Finite State Machines

Introduction

Finite State Machines (FSM) are sequential circuit used in many digital systems to control the behavior of systems and dataflow paths. Examples of FSM include control units and sequencers. This lab introduces the concept of two types of FSMs, Mealy and Moore, and the modeling styles to develop such machines. *Please refer to the Vivado tutorial on how to use the Vivado tool for creating projects and verifying digital circuits*.

Objectives

After completing this lab, you will be able to:

- Model Mealy FSMs
- Model Moore FSMs

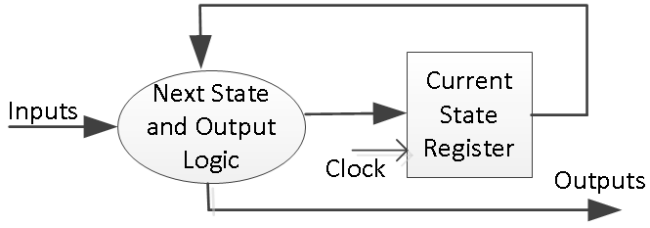
Mealy FSM Part 1

A finite-state machine (FSM) or simply a state machine is used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of user-defined states. The machine is in only one state at a time; the state it is in at any given time is called the *current state*. It can change from one state to another when initiated by a triggering event or condition; this is called a *transition*. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

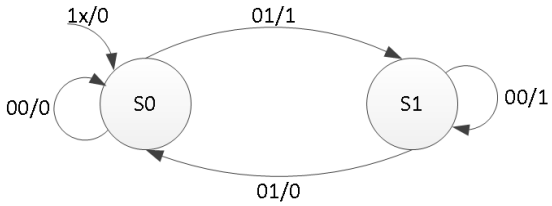
The behavior of state machines can be observed in many devices in modern society performing a predetermined sequence of actions depending on a sequence of events with which they are presented. Simple examples are vending machines which dispense products when the proper combination of coins are deposited, elevators which drop riders off at upper floors before going down, traffic lights which change sequence when cars are waiting, and combination locks which require the input of combination numbers in the proper order.

The state machines are modeled using two basic types of sequential networks- Mealy and Moore. In a Mealy machine, the output depends on both the present (current) state and the present (current) inputs. In Moore machine, the output depends only on the present state.

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. The two blocks Mealy machine can be viewed as

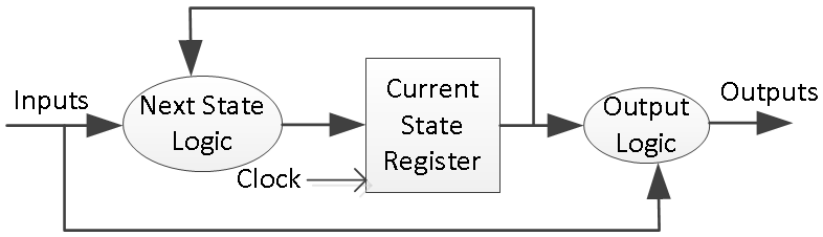


Here are the state diagram of a parity checker Mealy machine and the associated model.




```
module mealy_2processes(input clk, input reset, input x, output reg parity);
    reg state, nextstate;
    parameter S0=0, S1=1;

always @(posedge clk or posedge reset)    // always block to update state
if (reset)
     state <= S0;
else
    state <= nextstate;

always @(state or x) // always block to compute both output & nextstate
begin
    parity = 1'b0;
    case(state)
       S0: if(x)
           begin
               parity = 1; nextstate = S1;
           end
           else
                nextstate = S0;
       S1: if(x)
                nextstate = S0;
           else
           begin
                parity = 1; nextstate = S1;
           end
       default:
           nextstate = S0;
    endcase
end
endmodule
```

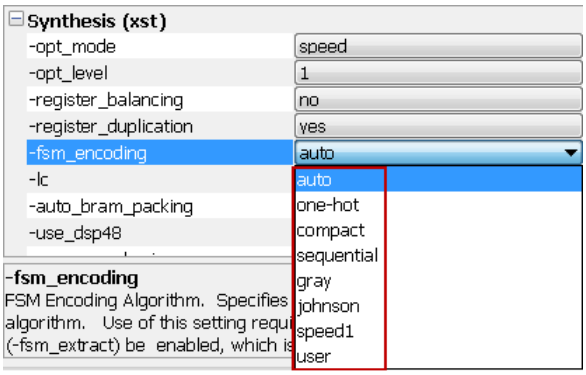
The three blocks Mealy machine and the associated model are shown below.



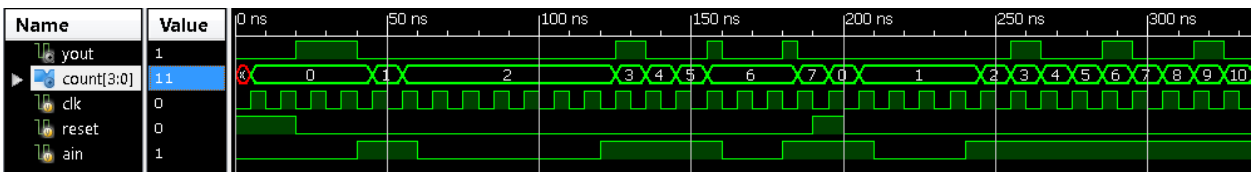

```
module mealy_3processes(input clk, input reset, input x, output reg parity);
    reg state, nextstate;
    parameter S0=0, S1=1;

  always @(posedge clk or posedge reset)  // always block to update state
  if (reset)
       state <= S0;
  else
      state <= nextstate;
```

```
always @(state or x)  // always block to compute output
begin
   parity = 1'b0;
   case(state)
      S0: if(x)
             parity = 1;
      S1: if(!x)
             parity = 1;
   endcase
end
always @(state or x)  // always block to compute nextstate
begin
   nextstate = S0;
   case(state)
      S0: if(x)
             nextstate = S1;
      S1: if(!x)
             nextstate = S1;
      endcase
end
endmodule
```

The state assignments can be of one-hot, binary, gray-code, and other types. Usually, the synthesis tool will determine the type of the state assignment, but user can also force a particular type by changing the synthesis property as shown below. The state assignment type will have an impact on the number of bits used in the state register; one-hot encoding using maximum number of bits but decodes very fast to compact (binary) encoding using smallest number of bits but taking longer to decode.

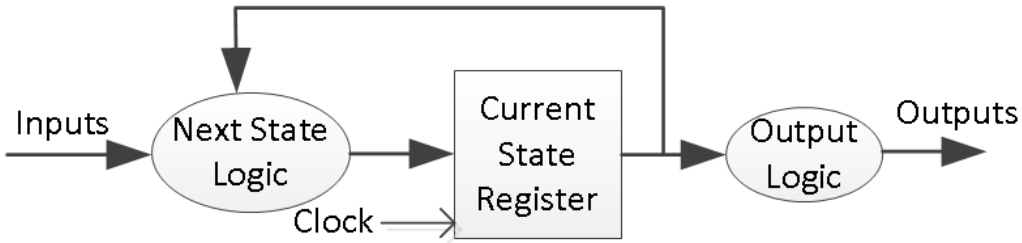


1-1. Design a sequence detector implementing a Mealy state machine using three always blocks. The Mealy state machine has one input (ain) and one output (yout). The output yout is 1 if and only if the total number of 1s received is divisible by 3 (hint: 0 is inclusive, however, reset cycle(s) do not count as 0- see in simulation waveform time=200). Develop a testbench and verify the model through a behavioral simulation. Use SW15 as the clock input, SW0 as the ain input, the BTNU button as reset input to the circuit, number of 1s count on LED7:LED4, and LED0 as the yout output. Go through the design flow, generate the bitstream, and download it into the Nexys4 board. Verify the functionality.

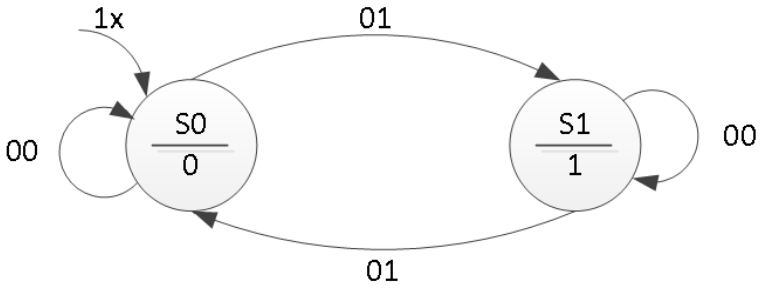


Moore FSM Part 2

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.



Here is the state graph of the same parity checker to be modeled as a Moore machine. The associate model is shown below.

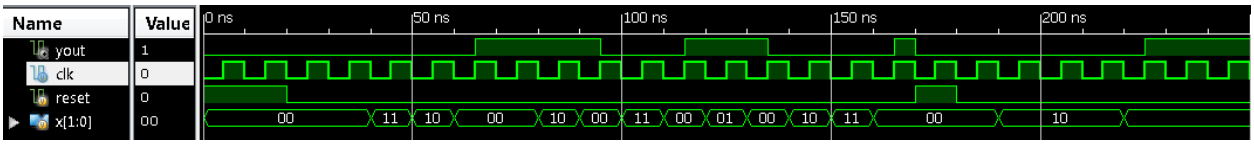



```
module moore_3processes(input clk, input reset, input x, output reg parity);
    reg state, nextstate;
    parameter S0=0, S1=1;

  always @(posedge clk or posedge reset)  // always block to update state
  if (reset)
       state <= S0;
  else
      state <= nextstate;

  always @(state)              // always block to compute output
  begin
      case(state)
         S0: parity = 0;
         S1: parity = 1;
      endcase
  end
  always @(state or x)         // always block to compute nextstate
  begin
      nextstate = S0;
      case(state)
         S0: if(x)
                nextstate = S1;
         S1: if(!x)
                nextstate = S1;
      endcase
  end
  endmodule
```

The output block when it is simple, as in this example, can be modeled using dataflow modeling constructs. The following code can be used instead of the always block. You also need to change the output type from reg to wire.

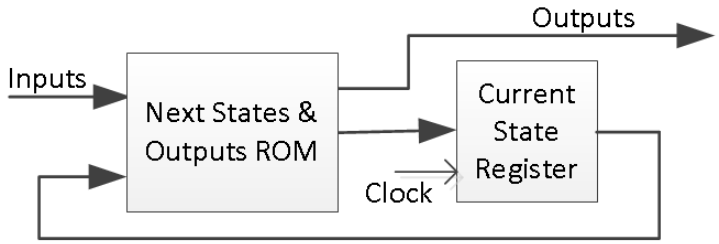
```
assign parity = (state==S0) ? 1'b0: 1'b1;
```

2-1. Design a sequence detector implementing a Moore state machine using three always blocks. The Moore state machine has two inputs (ain[1:0]) and one output (yout). The output yout begins as 0 and remains a constant value unless one of the following input sequences occurs:
(i) The input sequence ain[1:0] = 01, 00 causes the output to become 0
(ii) The input sequence ain[1:0] = 11, 00 causes the output to become 1
(iii) The input sequence ain[1:0] = 10, 00 causes the output to toggle.
Develop a testbench (similar to the waveform shown below) and verify the model through a behavioral simulation. Use SW15 as the clock input, SW1-SW0 as the ain[1:0] input, the BTNU button as reset input to the circuit, and LED0 as the yout output. Go through the design flow, generate the bitstream, and download it into the Nexys4 board. Verify the functionality.



Mealy FSM Using ROM — Part 3

A Mealy sequential machine can also be implemented using a ROM memory as shown below. The ROM memory holds the next state and output content. The external inputs and the current state form the address input to the ROM. The ROM typically is implemented using LUTs instead of BlockRAM since LUTs give a better utilization ratio resulting from a smaller number of states in a design.



3-1. Design a specific counts counter (counting sequence listed below) using ROM to develop a Mealy state machine. Develop a testbench and verify the model through behavioral simulation. Use SW15 as the clock input, the BTNU button as reset input to the circuit, and LED2:LED0 as the count output of the counter. Go through the design flow, generate the bitstream, and download it into the Nexys4 board. Verify the functionality.
The counting sequence will be: 000, 001, 011, 101, 111, 010 (repeat) 000, …

Conclusion

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