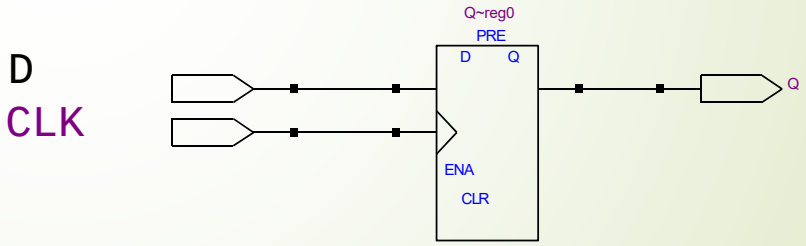
EGC221
Class Notes
11/20/2018

Baback Izadi
Division of Engineering Programs
bai@engr.newpaltz.edu

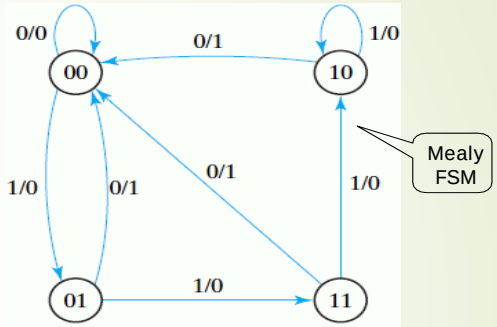
Verilog – D Flip-flop

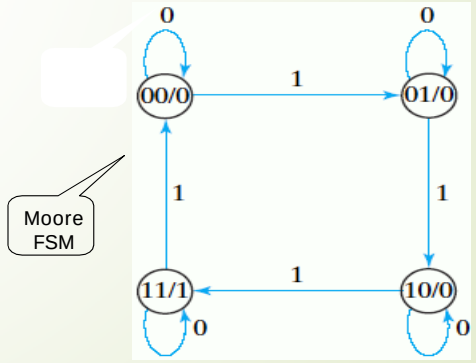
```
module D-flipflop (D, Clk, Q);

   input D, Clk;
   output Q;  reg Q;

   always @(posedge Clk)
      Q = D;

Endmodule
```



Finite State Machines (FSM)

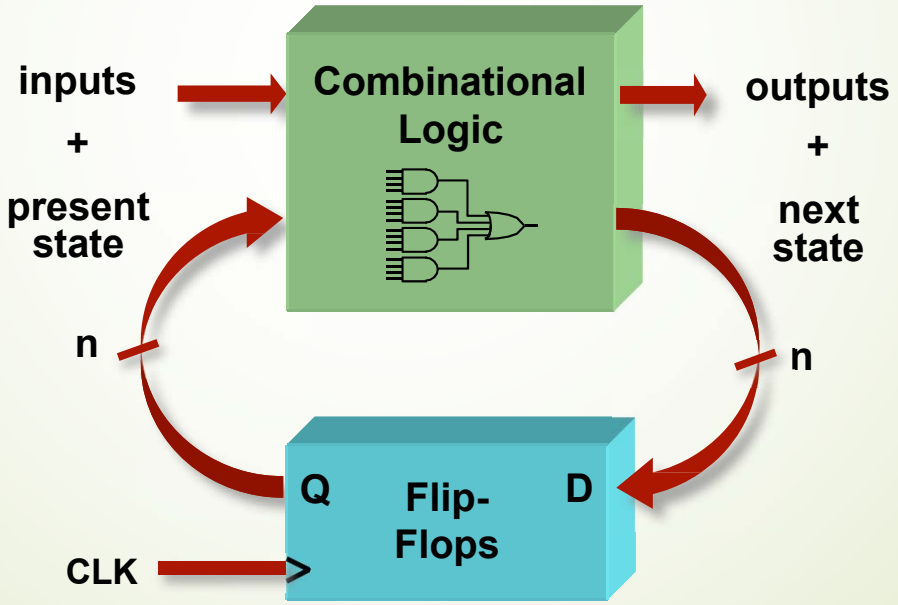
- State diagrams are representations of Finite State Machines (FSM)
- Mealy FSM
  - Output depends on input and state
  - Output is not synchronized with clock
  - can have temporarily unstable output
- Moore FSM
  - Output depends only on state





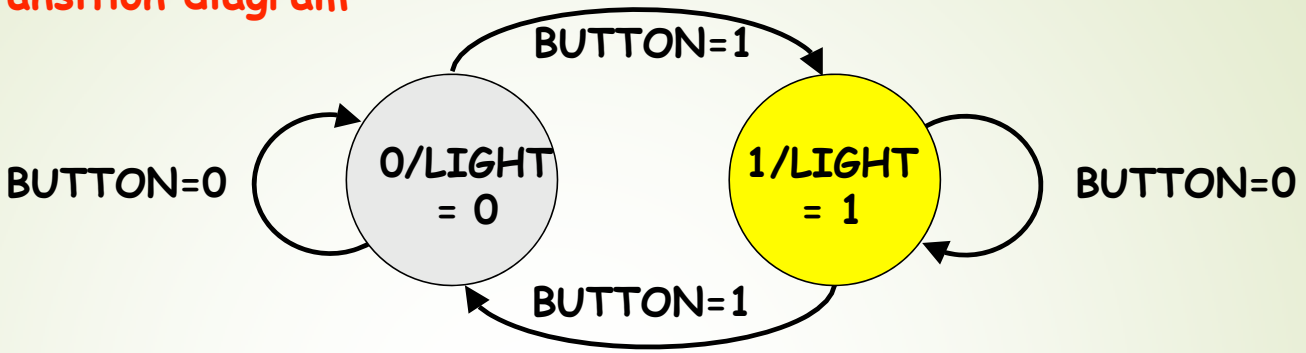
Finite State Machines

- **Finite State Machines (FSMs) are a useful abstraction for sequential circuits with centralized "states" of operation**
- **At each clock edge, combinational logic computes *outputs* and *next state* as a function of *inputs* and *present state***

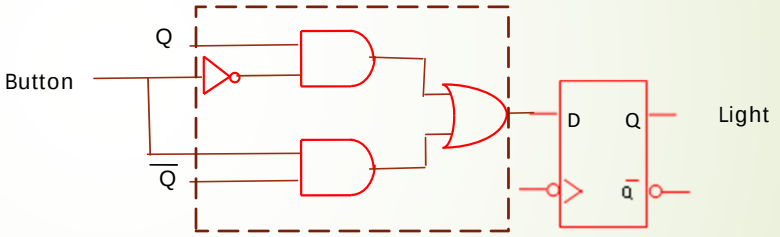


Example: Light Switch

- **State transition diagram**

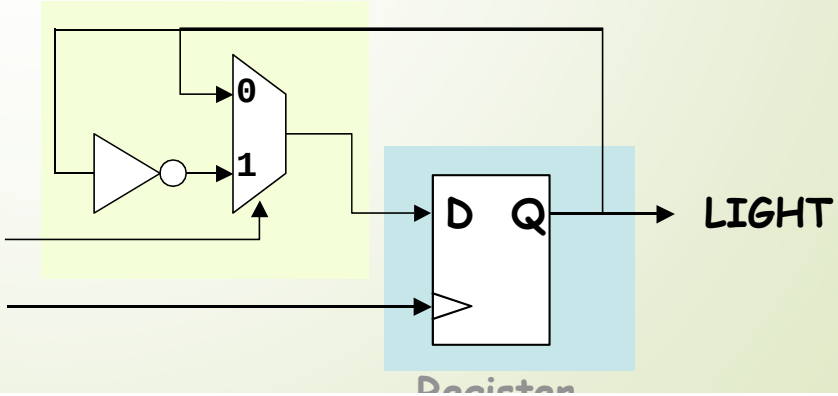


PS		NS	
Q	Button	Q & D	Light
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1



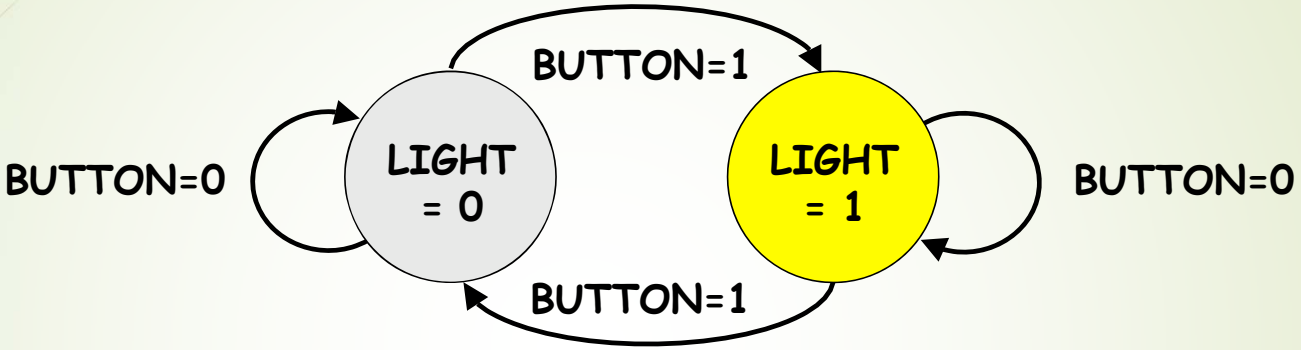
D = Q'B + QB'
Light = Q

Note: B = Button

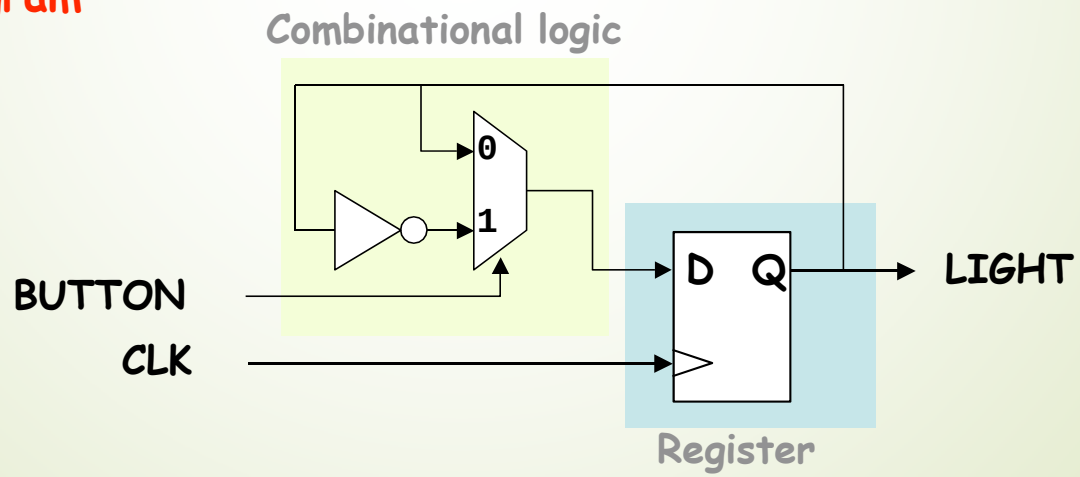


Example: Light Switch

- State transition diagram

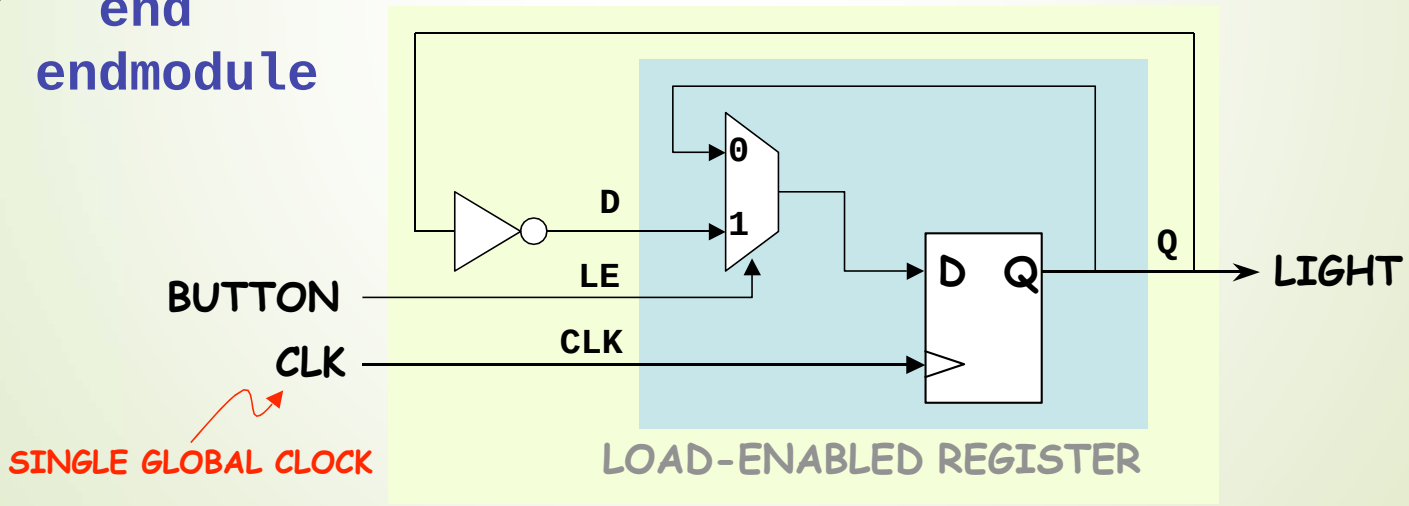


- Logic diagram



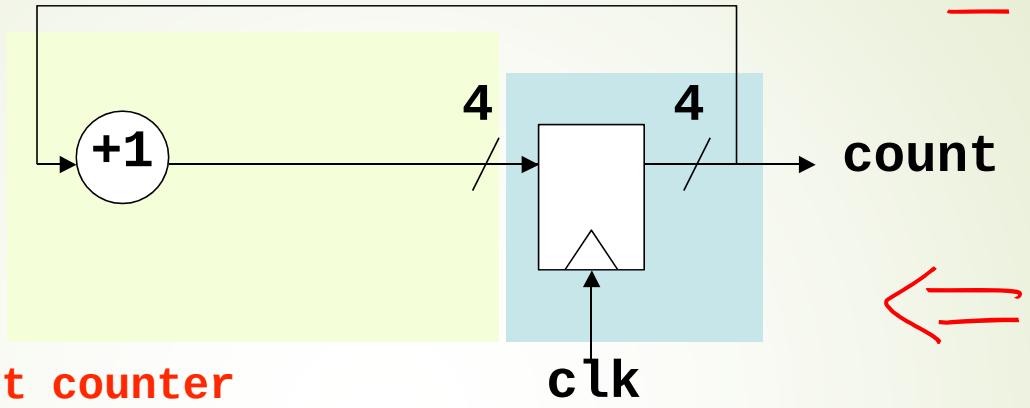
Clocked circuit for on/off button

```
module onoff(clk,button,light);
  input clk,button;
  output light; reg light;
  always @ (posedge clk) begin
   if (button) light <= ~light;
  end
endmodule
```



Example: 4-bit Counter

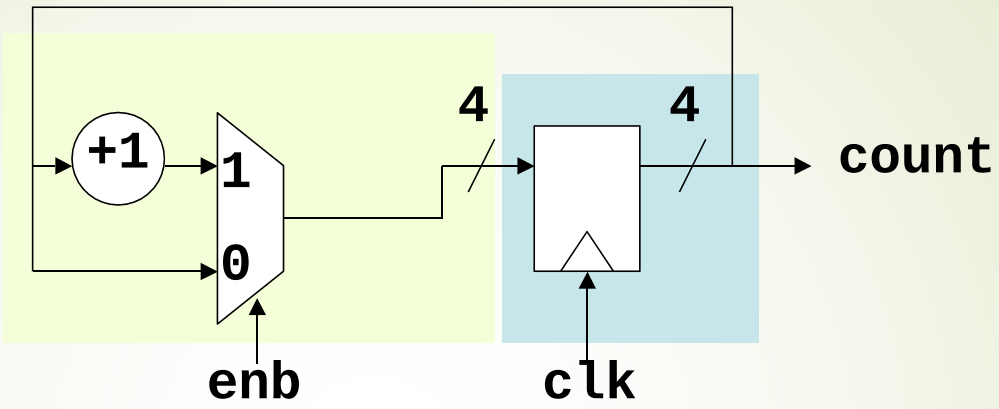
- Logic diagram



- Verilog

```
# 4-bit counter
module counter(clk, count);
  input clk;
  output [3:0] count;
  reg [3:0] count;

  always @ (posedge clk) begin
    count <= count+1;
  End
endmodule
```

- Logic diagram



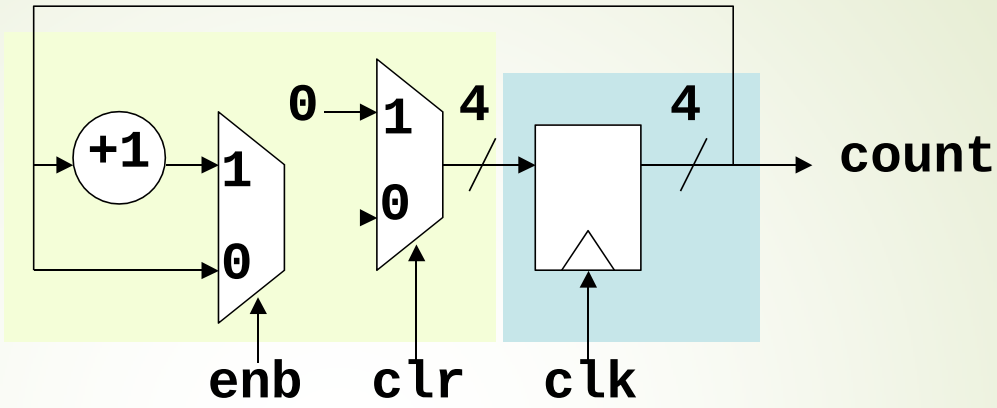
- Verilog

```
# 4-bit counter with enable
module counter(clk,enb,count);
  input clk,enb;
  output [3:0] count;
  reg [3:0] count;

  always @ (posedge clk) begin
    count <= enb ? count+1 : count;
  end
endmodule
```

Could I use the following instead?
`if (enb) count <= count+1;`
else

Example: 4-bit Counter

- Logic diagram



- Verilog

```
# 4-bit counter with enable and synchronous clear
module counter(clk,enb,clr,count);
  input clk,enb,clr;
  output [3:0] count;
  reg [3:0] count;

  always @ (posedge clk) begin
    count <= clr ? 4'b0 : (enb ? count+1 : count);
  end
endmodule
```

4-bit Shift Register with Reset

```
module srg_4_r_v (CLK, RESET, SI, Q,SO);
input CLK, RESET, SI;
output [3:0] Q;
output SO;
reg [3:0] Q;
assign SO = Q[3];
always@(posedge CLK or posedge RESET)  begin
  if (RESET)
    Q <= 4'b0000;
  else
    Q <= {Q[2:0], SI};
    end
endmodule
```

4-bit Binary Counter with Reset

```
module count_4_r_v (CLK, RESET, EN, Q, CO);
input CLK, RESET, EN;
output [3:0] Q;
output CO;
reg [3:0] Q;
assign CO = (count == 4'b1111 && EN == 1’b1) ? 1 : 0;
always@(posedge CLK or posedge RESET)
  begin
  if (RESET)
   Q <= 4'b0000;
  else if (EN)
   Q <= Q + 4'b0001;
  end
endmodule
```

```
//The goal of this always procedural block is to
generate 1Hz clock from a
//50MHz clock that is used in the Altera FPGA board.
module Divide_by_50M_counter(clr,clk,clk_1Hz);
input clr,clk;
output clk_1Hz;
reg clk_1Hz =1'b0;
integer counter_50M =0;
always @(posedge clk, posedge clr)
begin
  if (clr)
      counter_50M <=0;
  else if (counter_50M <25000000)
    begin
       counter_50M <= counter_50M + 1;
    end
  else if (counter_50M ==25000000)
     begin
       clk_1Hz <= !clk_1Hz;
       counter_50M <=0;
    end
end
endmodule
```

```
module up_down_counter(mode,clr,ld,D_in,clk,count,clk_1Hz);
input mode,clr,ld,clk;
input [3:0] D_in;
output clk_1Hz;
output [3:0] count;
reg [3:0] count;
reg clk_1Hz =1'b0;
integer counter_50M =0;
//The goal of this always procedural block is to generate 1Hz clock from a
//50MHz clock that is used in the Xilinx Nexus FPGA board.
always @(posedge clk)
begin
 if (clr)
    counter_50M =0;
 else if (counter_50M <25000000)
   begin
      counter_50M = counter_50M + 1;
   end
 else if (counter_50M ==25000000)
    begin
      clk_1Hz <= !clk_1Hz;
      counter_50M =0;
   end
end
```

```
//If "ld =1" we load the external data through D_in[3:0], if mode is active
// it will be counting up and if mode is inactive it will count down.

always @(posedge clk_1Hz, posedge clr)
  if(clr)
     count <=      ;
  else if(ld)
    count <=D_in ;
      else if (mode)
       count <=     ;
       else
       count <=     ;
```