DIGITAL DESIGN

ASSIGNMENT REPORT

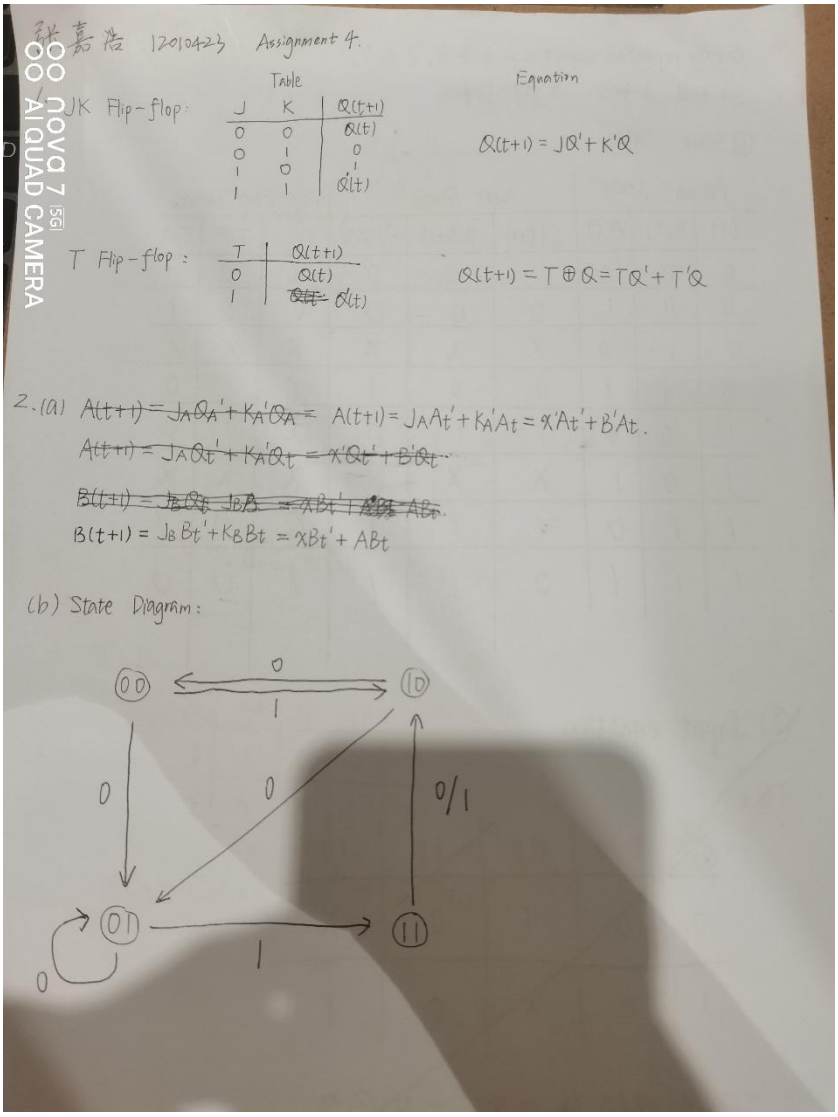
ASSIGNMENT ID: 12010423

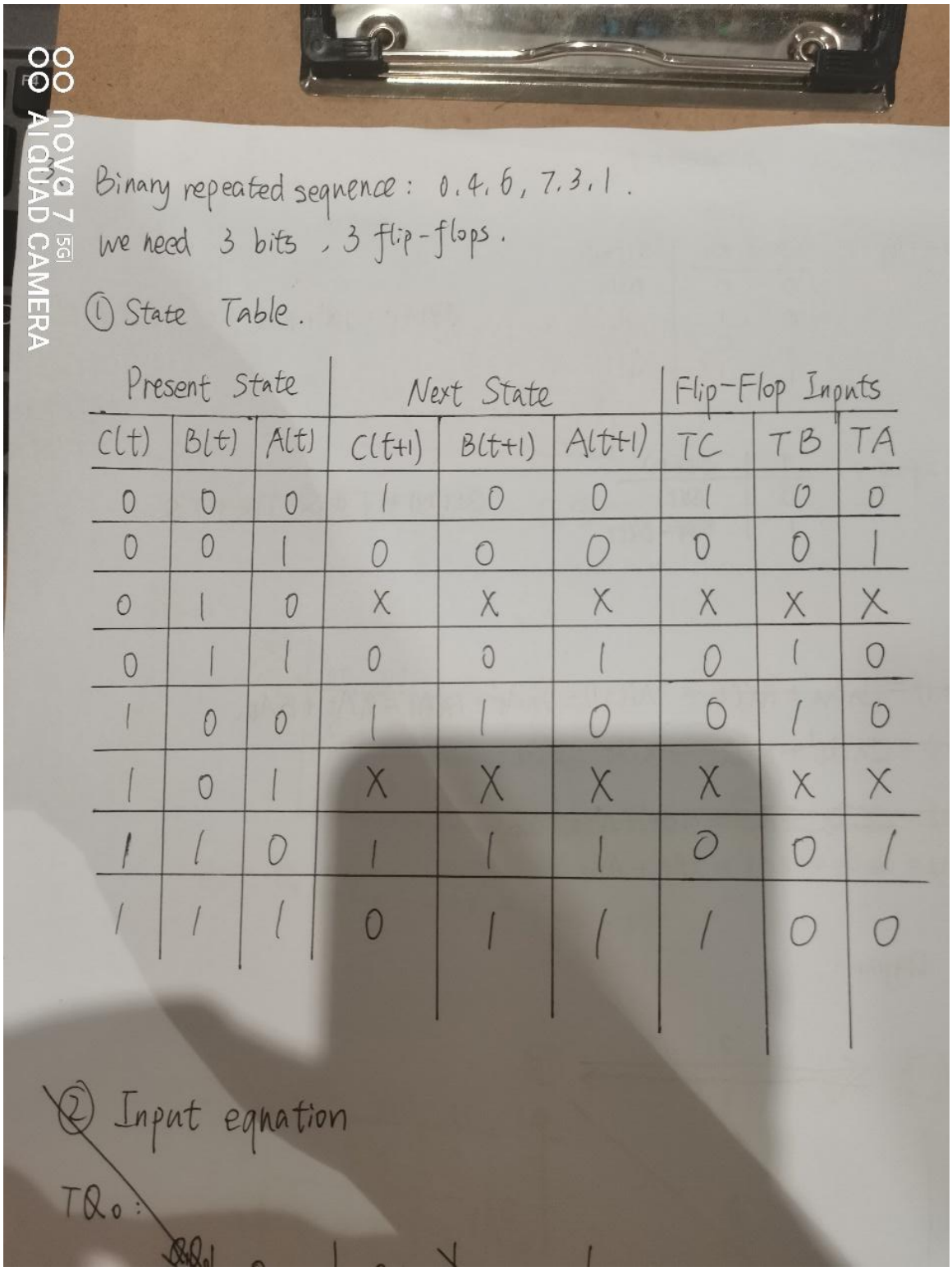
Student Name: 张嘉浩

Student ID: 12010423

PART 1: DIGITAL DESIGN THEORY

Provide your answers here:





Binary repeated sequence: 0, 4, 6, 7, 3, 1.
we need 3 bits, 3 flip-flops.

① State Table.

Present State			Next State			Flip-Flop Inputs		
C(t)	B(t)	A(t)	C(t+1)	B(t+1)	A(t+1)	TC	TB	TA
0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	0	0	1
0	1	0	X	X	X	X	X	X
0	1	1	0	0	1	0	1	0
1	0	0	1	1	0	0	1	0
1	0	1	X	X	X	X	X	X
1	1	0	1	1	1	0	0	1
1	1	1	0	1	1	1	0	0

② Input equation

TQ_0:

② Input Equations

TC_0:

C_2 \ C_1C_0	00	01	11	10
0	0	1	0	X
1	0	X	0	1

$TC_0 = C_1'C_0 + C_1C_0' = C_0 \oplus C_1$

TC_1:

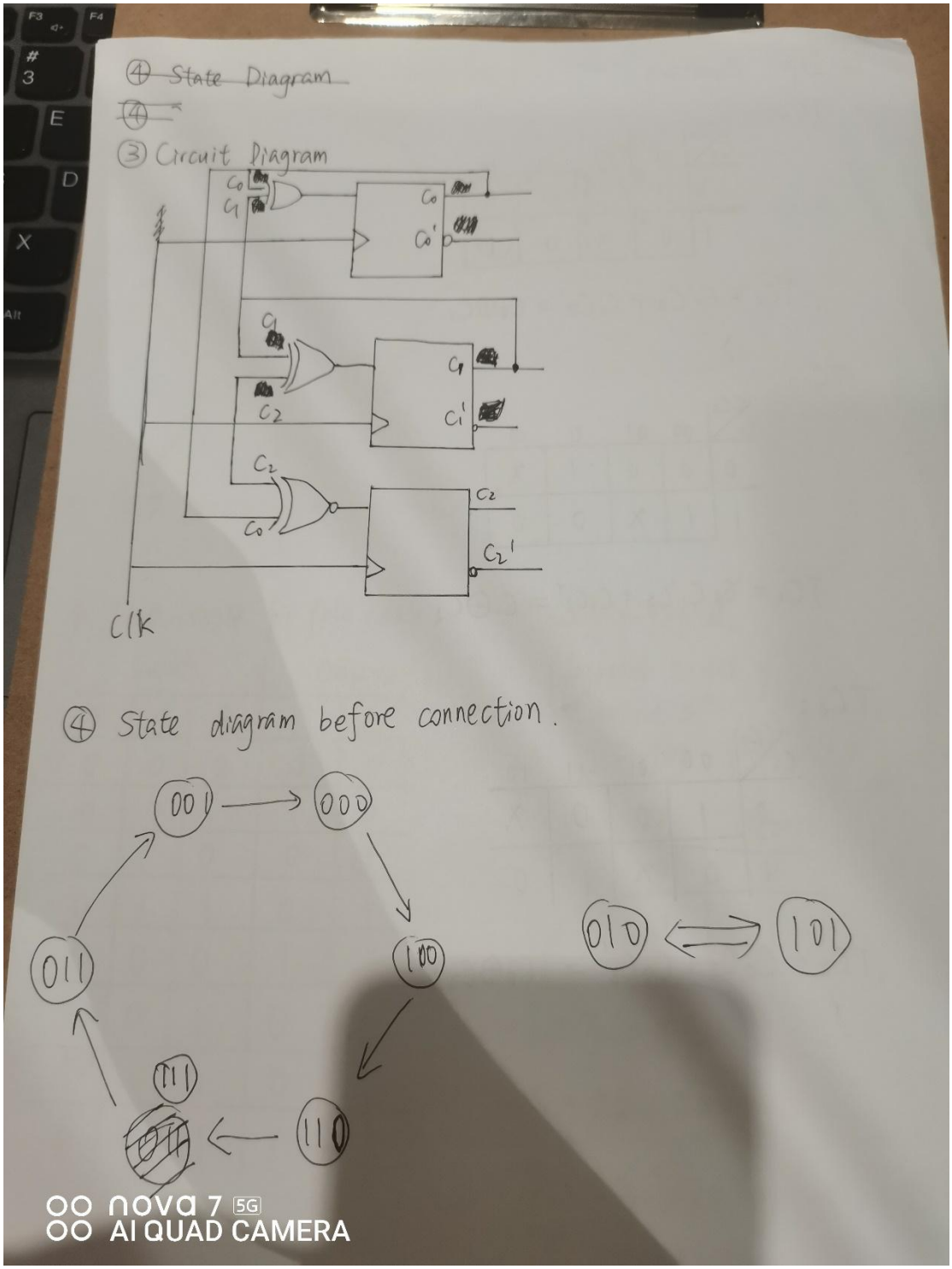
C_2 \ C_1C_0	00	01	11	10
0	0	0	1	X
1	1	X	0	0

$TC_1 = C_1'C_2 + C_1C_2' = C_1 \oplus C_2$

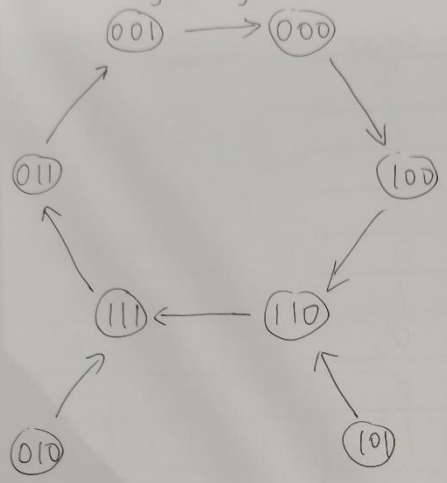
TC_2:

C_2 \ C_1C_0	00	01	11	10
0	1	0	0	X
1	0	X	1	0

$TC_2 = C_0C_2 + C_0'C_2' = (C_0 \oplus C_2)'$


④ State Diagram
④
③ Circuit Diagram
C0
C1
C0
C0'
C1
C2
C1
C1'
C2
C0
C2
C2'
Clk
④ State diagram before connection.
001
000
011
100
010
101
111
110
nova 7 5G
AI QUAD CAMERA

State Diagram after connection.



4. Truth-table for full adder.

Inputs			Outputs	
x	y	Q	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

" one flip-flop Q
two inputs x and y
one output s"
D flip-flop

D	Q(t+1)
0	0
1	1

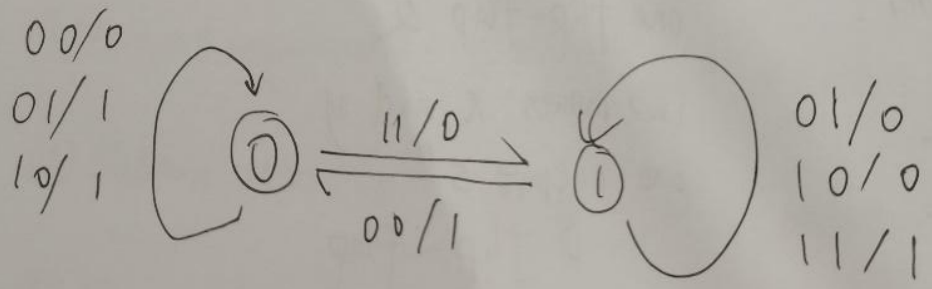
$S = X \oplus y \oplus Q$

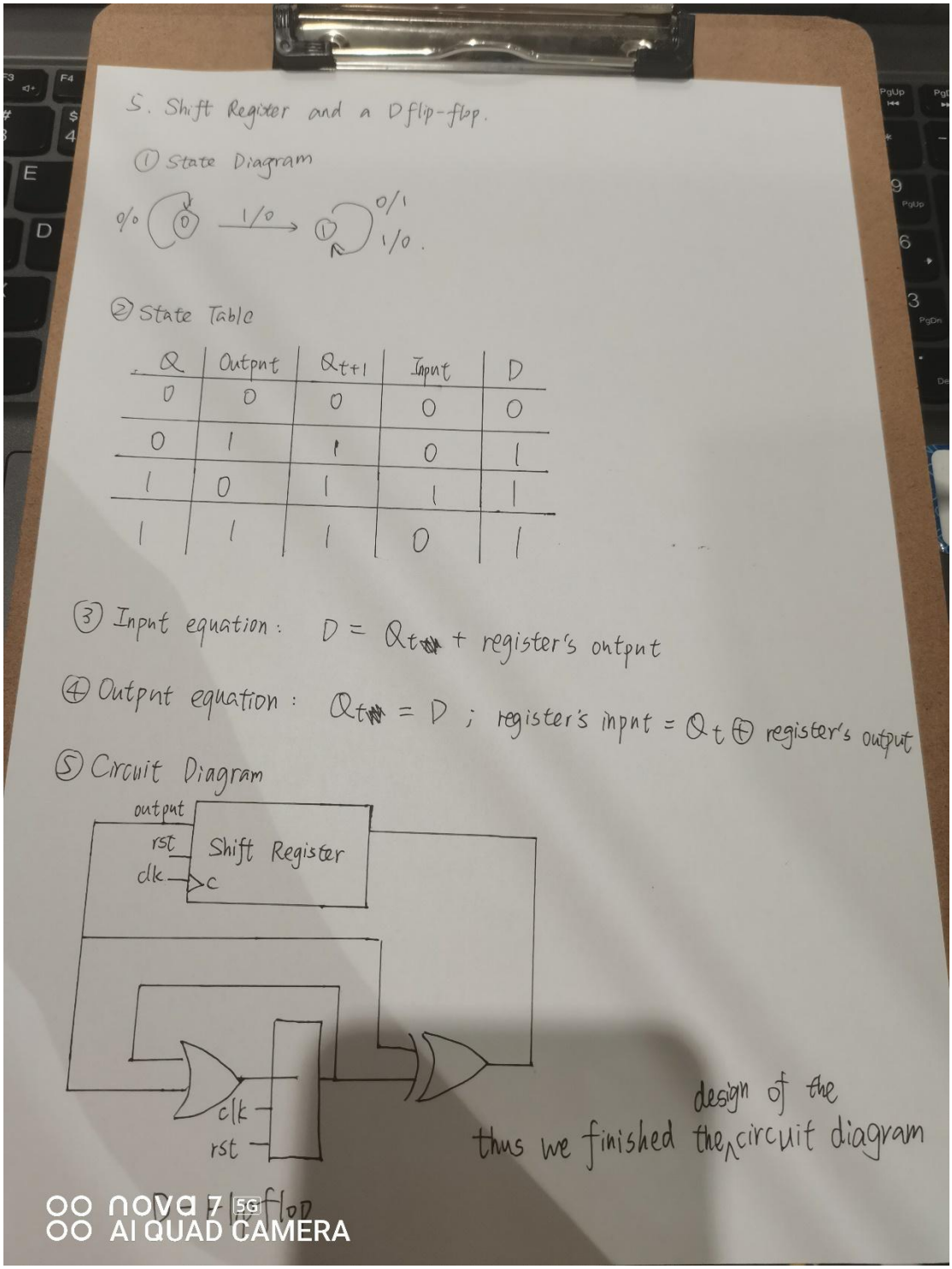
$C = xQ + xy + yQ$

$D = C = xQ + xy + yQ = Q(t+1)$

State Table :

Inputs				Output
x	y	Q(t)	Q(t+1)	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





PART 2: DIGITAL DESIGN LAB (TASK1)

DESIGN

(This part is always REQUIRED)

Describe the design of your system by providing the following information:

- *Verilog design (provide the Verilog code)*

```
`timescale 1ns / 1ps

module JK_FlipFlop(
    input clk,
    input rst_low,
    input J,
    input K,
    output reg Q
    );

    always@(posedge clk, negedge rst_low)
        if(~rst_low) begin
            Q <= 0;
        end
        else begin
            case ({J, K})
                2'b00 : Q = Q;
                2'b01 : Q = 0;
                2'b10 : Q = 1;
                2'b11 : Q = ~Q;
            endcase
        end
endmodule
```

-

```
1  `timescale 1ns / 1ps
2
3  module circuit(
4      input clk,
5      input rst_low,
6      input x,
7      output A,
8      output B
9      );
10
11     JK_FlipFlop JK1 (.clk(clk), .rst_low(rst_low), .J(~x), .K(B), .Q(A));
12     JK_FlipFlop JK2 (.clk(clk), .rst_low(rst_low), .J(x), .K(~A), .Q(B));
13
14
15 endmodule
```

-

SIMULATION

(This part is optional depending on the requirement of the lab task)

Describe how you build the test bench and do the simulation.

- *Using Verilog (provide the Verilog code)*
- *JK Flip-Flop simulation src*

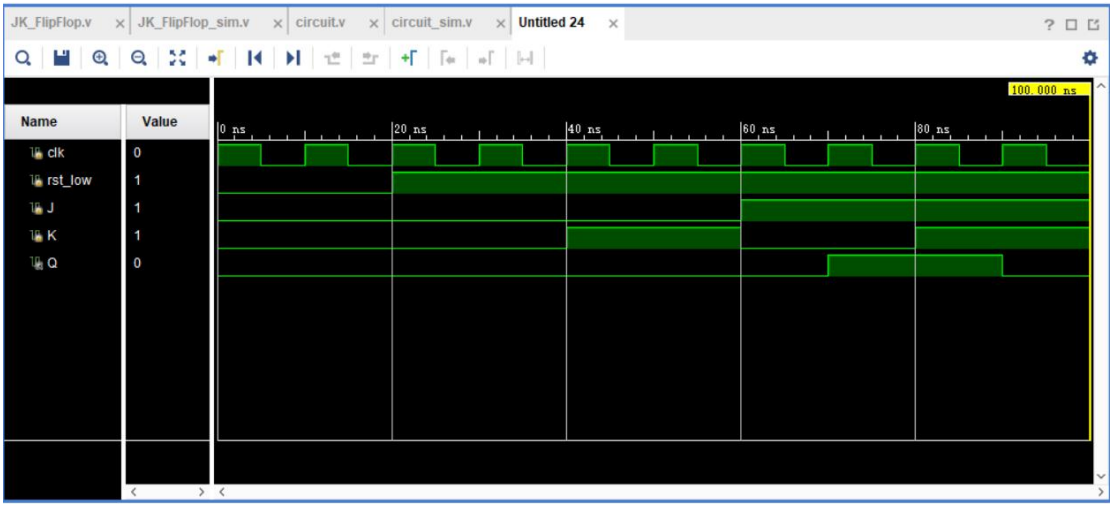
```
1  `timescale 1ns / 1ps
2
3  module JK_FlipFlop_sim();
4      reg clk = 1;
5      reg rst_low = 0;
6      reg J = 0;
7      reg K = 0;
8      wire Q;
9
10     JK_FlipFlop JKFF (.clk(clk), .rst_low(rst_low), .J(J), .K(K), .Q(Q));
11
12     always
13     #5 clk = ~clk;
```

```
15     initial begin
16         #20
17         rst_low = 1;
18         J <= 0;
19         K <=  0;
20         #20
21         J <= 0;
22         K <= 1;
23         #20
24         J <= 1;
25         K <= 0;
26         #20
27         J <= 1;
28         K <= 1;
29         #20
30     $finish;
31     end
32 endmodule
```

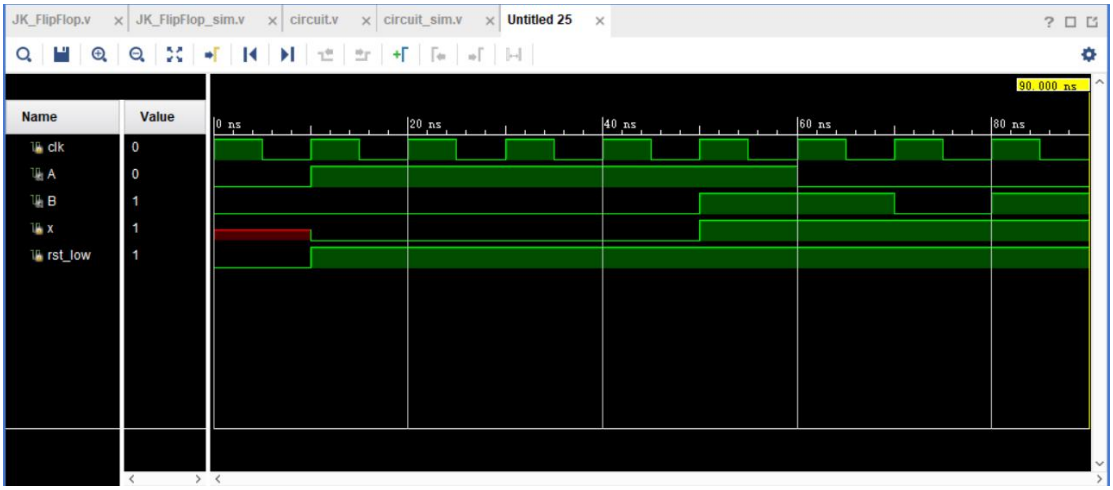
- *Circuit simulation src*

```
3   module circuit_sim( );
4     reg clk = 1;
5     wire A, B;
6     reg x;
7     reg rst_low;
8
9     always
10    #5 clk = ~clk;
11
12    circuit c(.clk(clk), .rst_low(rst_low), .x(x), .A(A), .B(B));
13
14    initial
15    begin
16      rst_low = 0;
17      #10
18      rst_low = 1;
19      x = 0;
20
21      #40
22      x = 1;
23      #40
24      $finish();
25    end
26  endmodule
```

- *Wave form of simulation result (provide screen shots)*
- *JK Flip-Flop simulation waveform*



- *Circuit simulation waveform*

- 

- *As far as you can see, the simulation result is exactly the same as it should be.*
- *As for the JK flip-flop, for instance, when rst_low is 1(at high electrical level), the process is on. When J is 0 and K is 0, we have Q = 0. When J is 0 and K is 1, we have Q = 0.*
- *As for the circuit, when rst_low is 1, A is 0 and B is 1, we have x = 0. When A is 0 and B is 0, we have x = 1.*

THE DESCRIPTION OF OPERATION

(This part is always REQUIRED unless you can get full marks for this lab task)

Describe the problem occurred while in the lab and your solution. Any suggestions are welcomed.

- *This is just a simple implementation of the knowledge we've learnt from the JK flipflop in sequential logic. Nevertheless, I still spent plenty of time just to write this simple code. The problem is that I cannot implement the knowledge I learnt into the Verilog code efficiently. I seek for the book and lecture slides for help, and finally managed to do it.*

PART 2: DIGITAL DESIGN LAB (TASK2)

DESIGN

- *Shift Register 74195 src*

```
module Shift_Register_74195 (
    input MR_n, CP, PE_n, J, K_n,
    input D3, D2, D1, D0,
    output reg Q3, Q2, Q1, Q0,
    output Q0_n
);
    wire K;
    assign Q0_n = ~Q0;
    assign K = ~K_n;
    always @(posedge CP, negedge MR_n) begin
        if(!MR_n)
            {Q3, Q2, Q1, Q0} <= 4'b0000;
        else
            if (!PE_n) begin
                {Q3, Q2, Q1, Q0} <= {D3, D2, D1, D0};
            end
            else
                case ({J, K})
                    2'b00:{Q3, Q2, Q1, Q0} <= {Q2, Q1, Q0, Q0};
                    2'b01:{Q3, Q2, Q1, Q0} <= {Q2, Q1, Q0, 1'b0};
                    2'b10:{Q3, Q2, Q1, Q0} <= {Q2, Q1, Q0, 1'b1};
                    2'b11:{Q3, Q2, Q1, Q0} <= {Q2, Q1, Q0, ~Q0};
                endcase
    end
endmodule
```

- *Johnson Counter src*

```
3  module Johnson_Counter(input clk,
4                         input rst_n,
5                         output [3:0] out);
6      reg [3:0] D = 0;
7      reg J, K_n;
8      reg PE_n = 1;
9      wire out0_n;
10
11     Shift_Register_74195 SR(.MR_n(rst_n), .CP(clk), .PE_n(PE_n), .J(J), .K_n(K_n),
12     .D3(D[3]), .D2(D[2]), .D1(D[1]), .D0(D[0]),
13     .Q3(out[0]), .Q2(out[1]), .Q1(out[2]), .Q0(out[3]), .Q0_n(out0_n));
14
15     always @(out[0]) begin
16       if (~out[0]) begin
17         J   = 1'b1;
18         K_n = 1'b1;
19       end
20       else begin
21         J   = 1'b0;
22         K_n = 1'b0;
23       end
24     end
25
26 endmodule
27
```

SIMULATION

Describe how you build the test bench and do the simulation.

- *Using Verilog (provide the Verilog code)*
- *Shift Register 74195 simulation src*

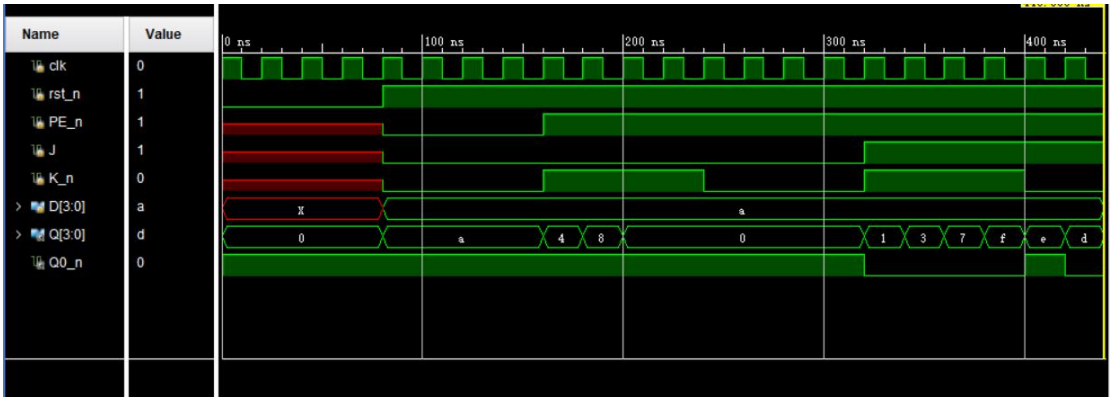
```
`timescale 1ns / 1ps

module Shift_Register_74195_sim();
    reg clk = 1;
    reg rst_n;
    reg PE_n;
    reg J, K_n;
    reg[3:0] D;
    wire[3:0] Q;
    wire Q0_n;

    Shift_Register_74195 sr(.MR_n(rst_n), .CP(clk), .PE_n(PE_n), .J(J), .K_n(K_n),
    .D3(D[3]), .D2(D[2]), .D1(D[1]), .D0(D[0]),
    .Q3(Q[3]), .Q2(Q[2]), .Q1(Q[1]), .Q0(Q[0]), .Q0_n(Q0_n));

    always #10
    clk = ~clk;
```

```
    initial
        begin
        rst_n = 0;
        #80
        D     = 4'b1010;
        rst_n = 1;
        PE_n  = 0;
        {J, K_n} = 2'b00;
        #80
        PE_n    = 1;
        {J,K_n} = 2'b01;
        #80
        {J,K_n} = 2'b00;
        #80
        {J,K_n} = 2'b11;
        #80
        {J,K_n} = 2'b10;
        #40
        $finish();
    end
endmodule
```

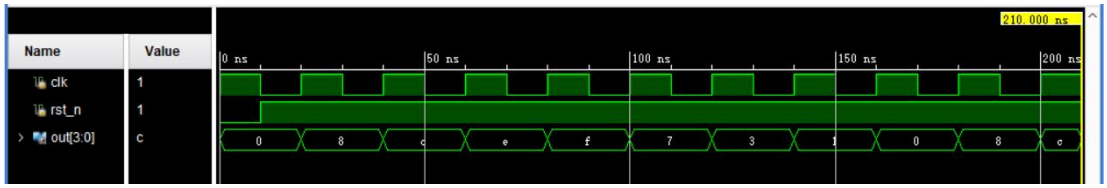
- *Johnson Counter simulation src*

```
1  `timescale 1ns / 1ps
2
3  module Johnson_Counter_sim();
4    reg clk = 1;
5    reg rst_n = 0;
6    wire[3:0] out;
7
8    Johnson_Counter JC(.clk(clk), .rst_n(rst_n), .out(out));
9
10   always
11   #10 clk = ~clk;
12
13   initial
14   begin
15     #10
16     rst_n = 1;
17     #200
18     $finish();
19   end
20
21 endmodule
```

- *Wave form of simulation result (provide screen shots)*
- *Shift Register 74195 simulation waveform*



- *Johnson Counter simulation waveform*



- *As far as you can see, the waveform result is exactly the same as it theoretically should be.*
- *For instance, as for the shift register 74195, when rst_n is 1, PE_n is 1, J is 1 and K_n is 0. We have D = a, Q = d.*
- *As for the Johnson Counter, when rst_n is 1, the output c changes from 0 to 8, c, e, f, 7, 3, 1 and 0 again as a circulation, as the simulation time goes.*

THE DESCRIPTION OF OPERATION

Describe the problem occurred while in the lab and your solution. Any suggestions are welcomed.

- *The main problem is that I'm not quite familiar with this part of knowledge. Thanks to the code in lab13, otherwise I'll spent much more time figuring out how to write shift register 74195 and Johnson Counter code. I should review the lecture slides Prof. G use in class and refer to the textbook more often.*

PART 2: DIGITAL DESIGN LAB (TASK3)

DESIGN

Describe the design of your system by providing the following information:

Top module

```
module top(
    input clk,
    input rst,
    input switch,
    output [7:0] seg_en,
    output [7:0] seg_out
);
    wire [2:0] scan_cnt;
    freq_div fd(.clk(clk), .rst(rst), .scan_cnt(scan_cnt));
    segtube st(.clk(clk), .scan_cnt(scan_cnt), .rst(rst), .switch(switch), .seg_en(seg_en), .seg_out(seg_out));

endmodule
```

Frequency divider module

```
module freq_div(
    input clk,
    input rst,
    output reg [2:0] scan_cnt
);

reg clk_seg;
reg [19:0] cnt;

parameter period = 250000;  //400Hz

always @(posedge clk, posedge rst)  // 分频为clk_seg
    begin
        if (rst)   // 复位
            begin
            cnt <= 0;  //cnt归0
            clk_seg <= 0;  //clk归0
            end
        else begin
            if (cnt == (period >> 1) - 1)     // 右移除以2为半个周期，当半个周期结束
                begin
                clk_seg <= ~clk_seg;   // clk取反，两次取反为一个周期
                cnt <= 0;      // cnt归0
                end
            else
                cnt <= cnt + 1; // cnt递增
        end
    end
```

```
always@(posedge clk_seg, posedge rst)   //根据clk_seg改变scan_cnt
  begin
    if (rst)   //复位
      scan_cnt <= 0; //scan_cnt归0
    else begin
      scan_cnt <= scan_cnt + 1;  //scan_cnt递增
      if (scan_cnt == 3'd7)  //scan_cnt一个周期完成
        scan_cnt <= 0;     //scan_cnt归0
    end
  end
endmodule
```

7-segtube module

```
`timescale 1ns / 1ps

module segtube(
  input clk,
  input [2:0] scan_cnt,
  input rst,
  input switch,
  output reg[7:0] seg_en,
  output reg[7:0] seg_out
);

  always @(scan_cnt) begin
    case(scan_cnt)
      0: seg_en = 8'b0111_1111;
      1: seg_en = 8'b1011_1111;
      2: seg_en = 8'b1101_1111;
      3: seg_en = 8'b1110_1111;
      4: seg_en = 8'b1111_0111;
      5: seg_en = 8'b1111_1111;
      6: seg_en = 8'b1111_1111;
      7: seg_en = 8'b1111_1111;
      default: seg_en = 8'b1111_1111;
    endcase
  end
```

```
26  always @(scan_cnt) begin
27      if(switch) begin
28          case(scan_cnt)
29              0 : seg_out <= 8'b10100100;
30              1 : seg_out <= 8'b11000000;
31              2 : seg_out <= 8'b10100100;
32              3 : seg_out <= 8'b11111001;
33              4 : seg_out <= 8'b10001110;
34              default: seg_out <= 8'b11111111;
35          endcase
36        end
37      else begin
38          case(scan_cnt)
39            0 : seg_out <= 8'b11000110;
40            1 : seg_out <= 8'b10010010;
41            2 : seg_out <= 8'b10100100;
42            3 : seg_out <= 8'b11000000;
43            4 : seg_out <= 8'b11111000;
44            default: seg_out <= 8'b11111111;
45          endcase
46      end
47    end
48  endmodule
```

CONSTRAINT FILE AND THE TESTING

- *Constraint File xdc*

```
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst}]
set_property IOSTANDARD LVCMOS33 [get_ports {switch}]
```

```
set_property PACKAGE_PIN C19 [get_ports {seg_en[0]}]
set_property PACKAGE_PIN E19 [get_ports {seg_en[1]}]
set_property PACKAGE_PIN D19 [get_ports {seg_en[2]}]
set_property PACKAGE_PIN F18 [get_ports {seg_en[3]}]
set_property PACKAGE_PIN E18 [get_ports {seg_en[4]}]
set_property PACKAGE_PIN B20 [get_ports {seg_en[5]}]
set_property PACKAGE_PIN A20 [get_ports {seg_en[6]}]
set_property PACKAGE_PIN A18 [get_ports {seg_en[7]}]

set_property PACKAGE_PIN F15 [get_ports {seg_out[0]}]
set_property PACKAGE_PIN F13 [get_ports {seg_out[1]}]
set_property PACKAGE_PIN F14 [get_ports {seg_out[2]}]
set_property PACKAGE_PIN F16 [get_ports {seg_out[3]}]
set_property PACKAGE_PIN E17 [get_ports {seg_out[4]}]
set_property PACKAGE_PIN C14 [get_ports {seg_out[5]}]
set_property PACKAGE_PIN C15 [get_ports {seg_out[6]}]
set_property PACKAGE_PIN E13 [get_ports {seg_out[7]}]

set_property PACKAGE_PIN Y18 [get_ports {clk}]
set_property PACKAGE_PIN P20 [get_ports {rst}]
set_property PACKAGE_PIN W4 [get_ports {switch}]
```

- 
XILINX
MINISYS
ARTIX
E-ELEMENTS
nova 7 5G
AI QUAD CAMERA
- .

XILINX
MINISYS
ARTIX
E-ELEMENTS
nova 7 5G
AI QUAD CAMERA

THE DESCRIPTION OF OPERATION

Describe the problem occurred while in the lab and your solution. Any suggestions are welcomed.

- *In charge of the 7segtube display section in my digital design project, I complete this task without much difficulty. The challenge for me is modular programming. I write the display of 7segtube all in one module, without separating the frequency divider and display part. Modular programming is absolutely a nice habit and an important programming thought that I must be skilled.*