中国科学技术大学计算机学院
《计算机组成原理实验》报告

实验题目：Lab1 运算器及其应用

学生姓名：___ 吴毅龙_____

学生学号：___PB19111749__

完成日期：___2021/4/7___

计算机实验教学中心制

2020 年 09 月

一、 算术逻辑单元（ALU）

1. 逻辑设计

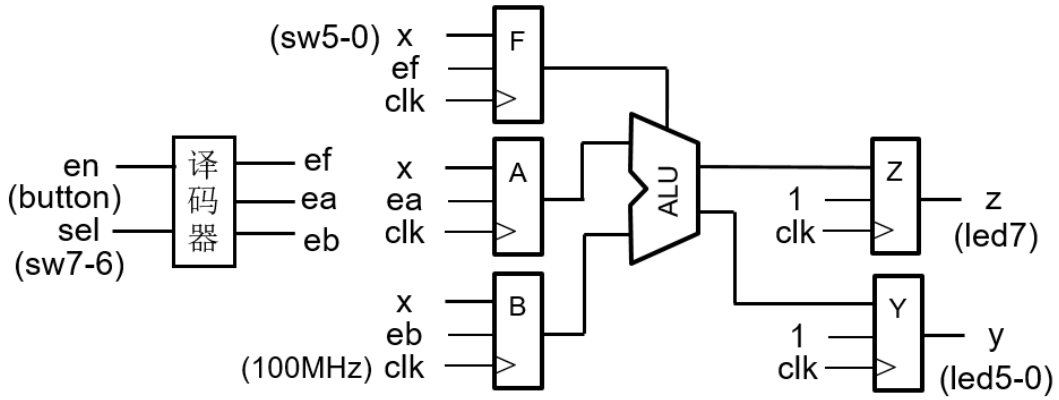
最基本的 ALU 模块比较容易实现，只需要根据输入的运算模式信号对操作数进行加、减、与、或、异或操作即可，输出信号 z 为零标志，当运算结果为 0 时其值为 1，因此只需对结果进行一次或非运算，当结果为 0 时得到 1，不为 0 时得到 0。

需要引起注意的地方是实现 ALU 时，由于硬件的限制需要对操作数、运算模式信号进行分时输入，这里就需要一个 2-4 译码器模块来区分各个输入。具体而言就是：

sel=00 时，ef=1，输入 x 选择操作类型 F：000 为加，001 为减，010 为与，011 为或，100 为异或

sel=01 时，ea=1，输入 x 赋值给操作数 A

sel=10 时，eb=1，输入 x 赋值给操作数 B



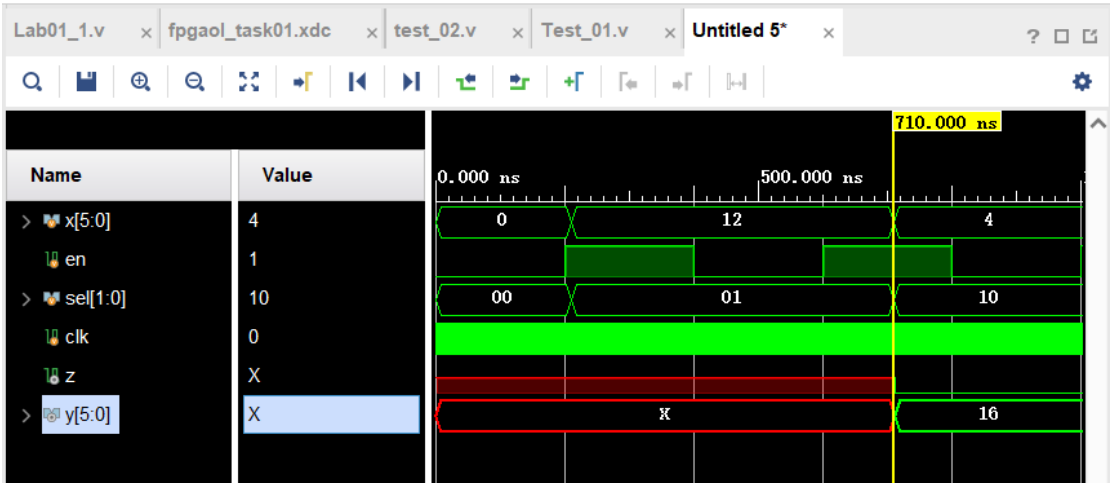
2. 核心代码

```
Decoder Decoder(.code(sel), .out(eout));
ALU #(6)ALU(.a(A), .b(B), .f(F), .z(Z), .y(Y));
assign ef = eout[0] & en;
assign ea = eout[1] & en;
assign eb = eout[2] & en;
always@(posedge clk)
begin
    if(ef) F <= x[2:0];
    if(ea) A <= x;
    if(eb) B <= x;
end
always@(posedge clk)
```

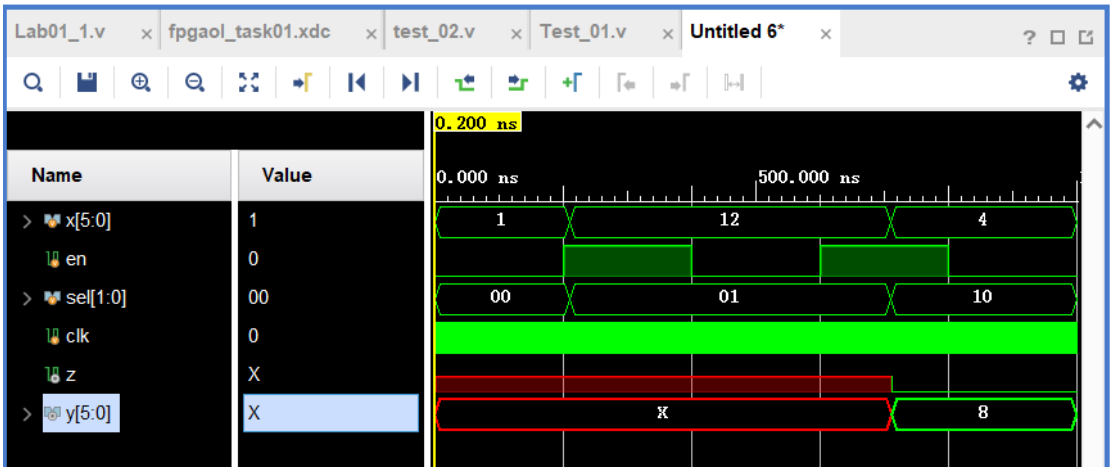
```
13. begin
14.     y <= Y;
15.     z <= Z;
16. end
```

3. 仿真结果

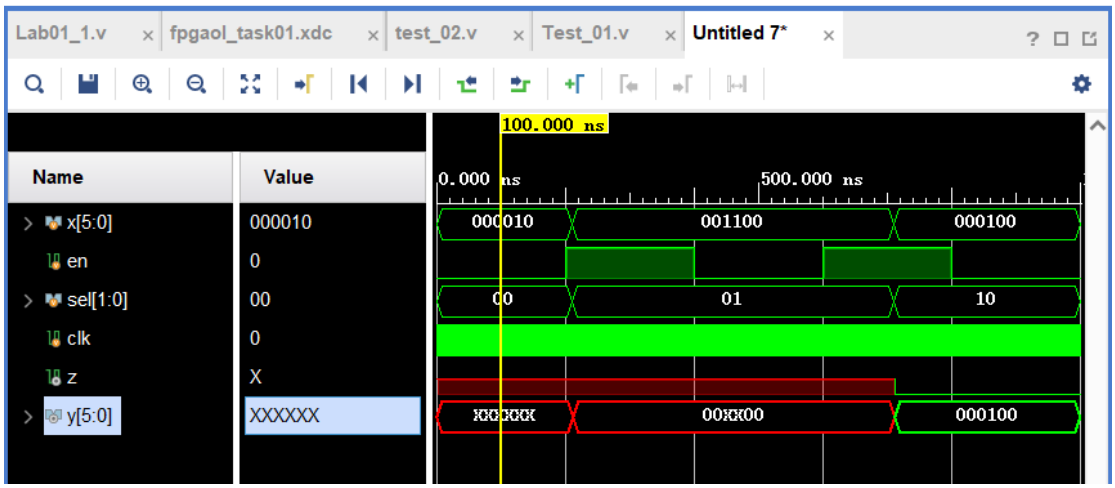
加法运算



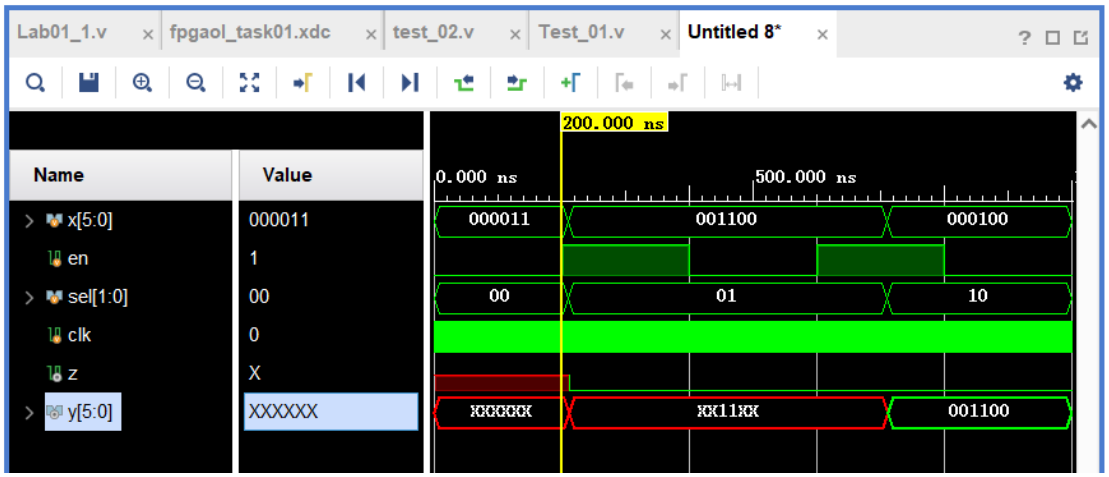
减法运算



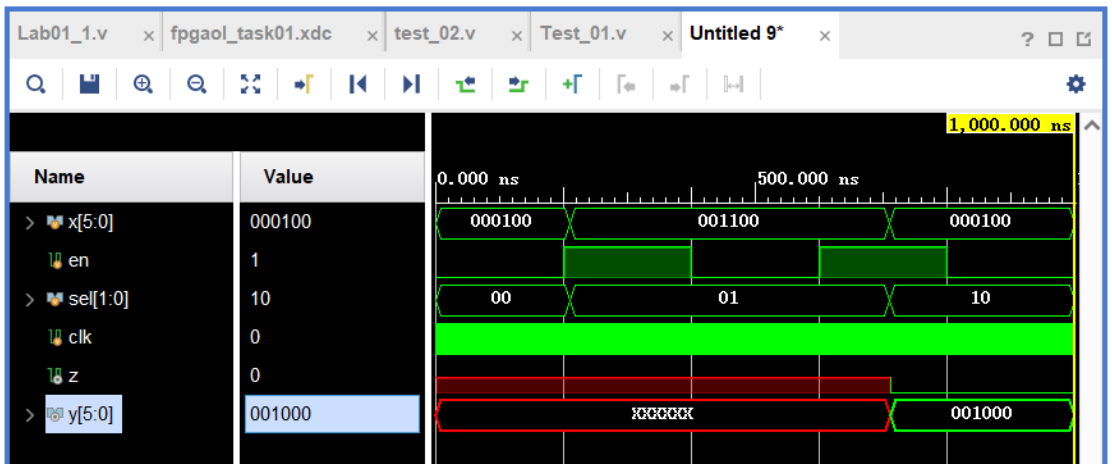
与运算



或运算

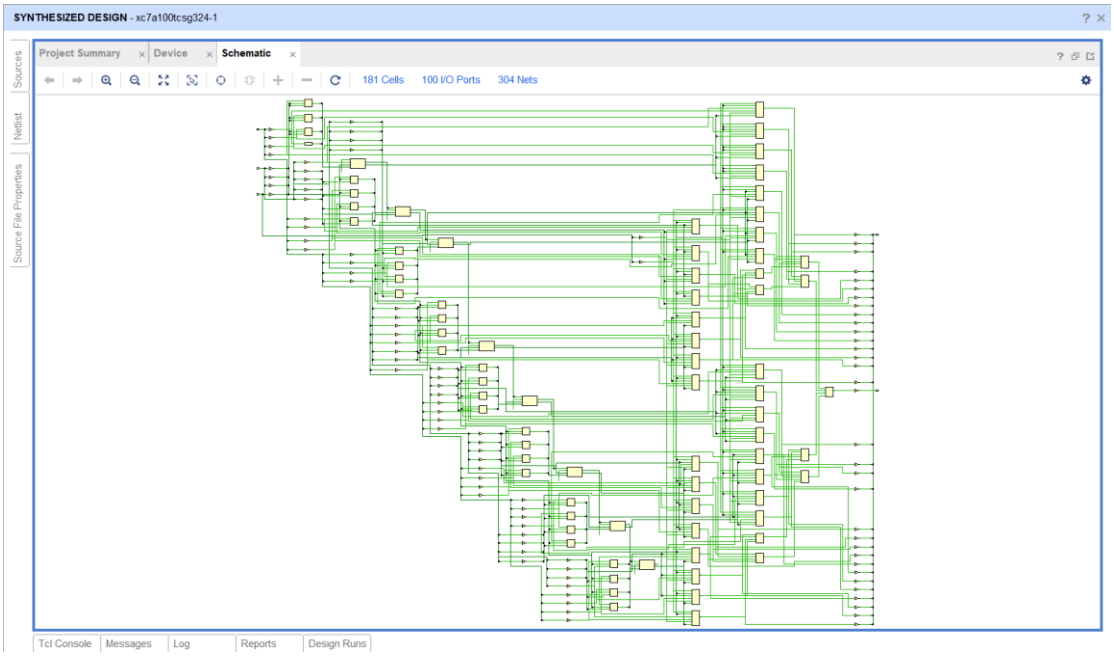


异或运算



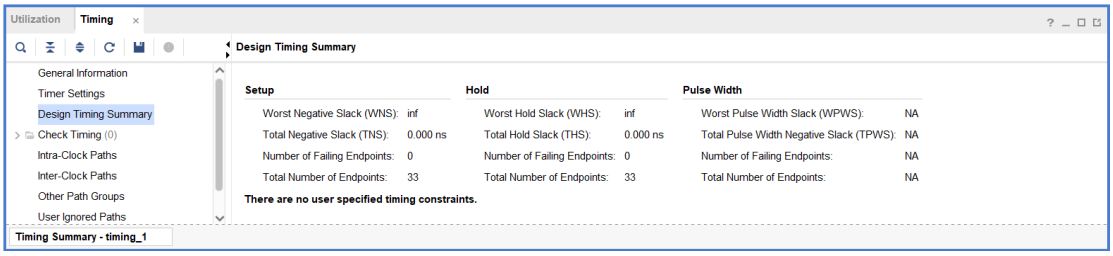
4. 实现后的电路图，电路资源和时间性能报告

32 位 ALU 综合电路



电路资源使用情况

时间性能报告



二、 ALU 的应用：计算斐波那契-卢卡斯数列

1. 逻辑设计

该实验主要考察有限状态机的使用和 ALU 模块的应用。设置 7 个状态，分别为 S0、S1、S2、S3、S4、S5、S6，其中 S0、S1、S2、S3 为赋值状态，S4、S5、S6 为数列计算状态

S0：初始状态，该状态为赋值的初始状态，rst 信号有效进入此状态，f=0,ff=0；en 信号为 0 时维持此状态，en 信号为 1 时进入 S1；

S1：赋值状态同时防抖动，将输入信号 d 赋给 f 和 ff，en 信号为 1 时维持此状态，en 信号为 0 时进入 S2；

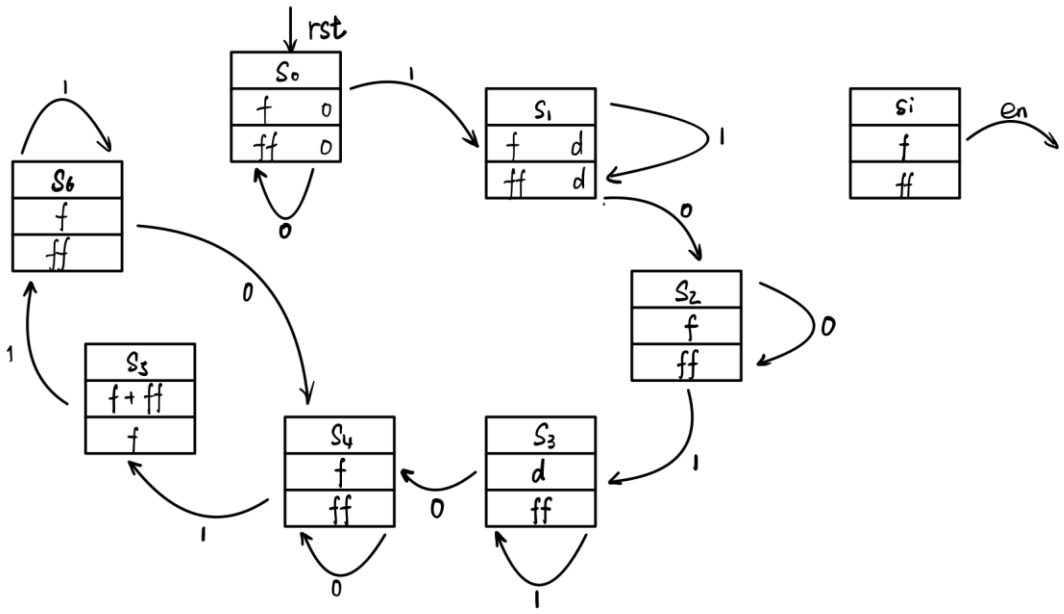
S2：赋值中间状态，en 信号为 0 时维持此状态，en 信号为 1 时进入 S3；

S3：赋值状态同时防抖动，将输入信号 d 赋给 f，en 信号为 1 时维持此状态，en 信号为 0 时进入 S4；

S4：计算中间状态，en 信号为 0 时维持此状态，en 信号为 1 时进入 S5；

S5：计算状态，调用 ALU 模块计算 f+ff，将 f 赋值给 ff，进入 S6；

S6：计算中间状态，en 信号为 1 时维持此状态，en 信号为 0 时进入 S4；

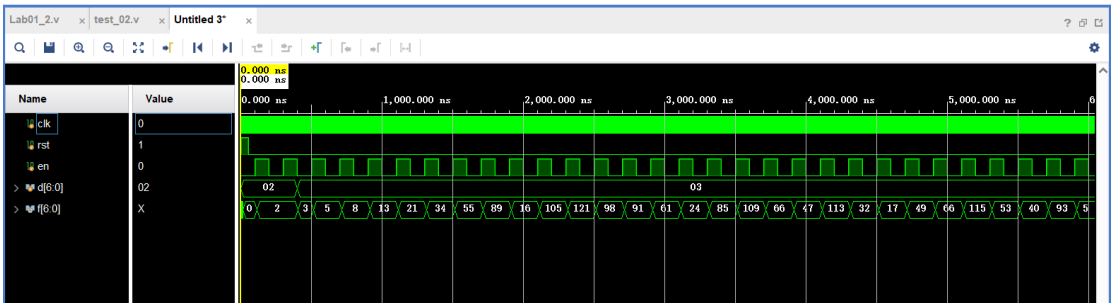


2. 核心代码

```
always @(posedge clk) begin
    if(rst)state <= s0;
    else begin
        case (state)
            s0: begin
                if(en)state <= s1;
                else state <= s0;
            end
            s1: begin
                if(en)state <= s1;
                else state <= s2;
            end
            s2: begin
                if(en)state <= s3;
                else state <= s2;
            end
            s3: begin
                if(en)state <= s3;
                else state <= s4;
```

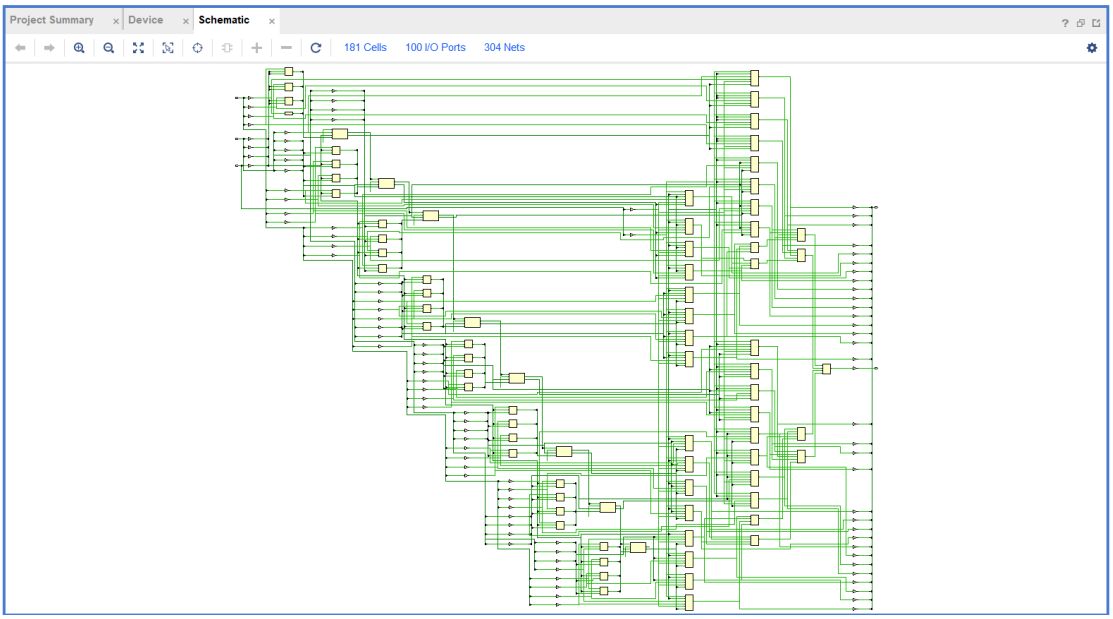
```
            end
            s4: begin
                if(en)state <= s5;
                else state <= s4;
            end
            s5: state <= s6;
            s6: begin
                if(en)state <= s6;
                else state <= s4;
            end
        endcase
    end
end

always @(posedge clk)
begin
    case (state)
        s0: ff <= 0;
        s1: ff <= d;
        s5: ff <= f;
        default: ff <= ff;
    endcase
end

always @(posedge clk)
begin
    case (state)
        s0: f <= 0;
        s1: f <= d;
        s3: f <= d;
        s5: f <= temp;
        default: f <= f;
    endcase
end
```

3. 仿真结果

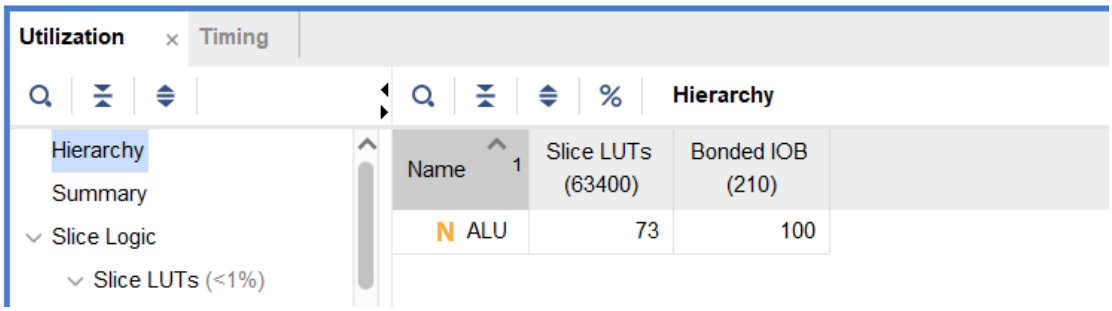


4. 实现后的电路图，电路资源和时间性能报告

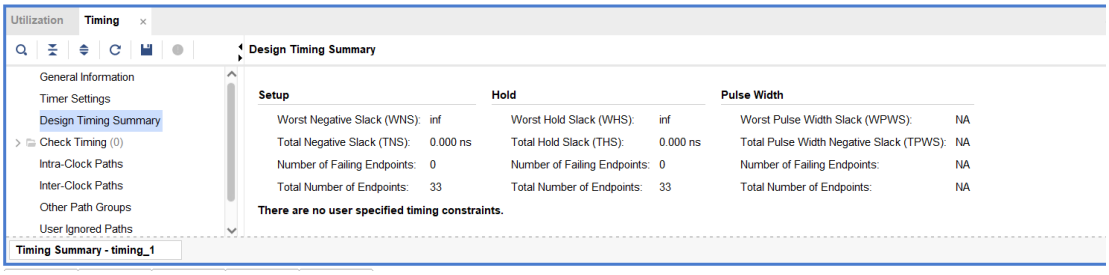
综合电路



电路资源使用情况



综合电路性能



三、 意见和建议

通过本次实验，我对 verilog 的理解和掌握又更近了一步，实验题目的难度层层递进，有基础操作的考核，也有所学知识的综合，难易结合，既有复习又有思考，让所学在实践中得以运用，加深了我对逻辑电路和时序电路知识的

理解。希望今后实验可以保持本次实验中详细实验指导描述的优点，辅助完成每项试验内容。

四、 设计和测试文件源代码

ALU 模块

```
module ALU #(parameter WIDTH = 6)
(input [WIDTH - 1:0]a, b,
 input [2:0]f,
 output reg[WIDTH - 1:0]y,
 output z);
reg[WIDTH:0]result;
always@(*)
begin
    case(f)
        3'b000:result = {0,a} + {0,b};
        3'b001:result = {0,a} - {0,b};
        3'b010:result = {0,a} & {0,b};
        3'b011:result = {0,a} | {0,b};
        3'b100:result = {0,a} ^ {0,b};
        default:result = 0;
    endcase
    y[WIDTH - 1:0] = result[WIDTH - 1:0];
end
assign z = ~| y;
endmodule
```

2-4 译码器模块

```
module Decoder
(input [1:0]code,
output reg[3:0]out);
always@(*)
begin
    case(code)
        2'b00:out = 4'b0001;
        2'b01:out = 4'b0010;
        2'b10:out = 4'b0100;
        2'b11:out = 4'b1000;
    endcase
end
endmodule
```

任务一

```
module Lab01_1(
input [5:0]x,
input en,
input [1:0]sel,     //sel=00 时，ef=1，输入 x 选择操作类型：000 为加，001 为减，
                      010 为与，011 为或，100 为异或
                    //sel=01 时，ea=1，输入 x 赋值给操作数 a
                    //sel=10 时，eb=1，输入 x 赋值给操作数 b
input clk,
output reg z,
output reg [5:0]y);
wire [3:0] eout;
wire ea, eb, ef, Z;
wire [5:0]Y;
reg [5:0]A, B;
reg [2:0]F;
Decoder Decoder(.code(sel), .out(eout));
ALU #(6)ALU(.a(A), .b(B), .f(F), .z(Z), .y(Y));
assign ef = eout[0] & en;
assign ea = eout[1] & en;
assign eb = eout[2] & en;
always@(posedge clk)
begin
    if(ef) F <= x[2:0];
    if(ea) A <= x;
    if(eb) B <= x;
end
always@(posedge clk)
begin
    y <= Y;
    z <= Z;
end

endmodule
```

任务二

```
module Lab01_2(
input clk, rst, en,
input [6:0]d,
output reg [6:0]f);
reg[6:0] ff;
```

```
parameter s0 = 3'h0, s1 = 3'h1, s2 = 3'h2, s3 = 3'h3, s4 = 3'h4, s5 = 3'h5, s6 = 3'h6;
reg[2:0] state;
wire[6:0]temp;
ALU #(7)ALU(.a(f), .b(ff), .f(3'b000), .y(temp), .z());
always @(posedge clk) begin
    if(rst)state <= s0;
    else begin
        case (state)
            s0: begin
                if(en)state <= s1;
                else state <= s0;
            end
            s1: begin
                if(en)state <= s1;
                else state <= s2;
            end
            s2: begin
                if(en)state <= s3;
                else state <= s2;
            end
            s3: begin
                if(en)state <= s3;
                else state <= s4;
            end
            s4: begin
                if(en)state <= s5;
                else state <= s4;
            end
            s5: state <= s6;
            s6: begin
                if(en)state <= s6;
                else state <= s4;
            end
        endcase
    end
end

always @(posedge clk)
begin
    case (state)
        s0:
ff <= 0;
        s1: ff <= d;
```

```
        s5: ff <= f;
        default: ff <= ff;
    endcase
end

always @(posedge clk)
begin
    case (state)
        s0: f <= 0;
        s1: f <= d;
        s3: f <= d;
        s5: f <= temp;
        default: f <= f;
    endcase
end
endmodule
```