Computer System Design - Lab 06

Xilinx PicoBlaze Microcontroller

Team ISIS

Team Member:	U#:	Work Distribution:
John Gangemi	6871-4612	33.3%
Chris Frazier	5883-1412	33.3%
Bassam Saed	5260-7130	33.3%

Introduction

PicoBlaze is an 8 bit soft microcontroller that can be synthesized onto a Xilinx FPGA board. We will attempt to use PicoBlaze to design and synthesize a loopback system and implement the functionality using assembly code. In order to accomplish this task, we must understand the architecture of PicoBlaze and how to manipulate it.

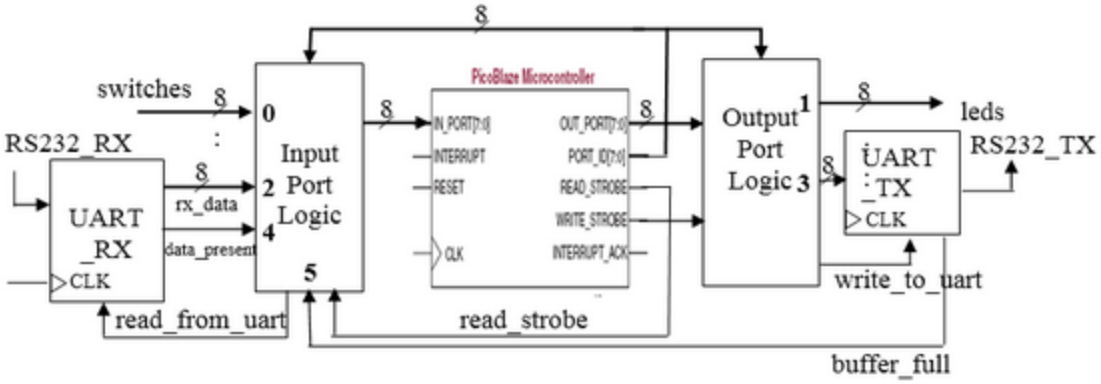
Our loopback system will have two goals:

1. Echo data in the form of keystrokes from a PC through the microcontroller via serial RS232 and back onto the PC screen.
2. Receive data from the onboard switches, invert the signals, and transmit the results to the onboard LEDs.

The first goal is a great way to test the microcontroller for basic functionality. It will receive data serially, pass through the PicoBlaze, and transmit the same data serially back to the PC. The second goal will test our knowledge of assembly code and PicoBlaze by taking the data from the switches as input, inverting it using instruction code, and outputting it to the LEDs.

Design

The loop back system will be based on the Universal Asynchronous Receiver Transmitter that PicoBlaze uses. The following diagram explains how loopback and UART works:



When the read_strobe signal strobes and the buffer_full signal is low, the Input Port Logic module uses the read_from_uart signal to receive the converted data from UART_RX. This data is then used by the PicoBlaze. When the write_strobe signal strobes, the Output Port Logic module uses the write_to_uart signal to transmit data to the UART_TX module, which serially transmits the data. The UART_RX module converts the data from serial to parallel bytes so that the PicoBlaze can use the data. The UART_TX module converts the data from parallel back to serial and into the buffer to be transmitted. Data from the switches can be input directly to the Input Port Logic module, and the LEDs can use the byte sized data directly from the Output Port Logic module.

Testing

Verification of the design required that the underlying assembly code be functionally correct, this could be proved through simulation using the pBlaze IDE. This meant that the KCPSM6 mnemonics had to be converted to the mnemonics supported in pBlaze, a rather trivial task. Upon successful compilation of the pBlaze assembly code (program_testbench.psm) a number of simulations were performed as detailed…

- Testing *switches* 8-bit input
- Testing *uart_data_rx* 8-bit input
- Testing *data_present* 1-bit input
- Testing *buffer_full* 1-bit input

In order to verify the design's functionality through a Verilog Test Fixture the simulation time had to be increased significantly to verify the 'cold_start' routine, therfore the only useful test was to verify the 'led_echo' routine. The test vector for the 'led_echo' routine simply inverted every other LED and simulation time had to be set to 6 milliseconds to generate an expected result. Testbench code for the top-module 'loopback.v' is shown below…

```
always begin
  clk = 1'b0;
  #5;
  clk = 1'b1;
  #5;
end

switches = 8'b00000000;
rs232_rx = 0;
reset = 0;

#20;
reset = 1;
switches = 8'b01010101;
#6000000 $finish;
```

Synthesis

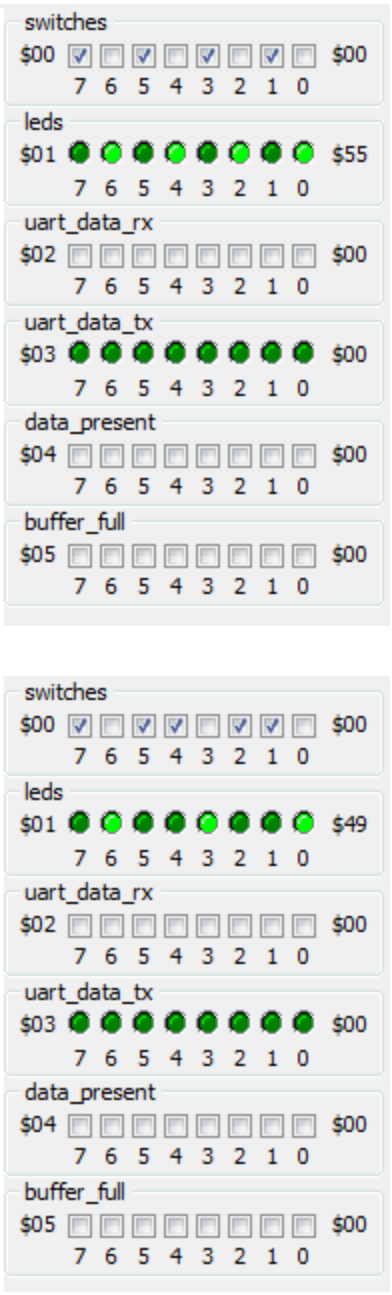
Generating a functional bit file from top-module synthesis proved successful upon initial project compilation. There were numerous warnings regarding 'FF/Latch' of a certain module that was not part of the necessary modules to alter and thus these warnings were ignored.

Testing a physical implementation of the design involved a variation of asserting reset, supplying input to the console from the keyboard, and methodically altering the configuration of all 8 switches.

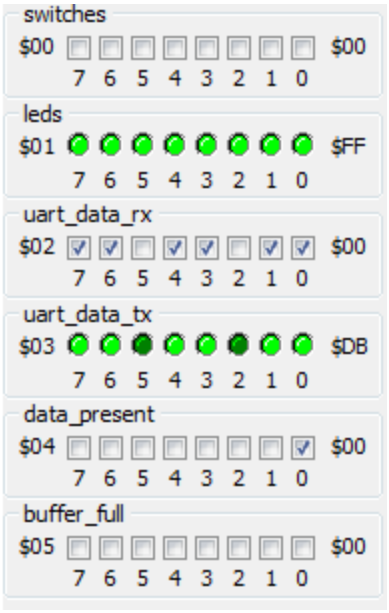
Results

Simulation results from pBlaze confirmed the assembly code was functionally correct. A number of images from various test vectors mimic the operation of the UART and PicoBlaze except inputs are controlled by the user.

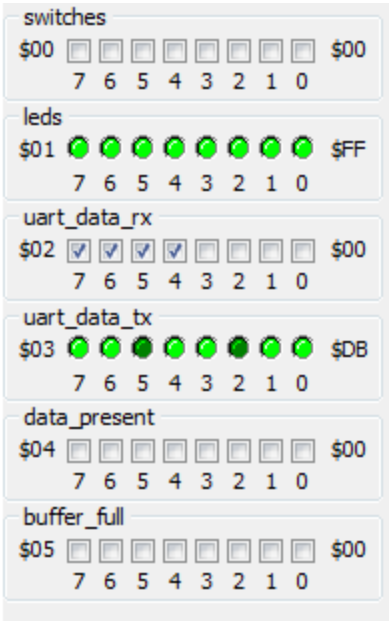
These two images show the *switches* input directly changing the *leds* output, this occurred in realtime during simulation.



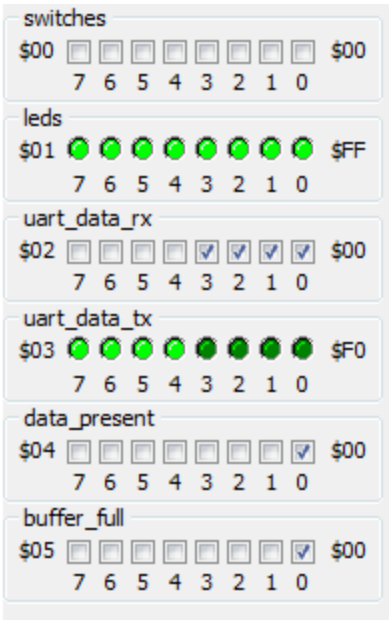
This image depicts the transmission of data supplied by *uart_data_rx* input while *data_present* is asserted high and *buffer_full* is held low.



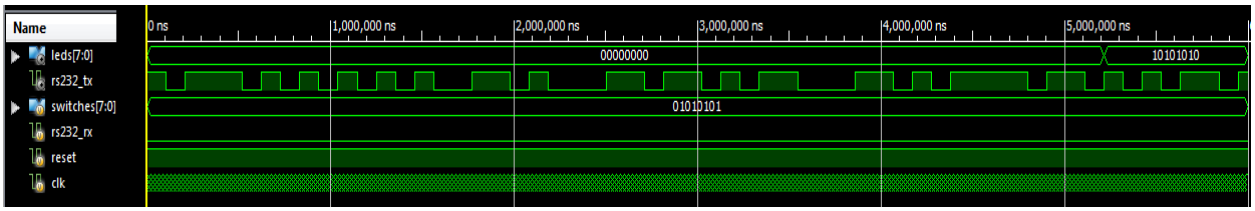
This image depicts the “halt” of data transmission supplied by *uart_data_rx* while *data_present* is asserted low regardless of *buffer_full* state.



This final pBlaze simulation image shows the “halt” of data transmission when *buffer_full* is asserted high regardless of *data_present* state.

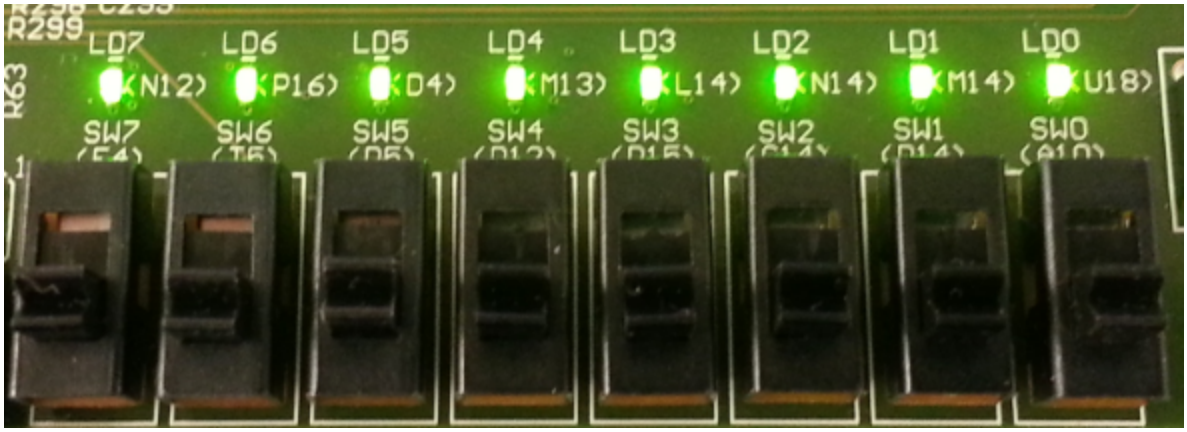


Simulation results from the Verilog Test Fixture of the top-module ‘loopback.v’ proves the functionality of the ‘led_echo’ routine using the PicoBlaze soft-microcontroller implementation. After roughly 5 ms the *leds* output changes to 10101010 accordingly given the *switches* input of 01010101.

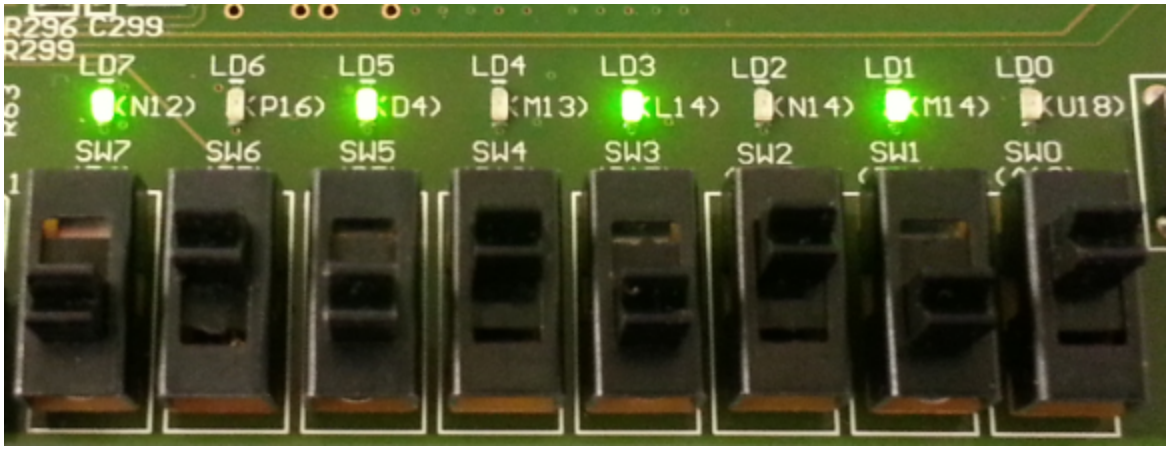


Images gathered from physical implementation of the design depicts the realtime operation of PicoBlaze and UART on the Atlys FPGA board.

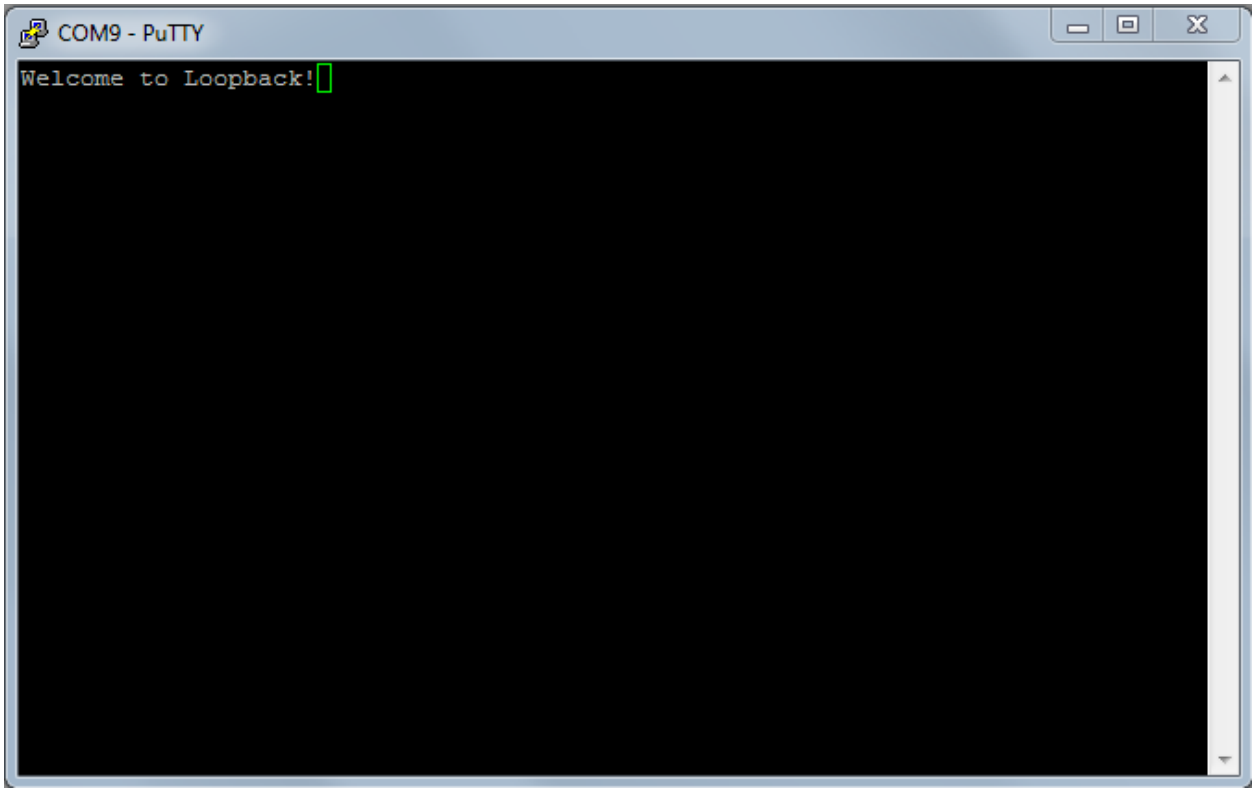
LED (led_echo) Test 1



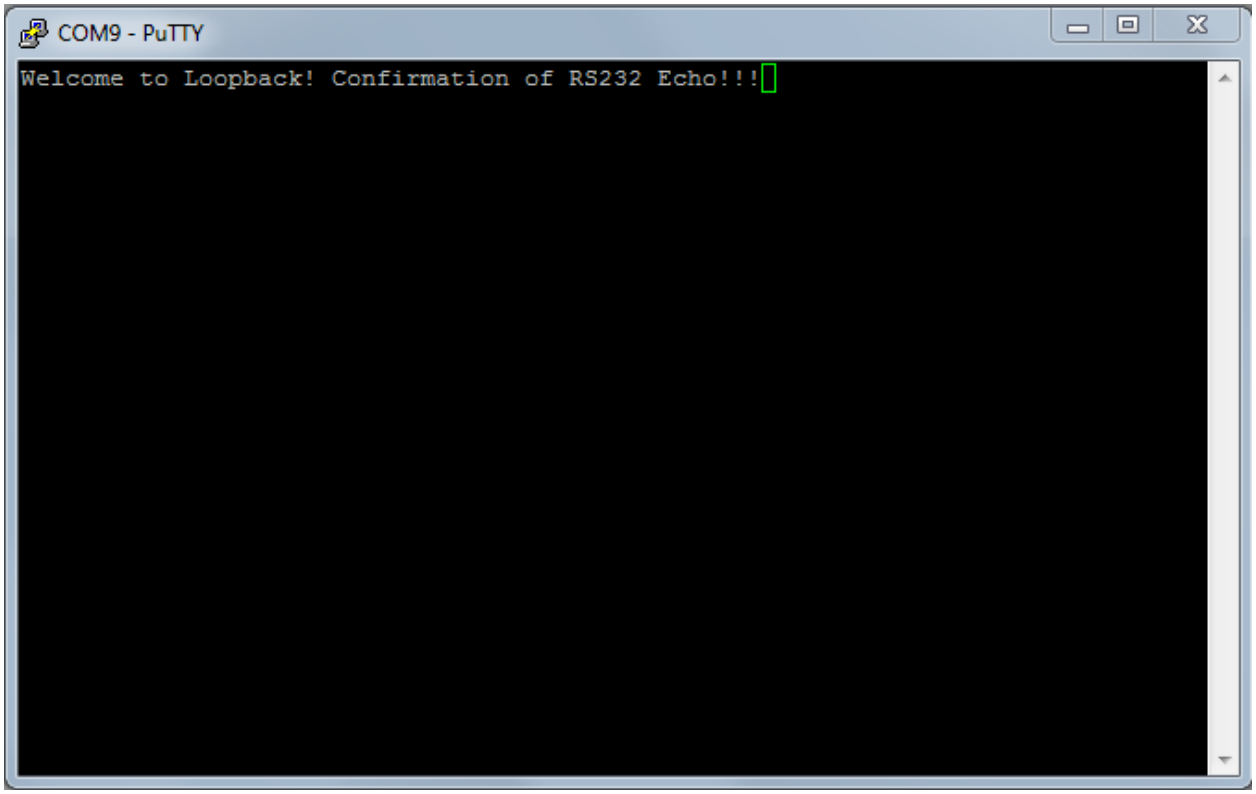
LED (led_echo) Test 2



Reset (cold_start) Test



Console Input (rs232_echo) Test



Conclusion

In previous labs we only saw relatively simple combinational and sequential circuits that performed easy tasks. This assignment demonstrates the amazing flexibility and capability of FPGAs. We saw how PicoBlaze, a completely functional 8 bit soft microcontroller, can be synthesized and utilized on an FPGA board. This lab also highlights the capabilities of hardware descriptive languages like Verilog, which can simulate entire processors without forcing us to spend a dime on manufacturing or fabrication. One aspect of this assignment we (and many others) struggled with was the mislabelling of the transmitter and receiver pins. This is due to the confusing documentation which refers to the UART_RX as the transmitter and the UART_TX as the receiver (from the USB-UART's point of view it is correct, but not from our point of view). Luckily, we received some advice from other students and this didn't cause us to waste too much time. All we had to do was switch the two pins and it worked fine. In summary, this was a very enlightening lab for all of us, and the possibilities for FPGAs have been broadened immensely. Up next is the final project, and we are excited to get started.