ECE 351- Spring 2021
Homework #4

Submit your deliverables to your Homework #4 dropbox by 10:00 PM on Wednesday 02-June-2021. THERE Will BE NO LATE ASSIGNMENTS ACCEPTED AFTER NOON ON THURSDAY, 03-JUNE

Source code should be structured and commented with meaningful variables. Include a header at the top of each file listing the author and a description. You may use the following template:

```
//////////////////////////////////////////////////////////////
// <filename>.sv - <one line description>
//
// Author:  <your name> (<your email address>)
// Date:    <date you created the code>
//
// Description:
// ------------
// <text description of what function the module performs>
////////////////////////////////////////////////////////////////
```

Files to submit:

Problem 1:
- Source code for uart_tx.sv (your implementation of the UART Transmitter)
- Source code for any starter code files that you changed
- A transcript showing that your UART loopback test worked correctly. The transcript should include the information identifying that the simulation results are from your code as in HW #3 and should be based on the loopback testbench provide in the release.

Problem 2:
- Source code for your carwash FSM even if it has not changed from HW #3
- Source code for your testbench. Should make use of an implicit FSM to exercise the FSM
- A transcript showing that your carwash FSM worked and testbench worked correctly. The results should be based on your testbench and include the information identifying that the simulation results are from your code as in HW #3.

Extra credit:
- Source code for your shift register implementation even if it has not changed from HW #3
- Source code for your testbench. Should be a self-checking testbench.
- A transcript showing that your shift register and testbench work correctly. The results should be based on your testbench and should include the information identifying that the simulation results are from your code as in HW #3.

Note: the name of the SystemVerilog source code files should match the name of the module. The file should have a .sv extension for a SystemVerilog source code file.

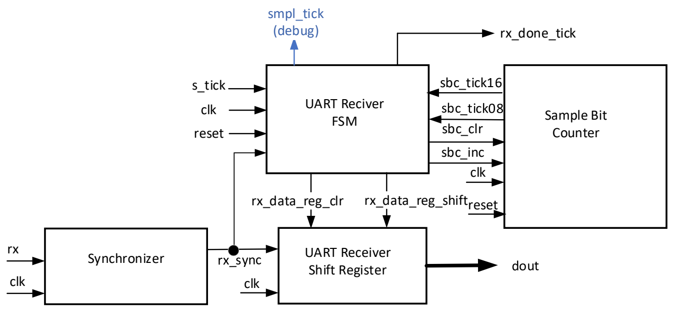
Create a single .zip or .rar file containing your source code files and your report. Name the file `<yourname>_hw4.zip` (ex: rkravitz_hw4.zip).

Problem 1 – UART (Serial) transmitter (60 pts)

In this problem you are going to write a SystemVerilog model that implements UART serial transmitter RTL module and simulate the receiver/transmitter pair using the loopback testbench provided in the release. You have been provided with code that implements the UART serial receiver (`uart_rx.sv`), baud rate generator (`baud_gen.sv`), and the loopback testbench (`tb_uart_loopback.sv`). The loopback testbench connects the UART transmitter that you write with the UART receiver that I wrote by connecting the Tx signal from the transmitter to the Rx signal on the receiver.

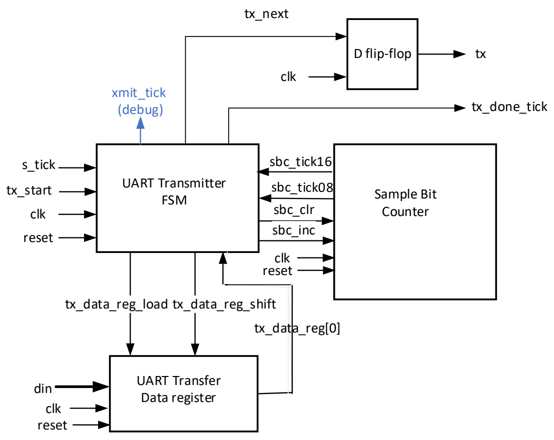
There are several ways (and I am sure you can find complete code on the internet…but don't) to implement UART transmitters and receivers. My receiver module is based on an FSMD (FSM + datapath) with control handled in the FSM and a separate sample bit counter and receiver shift register.

I strongly suggest that you model your serial transmitter on the serial receiver code. The functionality is pretty much the same. Pay attention to when your transmitter model should drive the start bit, data bits, and stop bit vs. where the receiver samples them, though. Also note that the transmitter starts with parallel data and serializes it while the receiver takes in serial data one bit a time and parallelizes it. Block diagrams of the UART serial receiver and serial transmitter are provided below:



UART Receiver

Note: A synchronizer is used to bring asynchronous signals (`rx` in this case) into the clock domain of a sequential circuit. A 2 flip-flop synchronizer greatly decreases the likelihood of metastability. More on this in ECE 540 and ECE 581.



UART Transmitter

Deliverables

- Source code for your UART transmitter module and any SystemVerilog starter files that you changed
- A transcript showing that your UART loopback test worked correctly.

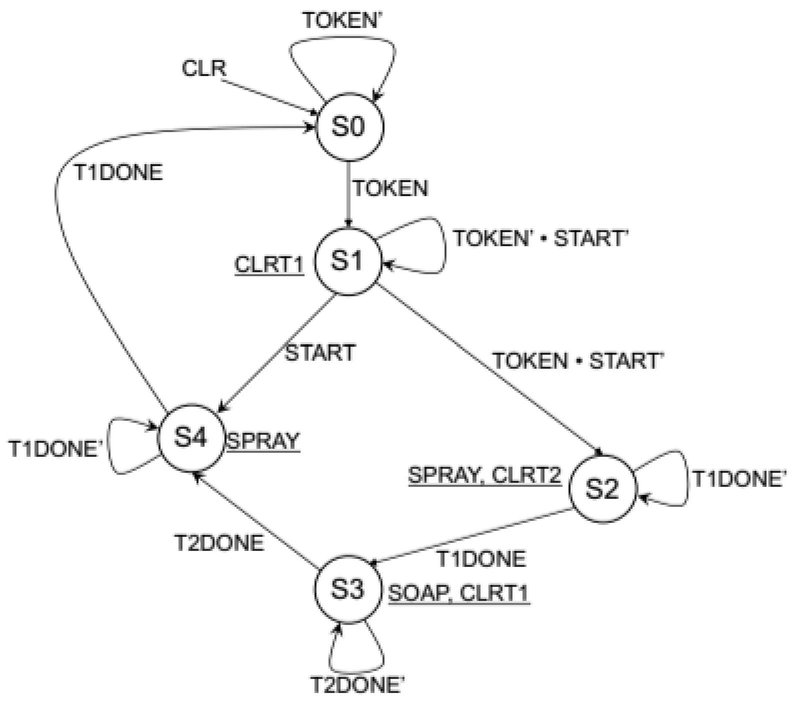
Grading

You will be graded problem as shown:

- Correct implementation of the UART transmitter. [35 pts]
- Top level simulation results. [20 pts]
- The quality of your design expressed in your SystemVerilog source code. Your code should be cleanly structured and commented. [5 pts]

Problem 2 – Testbenches (40 pts)

In this problem you are going to write a testbench for the carwash FSM controller you implemented in HW #3. You will test the FSM using an implicit FSM like the one I discussed in class (source code for the example is included in the release). The state transition diagram for the carwash controller FSM is shown below:



Your implicit FSM should traverse all the states and arcs by generating the inputs needed to move from state to state along each transition. My solution followed two paths to visit each state and each transition, but you may be more (or less) innovative in your approach. Incidentally, a question came up in class about what to do about the inputs that do not affect a state transition. They are don't cares. One way to check if the inputs are really don't cares for your carwash FSM implementation is to set the (supposedly) don't care inputs to x. You may use a `$monitor()` to display all your inputs and outputs and manually check your results. While it may be possible to implement a self-checking testbench, the FSM is simple enough that I didn't feel that it was worth the additional effort. You may feel differently, and that's just fine. The more you take upon yourself to learn the happier I am.

Deliverables

- Source code for your carwash controller FSM
- Source code for your testbench
- A transcript showing that your carwash controller FSM and testbench work correctly. You may want to add comments to your results to clarify what we see when we're grading your solution.

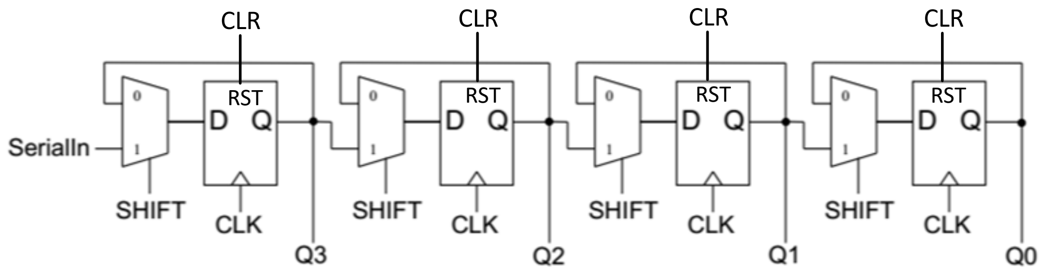
Grading

You will be graded problem as shown:

- Correct implementation of your carwash controller test bench. [25 pts]
- Top level simulation results. [10 pts]
- The quality of your design expressed in your SystemVerilog source code. Your code should be cleanly structured and commented. [5 pts]

Extra credit – Shift register testbench (up to 7 pts)

Write a testbench for the shift register you implemented for HW #3 problem 1. This testbench can make use of any of the stimulus vector generation techniques discussed in class ("on-the-fly", in an array, in an external file and should be a self-checking testbench. The schematic for the shift register is shown below:



Deliverables

- Source code for your shift register implementation from HW $3
- Source code for your testbench
- A transcript showing that your shift register and testbench work correctly. You may want to add comments to your results to clarify what we see when we're grading your solution.

<finis>