HDL 2 - Sequential Circuit

Nahin Ul Sadad/Farjana Parvin
Lecturer
CSE, RUET

Sequential Circuit

Sequential Circuit:
Sequential circuit acts as memory. It depends of present inputs and previous outputs. It usually changes its output only when clock is active. Applications of Sequential Circuit: Register Implementation, RAM Implementation etc.

Synchronous sequential circuit uses **clock** to determine when to update contents of memory.

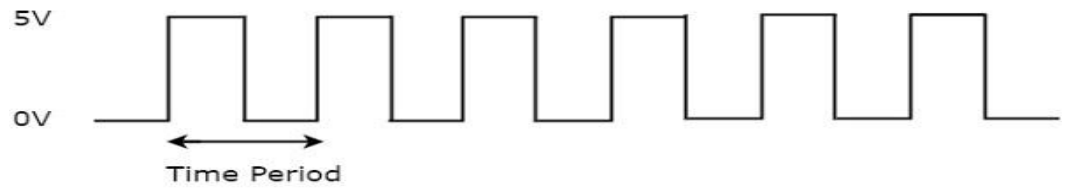


Fig (a) Clock signal

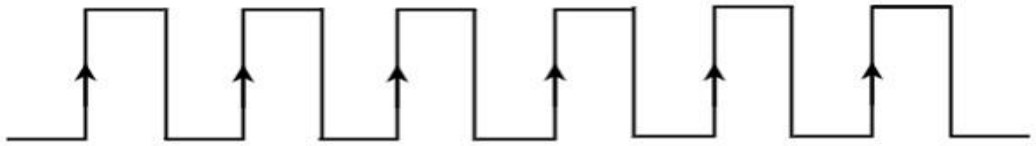


Fig (b) Positive Edge triggered Clock signal

1. Memory contents will be updated when clock signal is on positive edge.
2. Memory contents will remain same when clock signal is not on positive edge.

Sequential Circuit

- **Example:** D Flip-flop

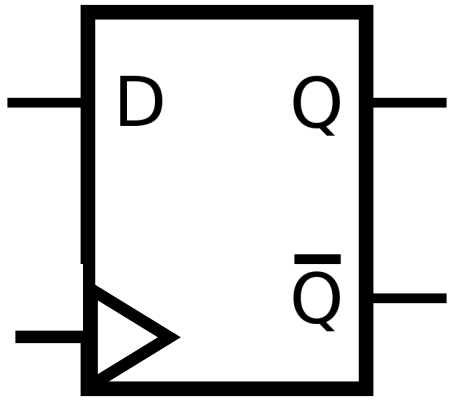


Fig (a) D Flip-flop Symbol

D	Q(t + 1)
0	0
1	1

Fig (b) D Flip-flop/D Latch truth table

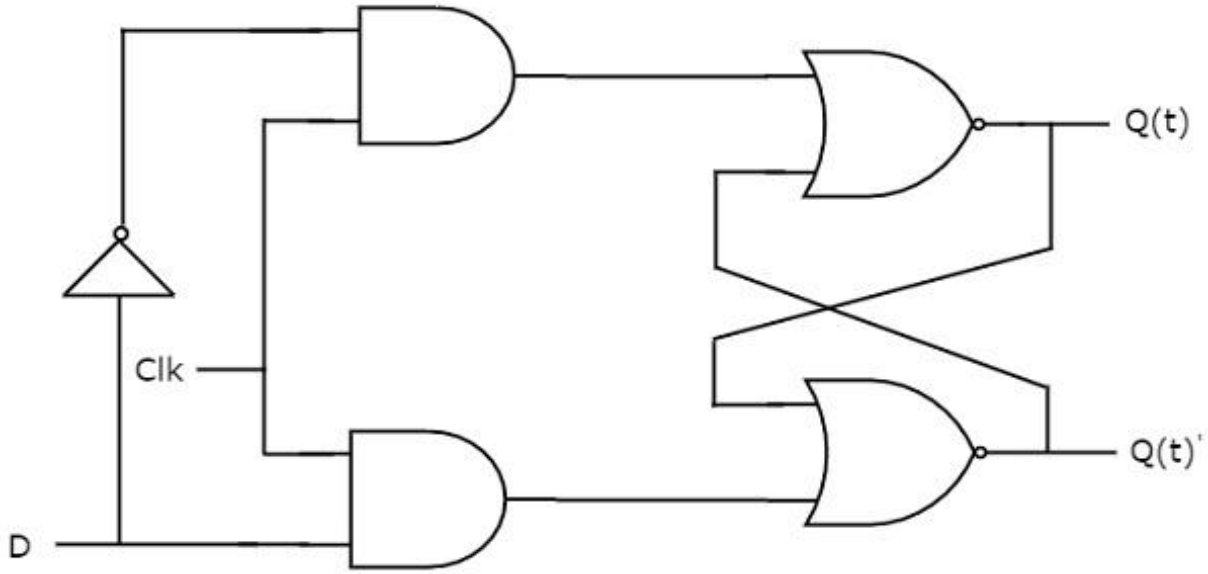


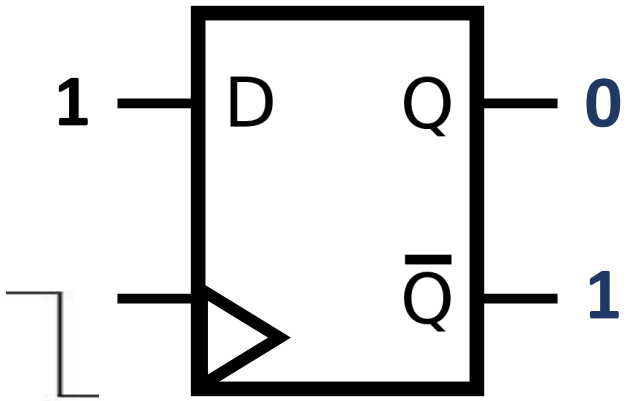
Fig (c) D Flip-flop implementation with logic gates
(Clock Driven and Synchronous)

Sequential Circuit

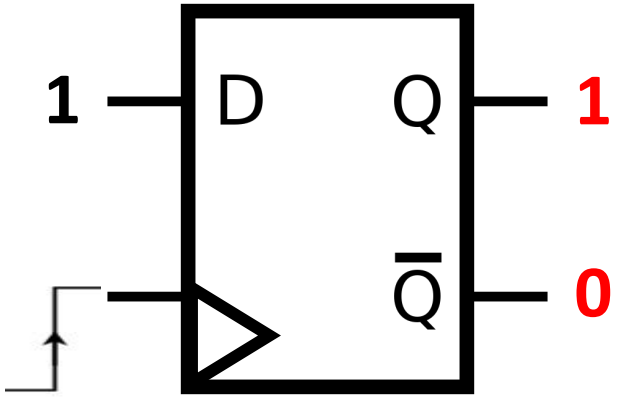
- **Example:** D Flip-flop

D	Q(t + 1)
0	0
1	1

Fig (b) D Flip-flop truth table



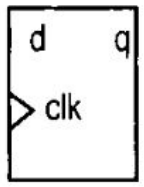
D Flip-flop will not update its content because clock is not on positive edge.



D Flip-flop will update its content because clock is on positive edge.

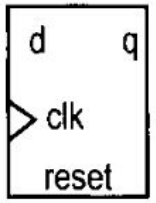
Sequential Circuit

- **Example:** D Flip-flop



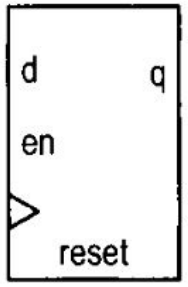
clk	q*
0	q
1	q
↓	d

(a) D FF



reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↓	d

(b) D FF with asynchronous reset



reset	clk	en	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	↓	0	q
0	↓	1	d

(c) D FF with synchronous enable

Figure: Block diagram and functional table of a D FF.

Sequential Circuit
Synchronous Design Methodology

The **Synchronous Design Methodology** is the most commonly used practice in designing a sequential circuit. In this methodology, all storage elements are controlled (i.e., synchronized) by a global clock signal and the data is sampled and stored at the rising or falling edge of the clock signal.

It allows designers to separate the storage components from the circuit and greatly simplifies the development process. This methodology is the most important principle in developing a large, complex digital system and is the foundation of most synthesis, verification, and testing algorithms.

Sequential Circuit
Synchronous Design Methodology

The block diagram of a synchronous system is shown in figure. It consists of the following parts:

1. **State Register:** A collection of D FFs controlled by the same clock signal.
2. **Next-State Logic:** Combinational logic that uses the external input and internal state (i.e., the output of register) to determine the new value of the register.
3. **Output Logic:** Combinational logic that generates the output signal.

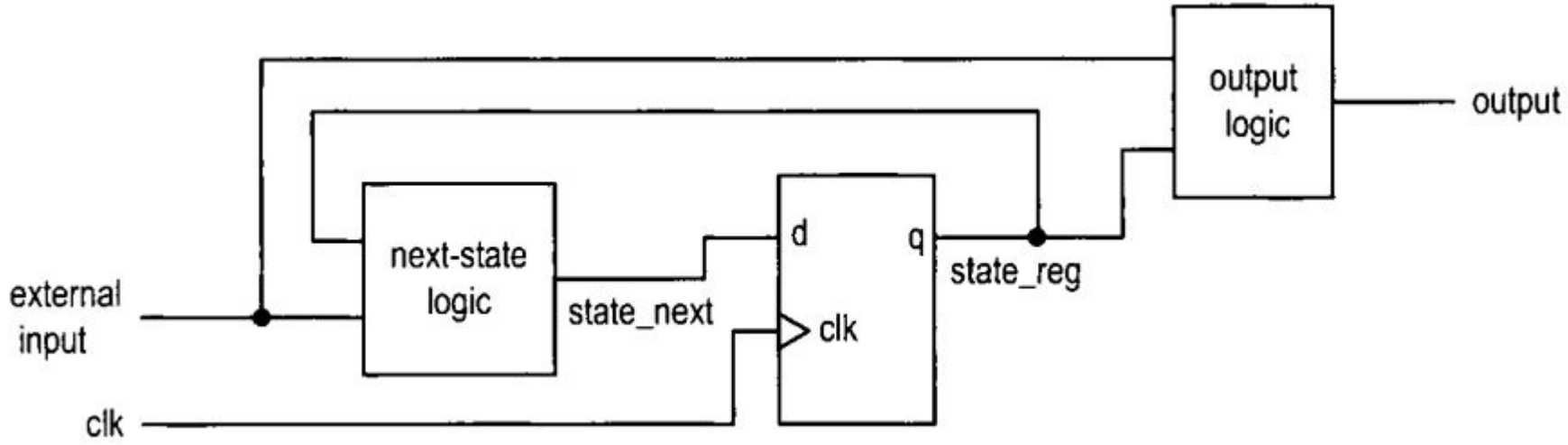


Figure: Block diagram of a synchronous system.

Sequential Circuit
D FF without asynchronous reset

d_ff.v

```
module d_ff
(
    input wire clk,
    input wire d,
    output reg q
);

always @(posedge clk) //posedge means positive edge
begin
    q <= d;
end

endmodule
```

Sequential Circuit
D FF without asynchronous reset

d_ff_tb.v

```
`timescale 1ns/1ns

module d_ff_tb;
    reg clk;
    reg d;
    wire q;

d_ff circuit1 (clk, d, q);

always begin
    clk = ~clk;
    #10;
end
```

```
initial begin
    $dumpfile("test.vcd");
    $dumpvars(0, d_ff_tb);

    clk <= 0;
    d <= 0;
    #20;

    d <= 1;
    #20;

    $finish;
end
```

```
initial begin
    $monitor("clk = %b, d =
      %b, q = %b", clk, d, q);
end


endmodule
```

Sequential Circuit
D FF with asynchronous reset

d_ff_reset.v

```
module d_ff_reset
(
    input wire clk,
    input wire reset,
    input wire d,
    output reg q
);

always @(posedge clk, posedge reset)
begin
    if (reset)
        q <= 1'b0;
    else
        q <= d;
end
endmodule
```

Sequential Circuit
D FF with asynchronous reset

d_ff_reset_tb.v

```
`timescale 1ns/1ns

module d_ff_reset_tb;
    reg clk;
    reg reset;
    reg d;
    wire q;

d_ff_reset circuit1
        (clk, reset, d, q);

always begin
    clk = ~clk;
    #10;
end
```

```
initial begin
$dumpfile("test.vcd");
$dumpvars(0,d_ff_reset_tb);

    clk <= 0;
    reset <= 1;
    d <= 0;
    #20;

    reset <= 0;
    d <= 1;
    #20;

    $finish;
end
```

```
initial begin
    $monitor("clk = %b,
     reset = %b, d = %b, q =
     %b", clk, reset, d, q);
end


endmodule
```

Sequential Circuit
D FF with synchronous enable

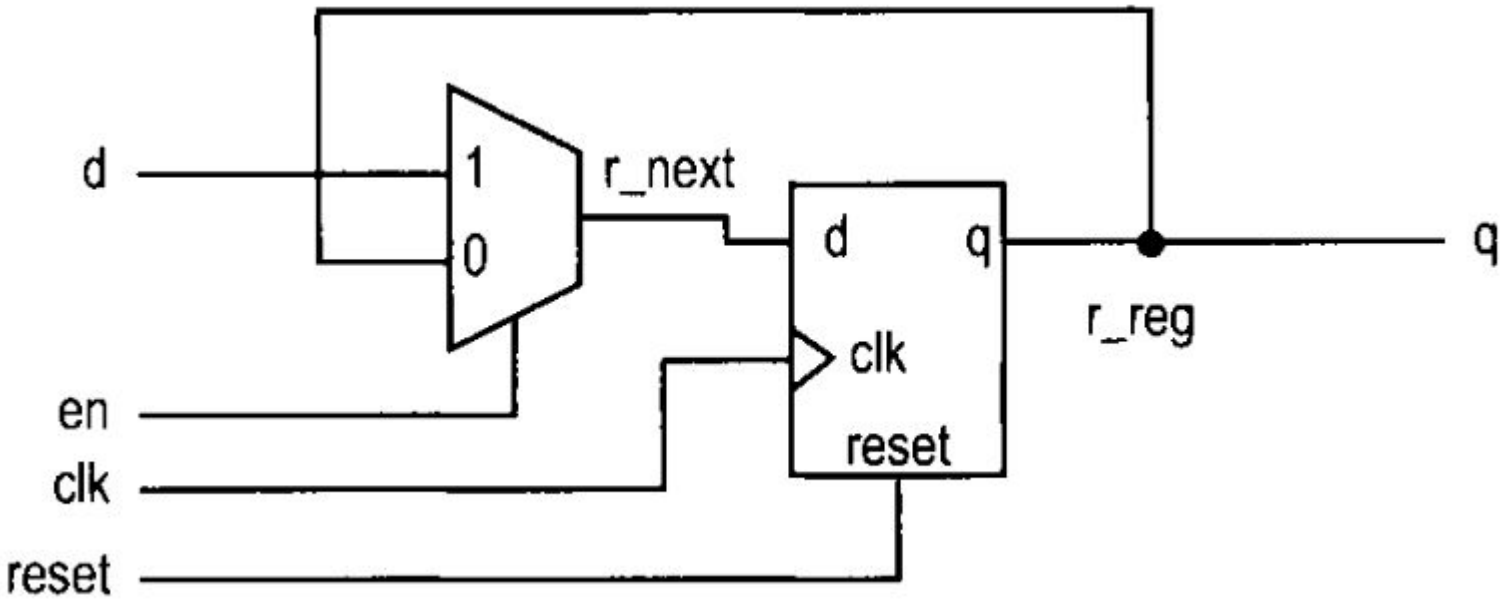


Figure: D FF with synchronous enable

Sequential Circuit
D FF with synchronous enable

d_ff_reset_en.v

```
module d_ff_reset_en
(
    input wire clk,
    input wire reset,
    input wire en,
    input wire d,
    output reg q
);

always @(posedge clk, posedge reset)
begin
    if (reset)
        q <= 1'b0;
    else if (en)
        q <= d;
end
endmodule
```

Sequential Circuit
D FF with synchronous enable

d_ff_reset_en_tb.v

```
`timescale 1ns/1ns

module d_ff_reset_en_tb;
    reg clk;
    reg reset;
    reg en;
    reg d;
    wire q;

d_ff_reset_en circuit1
    (clk, reset, en, d, q);

always begin
    clk = ~clk;
    #10;
end
```

```
initial begin
    $dumpfile("test.vcd");
    $dumpvars(0,
          d_ff_reset_en_tb);

    clk <= 0;
    reset <= 1;
    en <= 0;
    d <= 0;
    #20;

    reset <= 0;
    en <= 1;
    d <= 1;
    #20;

    $finish;
end
```

```
initial begin
   $monitor("clk = %b,
       reset = %b, en = %b,
       d = %b, q = %b", clk,
           reset, en, d, q);
end


endmodule
```

Sequential Circuit
D FF using Synchronous Design Methodology

d_ff_reset_en2.v

```
module d_ff_reset_en2
(
    input wire clk,
    input wire reset,
    input wire en,
    input wire d,
    output reg q
);

   reg r_next, r_reg;

   //memory/register/d_ff
   always @(posedge clk, posedge reset)
   begin
       if (reset)
           r_reg <= 1'b0;
       else
           r_reg <= r_next;
   end
```

Sequential Circuit
D FF using Synchronous Design Methodology

d_ff_reset_en2.v

```
    //next state logic
    always @(*)
    begin
         if (en)
             r_next = d;
         else
             r_next = r_reg;
    end

    //output logic
    always @(*)
    begin
         q = r_reg;
    end
endmodule
```

Sequential Circuit
7 bit Register

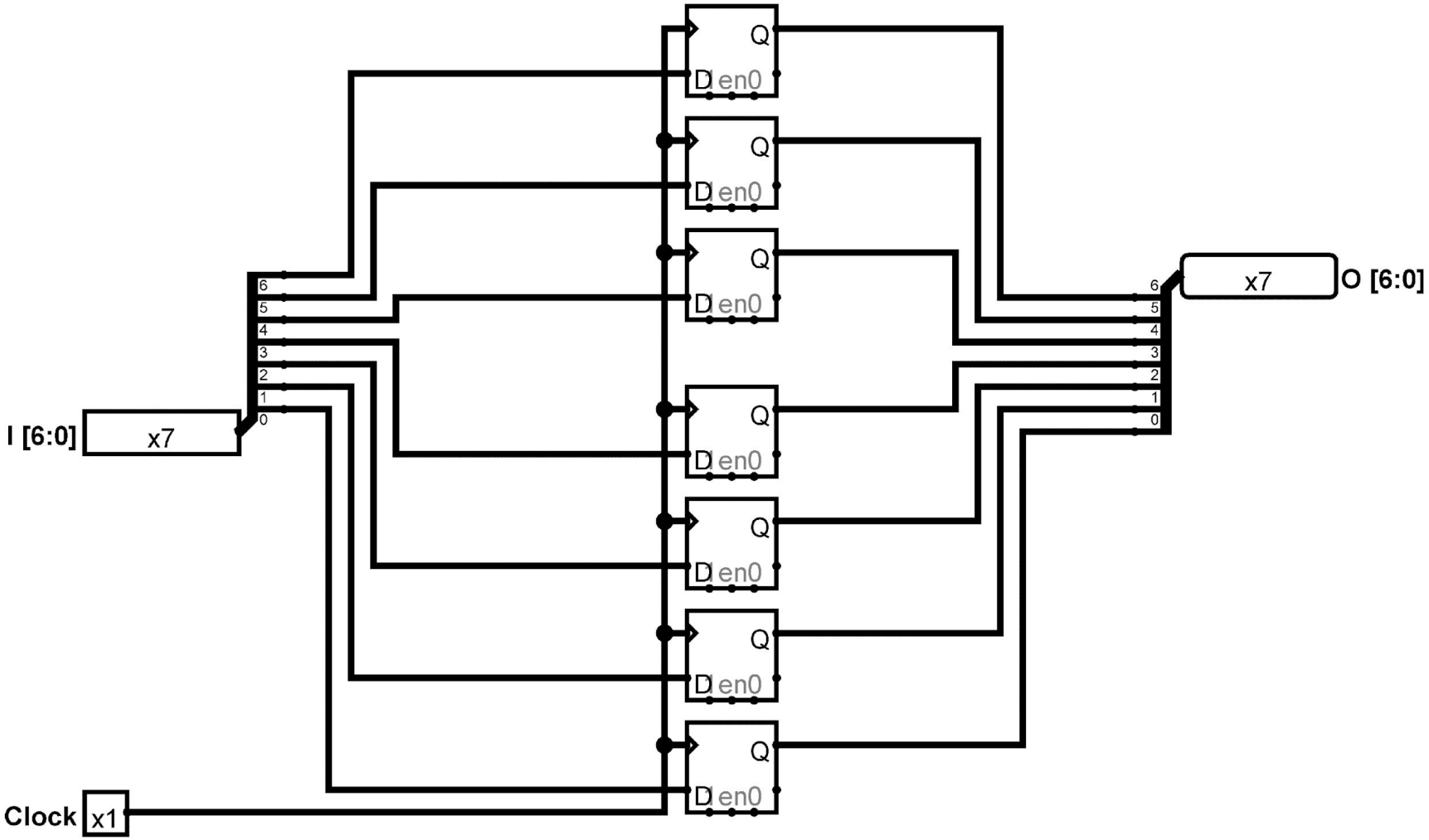


Figure: 7 bit Register

Sequential Circuit
7 bit Register

register.v

```
module register
(
    input wire clk,
    input wire reset,
    input wire [6:0] d,
    output reg [6:0] q
);

always @(posedge clk, posedge reset)
begin
    if (reset)
        q <= 7'b0000_000;
    else
        q <= d;
end

endmodule
```

Sequential Circuit
7 bit Register

register_tb.v

```
`timescale 1ns/1ns

module register_tb;
    reg clk;
    reg reset;
    reg [6:0] d;
    wire [6:0] q;

register circuit1 (clk,
                reset, d, q);

always begin
    clk = ~clk;
    #10;
end
```

```
initial begin
    $dumpfile("test.vcd");
    $dumpvars(0,
                register_tb);

    clk <= 0;
    reset <= 1;
    d <= 7'b0000_000;
    #20;

    reset <= 0;
    d <= 7'b0000_111;
    #20;

    $finish;
end
```

```
initial begin
    $monitor("clk = %b,
     reset = %b, d = %b, q =
     %b", clk, reset, d, q);
end


endmodule
```

Sequential Circuit
7 bit Register using Synchronous Design Methodology

register2.v

```
module register
(
    input wire clk,
    input wire reset,
    input wire [6:0] d,
    output wire [6:0] q
);

//signal declaration
reg [6:0] q_reg ;
wire [6:0] q_next ;
```

```
// body or memory or state register
always @(posedge clk, posedge reset)
begin
    if (reset)
        q_reg <= 7'b0000_000;
    else
        q_reg <= q_next;
end

//next state logic
assign q_next = d;

//output logic
assign q = q_reg;

endmodule
```

Sequential Circuit
7 bit Up Counter

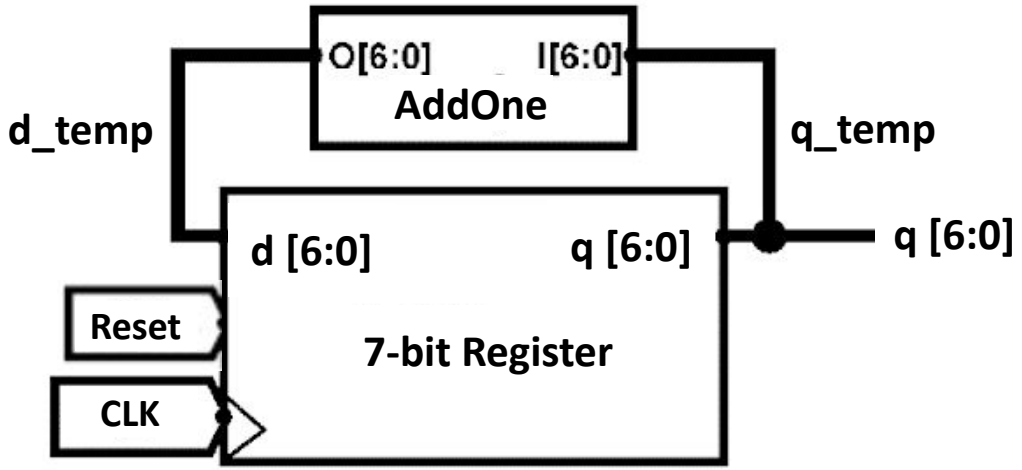


Figure: 7-bit Up Counter

Sequential Circuit
7 bit Up Counter

adder.v

```
module addOne
(
    input wire [6:0] I,
    output wire [6:0] O
);

assign O = I + 1;

endmodule
```

Sequential Circuit
7 bit Up Counter

counter.v

```
module counter
(
    input wire CLK,
    input wire Reset,
    output wire [6:0] q
);

    wire [6:0] d_temp, q_temp;
    register register_circuit1 (CLK, Reset, d_temp, q_temp);
    addOne adder_circuit1 (q_temp, d_temp);
    assign q = q_temp;

endmodule
```

Sequential Circuit
7 bit Up Counter

register_tb.v

```
`timescale 1ns/1ns

module counter_tb;
    reg clk;
    reg reset;
    wire [6:0] q;

counter counter_circuit1
          (clk, reset, q);

always begin
    clk = ~clk;
    #10;
end
```

```
initial begin
    $dumpfile("test.vcd");
    $dumpvars(0, counter_tb);

    clk <= 0;
    reset <= 1;
    #20;

    reset <= 0;
    #20;
    #20;
    #20;
    #20;
    $finish;
end
```

```
initial begin
    $monitor("clk = %b,
        reset = %b, q = %b",
            clk, reset, q);
end


endmodule
```

```
iverilog -o output counter.v counter_tb.v register.v adder.v
vvp output
gtkwave test.vcd &
```

Sequential Circuit
Types of Sequential Circuit

Based on the characteristics of the next-state logic, we divide sequential circuits into three categories:

1. **Regular Sequential Circuit:** The state transitions in the circuit exhibit a "regular" pattern, as in a counter or shift register. The next-state logic is constructed primarily by a predesigned, "regular" component, such as an incrementor or shifter.

2. **FSM:** The state transitions in the circuit do not exhibit a simple, repetitive pattern. The next-state logic is constructed by "random logic" and synthesized from scratch. It should be called a random sequential circuit, but is commonly known as an FSM (finite state machine).

3. **FSMD:** The circuit consists of a regular sequential circuit and an FSM. The two parts are known as a data path and a controlpath, and the complete circuit is known as an FSMD (FSM with data path). This type of circuit is used to implement an algorithm represented by register-transfer (RT) methodology, which describes system operation by a sequence of data transfers and manipulations among registers.

Exercises

1. Implement following:
    a. 16 bit Register
    b. 7 bit Register with synchronous enable input
    c. 4 bit down counter
    d. 4 bit down counter using synchronous design methodology
    e. 4 bit down counter using only one module

    in Verilog HDL along with a test bench.

Thank You ☺️