Sequential Circuits

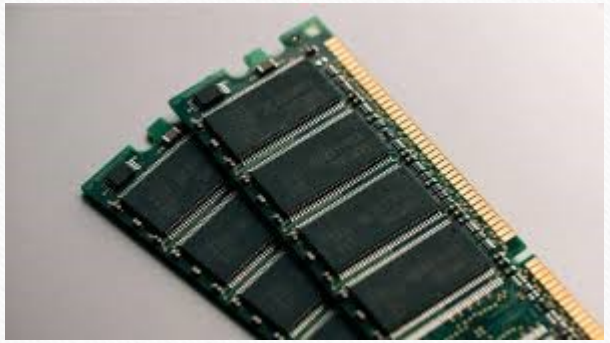
Verilog Implementation

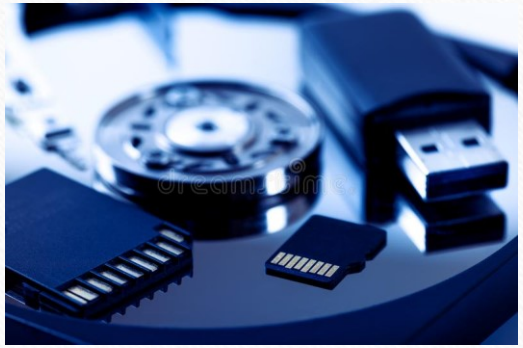
Outline

- Introduction
- Sequential logic vs. Combinational logic
- Latches
- D-flipflop implementation
- Edge triggering events
- Synchronous vs. Asynchronous resets
- Blocking vs. Non-blocking assignment
- Register and Counters
- UART Transmitter

Introduction

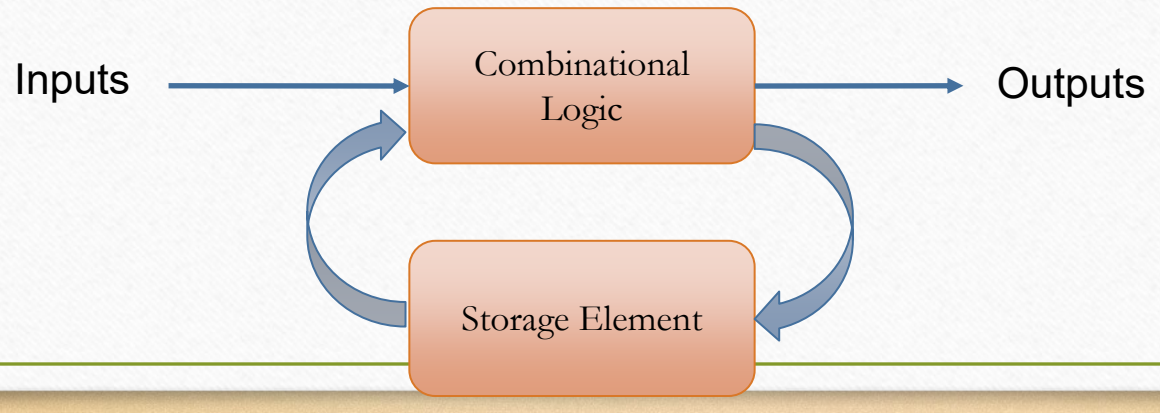
- All the electronic devices have the ability to send, receive, store, retrieve, and process information represented in a binary format.
- The technology enabling and supporting these devices is critically dependent on electronic components that can store information (i.e., have memory).





Sequential logic vs. Combinational logic

- A sequential circuit is a circuit with memory, which forms the internal state of the circuit.
- Unlike a combinational circuit, in which the output is a function of input only.
- The output of a sequential circuit is a function of the input and the internal state.



The always block

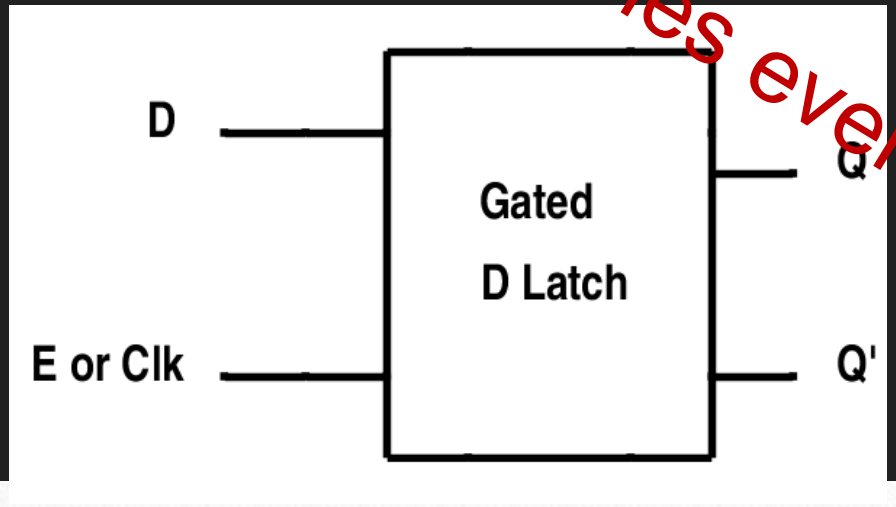
- **always**@(sensitivity list)
  - Triggers execution of the following code block
  - The statement in the brackets is called "Sensitivity List" which describes when the execution is triggered.
  - Allows us to use powerful statements inside it.

```
always@()
  begin
  ........
  statement;
  .........
  end
```

Latches (incomplete assignment ????)

- The first storage elements are latches.
- Behavior: whenever the clock is high Q changes to become equal to D, otherwise Q stays the same.

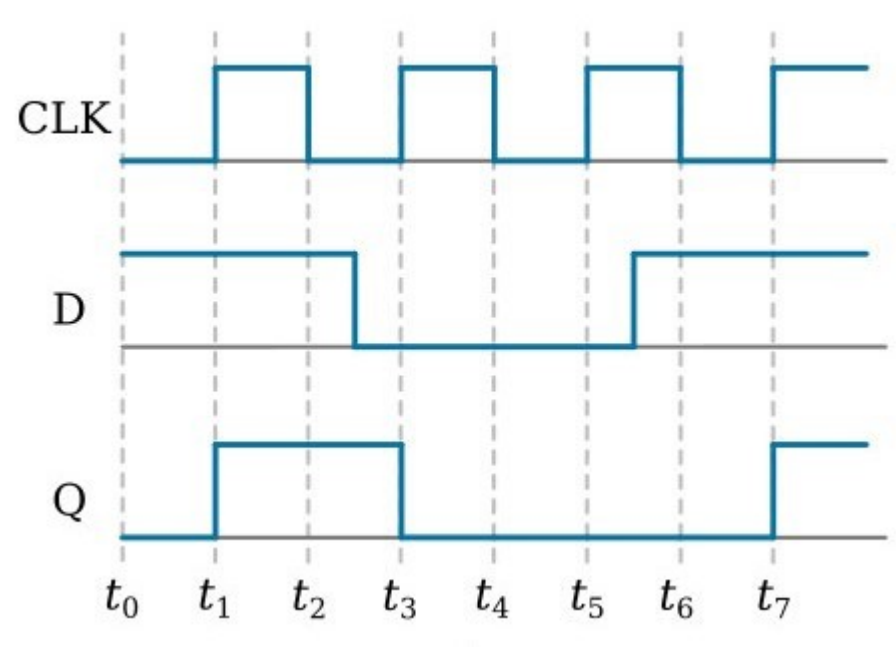
Don't use latches ever again




```
module latch(
    input D,clk,
    output Q_bar,
    out Q
);
  always(
  begin
    if(
      Q
  end
  assign Q_bar = ~Q;
endmodule
```

D-Flipflop

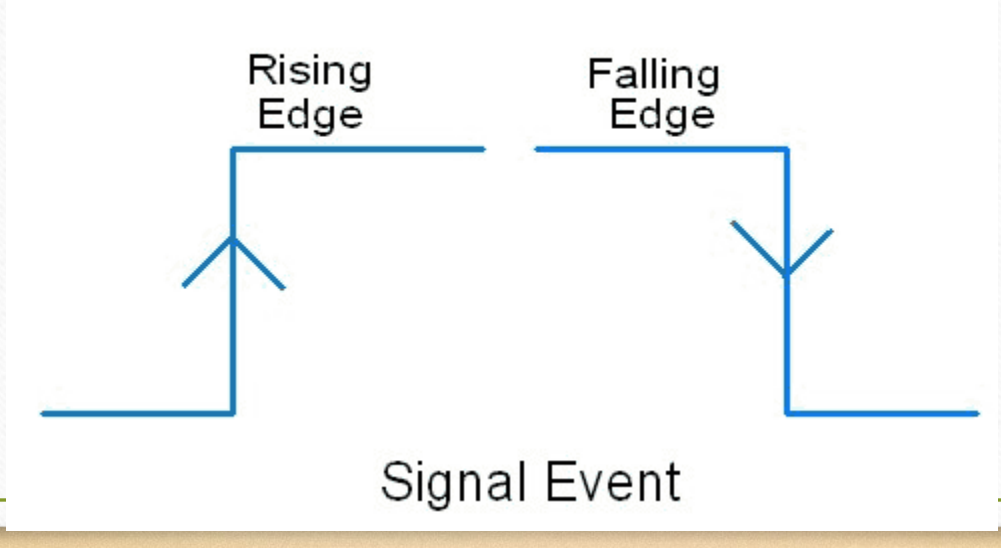
- The basic storage element in any sequential circuit.
- Behavior: for every positive edge of the clock Q changes to become equal to D.



CLK	D	Q
Rising edge	0	0
Rising edge	1	1
Non-rising	X	Q(prev)

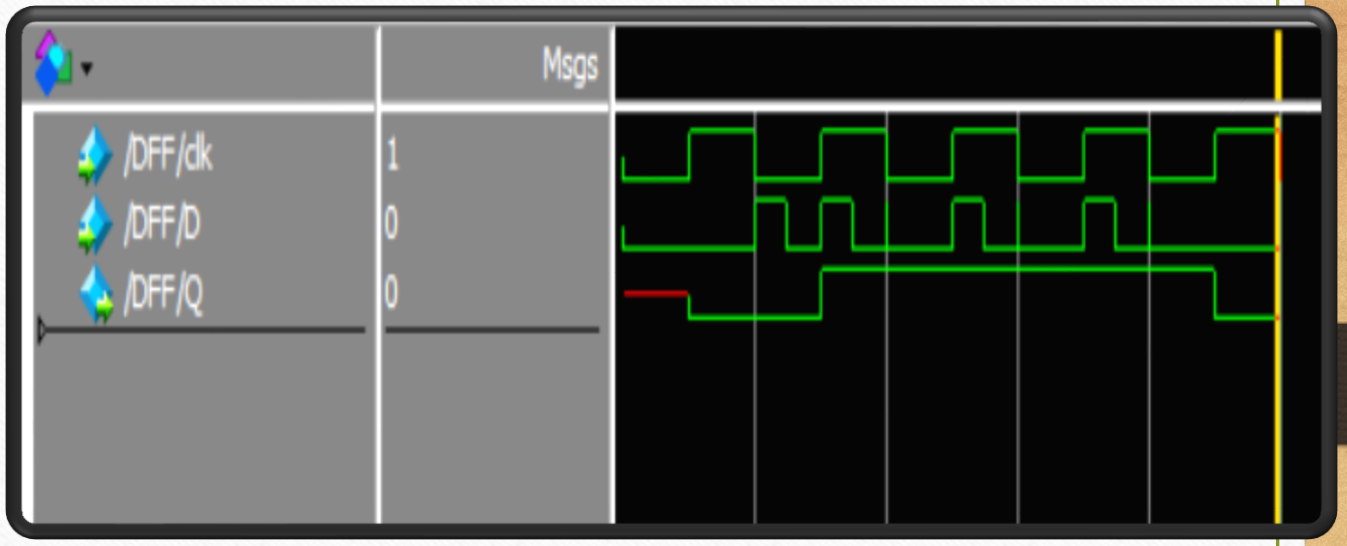
Edge triggering events

- **posedge** and **negedge**
- Both are keywords specify the direction of changing of the clock signal.
- They are used in the sensitivity list for sequential circuits.



D-Flipflop(implementation)

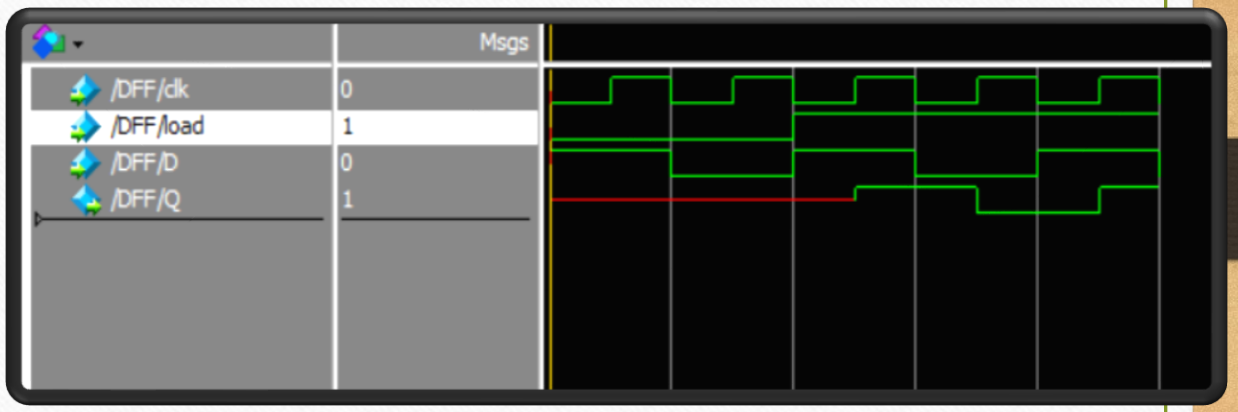
```
module DFF(
  input clk,D,
  output reg Q
);

  always@(posedge clk)
    Q = D;

endmodule
```



Note that:
In the start of the simulation Q value is unknown.

D-Flipflop with load (implementation)

```
module DFF(
  input clk,D,load,
  output reg Q
);


  always@(posedge clk)
    if(load)          //Incomplete assignment ???
      Q = D;


endmodule
```

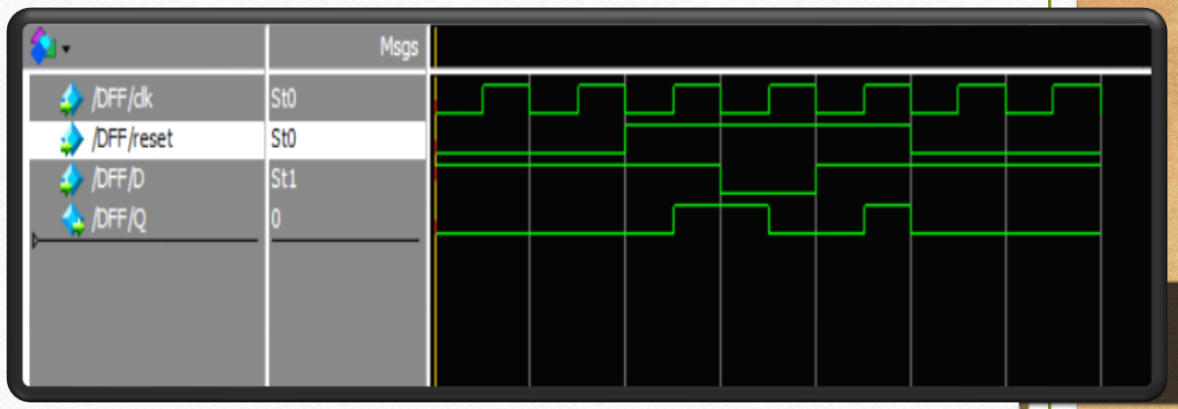


Synchronous vs. Asynchronous resets

- **Reset** is a signal that clears the D FF to zero.
- **Synchronous reset** is a signal that clears the D FF to zero and is controlled by the clock.
- **Asynchronous reset** is a signal that clears the D FF to zero and is not controlled by the clock.

D-Flipflop with Asynchronous reset(implementation)

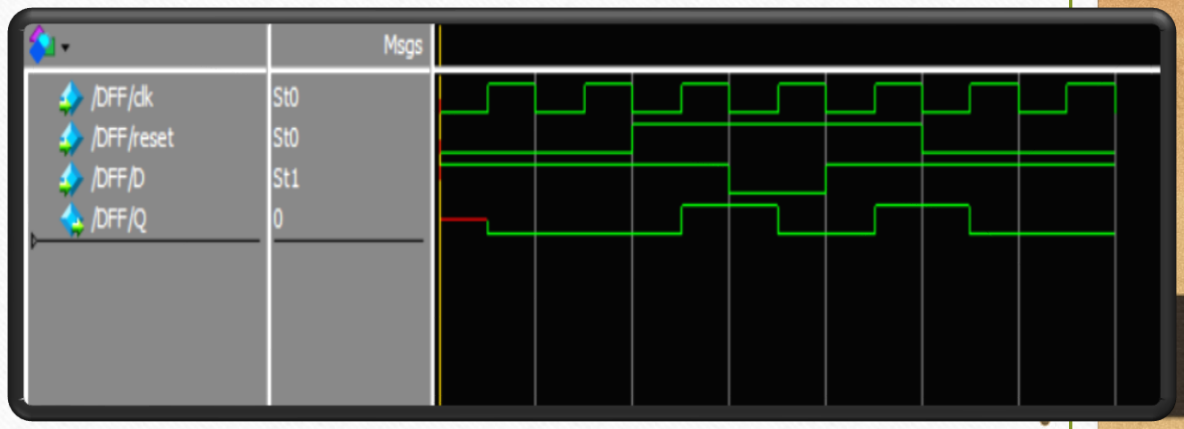
```
module DFF(
  input clk, D, reset,       //Asynchronous active low reset
  output reg Q
);

  always@(posedge clk or negedge reset)
  begin
    if(!reset)
     Q = 1'b0;
    else
     Q = D;
  end
endmodule
```



Note that:
Q changed regardless of the positive edge of the clock

D-Flipflop with Synchronous reset(implementation)

```
module DFF(
  input clk, D, reset,       //Synchronous active low reset
  output reg Q
);

  always@(posedge clk)
  begin
    if(!reset)
     Q = 1'b0;
    else
     Q = D;
  end
endmodule
```



Note the difference between this waveform and the previous one (same inputs different output).

Blocking vs. Non-blocking assignment

1. Blocking assignment [variable name] = [expression]
   - In a blocking assignment, the expression is evaluated and then assigned to the variable immediately before execution of the next statement.

2. Non-blocking assignment [variable name] <= [expression]
   - In a blocking assignment, the evaluated expression is assigned at the end of the always block.

Shift register ???

```
module SHreg(
  input clk,D,
  output reg a,b,c
);

  always@(posedge clk)
  begin
    a <= D;
    b <= a;
    c <= b;
  end
endmodule
```

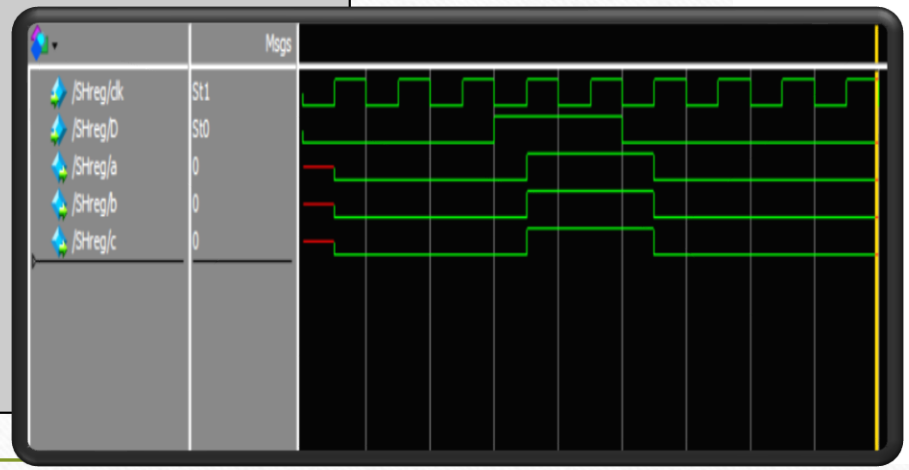
Is there a difference ?

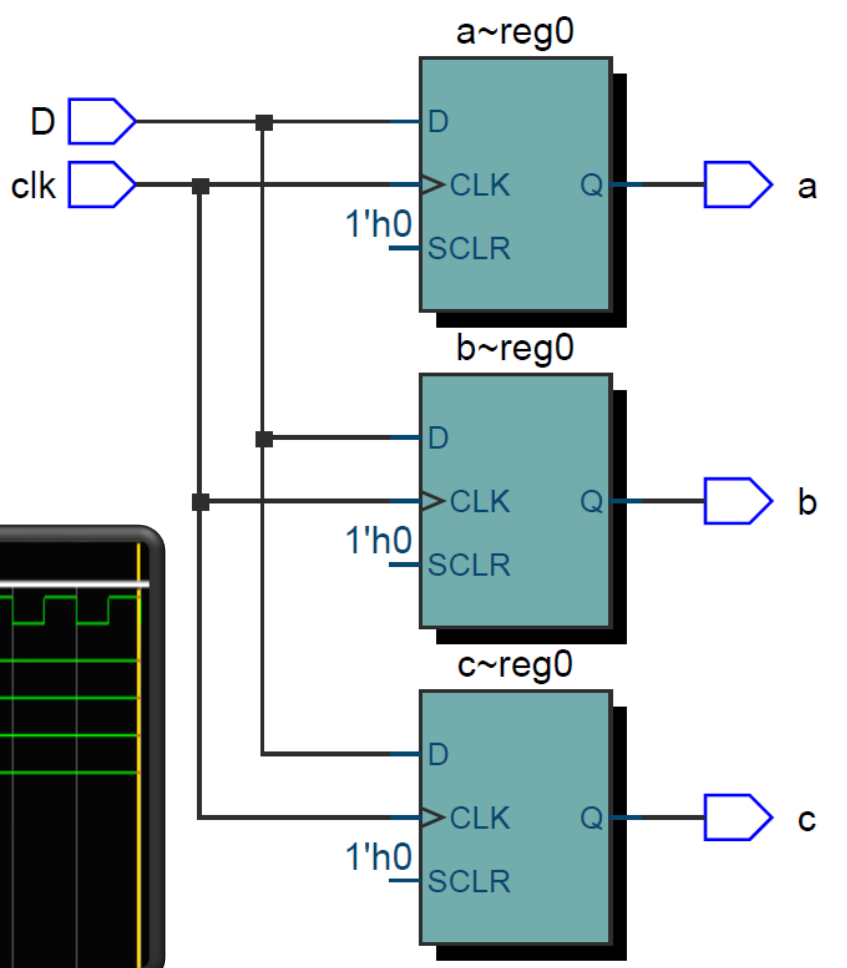
```
module SHreg(
  input clk,D,
  output reg a,b,c
);

  always@(posedge clk)
  begin
    a = D;
    b = a;
    c = b;
  end
endmodule
```

The first design

```
module SHreg(
  input clk,D,
  output reg a,b,c
);


  always@(posedge clk)
  begin
    a = D;
    b = a;
    c = b;
  end
endmodule
```

Is this what we want ?

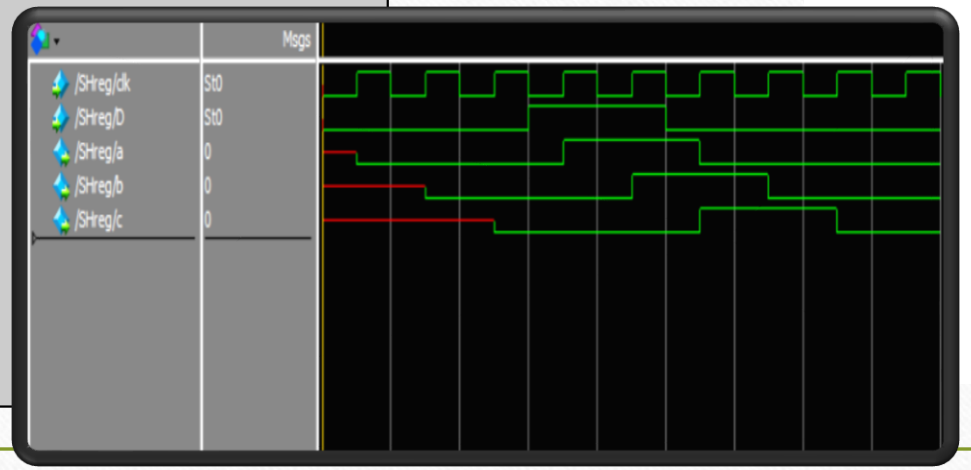


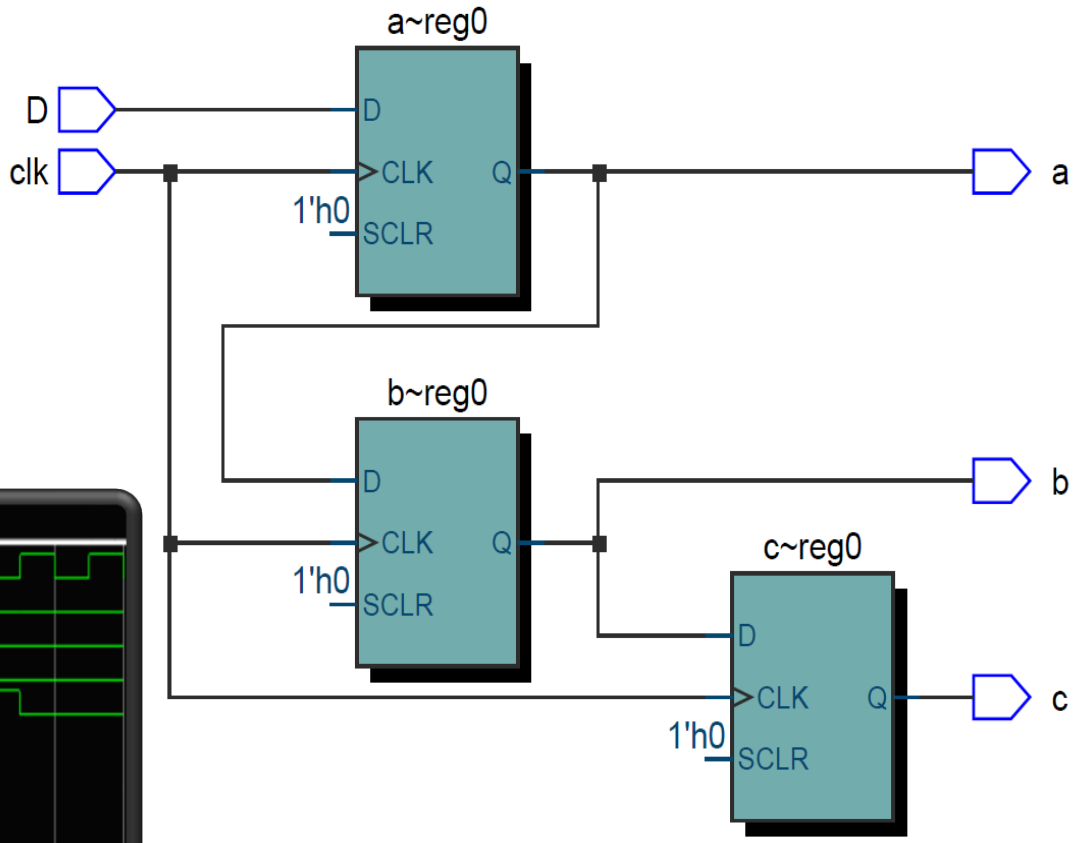


The second design

```
module SHreg(
  input clk,D,
  output reg a,b,c
);

  always@(posedge clk)
  begin
    a <= D;
    b <= a;
    c <= b;
  end
endmodule
```

Is this what we want ?





Simulation Races

- When describing flip-flops in separate always statements, as shown here. When a positive clock edge occurs in the simulation, both always statements will be executed. It is not possible to tell which one the simulator will execute first.

```
always@(posedge clk)
  k = l;

always@(posedge clk)
  p = k;
```

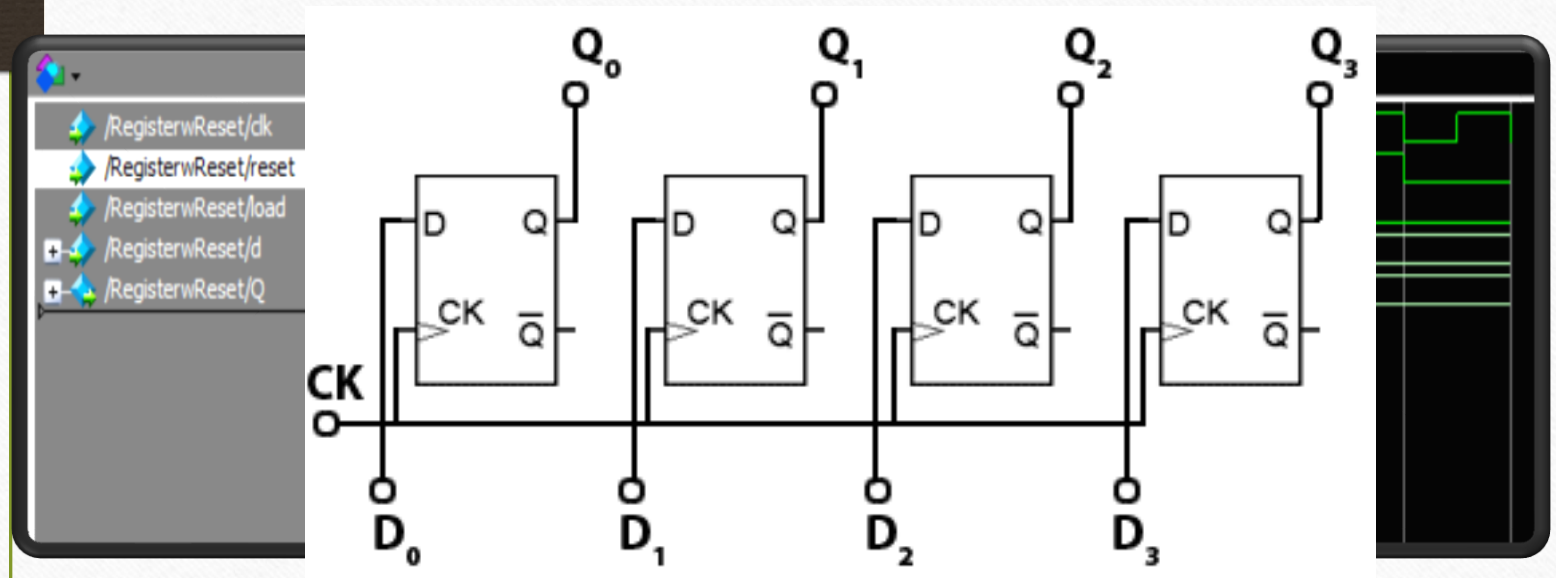
```
always@(posedge clk)
  k <= l;

always@(posedge clk)
  p <= k;
```

The design rules

- Always follow this rules
  - Use blocking assignment for the combinational circuits.
  - Use non-blocking assignment for the sequential circuits.
  - Only use edge-triggering flip-flops.
  - For your design use one clock source and only one edge of that clock.

Register implementation

- A register is a collection of D-FFs that are controlled by the same clock and reset signals.




```
module RegisterwReset(
  input clk,load,reset,
  input [15:0] d,
  output reg [15:0] Q
);

  always@(posedge clk)
  begin
    if(reset)
      Q <= 16'h0000;
    else if(load)
      Q <= d;
  end
endmodule
```

Synchronous Counter

- A counter is essentially a register that goes through a predetermined sequence of binary states.
- The sequence of states may follow the binary number sequence or any other sequence of states.
- A counter that follows the binary number sequence is called a binary counter .
- An n -bit binary counter consists of n flip-flops and can count in binary from 0 through 2^n - 1.

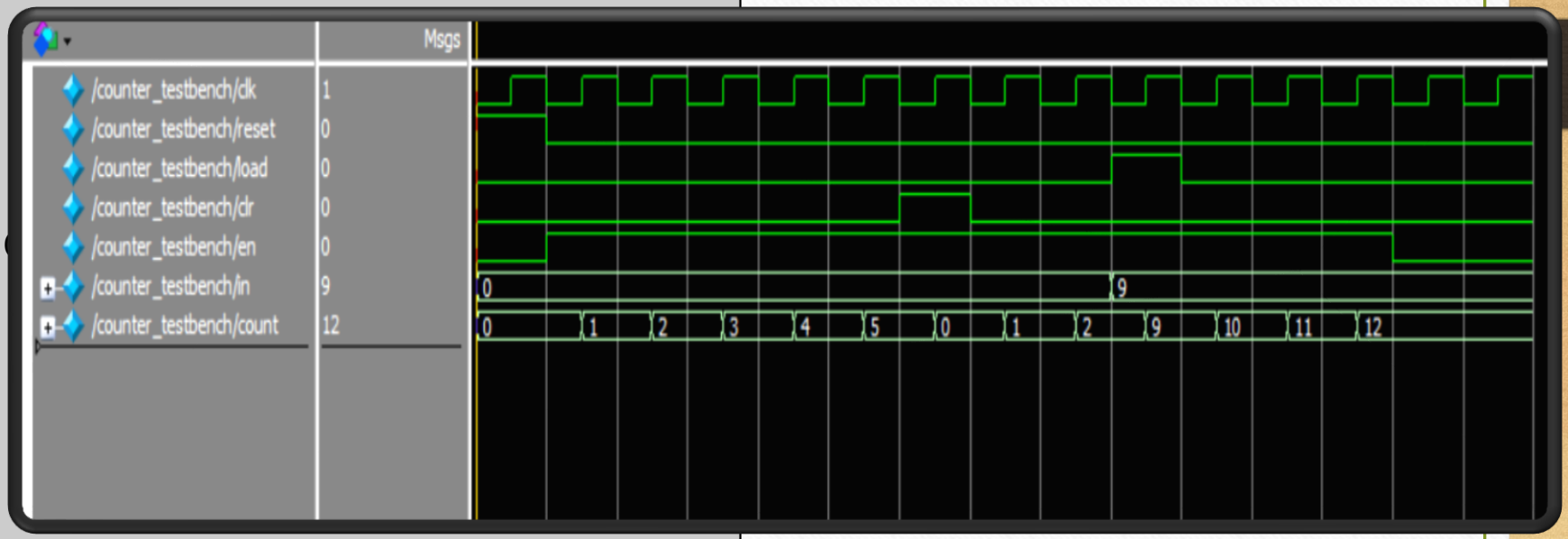
Synchronous Counter(Implementation)

```
module bin_counter #(parameter N = 8)(
  input clk,reset,
  input clr,load,en,
  input [N-1:0] in,
  output reg [N-1:0] count
);

  always@(posedge clk or posedge reset)
  begin
    if(reset)
      count <= 8'h00;
    else if(load)
      count <= in;
    else if(clr)
      count <= 8'h00;
    else if(en)
      count <= count + 1'b1;
  end
endmodule
```

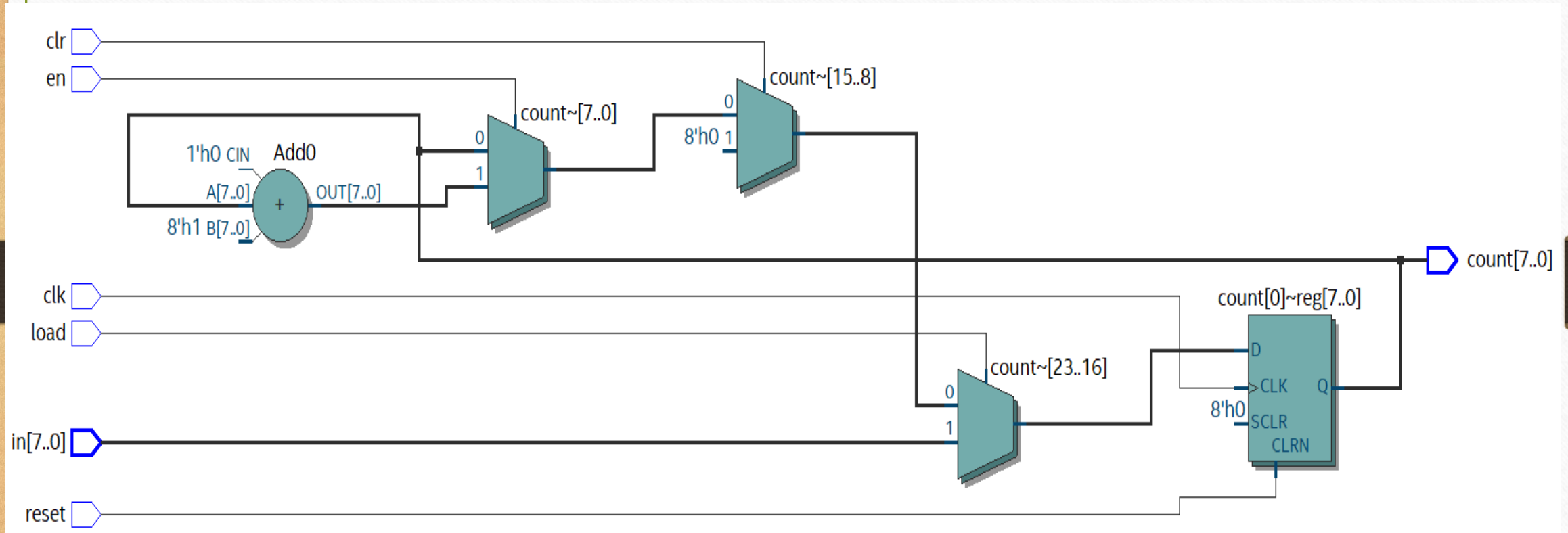
```
`timescale 1 ns/10 ps
module counter_testbench();
reg clk, reset, en, load, clr;
reg [7:0] in;
wire [7:0] count;
bin_counter u1
(.clk(clk),.reset(reset),.load(load),.en(en),.clr(clr),.in(in),.count(count));

always
  #10 clk = ~clk;

initial
begin
$monitor("The count is %d",count);
reset = 1; en = 0; load = 0; clr = 0; clk = 0; in = 0;
#20  reset = 0;  en = 1;
#100 clr = 1;
#20 clr = 0;
#40 load = 1; in = 8'h09;
#20 load = 0;
#60 en = 0;
#40
$stop;
end
endmodule
```

Synchronous Counter

```
`timescale 1 ns/10 ps
module counter_testbench();
reg clk, reset, en, load, clr;
reg [7:0] in;
wire [7:0] count;
bin_counter u1 (.clk(clk),.reset(reset),.load(load),.en(en),.clr(clr),.in(in),.count(count));

always
  #10 clk = ~clk;

initial
begin
$monitor("The count is %d",count);
reset = 1; en = 0; load = 0; clr = 0; clk =
#20  reset = 0;  en = 1;
#100 clr = 1;
#20 clr = 0;
#40 load = 1; in = 8'h09;
#20 load = 0;
#60 en = 0;
#40
$stop;
end
endmodule
```



Synchronous Counter

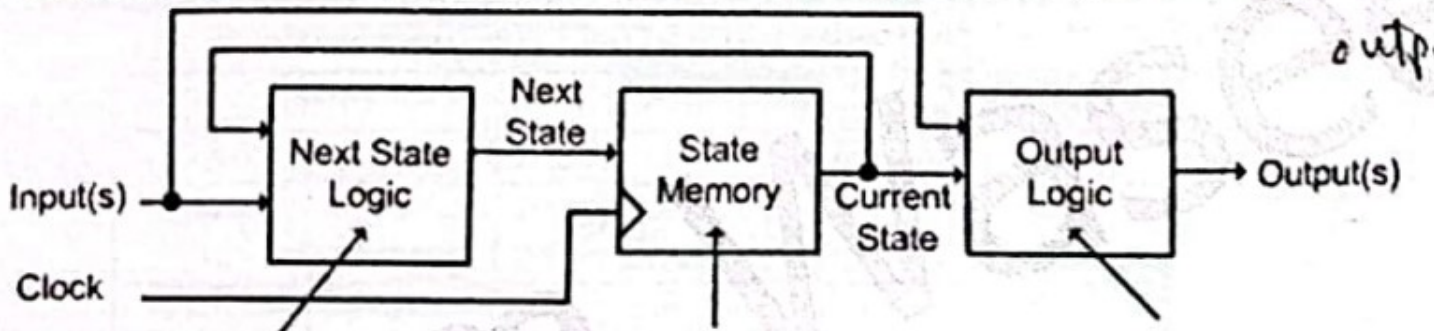


FSM

- Sequential logic circuit that makes decisions on its outputs based upon prior states and inputs
- It has a finite number of states which is the case for digital since we have a finite number of digits but in analog, we have finite numbers.
- FSM has the power to trace the history of events to reach a specific state
- BY knowing the history and the current inputs, an intelligent and complex system can be designed.
- Outputs do not only check for the instantaneous input available but the history of inputs seen in the prior states.

Main Components of a Finite State Machine

Mealy Machine – The output(s) depend on both the current state and system input(s).

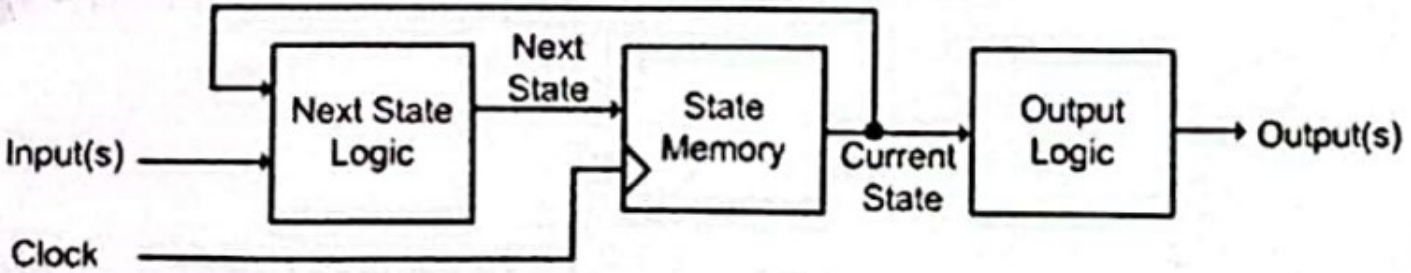


The next state logic creates the signal "next state" based on the current state and any system inputs. This block is implemented with combinational logic.

The state memory holds the current state. The current state is updated with "next state" on the rising edge of the clock. This block is implemented with D-Flip-Flops.

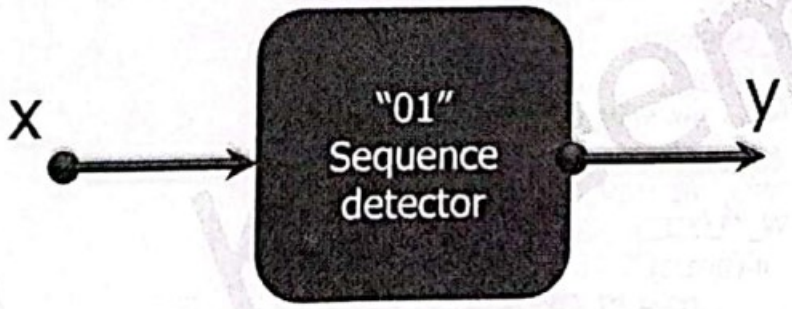
The output logic creates the system outputs. The output logic always depends on the current state of the machine and optionally (Mealy vs. Moore) the inputs of the system.

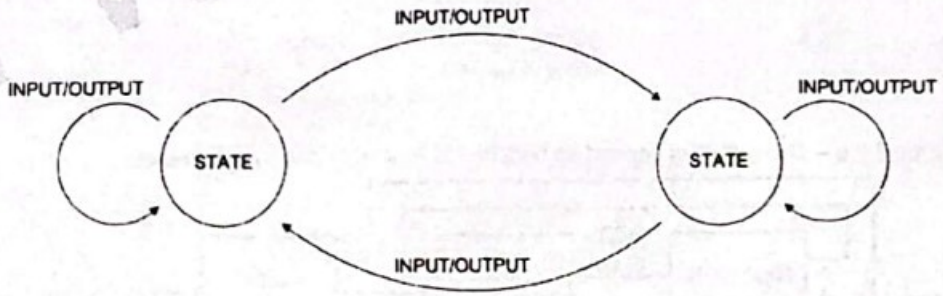
Moore Machine – The output(s) depends only on the current state.



EX: POSITIVE EDGE DETECTOR

- Design a circuit that asserts its output for one cycle when the input bit stream changes from 0 to 1 using Mealy State Machine
- Example
  - X = 00101101100
  - Y = 00101001000





GENERIC MEALY STATE MACHINE

THANK YOU!