EECS 470 Lab 3 Assignment

Note:

- The lab should be completed individually
- The lab check off is due by **Friday, September 19th**

1 Introduction

The focus of this lab is FSMs (finite state machines) and their implementation in SystemVerilog. The first part of this lab has you fix a buggy FSM implementation, the second part has you build an FSM from scratch to implement the UART communication protocol. The skills you learn in this lab will be useful for finishing your project 2 implementations, and your final project.

2 Find Them Bugs

The purpose of this exercise is to familiarize you with common bugs (both behavioral and on sythnesis) that appear in SystemVerilog designs. Fix the errors in this FSM (`buggy.sv`) to operate as shown in Fig. 1.

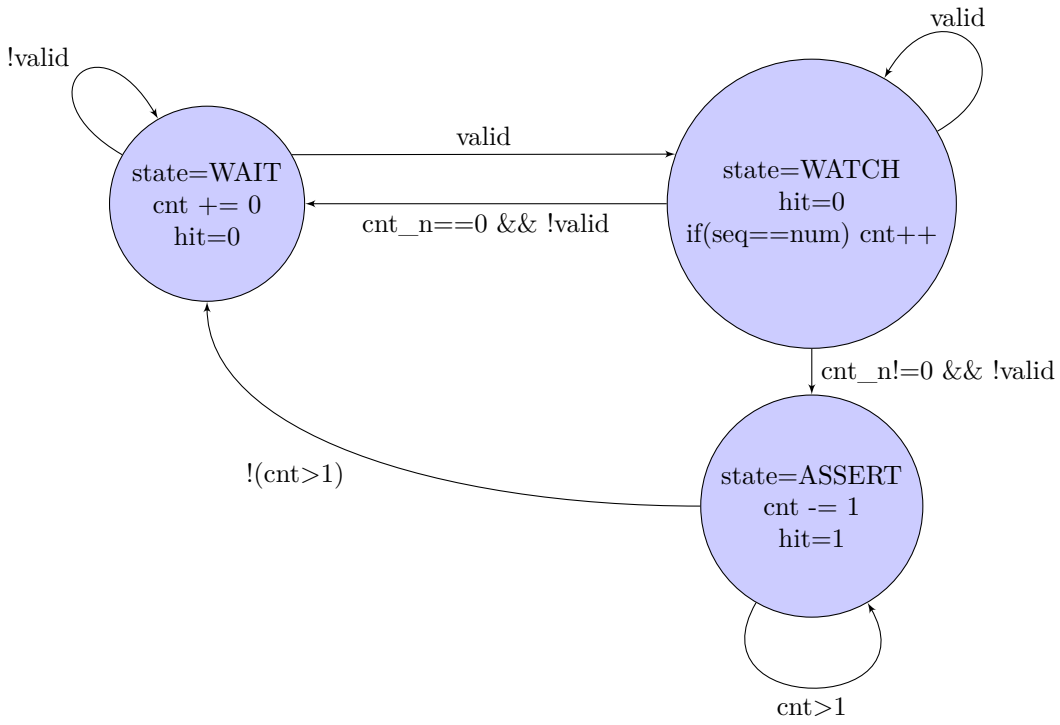

Figure 1: The `reset` signal should send the state machine back to state=WAIT and set `cnt` to zero.

To verify whether you've found all the bugs, run the given testbench using `make sim`.

After the FSM passes in simulation, you want to ensure it still passes after synthesis. Synthesize the FSM using `make syn`. After synthesis completes successfully, ensure that you did not synthesize any latches by inspecting the synthesis output files.

Use the following command to check for latches: `cat synth/buggy-synth.out | grep "Latch"`. There should be no results from the `grep` command.

3 FSM Design

In this part, you will design an FSM from scratch that implements the UART protocol.

3.1 UART Description

Universal Asynchronous Receiver-Transmitter (UART) is a communication protocol used to enable serial communication between devices. UART is commonly used today for embedded systems applications, and microcontrollers will often have one or more UART ports available.

UART messages are transmitted as "frames," which are serial sequences of bits with a defined length and specific delimiters. Many frame configurations are possible. In this problem, you will consider a UART frame with the following sequence of bits:

- **One start bit** - logic low, indicates start of frame.
- **Eight data bits** - the data to be transmitted, least-significant bit first.
- **One parity bit** - an error checking mechanism, logic high if the number of data bits which are high is odd (referred to as even parity, since the additional high parity bit creates an even number of 1s in the frame). One way the parity bit can be found is by XOR-ing all of the data bits together.
- **One stop bit** - logic high, indicates the end of the frame.

When no frame is being transmitted, the serial out line is held high.

3.2 Example

The following is an example of a UART frame transmitting the number 75 (binary 01001011). One bit it sent per clock cycle. Note that there is an even number of high bits in the number 75, therefore, the parity bit is low.

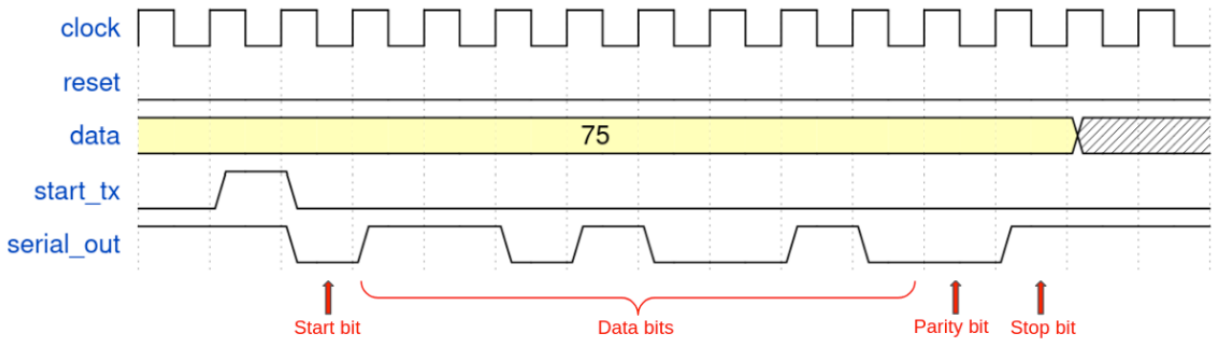

Figure 2: Example UART waveform transmitting the number 75

3.3 Implementation

In the `uart_tx.sv` file, **implement the given module**. We have given you an enum of states for your implementation. You can assume the following:

- The input `data` signal is registered outside of the module and is valid throughout the UART frame.
- When reset is asserted, the module should transition into the `IDLE` state. The reset must be synchronous.

We have given you a very simple testbench that drives your module over a single test input. You can modify the localparam to change the input data value. For the lab checkoff, you will need to show the instructor a waveform (either from Verdi or VCD viewer) of the module transmitting the number 174 (binary 10101110).

3.4 Optional: Using VCD Viewer VsCode Extension

In lab 2, we introduced the Verdi debugging tool. Verdi is a powerful tool, but sometimes it can be overkill and a bit slow to quickly examine a waveform. A quicker way to view a waveform is to generate a VCD (Value Change Dump) file and use a VCD viewer to parse this ASCII file into a graphical waveform.

3.4.1 Generating a VCD File

VCD files can be generated simply by adding the following to the start of the `initial` block in your testbench:

- Specify the name of the VCD dumpfile to be generated. This is done using with the following: `$dumpfile(<filename>)`.
- Specify what values should be tracked with `$dumpvars(levels, list of modules or variables)`. The levels argument specifies the number of hierarchy levels below the specified module instance to include in the dump file.

Uncomment the appropriate lines in the given testbench to generate the VCD file. Once you do so, re-run `make sim`. After the simulation completes successfully, you will find the VCD file inside the `uart/` directory.

3.4.2 VsCode Extension

VsCode has many extensions to convert the ASCII VCD files into a graphical waveform. To browse these extensions, go to the extensions tab in the left-hand ribbon and search VCD viewer. Many options will show up, and you are free to experiment with the various extensions. A really simple one is VaporView.

After you install the VaporView extension, a new tab with a waveform icon should appear in the left-hand ribbon. Open the VCD file to launch the waveform viewer. Then, click on the waveforms icon in the ribbon to launch the VaporView configuration settings.

In the VaporView configuration, use the drop-down menu to navigate to the signals of interest, and use the checkbox next to the signals to add them to the waveform viewer. You can right-click on a signal in the waveform to change its radix. After adding the appropriate signals, your waveform should look like the following:

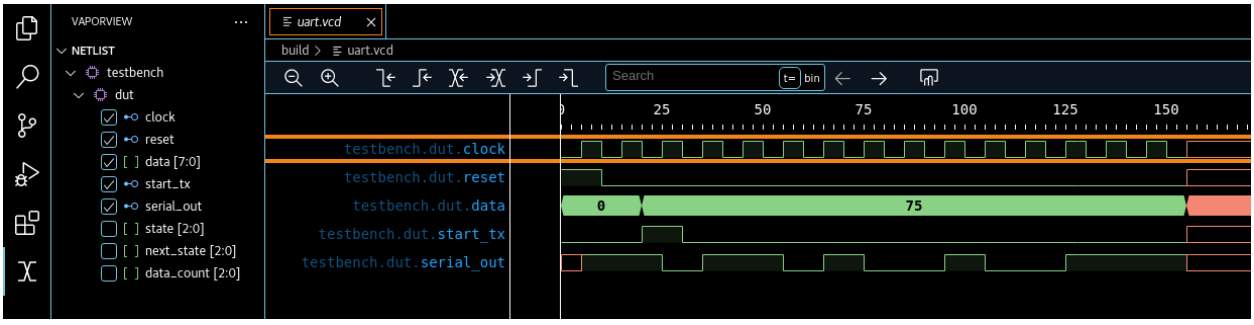

Figure 3: Example UART VCD waveform for transmitting the number 75

4 Submission

You will need to show an instructor the following:

- The buggy FSM successfully passing the given testbench on synthesis, and proof that no latches were synthesized.
- A Verdi or VCD viewer waveform of your `uart_tx` successfully transmitting the number 174.

Place yourself on the help queue during lab or office hours once you're confident you've completed the lab satisfactorily. Turn in your check-off to Gradescope by the end of the day of next week's lab.