MP-1 Write-Up

Blake Vermeer
Kris Hall
Rohit Zambre

September 24, 2014

Date Performed: September 19, 2014
Instructors: Joseph Zambreno

1 Objective

In this lab we were introduced to the Xilinx tools and the lab environment setup. Also, we did some basic VHDL coding to manipulate a UART link.

2 Initial Test

After generating the bitfile, we uploaded the file to the Xilinx board. After running the program, LEDs 1 and 4 lit up and remained lit. I then connected to the Xilinx board over the serial link using minicom. The Xilinx board simply echoed back any data sent to it over the serial link.

3 Uppercase Conversion

The TX_driver signal is the input to the hardware module of MP1_Top (TX_driver is the output of the MP1_top_driver module that acts as the UART for the simulation) while the RX_driver is the output of the hardware module of MP1_top. In the screen-shot below, we can see that the RX_driver signal is essentially a delay of the TX_driver signal. RX_driver echoes back the value of TX_driver as soon as the 8 bits have been completely transferred.

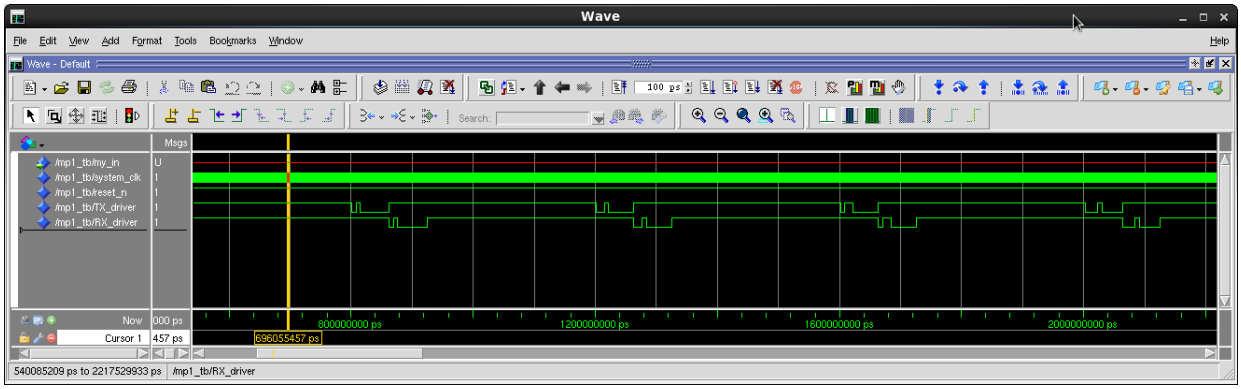

Figure 1: Part 3 - Normal Execution waveform

The upper case conversion is implemented with the following code in the echo module:

```
if (unsigned(data_in) >= 97 and unsigned(data_in) <= 122) then
  data_out_reg <= std_logic_vector(unsinged(data_in) - 32);    --convert to upper c

else
  data_out_reg <= data_in;
end if;
```

We generated the bitfile of this design and were able to see the successful results in minicom. The small-case letters were echoed back as upper-case letters in minicom. The bitfile for this part is included with the submission.

4 Exploring the Design

In this section we were asked to create we were asked to create a timing diagram and schematic for the UART module. First we started by creating the timing diagram which is pictured below.

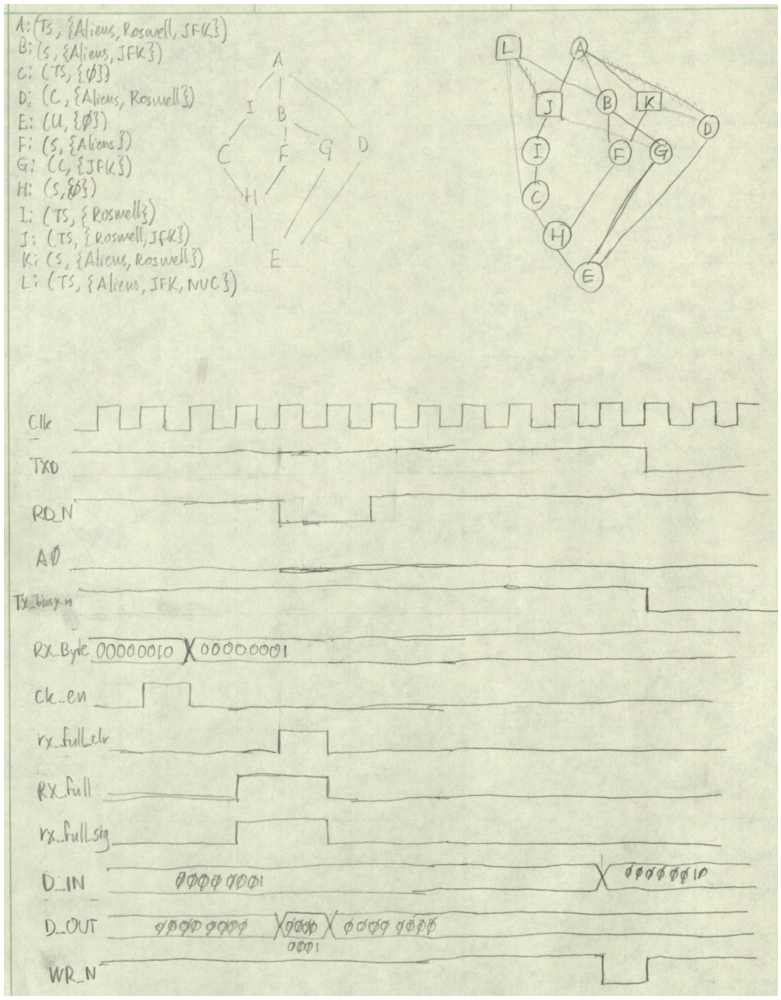

Figure 2: Part 4 Timing Diagram

This module needs to be hooked up to the hardware UART signals in the FPGA. After adding the module to the project, it is simple to use. Simply send data to it over the UART and it will echo the data back to you.

After creating the timing diagram. We moved forward and created the block diagram for the UART module which is pictured below:

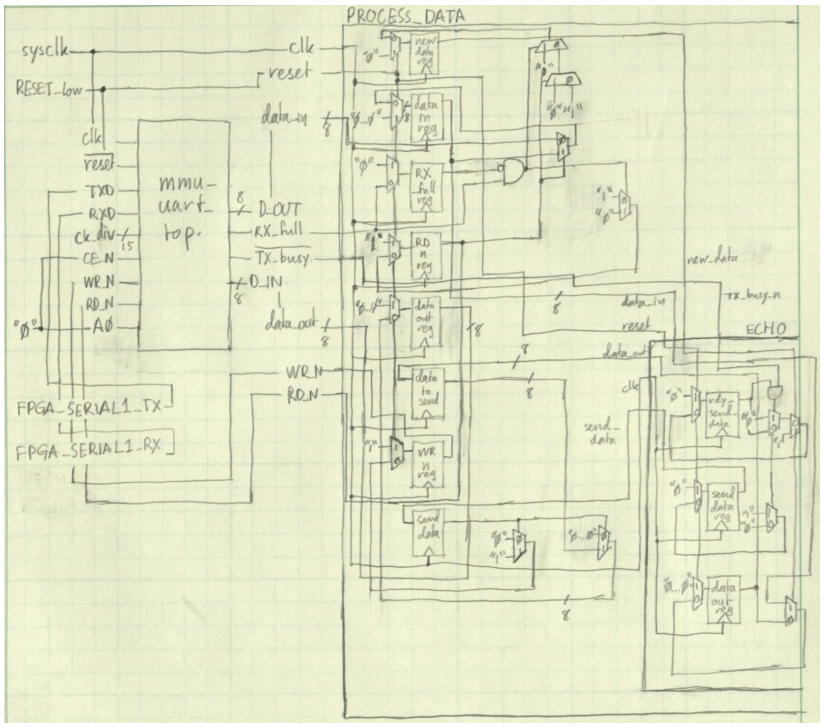

Figure 3: Part 4 Block Diagram

5 Echo Delay

The module works by taking in all signals sent between mmu_uart_top and process_data and holding that information for one second, before sending it out to the respective components.

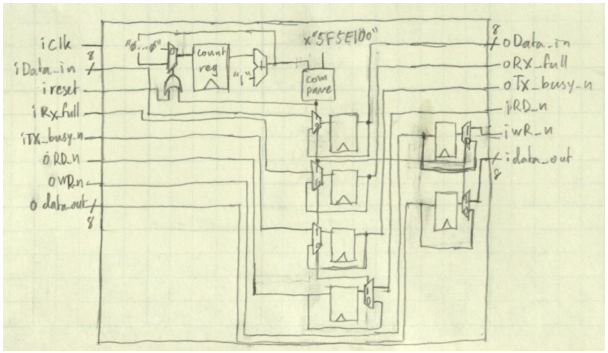

Figure 4: Part 5 Block Diagram

6 In-Flight Calculations

In this section, we modified the design so that it would echo back the sum of the previous two inputs if the last two characters entered we both 0 through 4. To do this we replaced the "echo" module with our own module called "num_adder". Below is a block diagram for the design:

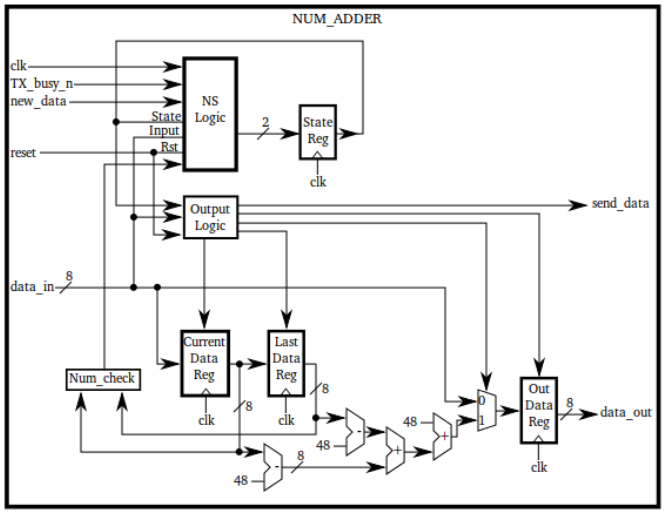

Figure 5: Part 6 - Block Diagram

The design is fairly simple. It primarily consists of a state machine and some supporting logic. The state machine is responsible for detecting new data, storing this data in registers, detecting when the UART is free, and detecting the occurrence of two consecutive numbers between 0-4 and outputting their sum. There is a submodule contained within this design (num_check) who's only job is to see if its two inputs are both 0-4 and return 1 if they are or 0 otherwise. Here is the state diagram for the state machine in "num_adder":

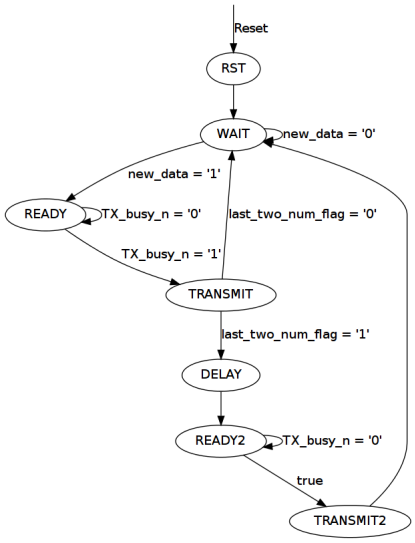

Figure 6: NUM_ADDER State Diagram

The state diagram is fairly straight forward. The module is normally waiting for new data to arrive. After new data arrives it waits for the UART module to be available to transmit the data. If the state machine received the signal from the "num_check" module then it proceeds to wait for the UART module to become available again and then transmits the second byte of data. The only slightly strange state is the DELAY state. This was necessary since the UART module has a one clock cycle delay between receiving new data to transmit and the time when it unsets the TX_busy_n flag to signal that the UART is busy.

7 Changing the Baud Rate

This section asked what changes would be necessary to change the baud rate of the module from 9600 to 38400. The well commented signals made that change fairly easy to find. In the "UART_driver" module you would change the "ck_div" input on the "UART_1" instance to x"011E". This is based on the formula to find the baud rate given below:

$$baud\_rate = F(clk)/(ck\_div * 3)$$

In our design, the base clock rate is 33MHz.

$$38400 = 33MHz/(ck\_div * 3)$$

$$\boldsymbol{ck\_div = 286}$$

8 Conclusion

In this lab we were introduced to the tools we will be using throughout the semester. We were also reintroduced to VHDL coding and some of the best design practices when coding with VHDL.