

Project A


Universal Asynchronous Receiver Transmitter (UART)

TCES 330 Digital Systems Design

Spring 2010

Authors:

XXXX XXXX 

XXXX XXXX 

Submission Date: May 25, 2010

Table of Contents

Project A	1
1: Requirements:	2
1.1 sub section	2
1.2 sub section	3
2. Design: (including SystemVerilog code and block diagrams)	3
2.1 sub section	3
2.2 sub section	4
3. Test Procedures: (including SystemVerilog test bench code)	5
3.1 sub section	5
3.2 sub section	5
3.3 sub section	6
4. Test Results: (including waveforms and data monitoring)	6
5. Observations: (including expected or unexpected results as well as analysis)	8
6. Conclusions: (lessons taken, experiences sharing, etc)	9
Reference (if any)	
Appendices (if any)	

Project A

The purpose of the project is to

Our team members are:

We divided the workload as follows:

1. Requirements

The objectives of the TXDriver and the UART are described in this section of the report. Part I outlines the requirements for the TXDriver module, and Part II specifies the requirements for the UART module.

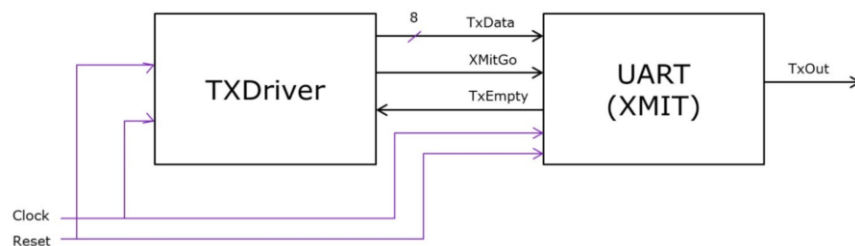


Figure 1. UART Block Diagram

1.1 Part I

The requirements for the TXDriver is to hold the data-to-be-transmitted and send one byte of data to the UART. It should only send in data when the UART is ready to receive data or when TXEmpty is true. XMitGo must be true when the TXDriver is ready to send in data to the transmitter. XMitGo is a variable that tells the TXDriver that new data is ready to be loaded into the UART. After sending the appropriate data to the UART, the TXDriver must pause for one second to send in the data again.

The data that the TXDriver will hold is each character of “Hello World!”. Each character is held in a memory block and will be ready to be sent to the transmitter. The module must also be implemented as a state machine for this laboratory exercise.

1.2 Part II

The purpose of the UART module is to read in data and then transmit the data to a receiver. In order to complete the task above, the UART must read in data from the TXDriver and save it when XMitGo is true. The UART module also adds a start and stop bit to the beginning and end of the sequence of bits during output. UART must output each bit separately, including the added start and stop bit, to the variable output TXOut. The bit rate, or baud rate, used for TXOut is **38,400 bits/sec** or **26.042 μ sec/bit**. For this lab exercise, the UART module must be implemented using a finite state machine.

2. Design

This section of the report will describe the analysis and implementation of the modules in SystemVerilog.

2.1 Part I

The first step to create the module in Quartus called TXDriver.sv. Then the module must hold the data that is to be sent to UART. The code for the module consists of localparams for each character of “Hello World!”. Each byte of the character is represented in ASCII, so each parameter will hold the appropriate hexadecimal value.

Example code from module:

```
localparam
    H = 8'h48,
    e = 8'h65,
    l = 8'h6c,
```

After setting the data, the TXDriver must be implemented as a state machine. For this module, there are 13 states. Twelve of the states represent the hexadecimal value that will be sent out to the UART, and the last state will be the idle state.

Here is an example of a state in the module:

```
case (currentstate)
    A: begin
        nextstate = B;
        TxData = H;
```

Each state will send out the character data if and only if TxEmpty is true. The module will check if UART is true first then send in the next data. When the TXDriver is sending the data to UART, it will send XMitGo as true to the UART so the UART can read in the data.

There is an internal counter for the module to count through the states so that when all the data is sent out, the TXDriver will go into an idle state for 1 second. During this one second pause, it will send XMitGo as false to UART so it will not read in any data coming in from the driver. After the second is over, it will go back to the first state and see if TxEmpty is true.

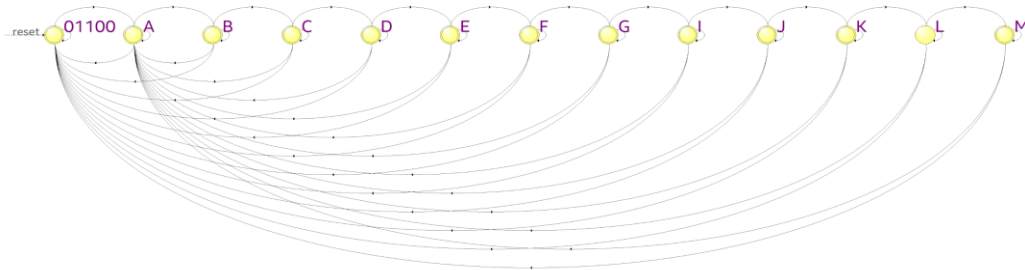


Figure 2. TXDriver State Machine

2.2 Part II

The UART module must first be created as a Quartus UART.sv file to be implemented. The module will store the input data from the TXDriver into the variable TxData, which is an 8 bit wide vector. The output variable TxEmpty will be set to zero to tell the TXDriver that it is not ready to receive new data at this time. A state machine is created for this module using 11 different states, 8 states output the 8 wide vector, containing 8 bits of data, from least significant bit to most significant bit. The last three states will be an idle state, or high state where the output remains high until XMitGo is true, a start bit before the least significant bit set to zero, and a stop bit coming after the most significant bit set to one.

This module also has an internal bit counter, called count, to check if all 10 bits have been output, along with a baudrate counter, called baudcount, that will help keep the timing in sync with the TXDriver. Once count has reached 10, the count will be reset and the state machine will begin from either idle or the startBit state depending on XMitGo, using new data brought in from TXDriver.

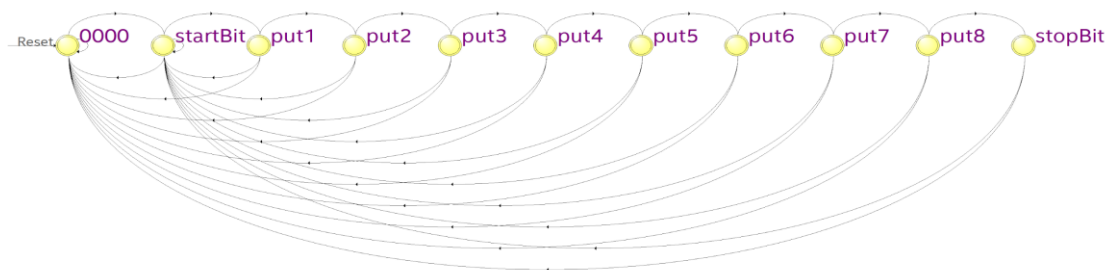


Figure 3. UART State Machine

3. Test Procedures

This section will describe how both the TXDriver and UART modules were tested in ModelSim. Part I will describe the testing of the TXDriver, Part II describes the test of UART, and Part III describes the testing of both modules connected.

3.1 Part I

To test the module in ModelSim, a clock was implemented, and only the TxEmpty and Reset were changed to observe the behavior of the module. The data that is sending must only change when TxEmpty is true, and if it is false, the state must hold.

Here is a picture of the tests in ModelSim:

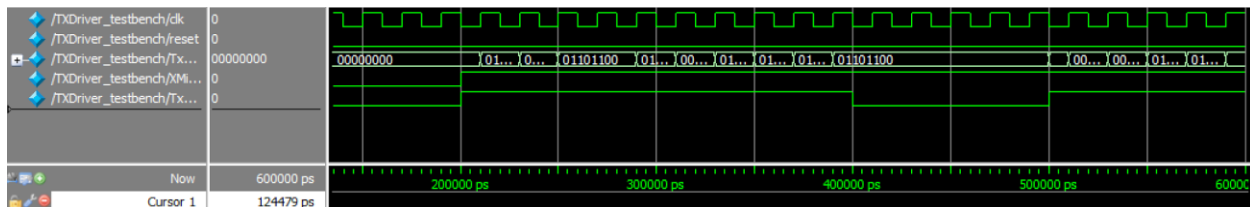


Figure 4. TXDriver ModelSim Simulation without Baud Rate

You can see that when TxEmpty is false, it still holds the previous data until TxEmpty is true again.

3.2 Part II

The module UART was tested using ModelSim with different random 8bit inputs. The ModelSim simulation was shown with the Clock input, Reset input, TxData input, TxOut, and XMitGo. The TxOut data shows all the input bits plus the start and stop bits for each input data byte, unless reset is set to 1. Baudrate was removed just for testing purposes on ModelSim, it was implemented for testing on the oscilloscope.

Here is a sample from the ModelSim test:

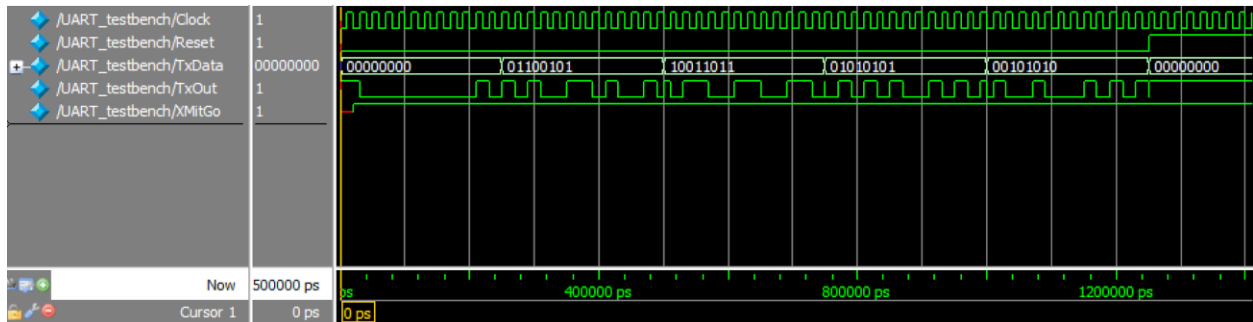


Figure 5. UART ModelSim Simulation without Baud Rate

From this ModelSim simulation, it is shown that the reset causes the UART state to switch to idle. The start bit and stop bits are also visible from this simulation with the reverse order of the eight bits between.

3.3 Part III

Another module was made to test the timing of both the TXDriver and the UART modules. This module is called TestProject, and it is located in the TestProject folder. For testing purposes, this test module did not implement the baud rate. This is to observe if TxOut is outputting the right bits into the receiver based on the data coming in from the TXDriver.

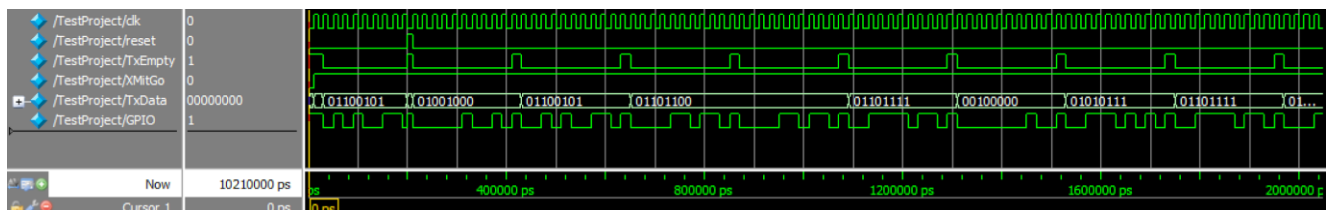


Figure 6. TXDriver with UART ModelSim Simulation without Baud Rate

You can see that TxOut is behaving properly and it is outputting the corresponding data starting with the least significant bit.

4. Test Results

To physically implement the modules, a high-level module must be created for the FPGA. The modules will use the FPGA's built in clock, CLOCK_50, KEY[0] to represent the reset button, and GPIO[1] for the receiver to get in the data coming from TxOut. This high-level module is called LabA.sv.

Once the files were compiled, an oscilloscope was used to measure the output coming from GPIO[1]. A trigger setting was used to capture the exact moment the whole message was sent. The picture below displays the output from the oscilloscope.

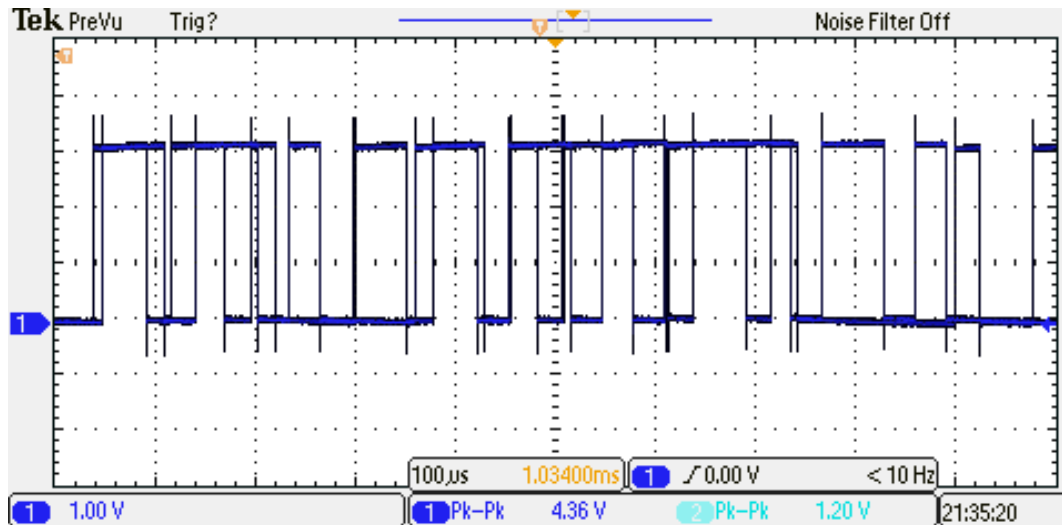


Figure 7. Output “Hello World!” From Oscilloscope

The picture that was saved from the oscilloscope is a little choppy but on the actual oscilloscope, the message is clearer. After the message was sent, the modules went into a one second pause. The test results of the lab were a success.

5. Observations

Both modules were not difficult to implement, but the hardest part for the transmitter to work is getting the timing right for both modules. The TXDriver and the UART must work together in sync to get the message into the receiver. There were a lot of timing problems initially but adding more conditions and tweaking the conditions helped with creating the transmitter. Once we got the right conditions for the modules to work together, the whole string of message was received.

Initially in the oscilloscope, the lines were not straight and had a lot of noise. Each bit did not look like a square wave. We then realized that we forgot to include the baud rate into the UART module. Once we fixed the baud rate for the UART module, the lines started to look better and cleaner.

6. Conclusion

This lab experiment taught us the importance of cooperation and timing for the modules. Getting through the timing issue for both the driver and the transmitter was a difficult task but our experiment turned out as a success.