EEE 2243
Digital System Design
Semester II 2011/12

Tutorial 2

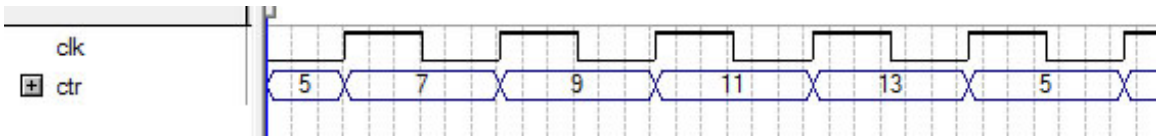
Verilog Sequential

1. Write the Verilog code for the following counters by behavioral approach:

   a. $5 \rightarrow 7 \rightarrow 9 \rightarrow 11 \rightarrow 13$ repeat

```
module Counter1a(clk, ctr);

input clk;
output [3:0] ctr;
reg [3:0] ctr;

initial ctr <= 5;

always@(posedge clk)
begin
   if (ctr == 13)
      ctr <= 5;
   else
      ctr <= ctr+2;
end

endmodule
```

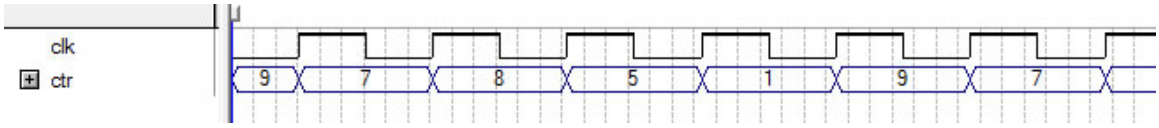
Simulation:



b. 9 → 7 → 8 → 5 → 1 repeat

```
module Counter1b(clk, ctr);

input clk;
output [3:0] ctr;
reg [3:0] ctr;

initial ctr <= 9;

always@(posedge clk)
case (ctr)
   9: ctr <= 7;
   7: ctr <= 8;
   8: ctr <= 5;
   5: ctr <= 1;
   1: ctr <= 9;
endcase

endmodule
```

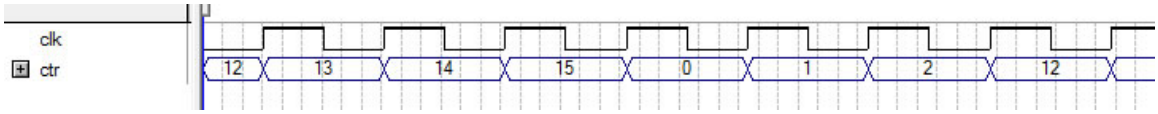
Simulation:



c. 4 bit mod 6 start at 12

```
module Counter1c(clk, ctr);

input clk;
output [3:0] ctr;
reg [3:0] ctr;

initial ctr <= 12;

always@(posedge clk)
if (ctr == 2)
   ctr <= 12;
else
   ctr <= ctr+1;

endmodule
```

Simulation:



d. 4 bit mod 6 start at 13 increment by 2

```
module Counter1d(clk, ctr);

input clk;
output [3:0] ctr;
reg [3:0] ctr;

initial ctr <= 13;

always@(posedge clk)
if (ctr == 7)
   ctr <= 13;
else
   ctr <= ctr+2;

endmodule
```

Simulation:

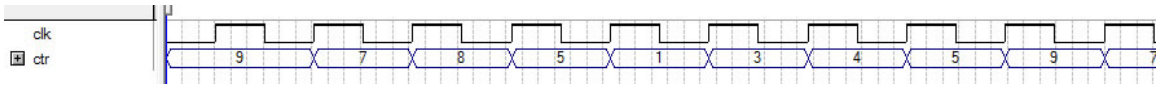
e. 9 → 7 → 8 → 5 → 1 → 3 → 4 → 5 repeat [This might be a difficult one]

This question might be misleading. The values that are written in the state transition list above are supposed to be the output of the FSM at each state, not the state value itself. In this case the first output 5 and the second output 5 are not from the same state; only the the output is the same which is 5. The total number of states is then 8 that we can number consecutively. The output assignment then can be set separately.

```
module Counter1e(clk, ctr);

input clk;
output [3:0] ctr;
reg [3:0] ctr;
reg [2:0] state;

initial
begin
   state <= 0;
   ctr <= 9;
end
```

```
always@(posedge clk)
begin
   state <= state+1;
   case (state)
      0: ctr <= 9;
      1: ctr <= 7;
      2: ctr <= 8;
      3: ctr <= 5;
      4: ctr <= 1;
      5: ctr <= 3;
      6: ctr <= 4;
      7: ctr <= 5;
   endcase
end

endmodule
```

Simulation:



Simulate each and every one of them.

2. Write the Verilog code for 8 bit Johnson and ring counter using Boolean expression and behavioral approach.

```
module JohnsonBoolean(clk, state);

input clk;
output [7:0] state;
reg [7:0] state;

initial state <= 0;

always@(posedge clk)
begin
   state[0] <=  state[1];
   state[1] <=  state[2];
   state[2] <=  state[3];
   state[3] <=  state[4];
   state[4] <=  state[5];
   state[5] <=  state[6];
   state[6] <=  state[7];
   state[7] <= ~state[0];
end

endmodule
```

```
module JohnsonBehavioral(clk, state);

input clk;
output [7:0] state;
reg [7:0] state;

initial state <= 0;

always@(posedge clk)
begin
   case (state)
   4'b00000000: state <= 4'b10000000;
   4'b10000000: state <= 4'b11000000;
   4'b11000000: state <= 4'b11100000;
   4'b11100000: state <= 4'b11110000;
   4'b11110000: state <= 4'b11111000;
   4'b11111000: state <= 4'b11111100;
   4'b11111100: state <= 4'b11111110;
   4'b11111110: state <= 4'b11111111;
   4'b11111111: state <= 4'b01111111;
   4'b01111111: state <= 4'b00111111;
   4'b00111111: state <= 4'b00011111;
   4'b00011111: state <= 4'b00001111;
   4'b00001111: state <= 4'b00000111;
   4'b00000111: state <= 4'b00000011;
   4'b00000011: state <= 4'b00000001;
   4'b00000001: state <= 4'b00000000;
   endcase
end
```

```
module RingBoolean(clk, state);

input clk;
output [3:0] state;
reg [3:0] state;

initial state <= 4'b10000000;

always@(posedge clk)
begin
   state[0] <= state[1];
   state[1] <= state[2];
   state[2] <= state[3];
   state[3] <= state[4];
   state[4] <= state[5];
   state[5] <= state[6];
   state[6] <= state[7];
   state[7] <= state[0];
end

endmodule
```

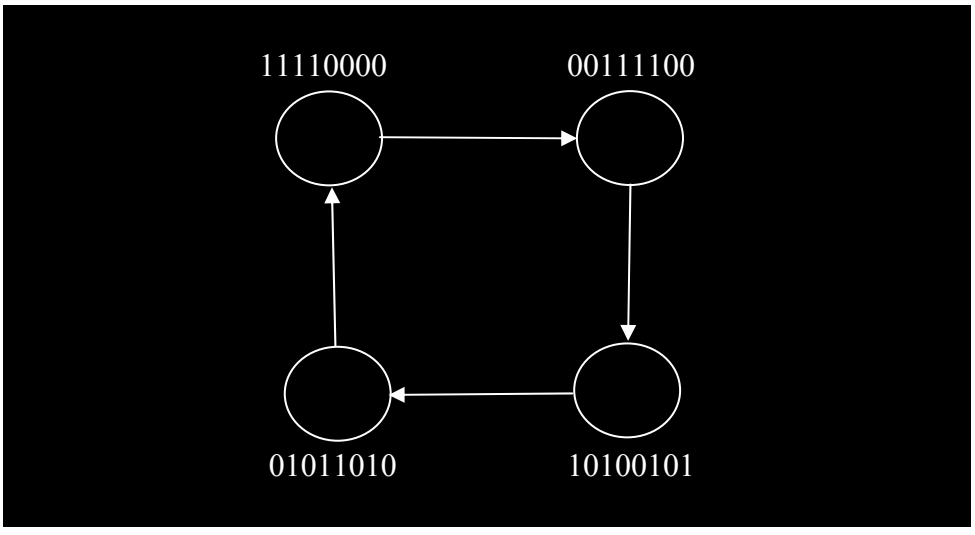
```
module RingBehavioral(clk, state);

input clk;
output [3:0] state;
reg [3:0] state;

initial state <= 4'b10000000;

always@(posedge clk)
begin
   case (state)
   4'b10000000: state <= 4'b01000000;
   4'b01000000: state <= 4'b00100000;
   4'b00100000: state <= 4'b00010000;
   4'b00010000: state <= 4'b00001000;
   4'b00001000: state <= 4'b00000100;
   4'b00000100: state <= 4'b00000010;
   4'b00000010: state <= 4'b00000001;
   4'b00000001: state <= 4'b10000000;
   endcase
end

endmodule
```

3. Let's design a state-machine-based controller using a similar (but not exactly the same) approach as given by Vahid in Chapter 2 slide's page 22. The OUTPUT transition is:

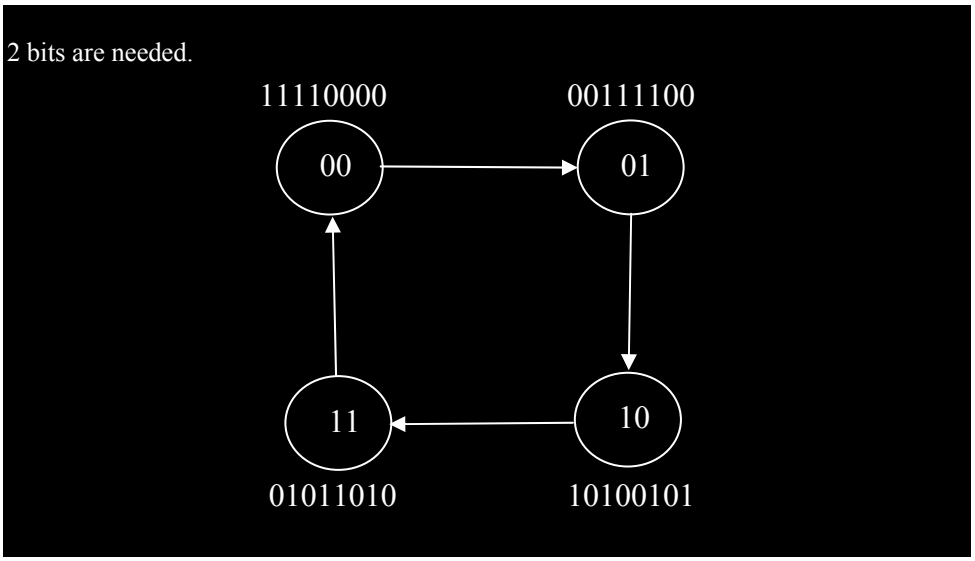
   11110000 → 00111100 → 10100101 → 01011010 repeat

   [This is an exercise to test whether you can follow instruction or not.]

   i. Assign each output to a state and draw the state diagram.



ii. Assign each state with a binary value. How many bits do you need?



iii. Construct an excitation table (truth table for sequential circuit) to show the state transition.

Current State	Next State
00	01
01	10
10	11
11	00

iv. Construct a truth table to show which state produces which output pattern.

State	Ouput
00	11110000
01	00111100
10	10100101
11	01011010

v. Write the Verilog code to implement the excitation table in (iii).

```
module StepperMotor(clk, Op);

input clk;
output [7:0] Op;
reg [7:0] Op;
reg [1:0] state;

initial state <= 2'b00;

always@(posedge clk)
begin
   case (state)
   2'b00: state <= 2'b01;
   2'b01: state <= 2'b10;
   2'b10: state <= 2'b11;
   2'b11: state <= 2'b00;
   endcase
end

endmodule
```

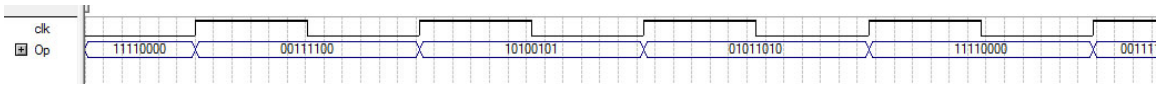
vi. In the same code, implement the Boolean expression to compute the output from the state as input.

```
module StepperMotor(clk, Op);

input clk;
output [7:0] Op;
reg [7:0] Op;
reg [1:0] state;

initial
begin
   state <= 2'b00;
   Op <= 8'b11110000;
end

always@(posedge clk)
begin
   case (state)
   2'b00: state = 2'b01;
   2'b01: state = 2'b10;
   2'b10: state = 2'b11;
   2'b11: state = 2'b00;
   endcase
   if      (state == 2'b00) Op = 8'b11110000;
   else if (state == 2'b01) Op = 8'b00111100;
   else if (state == 2'b10) Op = 8'b10101010;
   else                     Op = 8'b01011010;
end

endmodule
```

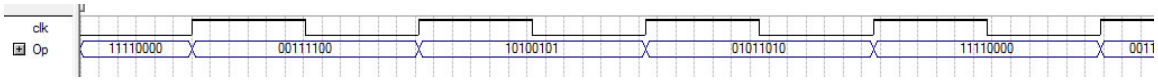
vii. Re-write the whole code using pure behavioral approach. The output simulation waveforms must be the same for the two implementations, but compare the RTL views and try to explain the differences.

```
module StepperMotor2(clk, Op);

input clk;
output [7:0] Op;
reg [7:0] Op;

initial Op <= 8'b11110000;

always@(posedge clk)
begin
   case (Op)
   8'b11110000: Op <= 8'b00111100;
   8'b00111100: Op <= 8'b10100101;
   8'b10100101: Op <= 8'b01011010;
   8'b01011010: Op <= 8'b11110000;
   endcase
end

endmodule
```

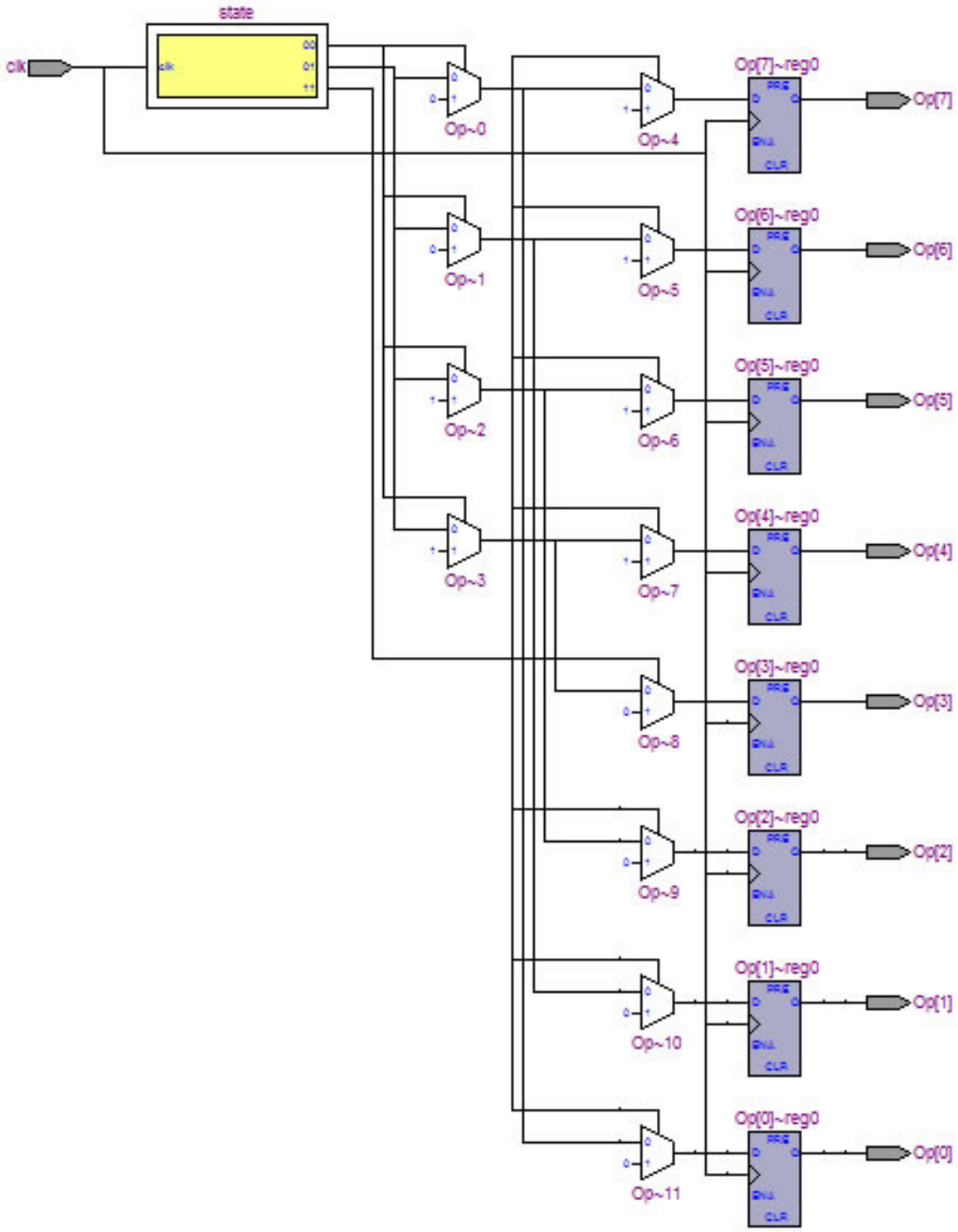
Simulation of `StepperMotor`:



Simulation of `StepperMotor2`:



RTL View of `StepperMotor`:



RTL View of `StepperMotor2` is too complicated to be posted in this document. This clearly shows that, while it requires more coding, Vahid's approach results in better circuit. The state machine (yellow box) in the approach has reduced many of the complications.

4. Using the 3 bit shift register given in Figure Q4, construct a sequential system that is capable of detecting an input sequence of 101, i.e. it gives an output 1 after it receives 101.

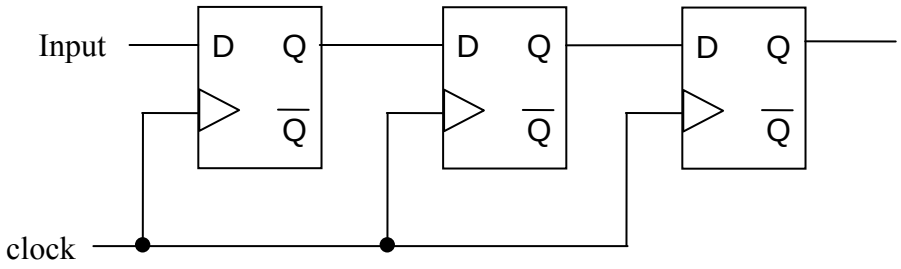
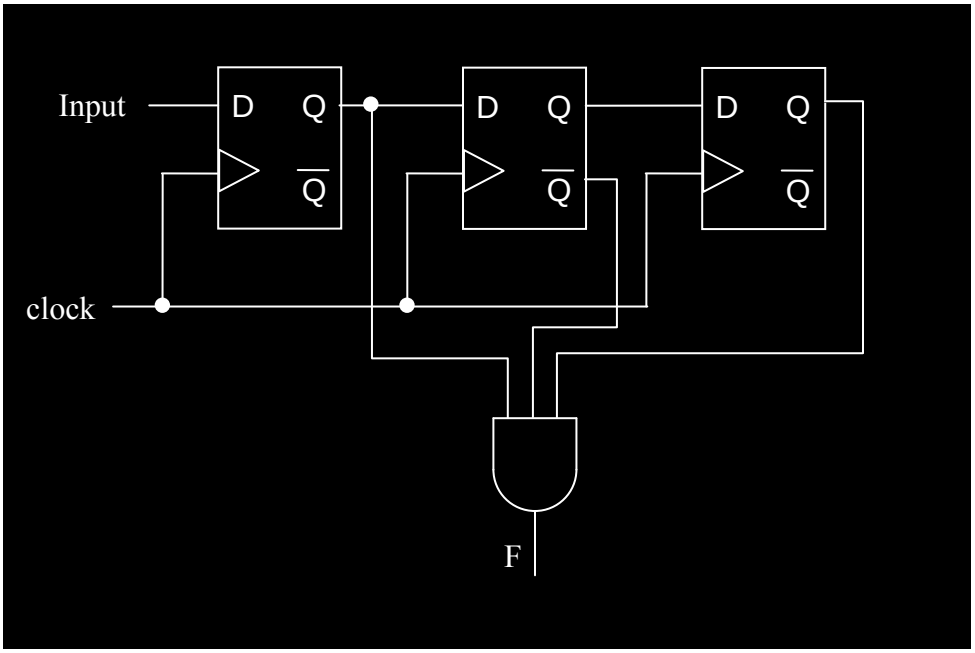
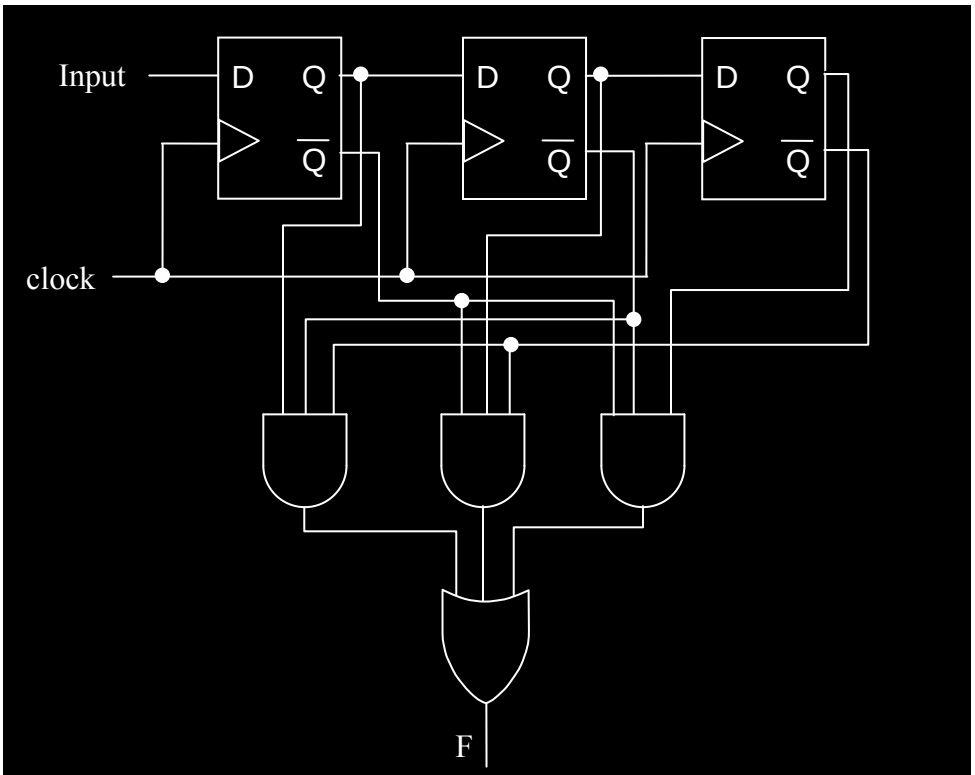


Figure Q4



5. Using the same 3 bit shift register as in Question 5, construct a sequential system that is capable of detecting a sequence of binary pattern that contains only one bit 1 stored in the register.



The earlier answer given in the class, i.e. $F = Q_2$ XOR Q_1 XOR Q_0, was wrong. This is because at $Q_2 = Q_1 = Q_0 = 1$, (3 ones) F also gives a 1.