Lab 1: Synchronous State Machines and the DE1-SoC

Concepts and Background

The use of memory elements in a digital logic circuit is the differentiating characteristic between sequential and combinational networks. The outputs of combinational circuits only depend on the present input values and the delay paths for the propagation of transitions through the circuit. The addition of feedback and storage in the network causes the circuit's output to depend on the previous inputs as well as the current inputs. The information stored in the circuit about the previous input values is known as the state of the circuit. Generally, the state machine will follow either the Moore machine or the Mealy machine models where the memory is separated from the computational circuitry. In this course, we will only consider synchronous systems: ones that use edge-trigged flip-flops as memory storage elements.

The Altera Quartus II compiler can detect when HDL code defines a state machine using a certain pattern and synthesize the machine for implementation in the FPGA. During synthesis, the compiler automatically calculates the state assignments and minimizes the state table, if possible. You will be using this feature to construct the synchronous state machines for this lab. Altera provides a reference Verilog state machine that lacks comments, but is simple enough to be easily readable http://www.altera.com/support/examples/verilog/ver_statem.html.

Implementation Requirements

This lab is about the use of state machines to design a turn signal/brake light control system, much like in EECS 281. However, this course will use the HDL state machine syntax rather than manual state assignment and transition equation development.

The design will consist of two independent machines: one maintains the turn signal state and the other maintains the brake light state. All circuits will be trigged from a slowed clock at ~3Hz using division by powers of two from the 50MHz system clock. For the input signals, switch 0 is the right-turn select, switch 1 is the left-turn select, and button 2 is the brake activation. The primary outputs are the three leftmost LEDs for the left turn signal, the three rightmost ones for the right turn signal, and the two middle ones for the brake lights. The remaining two green LEDs must remain dark. Segment 3 of display 3 (HEX0) is the state machine clock. Connect this clock indicator directly to the signal being used as the slow clock. The error state indicator is the third display (HEX2) and it should display E (segments 0, 3, 4, 5, 6) when the error is asserted. The remaining 7-segment displays must remain dark.

All modules should be written with active-high inputs and outputs. This will require the use of inverters to correctly utilize any active-low I/O provided by the DE1-SoC. Numbers in parentheses following the example module I/O names are the width of the signal vectors.

Turn signal machine (Moore Type)

Inputs: clock, left, right
Outputs: l_signal (3), r_signal (3), error

Specifications

- Initialize to idle loop state
- When one direction input is asserted, enter the signaling loop for that direction, starting at the 0-on state

- When both left and right are asserted, enter an error state with the error output asserted and both the left and right signal vectors are 000
- Left signal follows the progression 000, 001, 011, 111
- Right signal follows the progression 000, 100, 110, 111
- In the loop of one signal, remain in the 0-on state for 1 clock cycle, and remain in the 1-on, 2-on, and 3-on states for 3 clock cycles
- For simplicity, only check for error condition or de-assertion of the direction signal during the 0-on state and in the third of the 1-on, 2-on, and 3-on split states
- A direction loop always exits to either the idle state or the error state
- Remain in the error state so long as both direction inputs are asserted
- Exit the error state only to the idle state

Brake light machine (Mealy Type)
Inputs: clock, brake, l_signal (3), r_signal (3)
Internal: brake_active
Outputs: l_lights (3), c_lights (2), r_lights (3)

Specifications

- Initialize to idle loop state
- When the brake input is asserted, assert brake_active
- Assert brake_active for 2 clock cycles after the brake input is de-asserted
- If the brake input is re-asserted during the timeout wait, brake_active must remain asserted
- When brake_active is asserted, invert l_signal and r_signal to produce l_lights and r_lights, and assert both bits of c_lights

Demonstration Goals

- Correct I/O assignments
- Correct turn signal state progression
- Correct brake light operation

Code Organization Requirements

- Two state machine modules
- Top-level module containing only wires, buffers/inverters, and module instances
- Correctly formatted module headers and organized code
- Project submitted on BlackBoard, meeting the submission policy requirements

Design Organization Recommendations/Hints

- Start by determining how to make connections to the top-level interface pins
- Create files for all additional modules and write the port definitions for each
- Decide which machine to implement first and verify that the machine meets the specifications

Design Questions

- How many states did you use for each machine? How did you choose what these states were? Consider using sub-states to achieve the multi-cycle delay before transitioning out of a state (e.g. left_signal_three_lights_0, left_signal_three_lights_1, left_signal_three_lights_2).
- How many bits did you use in the counter to generate the ~3Hz clock?
- Which I/O signals that you need on the DE0 board are active-low?
- How did you verify that you correctly implemented the Verilog state machine syntax in the Quartus software?

Report Requirements

- Complete answers to the above design questions.
- Include the state diagram for each of the machines, *omitting* the transition conditions and just **drawing the directed edges** in the graph. The state names should describe the states in a human-readable way.
- Include a block diagram of the connections between the modules and their connections to external signals through the top-level.
- Include a screenshot of how the Quartus software did the state assignments for each state machine implemented.
- Give a brief summary of how the group members collaborated to complete this lab.

Grading Distribution

- Demonstration 30%
- Code Organization 10%
- Report Diagrams 20%
- Report Answers and Discussion 40%