Laboratory Exercise 1
UART

TCES 330 Digital Systems Design
Spring 2018

Author:
Spencer Sawyer

Requirements:

TX Driver:

For this part of the assignment, it was required that the driver should hold data in a block of memory to be transmitted to the UART at the proper time. It would then generate a new address for when the UART needed more data.

UART:

This part of the assignment required a module that would read in a byte of data and save if to a local register, and then add a stop and start bit before and after transmitting the data from least significant bit to the most significant bit. It would then tell the driver it was ready for more data through a wire.

Design considerations,

TXDriver:

For the driver of this assignment, I could be within the assignment parameters and still have a driver that did not function with a baud limited UART. The issues that I could have had, if I hadn't thought it through, was the if TxEmpty was high for more than one clock cycle consecutively, TXDriver could have changed the data it was outputting. This could have been solved by making sure the UART only ever was asking for data for one clock cycle at a time or, in my case, I made the TXDriver wait for TxEmpty to dip to low before it registered the UART as having gotten the data. Both solutions could be implemented at the same time to ensure that the UART and the TXDriver could work with a diverse set of implementations. The TXDriver could also be implemented with a state machine as well but that was not the approach I took.

In my implementation of the TXDriver, I also defined the data to be sent locally instead of instantiating an external memory module as I had already designed it before learning of that potential expectation. The driver likely wouldn't require much to get to the point where it used a memory module though since I used a pointer to address each data byte.

UART:

There are several ways to implement a UART; the path I used was setting up a state machine, however the methodology I used to implement the state machine lead to quartus not seeing it as such. Other methods could be using variables to track progress of the UART as it transmitted and pulled data from the driver. In fact, if I were to design another UART, I would not try to use a finite state machine since I think it would be less susceptible to bugs/implementation differences if I hadn't.

Testing:

I mostly tested my Driver and UART by stepping through the design as it was running in ModelSim and making sure that it was hitting all the right conditions for each external variable. It was during this phase I noted that I forgot to declare one of my variables as a vector and so my UART was never hitting the send data state. This was easy enough to fix. However, due to testing by stepping through the program with different external variables, I don't have very pretty waves to show off how it works through ModelSim. I do have pretty pictures of it working with the oscilloscope though which I believe is worth several model sims over.

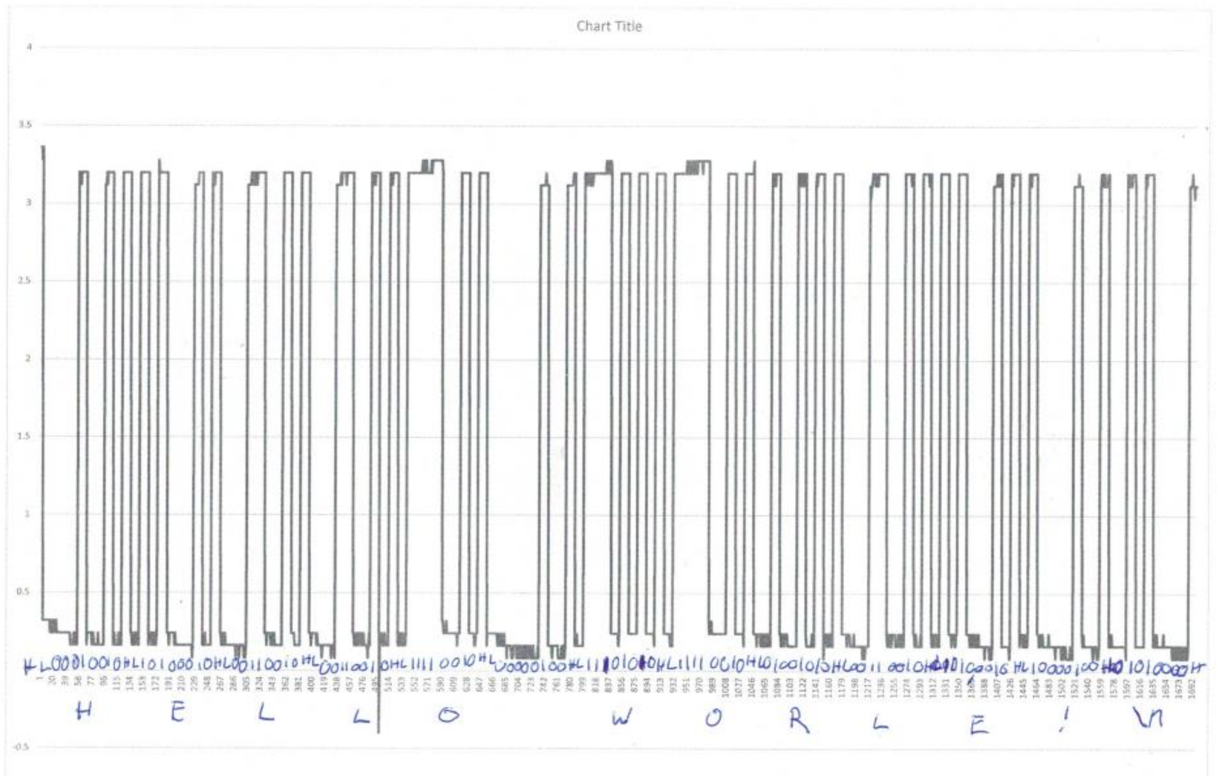

Figure 1;

While both modules worked, there was a difference in how it outputted from what I expected. I didn't define 'D' to be 68 but instead I defined it to be 69 which is 'E'.
Conclusion, this was a far more complex assignment than our previous homework, and looking back on it, I wish I had worked with a partner, so I wouldn't have had as much work to do by myself and I could have written more robust tests for whatever module I had the responsibility for. There are also design choices I would do differently. I would have used some sort of finite state machine for the driver while potentially forgoing it or at least making more states or perhaps putting two state machines together, one for fetching data and one for sending data.