Lab12--Design and Utilization of Shift Register

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：2019-12-12 实验地点：紫金港东4-509 指导老师：洪奇军

1. Objectives & Requirements

> - Master the working principle and design of shift register
> - Master the way of transferring between **serial and parallel data**
> - Master the way of **synchronous serial transmission**

2. Contents & Principles

Tasks

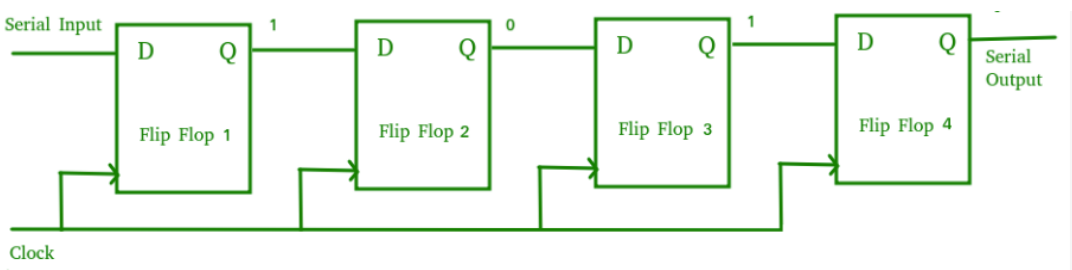
> - Design **4-bit bi-directional shift register with parallel input** by structural description.
> - Design **32-bit bi-directional shift register** and do physical test.
> - Design **bi-directional shift register with general bit width** and implement **parallel-to-serial** data transmission.
>   - Replace `P2S.ngc` module on the I/O framework.

Principles

Shift register

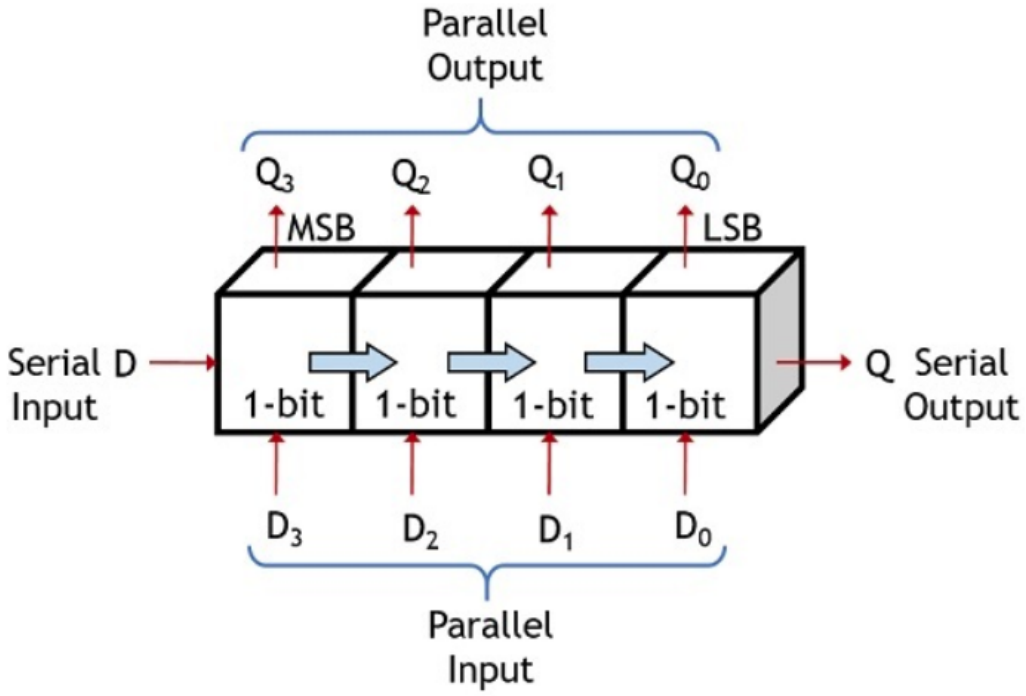
A **shift register** is a cascade of flip flops, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, "shifting in" the data present at its input and 'shifting out' the last bit in the array.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as "serial-in, parallel-out" (**SIPO**), as "parallel-in, serial-out" (**PISO**) or as "parallel in, parallel out" (**PIPO**).



common shift register

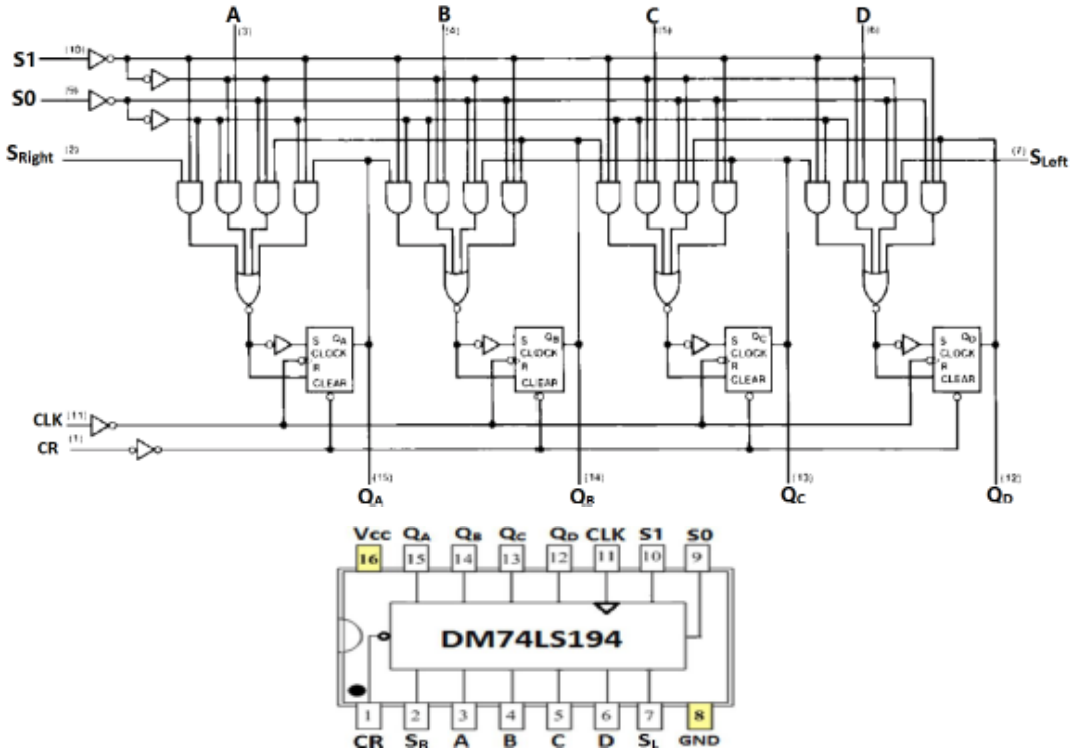
Shift register with parallel data input



This is an illustration of a shift register with parallel data input. The register has multiple way of inputting and outputting:

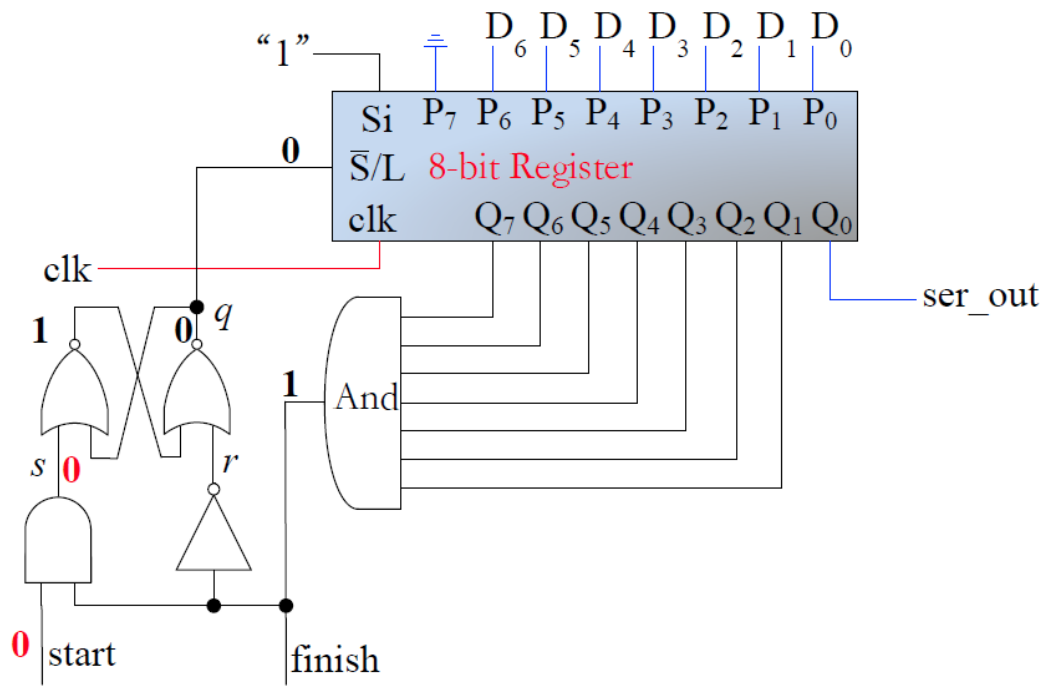
- Serial input
- Parallel input
- Serial output
- Parallel output

Bi-directional general shift register



DM74LS190 general shift register. Structural description with 4 MB_DFF modules.

Parallel to serial



The whole procedure can be concluded as the following:

- When the module is ready to start, parallel data is all 1s because the serial input is set to "1". At this state, **S/L** port is set to 0, which indicates moving data and the **finish** signal is always on.
- When the **start** signal is set to 1, **S/L** is set to 1 which indicates **parallel inputting**. and a "0" from **P7** is added to the parallel array. At this time, the parallel output is not all 1s, causing a negative edge in the **8-input AND** gate. Thus **S/L** is set to 0 again. On top of that, **finish** signal becomes 0.
- The shift register output the array serially after certain clock cycle. When all the data has been output serially, the added **0** is output and the parallel is all 1s again, making **finish** signal 1.

3. Major Experiment Instruments

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

4. Experiment Procedure

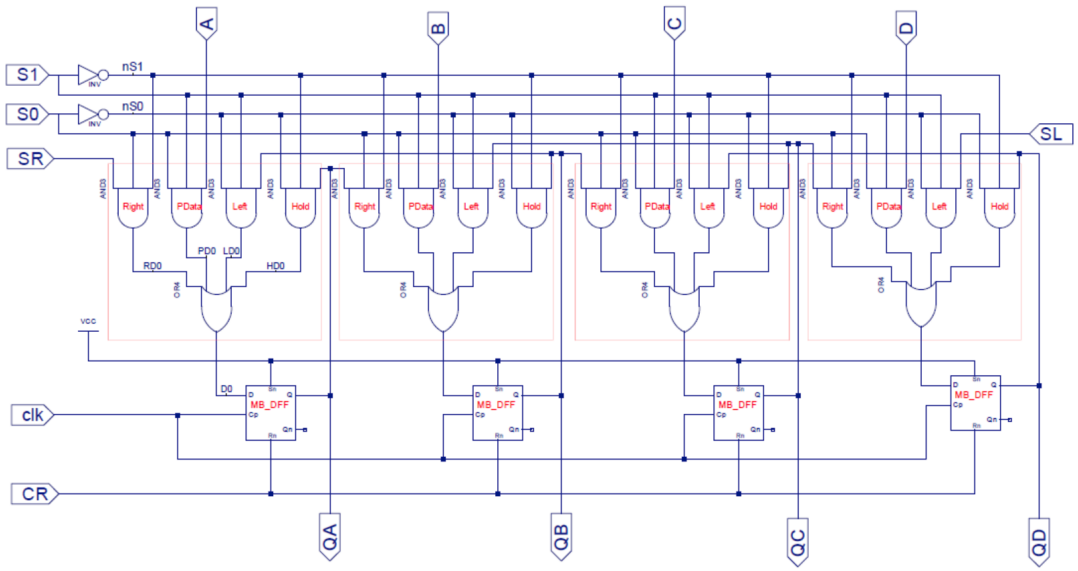
Task1: Exp120-Shift

- Implement 4-bit general shift register DM74LS194
  - call **Maintain Block D Flip-Flop**
  - structural description
- Implement 32-bit general shift register

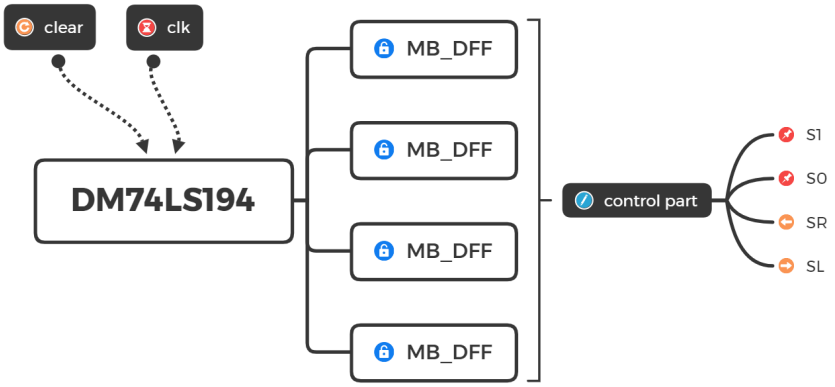
- call DM74LS194
- structural description

- Integrate to the I/O framework

DM74LS194



reference schematic




```
module DM74LS194(
    input clk,
    input CR,
    input S1,
    input S0,
    input A, B, C, D,
    input SL,
    input SR,
    output QA, QB, QC, QD
    );

wire nS0, nS1;
INV GS0(.I(S0), .O(nS0)),
    GS1(.I(S1), .O(nS1));

wire D0, D1, D2, D3;
MB_DFF  Shift0(.Cp(clk), .D(D0), .Rn(CR), .Sn(1'b1), .Q(QA), .Qn()),
        Shift1(.Cp(clk), .D(D1), .Rn(CR), .Sn(1'b1), .Q(QB), .Qn()),
        Shift2(.Cp(clk), .D(D2), .Rn(CR), .Sn(1'b1), .Q(QC), .Qn()),
        Shift3(.Cp(clk), .D(D3), .Rn(CR), .Sn(1'b1), .Q(QD), .Qn());

```

```
wire    HD0, RD0, LD0, PD0,
        HD1, RD1, LD1, PD1,
        HD2, RD2, LD2, PD2,
        HD3, RD3, LD3, PD3;

OR4 GD0(.I0(HD0), .I1(RD0), .I2(LD0), .I3(PD0), .O(D0)),
        GD1(.I0(HD1), .I1(RD1), .I2(LD1), .I3(PD1), .O(D1)),
        GD2(.I0(HD2), .I1(RD2), .I2(LD2), .I3(PD2), .O(D2)),
        GD3(.I0(HD3), .I1(RD3), .I2(LD3), .I3(PD3), .O(D3));

AND3    GH0(.I0(nS1), .I1(nS0), .I2(QA), .O(HD0)),
        GH1(.I0(nS1), .I1(nS0), .I2(QB), .O(HD1)),
        GH2(.I0(nS1), .I1(nS0), .I2(QC), .O(HD2)),
        GH3(.I0(nS1), .I1(nS0), .I2(QD), .O(HD3));

AND3    SR0(.I0(nS1), .I1(S0), .I2(SR), .O(RD0)),
        SR1(.I0(nS1), .I1(S0), .I2(QA), .O(RD1)),
        SR2(.I0(nS1), .I1(S0), .I2(QB), .O(RD2)),
        SR3(.I0(nS1), .I1(S0), .I2(QC), .O(RD3));

AND3    SL0(.I0(S1), .I1(nS0), .I2(QB), .O(LD0)),
        SL1(.I0(S1), .I1(nS0), .I2(QC), .O(LD1)),
        SL2(.I0(S1), .I1(nS0), .I2(QD), .O(LD2)),
        SL3(.I0(S1), .I1(nS0), .I2(SL), .O(LD3));

AND3    P0(.I0(S1), .I1(S0), .I2(A), .O(PD0)),
        P1(.I0(S1), .I1(S0), .I2(B), .O(PD1)),
        P2(.I0(S1), .I1(S0), .I2(C), .O(PD2)),
        P3(.I0(S1), .I1(S0), .I2(D), .O(PD3));
endmodule
```

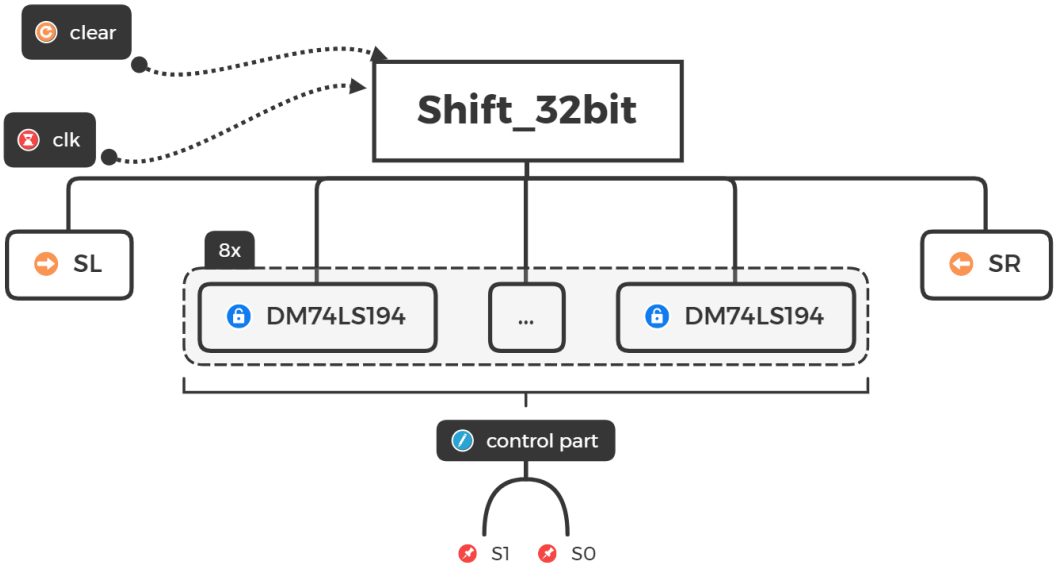
Simulation Code for DM74LS194

```
module DM74LS194_test;
    // Inputs
    reg clk;
    reg CR;
    reg S1;
    reg S0;
    reg A;
    reg B;
    reg C;
    reg D;
    reg SL;
    reg SR;

    // Outputs
    wire QA;
    wire QB;
    wire QC;
    wire QD;

    // Instantiate the Unit Under Test (UUT)
    DM74LS194 uut (
        .clk(clk), .CR(CR),
        .S1(S1), .S0(S0),
        .A(A), .B(B), .C(C), .D(D),
```

```
        .SL(SL), .SR(SR),
        .QA(QA), .QB(QB), .QC(QC), .QD(QD)
    );

    initial begin
        clk = 0;
        CR = 0;
        S1 = 0;
        S0 = 1;
        A = 1;
        B = 0;
        C = 0;
        D = 0;
        SL = 0;
        SR = 1;

        #10;
        CR = 1;
        #190;
        CR = 190;

        #40;
        CR = 1;
        S1 = 1;
        S0 = 0;
        SL = 1;

        #170;
        SR = 0;
        SL = 0;
        S0 = 1;

        #50;
        S1 = 0;
        S0 = 0;

        #50;
        S0 = 1;

        #170;
        S1 = 1;
        A = 0;
        D = 1;

        #40;
        S0 = 0;

    end
    always@* begin
        #20;
        clk <= ~clk;
    end

endmodule
```

Shift_32

This is the illustration of the 32-bit shift register:



My shift register is a bit of difference with professor shi's. That is my shift direction is more corresponded to people's nature.

```
module Shift_32(
    input clk, clear, S1,S0, SL, SR,
    input [31:0] PData,
    output [31:0] Q);

wire CR = ~clear;

DM74LS194   SH0(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(Q[4]), .SR(SR),
                    .A(PData[0]), .B(PData[1]), .C(PData[2]), .D(PData[3]),
                    .QA(Q[0]), .QB(Q[1]), .QC(Q[2]), .QD(Q[3])),

            SH1(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(Q[8]), .SR(Q[3]),
                    .A(PData[4]), .B(PData[5]), .C(PData[6]), .D(PData[7]),
                    .QA(Q[4]), .QB(Q[5]), .QC(Q[6]), .QD(Q[7])),

            SH2(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(Q[12]), .SR(Q[7]),

.A(PData[8]),.B(PData[9]),.C(PData[10]),.D(PData[11]),
                    .QA(Q[8]), .QB(Q[9]), .QC(Q[10]), .QD(Q[11])),

            SH3(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(Q[16]), .SR(Q[11]),

.A(PData[12]),.B(PData[13]),.C(PData[14]),.D(PData[15]),
                    .QA(Q[12]), .QB(Q[13]), .QC(Q[14]), .QD(Q[15])),

            SH4(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(Q[20]), .SR(Q[15]),

.A(PData[16]),.B(PData[17]),.C(PData[18]),.D(PData[19]),
                    .QA(Q[16]), .QB(Q[17]), .QC(Q[18]), .QD(Q[19])),
```

```
27
28              SH5(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(Q[24]),
    .SR(Q[19]),
29
    .A(PData[20]),.B(PData[21]),.C(PData[22]),.D(PData[23]),
30                      .QA(Q[20]), .QB(Q[21]), .QC(Q[22]), .QD(Q[23])),
31
32              SH6(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(Q[28]),
    .SR(Q[23]),
33
    .A(PData[24]),.B(PData[25]),.C(PData[26]),.D(PData[27]),
34                      .QA(Q[24]), .QB(Q[25]), .QC(Q[26]), .QD(Q[27])),
35
36              SH7(.clk(clk), .CR(CR), .S1(S1), .S0(S0), .SL(SL),
    .SR(Q[27]),
37
    .A(PData[28]),.B(PData[29]),.C(PData[30]),.D(PData[31]),
38                      .QA(Q[28]), .QB(Q[29]), .QC(Q[30]), .QD(Q[31]));
39  endmodule
```

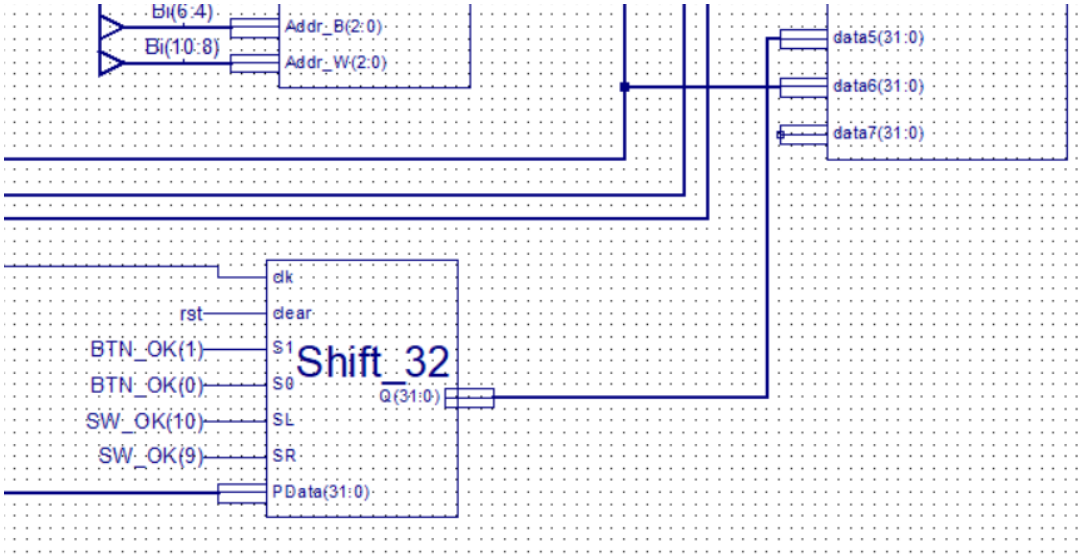
Simulation code for 32-bit shift register

```
1   module Shift_32_test;
2       // Inputs
3       reg clk;
4       reg clear;
5       reg S1;
6       reg S0;
7       reg SL;
8       reg SR;
9       reg [31:0] PData;
10
11      // Outputs
12      wire [31:0] Q;
13
14      // Instantiate the Unit Under Test (UUT)
15      Shift_32 uut (
16          .clk(clk),
17          .clear(clear),
18          .S1(S1),
19          .S0(S0),
20          .SL(SL),
21          .SR(SR),
22          .PData(PData),
23          .Q(Q)
24      );
25
26      initial begin
27          // Initialize Inputs
28          clk = 0;
29          clear = 1;
30          S1 = 0;
31          S0 = 0;
32          SL = 0;
33          SR = 0;
34          PData = 32'h80000000;
35
```

```
        #30;
        clear = 0;
        S0 = 1;
        S1 = 1;

        #50;
        S0 = 0;

        #600;
        SL = 1;

        #1000;
        PData = 32'hAAAAAAAA;
        S1 = 1;
        S0 = 1;

        #100;
        PData = 32'h55555555;
        S1 = 1;
        S0 = 1;

        #100;
        PData = 32'h00000000;
        S1 = 1;
        S0 = 1;

        #100;
        SR = 1;
        S1 = 0;

    end
    always@* begin
        #20;
        clk <= ~clk;
    end

endmodule
```

Physical Test

After having verified all the modules with simulation test, the 32-bit shift register can be integrated onto the I/O framework and do the physical test.



With the user constraint file:

```
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;

NET "SW[0]" LOC=AA10    |   IOSTANDARD=LVCMOS15;
NET "SW[1]" LOC=AB10    |   IOSTANDARD=LVCMOS15;
NET "SW[2]" LOC=AA13    |   IOSTANDARD=LVCMOS15;
NET "SW[3]" LOC=AA12    |   IOSTANDARD=LVCMOS15;
NET "SW[4]" LOC=Y13 |   IOSTANDARD=LVCMOS15;
NET "SW[5]" LOC=Y12 |   IOSTANDARD=LVCMOS15;
NET "SW[6]" LOC=AD11    |   IOSTANDARD=LVCMOS15;
```

```
NET "SW[7]" LOC=AD10    |   IOSTANDARD=LVCMOS15;
NET "SW[8]" LOC=AE10    |   IOSTANDARD=LVCMOS15;
NET "SW[9]" LOC=AE12    |   IOSTANDARD=LVCMOS15;
NET "SW[10]"    LOC=AF12    |   IOSTANDARD=LVCMOS15;
NET "SW[11]"    LOC=AE8 |   IOSTANDARD=LVCMOS15;
NET "SW[12]"    LOC=AF8 |   IOSTANDARD=LVCMOS15;
NET "SW[13]"    LOC=AE13    |   IOSTANDARD=LVCMOS15;
NET "SW[14]"    LOC=AF13    |   IOSTANDARD=LVCMOS15;
NET "SW[15]"    LOC=AF10    |   IOSTANDARD=LVCMOS15;

NET "SEGMENT[0]"    LOC=AB22    |   IOSTANDARD=LVCMOS33;
NET "SEGMENT[1]"    LOC=AD24    |   IOSTANDARD=LVCMOS33;
NET "SEGMENT[2]"    LOC=AD23    |   IOSTANDARD=LVCMOS33;
NET "SEGMENT[3]"    LOC=Y21 |   IOSTANDARD=LVCMOS33;
NET "SEGMENT[4]"    LOC=W20 |   IOSTANDARD=LVCMOS33;
NET "SEGMENT[5]"    LOC=AC24    |   IOSTANDARD=LVCMOS33;
NET "SEGMENT[6]"    LOC=AC23    |   IOSTANDARD=LVCMOS33;
NET "SEGMENT[7]"    LOC=AA22    |   IOSTANDARD=LVCMOS33;

NET "AN[0]" LOC=AD21    |   IOSTANDARD=LVCMOS33;
NET "AN[1]" LOC=AC21    |   IOSTANDARD=LVCMOS33;
NET "AN[2]" LOC=AB21    |   IOSTANDARD=LVCMOS33;
NET "AN[3]" LOC=AC22    |   IOSTANDARD=LVCMOS33;

NET"LED[0]"LOC=W23  | IOSTANDARD=LVCMOS33 ;#D1
NET"LED[1]"LOC=AB26 | IOSTANDARD=LVCMOS33 ;#D2
NET"LED[2]"LOC=Y25  | IOSTANDARD=LVCMOS33 ;#D3
NET"LED[3]"LOC=AA23 | IOSTANDARD=LVCMOS33 ;#D4
NET"LED[4]"LOC=Y23  | IOSTANDARD=LVCMOS33 ;#D5
NET"LED[5]"LOC=Y22  | IOSTANDARD=LVCMOS33 ;#D6
NET"LED[6]"LOC=AE21 | IOSTANDARD=LVCMOS33 ;#D7
NET"LED[7]"LOC=AF24  | IOSTANDARD=LVCMOS33 ;#D8
```

Task2: parallel to serial module

- Design n-bit bi-directional shift register
  - by structural description
  - variable bit width: **n**
  - sequential simulation test
- Design parallel-to-serial module: P2S
  - call the above shift register
- Replace the `ngc` file

64-bit shift register

```
module SHIFT64(
    input clk, SR, SL, S1, S0,
    input [DATA_BITS:0]D,
    output reg[DATA_BITS:0]Q
    );

parameter
    DATA_BITS = 64,         //data length
    DATA_COUNT_BITS = 4;    //data shift bits
```

```
10
11  always@(posedge clk) begin
12      case({S1, S0})
13          2'b00: Q <= Q;                      //hold
14          2'b01: Q <= {SR, Q[DATA_BITS:1]};   //shift right
15          2'b10: Q <= {Q[DATA_BITS-1:0], SL}; //shift left
16          2'b11: Q <= D;                      //parallel input
17      endcase
18  end
19
20  endmodule
```

P2S

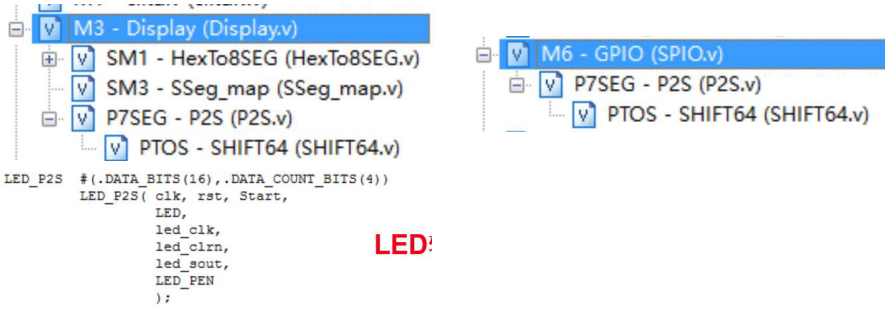
```
1   `timescale 1ns / 1ps
2   module P2S(                             //parallel to serial
3       input clk, rst, Start,
4       input [DATA_BITS-1:0] PData,
5       output s_clk, s_clrn, sout,
6       output reg EN
7       );
8
9   parameter   DATA_BITS = 64,             //data length
10              DATA_COUNT_BITS = 4,        //data shift bits
11              DIR = 1;                    //shift direction
12
13  wire S1, S0, SL, SR, finish, shift;
14  wire [DATA_BITS:0] D, Q;
15  reg [1:0] Go = 00, S = 00;
16
17  assign {SR, SL} = 2'b11;
18  assign {S1, S0} = DIR ? {S[0], S[1]} : S;                   //adjust
    to shift direction
19  assign D = DIR ?   {1'b0, PData}:{PData, 1'b0};
20  assign finish = DIR ? &Q[DATA_BITS:1] : &Q[DATA_BITS-1:0];  //finish
    flag
21  assign sout = DIR ? Q[0]:Q[DATA_BITS];                     //serial
    output
22
23  SHIFT64 #(.DATA_BITS(DATA_BITS))
24  PTOS(                                   //call shift register
25      .clk(clk),
26      .SR(SR), .SL(SL),
27      .S1(S1), .S0(S0),
28      .D(D), .Q(Q)
29      );
30
31  always@(posedge clk)
32      Go <= {Go[0], Start};
33
34  assign shift = (Go==2'b01) ? 1:0;
35
36  always @ (posedge clk or posedge rst) begin
37      if(rst) begin EN = 1; S = 2'b11; end        //parallel input
38      else begin
39          if(shift)   begin EN = 0; S=2'b11; end
40          else begin
```

```
            if(!finish) begin EN = 0; S = 2'b10; end
            else        begin EN = 1; S = 2'b00; end
        end
    end
end

assign s_clk = finish | clk;                    //disable clock
assign s_clrn = 1;
endmodule

```

The parameter (bit width) is flexible, and thus can be modified where it is called.

```
SHIFT64 #(.DATA_BITS(DATA_BITS))
PTOS(                                   //call shift register
    .clk(clk),
    .SR(SR), .SL(SL),
    .S1(S1), .S0(S0),
    .D(D), .Q(Q)
    );
```

After having finished the P2S module, replace the `ngc` file in the I/O framework.



It is worth noting that the shift direction is actually to **right** in the P2S module. Thus we have to change `DIR` to 1.⚠

5. Experiment Results and Analyses

Problems

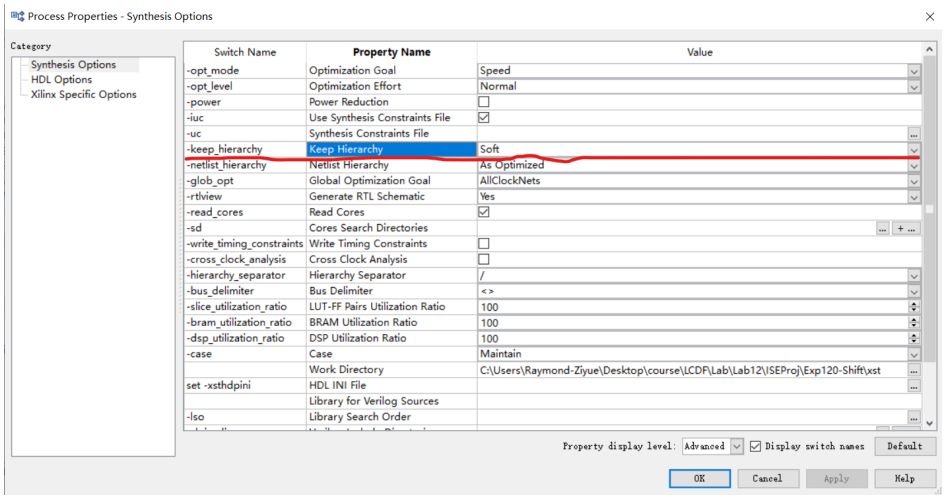
Synthesis error

At the first time I met with such complication during the **synthesis-XST** phase:

```
INTERNAL_ERROR:Xst:cmain.c:3423:1.29 - Process will terminate. For technical support on this issue, please open a WebCase with this project attached at http://www.xilinx.com/support.
Process "Synthesize - XST" failed
```

error log

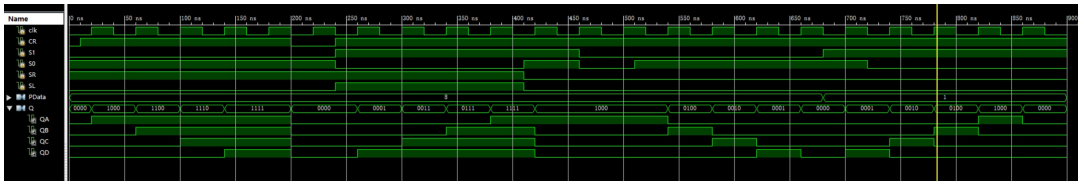
It's a bit of annoying that there's nothing wrong with my design. After browsing the xilinx official website, I found the solution: setting `keep_hierarchy to soft`.



Then the synthesis can run perfectly.

Task1: Exp120-Shift

Simulation test for DM74LS194

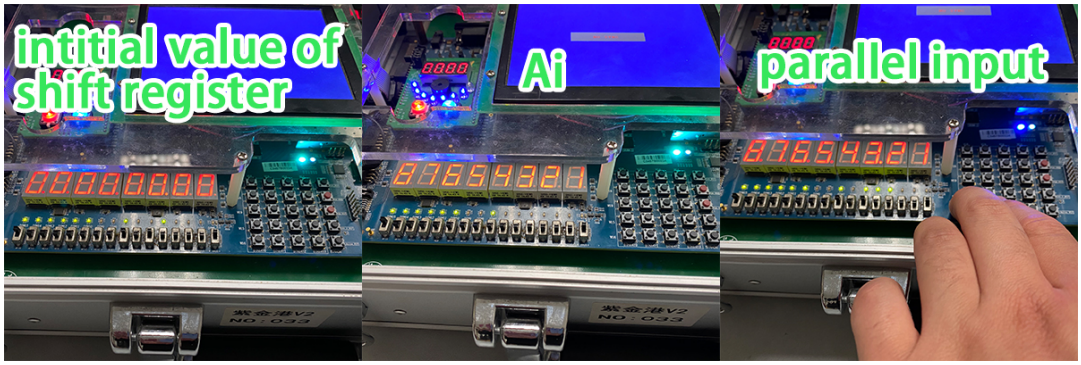


According to the simulation waveform, the shift function has all been verified.

Physical Test

Step 1:

First we verify the parallel input function. As what can be seen from the picture, after both S1 and S0 has been activated, value in Ai is input into the shift register.

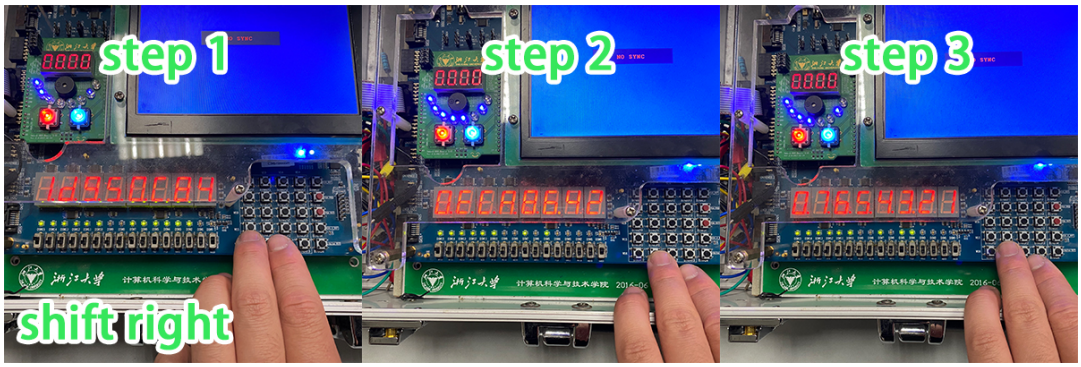


Step 2:

Shift left without shift input. (Only S0 is activated)

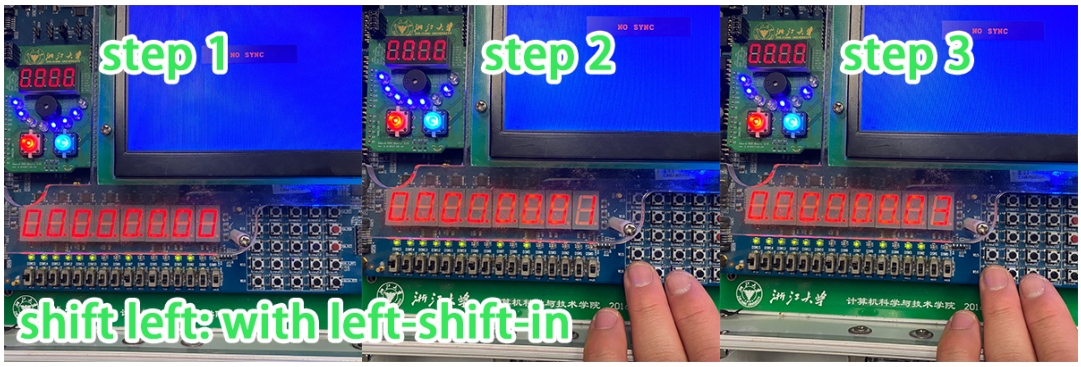
Step 3:

Shift right without shift input. (Only S1 is activated)



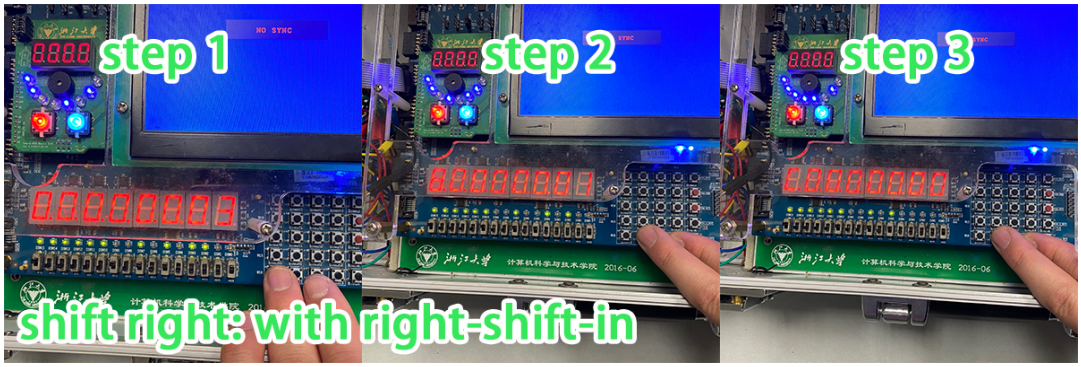
Step 4:

Shift left with shift input. (Only S0 and SW_OK[9] is activated)



Step 5:

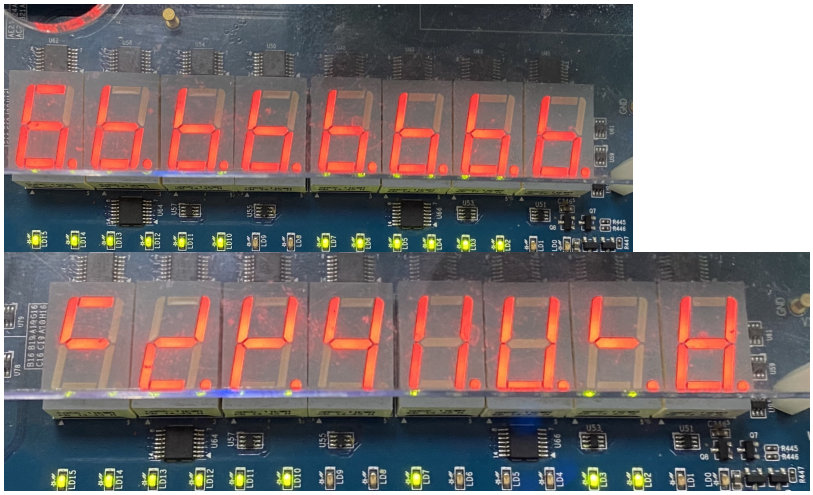
Shift right with shift input. (Only S1 and SW_OK[10] is activated)



The value in each step can be verified. Thus the implementation of the shift register has been perfectly verified.

Task2: PS2

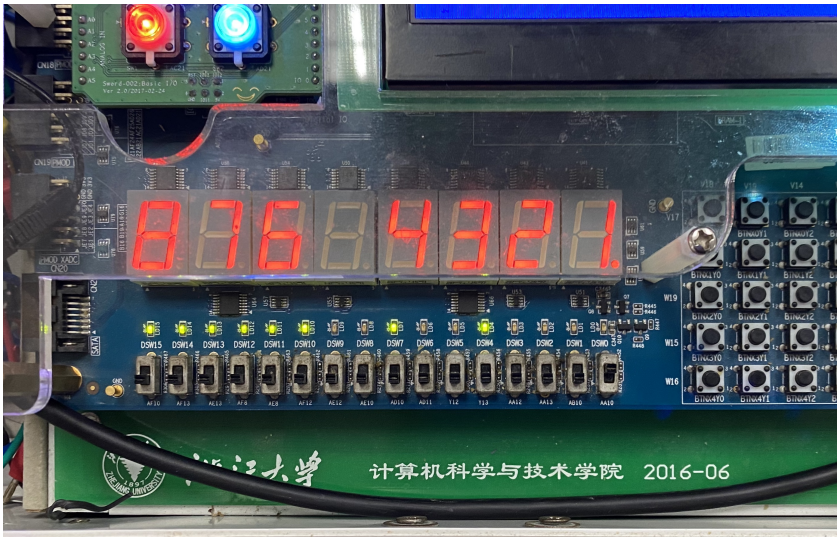
PS2 module has caused much complication to me. At first, my physical test result is like this:



After looking up the structure of 7-segment code, I infer that the shift direction is just reversed. Then I tried many times and finally succeeded by setting `Dir` variable to 1 (shift right).

```
parameter    DATA_BITS = 64,          //data length
             DATA_COUNT_BITS = 4,     //data shift bits
             DIR = 1;                 //shift direction
```

Then the problem is solved:



6. Discussion and Conclusion

Right to say this experiment is much meaningful and simple. It's not about making hundreds of modules or designing complicated logical structure. We just master the basic design method and working principles of shift register and parallel-to-serial module.

However, no matter how easy the principles are, I still met with nuisances and complications. Especially the P2S module.

When I finally solved the problem, I totally master the principle of transmit a parallel input to serial output, which is definitely useful in our future study. And thus I know it further that how important practice is in logic and computer design. 💻