Lab 6 Report

by *Automagically*

Sean Murphy
U49850246
(33%)

Jesse Walton
U89823440
(33%)

Austin Matthew
U89904116
(33%)

Introduction:

The purpose of this lab is to implement a system that can echo RS232 and respond to switch data simultaneously. This implementation requires the use of the KCPSM6 microcontroller (aka PicoBlaze), as well as the assembly code as programmed in pBlaze, to serve as the 'controller' of the project. It is also necessary to instantiate a UART and link it to the physical USB micro B connector using the port connector file.

While most of the code is prewritten, this lab serves as a walkthrough for our first time using the simulated embedded microcontroller. The flexibility of this design, and small space requirements allow for easy, 'drop-in' functionality of a microcontroller that allows for maximum functionality and utility.

After completing this lab, we expect to have a solid grasp on how the FPGA emulates a microcontroller and connects to it's instruction set to simulate a physical microcontroller physically programmed with assembly code. We also expect that, when we see how easy it is to implement this dynamic hardware with such a low time and cost overhead, we'll likely prefer implementing many future hardware designs this way.

Design:

overview

At a high level, the design of this project is the construction of the KCPSM6 microcontroller, paired with its assembly code, along with the datapath and UART. The datapath connects the microcontroller to its various IO, specifically:

- interrupt
- out_port[7:0]
- in_port[7:0]
- port_id[7:0]
- write_strobe
- read_strobe

The in_port and out_port are muxed by the port_id line to allow the KCPSM6 instance to access multiple different hardware modules.

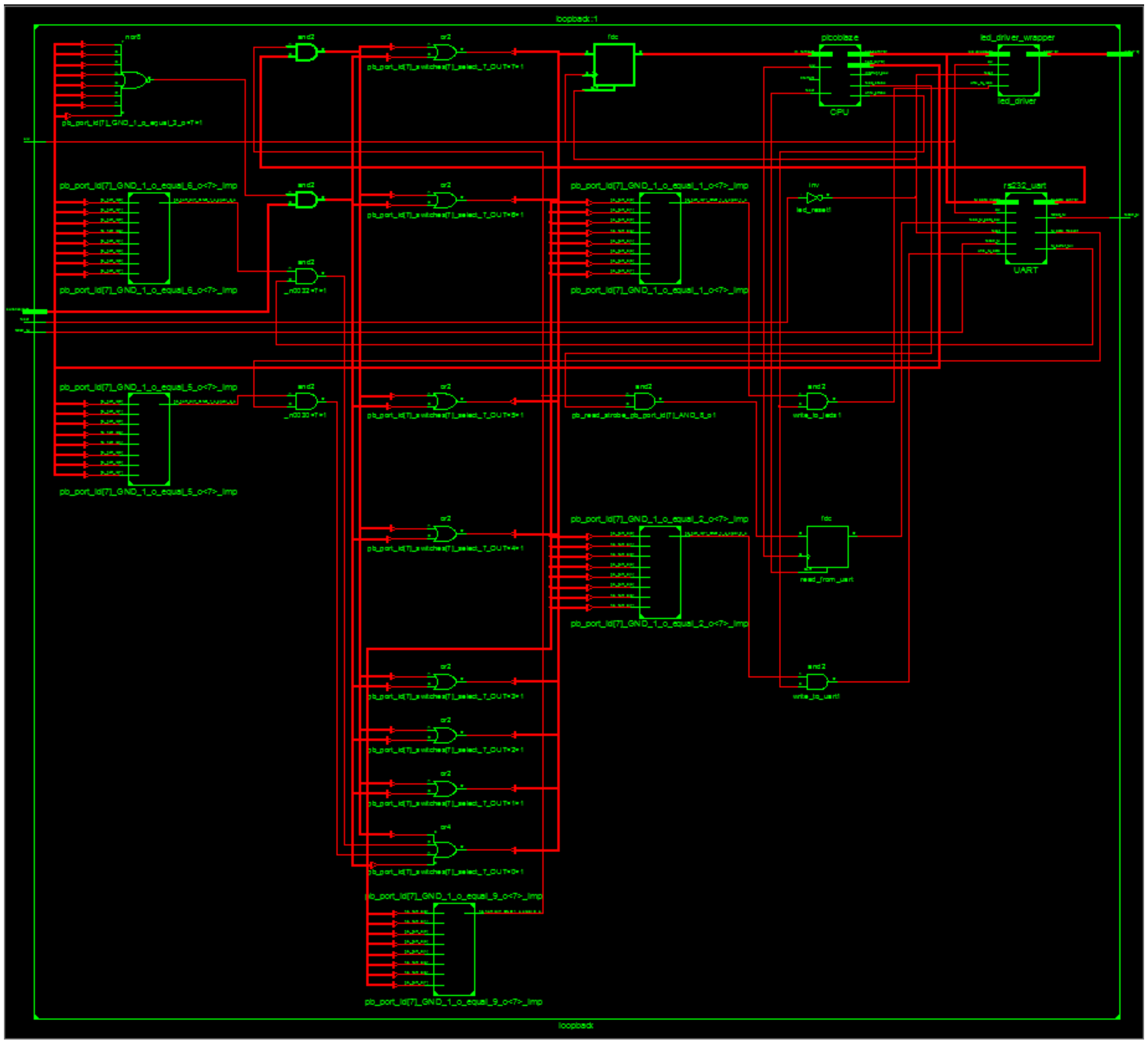
functionality

The functionality required for this project is implemented using the KCPSM6 embedded microcontroller and assembly code on the Spartan 6 FPGA. Since the KCPSM6 is essentially an emulated microprocessor with most structure predefined, the code required very little modification in Verilog. Once the KCPSM6 code was compiled into a verilog file, the only modification made was a change to the module name to match what was expected by the rest of the project.

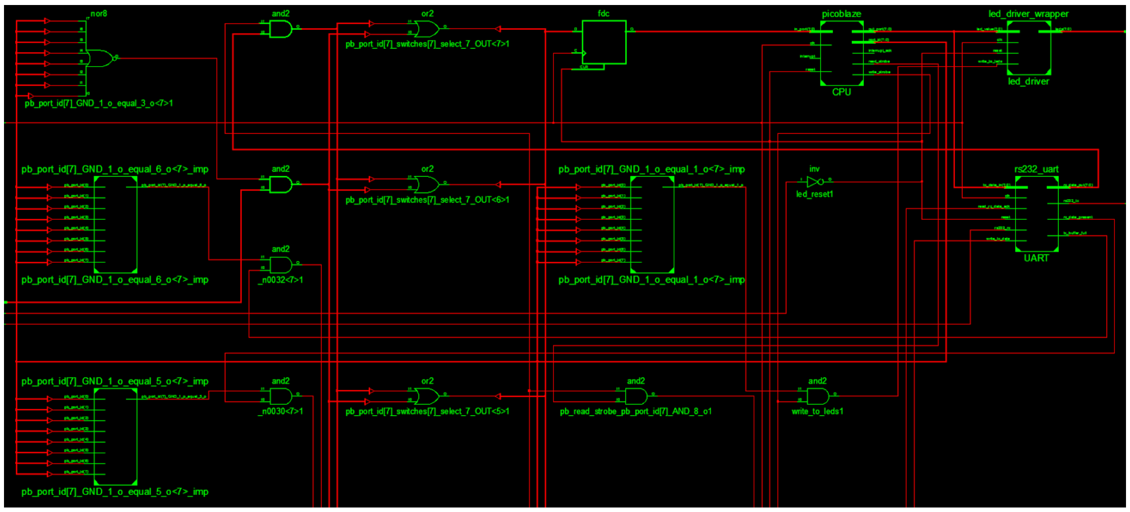
This design requires the separate implementation of the microprocessor and datapath, both of which are assembled on the FPGA using Verilog and implemented using the Xilinx Project Manager. Again, since we are implementing such a common component as the UART, the structure is already predefined for the most part. The UART module did require us to update the connections to the microcontroller to properly mux the data and signals such as buffer_full.

The microprocessor is implemented as a copy of the KCPSM6 along with assembly code to serve as the instruction set. The provided code was update to match the address length (4K instructions requires a 12bit address) and the instruction length (instructions are 18bits wide).

The KCPSM6 code was only slightly modified from our solutions to the previously assigned homework. Instead of three different modules for coldstart, led inverting, and loopback, a single program was created that would execute coldstart once then loop continuously between the led inverting and serial loopback.

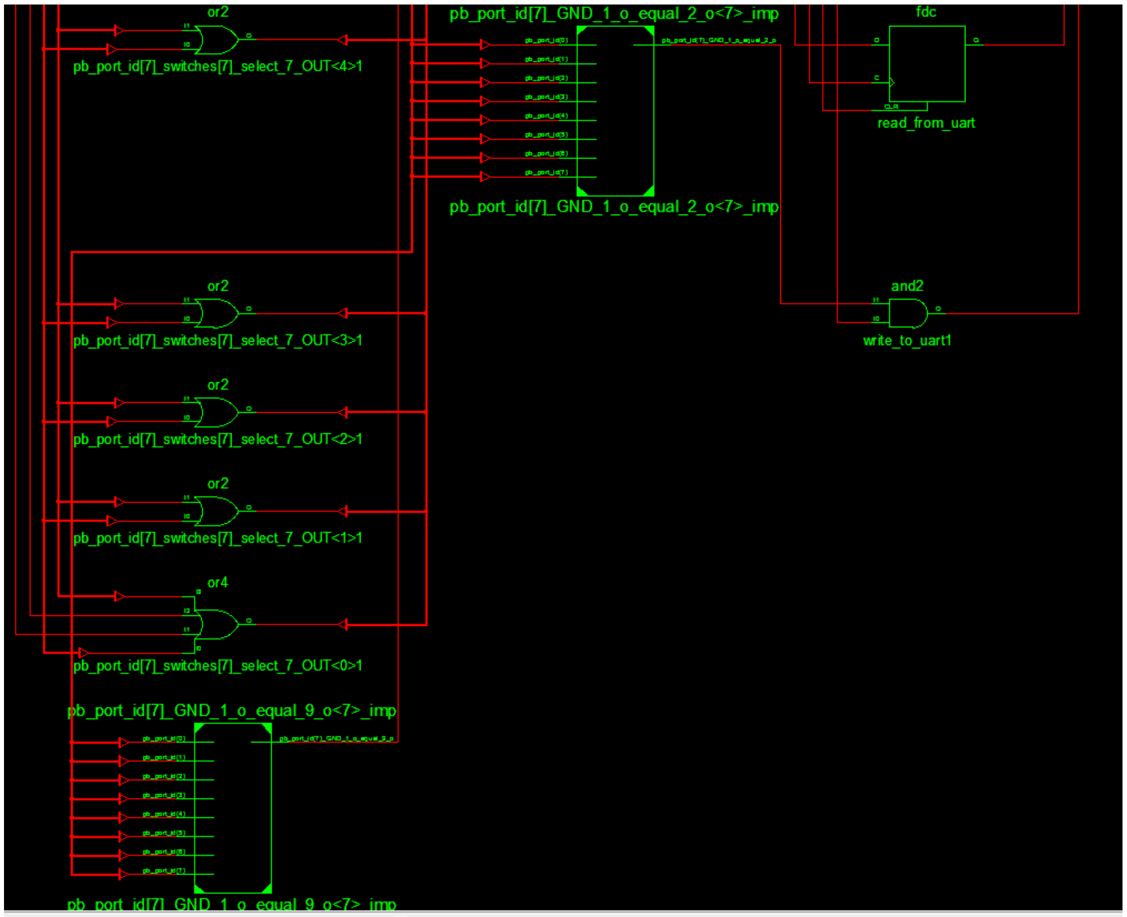

loopback:1
picoblaze
CPU
led_driver_wrapper
led_driver
rs232_uart
UART
fdc
read_from_uart
write_to_leds1
write_to_uart
led_reset1
pb_read_strobe_pb_port_id[7]_AND_6_o1
pb_port_id[7]_GND_1_o_equal_6_o<7>_imp
pb_port_id[7]_GND_1_o_equal_5_o<7>_imp
pb_port_id[7]_GND_1_o_equal_1_o<7>_imp
pb_port_id[7]_GND_1_o_equal_2_o<7>_imp
pb_port_id[7]_GND_1_o_equal_9_o<7>_imp
loopback

Full Schematic



nor8
pb_port_id[7]_GND_1_o_equal_3_o<7>1
and2
or2
pb_port_id[7]_switches[7]_select_7_OUT<7>1
fdc
picoblaze
CPU
led_driver_wrapper
led_driver
pb_port_id[7]_GND_1_o_equal_6_o<7>_imp
pb_port_id[7]_switches[7]_select_7_OUT<6>1
pb_port_id[7]_GND_1_o_equal_1_o<7>_imp
inv
led_reset1
rs232_uart
UART
_n0032<7>1
pb_port_id[7]_GND_1_o_equal_5_o<7>_imp
_n0030<7>1
pb_port_id[7]_switches[7]_select_7_OUT<5>1
pb_read_strobe_pb_port_id[7]_AND_8_o1
write_to_leds1

Zoomed upper half



or2
pb_port_id[7]_switches[7]_select_7_OUT<4>1
pb_port_id[7]_GND_1_o_equal_2_o<7>_imp
pb_port_id(0)
pb_port_id(1)
pb_port_id(2)
pb_port_id(3)
pb_port_id(4)
pb_port_id(5)
pb_port_id(6)
pb_port_id(7)
pb_port_id[7]_GND_1_o_equal_2_o
fdc
D
C
CLR
Q
read_from_uart
and2
write_to_uart1
pb_port_id[7]_switches[7]_select_7_OUT<3>1
pb_port_id[7]_switches[7]_select_7_OUT<2>1
pb_port_id[7]_switches[7]_select_7_OUT<1>1
or4
pb_port_id[7]_switches[7]_select_7_OUT<0>1
pb_port_id[7]_GND_1_o_equal_9_o<7>_imp
pb_port_id[7]_GND_1_o_equal_9_o

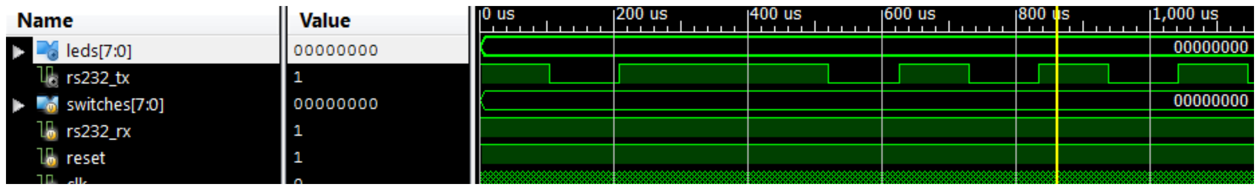
Zoomed lower half

Testing:

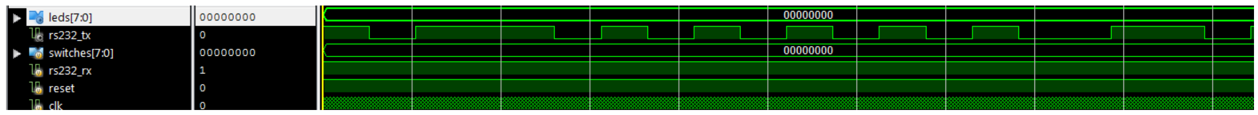
We took the following scenarios and tested them in the simulator and then again on the physical board.

1. Initial power on should output "Welcome to loopback!"
2. At any time, serial data transmitted to the FPGA should be echoed back.
3. At any time, the leds should reflect the inverted state of the switches

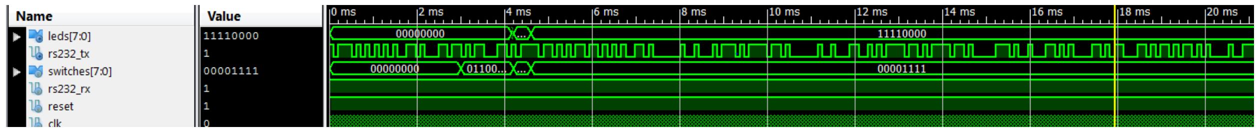
Capital "W" in reversed Ascii code. initiate/constant high signal at front and rear. Also start bit is present.



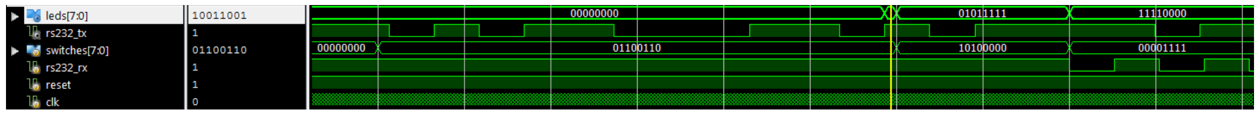
"We". both letters reversed in Ascii



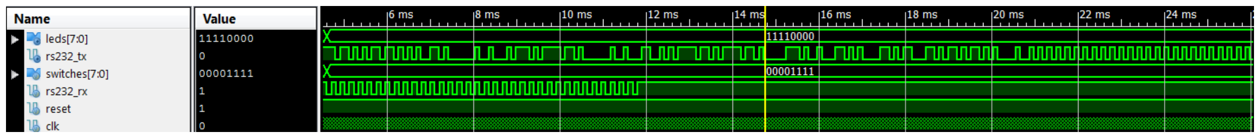
Full "Welcome to loopback!"



Leds responding to switch positions

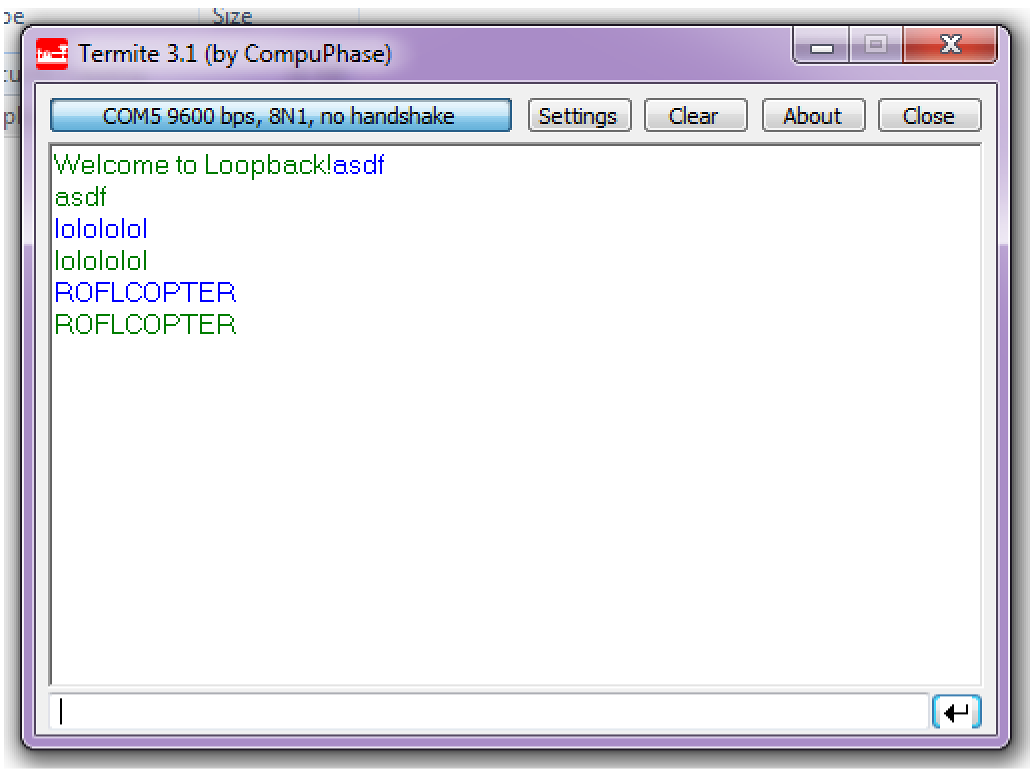


Information sitting on buffer, and then properly displaying after "welcome to loopback!"



Results:

All test results were compared with both their conceptual and waveform diagrams. Initially, the echo function was not working, but was fixed with a change to a different serial communication terminal. After that change, each test returned results that were within their expected behavior, as a result, all diagnostic tests were a success.



```
Welcome to Loopback!asdf
asdf
lolololol
lolololol
ROFLCOPTER
ROFLCOPTER
```

Conclusion:

In conclusion, this lab introduced the process of implementing the KCPSM6 microcontroller on the Spartan 6 FPGA from start to finish. In addition to implementing the microcontroller, we also programmed it by writing the assembly code and converting it to a verilog module. The assembly language was written in pBlaze IDE and converted into a hardware implementation of the code so that it could be implemented through the use of LUTs on the FPGA. A UART was created from a reference design as it is a common component and a datapath was constructed to connect the microcontroller to the external IO (via the switches and leds) and the UART, which was also connected to the external IO (via the USB port).

This was a very interesting lab as we've come full circle in designing and implementing discrete modules, the KCPSM6 microprocessor with assembly program and the external inputs/outputs. The UART gave us more experience with implementing predefined modules into our system so that modules can be reused without having to be recreated each time. This approach seems very similar to programming and makes designing complex systems much faster and easier.

It was very cool to see the software that we wrote, be converted into physical hardware emulation that was able to interact with the outside world and process data correctly. This project specifically gives you the feeling that you could create a system, as needed, for any situation. The flexibility of the FPGA, combined with the external ports to the physical world, as well as the ease of which it is programmed via the Xilinx IDE, makes this whole process incredibly simple. I can't imagine how long creating a similar project would have taken, given only traditional tools like logic gates and/ or designing and printing a circuit board or manufacturing a custom ASIC.