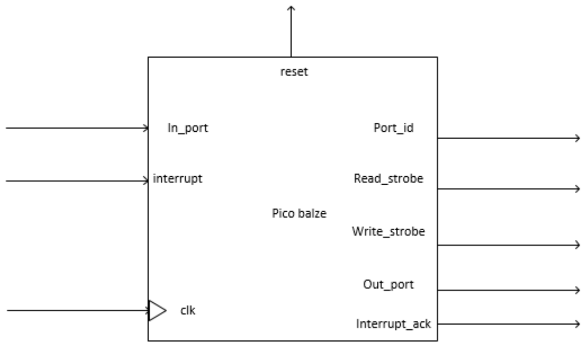
Fig. UART module

PROJECT 2 Sai Krishna Kandagatla
FINAL REPORT U15974415



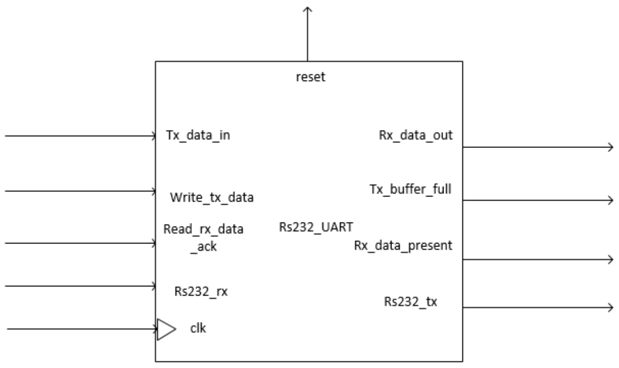
Top level Loopback MODULE

Fig. Loopback module



Top level Pico Blaze MODULE

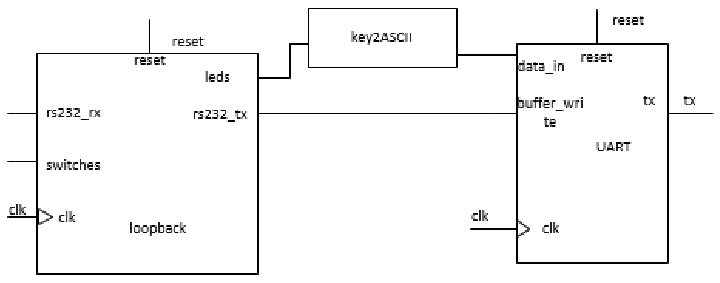
Fig. Pico Blaze module



Top level rs232_UART MODULE

Fig. rs232_UART module

PROJECT 2 Sai Krishna Kandagatla
FINAL REPORT U15974415



Toplevel Module

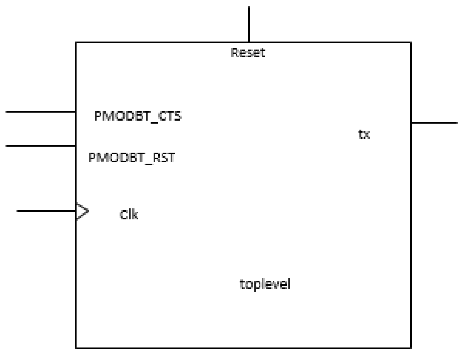


Fig. Top level module

The Project consists of synthesis and Implementation of

i) Pico Blaze Microcontroller

ii) Loopback module

iii) UART Modules

iv) Assembler module

v) RS232 using UART Module

vi) Bluetooth Module

i) Pico Blaze Microcontroller:

Here a Pico Blaze Microcontroller is designed in Verilog to test it. The inputs are in port and interrupt is given for the kcpsm6 pblaze cpu which is used as component for the program and also it is interfaced with the pblaze_rom which is used to transfer the data to the output and the clock and reset are used for the synchronization of the microcontroller.

ii) Loopback module:

Here a Loopback module is designed and implemented using RS232 and pico blaze components. During the synthesis of Loopback, the inputs switches and rs232 receiver are used to get the data. The LEDs are used to give the output which is in a loopback fashion. If we consider the reset, then in_port and uart rx will be zero in the process which are from the RS232 and pico blaze components. The led driver is also instantiated for the process.

If we take port_id then output will be the switches for 00, received uart data for 02, uart data present will be for 04 and the default will be for 00 and the data will be transmitted.

iii) UART Modules:

Here a uart_tx and uart_rx is designed and implemented. The baud rate is 9600 bits per second. The datain and serialin are inputs for the transmission and receiving. The buffer data which is to be transmitted from UART modules to the output is done in bit by bit format with synchronization.

iv) Assembler module:

Here an assembler program is designed and implemented in assembly language. During the synthesis of assembler program it should be kept in the assembler application to convert it to the Verilog code which can be used for the project. In the code, addresses are defined and the required constants are initialized. The constant declarations which are made are written with respect to ascii values which can be used in the project. Then the data which should transmitted between the microcontroller to the serial terminal and to the mobile phone is written in one function with its ascii values loading them into the uart transmitter. Then for the output to be displayed on the leds and inputs which are switches are written for the uart receiver.

v) RS232 using UART Module:

Here a RS232 program is designed and implemented. The components uart transmitter and uart receiver are used in this program. It is used as the barrier for the data to transfer to serial monitor from the microcontroller. During the synthesis of RS232 program datain and dataout are the input and output. The baud rate is 9600 bits per second. The maximum baud count is 651. The baud rate is used for the transmission of data bit by bit to the output. The Output is observed by glowing of the LEDs.

vi) Bluetooth Module:

Here a pmod program is designed and implemented. The last pins on the Pmod are always used as a ground and power and the first ones are used for data. With a 12-pin Pmod, the pins 1-4 and 7-10 are data pins while 6 and 12 are VCC and pins 5 and 11 are ground. The components uart tx and uart rx and pmod bridge are used in this program. It is used as the barrier for the data to transfer to mobile phone from the microcontroller. The program consists of the address, interrupt, valid, data, ready are signals used in the program for the transmission of the data.

Trails I have made to get the output:

Trail-1:

In Trail -1, I have generated the data i.e. obtained by giving assembler output to parallel in serial out block and from that output data is generated using the clock and the output data is directly given to Microcontroller and the output is given to UART block and UART block communicated with Serial communication using the Microcontroller.

Here I generated clock with frequency of 9600 Hz and UART Baud rate is 9600.

Trail-2:

In Trail -2, I have generated the data i.e. obtained by giving Microcontroller output to parallel in serial out block and from that output data is generated using the clock and the output data is directly given to Loopback and the output is given to UART block and UART block communicated with Loopback using the TX output to display it on serial monitor.

Here I generated clock with frequency of 9600 Hz and UART Baud rate is 9600.

Trail-3:

In Trail -3, I have generated the data i.e. obtained by giving output to Mobile Phone parallel in serial out block and from that output data is generated using the clock and the output data is directly given to Mobile Phone and the output is given to UART block and UART block communicated with Bluetooth module using the TX output.

Here I generated clock with frequency of 9600Hz and UART Baud rate is 9600.

Trail-4:

In Trail -4, I have generated the data i.e. obtained by giving output to Mobile Phone parallel in serial out block and from that output data is generated using the clock and the output data is directly given to Mobile Phone and the output is given to UART block and UART block communicated with Bluetooth module using the TX output.

Here I generated clock with frequency of 2400Hz and UART Baud rate is 2400.

Trail-5:

In Trail -5, I have generated the data i.e. obtained by giving output to Mobile Phone parallel in serial out block and from that output data is generated using the clock and the output data is directly given to Mobile Phone and the output is given to UART block and UART block communicated with Bluetooth module using the TX output.

Here I generated clock with frequency of 19200Hz and UART Baud rate is 19200.

Trail-6:

In Trail -6, I have generated the data i.e. obtained by giving output to Mobile Phone parallel in serial out block and from that output data is generated using the clock and the output data is directly given to Mobile Phone and the output is given to UART block and UART block communicated with Bluetooth module using the TX output.

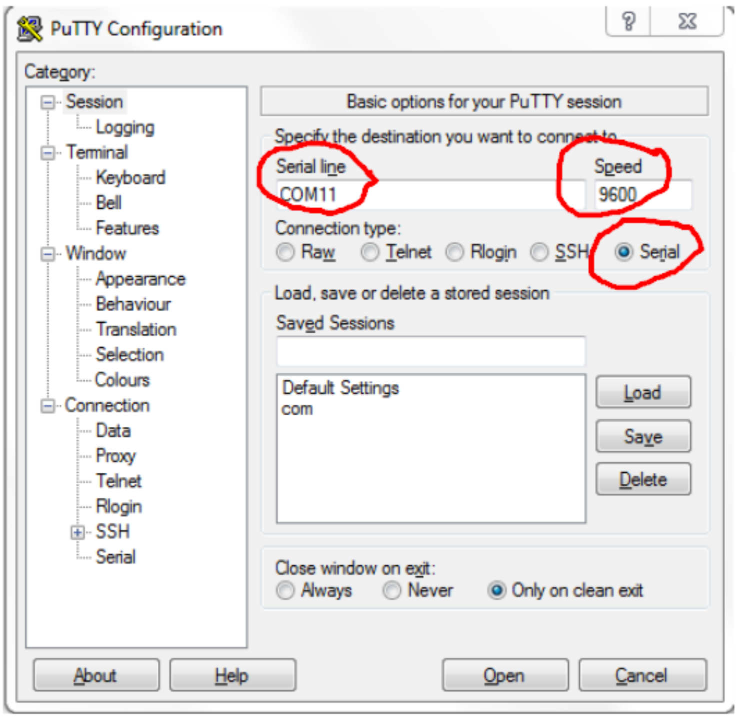
Here I generated clock with frequency of 2700Hz and UART Baud rate is 2700.

Setting up of Serial Terminal using Putty:

To verify the RS232 serial port behavior, connect the Spartan-6 FPGA board to the COM serial port on your desktop computer using a serial cable. Use the provided terminal session file to launch the Serial Terminal application. If you need to use some other COM port, you will need to change the Serial Terminal session options. Otherwise, leave the settings unchanged.

Upon hardware reset of the Pico Blaze system, you should receive the message in the Serial Terminal window. Note that the reset push button is not de-bounced; sometimes, when the button is released, you may receive a few garbage characters, in addition to the message, in the Serial Terminal window. Afterwards, if serial loopback is implemented, anything you type in the Serial Terminal window should be echoed back to the window. If loopback is not implemented or not functioning properly, you will see nothing in response to typing in the Serial Terminal window.

First, we need to setup the COM port and the baud rate for the receiving of data on the serial monitor. The COM properties should be the same as in the figure given below i.e., baud rate is 9600, data transfer rate is 8 bits per second, parity is none and flow control is none. Then the ASCII setup for the data to be in alphabets. After this the Serial terminal will be displayed to see the output.



Specify the destination you want to connect to

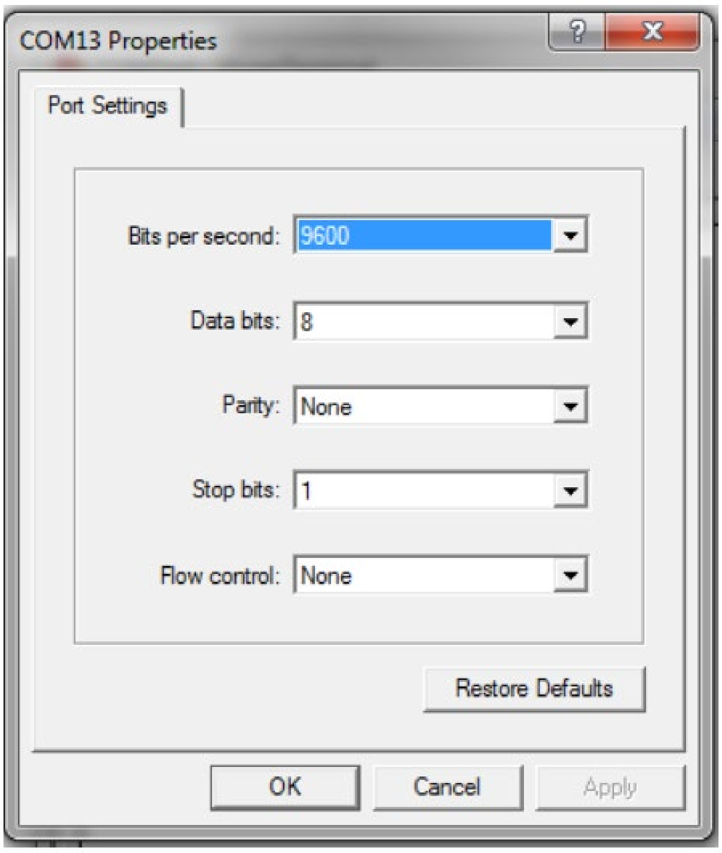
Serial line: COM11

Speed: 9600

Connection type: ( ) Raw ( ) Telnet ( ) Rlogin ( ) SSH [✓ Serial]

Saved Sessions:
Default Settings
com

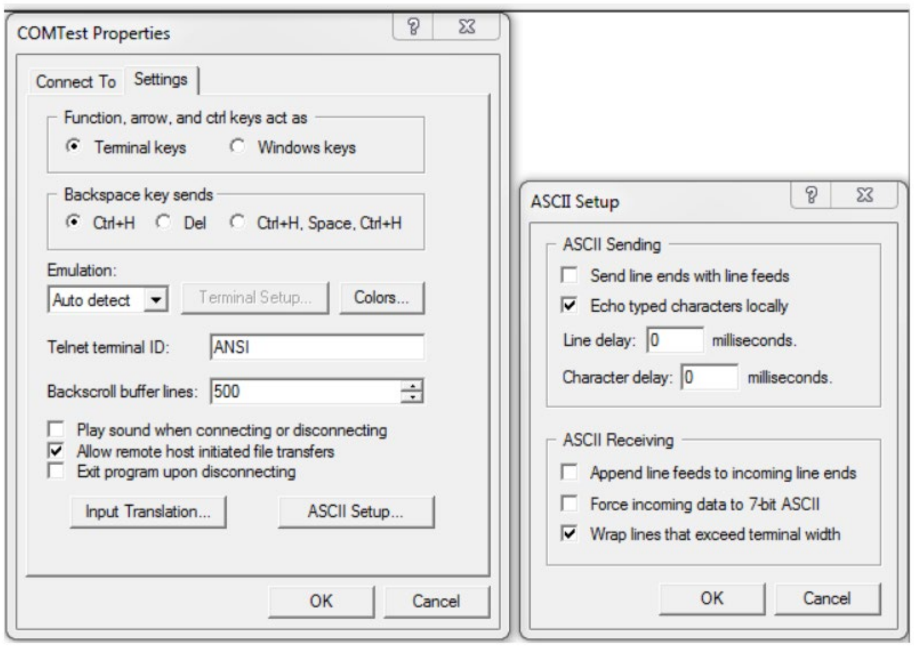
Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]



Bits per second:	9600
Data bits:	8
Parity:	None
Stop bits:	1
Flow control:	None

PROJECT 2
FINAL REPORT

Sai Krishna Kandagatla
U15974415


COMTest Properties

Connect To | Settings

Function, arrow, and ctrl keys act as
[✓ Terminal keys] Windows keys

Backspace key sends
[✓ Ctrl+H] Del Ctrl+H, Space, Ctrl+H

Emulation: Auto detect
Terminal Setup... Colors...

Telnet terminal ID: ANSI

Backscroll buffer lines: 500

☐ Play sound when connecting or disconnecting
☑ Allow remote host initiated file transfers
☐ Exit program upon disconnecting

Input Translation... ASCII Setup...

OK Cancel

ASCII Setup

ASCII Sending
☐ Send line ends with line feeds
☑ Echo typed characters locally
Line delay: 0 milliseconds.
Character delay: 0 milliseconds.

ASCII Receiving
☐ Append line feeds to incoming line ends
☐ Force incoming data to 7-bit ASCII
☑ Wrap lines that exceed terminal width

OK Cancel

Figs. Setup of Serial Terminal

Tools used:
Tools Used For this project an Atlys board with a Spartan6 FPGA is used. Xilinx ISE version 13.2 is used for synthesizing the project. Diligent adept tool was used for downloading the bit file. UART, RS232, Pico Blaze microcontroller, LED features and pmodbt2 module for the Diligent board were used to debug during the development phase.

Output Port Configuration:
The ucf file was modified to add all the input and output ports for the project.
The ports added to the ucf file are:

# Plan Ahead Generated physical constraints

Sai Krishna Kandagatla
U15974415

```
NET "PMODBT_CTS" LOC = N5 | IOSTANDARD = LVCMOS33 ;
NET "PMODBT_RST" LOC = T9 | IOSTANDARD = LVCMOS33 ;
NET "switches[7]" LOC= E4;
NET "switches[6]" LOC= T5;
NET "switches[5]" LOC= R5;
NET "switches[4]" LOC= P12;
NET "switches[3]" LOC= P15;
NET "switches[2]" LOC= C14;
NET "switches[1]" LOC= D14;
NET "switches[0]" LOC= A10;

NET "leds[7]" LOC=N12;
NET "leds[6]" LOC=P16;
NET "leds[5]" LOC=D4;
NET "leds[4]" LOC=M13;
NET "leds[3]" LOC=L14;
NET "leds[2]" LOC=N14;
NET "leds[1]" LOC=M14;
NET "leds[0]" LOC=U18;

NET "rs232_tx" LOC=B16;
NET "rs232_rx" LOC=A16;
NET "reset" LOC=T15;
NET "clk" LOC=L15;
NET "tx" LOC = R3       | IOSTANDARD = LVCMOS33;
```

Experimental Results

Simulation Results:

Simple Loopback System Transmission and Receiver

KANDAGATLA U15974415 USF EE

This is the simple loopback system. The output is the value transmitted into the system i.e., HELLO. If you observe the receiver data we can see the same data sent from transmitter. All these signals depends on rising edge of the clock.The reset is in active high state. The transmission and receiving signals are in the rising state at the same intervals. The data is transmitted when the txsig signal is 1 and then it goes to 0 and then the txrdy signal becomes 1 and the process continues for every data. It has clk baud rate for sending the data. It is the total view of sending the data in loopback system.

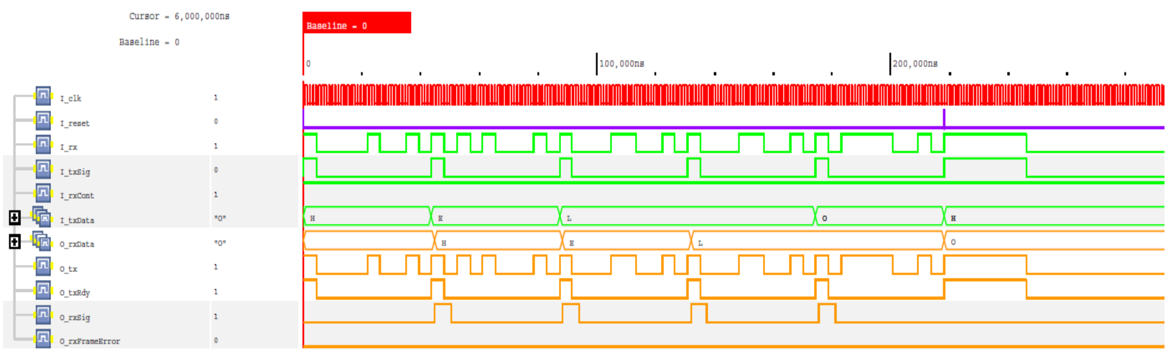
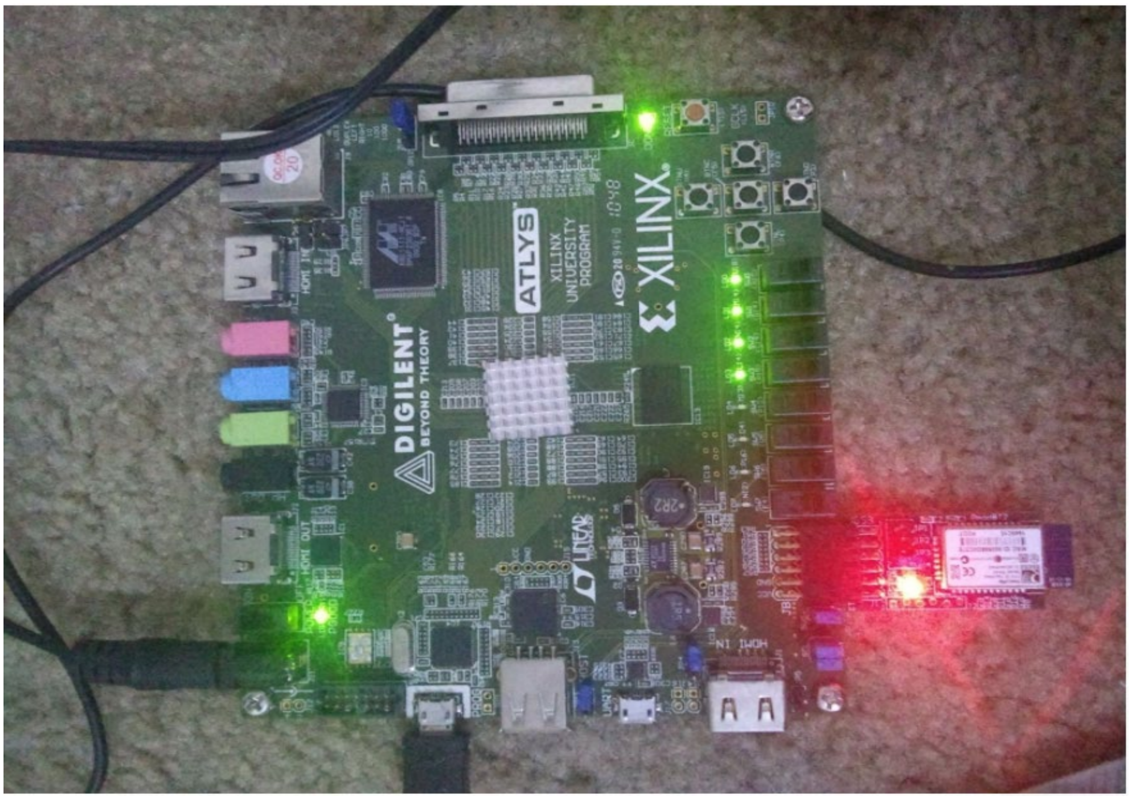
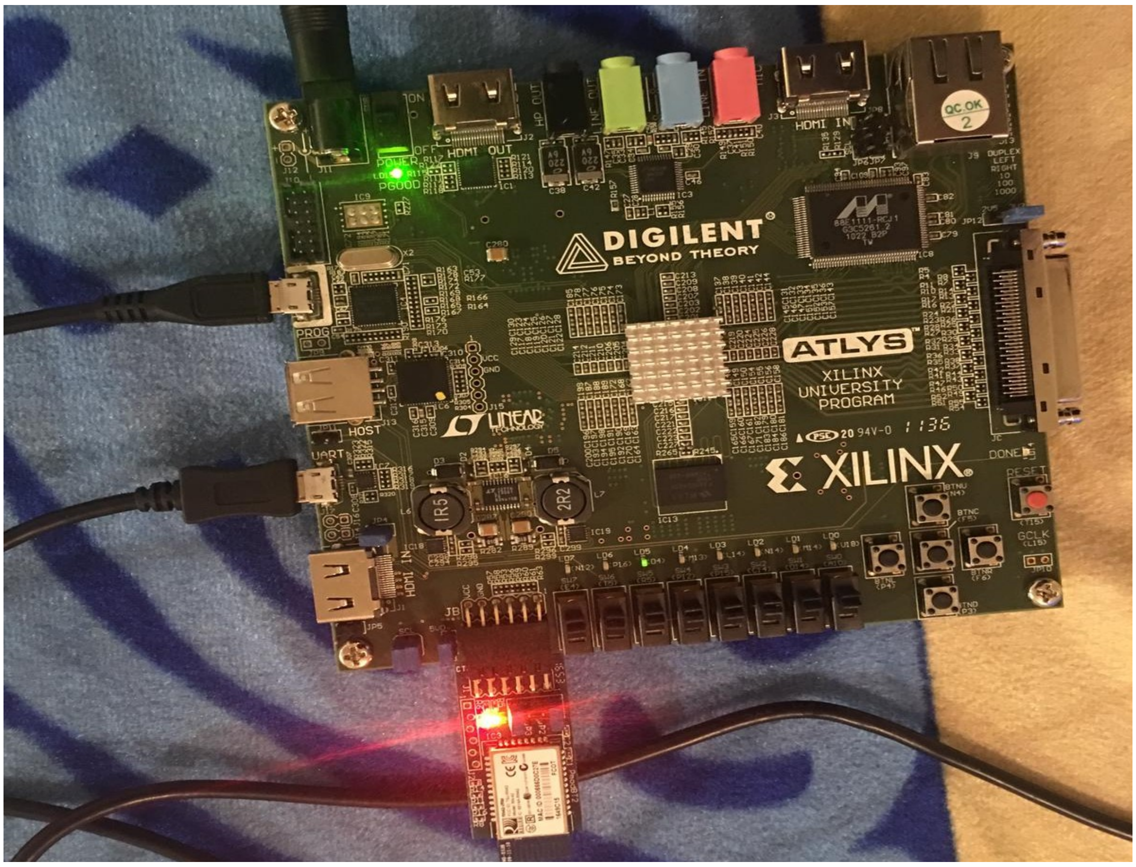


Fig. Simulation results of Loopback system

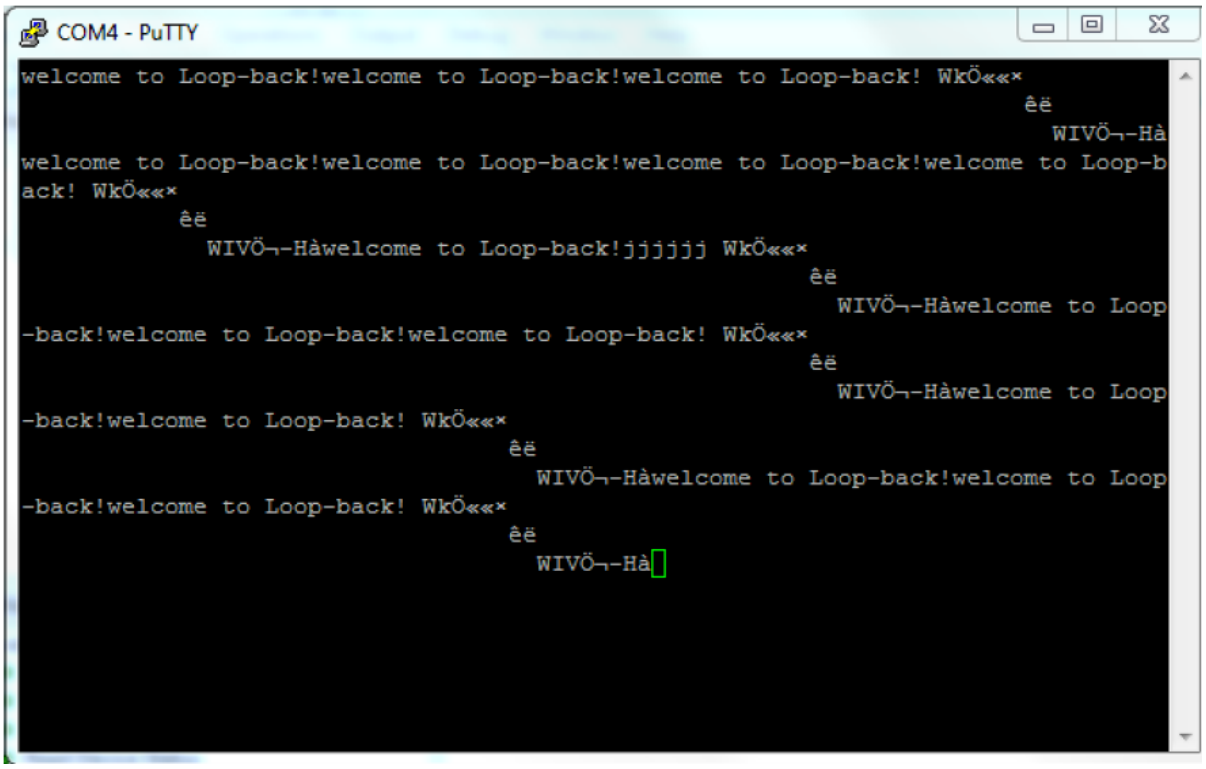
PROJECT 2 Sai Krishna Kandagatla
FINAL REPORT U15974415

Hardware results:



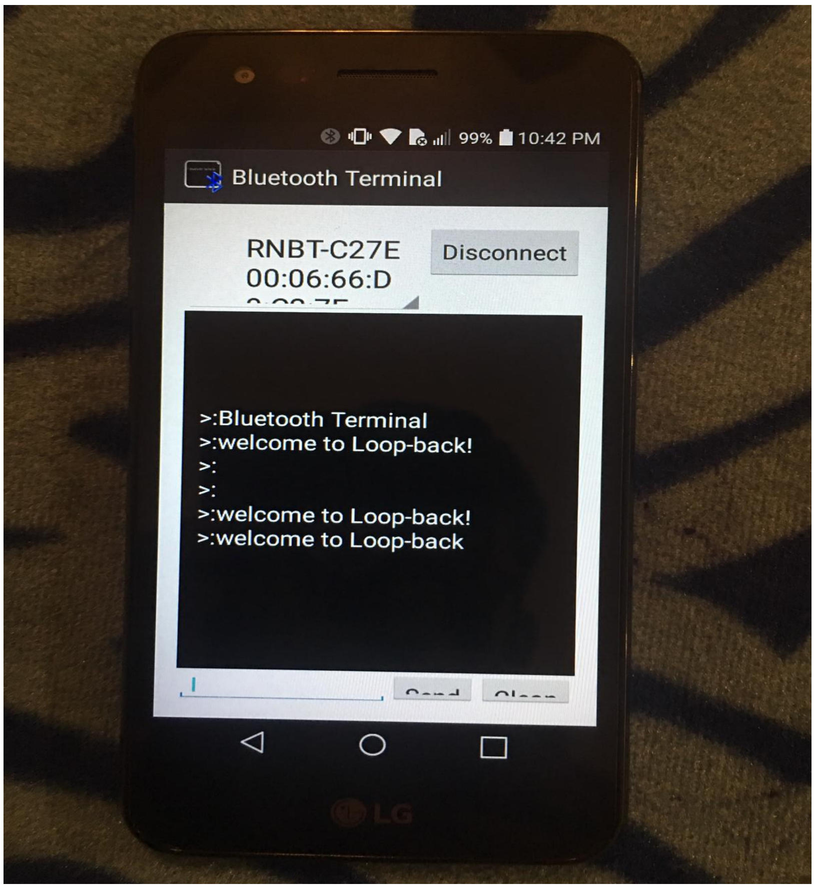


Figs. Spartan-6 implementing Loopback system and transmitting to a PC using UART



```
welcome to Loop-back!welcome to Loop-back!welcome to Loop-back! WkÖ«««
                                                                      êë
                                                                        WIVÖ¬-Hà
welcome to Loop-back!welcome to Loop-back!welcome to Loop-back!welcome to Loop-b
ack! WkÖ«««
          êë
            WIVÖ¬-Hàwelcome to Loop-back!jjjjjj WkÖ«««
                                                       êë
                                                         WIVÖ¬-Hàwelcome to Loop
-back!welcome to Loop-back!welcome to Loop-back! WkÖ«««
                                                       êë
                                                         WIVÖ¬-Hàwelcome to Loop
-back!welcome to Loop-back! WkÖ«««
                                  êë
                                    WIVÖ¬-Hàwelcome to Loop-back!welcome to Loop
-back!welcome to Loop-back! WkÖ«««
                                  êë
                                    WIVÖ¬-Hà
```

Fig. Output on Serial Terminal



```
>:Bluetooth Terminal
>:welcome to Loop-back!
>:
>:
>:welcome to Loop-back!
>:welcome to Loop-back
```

Fig. Output on Mobile Phone

Here I had clearly observed two points. First the Loopback is generating the data at 9600bps and the second Bluetooth is connected to the FPGA board indicated by single red light on Bluetooth module. Since the baud rate and clock are synchronised exactly we can see the output on the Serial terminal. From this Project we can come to know that synchronisation of Clock is important and how it is difficult to synchronize the clock.

Conclusion:

Loopback System requires the use of an efficient Multi processing system i.e., Pico Blaze Microcontroller. Although the output is achievable by serial communication, it can be beneficial to take advantage of the parallelism, low cost, and low power consumption offered by FPGAs. The successful implementation of this project illustrates that it can be efficiently implemented on FPGA hardware and it also uses the Bluetooth module successfully.

When we see the outputs obtained from FPGA on serial terminal, we find the outputs obtained using the Spartan6 kit and from the simulation are computationally efficient and same. The same output is obtained on the mobile phone through Bluetooth module.

Bibliography and Links:

[1] D.Antonio-Torres, D.Villanueva-Perez, E.Sanchez-Canepa, "*A Pico Blaze-Based Embedded System for Monitoring Applications*", Vol.5, No.2, March 2009.

[2] B.Muralikrishna, K. Gnana Deepika, "*Input/output Peripheral Devices Control Through Serial Communication Using Micro Blaze Processor*", 978- 1-4577-1545-7, IEEE,2012.

[3] A. Meixner, A. Kakizawa, B. Provost, and S. Bedwani, "*External Loopback Testing Experiences with High Speed Serial Interfaces*," IEEE International Test Conference, 2012.

[4] B. Provost and T. Huang, ''*AC IO Loopback Design for High Speed Microprocessor IO Test*,'' IEEE International Test Conference, 2014.

[5] Han Dehong, Zhang Xiancai, Li Xiangdong. "*Based on the FPGA serial controller design and its implementation* ", Journal of Air Force Radar Academy, 2010.

[6] V. S. P. Nayak, G. K. Saitejdeep, L. R. Kumar, N. Ramchander and K. Madhukar, "*Modular approach for customizable UART*," *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, 2016, pp. 748-750.

[7] R. Joaquinito and H. Sarmento, "*A wireless biosignal measurement system using a SoC FPGA and Bluetooth Low Energy*," *2016 IEEE 6th International Conference on Consumer Electronics - Berlin, Berlin*, 2016, pp. 36-40.

[8] Koji Nakano and Yasuaki Ito, "*Processor, Assembler, and Compiler Design Education Using an FPGA*" *2009 IEEE 14th International Conference, 2009.*

[9] Pico Blaze 8-bit Embedded Microcontroller User Guide for Extended Spartan-3 and Virtex-5 FPGAs Introducing Pico Blaze for Spartan-6, Virtex-6 and 7 Series FPGAs, UG 129 June 22, 2011.

[10] http://www.xilinx.com/picoblaze

[11] https://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf

[12] https://reference.digilentinc.com/reference/pmod/pmodbt2/reference-manual

Appendix

// I have implemented the loopback-based program in Verilog

// Design code

```
`timescale 1ns / 1ps

module loopback (switches, leds, rs232_tx, rs232_rx, reset, clk);

        input               reset;                  // Remember: ACTIVE LOW!!!

        input               clk;                    // 100 MHz

         // GPIO

        input     [7:0]     switches;

        output    [7:0]     leds;

        // RS232 Lines

        input               rs232_rx;

        output              rs232_tx;

        // Wires and Register Declarations

        // PicoBlaze Data Lines

        wire      [7:0]     pb_port_id;

        wire      [7:0]     pb_out_port;

        reg       [7:0]     pb_in_port;

        wire                 pb_read_strobe;

        wire                pb_write_strobe;

        // PicoBlaze CPU Control Wires

        wire                pb_reset;

        wire                pb_interrupt;

        wire                pb_int_ack;

        // UART wires

        wire                write_to_uart;

        wire                uart_buffer_full;

        wire                uart_data_present;

        reg                 read_from_uart;

        wire                uart_reset;


// UART Data Lines

        wire      [7:0]     uart_rx_data;
```

```
// LED wires
wire write_to_leds;
wire led_reset;
// LED Driver and control logic
assign led_reset = ~reset;
// LED driver instantiation
led_driver_wrapper led_driver (
        .led_value(pb_out_port),
        .leds(leds),
        .write_to_leds(write_to_leds),
        .reset(led_reset),
        .clk(clk)
);
// UART and control logic
// UART expects ACTIVE-HIGH reset
assign uart_reset =  ~reset;
// UART instantiation
rs232_uart UART (
        .tx_data_in(pb_out_port), // The UART only accepts data from PB, so we just tie the PB output to the UART input.
        .write_tx_data(write_to_uart), // Goes high when PB sends write strobe and PORT_ID is the UART write port number
        .tx_buffer_full(uart_buffer_full),
        .rx_data_out(uart_rx_data),
        .read_rx_data_ack(read_from_uart),
        .rx_data_present(uart_data_present),
        .rs232_tx(rs232_tx),
        .rs232_rx(rs232_rx),
        .reset(uart_reset),
        .clk(clk)
);


  // PicoBlaze and control logic
assign pb_reset = ~reset;
```

```
        assign pb_interrupt = 1'b0;
        // PB CPU instantiation
        picoblaze CPU (
                .port_id(pb_port_id),
                .read_strobe(pb_read_strobe),
                .in_port(pb_in_port),
                .write_strobe(pb_write_strobe),
                .out_port(pb_out_port),
                .interrupt(pb_interrupt),
                .interrupt_ack(),
                .reset(pb_reset),
                .clk(clk)
        );
        assign write_to_leds = pb_write_strobe & (pb_port_id == 8'h01);
        assign write_to_uart = pb_write_strobe & (pb_port_id == 8'h03);
        always @(posedge clk or posedge pb_reset)
        begin
                if(pb_reset) begin
                        pb_in_port <= 0;
                        read_from_uart <= 0;
                end else begin
                        // Set pb input port to appropriate value
                        case(pb_port_id)
                                8'h00: pb_in_port <= switches;
                                8'h02: pb_in_port <= uart_rx_data;
                                8'h04: pb_in_port <= {7'b0000000,uart_data_present};
                                8'h05: pb_in_port <= {7'b0000000,uart_buffer_full};
                                default: pb_in_port <= 8'b0000000;
                        endcase
                        read_from_uart <= pb_read_strobe & (pb_port_id == 8'h02);          end
        end
endmodule
```

//Test bench

```
`timescale 1ns / 1ps

module loopback_testBench;

	// Inputs

	reg [7:0] switches;

	reg rs232_rx;

	reg reset;

	reg clk;

	// Outputs

	wire [7:0] leds;

	wire rs232_tx;

	// Instantiate the Unit Under Test (UUT)

	loopback uut (

		. switches(switches),

		. leds(leds),

		.rs232_tx(rs232_tx),

		.rs232_rx(rs232_rx),

		. reset(reset),

		. clk(clk)

	);

	initial begin

		// Initialize Inputs

		switches = 0;

		rs232_rx = 0;

		reset = 0;

		clk = 0;

		// Wait 100 ns for global reset to finish

		#100;

		// Add stimulus here

	end

endmodule
```

PROJECT 2 Sai Krishna Kandagatla
FINAL REPORT U15974415

//Pico Blaze Code

```
`timescale 1ns / 1ps
module picoblaze (
        port_id,
        read_strobe, in_port,
        write_strobe, out_port,
        interrupt, interrupt_ack,
        reset, clk
  );
         // Port Specifier
         output   [7:0]      port_id;
         // Input port & acknowledge strobe
         output   read_strobe;
         input    [7:0]      in_port;
         // Output port & ready strobe
         output   write_strobe;
         output   [7:0]  out_port;
         // Interrupt input & acknowledge
         input    interrupt;
         output   interrupt_ack;
         // global reset and clock
         input reset;
         input clk;
         // Reset Handling Logic
         wire cpu_reset;
         assign cpu_reset = reset;
         wire pb_sleep;
         assign  pb_sleep = 1'b0;
         wire     [7:0]      address;
         wire     [7:0]      instruction;
         wire                bram_enable;
        kcpsm6 pblaze_cpu (
                .address(address),
                .instruction(instruction),
                .bram_enable(bram_enable),
                .port_id(port_id),
                .write_strobe(write_strobe),
                .k_write_strobe(),
                .out_port(out_port),
                .read_strobe(read_strobe),
                .in_port(in_port),
                .interrupt(interrupt),
                .interrupt_ack(interrupt_ack),
                .reset(cpu_reset),
                .sleep(pb_sleep),
                .clk(clk)
        );
        program pblaze_rom (
                .enable(bram_enable),
                .address(address),
                .instruction(instruction),
                .clk(clk)
        );

endmodule
```