浙江大学

本科实验报告

课程名称：	数字逻辑电路设计
姓　　名：	金祺书
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
指导教师：	洪奇军
报告日期：	2024 年 4 月 25 日

浙江大学实验报告

课程名称：数字逻辑设计 实验类型：综合

实验项目名称：加法器、加减法器和 ALU 基本原理与设计

学生姓名：金祺书 学号：3230104248 同组学生姓名：蒋翼泽

实验地点：紫金港东四 509 室 实验日期：2024 年 4 月 25 日

一、操作方法与实验步骤

1、一位全加器

根据一位全加器功能，很容易得到其真值表：

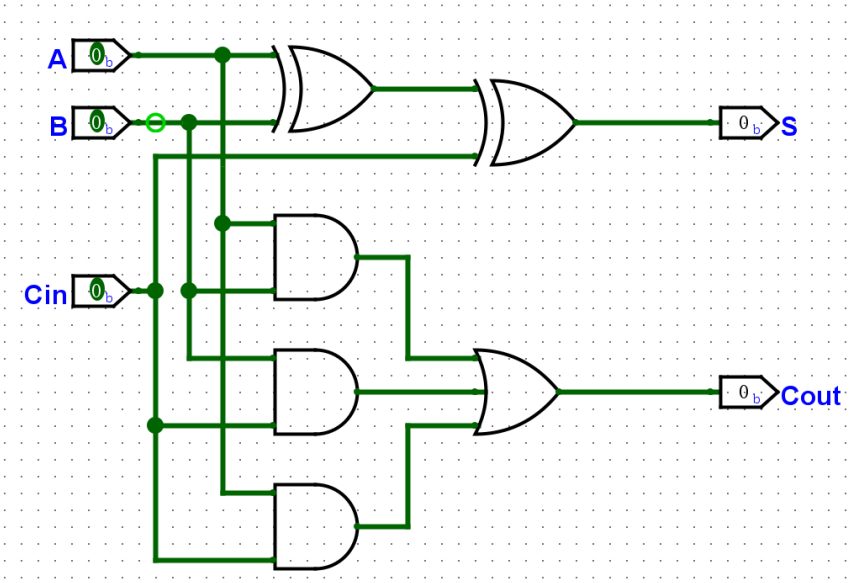
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

根据真值表，可以获得输出 S,C_{out} 关于输入 A,B,C_{in} 的函数：

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + BC_{in} + AC_{in}$$

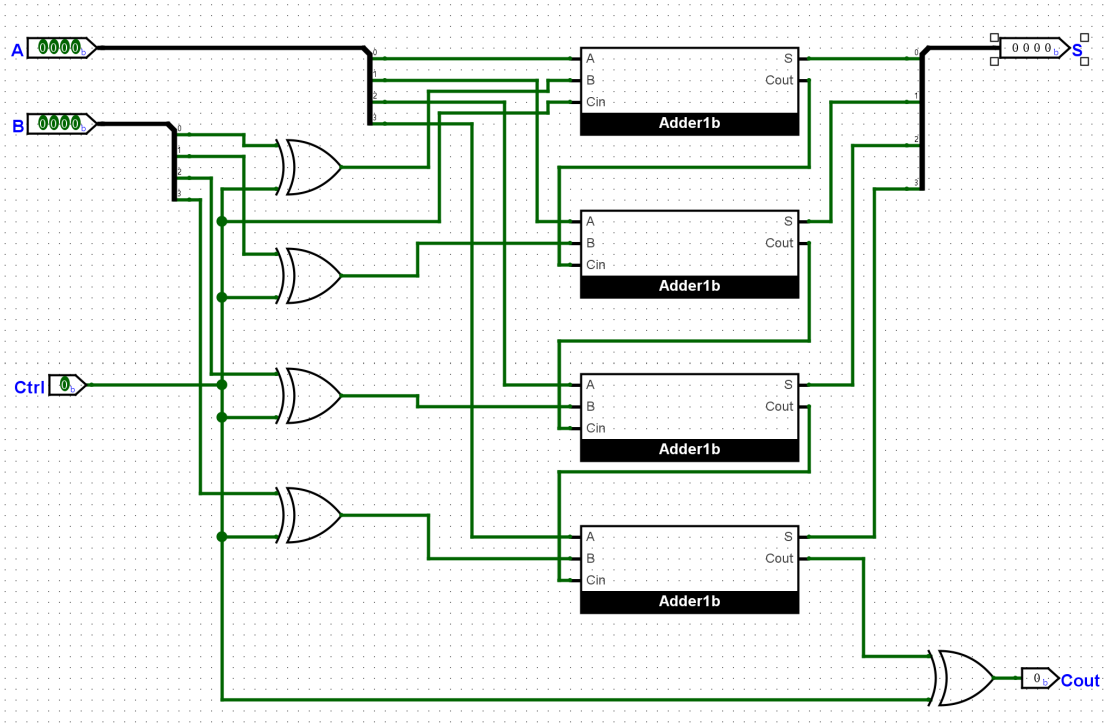
使用 Logisim 绘制 Adder1b 的原理图，并将其转换成 Verilog Code



2、四位加减法器

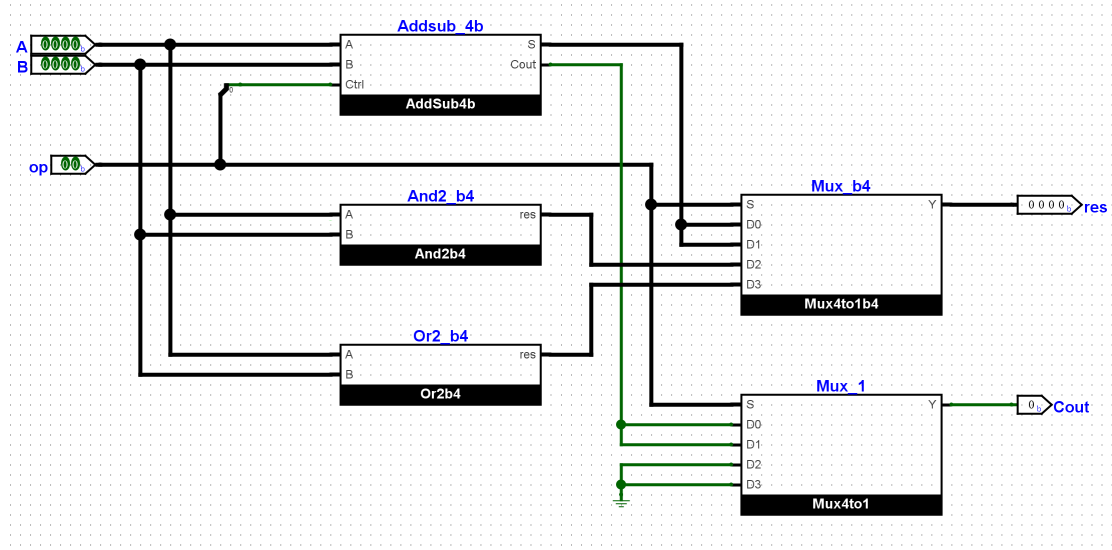
在得到全加器实现后，可以为其添加一些逻辑实现加减法器。减法 A-B 可以看作 A+(-B)，

其中-B 使用 B 的补码表示，即可使用全加器得到减法的结果。
减数 B 的补码可以写作~B+1。观察加法的 B 与减法的-B，差别在于加法器第二个操作数是否取反以及是否有最低的进位，联想到异或操作的特性（一个操作数为 1 时，结果为另一个操作数的反；一个操作数为 0 时，结果与另一个操作数相同），可以得到以下原理图：



3、ALU 实现及应用

最后我们将实现一个简单的 4 位 ALU，进行加减法、与、或操作。
（1）使用 Logisim 绘制 ALU 的原理图，并将其转换成 Verilog Code



（2）Vivado 新建工程，对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

```
01  `timescale 1ns / 1ps
02
```

```
03 module ALU_tb();
04 //Input
05     reg[3:0] A;
06     reg[3:0] B;
07     reg[1:0] op;
08 //Output
09     wire[3:0] res;
10     wire[1:0] Cout;
11 //Instantiate the UUT
12     ALU ALU_inst(
13         .A(A),
14         .B(B),
15         .op(op),
16         .res(res),
17         .Cout(Cout)
18     );
19     integer i;
20     initial begin
21         op=00;
22 
23         A=4'b1111;
24         B=4'b0001;
25         for(i=0;i<=3;i=i+1) begin
26             {op}=i;#50;
27         end
28 
29         A=4'b1000;
30         B=4'b0001;
31         for(i=0;i<=3;i=i+1) begin
32             {op}=i;#50;
33         end
34 
35         A=4'b1000;
36         B=4'b0011;
37         for(i=0;i<=3;i=i+1) begin
38             {op}=i;#50;
39         end
40 
41         A=4'b1000;
42         B=4'b0100;
43         for(i=0;i<=3;i=i+1) begin
44             {op}=i;#50;
45         end
46 
```

```
47          A=4'b1001;
48          B=4'b0101;
49          for(i=0;i<=3;i=i+1) begin
50              {op}=i;#50;
51          end
52      end
53  endmodule
```

（3）添加去抖动模块

因为按键过程中会有机械原因导致的信号抖动，在 lab7 中也可以观察到，一次按下按钮数字会跳动多次。我们可以实现一个去抖动模块，当信号稳定一段时间后才更改信号值。代码如下：

```
01  module pbdebounce(
02      input wire clk,
03      input wire button,
04      output reg pbreg
05      );
06
07      reg [7:0] pbshift;
08
09      always@(posedge clk) begin
10          pbshift = pbshift<<1;
11          pbshift[0] = button;
12          if (pbshift==8'b0)
13              pbreg=0;
14          if (pbshift==8'hFF)
15              pbreg=1;
16      end
17
18  endmodule
```

（4）七段数码管静态显示

我们希望使用 SWORD 板上的 8 个七段数码管，它接收串行数据以减少管脚使用。我们需要一个模块将 64 位的七段数码管数据转换为 1 位串行输出以及相应的同步信号，即并转串模块 P2S，将 P2S_io.v 和 P2S.edf 添加进工程即可，需要注意将工程中 P2S.edf 的文件格式设置为 EDIF（右键文件选择 Set file type）。

（5）CreateNumber 模块

在本模块中使用 AddSub4b 模块，实现按键自增/自减。代码如下：

```
01  `timescale 1ns / 1ps
02
```

```
03 module Top(
04         input wire clk,
05         input wire [1:0] BTN,
06         input wire [1:0] SW1,
07         input wire [1:0] SW2,
08         input wire [11:0] SW,
09         output wire [3:0] AN,
10         output wire [7:0] SEGMENT,
11         output wire BTNX4,
12         output wire seg_clk,
13         output wire seg_clrn,
14         output wire seg_sout,
15         output wire SEG_PEN
16 );
17     wire [15:0] num;
18     wire [1:0] btn_out;
19     wire [3:0] res;
20     wire Co;
21     wire [31:0] clk_div;
22     wire [15:0] disp_hexs;
23     wire [15:0] disp_hexs_my;
24
25     assign disp_hexs[15:12] = num[7:4];          // B
26     assign disp_hexs[11:8]  = num[3:0];          // A
27     assign disp_hexs[7:4]   = {3'b000, Co};
28     assign disp_hexs[3:0]   = res[3:0];             // C
29
30     /* Code here */
31     assign disp_hexs_my = 16'b0100_0010_0100_1000; // Fill the
last four digits of your student id in ()
32
33     assign BTNX4 = 1'b0;
34
35     clkdiv m2(.clk(clk), .rst(0), .div_res(clk_div));
36     pbdebounce
m0(.clk(clk_div[17]), .button(BTN[0]), .pbreg(btn_out[0]));
37     pbdebounce
m1(.clk(clk_div[17]), .button(BTN[1]), .pbreg(btn_out[1]));
38
39     CreateNumber m3(.btn({2'b0, btn_out}), .sw({2'b0,
SW1}), .num(num)); // Attachment
40
41     // The ALU module you wrote
42     ALU m5(    .A(num[3:0]),
```

```
43                  .B(num[7:4]),                          // fill
sth. in ()
44                  .op(SW2),                          // fill sth. in
()
45                  .res(res),                          // fill sth.
in ()
46                  .Cout(Co));
47
48      // Module you design in Lab7
49      DisplayNumber m6(    .clk(clk), .hexs(disp_hexs), .LEs(4'b0),
// fill sth. in ()
50                              .points(4'b0), .rst
51                              (1'b0),                          //
fill sth. in ()
52                              .AN(AN), .SEGMENT(SEGMENT));
53
54      // Attachment
55      SSeg_Dev
m7(.clk(clk), .flash(clk_div[25]), .Hexs({disp_hexs_my,
disp_hexs}), .LES(SW[11:4]),
56                  .point({4'b0000,
SW[3:0]}), .rst(1'b0), .Start(clk_div[20]), .seg_clk(seg_clk),
57
    .seg_clrn(seg_clrn), .SEG_PEN(SEG_PEN), .seg_sout(seg_sout));
58
59  endmodule
```

（6）通过约束文件：

```
01  # Filename: constraints_lab8.xdc
02  ## Constraints file for Lab8
03
04  # Main clock
05  set_property PACKAGE_PIN AC18 [get_ports clk]
06  set_property IOSTANDARD LVCMOS18 [get_ports clk]
07
08  create_clock -period 10.000 -name clk [get_ports "clk"]
09
10  # Switches as inputs
11  set_property PACKAGE_PIN AA10 [get_ports {SW[0]}]
12  set_property PACKAGE_PIN AB10 [get_ports {SW[1]}]
13  set_property PACKAGE_PIN AA13 [get_ports {SW[2]}]
14  set_property PACKAGE_PIN AA12 [get_ports {SW[3]}]
15  set_property PACKAGE_PIN Y13 [get_ports {SW[4]}]
16  set_property PACKAGE_PIN Y12 [get_ports {SW[5]}]
17  set_property PACKAGE_PIN AD11 [get_ports {SW[6]}]
```

```
set_property PACKAGE_PIN AD10 [get_ports {SW[7]}]
set_property PACKAGE_PIN AE10 [get_ports {SW[8]}]
set_property PACKAGE_PIN AE12 [get_ports {SW[9]}]
set_property PACKAGE_PIN AF12 [get_ports {SW[10]}]
set_property PACKAGE_PIN AE8 [get_ports {SW[11]}]
set_property PACKAGE_PIN AF8 [get_ports {SW1[0]}]
set_property PACKAGE_PIN AE13 [get_ports {SW1[1]}]
set_property PACKAGE_PIN AF13 [get_ports {SW2[0]}]
set_property PACKAGE_PIN AF10 [get_ports {SW2[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[3]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[4]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[5]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[6]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[7]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[8]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[9]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[10]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[11]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW1[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW1[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW2[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW2[1]}]

# Key as inputs
set_property PACKAGE_PIN W16 [get_ports BTNX4]
set_property IOSTANDARD LVCMOS18 [get_ports BTNX4]
set_property PACKAGE_PIN V14 [get_ports {BTN[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN[1]}]
set_property PACKAGE_PIN W14 [get_ports {BTN[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN[0]}]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets BTN*]

# Arduino-Segment & AN
set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]
set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
```

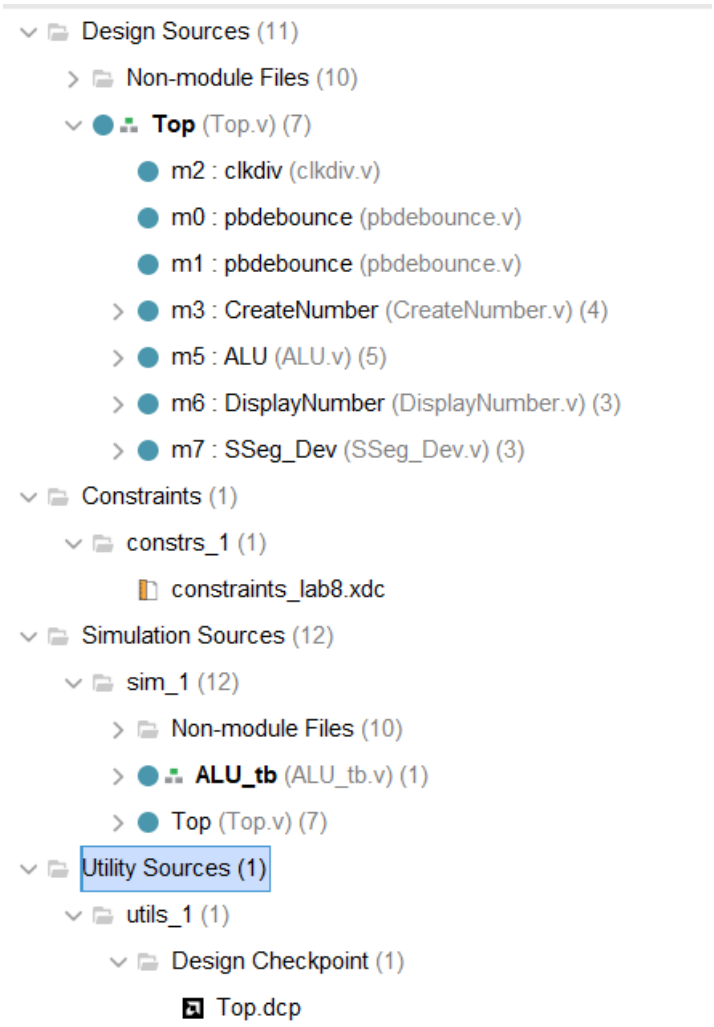
```
set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]
set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]
set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]

set_property PACKAGE_PIN M24 [get_ports {seg_clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_clk}]
set_property PACKAGE_PIN M20 [get_ports {seg_clrn}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_clrn}]
set_property PACKAGE_PIN L24 [get_ports {seg_sout}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_sout}]
set_property PACKAGE_PIN R18 [get_ports {SEG_PEN}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG_PEN}]
```

生成比特流下板验证。

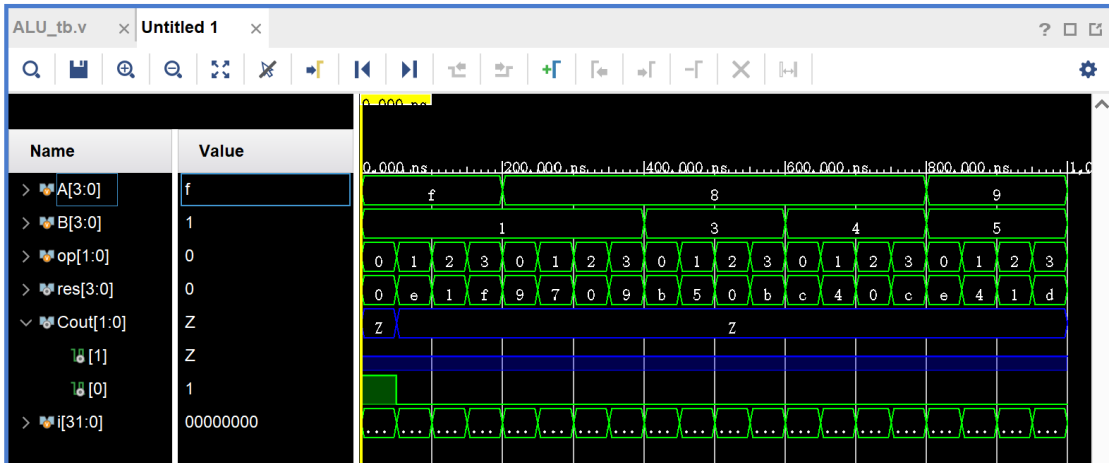
二、实验结果与分析

1、ALU 实现及应用

Vivado 工程中的文件结构如下：

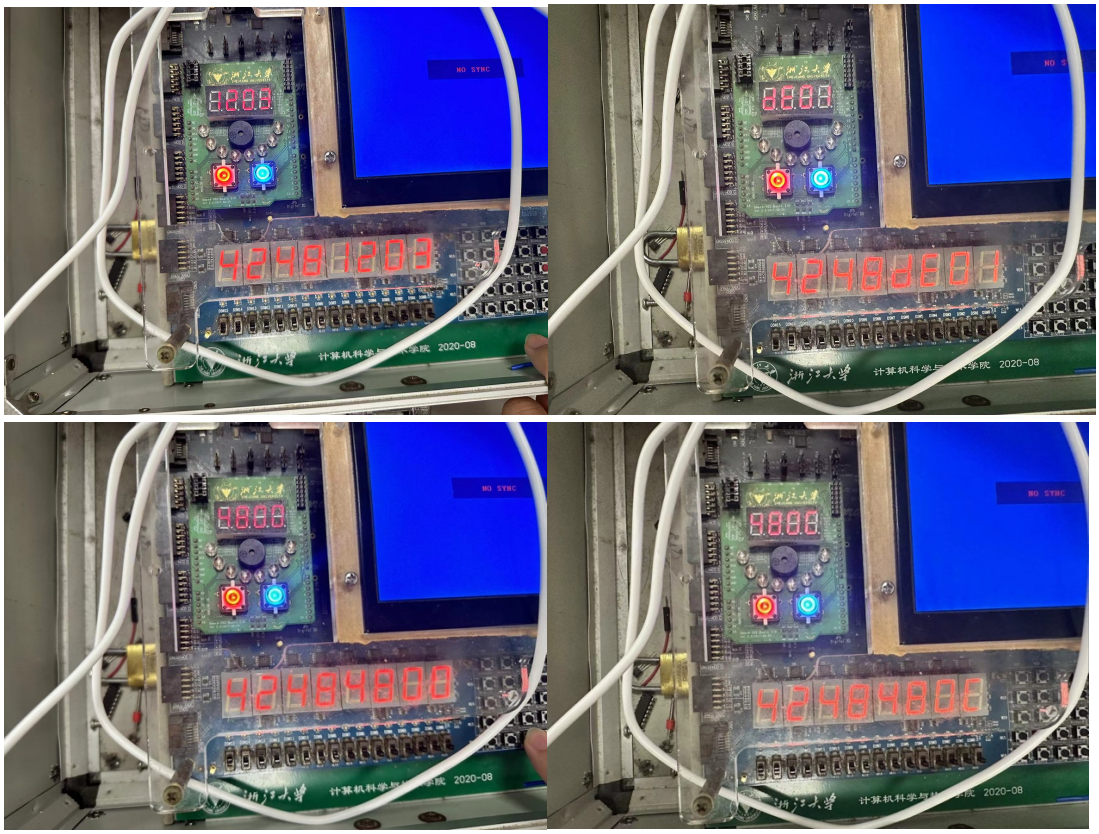


导入 Vivado 后的仿真界面截图如下：



在该仿真过程中，A，B 分别代表两个参与运算的数字，op 从 0 到 3 分别代表+，-，&，|的运算符，res 则是运算结果，以 A=f，B=1 为例，A+B=0，A-B=e，A&B=1，A|B=f，仿真结果符合预期。

下板结果如下：



从左到右两个开关控制运算方式，00 代表+，01 代表-，10 代表按位与，11 代表按位或。
两个按钮分别控制前后两个数字的自增，从 0 到 F，到 F 后再增变为 0.
前四位数字为学号的后四位数字，后四位数字分别代表两个参与运算的数字，一个 0 代表等于号，最后一位数字代表运算结果，符合预期。

三、讨论、心得

在这次的实验过程中，原理图与仿真过程还算顺利，但是在上板时因上下两块数码管示数不一致而困扰了很久，最后在向老师寻求帮助后发现是之前 lab6 和 lab7 原理图绘制过程中接线接错没有保存正确接线导致 lab8 反复调用出错的。说明以后应当更仔细避免小错误，更应该保存每次实验的正确结果代码与原理图。

浙江大学

本科实验报告

课程名称：	数字逻辑电路设计
姓　　名：	金祺书
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
指导教师：	洪奇军
报告日期：	2024 年 5 月 11 日

浙江大学实验报告

课程名称：数字逻辑设计 实验类型：综合

实验项目名称：同步时序电路设计

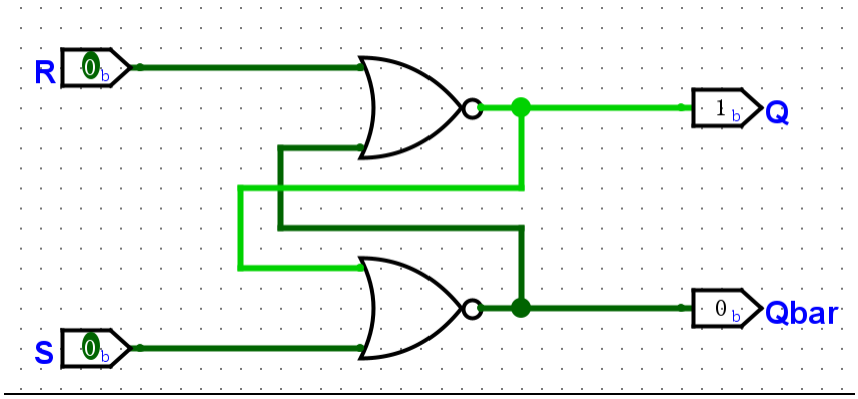
学生姓名：金祺书 学号：3230104248 同组学生姓名：蒋翼泽

实验地点：紫金港东四 509 室 实验日期：2024 年 5 月 9 日

一、操作方法与实验步骤

1、SR 锁存器

（1）使用 Logisim 绘制 SR_latch 的原理图，并将其转换成 Verilog Code



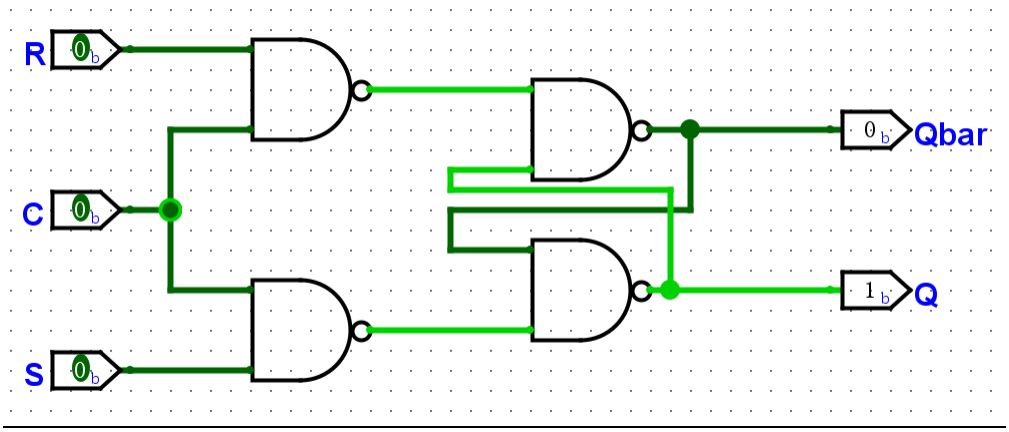
（2）Vivado 新建工程，对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

```
01  `timescale 1ns / 1ps
02
03  module SR_latch_tb();
04  //Inputs
05      reg S;
06      reg R;
07  //Outputs
08      wire Q;
09      wire Qbar;
10  //Instantiate the UUT
11      SR_latch SR_latch_inst(
12          .S(S),
13          .R(R),
14          .Q(Q),
15          .Qbar(Qbar)
```

```
    );
//Initialize Inputs
    initial begin
        R=1;S=0; #50;
        R=0;S=0; #50;
        R=0;S=1; #50;
        R=0;S=0; #50;
    end
endmodule
```

2、门控 SR 锁存器

（1）使用 Logisim 绘制 C_SR_latch 的原理图，并将其转换成 Verilog Code



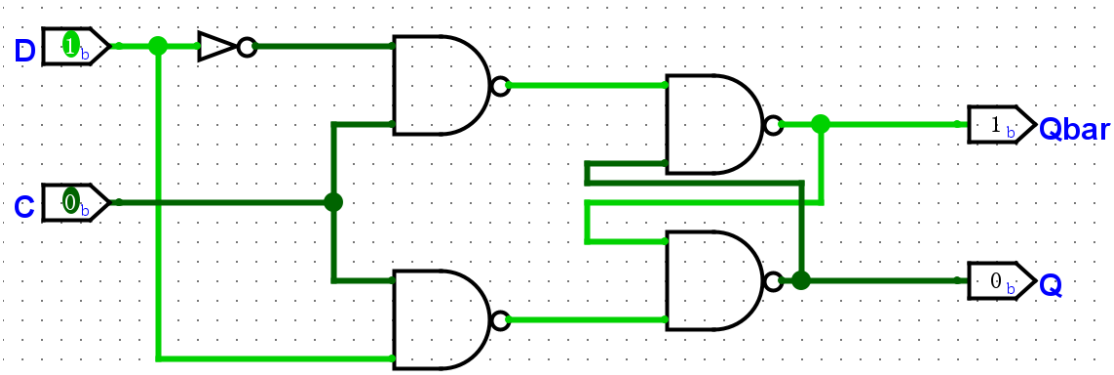
（2）在 Vivado 工程中对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

```
`timescale 1ns / 1ps

module C_SR_latch_tb();
//Inputs
    reg C;
    reg S;
    reg R;
//Outputs
    wire Q;
    wire Qbar;
//Instantiate the UUT
    C_SR_latch C_SR_latch_inst(
        .C(C),
        .S(S),
        .R(R),
        .Q(Q),
        .Qbar(Qbar)
    );
```

```
19 //Initialize Input
20     initial begin
21         C=1;R=1;S=0; #50;
22         R=0;S=0; #50;
23         R=0;S=1; #50;
24         R=0;S=0; #50;
25         C=0;R=1;S=0; #50;
26         R=0;S=0; #50;
27         R=0;S=1; #50;
28         R=0;S=0; #50;
29     end
30 endmodule
```

3、门控 D 锁存器

（1）使用 Logisim 绘制 C_D_latch 的原理图，并将其转换成 Verilog Code



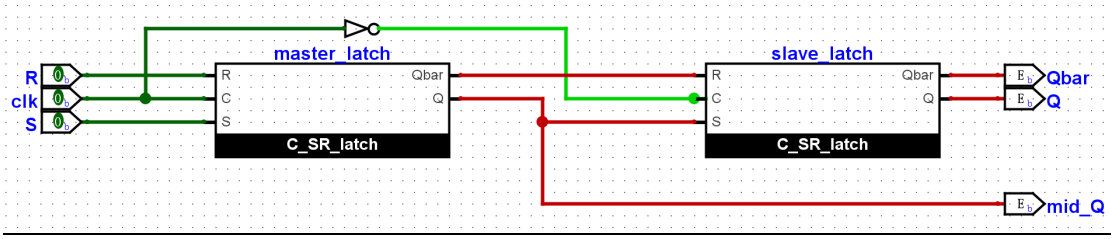
（2）在 Vivado 工程中对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

```
01 `timescale 1ns / 1ps
02
03 module C_D_latch_tb();
04 //Inputs
05     reg C;
06     reg D;
07 //Outputs
08     wire Q;
09     wire Qbar;
10 //Instantiate the UUT
11     C_D_latch C_D_latch_inst(
12         .C(C),
13         .D(D),
14         .Q(Q),
15         .Qbar(Qbar)
16     );
17 //Initialize Input
```

```
18      initial begin
19          C=1;D=1; #50;
20          D=0; #50;
21          D=1; #50;
22          D=0; #50;
23          C=0;D=1; #50;
24          D=0;
25      end
26  endmodule
```

4、正边沿 SR 主从触发器

（1）使用 Logisim 绘制 MS_SR_flip_flop 的原理图，并将其转换成 Verilog Code



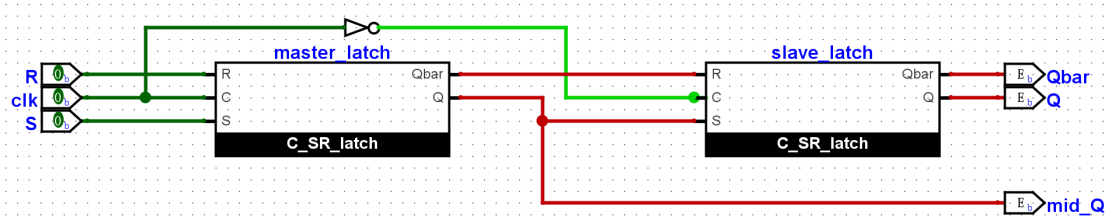
（2）在 Vivado 工程中对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

```
01  `timescale 1ns / 1ps
02
03  module MS_SR_flip_flop_tb();
04  //Inputs
05      reg S;
06      reg R;
07      reg clk;
08  //Outputs
09      wire Q;
10      wire Qbar;
11      wire mid_Q;
12  //Instantiate the UUT
13      MS_SR_flip_flop MS_SR_flip_flop_inst(
14          .S(S),
15          .R(R),
16          .clk(clk),
17          .Q(Q),
18          .Qbar(Qbar),
19          .mid_Q(mid_Q)
20      );
21  //Initialize Input
22      always begin
23          clk=1; #50;
24          clk=0; #50;
```

```
25      end
26      initial begin
27          R=0;S=0; #100;
28          R=0;S=1; #100;
29          R=0;S=0; #100;
30          R=1;S=0; #100;
31          R=0;S=0; #190;
32          R=0;S=1; #20;
33          R=0;S=0; #5;
34          R=1;S=0; #20;
35          R=0;S=0; #55;
36          R=0;S=1; #20;
37          R=0;S=0; #90;
38      end
39  endmodule
```

5、正边沿 D 触发器

（1）使用 Logisim 绘制 ET_D_flip_flop 的原理图，并将其转换成 Verilog Code



（2）在 Vivado 工程中对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

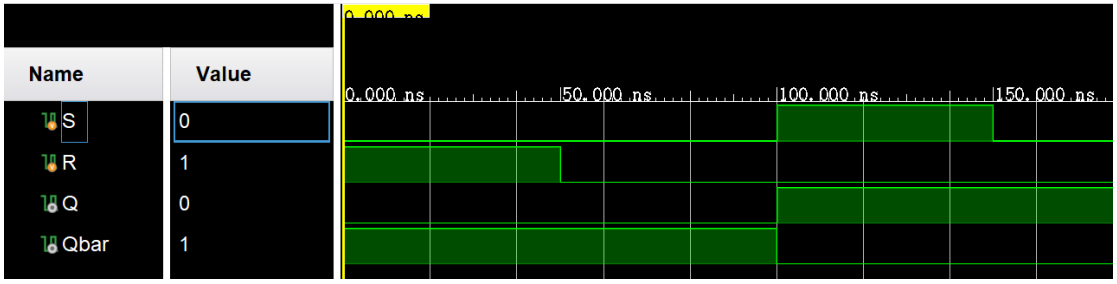
```
01  `timescale 1ns / 1ps
02
03  module ET_D_flip_flop_tb();
04  //Inputs
05      reg D;
06      reg clk;
07  //Outputs
08      wire Q;
09      wire Qbar;
10  //Instantiate the UUT
11      ET_D_flip_flop ET_D_flip_flop_inst(
12          .D(D),
13          .clk(clk),
14          .Q(Q),
15          .Qbar(Qbar)
16      );
17  //Initialize Input
18      always begin
```

```
19          clk=1; #50;
20          clk=0; #50;
21      end
22      initial begin
23          D=0; #150;
24          D=1; #150;
25          D=0; #150;
26          D=1; #150;
27      end
28  endmodule
```

二、实验结果与分析

1、SR 锁存器

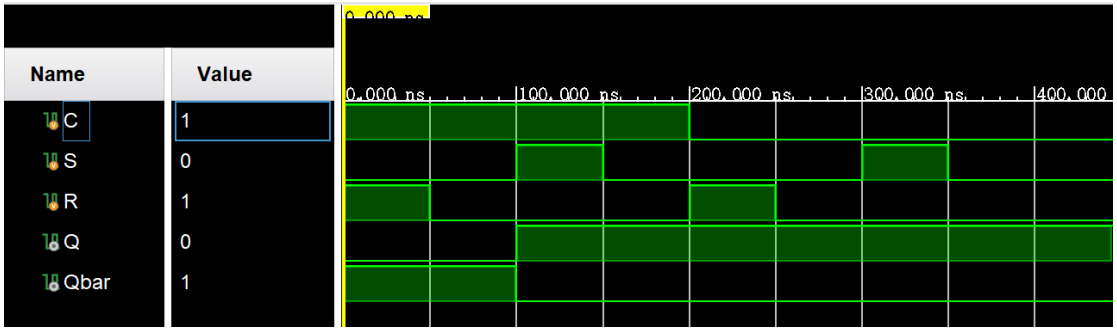
导入 Vivado 后的仿真界面截图如下：



当 SR 为 01 时，Q 置 0；SR 为 00 时，保持上一状态；SR 为 10 时，Q 置 1，符合预期要求。

2、门控 SR 锁存器

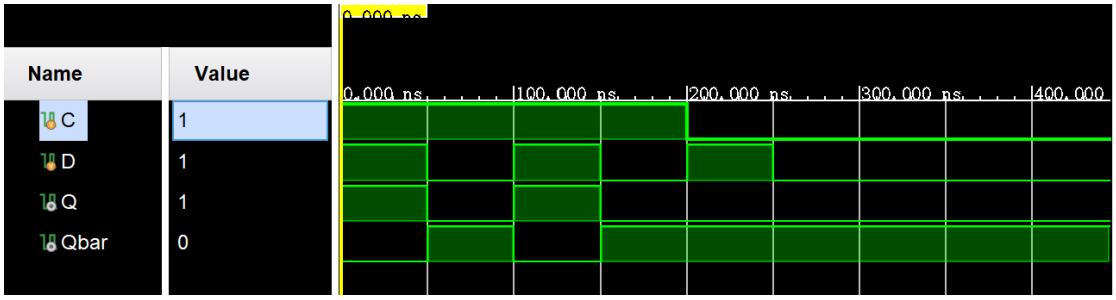
导入 Vivado 后的仿真界面截图如下：



当 C 为 0 时，保持原有信号，RS 信号无效；当 C 为 1 时，RS 信号有效，RS=10，Q 置 0，RS=01，Q 置 1，RS=00，信号保持上一状态，符合预期要求。

3、门控 D 锁存器

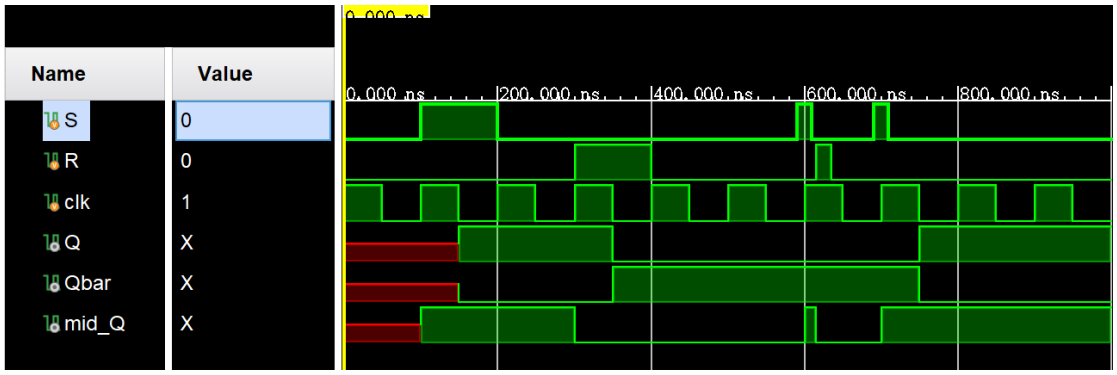
导入 Vivado 后的仿真界面截图如下：



C 为 0 时，D 无效，Q 信号保持，C 为 1 时，D=1，Q 置 1，D=0，Q 置 0，符合预期要求。

4、正边沿 SR 主从触发器

导入 Vivado 后的仿真界面截图如下：



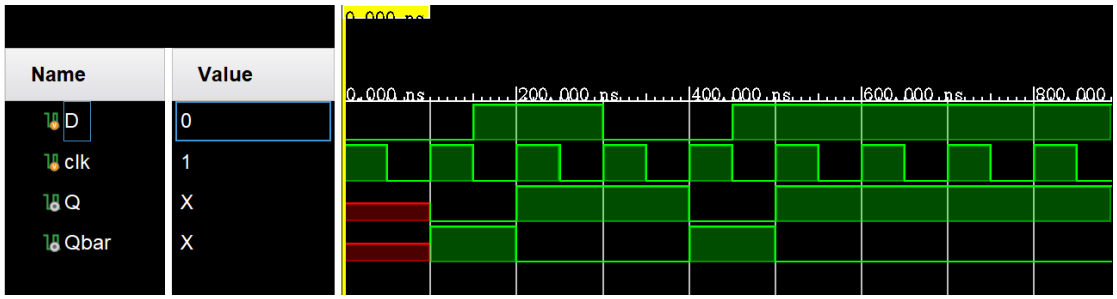
clk 处于低位时，master 门控有效，slave 门控无效，对 D 进行修改，master 的输出会改变，但 slave 的输出（也就是触发器的输出）不会改变，clk 上升沿后一小段时间内，master 门控变为无效，slave 门控有效，master 输出不会再改变，slave 接收到 master 的输出并改变或维持自身输出。

clk 处于高位时，master 门控无效，slave 门控有效，对 D 进行修改，master 的输出不会改变，因此 slave 的输出也不会改变

但是该主从触发器存在一次性采样问题，发现 S 出现噪音信号时可能触发器的输出，如途中的 S，R 的小突起，使得输出发生了保持的变化。

5、正边沿 D 触发器

导入 Vivado 后的仿真界面截图如下：



clk 处于低位时，不论 D 为何，输入处理阶段的两个锁存器均输出 1，输出处理阶段的锁存器保持之前的状态。clk 上升沿，输入处理阶段两个锁存器有且只有一个会输出 0，若 D 为 0，则图中靠下锁存器输出 0，若 D 为 1，则图中靠上锁存器输出 0。

clk 处于高位时，输入处理阶段的两个锁存器维持上升沿改变后的输出（即有且只有一个输出 0），且 D 的改变并不会影响锁存器的状态。

输出处理阶段的锁存器会根据输入处理的锁存器输出进行 set 或 reset，即改变或维持触发器的输出。

三、讨论、心得

本次实验较为简单，很快速就完成了，应当继续保持对此的熟悉程度。

浙江大学

本科实验报告

课程名称：	数字逻辑电路设计
姓　　名：	金祺书
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
指导教师：	洪奇军
报告日期：	2024 年 5 月 11 日

浙江大学实验报告

课程名称：数字逻辑设计 实验类型：综合

实验项目名称：同步时序电路设计

学生姓名：金祺书 学号：3230104248 同组学生姓名：蒋翼泽

实验地点：紫金港东四 509 室 实验日期：2024 年 5 月 11 日

一、操作方法与实验步骤

1、四位同步二进制计数器

四位二进制计数器输入时钟信号，输出为四位二进制值，二进制值从高位到低位分别是Q_d,Q_c,Q_b,Q_a。每一个时钟信号的上升沿，{Q_d,Q_c,Q_b,Q_a}的值就自增。根据功能描述，可以得到真值表及卡诺图如下，其中 Q 表示当前状态（输出），D 表示下一状态（下一个时钟上升沿到来后的输出）：

	Q_A	Q_B	Q_C	Q_D	D_A	D_B	D_C	D_D
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

1	0	0	1
1	0	0	1
1	0	0	1
1	0	0	1

D_A

0	1	0	1
0	1	0	1
0	1	0	1
0	1	0	1

D_B

0	0	1	0
1	1	0	1
1	1	0	1
0	0	1	0

D_C

0	0	0	0
0	0	1	0
1	1	0	1
1	1	1	1

D_D

由卡诺图可以化简得到激励函数如下：

$$D_A = \overline{Q_A}$$

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

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

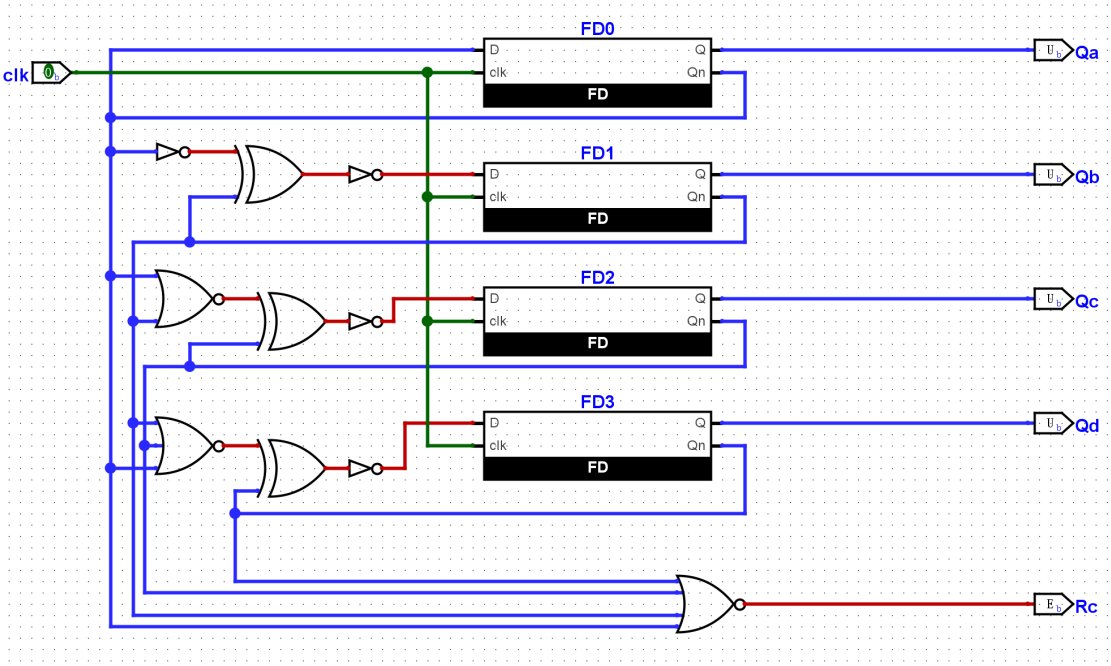
$$D_D = \overline{Q_B}Q_D + \overline{Q_A}Q_D + \overline{Q_C}Q_D + Q_AQ_BQ_C\overline{Q_D} = \overline{Q_AQ_BQ_C}Q_D + Q_AQ_BQ_C\overline{Q_D} = \overline{Q_AQ_BQ_C \oplus \overline{Q_D}}$$

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

进位发生在四位数据均为 1 时，因此可以得到：

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

（1）使用 Logisim 绘制 Counter4b 的原理图，并将其转换成 Verilog Code



（2）电路图中的 FD 为 D 触发器，但我们只需要在 Logisim 中画一个“空壳”即可，导出 Verilog 代码后，将 FD.v 中的内容替换如下：

```
01 module FD(
02     input clk,
03     input D,
04     output Q,
05     output Qn
06 );
07
08     reg Q_reg = 1'b0;
```

```
    always @(posedge clk) begin
        Q_reg <= D;
    end

    assign Q = Q_reg;
    assign Qn = ~Q_reg;

endmodule
```

（3）Vivado 新建工程，对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

```
`timescale 1ns / 1ps
module Counter4b_tb();
//Input
    reg clk;
//Outputs
    wire Qa;
    wire Qb;
    wire Qc;
    wire Qd;
    wire Rc;
//Instantiate the UUT
    Counter4b Counter4b_inst(
        .clk(clk),
        .Qa(Qa),
        .Qb(Qb),
        .Qc(Qc),
        .Qd(Qd),
        .Rc(Rc)
    );
//Initialize Input
    always begin
        clk=0; #10;
        clk=1; #10;
    end
endmodule
```

（4）导出 Verilog 代码，顶层代码 Top 如下：

```
module Top(
    input wire clk,
    output wire LED,
    output wire [7:0] SEGMENT,
    output wire [3:0] AN
);
```

```
07
08      wire Qa;
09      wire Qb;
10      wire Qc;
11      wire Qd;
12      wire [3:0] Hex;
13
14      /* module clk_1s at submodules/clk_1s.v */
15      clk_1s m0(.clk(clk), .clk_1s(clk_1s));
16
17      /* You need to implement module Counter4b */
18      Counter4b
m1(.clk(clk_1s), .Qa(Qa), .Qb(Qb), .Qc(Qc), .Qd(Qd), .Rc(LED));
19
20      assign Hex = {Qd, Qc, Qb, Qa};
21
22      // Please replace module below with your module completed in
Lab 6
23      // Pay attention to the correctness of the module name and
port name
24      // NOTE: SEGMENT and Segement are different port names
25
26      // BTN[0]: LE, valid with value 0
27      // BTN[1]: point, light with value 1
28      // SW[7:4]: AN, light with value 1(AN[i] = ~SW[i+4])
29      // SW[3:0]: number to display
30      DispNum display(.BTN(2'b00), .SW({4'b0001,
Hex}), .SEGMENT(SEGMENT), .AN(AN));
31
32  endmodule
```

子模块 clk_1s 代码如下：

```
01  `timescale 1ns / 1ps
02
03  module clk_1s(
04      input clk,
05      output reg clk_1s
06  );
07
08      reg [31:0] cnt;
09
10      initial begin
11          cnt = 32'b0;
12      end
13
```

```
    wire[31:0] cnt_next;
    assign cnt_next = cnt + 1'b1;

    always @(posedge clk) begin
        if(cnt<50_000_000)begin
            cnt <= cnt_next;
        end
        else begin
            cnt <= 0;
            clk_1s <= ~clk_1s;
        end
    end

endmodule
```

通过约束文件：

```
# Filename: constraints_labA_part1.xdc
## Constraints file for LabA-part1

# Main clock
set_property PACKAGE_PIN AC18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]

create_clock -period 10.000 -name clk [get_ports "clk"]

# LED
set_property PACKAGE_PIN AF24 [get_ports {LED}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED}]

set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]
set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]
set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]
set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]
```

生成比特流下板验证。

2、十六位可逆同步二进制计数器

与“四位可逆同步二进制计数器”相比，十六位可逆同步二进制计数器拓展位宽到 16 位，且添加了一个控制信号 s 用来选择“自增”或“自减”。计数器功能如下：

①在时钟上升沿对输出 cnt 进行修改；当 s＝0 时进行自增，s＝1 时进行自减。

②同步重置信号 rst 高位有效，即在时钟上升沿时若 rst＝1 才进行重置，重置时将 cnt 修改为 0。

③非饱和计数，当前计数若为 16'hFFFF 则自增后为 16'h0；当前计数若为 16'h0 则自减后为 16'hFFFF。

（1）设计模块 RevCounter 代码如下：

```
`timescale 1ns / 1ps

module RevCounter(
    input wire clk,
    input wire rst,
    input wire s,
    output reg [15:0] cnt=0,
    output wire Rc
);
    assign Rc=(~s & (~|cnt)) | (s & (&cnt));
    always@(posedge clk) begin
        if(rst==1)
            cnt=0;
        if(s==0)
           cnt<=cnt+1;
        else
           cnt<=cnt-1;
    end
endmodule
```

（2）在 Vivado 工程中对该模块进行仿真激励，检验该模块功能是否正确，仿真激励代码如下：

```
`timescale 1ns / 1ps

module RevCounter_tb();
//Inputs
```

```
    reg clk;
    reg rst;
    reg s;
//Outputs
    wire[15:0] cnt;
    wire Rc;
//Instantiate the UUT
    RevCounter RevCounter_inst(
        .clk(clk),
        .rst(rst),
        .s(s),
        .cnt(cnt),
        .Rc(Rc)
    );
//Initialize Input
    initial begin
        s=1'b0;
        rst=0; #93;
        rst=1; #5;
        rst=0; #101;
        s=1'b1; #101;
    end
    always begin
        clk=1'b0; #5;
        clk=1'b1; #5;
    end
endmodule
```

（3）导出 Verilog 代码，顶层代码 Top 如下：

```
`timescale 1ns / 1ps

module Top(
    input wire clk,
    input wire [1:0] SW,
    output wire LED,
    output wire [7:0] SEGMENT,
    output wire [3:0] AN
);

    wire[15:0] cnt;
    wire [3:0] Hex;
    wire clk_1s;

    /* module clk_100ms at submodules/clk_1s.v */
    clk_1s clk_div_1s (.clk(clk), .clk_1s(clk_1s));
```

```
17
18      /* You need to implement module RevCounter */
19      RevCounter
counter(.clk(clk_1s), .rst(SW[1]), .s(SW[0]), .cnt(cnt), .Rc(LED));
20
21      // Please replace module below with your module completed in
Lab **7**
22      // imoprt submodules for module DisplayNumber from your prev.
project
23      DisplayNumber
display(.clk(clk), .rst(1'b0), .hexs(cnt), .LEs(4'b0000), .points(4'b
0000), .AN(AN), .SEGMENT(SEGMENT));
24
25  endmodule
```

通过约束文件：

```
01  # Filename: constraints_labA_part1.xdc
02  ## Constraints file for LabA-part1
03
04  # Main clock
05  set_property PACKAGE_PIN AC18 [get_ports clk]
06  set_property IOSTANDARD LVCMOS18 [get_ports clk]
07
08  create_clock -period 10.000 -name clk [get_ports "clk"]
09
10  set_property PACKAGE_PIN AA10 [get_ports {SW[0]}]
11  set_property PACKAGE_PIN AB10 [get_ports {SW[1]}]
12  set_property IOSTANDARD LVCMOS15 [get_ports {SW[0]}]
13  set_property IOSTANDARD LVCMOS15 [get_ports {SW[1]}]
14
15
16  # LED
17  set_property PACKAGE_PIN AF24 [get_ports {LED}]
18  set_property IOSTANDARD LVCMOS33 [get_ports {LED}]
19
20  set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]
21  set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
22  set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
23  set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
24  set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
25  set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
26  set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
27  set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
28  set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
29  set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]
```

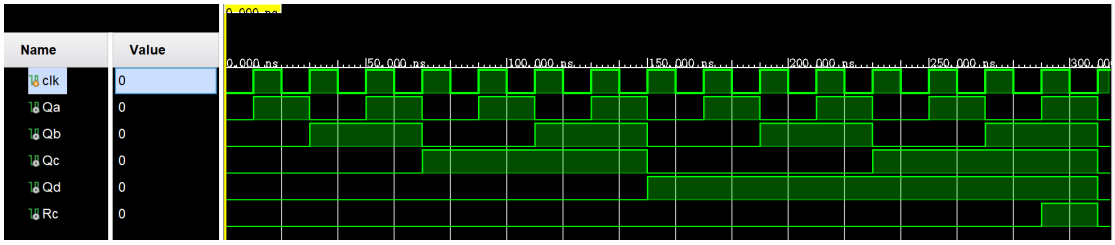
```
30  set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]
31  set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
32  set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
33  set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
34  set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
35  set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
36  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
37  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
38  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
39  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
40  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
41  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
42  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
43  set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]
```

生成比特流下板验证。

二、实验结果与分析

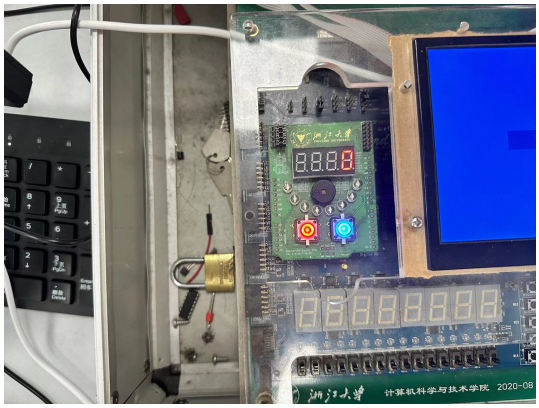
1、四位同步二进制计数器

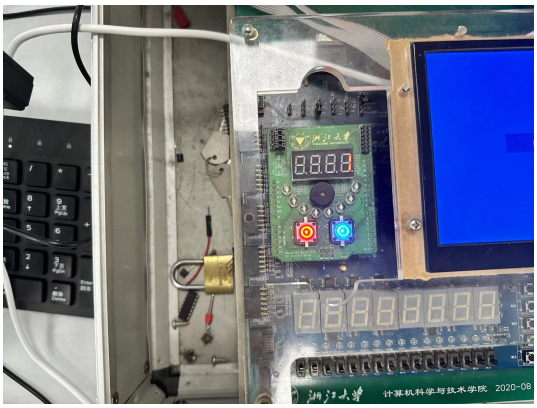
导入 Vivado 后的仿真界面截图如下：

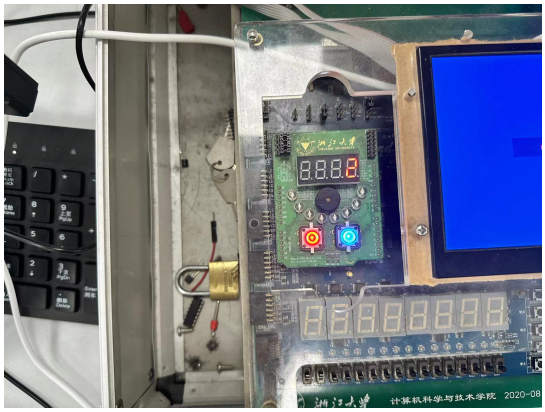
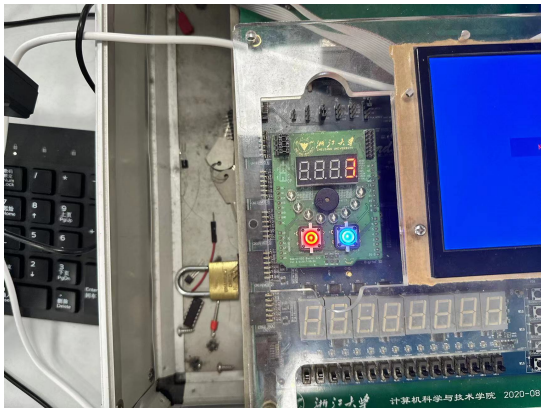


可以看到，只有当 clk 上升沿时，$\{Q_A,Q_B,Q_C,Q_D\}$ 实现自增，在最后$\{Q_A,Q_B,Q_C,Q_D\}$=4'b1111 时且 clk 上升沿时 R_C 置 1，仿真结果符合预期。

下板结果如下图：



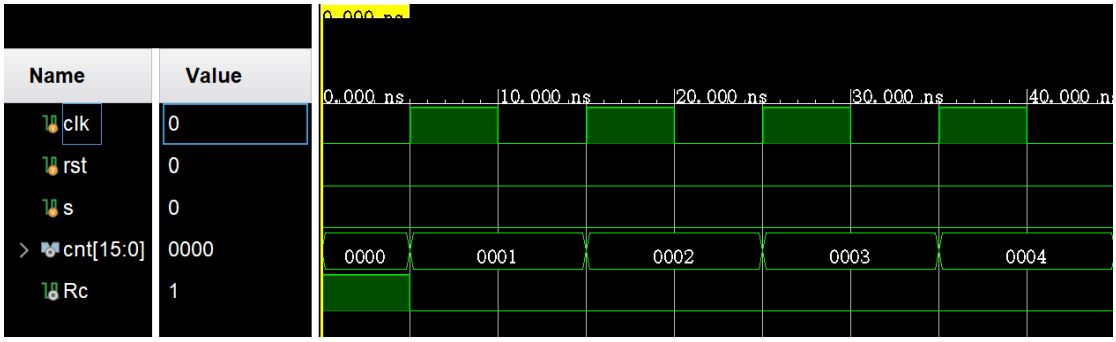


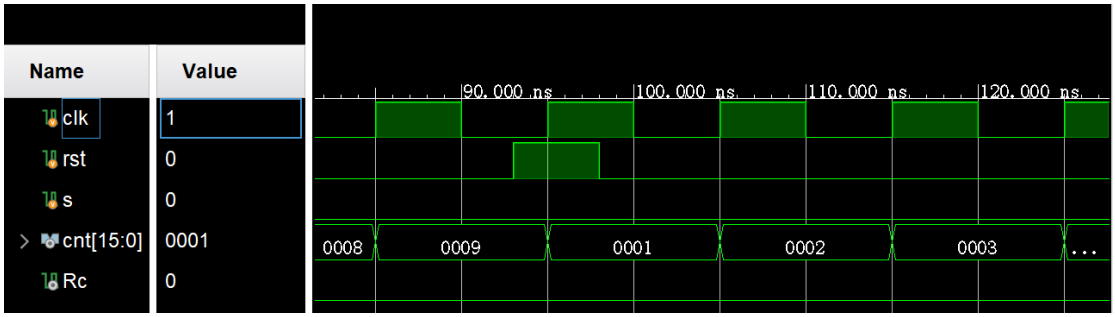



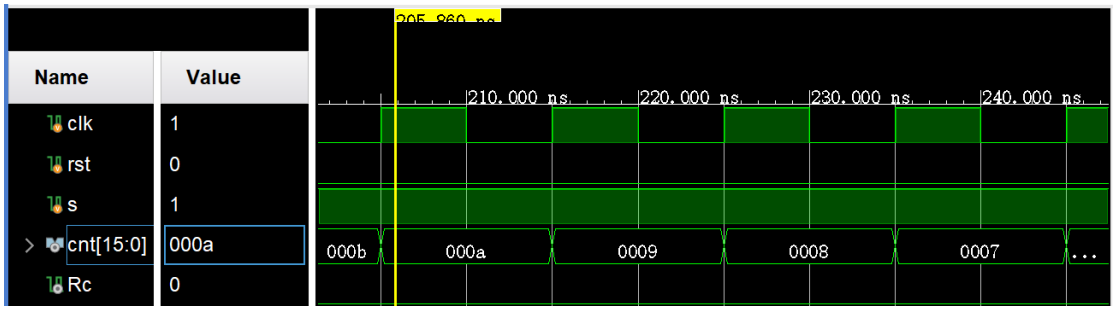
可以看到，当比特流下板后，数字面板上从 0 开始自增，符合预期要求。

2、十六位可逆同步二进制计数器

导入 Vivado 后的仿真界面截图如下：







可以看到，在 rst=0,s=0 时，cnt 实现自增；s=1 时 cnt 实现自减。rst=1 时，cnt 实现复位，由仿真代码可知复位立马执行自增操作，所以在仿真界面上展示的则是复位为 1，符合预期要求。

三、讨论、心得

本次实验有了 lab9 的经验与铺垫整体较为顺利，下板之后的结果也一次就成功符合预期，接下来应当保持，为期末大程做准备。