# CDA 4203L Spring 2022

**Computer System Design Lab**

**Lab 6 – Loopback System on Picoblaze Microcontroller**

**Assigned on Wednesday, 2nd March**

**Due Date: 11:59 PM, Sunday, 13th March**

## Teaming allowed. No more than three (3) members per group.

**Objective:** To design and synthesize a loopback system using a PicoBlaze microcontroller, and to implement functionality using Assembly code.

**Description:** Consider a simple “loopback” system as in Fig. 1. In such a setup, a PC can send a message which is returned (looped) back to PC by the processor. Such a loopback test is helpful to test the processor is alive or not. These are already explained in course slides on Canvas at the beginning of the semester.

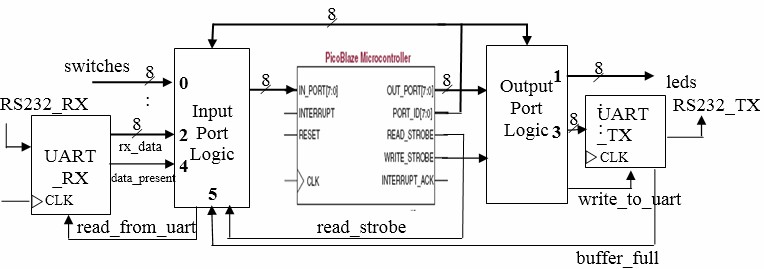


Fig. 1. A simple loopback system



Fig. 2. UART Transmit and Receive Blocks – High Level View

A PC (not shown) interfaces with PicoBlaze processor using an RS232 (serial) connection. Let us examine the case when the PC acts as a transmitter. In this case the data arrives serially on RS232 RX line. But we know that PicoBlaze accepts only byte-size data. So, we need to convert the serial data into parallel (8-bit) data. In such a case, a UART (Universal Asynchronous Receiver Transmitter) block can help (Fig. 2). UART\_RX does the serial-to-parallel conversion i.e., the data arriving on RS232\_RX line is buffered in a 16-byte buffer. Whenever new data is presented on rx\_data output, UART\_RX will assert data\_present i.e., data\_present = 1 in other words data\_ present validates rx\_data. The Input Port Logic block consists of glue logic that connects UART\_RX to the IN\_PORT. After PicoBlaze reads the data, it can request the next data item by asserting read by uart signal.

Now, let us consider the case when PicoBlaze acts as a transmitter i.e., PC is the receiver. In this case, PicoBlaze puts out 8-bit data while PC is expecting serial data. Again, UART (UART\_TX in Fig. 1) comes to our rescue! The output bus with id=3 is connected to UART\_TX. Output Port Logic block interfaces PicoBlaze with leds and UART\_TX. In this system, we can also manually provide 8-bit data via 8 switches (switches bus in Fig. 1). PicoBlaze can write 8-bit data to drive 8 leds (leds bus in Fig. 1). Recall that PicoBlaze can accept up to 256 input ports each of which can addressed by 8-bit PORT\_ID bus. Similarly, it can drive up to 256 ports and the address of the port being driven appears on PORT\_ID bus. In Fig. 1, for example, the id of the input port, switches, is 0.

## Lab 6 – Step by Step Instructions

It's strongly recommended that you perform the steps in the given order below.

**First, read again “PicoBlaze\_Development\_tools.ppt” in the lab 6 folder on Canvas which is basically related to PicoBlaze**

1. **Download and Unzip**
2. In your home directory, create a new folder ‘lab7’.
3. Download the loopback.zip file from canvas into ‘lab7’.
4. Unzip the file.

## Create ISE Project

1. Create a new ISE project.
2. Ensure that you are using the correct settings.
3. Add copies of all Verilog source files and the UCF file from the loopback directory.

## View the ‘loopback.v’ which is already written for you

1. Refer to Worksheet 1 and understand the answers at the end of this description (Page 4 below)
2. Open ‘loopback.v’ and read it.
3. Fill in the ‘picoblaze.v’ file (<FILL\_IN>).
4. Edit ‘rs232 uart.v’ in the folder and fill in with correct values where you see (<FILL\_IN>).

## Edit ‘loopback.UCF'’

1. Refer to Worksheet 2 and understand the FPGA Pin Assignments (Page 4 below)
2. Open ‘loopback.UCF'’ and fill in the blanks.

## Write Assembly Code

1. Open the “picoblaze” folder in “loopback”.
2. Open the ‘program.psm’ file with a text editor.
3. Enter your assembly code based on the below explanations:

We start with this part to get familiar with the steps. However, eventually, in Step 9, we need to have a combined three programs and re-assemble every time we update our PSM file.

In your lecture slides, we already have an example for part of

* 1. **(25 pts) Code 1: cold\_start:** code to output a message "Welcome to Loop-back!" to the serial port. You need to encode the message in ASCII format. CAUTION: UART\_TX buffer size is only 16 bytes.

Note that this file already contains many constants and some place-holders for your program. The ASCII codes are already entered as constants. For example, if you want to load the ASCII code for ‘h’ then you can write: **Load sX, ascii\_h**

1. Read the assembler.txt file to go over how to assemble your PSM file, it is easy. Note that ROM\_form.v and the PSM files need to be in the same directory of the assembler.

To assemble the code, the syntax is: **./picoasm -i program.psm -t ROM form.v**

1. It will generate a file, ‘program.v’. Add a copy of this file to your project. It contains the assembled code for PicoBlaze.

## Each time you modify your PSM file, you need to run the assembler again so that the changes are also included in the HDL program memory definition file.

1. **FPGA Synthesis and Implementation**
2. Add all the “.v” Verilog files including the ones in the loopback/verilog folder and also the generated program.v file to the project. Synthesize and generate the .bit file (bitstream file).
3. Load the design onto the board.

## USB/mini USB Cable Connection

Connect the USB serial cable to the computer, as well as to the FPGA board.

## Connect to Serial Port

Use PuTTY to connect to the serial port, the connection type will be "serial" and the serial line should be mentioned properly.

Use the below pin mapping for Tx and Rx in your .ucf file:

NET "rs232\_tx" LOC=T20; NET "rs232\_rx" LOC=T19;

## Refer to Canvas for “Connecting to serial port.pdf” file created for you to explain this.

1. **Finish**

Write assembly code for led echo program and rs232\_echo program, reassemble the combined PSM file.

* 1. **(25 pts) Code 2: led\_echo:** code to read switches and write the values, inverted, to the LEDs.
  2. **(25 pts) Code 3: rs232\_echo:** code to check if a byte has been received by UART\_RX. If so, send it back to the PC via UART\_TX.

The example codes in your lecture notes are just one of many ways to perform the task. Use the presented codes in lecture notes (you might need to modify them or complete them if you feel something is missing) or come up with your own codes.

## Design Constraints:

* The design must both functions correctly in simulation and on the FPGA board.
* The 100MHz Clock is used as the system clock.
* Every time the design is initialized (reset) it runs cold start.
* Until it is turned off or reset, it performs the following two actions continuously:
  + Echo inverted Switch values to LEDs.
  + Echo RS232 data back to PC.
* It must do this indefinitely.
* The 'RESET' button must provide a global reset functionality. This includes resetting the microprocessor.

## Worksheet 1

1. What is the instruction bus width for the ROM?
2. For the UART 16x bit rate counter, the maximum baud count using the following formula:

*max\_baud\_count = fclk/ (16\*req\_baud\_rate)*

If fclk = 100MHz and the required baud rate is 9600, what is the maximum baud count?

1. Write the boolean expression for the write to uart signal.
2. Write the boolean expression for the write to leds signal.
3. What are the port\_id values for the following input ports?
   1. switches:
   2. data\_rx:
   3. data\_present:
   4. buffer\_full:
4. Write the boolean expression for the read\_from\_uart signal.
5. Continue with step 3 (b) on Page 2 above.

## Worksheet 2

We have filled in the pin locations for the following components/signals. Refer to the FPGA Tutorial and Anvyl\_Master.ucf available on Canvas from your previous labs.

1. Clock (100MHz):
2. Push Button BTN0 (for reset signal):
3. LED 0:
4. LED 1:
5. LED 2:
6. LED 3:
7. LED 4:
8. LED 5:
9. LED 6:
10. LED 7:
11. SW 0:
12. SW 1:
13. SW 2:
14. SW 3:
15. SW 4:
16. SW 5:
17. SW 6:
18. SW 7:
19. RS232 TX (FPGA\_SERIAL1):
20. RS232 RX (FPGA\_SERIAL1):

Now continue with Step 4 (b) on Page 2.

**Deliverables (Only one .zip file per group)** A zipped file (.zip) which includes these two items:

1. Your group assembly code
2. A concise PDF group report (that includes your assembly code)

**Who submits?** Only one of group members needs to submit.

## Report Organization:

* + Cover sheet
  + Demonstration page (available on Canvas have it signed/marked by TA, attach to your report)
  + Your design details and assembly codes for above 3 problems
  + Feedback: Hours spent, Exercise difficulty (Easy, Medium, Hard)