**California State University, Fresno**

**Lyles College of Engineering**

**Electrical and Computer Engineering Department**

**TECHNICAL REPORT**

**Experiment Title:** Examine a State Machine Implementation

**Course Title:** ECE 176 Computer Aided Design

**Date Submitted:** November 4, 2014

**Honor Code Statement:**

**“I have done my own work and have neither given nor received**

**unauthorized assistance on this work.”**

|  |  |
| --- | --- |
| **Prepared By:** |  |
| **Christopher Hays** |  |
|  |  |
| **Signature:** |  |

**INSTRUCTOR SECTION**

**Comments:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TABLE OF CONTENTS**

**Section:** **Page:**

Title Page ……………………………………………………………………...…. 1.

Table of Contents ……………………………………………………………….... 2.

1. Statement of Objectives ……………………………………………………….. 3.

2. Theoretical Background …………………….…………………………………. 3.

3. Experimental Procedure ………………………………………………………. 3.

3.1 Equipment Used ……………………………………………………... 3.

3.2 Laboratory Procedure ……………………………………………...… 3.

4. Analysis ……………………………………………………………………...... 4.

5. Conclusions ……………………………………………………………………. 5.

6. Appendix ………………………………………………………………………. 6.

**1. STATEMENT OF OBJECTIVES**

The objective of this assignment was to examine a state machine implementation in Quartus II using the Verilog Hardware Definition Language. The machine was designed to control an 8x8 multiplier, with inputs for a clock, a reset bit, a start bit, and a counter. The design was supplied by the instructor and was synthesized with the Cyclone II as the target hardware and an RTL simulation was conducted in ModelSim software to verify the operation of the state machine.

**2. THEORETICAL BACKGROUND**

Finite state machines have a set number of states that they can be in, and can only be in one state at a time. The machine changes from one state to the next when certain conditions have been met; these can be an input change or some timing criteria. This state machine has 6 states and is used to control a hardware multiplier. The IDLE state is used when the machine is ready to start a multiplication cycle, the CALC\_DONE state is used when the process is complete, LSB, MSB, and MID handle the three phases of multiplication and the ERR state is entered when something goes wrong.

**3. EXPERIMENTAL PROCEDURE**

**3.1 Equipment Used**

Altera ModelSim Software

Altera Quartus II

**3.2 Laboratory Procedure**

A new project was created in Quartus II, using the Cyclone II as the targeted hardware, Verilog as the design language, and ModelSim as the simulation environment, and the supplied module was added to the project. The inputs are start, clk, reset\_a, and count. Count is used to send the controller to the next state when the hardware is done with that portion of the multiplication. Outputs include input\_sel, shift\_sel, state\_out, done, clk\_ena, and sclr\_n. These outputs control the hardware or provide information to the user.

Four different “always” blocks control the operation of the module. The first one controls the synchronous updating of the state machine; on every positive clock edge the state is updated to the next state. The next block updates the next\_state variable based on the current state and the inputs. The states used are described in Figure 4. The third block creates a process for the Mealy output logic as a function of the current state and current inputs. The final block creates the Moore output, which is only a function of the current state. These output signals are used to control piecewise multiplication; when in LSB, the lowest 4 bits of the multiplicands are multiplied and added to an accumulator. MID cross-multiplies the middle bits and MSB handles the top 4 bits. The final output is then ready and the state machine is in the CALC\_DONE state.

The module was then compiled in order to create a .vo file. Next, analysis and synthesis was done which opened up ModelSim for RTL simulation (Figure 1). A simple testbench file that set the initial input values, asserted the start bit, set the counter values, then stopped; this was compiled in ModelSim along with the .vo file, which provided the hardware timing. A clock is simulated by inverting the clk signal every 20 nanoseconds. The code used is in the Appendix.

**4. ANALYSIS**

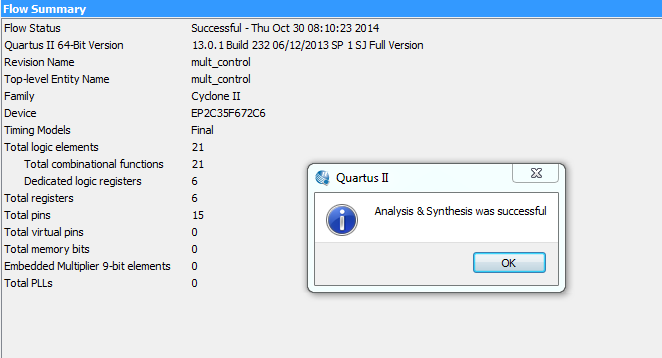


Figure 1: Successful Synthesis

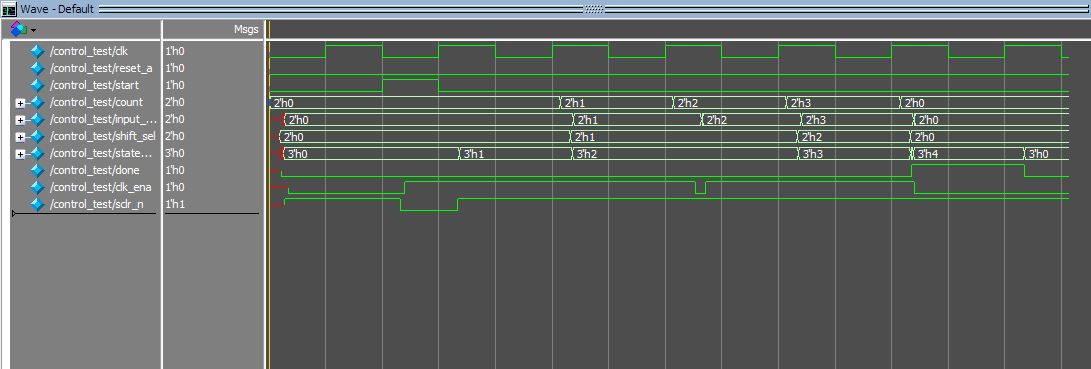


Figure 2: Output Waveforms

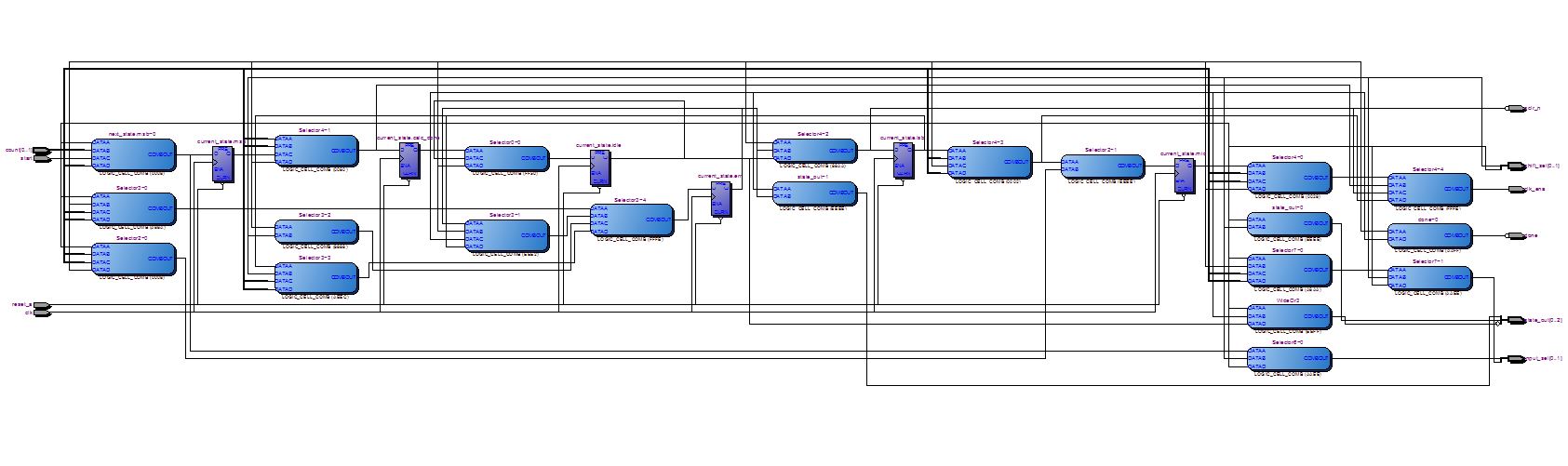


Figure 3: Block Diagram from Quartus II

The simulation begins by setting the initial values of the inputs at time 0 (Figure 2). The clk, reset\_a, start, and count all initialize to zero. After 40 nanoseconds the start bit is asserted for half a clock cycle, starting the state transitions. The counter input counts 0, 1, 2, 3, 4, and back to 0 in 40 nanosecond intervals. As the clock input is changing, the current state output is also changing appropriately according to the supplied schematic. The sclr\_n is driven low as the start bit is asserted, sending a clear signal to the accumulator to prepare for the calculation. The simulation is then ended with the “$stop” command 20 nanoseconds later. When the machine is in the CALC\_DONE state, the “done” output line is asserted. The block diagram of the register design is shown in Figure 3.

**5. CONCLUSIONS**

The control\_test.v testbench simulation file demonstrated that the design of the state machine was correct and working. The states changed as expected according to the input and the error state was avoided. The .vo file generated by Quartus II helped identify that the propagation delay on the Cyclone II is about 3 nanoseconds in this case. This controller design is helpful for controlling the specific steps involved in hardware multiplication, resulting in more control over the process compared to designing a multiplication module and having Quartus II design the hardware implementation.

**6. APPENDIX**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// ECE 176 Fall 2014

// Module Name: mult\_control File Name: mult\_control.v

// Module Function: This file contains the state machine control logic for

// the multiplier

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Begin module declaration for "mult\_control"

module mult\_control (

input clk, reset\_a, start, // Declare control inputs "clk", "reset\_a", "start", "count"

input [1:0] count,

output reg [1:0] input\_sel, shift\_sel, // Declare output control signals "in\_sel", "shift\_sel",

output reg [2:0] state\_out, // "state\_out", "done", "clk\_ena" and "sclr\_n"

output reg done, clk\_ena, sclr\_n

);

// Declare two variables named "current\_state" and "next\_state" to be state variables

reg [2:0] current\_state, next\_state;

// Declare parameters for states consisting of 6 values: "idle", "lsb", "mid", "msb", "calc\_done" and "err"

parameter idle=0, lsb=1, mid=2, msb=3, calc\_done=4, err=5;

// Create sequential process to control state transitions by making current\_state equal to next state on

// rising edge transitions; Use asynchronous clear control

always @(posedge clk, posedge reset\_a) begin

if (reset\_a)

current\_state <= idle;

else

current\_state <= next\_state;

end

// Create combinational process & case statement to determine next\_state based on current state and inputs\

always @ (\*) begin

case (current\_state)

idle :

if (start)

next\_state = lsb;

else

next\_state = idle;

lsb :

if (count == 0 && start == 0)

next\_state = mid;

else

next\_state = err;

mid :

if (start == 0 && count == 2)

next\_state = msb;

else if (start == 0 && count == 1)

next\_state = mid;

else

next\_state = err;

msb :

if (count == 3 && start == 0)

next\_state = calc\_done;

else

next\_state = err;

calc\_done :

if (!start)

next\_state = idle;

else

next\_state = err;

err :

if (start)

next\_state = lsb;

else

next\_state = err;

endcase

end

// Create process for Mealy output logic for input\_sel, shift\_sel, done, clk\_ena and sclr\_n (outputs function of inputs and current\_state)

always @ (\*) begin

// Initialize outputs to default values so case only covers when they change

input\_sel = 2'bxx;

shift\_sel = 2'bxx;

done = 1'b0;

clk\_ena = 1'b0;

sclr\_n = 1'b1;

case (current\_state)

idle :

if (start) begin

clk\_ena = 1'b1;

sclr\_n = 1'b0;

end

lsb :

if (count == 0 && start == 0) begin

input\_sel = 2'd0;

shift\_sel = 2'd0;

clk\_ena = 1'b1;

end

mid :

if (count == 2 && start == 0) begin

input\_sel = 2'd2;

shift\_sel = 2'd1;

clk\_ena = 1'b1;

end

else if (count == 1 && start == 0) begin

input\_sel = 1'd1;

shift\_sel = 1'd1;

clk\_ena = 1'b1;

end

msb :

if (count == 3 && start == 0) begin

input\_sel = 2'd3;

shift\_sel = 2'd2;

clk\_ena = 1'b1;

end

calc\_done :

if (!start)

done = 1'b1;

err :

if (start) begin

clk\_ena = 1'b1;

sclr\_n = 1'b0;

end

endcase

end

// Create process for Moore output logic for state\_out (outputs function of current\_state only)

always @(current\_state) begin

// Initialize state\_out to default values so case only covers when they change

state\_out = 3'd0;

case (current\_state)

idle : ;

lsb : state\_out = 3'd1;

mid : state\_out = 3'd2;

msb : state\_out = 3'd3;

calc\_done : state\_out = 3'd4;

err : state\_out = 3'd5;

endcase

end

endmodule // End module

// Christopher Hays

// ECE 176

// Assignment 9

// Multiplier Controller State Machine testbench

`timescale 1ns/100ps

module control\_test();

reg clk, reset\_a, start;

reg [1:0] count;

wire [1:0] input\_sel, shift\_sel;

wire [2:0] state\_out;

wire done, clk\_ena, sclr\_n;

initial begin

#0 clk = 0;

#0 reset\_a = 0;

#0 start = 0;

#0 count = 0;

#40 start = 1;

#20 start = 0;

#43 count = count+1;

#40 count = count+1;

#40 count = count+1;

#40 count = count+1;

#40 count = 0;

#20 $stop;

end

always

#20 clk = ~clk;

mult\_control U1 (clk, reset\_a, start, count, input\_sel, shift\_sel, state\_out, done, clk\_ena, sclr\_n);

endmodule

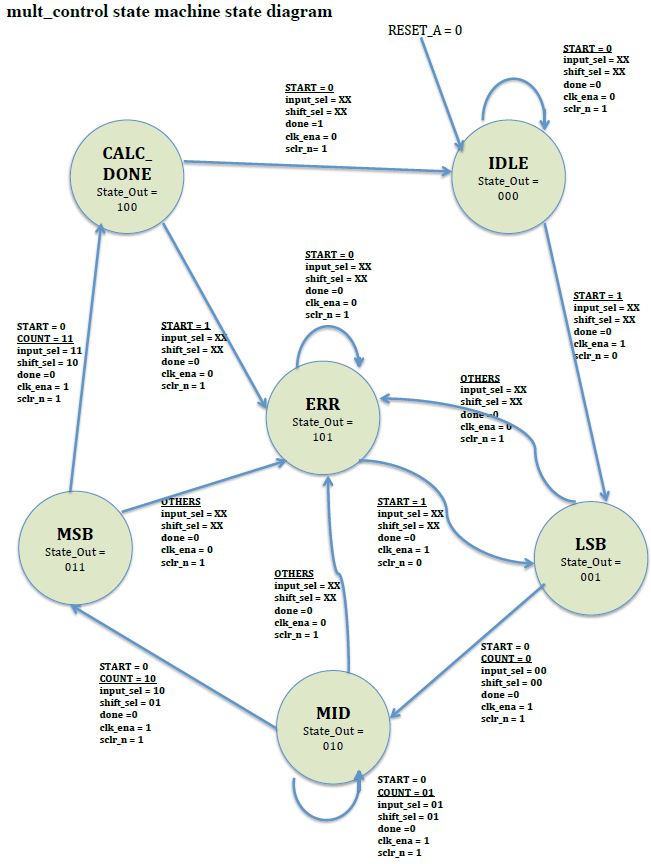


Figure 4: State Machine Diagram