Lab 6: Sequential Circuits

TA: Andrew White
Oct 6, Oct 8, Oct 13

1 Overview

This lab will continue the development of sequential logic circuits.

2 Finite State Machines

The most difficult part in designing hardware is to correctly implement the control logic of a model. Data-transforming and data-storing units (alu, multiplier, shift register, register file, ...) have to be supplied with control signals at the right time, so the whole implementation behaves as expected. Finite State Machines (**FSM**) provide an easy-to-use model for defining, testing and implementing complex control circuits. An FSM consists of a sequential part storing the current state of the machine in registers, and a combinational part for evaluation of the output signals and the next state from the input signals and the current state. The machine can change its state on every active clock edge.

2.1 Machine Models

There are several models for FSM design.

Mealy-FSM
In a Mealy machine, the outputs are a function of the current state as well as the inputs. The asynchronous nature of the outputs of the Mealy machine can make the behavior unpredictable.

Moore-FSM
In a Moore machine, the outputs are only a function of the current state.

Simple Moore-FSM
In a Simple Moore machine, the state encoding is the output of the machine. This implementation is the fastest because there is no combinational translation of the current state into an output.

2.2 State Encoding

Binary
In Binary state encoding, states are numbered starting from 0 and up. States are assigned in a binary sequence. The advantages to binary state encoding are that it requires a lesser number of flip-flops and uses less area. The disadvantages are it requires more power, prone to glitches, complex decoding logic to translate the current state to next state and output, and it is also possible to get in to a *stale* state (unused and undefined state).

Gray Code
Gray encoding guarantees that only one state variable switches between two consecutive states. It is appropriate for controllers exhibiting long paths without branching. The advantages to using gray code encoding are is uses the same number of flip-flops as binary encoding, lesser chance of getting into stale states, reduces gliteches and hazards, and since only one flip-flop changes for each adjacent state it reduces power usage. The disadvantage is that decoding logic is more complex.

One-Hot
One-hot encoding associates one code bit and also one flip-flop to each state. At a given clock cycle during operation, one and only one state variable is asserted. Only two state variables toggle during a transition between two states. One-hot encoding is very appropriate with most FPGA targets where a large number of flip-flops are available. The advantages to using one-hot encoding are very simple decoding logic and it uses less power. The disadvantage is that it uses more flip-flops.

Enumerated Types
In VHDL, it is common to use enumerated types to represent states in a state machine. The states will implicitly use binary encoding. This will be explored further in following labs

2.3 Modeling State Machines in VHDL

In VHDL, the use of the process statement is the most suitable way to describe an FSM. The VHDL implementation may use several processes (1, 2, or 3) depending upon how the components of the FSM are considered. The 3 primary components of an FSM are the Next-State Function, State (and optionally Output) Registers, and the Output Function. In this description, processes that describe the behavior of combinational logic elements will be referred to as *combinational processes* and those that describe state or storage elements as *sequential processes*. The current state and the next state values are represented using signals. For each state, two things need to be determined: next state and output.

Next-State
Next state equations can be described directly in the sequential process or in a distinct combinational process. The simplest template, which should be used in this lab, is based on a Case statement. If using a separate process, its sensitivity list should contain the state signal and all FSM inputs. The next state signal is assigned a state value based on the current state and optionally the inputs (as with Mealy machines).

Output
Non-registered outputs are described either in the combinational process or in concurrent assignments (for more information on concurrent assignment method, see *Concurrent Selected Signal Assignment Statements*, http://www.csee.umbc.edu/help/VHDL/concurrent.html). Registered outputs must be assigned within the sequential process. Just as with the Next-State function, the output can be simply described using case statements to assign the appropriate output based upon the current state and (for Mealy Machines) the FSM inputs. If using a separate process, its sensitivity list should contain the current state signal and all FSM inputs.

State/Output Registers
The state register assigns the next state signal's value to the current state signal. The format for this register will follow the one for the last lab. State registers must be initialized with an asynchronous or synchronous (reset) signal. An example would follow the last lab: upon asserting the reset, the current state would be reset to some initial state by assigning to it the value of the initial state (initializing the FSM state register).

3 Lab

3.1 Procedure

For this lab, you are to create two Finite State Machines. The first will be a Simple Moore machine that uses gray code encoding and implements a **3-bit gray code up-counter**. This FSM should use two processes: one for next-state/output function, and one for the state register. The output can either be registered in the sequential process of the state register or unregistered and assigned in the (case statement of the) next-state/output combinational process. The second will be a Moore machine **3-bit binary up-counter** that also uses gray code state encoding. This FSM should be implemented using two or three process: one sequential process for the state register, and either seperate or a combined combinational process for the next-state/output functions. For this, the output should not be registered. Both of these state machines will have eight states. Use an **asynchronous reset** and use a **count enable** to enable the counter to change count values at the rising edge of the clock. For testing, use of force statements to generate the clock, reset, and count signals is acceptable. At the beginning of the simulation, generate a reset to initialize the FSM. Test the count enable by lowering it during the simulation to show the FSM stops counting.

3.2 Report

The lab report will consist of a cover sheet, your VHDL code for both machines, and the simulation waveforms for both.