CPEEEE64

DECEMBER 24, 2013

Lab Five: State Machines

Authors:
Benjamin SMITH

Instructor:
Dennis DHALQUIST

Abstract

Two input logic gates are synthesized for the Altera Cyclone IV FPGA using the Quartus IDE. The logic gates are verified using a System Verilog testbench and Mentor's Modelsim HDL simulator

CONTENTS

LIST OF FIGURES

I. Introduction

In this lab state machines will be explored. We will walk through a number of examples from Altera to investigate the different types of state machines available to the digitial designer and why you might choose a particular one.

1) The Case Statement: The "Enumeration" of case statements is the first unique feature of System Verilog that we will use in these labs. before you could have named your files .v or .sv and it would not have mattered. Now for the project to compile, it must be .sv.

A. Implement state machine using state table and K-Map

B. The Difference between Meely and Moore State Machines

II. The Case Statement

Verilog makes use of the case statement like most other progamming languages. The case statement provides a clear way for your code to step through a procedure. It is common to implement a state machine using the case statement for a number of reasons.

1) Enumerated types show up in Signaltap for easy debugging.
2) The organized syntax creates more readable code.
3) easily expandable to include more states.

III. The state machine

This lab will assume that you have had a basic introduction to state machines in the lecture. We will cover some topics that are particular to the FPGA and HDL implementation of the logic. Altera offers a number of templates for the creation of a state machine [1]:

A. 4-State Mealy Machine:

This style of logic was coined in George Mealy's 1955 paper A Method for Synthesizing Sequential Circuits. The trademark feature is that it's outputs are determined by both the current state and the current inputs. [2]

B. 4-State Moore State Machine:

created a year after the Mealy machine the Moore Machine was described in a 1956 paper Gedanken-experiments on Sequential Machines. The difference is the Moore machine is only dependant on it's current state. [3]

C. Safe State Machine:

This style of machine uses a specific altera directive that inserts extra logic to detect invalid states and returns the state machine to the initial state.

D. User-Encoded State Machine:

This can be incorporated into all of the previous types. It allows the states to be named which aids in debugging and overall code readability.

```
module enum_fsm (
  input         clk,        //Clock for synchronus logic
  input         reset,      //Asynchronus reset for powerup
  input   int   data[3:0],  //Input bus
  output  int   o           //State Machine output
  );

enum int unsigned { S0 = 0, S1 = 2, S2 = 4, S3 = 8 } state,
     next_state;

always_comb begin : next_state_logic
  next_state = S0;
  case(state)
    S0: next_state = S1;
    S1: next_state = S2;
    S2: next_state = S3;
    S3: next_state = S3;
  endcase
end

always_comb begin
  case(state)
    S0: o = data[3];
    S1: o = data[2];
    S2: o = data[1];
    S3: o = data[0];
  endcase
end

always_ff@(posedge clk or negedge reset) begin
  if(~reset)
    state <= S0;
  else
    state <= next_state;
  end
endmodule
```

Listing 1. Example of enumerated state machine

1) Important reccomendations from altera: Quartus will recognise when you have created a state machine during synthesis. This will allow Quartus to optimize the design based on the known behavior of state machines. The Quartus II handbook offers the following reccomendations for writing state machines that we will follow.

1) Assign default values to outputs derived from the state machine so that synthesis does not generate unwanted latches.
2) Separate the state machine logic from all arithmetic functions and data paths, including assigning output values.

3) If your design contains an operation that is used by more than one state, define the operation outside the state machine and cause the output logic of the state machine to use this value.
4) Use a simple asynchronous or synchronous reset to ensure a defined power-up state. If your state machine design contains more elaborate reset logic, such as both an asynchronous reset and an asynchronous load, the Quartus II software generates regular logic rather than inferring a state machine.

IV. Lab Procedure

A. *Case State Warmup: The Multiplexer*

V. Lab Report

VI. Conclusion

References

[1] Altera. (2013) Quartus ii handbook. [Online]. Available: http://www.altera.com/literature/hb/qts/qts_qii51007.pdf
[2] W. Foundation. (2013) Mealy machines. [Online]. Available: https://en.wikipedia.org/wiki/Mealy_machine
[3] ——. (2013) Moore machines. [Online]. Available: https://en.wikipedia.org/wiki/Moore_machine