**ECE 6213 – Design of VLSI Circuits**

**Fall 2022 – The George Washington University – Dr. Jerry Wu**

**Osama Yousuf – HW 5**

**Problem 3 – Finite State Machines**

**Mealy State Machine**

Following is a list of key points for a Mealy state machine.

* The output is a function of the current state as well as the input
* The input can directly cause a change in the output, which might violate time requirements.
* Compared to a Moore state machine, a Mealy state machine often has less total number of states.

**Code:** *mealy.v*

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**Testbench:** *mealy\_tb.v*

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Testbench Features**

* My testbench is self-reporting
* It starts with a reset to the *WhiteState*, and checks for the validity of every state transition sequentially.
* I implemented events for:
  + Resetting the state machine
  + Updating testbench cycles
  + Watchdog for exiting the simulation
  + Looping over the various states of the state machine and checking for validity
* I implemented logic for counting errors in every test case, as well as for counting total errors in the simulation
* I also implemented a simple task for printing the status of the errors encountered for self-reporting

The final text output of my testbench is given below; all test cases pass without any errors:

A picture containing text

Description automatically generated

Waveform snippets for a few cases from my testbench are also displayed below:

Graphical user interface

Description automatically generated

* Firstly, the last 4 signals in my waveform indicate the state encodings for each of the four states. These are:
  + WhiteState *white\_state* 00
  + RedState *red\_state* 01
  + GreenState *green\_state* 10
  + BlueState *blue\_state* 11
* It can be seen that the asynchronous reset at the start means that both current and next states are white.
* When the red input is asserted, the next state becomes red, and the output of the FSM (*mealy\_out*) is asserted, both at the exact same time. This is correct expected behavior since we have a mealy FSM; the input causes a direct change with the output.
* When the red input is made 0 again, the FSM correctly cycles back to the white state.
* A similar thing happens for other transitions, namely from white state to green state (and back), and white state to blue state (and back).

**Moore State Machine**

Following is a list of key points for a Moore state machine.

* The output is a function of only the current state.
* As a result, the output changes only after a state transition has occurred, contrary to a Mealy state machine.
* Compared to a Mealy state machine, a Moore state machine often has higher total number of states.

**Code:** *moore.v*

Text

Description automatically generated

Text

Description automatically generated

**Testbench:** *moore\_tb.v*

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

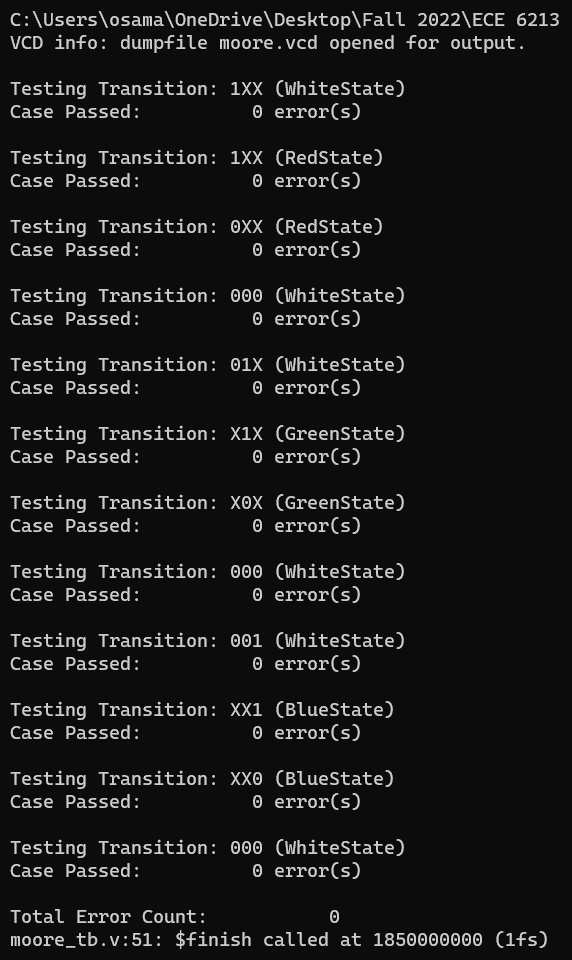
Text

Description automatically generated

**Testbench Features**

* My testbench is self-reporting
* It starts with a reset to the *WhiteState*, and checks for the validity of every state transition sequentially.
* I implemented events for:
  + Resetting the state machine
  + Updating testbench cycles
  + Watchdog for exiting the simulation
  + Looping over the various states of the state machine and checking for validity
* I implemented logic for counting errors in every test case, as well as for counting total errors in the simulation
* I also implemented a simple task for printing the status of the errors encountered for self-reporting

The final text output of my testbench is given below; all test cases pass without any errors:



Waveform snippets for a few cases from my testbench are also displayed below:

Graphical user interface

Description automatically generated

* Firstly, the last 4 signals in my waveform indicate the state encodings for each of the four states. These are:
  + WhiteState *white\_state* 00
  + RedState *red\_state* 01
  + GreenState *green\_state* 10
  + BlueState *blue\_state* 11
* It can be seen that the asynchronous reset at the start means that both current and next states are white.
* When the red input is asserted, the next state becomes red, but the output of the FSM (*moore\_out*) is not asserted at the same clock. This is correct expected behavior since we have a Moore FSM; the input does not cause a direct change to the output. The output is asserted in the next clock cycle.
* When the red input is made 0, the FSM correctly cycles back to the white state.
* A similar thing happens for other transitions, namely from white state to green state (and back), and white state to blue state (and back).

Note: Instead of $fopen and $fclose, I used the “>” operator to directly re-direct terminal output to a text files for both of my FSMs (Mealy and Moore), as follows:

**For Mealy output:**

* iverilog -o mealy mealy.v mealy\_tb.v
* vvp mealy > mealy\_output.txt

**For Moore output:**

* iverilog -o moore moore.v moore\_tb.v
* vvp moore > moore\_output.txt

The end result are identical text files that contain all of the $display information from the two test benches, as shown in the snippets below:

Graphical user interface, text

Description automatically generated

The waveform files are also provided in the fsm folder (in .vcd format).