**Lab Report**

|  |  |
| --- | --- |
| **Course：** | **Digital Design** |
| **Group Leader：** |  |
| **Group Students：** |  |
| **Major/**  **Class：** | **软件工程（卓越班）** |
| **Semester：** | **2017-2018 Second** |

**School of Software Engineering**

**2018/06**

# Lab 4 Finite State Machines

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Address：** | Room 333, Building B7 | | **Teacher：** | 陈俊颖 | |
| **Date：** | 2018/6/8 | | **Score：** |  | |
| **Lab Report File Name：** | | \_Digital Design\_Lab4 Report | | |
| **Comments：** |  | | | | |

1. **Experimental objective**

•Model Mealy FSMs

• Model Moore FSMs

**2. Principles**

(1)Mealy FSM

A general model of a Mealy sequential machine consists of a combinatorial network, which generates the outputs and the next state, and a state register which holds the present state as shown below. The state register is normally modeled as D flip-flops. The state register must be sensitive to a clock edge. The other block(s) can be modeled either using the always procedural block or a mixture of the always procedural block and dataflow modeling statements; the always procedural block will have to be sensitive to all inputs being read into the block and must have all output defined for every branch in order to model it as a combinatorial block.

(2) Moore FSM

A general model of a Moore sequential machine is shown below. Its output is generated from the state register block. The next state is determined using the present (current) input and the present (current)state. Here the state register is also modeled using D flip-flops. Normally Moore machines are described using three blocks, one of which must be a sequential and the other two can be modeled using always blocks or a combination of always and dataflow modeling constructs.

**3. Lab environments**

(1) Hardware requirement:

BASYS3

(2) Software requirements:

Vivado 2014.4

**4. Experiment contents**

•Design a sequence detector implementing a Mealy state machine using three always blocks

•Design another sequence detector implementing a Moore state machine using three always blocks

• Develop a testbench and verify the model through a behavioral simulation

• Verify the functionality in hardware using the Basys3 DDR board

**5. Experiment results and conclusions**

**(1) Mealy FSM**

**mealy\_sequence\_detector\_3processes:**

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////

// Module Name: mealy\_sequence\_detector\_3processes

/////////////////////////////////////////////////////////////////

module mealy\_sequence\_detector\_3processes(

input clk,

input reset,

input ain,

output reg [3:0] count,

output reg yout

);

reg [2:0] state, nextstate;

parameter [2:0] S0=0, S1=1, S2=2, S3=3, S4=4, S5=5;

//state register and counter

always @(posedge clk or posedge reset)

begin

if (reset)

begin

state <= S0;count<=4'b0000;

end

else

begin

state <= nextstate;

if(ain)

count<=count+1;

end

end

//output logic

always @(state or ain)

begin

yout= 1'b0;

case(state)

S0:if(!ain)

yout=1;

else

yout=0;

S1:

yout=0;

S2:

if(ain)

yout=1;

else

yout=0;

endcase

end

//next state logic

always @(ain or state)

begin

case(state)

S0:

if(ain)

nextstate = S1;

else

nextstate=S0;

S1:

if(ain)

nextstate = S2;

else

nextstate=S1;

S2:

if(ain)

nextstate = S0;

else

nextstate=S2;

endcase

end

endmodule

mealy\_sequence\_detector\_3processes testbench:

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Module Name: mealy\_sequence\_detector\_3processes\_tb

//////////////////////////////////////////////////////////////////////////////////

module mealy\_sequence\_detector\_3processes\_tb(

);

reg clk;

reg reset;

reg ain;

wire [3:0] count;

wire yout;

mealy\_sequence\_detector\_3processes DUT (.clk(clk), .reset(reset), .ain(ain), .count(count), .yout(yout));

initial

begin

clk = 0;

forever

begin

#5 clk = 1;

#5 clk = 0;

end

end

initial

begin

reset = 1'b1;

ain = 1'b0;

#20; // wait for 2 clock cycles

reset = 1'b0;

#10 ain = 1'b0;

#30 ain = 1'b1;

#20 ain = 1'b0;

// Write the rest of the input signal test cases.

#20 ain=1'b1;

#30 ain=1'b0;

#30 ain=1'b1;

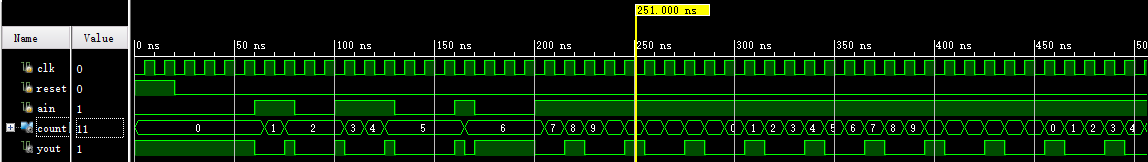
#10 ain=1'b0;

#30 ain=1'b1;

end

endmodule

**Simulation waveforms:**



**Experiment board testing result:**







Use SW15 as the clock input, SW0 as the a in input, the BTNU button as reset input to the circuit, number of 1s count on LED7:LED4, and LED0 as the y out output. Go through the design flow, generate the bitstream, and download it into the Basys3 board.

**(2) Moore FSM**

**moore\_sequence\_detector\_3processes:**

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////

// Module Name: moore\_sequence\_detector\_3processes

/////////////////////////////////////////////////////////////////

module moore\_sequence\_detector\_3processes(

input clk,

input reset,

input [1:0] ain,

output reg yout

);

reg [1:0] state, nextstate;

parameter [1:0] S0=0, S1=1, S2=2, S3=3;

//state register logic

always @(posedge clk or posedge reset)

begin

if(reset)state<=S0;

else state<=nextstate;

end

//next state logic

always @(state or ain)

begin

case(state)

S0:

begin

if(ain==2'b10|ain==2'b11)nextstate=S1;

else if(ain==2'b00|ain==2'b01)nextstate=S0;

end

S1:

begin

if(ain==2'b10|ain==2'b11)nextstate=S1;

else if(ain==2'b00)nextstate=S2;

else if(ain==2'b01)nextstate=S0;

end

S2:

begin

if(ain==2'b11|ain==2'b00)nextstate=S2;

else if(ain==2'b01|ain==2'b10)nextstate=S3;

end

S3:

begin

if(ain==2'b10|ain==2'b01)nextstate=S3;

else if(ain==2'b00)nextstate=S0;

else if(ain==2'b11)nextstate=S2;

end

default: nextstate=S0;

endcase

end

//output logic

always @(state)

begin

if(state==S0|state==S1) yout=1'b0;

else if(state==S2|state==S3)yout=1'b1;

end

endmodule

**moore\_sequence\_detector\_3processes testbench:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Module Name: moore\_sequence\_detector\_3processes\_tb

//////////////////////////////////////////////////////////////////////////////////

module moore\_sequence\_detector\_3processes\_tb(

);

reg clk;

reg reset;

reg [1:0] ain;

wire yout;

moore\_sequence\_detector\_3processes DUT (.clk(clk), .reset(reset), .ain(ain), .yout(yout));

initial

begin

clk = 0;

forever

begin

#5 clk = 1;

#5 clk = 0;

end

end

initial

begin

reset = 1'b1;

ain = 2'b00;

#20; // wait for 2 clock cycles

reset = 1'b0;

#10 ain = 2'b00;

#10 ain = 2'b11;

#10 ain = 2'b10;

#10 ain = 2'b00; // toggle output to 1

#20 ain = 2'b10;

#10 ain = 2'b00; // toggle output to 0

// Write the rest of the input signal test cases.

#10 ain = 2'b11;

#10 ain = 2'b00;

#10 ain = 2'b01;

#10 ain = 2'b00;

#10 ain = 2'b10;

#10 ain = 2'b11;

#10 ain = 2'b00;

#10 reset = 1'b1; ain = 2'b00;

#10 ain = 2'b00; reset= 1'b0;

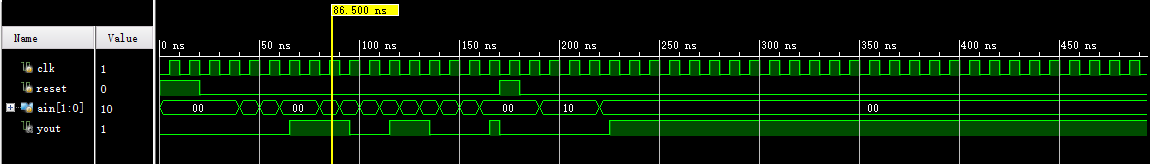
#10 ain = 2'b10;

#30 ain = 2'b00;

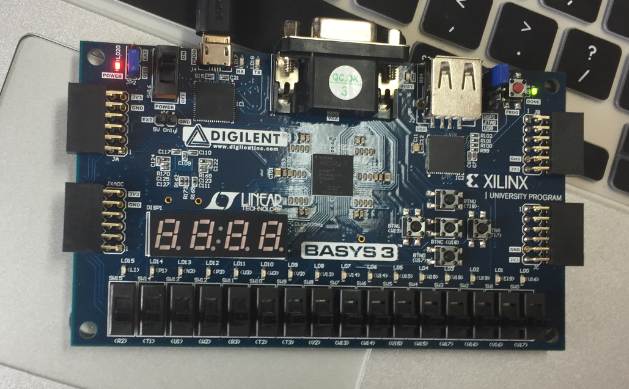
end

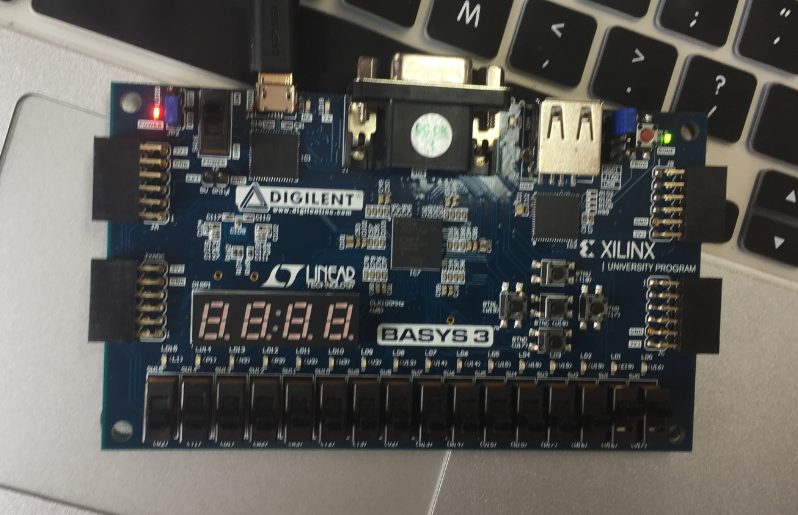
endmodule

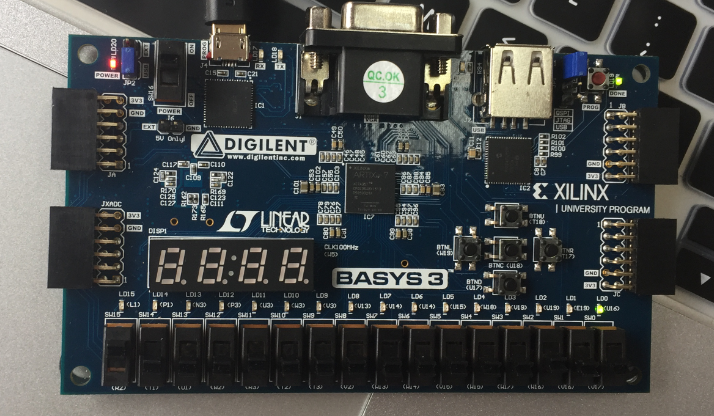
**Simulation waveforms:**



**Experiment board testing result:**







Switching SW15 provides clock cycles and the state of SW [1:0] represents input. Initially, the initial sequence is 00, then we provide input sequence 11,00, along with the rising edge of the clock, and that makes LED0 turning on. Other input sequence has similar effect.

**Conclusion:**

In this lab, we learned Mealy and Moore state machine modeling methodologies. We designed and implemented sequence detector, a sequence generator, and code converters using the two and three always blocks styles.