Ho Chi Minh City University of Technology
Department of Computer Science & Engineering

Digital Systems Report

LAB 05

Group 1

No.	Name	ID
1	Vo Dong Ho	1752219
2	Nguyen Minh Nhat	1752039
3	Huynh Gia An Tien	1752538
4	Pham Minh Tuan	1752595

Ho Chi Minh City, May 2018

Table of Contents

Content

Question 1. *Distinguish between Combinational circuit and Sequential circuit.*

- A combinational circuit has no memory characteristic, so its output depends only on the current value of its inputs.
- A sequential circuit is one in which the outputs follow a predetermined sequence of states, with a new state occurring each time a clock pulse occurs.

Question 2.

- *How can you declare a negative triggered edge in sensitive list?*
- *In the example, assume that*

$$\begin{cases} out1 = 5 \\ out2 = 0 \\ in = 1 \end{cases}$$

  What is the value of and out2 in the clock trigger? Explain your results? Answer: $out1 = 0$ and $out2 = 5$ - since the assignments are non-blocking assignments.

- *In the above example, we are using rst to reset the output value. This kind of reset is called synchronous reset. You have to change the code to describe an asynchronous reset.*

```
1  always @ (posedge clk or posedge rst)
2    begin
3      if (rst)
4        begin
5          out1 <= 0;
6          out2 <= 0;
7        end
8      else
9        begin
10         if (in == 1)
11           begin
12             out1 <= out2;
13             out2 <= out1;
14           end
15         else
16           begin
17             out1 <= out1 + 1;
18             out2 <= 0;
19           end
20       end
21   end
```

Exercises

The module **seven_seg.v** (7seg-led decoder) is used in most exercises.

```
1  module seven_seg(
2    seg,
3    bcd
4  );
5
6    output [6:0] seg;
7    input  [3:0] bcd;
8
9    assign seg = (bcd == 4'd0) ? 7'b1000000
10              : (bcd == 4'd1) ? 7'b1111001
11              : (bcd == 4'd2) ? 7'b0100100
12              : (bcd == 4'd3) ? 7'b0110000
13              : (bcd == 4'd4) ? 7'b0011001
14              : (bcd == 4'd5) ? 7'b0010010
15              : (bcd == 4'd6) ? 7'b0000011
16              : (bcd == 4'd7) ? 7'b1111000
17              : (bcd == 4'd8) ? 7'b0000000
18              : (bcd == 4'd9) ? 7'b0011000
19              : /* undefined */ 7'b1111111;
20
21 endmodule
```

Exercise 1.

Implement D FF and JK FF

1. **D FF**

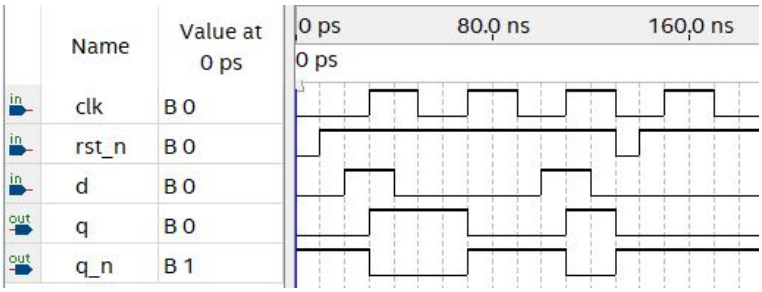
- Function table:

R	CLK	D	Q	Q'
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

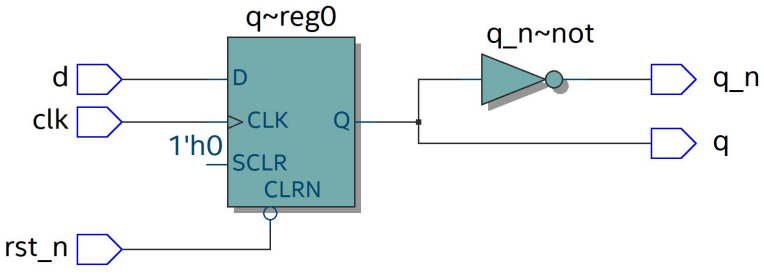
- Verilog:

```
1  module d_ff(
2    q, q_n,
3    clk, d, rst_n
4  );
5
6    output q, q_n;
7    reg    q;
8    input  clk,
9           d,
10          rst_n;
11
12   always @ (posedge clk or negedge rst_n)
13     begin
14       if (!rst_n)
15         q <= 0;
16       else
17         q <= d;
18     end
19
20   assign q_n = ~q;
21
22 endmodule
```

- Waveform:



- Generated Circuit:



2. **JK FF**

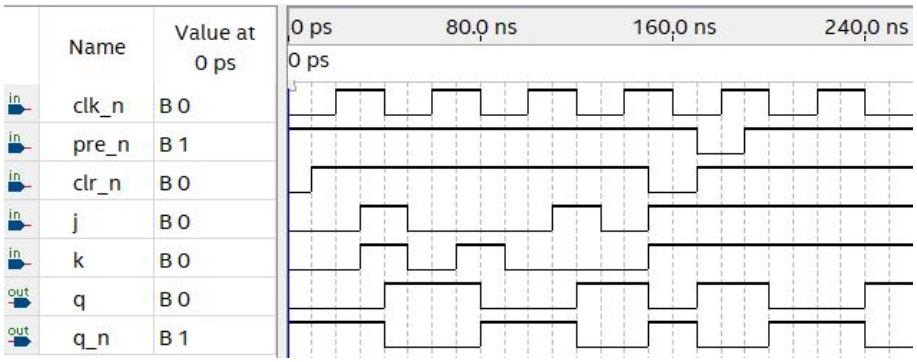
- Function table:

$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q
1	1	$\downarrow$	0	0	Q (no change)
1	1	$\downarrow$	0	1	0 (synch reset)
1	1	$\downarrow$	1	0	1 (synch set)
1	1	$\downarrow$	1	1	$\overline{Q}$ (synch toggle)
1	1	X	X	X	Q (no change)
1	0	X	X	X	0 (asynch clear)
0	1	X	X	X	1 (asynch preset)
0	0	X	X	X	(invalid)

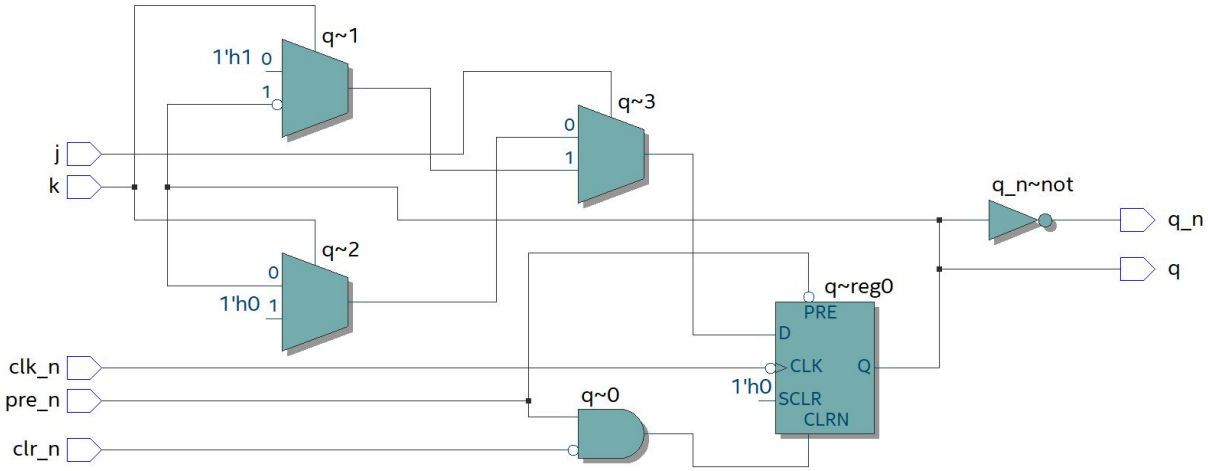
- Verilog:

```
1  module jk_ff(
2    q, q_n,
3    pre_n, clr_n,
4    clk_n,
5    j, k
6  );
7
8    output q,
9           q_n;
10   reg    q;
11   input  pre_n,
12          clr_n,
13          clk_n,
14          j,
15          k;
16
17   always @ (negedge clk_n or negedge pre_n or negedge clr_n)
18     begin
19       if (!pre_n)
20         q <= 1;
21       else if (!clr_n)
22         q <= 0;
23       else
24         if (j)
25           if (k)
26             q <= ~q;
27           else
28             q <= 1;
29         else
30           if (k)
31             q <= 0;
32     end
33
34   assign q_n = ~q;
35
36 endmodule
```

- Waveform:



- Generated Circuit:



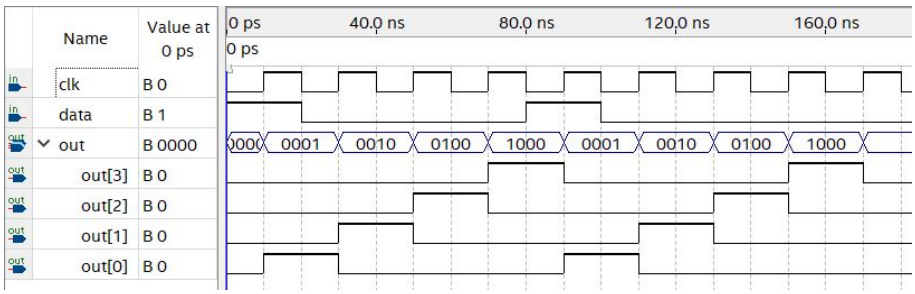
Exercise 2.

Implement a shift 4-bit circuit by using 4 D FF.

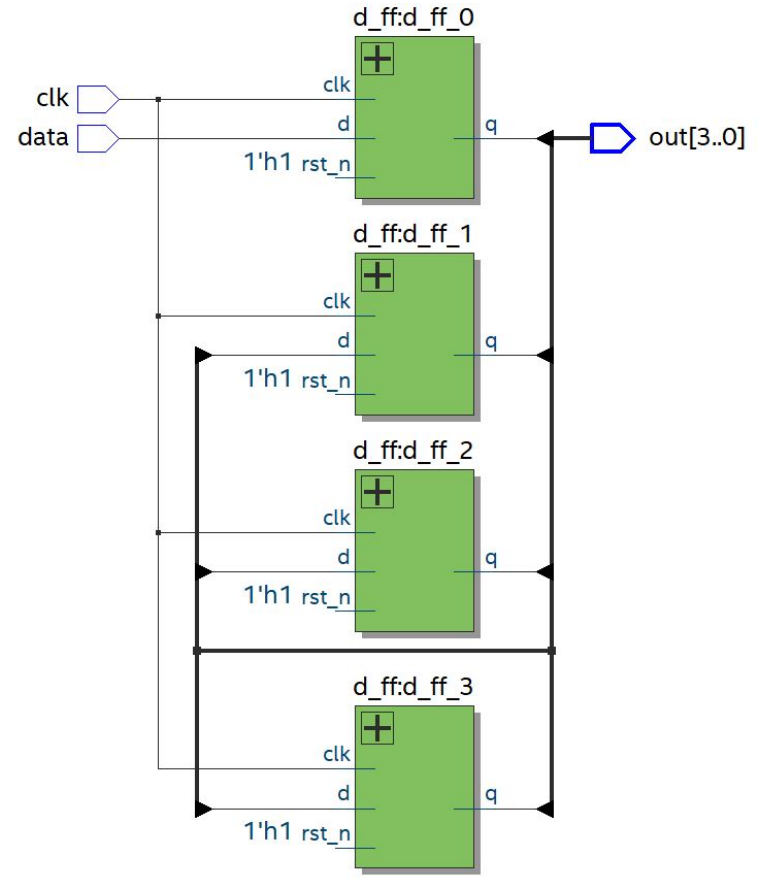
- **Design**
  - The output Q of the previous D FF is connected to the input D of the next FF.
  - One common CLK signal is used for all FFs.
- **Verilog Code**

```
1  module four_bit_shift_reg(
2    out,
3    clk,
4    data
5  );
6
7    output [3:0] out;
8    input  clk,
9           data;
10
11   d_ff d_ff_0(
12     .q(out[0]),
13     .clk(clk),
14     .d(data),
15     .rst_n(1)
16   );
17
18   d_ff d_ff_1(
19     .q(out[1]),
20     .clk(clk),
21     .d(out[0]),
22     .rst_n(1)
23   );
24
25   d_ff d_ff_2(
26     .q(out[2]),
27     .clk(clk),
28     .d(out[1]),
29     .rst_n(1)
30   );
31
32   d_ff d_ff_3(
33     .q(out[3]),
34     .clk(clk),
35     .d(out[2]),
36     .rst_n(1)
37   );
38
39 endmodule
```

- **Waveform**



- **Generated Circuit**



Exercise 3.

Design clock divider circuit provided a 50 MHz clock.

- **Design**
  - The frequency is reduced by half for each D FF it passes through.
  - For an input frequency of 50MHz, to obtain the output frequency of 1Hz (1 led blink per second), there needs to be:

$$\lceil \log_2(50 \cdot 10^6) \rceil = 26 \qquad (DFF)$$

  - The output Q of the previous D FF is the CLK signal of the next FF.
  - Each time a D FF is clock-trigged, it "toggles" itself. D FFs do not have the built-in toggle function like JK FFs, but we can obtain the same functionality by connecting the D input of one FF to its inverted output $\overline{Q}$.

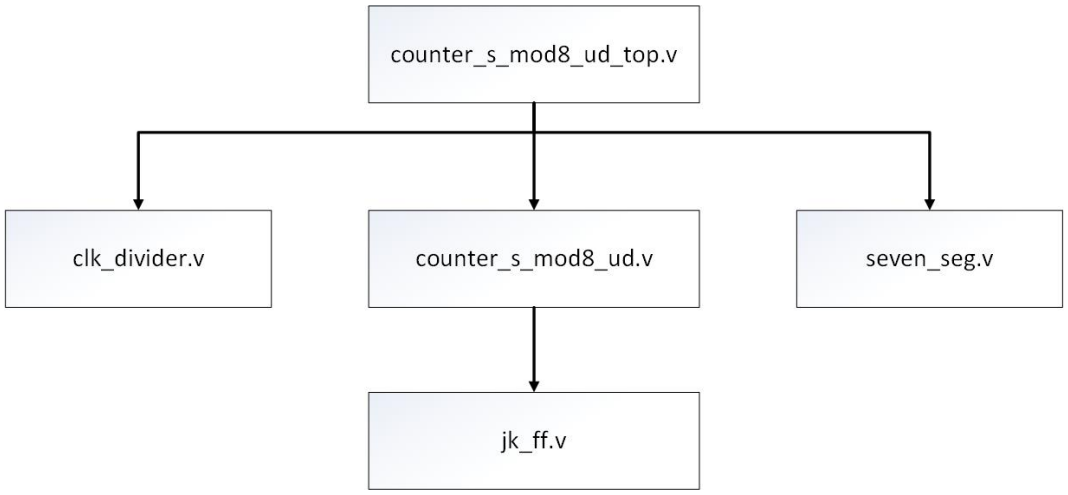
- **Verilog Code**

```
1  module clk_divider(
2    fout,
3    fin
4  );
5
6    output fout;
7    wire   [26:0] f;
8    input  fin;
9
10   assign f[0] = fin;
11
12   genvar i;
13   generate
14     for (i = 1; i <= 26; i = i + 1)
15       begin : d_ff_label
16         d_ff d_ff(
17           .q(f[i]),
18           .q_n(),
19           .d(~f[i]),
20           .clk(f[i - 1]),
21           .rst_n(1)
22         );
23       end
24   endgenerate
25
26   assign fout = f[26];
27
28 endmodule
```

Exercise 4.

Implement a synchronous counter mod 8 which can count Up/Down by using JK FF and show the result on 7seg-led.

- **Design**
  - Since the counter is synchronous, each JK FF has the same CLK signal.
  - If UP is 1, then the counter counts up. Otherwise, it counts down.
  - If the counter counts up, each FF toggles when all outputs of previous FFs are 1s.
  - If the counter counts down, each FF toggles when all outputs of previous FFs are 0s.
  - Hierarchical design:



- **Verilog Code**

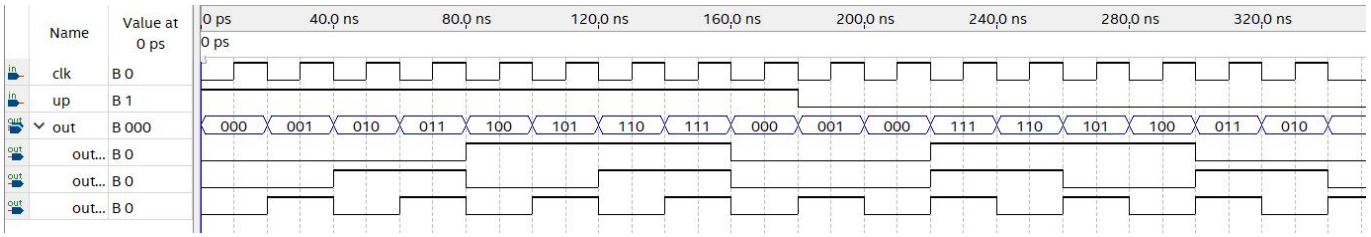
```
1  module counter_s_mod8_ud(
2    out,
3    up,
4    clk
5  );
6
7    output [2:0] out;
8    input  up,
9           clk;
10   wire   [2:1] in;
11   wire   [2:1] and_wire;
12   wire   [2:1] and_wire_n;
13
14   // FF 0
15   jk_ff jk_ff_0(
16     .q(out[0]),
17     .pre_n(1),
18     .clr_n(1),
19     .j(1),
20     .k(1),
21     .clk_n(clk)
22   );
23
24   // FF 1
25   and and_gate_1(
26     and_wire[1],
27     up, out[0]
28   );
```

```
29
30    and and_gate_n_1(
31      and_wire_n[1],
32      ~up, ~out[0]
33    );
34
35    or or_gate_1(
36      in[1],
37      and_wire[1], and_wire_n[1]
38    );
39
40    jk_ff jk_ff_1(
41      .q(out[1]),
42      .pre_n(1),
43      .clr_n(1),
44      .j(in[1]),
45      .k(in[1]),
46      .clk_n(clk)
47    );
48
49    // FF 2
50    and and_gate_2(
51      and_wire[2],
52      up, out[0], out[1]
53    );
54
55    and and_gate_n_2(
56      and_wire_n[2],
57      ~up, ~out[0], ~out[1]
58    );
59
60    or or_gate_2(
61      in[2],
62      and_wire[2], and_wire_n[2]
63    );
64
65    jk_ff jk_ff_2(
66      .q(out[2]),
67      .pre_n(1),
68      .clr_n(1),
69      .j(in[2]),
70      .k(in[2]),
71      .clk_n(clk)
72    );
73
74  endmodule
```

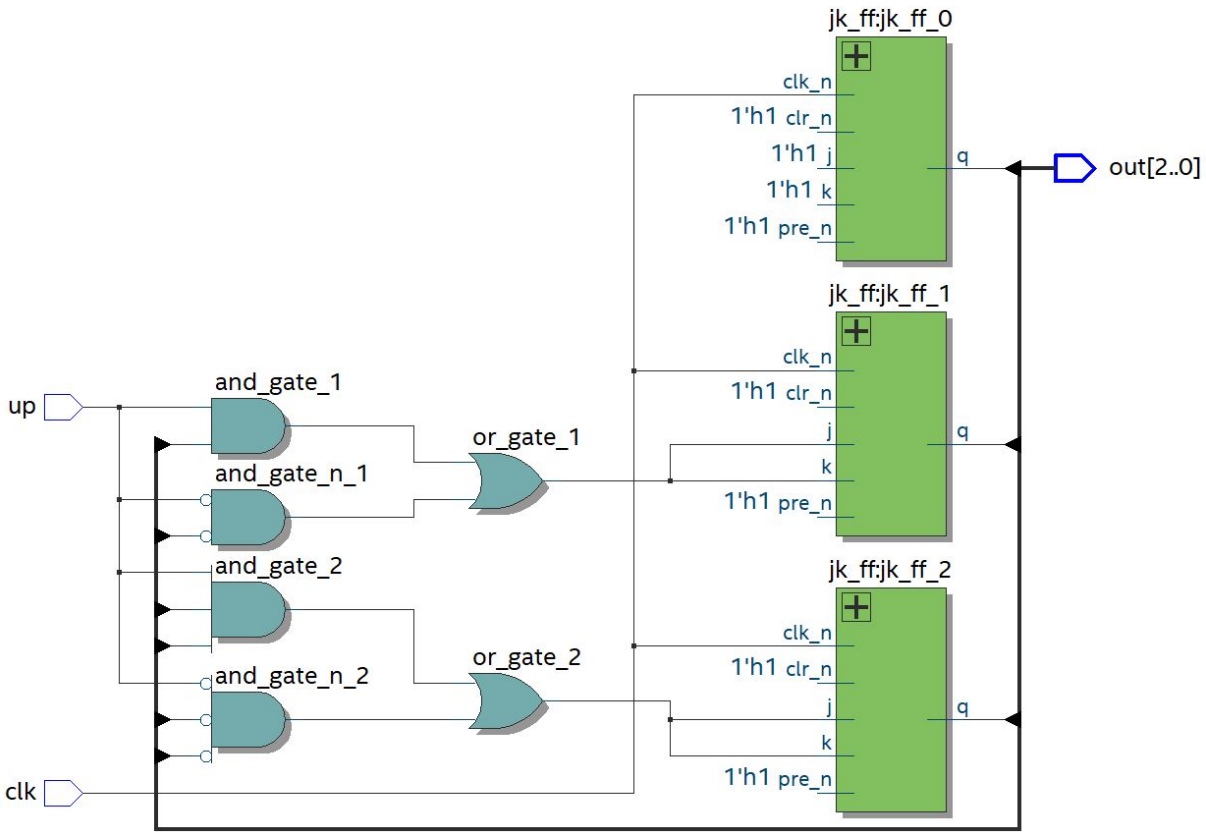
```
1   module counter_s_mod8_ud_top(
2     seg,
3     raw_clk,
4     up
5   );
6
7     output [6:0] seg;
8     input  raw_clk;
9     input  up;
10
11    wire  clk;
12    wire  [2:0] counter_out;
13
14    clk_divider clk_divider(
15      .fout(clk),
16      .fin(raw_clk)
```

```
17    );
18
19    counter_s_mod8_ud counter(
20      .out(counter_out),
21      .up(up),
22      .clk(clk)
23    );
24
25    seven_seg seven_seg(
26      .bcd(counter_out),
27      .seg(seg)
28    );
29
30  endmodule
```

– **Waveform**



– **Generated Circuit**



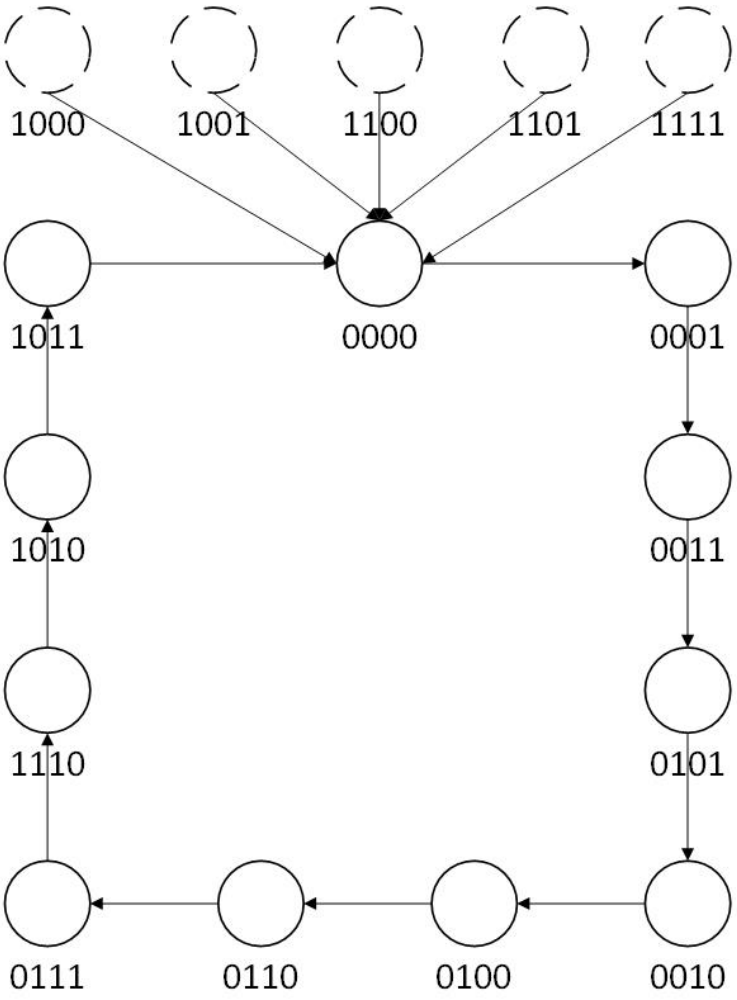
Exercise 5.

Implement a synchronous counter mod 11 by using D FF with the rule: 0 1 3 5 2 4 6 7 14 10 11 0... show the result on 7seg-led bonus if you use JK FF instead of D FF.

– **Design**

Using JK FFs.

* Since the counter is synchronous, each JK FF has the same CLK signal.
* A CLR signal is added to the counter to set initial state for waveform.
* To design the synchronous counter with special rule, we need to take these steps:
  1. **Determined the number of FFs needed**.
     Since the largest number is $14 < 16 = 2^4$, 4 FFs are used.
  2. **Draw the state diagram of the counter, including all used and unused states.**
     Used states are full circles; unused states are dashed circles.



3. **Create the present state - next state table**

	Previous State	**Next State**
0	0000	0001
1	0001	0011
2	0010	0100
3	0011	0101
4	0100	0110
5	0101	0010
6	0110	0111
7	0111	1110
8	1000	0000
9	1001	0000
10	1010	1011
11	1011	0000
12	1100	0000
13	1101	0000
14	1110	1010
15	1111	0000

4. **For each PRESENT state, determine the required J and K input in order to produce the transition to the NEXT state**.

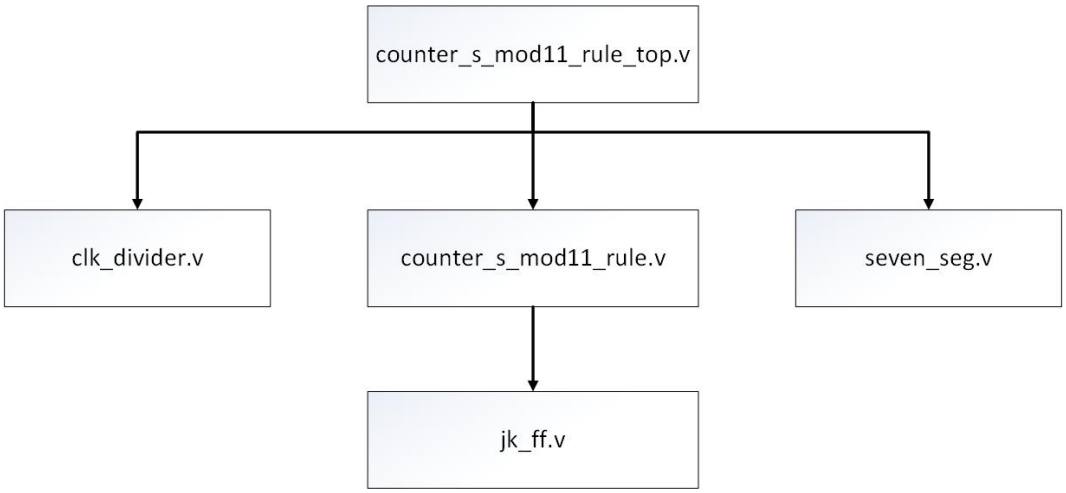
	Present state	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0000	0	X	0	X	0	X	1	X
1	0001	0	X	0	X	1	X	X	0
2	0010	0	X	1	X	X	1	0	X
3	0011	0	X	1	X	X	1	X	0
4	0100	0	X	X	0	1	X	0	X
5	0101	0	X	X	1	1	X	X	1
6	0110	0	X	X	0	X	0	1	X
7	0111	1	X	X	0	X	0	X	1
8	1000	X	1	0	X	0	X	0	X
9	1001	X	1	0	X	0	X	X	1
10	1010	X	0	0	X	X	0	1	X
11	1011	X	1	0	X	X	1	X	1
12	1100	X	1	X	1	0	X	0	X
13	1101	X	1	X	1	0	X	X	1
14	1110	X	0	X	1	X	0	0	X
15	1111	X	1	X	1	X	1	X	1

5. **Simplify each J and K with Kmap or algebraically.**
   Final result after simplifying with Kmap:

   $J_0 = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C$
   $K_0 = A + B$
   $J_1 = \bar{A}B + \bar{A}D$
   $K_1 = \bar{A}\bar{B} + AD$
   $J_2 = \bar{A}C$
   $K_2 = A + \bar{C}D$
   $J_3 = BCD$
   $K_3 = \bar{C} + D$

   where A is the MSB.

* Hierarchical design:



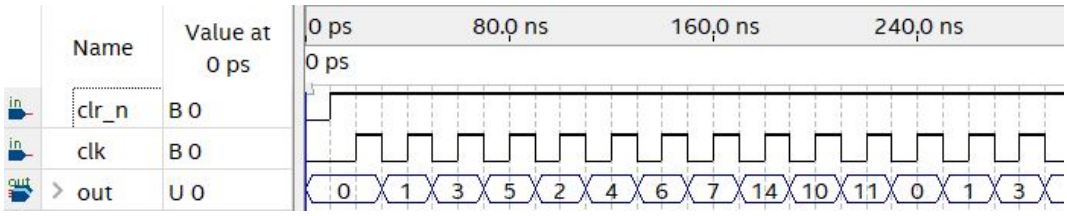
– **Verilog Code**

```
1  module counter_s_mod11_rule(
2    out,
3    clk,
4    clr_n,
5  );
6
7    output [3:0] out;
8    input  clk,
9           clr_n;
10
11   wire   [3:0] j;
12   wire   [3:0] k;
13
14   /********* Wire assignment *********/
15   wire   a, b, c, d;
16   assign a = out[3];
17   assign b = out[2];
18   assign c = out[1];
19   assign d = out[0];
20
21   /*********** FF Inputs ***********/
22   assign j[0] = (~a & ~b & ~c) | (~a & b & c) | (a & ~b & c);
23   assign k[0] = (a) | (b);
24   assign j[1] = (~a & b) | (~a & d);
25   assign k[1] = (~a & ~b) | (a & d);
26   assign j[2] = (~a & c);
27   assign k[2] = (a) | (~c & d);
28   assign j[3] = (b & c & d);
29   assign k[3] = (~c) | (d);
30
31   /*********** FFs **************/
32   jk_ff jk_ff_0(
33     .q(out[0]),
34     .pre_n(1),
35     .clr_n(clr_n),
36     .clk_n(clk),
37     .j(j[0]),
38     .k(k[0])
39   );
40
41   jk_ff jk_ff_1(
42     .q(out[1]),
43     .pre_n(1),
44     .clr_n(clr_n),
45     .clk_n(clk),
46     .j(j[1]),
47     .k(k[1])
48   );
49
50   jk_ff jk_ff_2(
51     .q(out[2]),
52     .pre_n(1),
53     .clr_n(clr_n),
54     .clk_n(clk),
55     .j(j[2]),
56     .k(k[2])
57   );
58
59   jk_ff jk_ff_3(
60     .q(out[3]),
61     .pre_n(1),
```

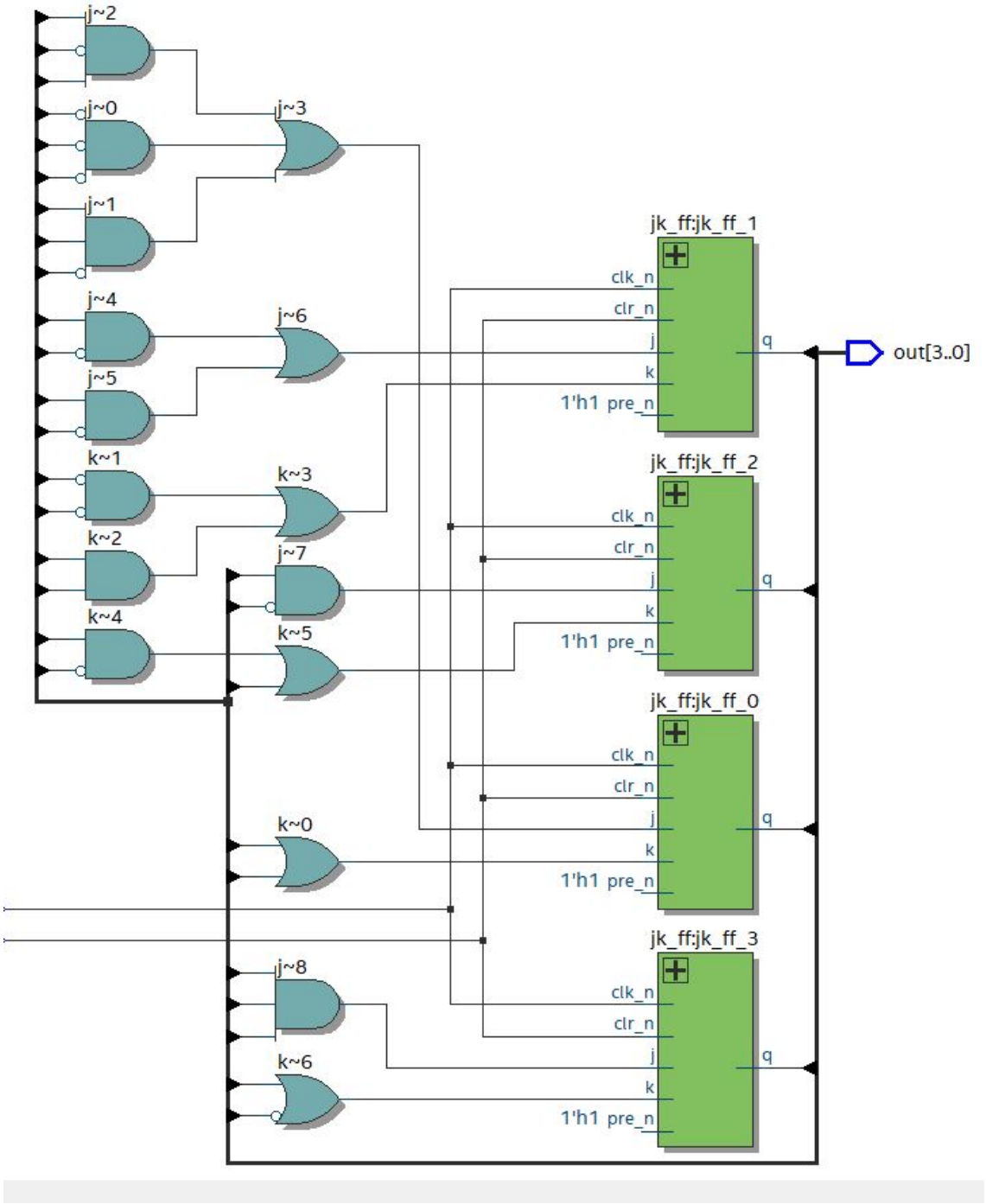
```
62      .clr_n(clr_n),
63      .clk_n(clk),
64      .j(j[3]),
65      .k(k[3])
66    );
67
68  endmodule
```

```
1   module counter_s_mod11_rule_top(
2     seg,
3     raw_clk,
4     clr_n
5   );
6
7     output [13:0] seg;
8     input  raw_clk;
9     input  clr_n;
10    wire   clk;
11    wire   [3:0] counter_out;
12    wire   [3:0] unit_bcd;
13    wire   [3:0] tenth_bcd;
14
15    clk_divider clk_divider(
16      .fout(clk),
17      .fin(raw_clk)
18    );
19
20    counter_s_mod11_rule counter(
21      .out(counter_out),
22      .clk(clk),
23      .clr_n(clr_n)
24    );
25
26    assign unit_bcd = (counter_out > 4'd9) ? (counter_out - 10)
27                                           : (counter_out);
28    assign tenth_bcd = (counter_out > 4'd9) ? (1) : (0);
29
30    seven_seg seven_seg_0(
31      .bcd(unit_bcd),
32      .seg(seg[6:0])
33    );
34
35    seven_seg seven_seg_1(
36      .bcd(tenth_bcd),
37      .seg(seg[13:7])
38    );
39
40  endmodule
```

– **Waveform**



– **Generated Circuit**



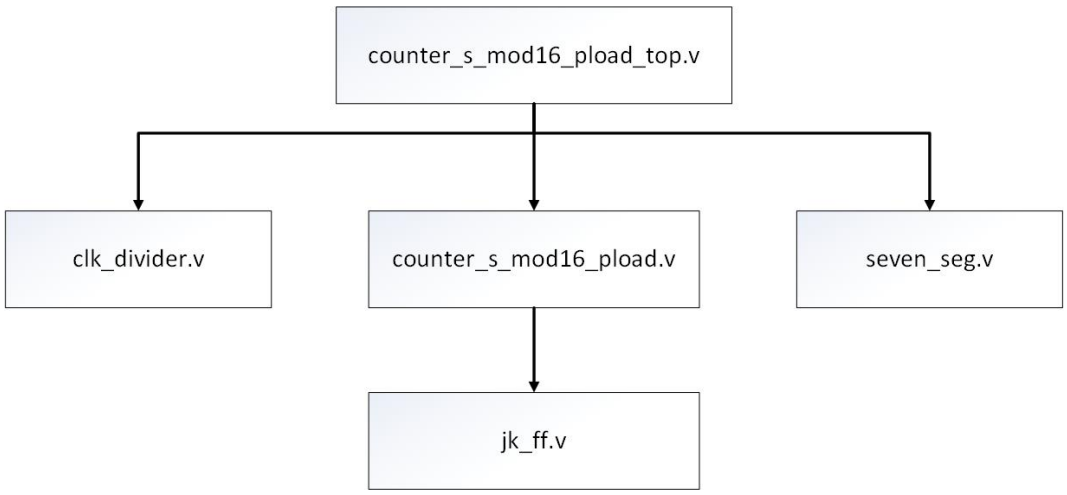
Exercise 6.

Implement 4 bit Synchronous counter with asynchronous parallel load.

- **Design**

  FF type: JK

  * Since the counter is synchronous, each JK FF has the same CLK signal.
  * Use the RTL branching structure, we can assign the suitable value for PRE and CLR when PL is active:
    - If $DATA = 1$, then $PRE = 0$ and $CLR = 1$.
    - If $DATA = 0$, then $PRE = 1$ and $CLR = 0$.
  * Hierarchical design:



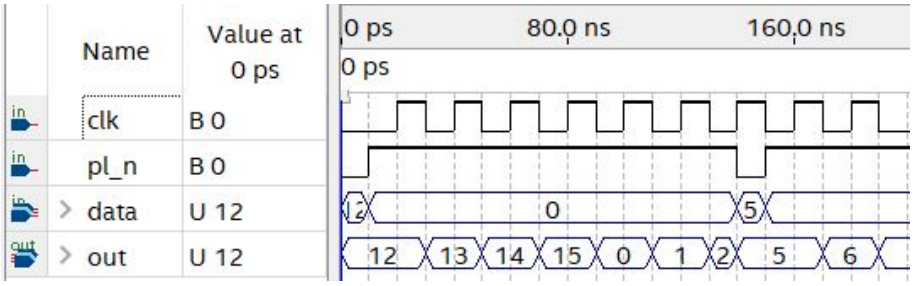
– **Verilog Code**

```
1  module counter_s_mod16_pload(
2    out,
3    clk,
4    data,
5    pl_n
6  );
7
8    output [3:0] out;
9    input  clk;
10   input  [3:0] data;
11   input  pl_n;
12
13   wire   [3:0] pre_n;
14   wire   [3:0] clr_n;
15   wire   [3:1] j;
16   wire   [3:1] k;
17
18   assign pre_n[0] = (!pl_n) ? (~data[0]) : (1);
19   assign clr_n[0] = (!pl_n) ?  (data[0]) : (1);
20   assign pre_n[1] = (!pl_n) ? (~data[1]) : (1);
21   assign clr_n[1] = (!pl_n) ?  (data[1]) : (1);
22   assign pre_n[2] = (!pl_n) ? (~data[2]) : (1);
23   assign clr_n[2] = (!pl_n) ?  (data[2]) : (1);
24   assign pre_n[3] = (!pl_n) ? (~data[3]) : (1);
25   assign clr_n[3] = (!pl_n) ?  (data[3]) : (1);
26
27   jk_ff jk_ff_0(
28     .q(out[0]),
29     .pre_n(pre_n[0]),
30     .clr_n(clr_n[0]),
31     .clk_n(clk),
32     .j(1),
33     .k(1)
34   );
35
36   assign j[1] = out[0];
37   assign k[1] = out[0];
38
39   jk_ff jk_ff_1(
40     .q(out[1]),
41     .pre_n(pre_n[1]),
42     .clr_n(clr_n[1]),
43     .clk_n(clk),
44     .j(j[1]),
45     .k(k[1])
46   );
47
48   assign j[2] = out[0] & out[1];
49   assign k[2] = out[0] & out[1];
50
51   jk_ff jk_ff_2(
52     .q(out[2]),
53     .pre_n(pre_n[2]),
54     .clr_n(clr_n[2]),
55     .clk_n(clk),
56     .j(j[2]),
57     .k(k[2])
58   );
59
60   assign j[3] = out[0] & out[1] & out[2];
61   assign k[3] = out[0] & out[1] & out[2];
```

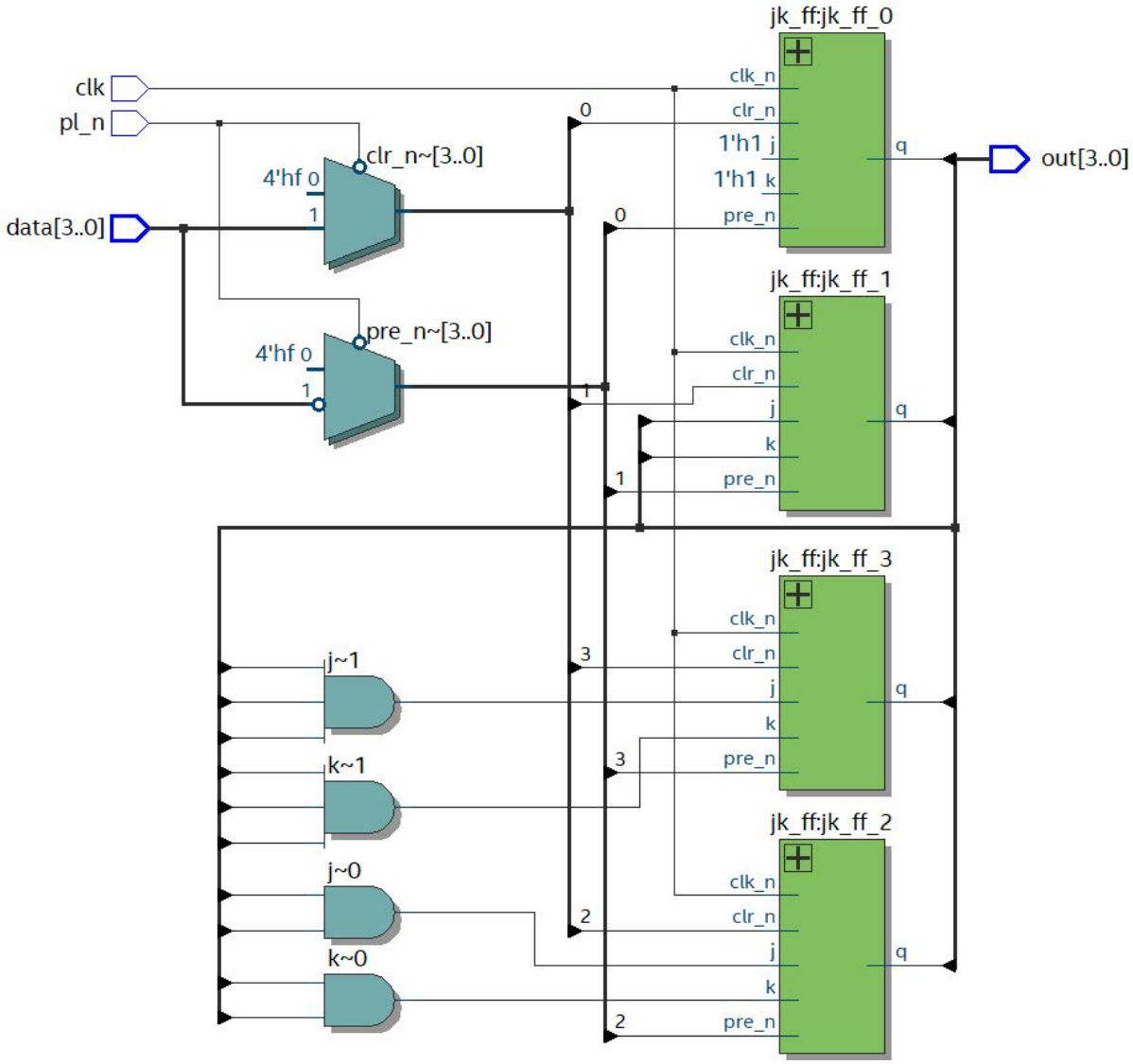
```
62
63    jk_ff jk_ff_3(
64      .q(out[3]),
65      .pre_n(pre_n[3]),
66      .clr_n(clr_n[3]),
67      .clk_n(clk),
68      .j(j[3]),
69      .k(k[3])
70    );
71
72  endmodule
```

```
1   module counter_s_mod16_pload_top(
2     seg,
3     raw_clk,
4     data,
5     pl_n
6   );
7
8     output [13:0] seg;
9     input  raw_clk;
10    input  [3:0] data;
11    input  pl_n;
12    wire   clk;
13    wire   [3:0] counter_out;
14    wire   [3:0] unit_bcd;
15    wire   [3:0] tenth_bcd;
16
17    clk_divider clk_divider(
18      .fout(clk),
19      .fin(raw_clk)
20    );
21
22    counter_s_mod16_pload counter(
23      .out(counter_out),
24      .clk(clk),
25      .data(data),
26      .pl_n(pl_n)
27    );
28
29    assign unit_bcd = (counter_out > 4'd9) ? (counter_out - 4'd10) : (
          counter_out);
30    assign tenth_bcd = (counter_out > 4'd9) ? (4'd1) : (4'd0);
31
32    seven_seg seven_seg_0(
33      .bcd(unit_bcd),
34      .seg(seg[6:0])
35    );
36
37    seven_seg seven_seg_1(
38      .bcd(tenth_bcd),
39      .seg(seg[13:7])
40    );
41
42  endmodule
```

– **Waveform**



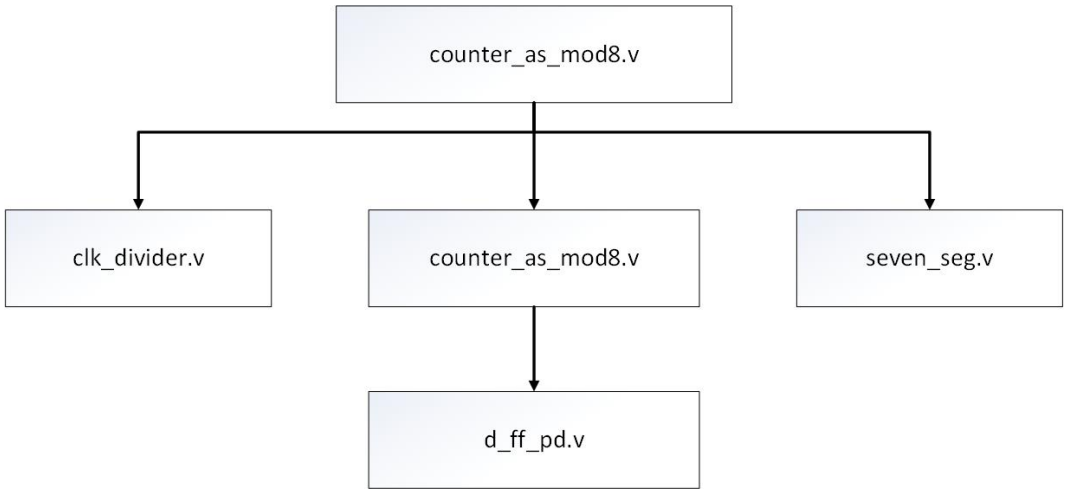
– **Generated Circuit**



Exercise 7.

Implement an asynchronous counter mod 8 by using D FF and show the result on 7seg-led.

- **Design**
  - The counter is asynchronous. Therefore, to correctly implement the counter, we have to take into account propagation delay.
  - Each time a D FF is clocked-triggered, it toggles itself. This result is obtained by connecting each FF's input D to its inverted output $\bar{Q}$.
  - The CLK signal of a D FF is the inverted output of the previous FF. Because of that, any D FF (except the first one) would toggles twice as slow as the previous one.
  - Hierarchical design:



- **Verilog Code**

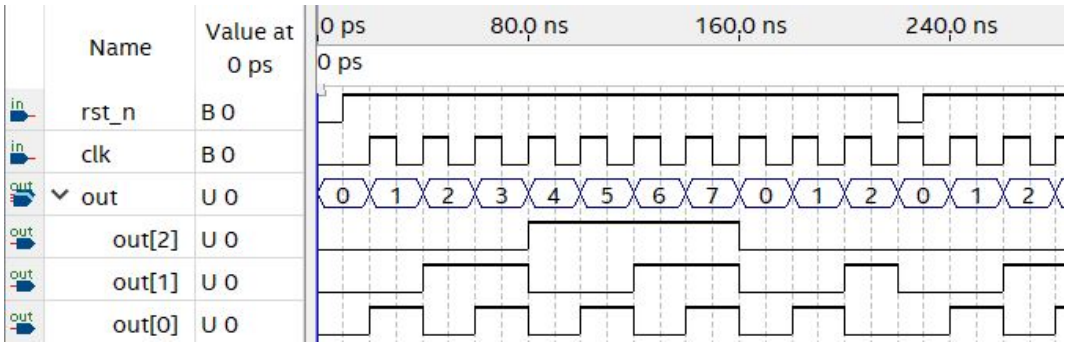
```
1  module d_ff_pd(
2    q,
3    clk, d, rst_n
4  );
5
6    output q;
7    reg    q_nodelay;
8    input  clk,
9           d,
10          rst_n;
11
12   always @ (posedge clk or negedge rst_n)
13     begin
14       if (!rst_n)
15         q_nodelay <= 0;
16       else
17         q_nodelay <= d;
18     end
19
20   assign #2 q = q_nodelay;
21
22 endmodule
```

```
1  module counter_as_mod8(
2    out,
3    clk,
4    rst_n
```

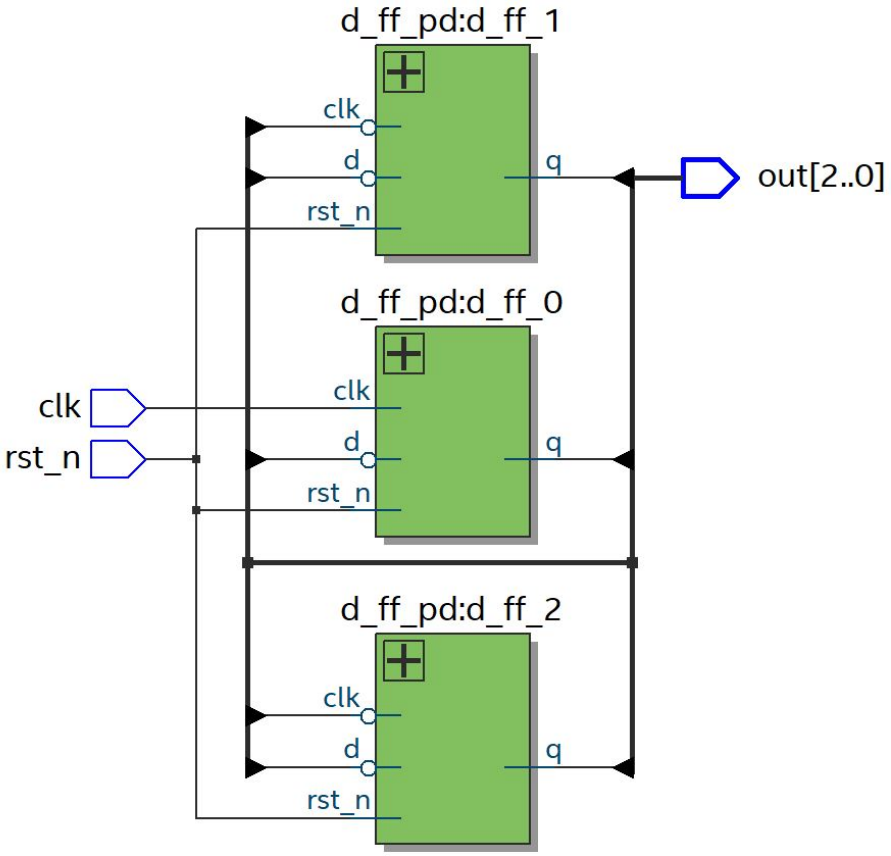
```
);

  output [2:0] out;
  input  clk,
         rst_n;

  d_ff_pd d_ff_0(
    .q(out[0]),
    .rst_n(rst_n),
    .clk(clk),
    .d(~out[0])
  );

  d_ff_pd d_ff_1(
    .q(out[1]),
    .rst_n(rst_n),
    .clk(~out[0]),
    .d(~out[1])
  );

  d_ff_pd d_ff_2(
    .q(out[2]),
    .rst_n(rst_n),
    .clk(~out[1]),
    .d(~out[2])
  );

endmodule
```

```
module counter_as_mod8_top(
  seg,
  raw_clk,
  rst_n
);

  output [6:0] seg;
  input  raw_clk;
  input  rst_n;

  wire   clk;
  wire   [2:0] clk_out;

  clk_divider clk_divider(
    .fout(clk),
    .fin(raw_clk)
  );

  counter_as_mod8 counter(
    .out(clk_out),
    .clk(clk),
    .rst_n(rst_n)
  );

  seven_seg seven_seg(
    .seg(seg),
    .bcd(clk_out)
  );

endmodule
```

– **Waveform**



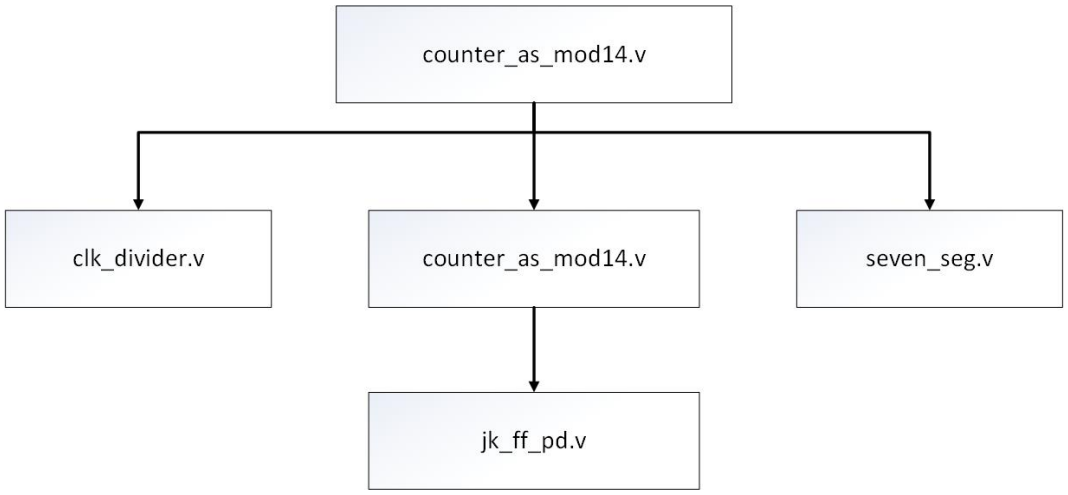
– **Generated Circuit**



Exercise 8.

Implement an asynchronous counter mod 14 by using JK FF and show the result on 7seg-led.

- **Design**
  - Similar to exercise 7, since the counter is asynchronous, we have to take into account propagation delay.
  - J and K are always kept at 1. Each time a JK FF is clocked-triggered, it toggles itself.
  - To reset at 14, A NAND gate which has
    - Inputs: all signals that are 1 at number 14.
    - Output: the common CLR signal of all JK FF.
  - There is a manual reset signal to stabilize the initial state of the counter.
  - Hierarchical design:



- **Verilog Code**

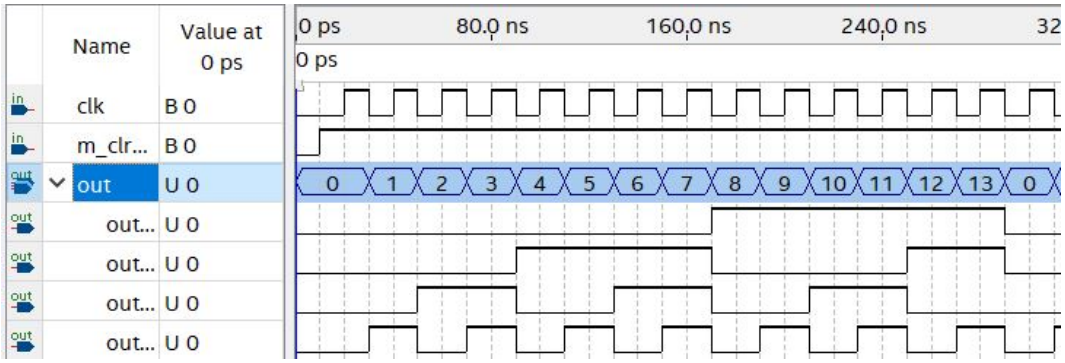
```
1  module jk_ff_pd(
2    q,
3    pre_n, clr_n,
4    clk_n,
5    j, k
6  );
7
8    output q;
9    reg    q_nodelay;
10   input  pre_n,
11          clr_n,
12          clk_n,
13          j,
14          k;
15
16   always @ (negedge clk_n or negedge pre_n or negedge clr_n)
17     begin
18       if (!pre_n)
19         q_nodelay <= 1;
20       else if (!clr_n)
21         q_nodelay <= 0;
22       else
23         if (j)
24           if (k)
25             q_nodelay <= ~q_nodelay;
```

```
26            else
27              q_nodelay <= 1;
28          else
29            if (k)
30              q_nodelay <= 0;
31      end
32
33    assign #2 q = q_nodelay;
34
35  endmodule
```

```
1   module counter_as_mod14(
2     out,
3     clk,
4     m_clr_n
5   );
6
7     output [3:0] out;
8     input  clk,
9            m_clr_n;
10
11    wire   clr_n;
12
13    assign clr_n = m_clr_n & ~(out[3] & out[2] & out[1]); // 14 == 1110
14
15    jk_ff_pd jk_ff_0(
16      .q(out[0]),
17      .pre_n(1),
18      .clr_n(clr_n),
19      .clk_n(clk),
20      .j(1),
21      .k(1)
22    );
23
24    jk_ff_pd jk_ff_1(
25      .q(out[1]),
26      .pre_n(1),
27      .clr_n(clr_n),
28      .clk_n(out[0]),
29      .j(1),
30      .k(1)
31    );
32
33    jk_ff_pd jk_ff_2(
34      .q(out[2]),
35      .pre_n(1),
36      .clr_n(clr_n),
37      .clk_n(out[1]),
38      .j(1),
39      .k(1)
40    );
41
42    jk_ff_pd jk_ff_3(
43      .q(out[3]),
44      .pre_n(1),
45      .clr_n(clr_n),
46      .clk_n(out[2]),
47      .j(1),
48      .k(1)
49    );
50
51  endmodule
```

```
1  module counter_as_mod14_top(
2    seg,
3    raw_clk,
4    clr_n
5  );
6
7    output [13:0] seg;
8    input  raw_clk;
9    input  clr_n;
10
11   wire   clk;
12   wire   [3:0] counter_out;
13   wire   [3:0] unit_bcd;
14   wire   [3:0] tenth_bcd;
15
16   clk_divider clk_divider(
17     .fout(clk),
18     .fin(raw_clk)
19   );
20
21   counter_as_mod14 counter(
22     .out(counter_out),
23     .clk(clk),
24     .m_clr_n(clr_n)
25   );
26
27   assign unit_bcd  = (counter_out > 4'd9) ? (counter_out - 4'd10)
28                                           : (counter_out);
29   assign tenth_bcd = (counter_out > 4'd9) ? (4'd1) : (4'd0);
30
31   seven_seg seven_seg_0(
32     .bcd(unit_bcd),
33     .seg(seg[6:0])
34   );
35
36   seven_seg seven_seg_1(
37     .bcd(tenth_bcd),
38     .seg(seg[13:7])
39   );
40
41 endmodule
```

– **Waveform**



– **Generated Circuit**