Lab10--Typical Design of Synchronous Sequential Circuit

姓名：潘子曰 学号：3180105354 专业：计算机科学与技术
课程名称：逻辑与计算机设计基础实验 同组学生姓名：张佳文
试验时间：2019-11-28 实验地点：紫金港东4-509 指导老师：洪奇军

1. Objectives & Requirements

1. Master the working principle and design method of typical **synchronous sequential circuit**.
2. Master the describing and implementing method of **finite state machine**.
3. Master **state diagrams**, **state functions** and **activation functions triggers**.
4. Master how to **design, simulate and debug** a finite state machine with FPGA.

2. Contents & Principles

2.1 Tasks

1. Design a 4-bit synchronous counter based on state functions with schematic.
2. Design a **32-bit synchronous two-way counter** with HDL behavioral code.
3. Integrate to experiment environment (ALU).

2.2 Principles

4-bit Synchronous Counter State Function

State	Q_A	Q_B	Q_C	Q_D	Q_A^{n+1}	Q_B^{n+1}	Q_C^{n+1}	Q_D^{n+1}
0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	1	0	0
2	0	1	0	0	1	1	0	0
3	1	1	0	0	0	0	1	0
4	0	0	1	0	1	0	1	0
5	1	0	1	0	0	1	1	0
6	0	1	1	0	1	1	1	0
7	1	1	1	0	0	0	0	1
8	0	0	0	1	1	0	0	1
9	1	0	0	1	0	1	0	1
10	0	1	0	1	1	1	0	1
11	1	1	0	1	0	0	1	1
12	0	0	1	1	1	0	1	1
13	1	0	1	1	0	1	1	1
14	0	1	1	1	1	1	1	1
15	1	1	1	1	0	0	0	0

$$D_A = \overline{Q_A}$$

$$D_B = \overline{Q_A}Q_B + Q_A\overline{Q_B}$$

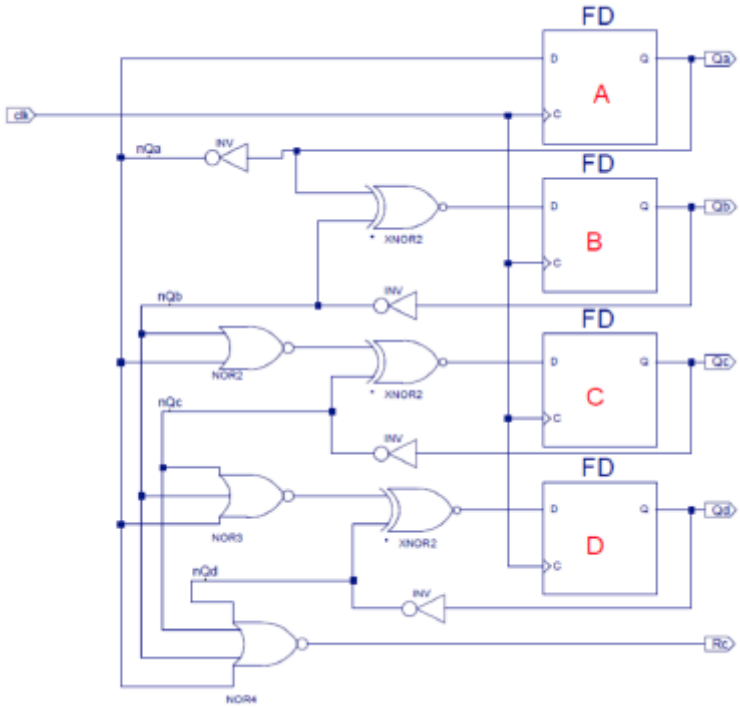
$$D_C = \overline{Q_A}Q_C + \overline{Q_B}Q_C + Q_AQ_B\overline{Q_C}$$

$$D_D = \overline{Q_A}Q_D + \overline{Q_B}Q_D + \overline{Q_C}Q_D + Q_AQ_BQ_C\overline{Q_D}$$

Plus a 4-bit carry output:

$$R_C = \overline{\overline{Q_A} + \overline{Q_B} + \overline{Q_C} + \overline{Q_D}}$$

4-bit Synchronous Counter Schematic



Gate Level Description

Examples:

```
INV A(.I(a), .O(na));
NOR2 G1(.I0(a), .I1(b), .O(na_b));
NOR3...
NOR4...
XNOR2...
FD...   //D Trigger
NAND2...
```

3. Major Experiment Instruments

Equipment
Computer (Intel Core i7-9750H, 16GB memory)
Sword circuit design box
Xilinx ISE 14.7

4. Experiment Procedure

Task: FSM

- Name the project **FSM**
- Implement 4-bit counter
  - Design logic circuit based on activation functions
- Implement 32-bit two-way counter
  - By behavioral description
  - With direction control signal ***s***
  - With load signal which controls loading value from A_i
- Integrate the two modules into I/O framework

4-bit counter by Gate Level Description

```
`timescale 1ns / 1ps
module counter_4bit(
    input clk,
    output Qa,
    output Qb,
    output Qc,
    output Qd,
    output Rc
 );

wire Da, Db, Dc, Dd;
wire Nor_nQa_nQb, Nor_nQa_nQb_nQc;
wire nQa, nQb, nQc, nQd;

//define initial value of the D type Flip-Flop
FD  FFDA(.C(clk), .D(Da), .Q(Qa)),
    FFDB(.C(clk), .D(Db), .Q(Qb)),
    FFDC(.C(clk), .D(Dc), .Q(Qc)),
    FFDD(.C(clk), .D(Dd), .Q(Qd));

defparam FFDA.INIT = 1'b0;
defparam FFDB.INIT = 1'b0;
defparam FFDC.INIT = 1'b0;
defparam FFDD.INIT = 1'b0;

INV     GQa(.I(Qa), .O(nQa)),
        GQb(.I(Qb), .O(nQb)),
        GQc(.I(Qc), .O(nQc)),
        GQd(.I(Qd), .O(nQd));

assign Da = nQa;

XNOR2   ODb(.I0(Qa), .I1(nQb), .O(Db)),
        ODc(.I0(Nor_nQa_nQb), .I1(nQc), .O(Dc)),
        ODd(.I0(Nor_nQa_nQb_nQc), .I1(nQd), .O(Dd));

NOR4 ORc(.I0(nQa), .I1(nQb), .I2(nQc), .I3(nQd), .O(Rc));
NOR2 G1(.I0(nQa), .I1(nQb), .O(Nor_nQa_nQb));
NOR3 G2(.I0(nQa), .I1(nQb), .I2(nQc), .O(Nor_nQa_nQb_nQc));
```

```
endmodule

//module counter_4bit(
//  input clk,
//  output Qa,
//  output Qb,
//  output Qc,
//  output Qd,
//  output Rc
//    );
//
//  wire Da, Db, Dc, Dd, nQa, nQb, nQc, nQc, Rc;
//  reg Qa,Qb,Qc,Qd;
//
//  assign Da = nQa;
//  assign Db = ~(nQa^nQb);
//  assign Dc = ~( (~(nQa| nQb)) ^ nQc);
//  assign Dd= ~((~(nQa| nQb| nQc)) ^ nQd);
//  assign Rc= ~(nQa| nQb| nQc| nQd));
//
//  always @ (posedgeclk)
//      if (rst) {Qa,Qb,Qc,Qd} <= 4'b0000;//同步清零
//      else begin
//          Qa<= Da;
//          Qb<-Db;
//          Qc <= Dc;
//          Qd<= Dd;
//      end
//endmodule
```

Simulation Code for 4-bit counter

```
`timescale 1ns / 1ps
module counter_4bit_test;

    // Inputs
    reg clk;

    // Outputs
    wire Qa;
    wire Qb;
    wire Qc;
    wire Qd;
    wire Rc;

    // Instantiate the Unit Under Test (UUT)
    counter_4bit uut (
        .clk(clk),
        .Qa(Qa),
        .Qb(Qb),
        .Qc(Qc),
        .Qd(Qd),
        .Rc(Rc)
    );

    initial begin
        // Initialize Inputs
```

```
        clk = 0;
        forever #50 clk<=~clk;
    end

endmodule
```

32-bit Synchronous Two-way Counter by Behavioral Description

☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆

I made an advance in this module that I set a single-bit register **s** as the indication for direction. When there is a signal from **input s_ctrl**, s is then inverted.

By doing this, there won't be situations that we have to keep pressing BTN(0) to set the counting direction.

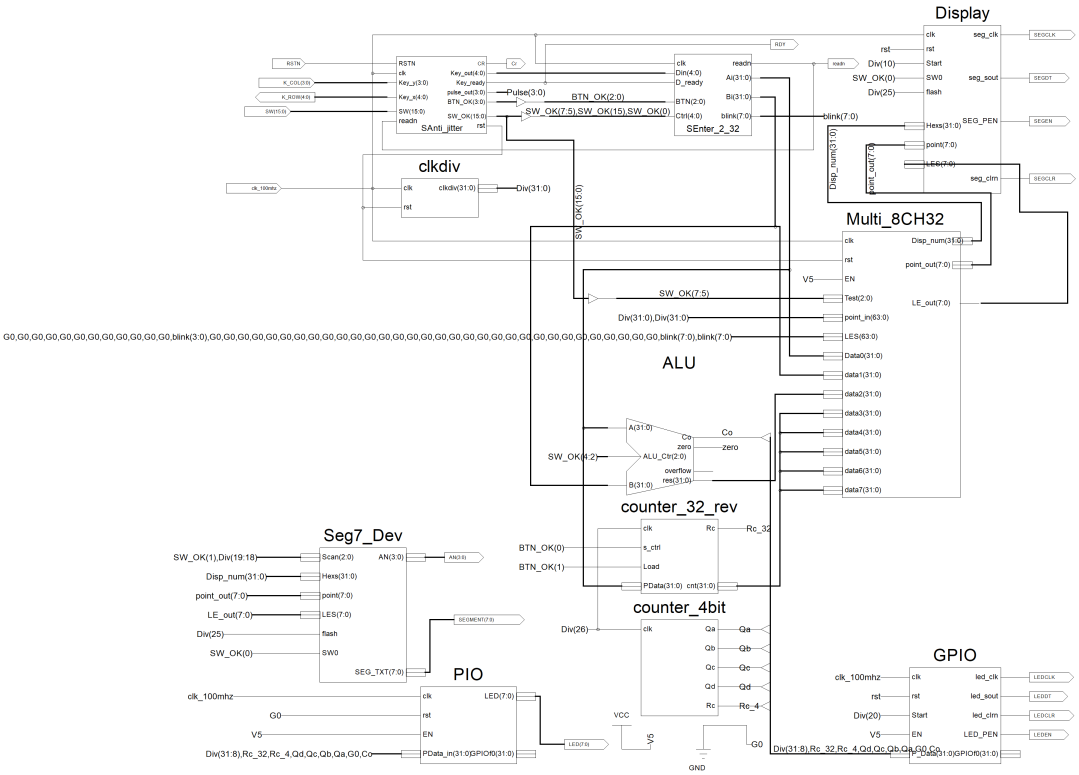
```
`timescale 1ns / 1ps
module counter_32_rev(
    input clk,
    input s_ctrl,
    input Load,
    input [31:0] PData,
    output reg[31:0] cnt,
    output reg Rc
);

    reg zero = 32'h00000000, full = 32'hffffffff, s=0;

    always@* if(s_ctrl) s<=~s;

    always@(posedge clk) begin
        if(Load) cnt<=PData;
        else begin
            if(s) cnt<=cnt+1;
            else cnt<=cnt-1;

            if(~s&~(zero|cnt) | s&(full&cnt)) Rc <= 1;
            else Rc <= 0;
        end
    end
endmodule
```

Simulation Code for 32-bit Two-way Synchronous Counter

```
`timescale 1ns / 1ps
module counter_32bit_test;

    // Inputs
    reg clk;
    reg s;
    reg Load;
    reg [31:0] PData;

    // Outputs
    wire [31:0] cnt;
    wire Rc;
```

```
13
14      // Instantiate the Unit Under Test (UUT)
15      counter_32_rev uut (
16          .clk(clk),
17          .s(s),
18          .Load(Load),
19          .PData(PData),
20          .cnt(cnt),
21          .Rc(Rc)
22      );
23
24      integer i=0;
25      initial begin
26          // Initialize Inputs
27          clk = 0;
28          s = 1;
29          Load = 1;
30          PData = 0;
31
32          #100;
33          Load = 0;
34
35          #300;
36          s = 0;
37
38          #200;
39          Load = 1;
40      end
41  always@*
42      for(i=0;i<20;i=i+1) begin
43              #50;
44              clk <= ~clk;
45      end
46  endmodule
```

Physical Test in I/O Framework



- This time we have to do some change to several input drives. Check the schematic carefully

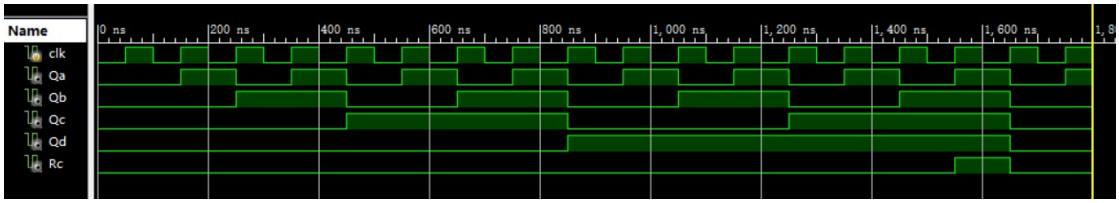
UCF

```
NET "clk_100mhz"    LOC=AC18    |    IOSTANDARD=LVCMOS18;
NET "clk_100mhz"    TNM_NET=TM_CLK;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK" 10 ns HIGH 50%;

NET "RSTN"  LOC=W13 |   IOSTANDARD=LVCMOS18;

NET "K_ROW[0]"  LOC=V17 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[1]"  LOC=W18 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[2]"  LOC=W19 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[3]"  LOC=W15 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[4]"  LOC=W16 |   IOSTANDARD=LVCMOS18;

NET "K_COL[0]"  LOC=V18 |   IOSTANDARD=LVCMOS18;
NET "K_COL[1]"  LOC=V19 |   IOSTANDARD=LVCMOS18;
NET "K_COL[2]"  LOC=V14 |   IOSTANDARD=LVCMOS18;
NET "K_COL[3]"  LOC=W14 |   IOSTANDARD=LVCMOS18;

NET "readn" LOC=U21 |   IOSTANDARD=LVCMOS33;
NET "RDY"   LOC=U22 |   IOSTANDARD=LVCMOS33;
NET "CR"    LOC=V22 |   IOSTANDARD=LVCMOS33;

NET "SEGCLK"    LOC=M24 |   IOSTANDARD=LVCMOS33;
NET "SEGCLR"    LOC=M20 |   IOSTANDARD=LVCMOS33;
NET "SEGDT" LOC=L24 |   IOSTANDARD=LVCMOS33;
NET "SEGEN" LOC=R18 |   IOSTANDARD=LVCMOS33;

NET "LEDCLK"    LOC=N26 |   IOSTANDARD=LVCMOS33;
NET "LEDCLR"    LOC=N24 |   IOSTANDARD=LVCMOS33;
NET "LEDDT" LOC=M26 |   IOSTANDARD=LVCMOS33;
NET "LEDEN" LOC=P18 |   IOSTANDARD=LVCMOS33;
```

```
31
32 NET "SW[0]" LOC=AA10    |   IOSTANDARD=LVCMOS15;
33 NET "SW[1]" LOC=AB10    |   IOSTANDARD=LVCMOS15;
34 NET "SW[2]" LOC=AA13    |   IOSTANDARD=LVCMOS15;
35 NET "SW[3]" LOC=AA12    |   IOSTANDARD=LVCMOS15;
36 NET "SW[4]" LOC=Y13 |   IOSTANDARD=LVCMOS15;
37 NET "SW[5]" LOC=Y12 |   IOSTANDARD=LVCMOS15;
38 NET "SW[6]" LOC=AD11    |   IOSTANDARD=LVCMOS15;
39 NET "SW[7]" LOC=AD10    |   IOSTANDARD=LVCMOS15;
40 NET "SW[8]" LOC=AE10    |   IOSTANDARD=LVCMOS15;
41 NET "SW[9]" LOC=AE12    |   IOSTANDARD=LVCMOS15;
42 NET "SW[10]"    LOC=AF12    |   IOSTANDARD=LVCMOS15;
43 NET "SW[11]"    LOC=AE8 |   IOSTANDARD=LVCMOS15;
44 NET "SW[12]"    LOC=AF8 |   IOSTANDARD=LVCMOS15;
45 NET "SW[13]"    LOC=AE13    |   IOSTANDARD=LVCMOS15;
46 NET "SW[14]"    LOC=AF13    |   IOSTANDARD=LVCMOS15;
47 NET "SW[15]"    LOC=AF10    |   IOSTANDARD=LVCMOS15;
48
49 NET "SEGMENT[0]"    LOC=AB22    |   IOSTANDARD=LVCMOS33;
50 NET "SEGMENT[1]"    LOC=AD24    |   IOSTANDARD=LVCMOS33;
51 NET "SEGMENT[2]"    LOC=AD23    |   IOSTANDARD=LVCMOS33;
52 NET "SEGMENT[3]"    LOC=Y21 |   IOSTANDARD=LVCMOS33;
53 NET "SEGMENT[4]"    LOC=W20 |   IOSTANDARD=LVCMOS33;
54 NET "SEGMENT[5]"    LOC=AC24    |   IOSTANDARD=LVCMOS33;
55 NET "SEGMENT[6]"    LOC=AC23    |   IOSTANDARD=LVCMOS33;
56 NET "SEGMENT[7]"    LOC=AA22    |   IOSTANDARD=LVCMOS33;
57
58 NET "AN[0]" LOC=AD21    |   IOSTANDARD=LVCMOS33;
59 NET "AN[1]" LOC=AC21    |   IOSTANDARD=LVCMOS33;
60 NET "AN[2]" LOC=AB21    |   IOSTANDARD=LVCMOS33;
61 NET "AN[3]" LOC=AC22    |   IOSTANDARD=LVCMOS33;
62
63 NET"LED[0]"LOC=W23  | IOSTANDARD=LVCMOS33 ;#D1
64 NET"LED[1]"LOC=AB26 | IOSTANDARD=LVCMOS33 ;#D2
65 NET"LED[2]"LOC=Y25  | IOSTANDARD=LVCMOS33 ;#D3
66 NET"LED[3]"LOC=AA23 | IOSTANDARD=LVCMOS33 ;#D4
67 NET"LED[4]"LOC=Y23  | IOSTANDARD=LVCMOS33 ;#D5
68 NET"LED[5]"LOC=Y22  | IOSTANDARD=LVCMOS33 ;#D6
69 NET"LED[6]"LOC=AE21 | IOSTANDARD=LVCMOS33 ;#D7
70 NET"LED[7]"LOC=AF24  | IOSTANDARD=LVCMOS33 ;#D8
```

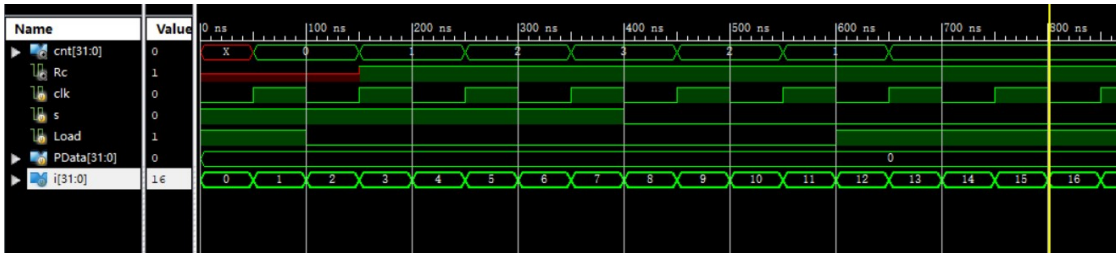
5. Experiment Results and Analyses

Simulation for 4-bit Counter



Clearly, the state variable went rightfully as the state table has described.

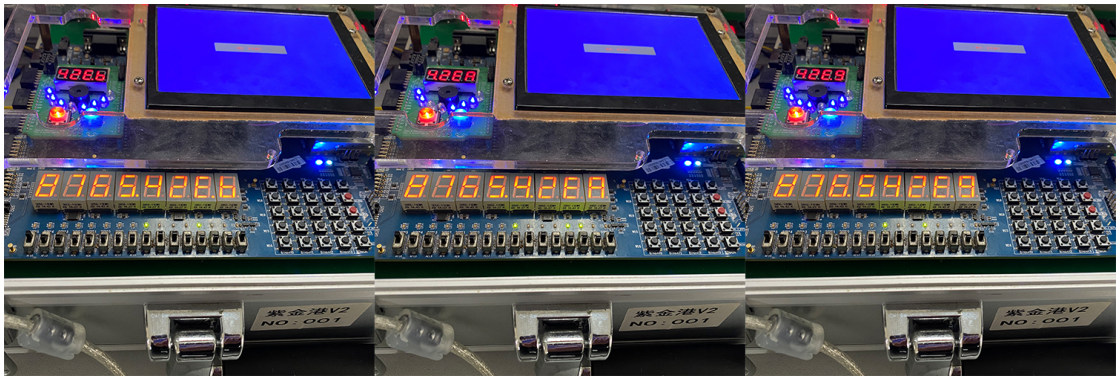
Simulation for 32-bit Two-way Counter



- At **400ns** when s is set from 1 to 0, **cnt** then start to decrement.
- At **50ns** when **Load** signal is set 1, **cnt** load data from PData[31:0], which is 32'h00000000.

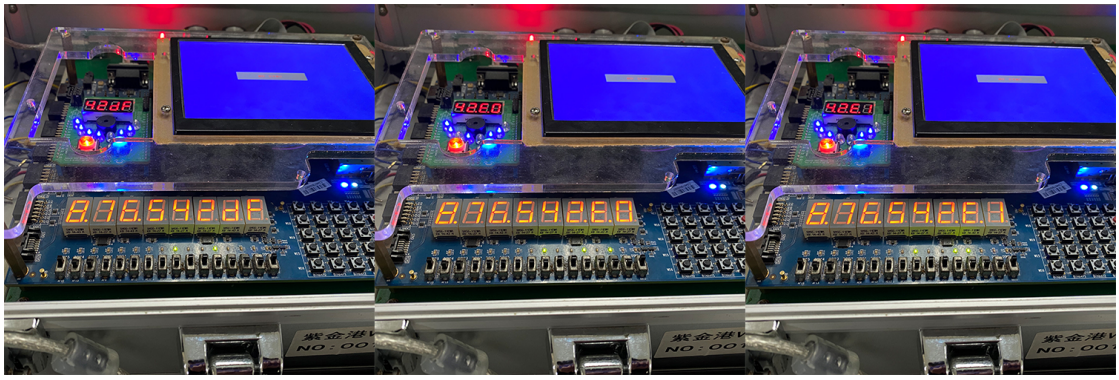
Physical Test

Without doing anything, the counter is decrementing itself automatically:



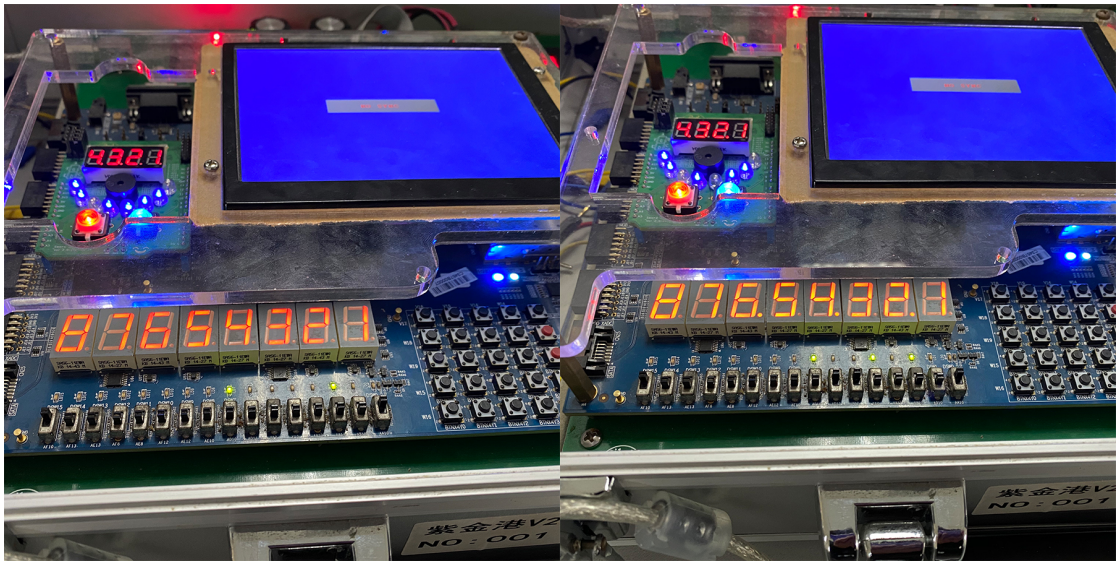
Decrementing

I pressed the BTN(0), which changes the counting direction. Then the counters went like this (incrementing):



Incrementing

Then I tested the **Loading** module: I entered a number into channel $\mathbf{A_i}$ which is controlled by **SW[7:5]=000**, and then pressed **BTN(1)**, which controls the loading signal for the counter. After change SW[7:5] to 011, the results went like this:



Load Data

Thus I have successfully verified my design of **32-bit bidirectional counters**.

6. Discussion and Conclusion

Complication I met

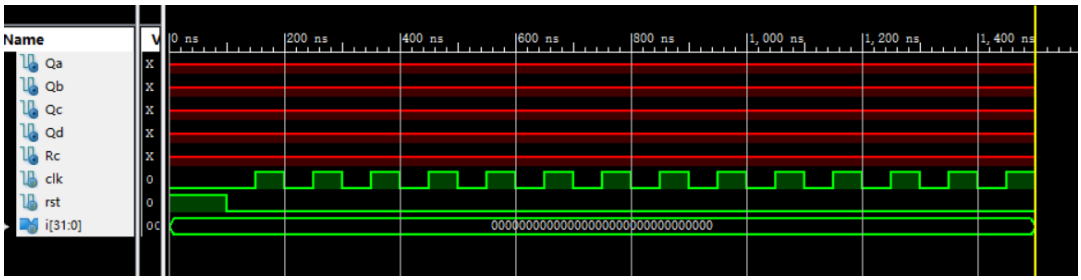
Admittedly, this experiment didn't go so hard that I just finished it so readily. Yet I did meet with some problems:

- `forever` cannot run parallelly within simulation test;
- cannot enter data into channel one when some change to BTN(0) has been done;

I wrote the following code to implement simulation **clock** division in simulation test at first:

```
1 forever #50 clk <= ~clk;
```

Yet when I tried to do simulation test, result went like this:



Wrong!

So, it is much reasonable to speculate that `forever` code cannot run parallelly with the previous codes.

The second error is much complicated. I made some change to the **direction switch control** module: using a register to store the current direction, if a signal from **BTN(0)** has been received, invert the register value.

> However, when I downloaded on the experiment box and did physical test, I found that **entering data into channel A** has been disabled.
>
> This situation was finally solved by Professor. Shi. **He checked the switches and toggled SW(15) several times**. Then all went rightfully, with my redesign on the BTN(0) perfectly improving the controlling performance.

Feelings

> On top of the problems I met, I also help my classmates and roommates solve their problems, like errors in behavioral description or simulation test.
>
> With our ISE framework being more complete, I'm getting further interested in computing systems and hardware design. Seeing my improvements is such a touchable happiness with which I would always please myself. ♡