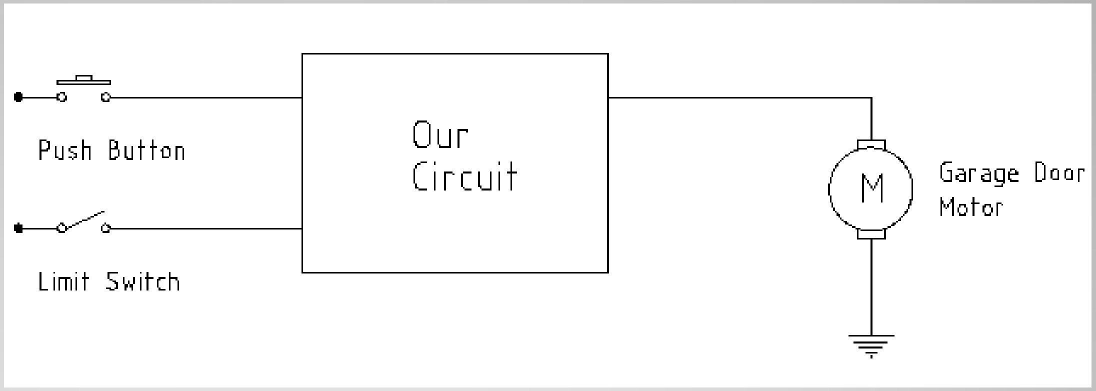
State Machines Introduced

Lecture 8

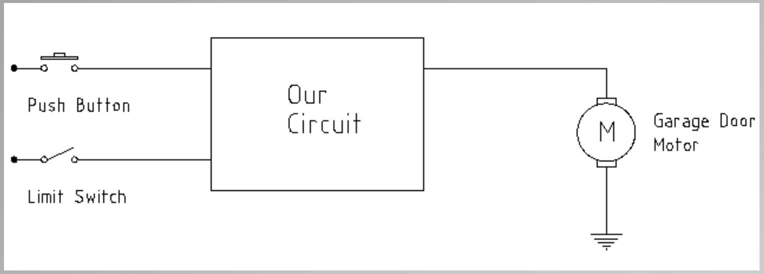
Finite State Machines

- “Brains of the outfit”
- Use registers to save current state.
- Used to control datapath components
- In Verilog we build state machines pretty much like we would in software.



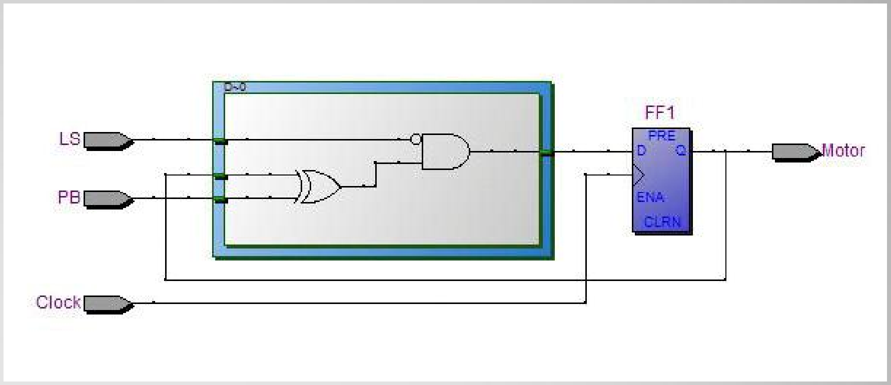
We want a circuit that fully opens the garage door when the push-button is momentarily pressed. Limit switch turns off the motor.

Garage Door Example

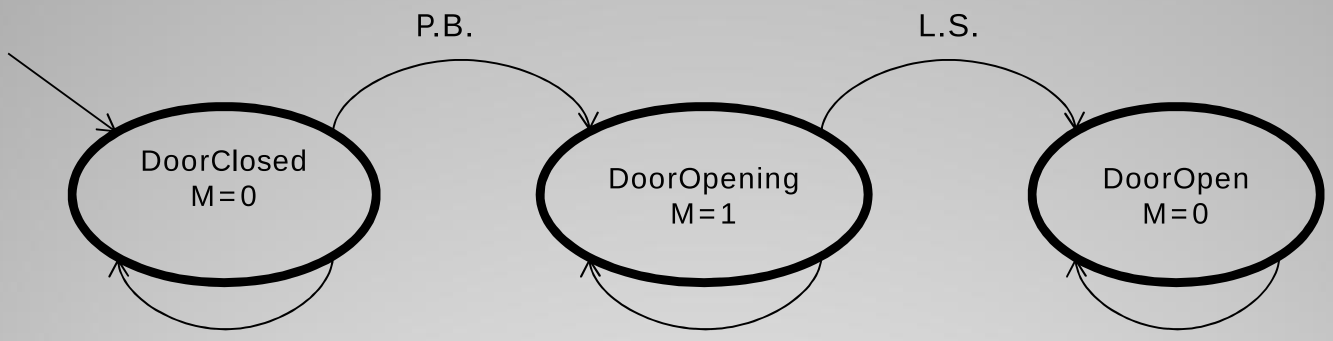


Truth Table for Motor Circuit FF

Push-Button	Limit Switch	Current Q	D (next Q)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

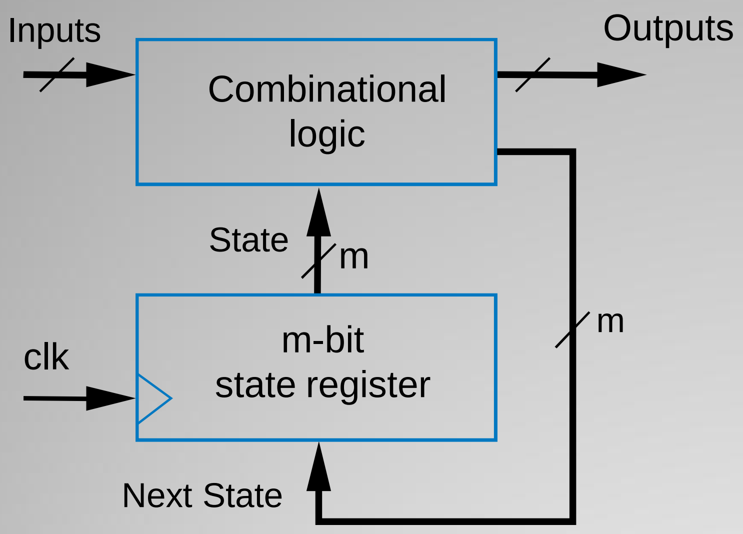


Garage Door Circuit Using D Flip-Flop (TCES 220 Method)



Motor State = M
Pushbutton State = PB
Limit Switch state = LS

Garage Door Opener States and Transitions



General FSM Architecture

You are used to forming the truth table (TT) for the combinational part:

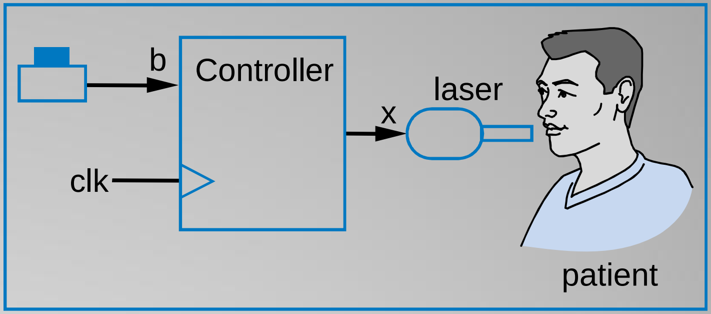
- Inputs to TT would be the FSM inputs plus the current state
- Outputs to the TT would be the FSM outputs plus the next state

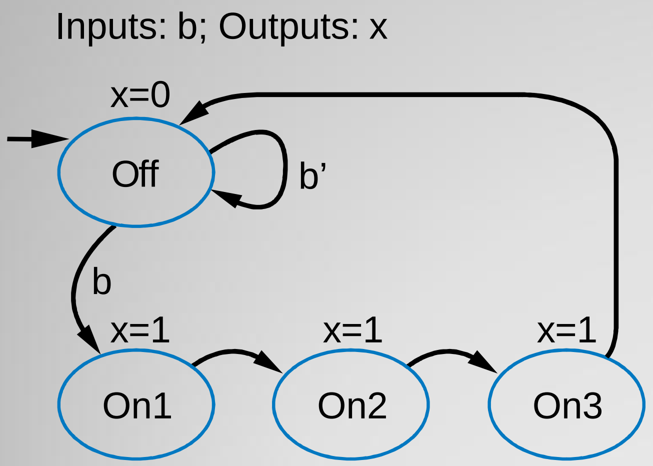
Verilog Code Design (Architecture)

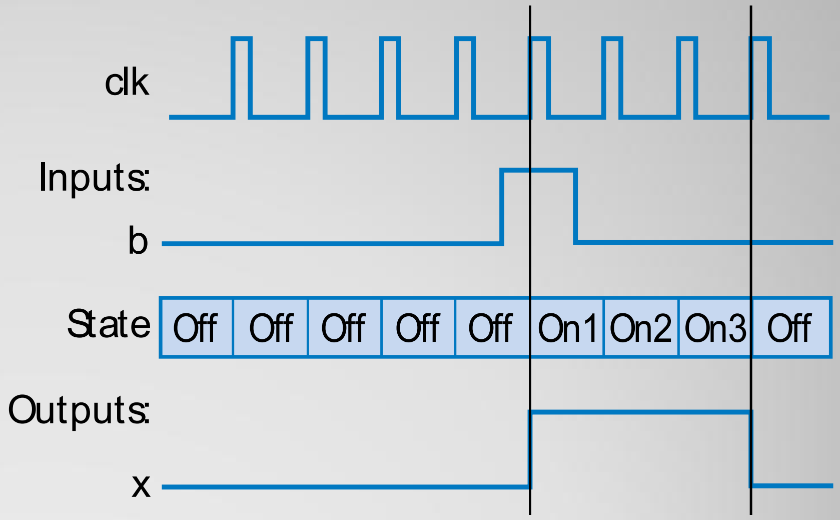
SM Rules of Thumb

- Name states with localparams
- Have State and NextState reg varialbes of the correct size
- Have a combinational loop and a sequential loop
- Combinational Loop
  - Set all variables to default values
  - Have a case for each state
  - Have a default case
- Sequential Loop
  - State <= NextState
  - Take care of Reset here

- Consider a laser surgery system
- Laser should be ON exactly 30 ns
- Assume a clock with a 10 ns period
- Example system where software in a µProcessor is not such a good idea.







Transitions only on clock edges

Example FSM

From F. Vahid's Digital Design, 2007

```
module LaserSM( B, X, Clk, Rst );
  input B;
  input Clk, Rst;
  output reg X;

  // State Name assignments
  localparam S_Off = 0,
             S_On1 = 1,
             S_On2 = 2,
             S_On3 = 3;

  reg [1:0] State, StateNext;
```

Verilog Laser Surgery FSM, Front end

```
// CombLogic (use blocking assignments)
 always @ ( State, B ) begin
   case (State)

     S_Off: begin
       X = 0;  // laser OFF
       if ( B == 0 )  begin
         StateNext = S_Off;  // button not pushed
       end
       else begin
         StateNext = S_On1;  // button pushed
       end
     end

     S_On1: begin
       X = 1; // laser ON
       StateNext = S_On2;   // always go to next state
     end
```

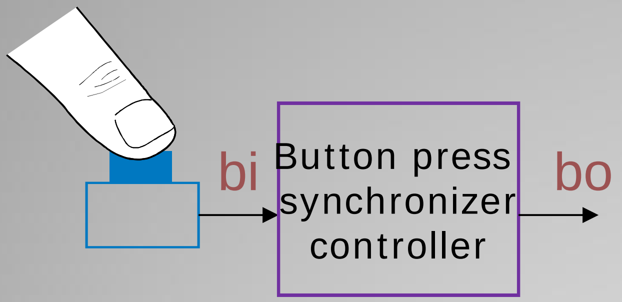
Combinational Logic Part

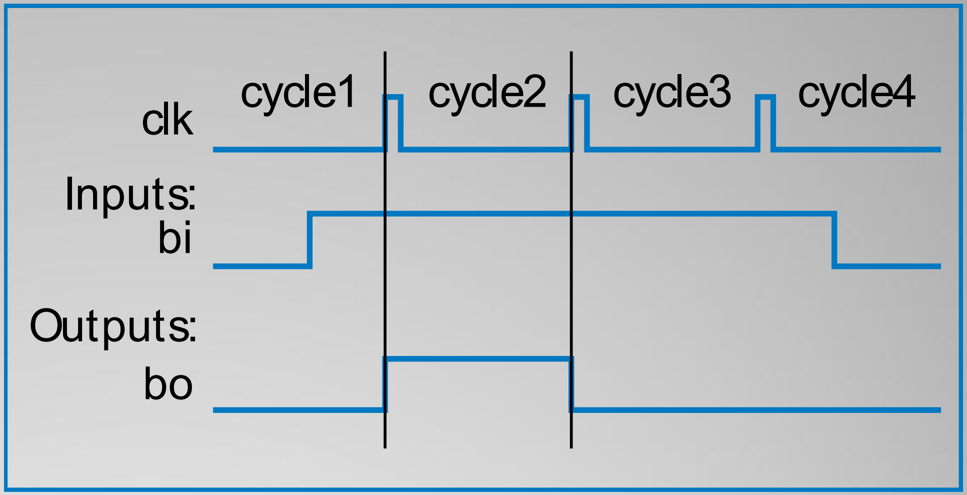
```
S_On2: begin
      X = 1; // laser ON
      StateNext = S_On3;  // always transition to next state
    end

    S_On3: begin
      X = 1; // laster still ON
      StateNext = S_Off;  // always transition to off state
    end

    default: begin
      X = 0;
      StateNext = S_Off;
    end
  endcase
end // always
```

Combinational Logic, cont.

```
 // StateReg (use non-blocking assignments)
  always @( posedge Clk ) begin
    if ( Rst == 1 ) begin
      State <= S_Off;     // reset overrides everything else
    end
    else begin
      State <= StateNext; // otherwise go to the state we set
    end
  end  // always

endmodule
```

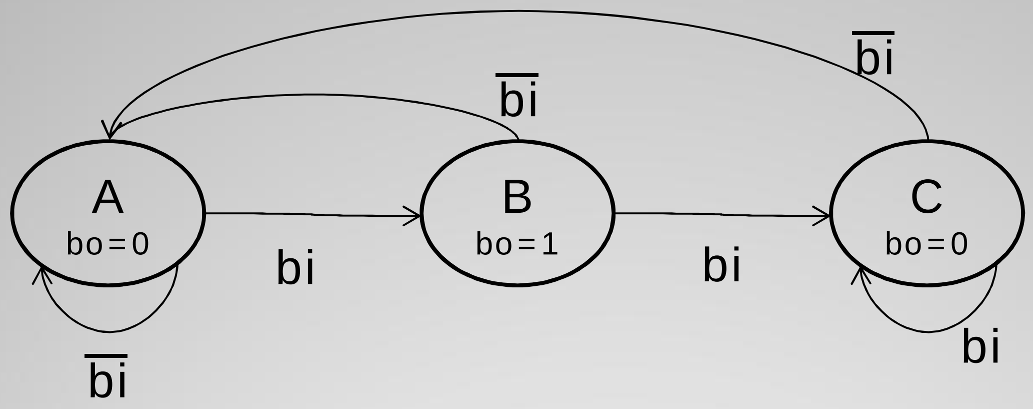
State Register





Button Press Synchronizer

From F. Vahid's *Digital Design*, 2007



Button Press State Transition Diagram

```
module ButtonSync( Bi, Bo, Clk );
  input Bi;      // input button press
  input Clk;     // system clock
  output reg Bo;  // our output

  // State assignments
  localparam S_A = 2'h0,
             S_B = 2'h1,
             S_C = 2'h2;

  reg [1:0] State, StateNext;
```

Verilog for Button Synchronizer

```
// CombLogic
 always @ ( State, Bi ) begin
   case (State)

     S_A: begin
       Bo = 0;  // turn output OFF
       if ( Bi == 1'b1 )
         StateNext = S_B;  // button push detected
       else
         StateNext = S_A;
     end

     S_B: begin
       Bo <= 1; // turn output ON
       if ( Bi == 1'b1 )
         StateNext <= S_C;
       else
         StateNext = S_A;
     end
```

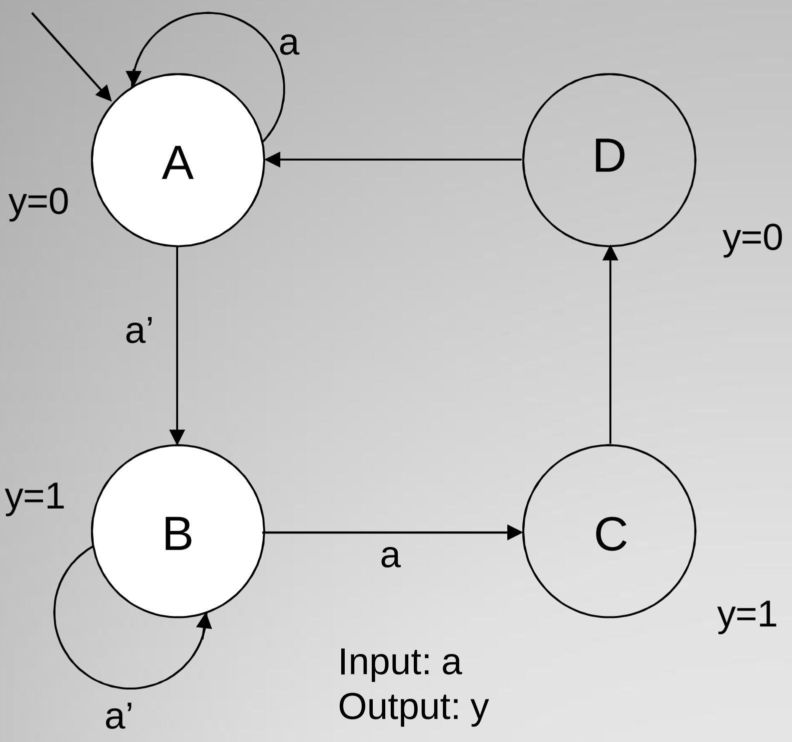
Combinational Part

```
S_C: begin
       Bo = 0; // turn output OFF
       if ( Bi == 1'b1 )
         StateNext = S_C;  // stay in this state
       else
         StateNext = S_A;  // otherwise, back to A
     end

     default: begin
       Bo = 0;
       StateNext = S_A;
     end
   endcase
 end // always
```

Combinational Part, cont.

```
  // StateReg
   always @( posedge Clk ) begin
       State <= StateNext;   // new state
   end  // always

endmodule
```

State Register Part



Design a Verilog module that implements the state machine shown in the diagram.

Compile your design using Quartus.

Inspect the state machine that Quartus produces and verify that it's what you expected.

Test your design using ModelSim

Can You Build This State Machine?