Computer System Design Lab # 3
RS232 Signal Generation - Hardware

Amy Guo
Lab Partner: Charlie Coleman

January 29, 2018

Pre-Lab: N.A.

Objective: The objective of this lab is to use an FPGA to transmit an RS232 and read using a PC.
Circuit Diagram: N.A

Outcome Predictions: It is expected that we will be able to successfully implement the VHDL code required to implement a sort of rudimentary UART that sends a simple message across an RS232 channel.

Equipment:
- Xilinx Spartan 3 on a Mesa 4i38
- PC

Procedure:
1. Using Xilinx ISE, create a project with a baud rate generator, process controller, address incrementer, character ROM, and PISO.
2. Create a UART component that will implement each part
3. Implement the baud rate generator
   (a) The only input for a baud generator is the master clock
   (b) The only output is the baud clock
   (c) Divide the frequency so that the output frequency is 9600 baud
4. Implement the process controller
   (a) The input to the process controller is the baud clock
   (b) The output is 2 signals, one to trigger the load from memory, one to increment the address
   (c) We want to trigger the address increment 2 bit periods before the load signal is sent
5. Implement the address incrementer
   (a) Input is the address increment signal from the process controller.
   (b) Output is the current address for the character ROM
   (c) Cycle through the addresses, incrementing at each address increment signal
6. Implement the character ROM
   (a) Input is the address
   (b) Output is the character
   (c) Using an array of characters, implement a character ROM that holds a preset message
   (d) Set the character output to be the character at the address location
7. Implement the PISO
   (a) The PISO accepts the baud clock, load signal, and character data as inputs
   (b) The PISO outputs a std_logic
   (c) The PISO should accept the character data when the load signal goes high, and should output the character 1 bit at a time (including start and stop bits)
8. Add each component to the UART, and generate the programming file
   - fix any errors in the code
9. Complete the pin setup
10. Copy the programming file onto the Xilinx Spartan 3
11. Implement the file, verify the outputs on the RS232 channel using the PC
12. Connect the channel to the BlueFruit
13. Verify correct functionality on the BlueFruit using nRF Connect

Recalculations and Predictions: N.A

Data and Observations: No data was collected from this lab, but we were able to observe the output data from the Spartan 3 on the PC. The output matched the data that was meant to be transmitted, and this is how we ensured that our design was operational.

Analysis & Discussion: We saw RS232 communication happen. We did not collected data for this experiments, but from observation we know that the data was received and read correctly, so that means that our

design was good.

Lab Questions:

Q: How are PC baud rates historically generated?
A: Baud clocks were generated by dividing the master clock in the system
Q: How are baud rates generated in your FPGA?
A: The baud rate is generated by dividing the 50MHz master clock
Q: What is the percent difference in the PC baud rate and the baud rate of your designs?
A: The generated clock is 9600.6 baud, so the difference is 0.0064%
Q: What is the maximum allowed difference in baud rates at the rate you choose, demonstrate your answer with a plot, annotation, and text.
A: It is recommended to be within 1-2% of the desired baud rate, so at 9600 baud, we can be off by about 96-192. If we sample at the middle of each bit, we will be able to sample 25-50 bits before the drift will cause losses.

Results: This lab was a success. We were able to create a VHDL description that correctly enabled RS232 functionality and output a string from character ROM.

Conclusions: In this lab, we gained experience with common hardware components in RS232 communication. Most components involved in the design are applicable to all types of communication inside hardware, such as the clock divider, PISO, etc. Because these components are universally useful for the most part, they can be used in any future hardware design that needs component-to-component communication.