Lab1 实验报告

1. ALU模块的逻辑设计和仿真

1.1 逻辑设计

由 ALU 的**状态转化图**（是否叫这个）

s	y	f
0	a - b	*
1	a + b	0
2	a & b	0
3	a | b	0
4	a ^ b	0
5	a >>> b	0
6	a <<< b	0
7	a >>>> b	0

大小关系	f		
	ltu	lt	eq
$a = b$	0	0	1
$a \neq b$	x	x	0
$a <_s b$	x	1	0
$a \geq_s b$	x	0	x
$a <_u b$	1	x	0
$a \geq_u b$	0	x	x

设计时使用参数 `parameter WIDTH = 32`， 可以方便例化不同位宽数据，提高 ALU 模块的使用率。通过组合逻辑，使用 `case(s)` 进行功能选择，对应到不同的运算，实现 ALU 的功能。

对于运算结果 y 的计算，直接使用 verilog 中的运算符进行赋值即可，即采取此类方法： `y = a (opcode) b`。

对于标志 f 的设置，可以采用枚举的方法，直接对其完成设置：

当数据宽度 `WIDTH == 1` 时（虽然感觉这种情况比较少），则 a 和 b 对应有4种情况：

1. a == 0，b == 0　　此时 a 和 b 相等，f 设置为 3'b001
2. a == 0，b == 1　　此时情况为 无符号小于，有符号大于，f 设置为 3'b100
3. a == 1，b == 0　　此时情况为 无符号大于，有符号小于，f 设置为 3'b010
4. a == 1，b == 1　　此时 a 和 b 相等，f 设置为 3'b001

当数据宽度 `WIDTH != 1` 时，则先对 a 和 b 的符号位进行比较，然后对后面 `a[WIDTH-2:0]` 和 `b[WIDTH-2:0]` 进行比较：

1. `a[WIDTH-1] == 0`，`b[WIDTH-1] == 0`
   - `a[WIDTH-2:0] == b[WIDTH-2:0]`　　此时 a 和 b 相等，f 设置为 3'b001
   - `a[WIDTH-2:0] > b[WIDTH-2:0]`　　此时情况为 无符号大于，有符号大于，f 设置为 3'b000
   - `a[WIDTH-2:0] < b[WIDTH-2:0]`　　此时情况为 无符号小于，有符号小于，f 设置为 3'b110
2. `a[WIDTH-1] == 0`，`b[WIDTH-1] == 1`　　此时情况为 无符号小于，有符号大于，f 设置为 3'b100
3. `a[WIDTH-1] == 1`，`b[WIDTH-1] == 0`　　此时情况为 无符号大于，有符号小于，f 设置为 3'b010
4. `a[WIDTH-1] == 1`，`b[WIDTH-1] == 1`
   - `a[WIDTH-2:0] == b[WIDTH-2:0]`　　此时 a 和 b 相等，f 设置为 3'b001
   - `a[WIDTH-2:0] > b[WIDTH-2:0]`　　此时情况为 无符号大于，有符号小于，f 设置为 3'b010
   - `a[WIDTH-2:0] < b[WIDTH-2:0]`　　此时情况为 无符号小于，有符号大于，f 设置为 3'b100

上述便是 f 标志在不同数据情况下的设置。

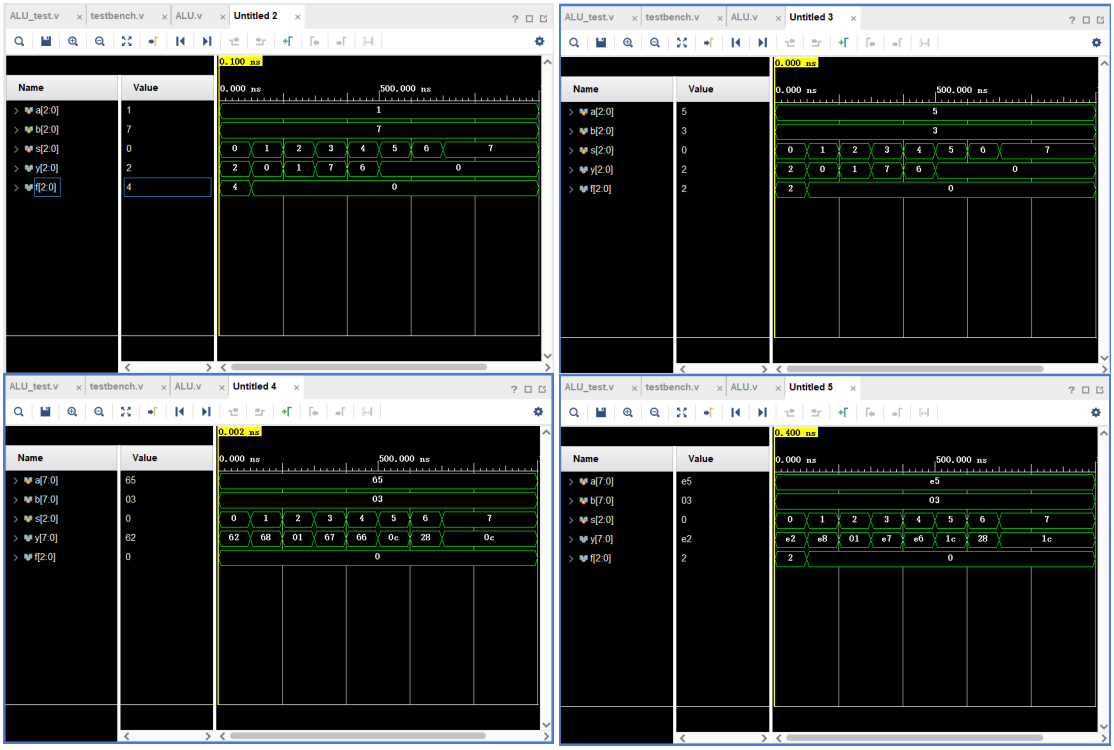
1.2 核心代码

```
module ALU #(parameter WIDTH = 32)(
    input [WIDTH-1:0] a, b,
    input [2:0] s,
    output [WIDTH-1:0] y,
    output [2:0] f
    );
    reg [WIDTH-1:0] alu_y;
    reg [2:0] alu_f;
    always @(*)
    begin
        alu_f = 3'b000;
        case(s)
            3'b000: begin
                alu_y = a - b;
                if (WIDTH == 1) begin
                    if (a > b)
                        alu_f = 3'b010;
                    else if (a == b)
                        alu_f = 3'b001;
                    else
                        alu_f = 3'b100;
                end
                else begin
                    case({a[WIDTH-1], b[WIDTH-1]})
                        2'b00: begin
                            if (a[WIDTH-2:0] < b[WIDTH-2:0])
                                alu_f = 3'b110;
                            else if (a[WIDTH-2:0] == b[WIDTH-2:0])
                                alu_f = 3'b001;
                            else
                                alu_f = 3'b000;
                        end
                        2'b01: begin
                            alu_f = 3'b100;
                        end
                        2'b10: begin
                            alu_f = 3'b010;
                        end
                        2'b11: begin
                            if (a[WIDTH-2:0] < b[WIDTH-2:0])
                                alu_f = 3'b100;
                            else if (a[WIDTH-2:0] == b[WIDTH-2:0])
                                alu_f = 3'b001;
                            else
                                alu_f = 3'b010;
                        end
                    endcase
                end
            end
            3'b001: begin
                alu_y = a + b;
            end
            3'b010: begin
                alu_y = a & b;
            end
```

```
            3'b011: begin
                alu_y = a | b;
            end
            3'b100: begin
                alu_y = a ^ b;
            end
            3'b101: begin
                alu_y = a >> b;
            end
            3'b110: begin
                alu_y = a << b;
            end
            3'b111: begin
                alu_y = a >>> b;
            end
        endcase
    end
    assign y = alu_y;
    assign f = alu_f;
endmodule
```

1.3 仿真结果

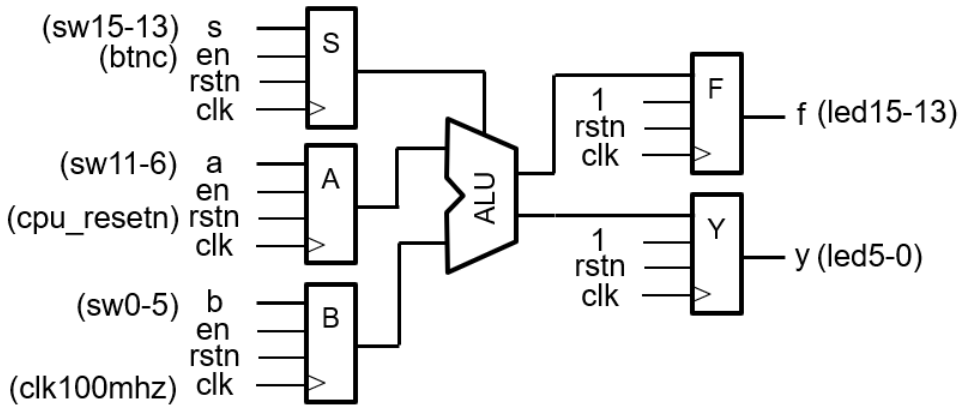
对 ALU 进行4组数据测，前两组是位宽为3的数据，后两组是位宽为8的数据：



2. 6位ALU的下载测试

2.1 逻辑设计

根据 PPT 给出的数据通路，可以分析出我们需要使用5个复位使能的 D 触发器和由前面设计的 ALU 模块：



则我们需要设计一个复位使能的 D 触发器，再将其例化组合就可以得到6位 ALU 的模块。

鉴于这一部分比较简单，就不再赘述相关设计。

2.2 核心代码

2.2.1 DFF

```
module DFF #(parameter WIDTH = 32, RST_VALUE = 0)(
    input clk, rstn, en,
    input [WIDTH-1:0] d,
    output reg [WIDTH-1:0] q
    );
  always @(posedge clk or negedge rstn) begin
        if(!rstn)
            q <= RST_VALUE;
        else if(en)
            q <= d;
        else
            q <= q;
  end
endmodule
```

2.2.2 6ALU

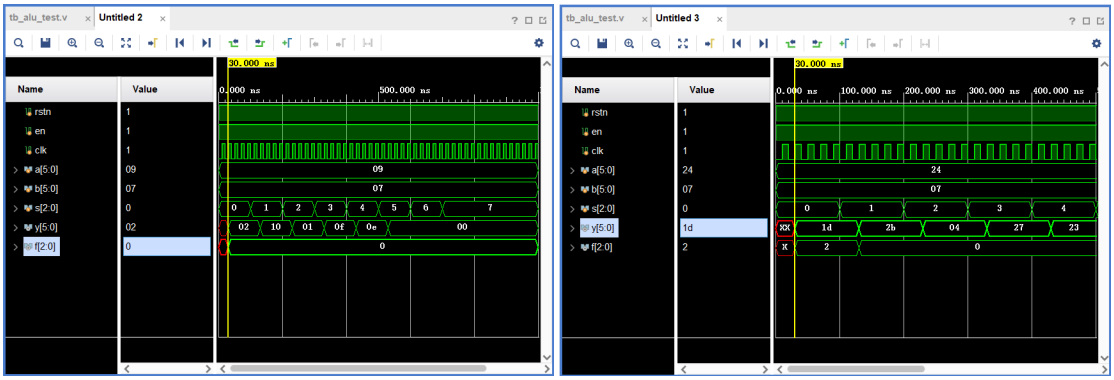
```
module ALU_test(
    input [5:0] a, b,
    input clk100mhz, rstn, en,
    input [2:0] s,
    output [5:0] y,
    output [2:0] f
    );
    wire [2:0] wire_s;
    wire [5:0] wire_a, wire_b;
    wire [5:0] wire_y;
    wire [2:0] wire_f;

    DFF #(3,0) DFF_s (.clk(clk100mhz), .rstn(rstn), .en(en), .d(s), .q(wire_s));
    DFF #(6,0) DFF_a (.clk(clk100mhz), .rstn(rstn), .en(en), .d(a), .q(wire_a));
    DFF #(6,0) DFF_b (.clk(clk100mhz), .rstn(rstn), .en(en), .d(b), .q(wire_b));
```

```
    ALU #(6) ALU_test (.a(wire_a), .b(wire_b), .s(wire_s), .y(wire_y),
.f(wire_f));

    DFF #(6,0) DFF_y (.clk(clk100mhz), .rstn(rstn), .en(1'b1), .d(wire_y),
.q(y));
    DFF #(3,0) DFF_f (.clk(clk100mhz), .rstn(rstn), .en(1'b1), .d(wire_f),
.q(f));
endmodule
```

2.3 仿真结果和下载测试

2.3.1 仿真结果

对6位 ALU 赋两组数据：



2.3.2 下载测试

编写约束文件如下，将数据链接到对应管脚（如6位 ALU 的数据通路图）：

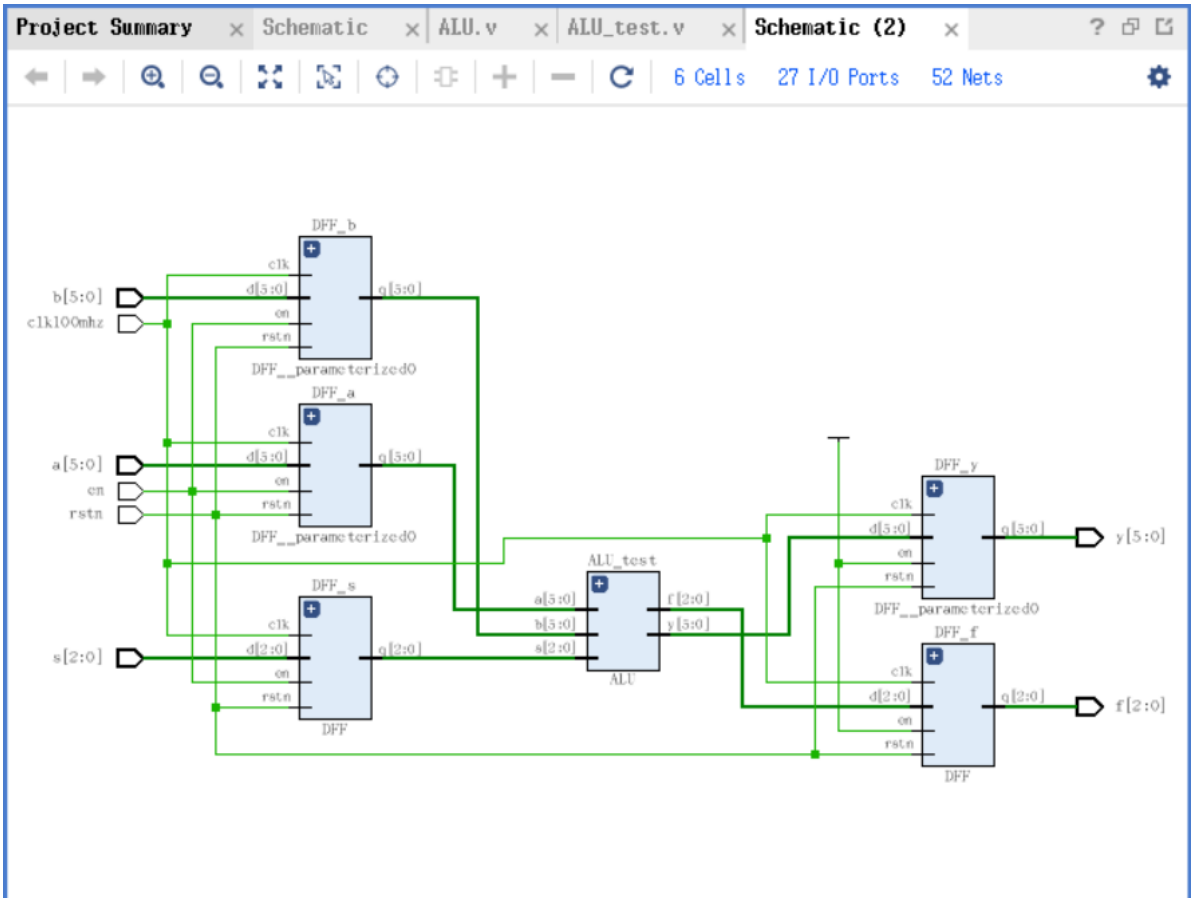
```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk100mhz
}];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {
clk100mhz }];
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { en }];
set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports { rstn }];

set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { s[2] }];
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { s[1] }];
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { s[0] }];

set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { a[5] }];
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { a[4] }];
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS33 } [get_ports { a[3] }];
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS33 } [get_ports { a[2] }];
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { a[0] }];

set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { b[5] }];
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { b[4] }];
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { b[3] }];
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { b[2] }];
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { b[1] }];
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { b[0] }];

set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { y[5] }];
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { y[4] }];
```

```
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { y[3] }];
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { y[2] }];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { y[1] }];
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { y[0] }];

set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { f[2] }];
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { f[1] }];
set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { f[0] }];
```

略略略略略略略略略

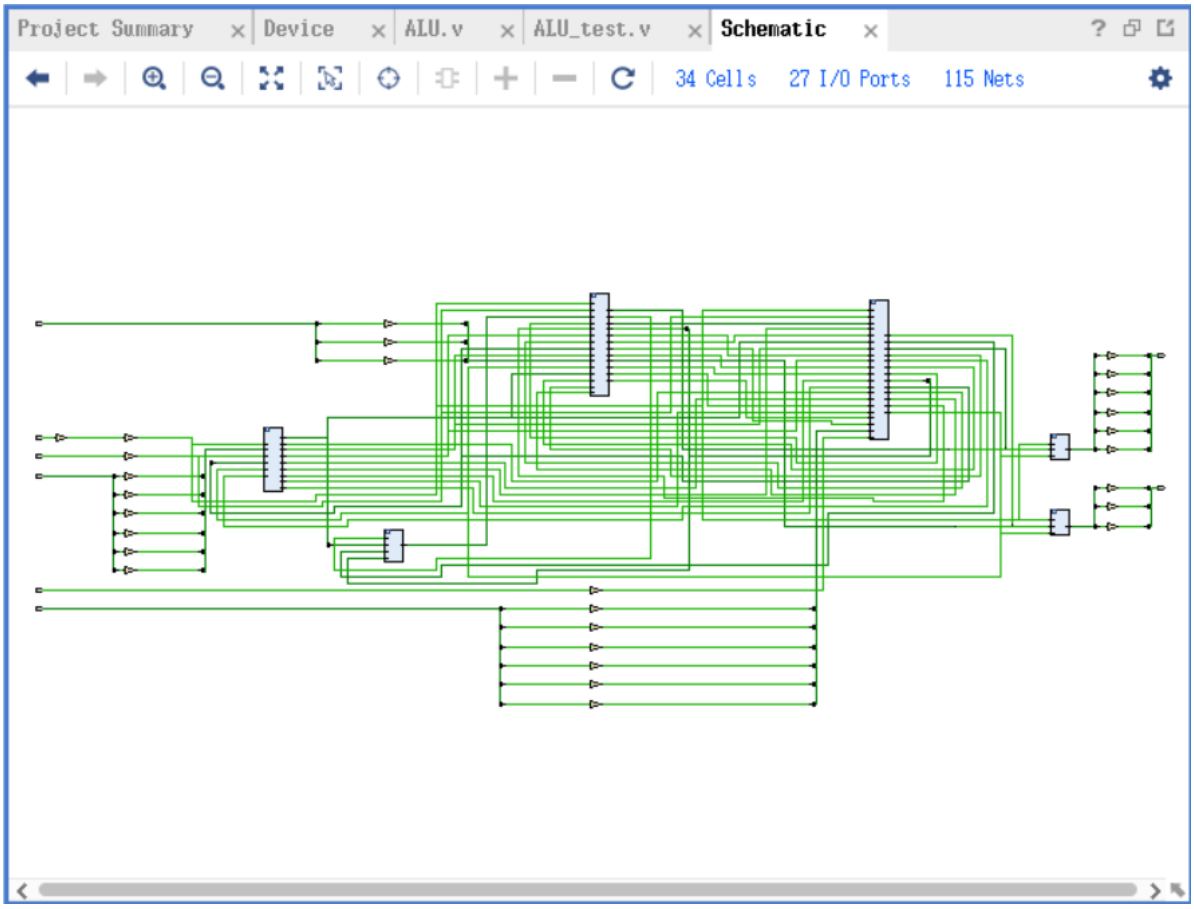
2.4 电路资源和时间性能

2.4.1 RTL电路图

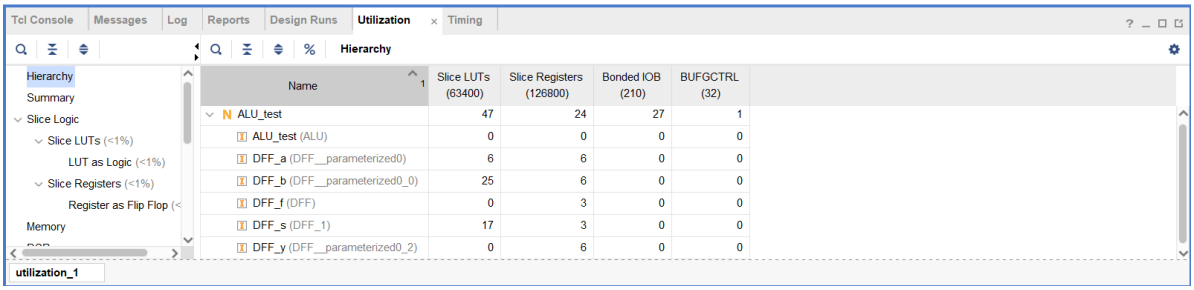


可以看出，这个 RTL 电路图和 PPT 中所给出的电路图几乎是吻合的，其结构形式是一致的。

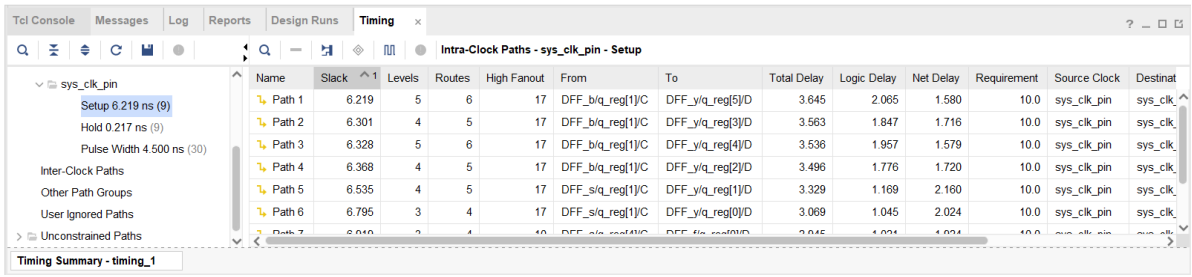
2.4.2 综合电路



2.4.3 电路资源



2.4.4 电路性能



可以看出，Slack 还剩余 6.219秒，电路性能良好。

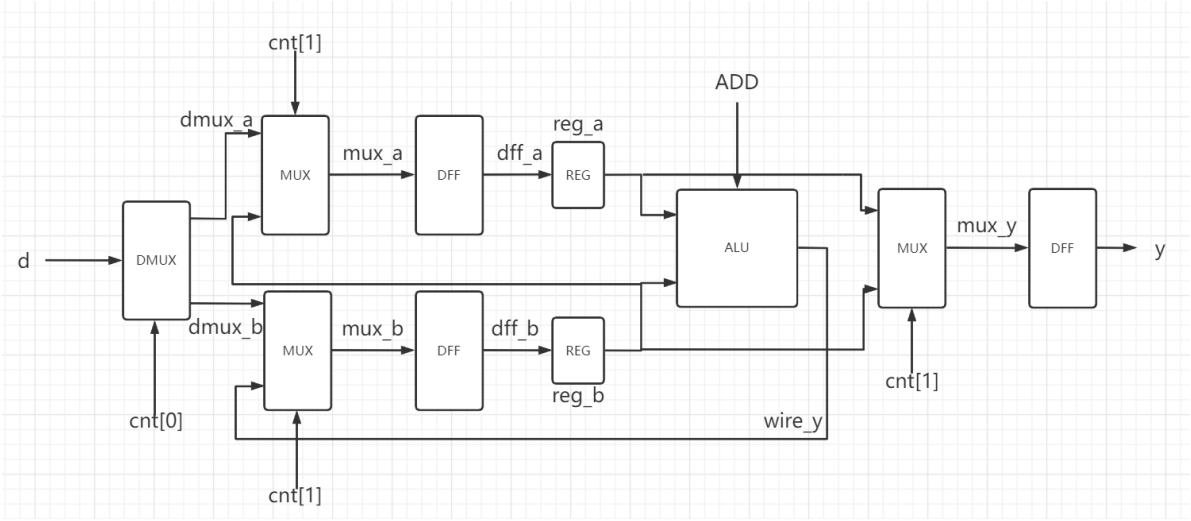
3. FLS的逻辑设计、仿真和下载测试

3.1 逻辑设计

3.1.1 数据通路

从所需的实现功能考虑，首先要能够存储前两次的输入数据（即初始数据），所以我们采用两个寄存器分别为 reg_a 和 reg_b 。然后再接下来的操作中，每次按下使能键时，都可以产生一个新的斐波那契数并且显示出来。所以需要使用 D 触发器来进行 “保留” 数据。为了能够在正确次序中显示正确的斐波那契数，还应该为输出结果加上数据选择器进行选择。同理，再后续的操作中，与输入 d 数据无关，所以对于 reg_a 和 reg_b ，也需要使用数据选择器进行选择：在初始阶段时，选择输入 d ，而在后续阶段时，把数据进行 “移位” 操作，只需要保存计算下一个斐波那契数的两个数据即可。为了能够对其进行选择，我们还使用了一个计数器 reg_cnt 进行计数，以完成对数据的选择和分配。

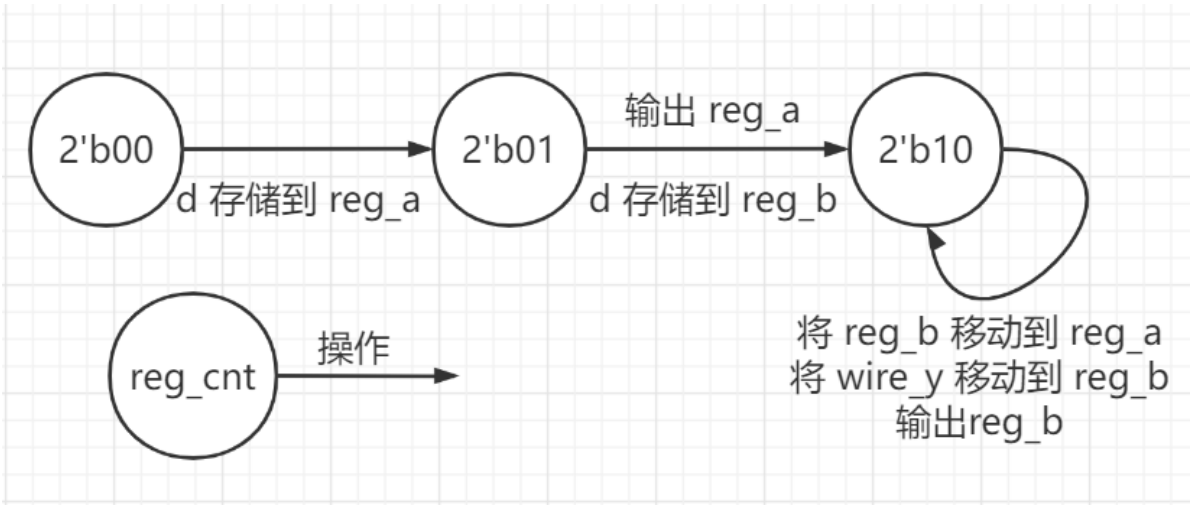
经过多次尝试设计，最终设计出来的数据通路图如下（省略了一些不是很重要的部分，例如 clk，rstn 等等）：



3.1.2 状态转化

初始状态时，`reg_cnt = 2'b00`，此时我需要在 en 按钮按下时（此时`reg_cnt = 2'b01`），将 d 数据存储到 reg_a 中，那么下一次 en 按钮按下之前，应该把 d 数据分配到 mux_b 中，再按下 en 按钮时（此时`reg_cnt = 2'b10`），将 d 数据存储到 reg_b 中，而再后续的操作中，都与 d 不再有任何关系了，所以可以保持计数器不变。同时可以看出，将 ALU 的计算结果存储到了 reg_b 当中，那么输出 y 只需要在 reg_a 和 reg_b 当中选择即可。

由上述分析，可以得知 reg_cnt 的变化就是一个状态转化的过程。用状态图总结上述就是：



当然，当 rstn 有效时，一切全部归0。

3.1.3 去除毛刺

由于使能信号以及复位信号都是由按钮进行的，所以难免会出现毛刺，所以我们需要对输入的信号进行去除毛刺处理。采用上学期数电实验的方法：

```
module jitter_clr(
    input clk,
    input button,
    output button_clean
);
    reg [3:0] cnt;
    always@(posedge clk)
        begin
            if(button == 1'b0)
                cnt <= 4'h0;
            else if(cnt < 4'h8)
                cnt <= cnt + 1'b1;
        end
    assign button_clean = cnt[3];
endmodule
```

由于我们这里所用的复位信号是低电平有效，而按钮按下是会产生高电平的，所以我们对复位信号做了一些不同的处理，详情在核心代码 3.2.3 FLS 中。

3.1.4 取边缘信号

尽管我们按下按钮只持续 "一瞬间"，但是这也会持续多个周期，为了让其只发挥一次作用，我们需要对其取边缘信号，使信号只维持一个周期，避免出现按下一次按钮却完成了多次操作的情况。采用上学期数电实验的方法：

```
module signal_edge(
    input clk,
    input button,
    output button_edge
);
    reg button_r1,button_r2;
    always@(posedge clk)
        button_r1 <= button;
    always@(posedge clk)
        button_r2 <= button_r1;
    assign button_edge = button_r1 & (~button_r2);
endmodule
```

由于我们这里所用的复位信号是低电平有效，而按钮按下是会产生高电平的，所以我们对复位信号做了一些不同的处理，详情在核心代码 3.2.3 FLS 中。

3.2 核心代码

3.2.1 DMUX

```
module DMUX2 #(parameter MSB = 31, LSB = 0)(
    input [MSB: LSB] d,
    input s,
    output reg [MSB: LSB] sel1, sel0
    );
```

```
    always @(*) begin
        if (s == 1'b0) begin
            sel0 = d;
        end
        else begin
            sel1 = d;
        end
    end
endmodule
```

3.2.2 MUX

```
module MUX2 #(parameter MSB = 31, LSB = 0)(
    input [MSB: LSB] sel1, sel0,
    input s,
    output [MSB: LSB] y
    );
    assign y = s ? sel1 : sel0;
endmodule
```

3.2.3 FLS

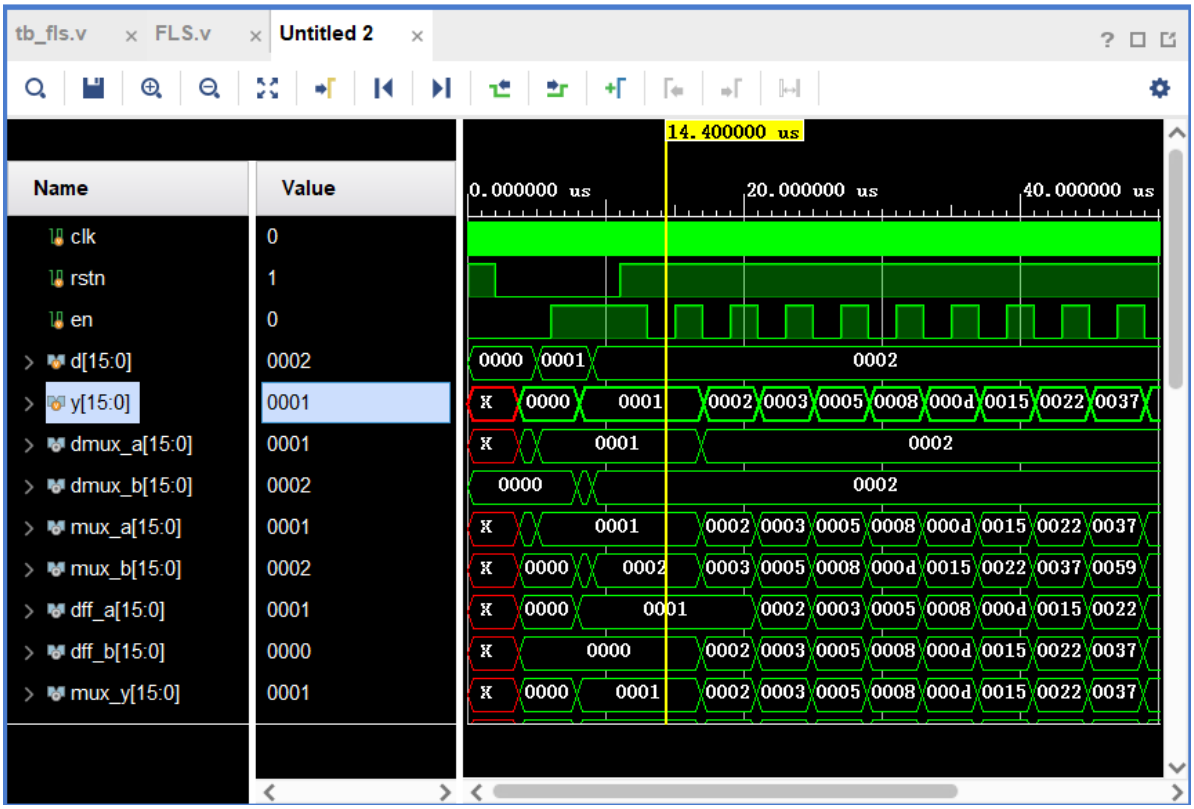
```
module FLS(
    input clk, rstn, en,
    input [15:0] d,
    output reg [15:0] y
    );
    wire [15:0] dmux_a, dmux_b, mux_a, mux_b, dff_a, dff_b, mux_ab, mux_y,
wire_y, out_y;
    wire [2:0] wire_f;
    reg [15:0] reg_a, reg_b;
    reg [1:0] reg_cnt;

    reg [3:0] en_cnt, rstn_cnt;
    wire en_clean, rstn_clean;

    reg en_btn1, en_btn2, rstn_btn1, rstn_btn2;
    wire en_edge, rstn_edge;

    DMUX2 #(16,0) DMUX_d(.d(d), .s(reg_cnt[0]), .sel0(dmux_a), .sel1(dmux_b));

    MUX2 #(16,0) MUX_a(.sel0(dmux_a), .sel1(reg_b), .s(reg_cnt[1]), .y(mux_a));
    MUX2 #(16,0) MUX_b(.sel0(dmux_b), .sel1(wire_y), .s(reg_cnt[1]), .y(mux_b));

    DFF #(16,0) DFF_a(.clk(clk), .rstn(rstn_edge), .en(en_edge), .d(mux_a),
.q(dff_a));
    DFF #(16,0) DFF_b(.clk(clk), .rstn(rstn_edge), .en(en_edge), .d(mux_b),
.q(dff_b));

    ALU #(16) ALU_test (.a(reg_a), .b(reg_b), .s(3'b001), .y(wire_y),
.f(wire_f));

    MUX2 #(16,0) MUX_y(.sel0(reg_a), .sel1(reg_b), .s(reg_cnt[1]), .y(mux_y));

    DFF #(16,0) DFF_y(.clk(clk), .rstn(rstn_edge), .en(1'b1), .d(mux_y),
.q(out_y));
```

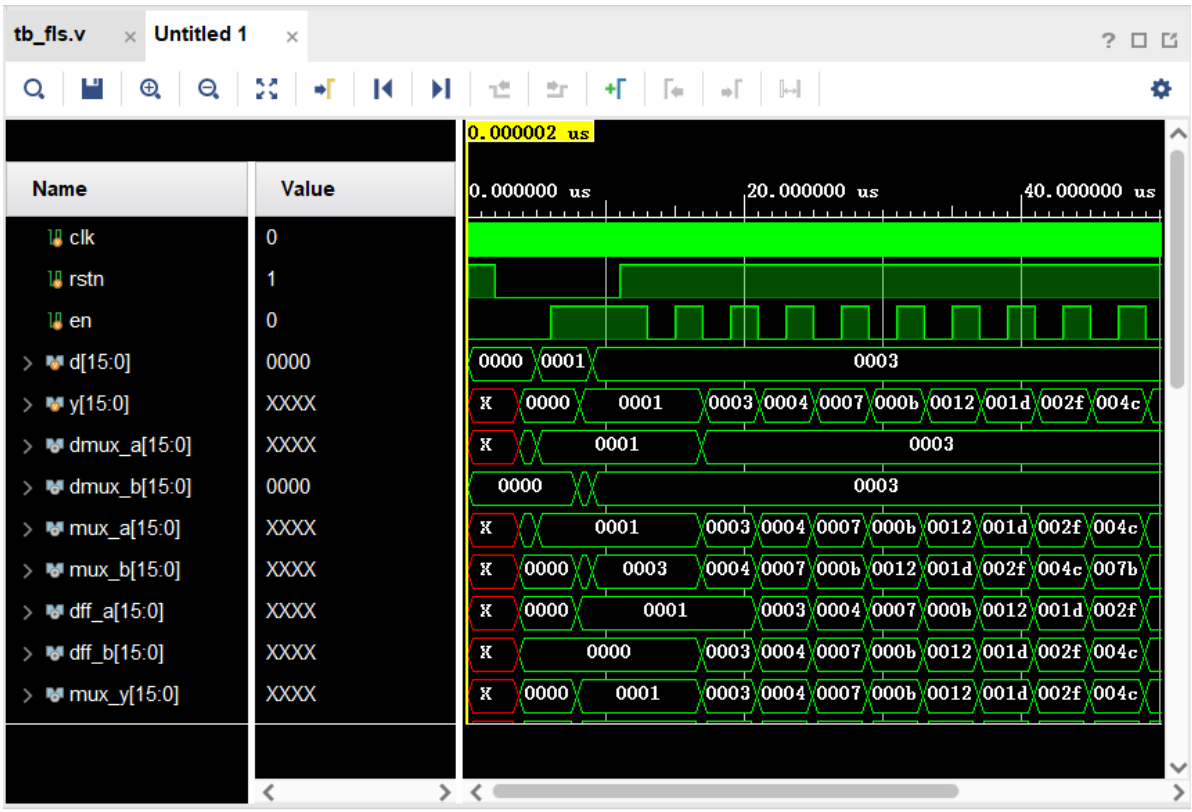
```
    always@(*) begin
        y = out_y;
        reg_a = dff_a;
        reg_b = dff_b;
    end

    always@(posedge clk) begin
        if (en == 1'b0)
            en_cnt <= 4'h0;
        else if (en_cnt < 4'h8)
            en_cnt <= en_cnt + 1'b1;
        else
            en_cnt <= en_cnt;
    end
    assign en_clean = en_cnt[3];

    always@(posedge clk) begin
        if (rstn == 1'b1)
            rstn_cnt <= 4'h0;
        else if (rstn_cnt < 4'h8)
            rstn_cnt <= rstn_cnt + 1'b1;
        else
            rstn_cnt <= rstn_cnt;
    end
    assign rstn_clean = ~rstn_cnt[3];

    always@(posedge clk) begin
        en_btn1 <= en_clean;
        en_btn2 <= en_btn1;
        rstn_btn1 <= rstn_clean;
        rstn_btn2 <= rstn_btn1;
    end
    assign en_edge = en_btn1 & (~en_btn2);
    assign rstn_edge = ~((~rstn_btn1) & rstn_btn2);

    always@(posedge clk or negedge rstn_edge) begin
        if (!rstn_edge)
            reg_cnt = 2'b00;
        else if (reg_cnt == 2'b10)
            reg_cnt = reg_cnt;
        else if (en_edge == 1'b1)
            reg_cnt = reg_cnt + 1'b1;
        else
            reg_cnt = reg_cnt;
    end
endmodule
```

3.3 仿真结果和下载测试

3.3.1 仿真结果

选用两组数据 `16'b1, 16'b10` 和 `16'b1, 16'b11` 进行仿真，仿真结果如下：





3.3.2 下载测试

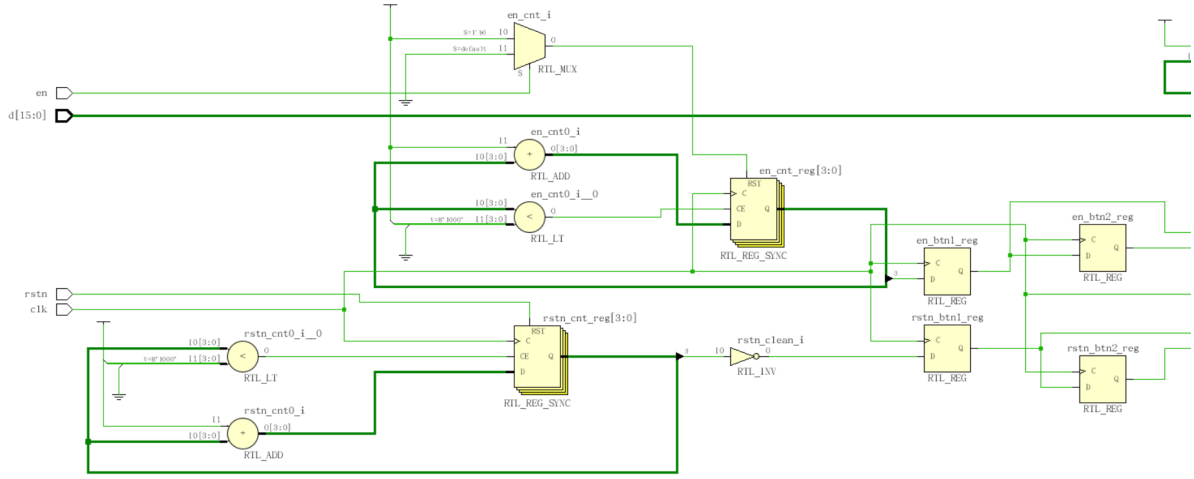
编写约束文件如下，将数据链接到对应管脚

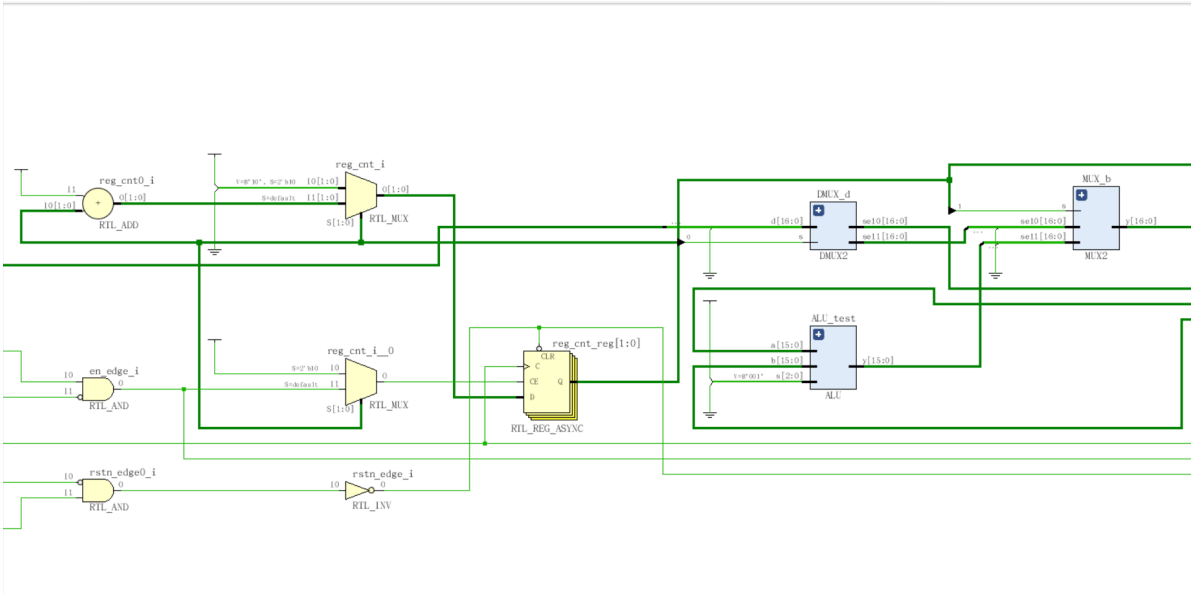
```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {
clk }];
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { en }];
set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports { rstn }];
```

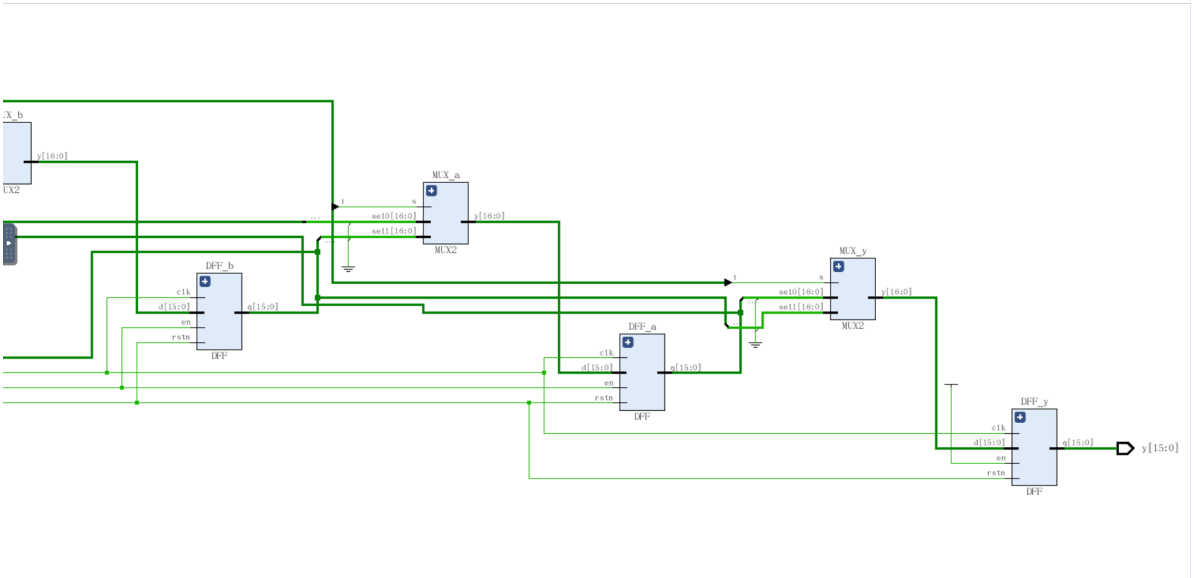
```
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { d[15]
}];
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { d[14]
}];
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { d[13]
}];
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { d[12] }];
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { d[11]
}];
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { d[10]
}];
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS33 } [get_ports { d[9] }];
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS33 } [get_ports { d[8] }];
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { d[7] }];
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { d[6] }];
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { d[5] }];
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { d[4] }];
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { d[3] }];
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { d[2] }];
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { d[1] }];
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { d[0] }];

set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { y[15]
}];
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { y[14]
}];
set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { y[13]
}];
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { y[12]
}];
set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { y[11]
}];
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { y[10]
}];
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { y[9] }];
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { y[8] }];
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { y[7] }];
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { y[6] }];
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { y[5] }];
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { y[4] }];
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { y[3] }];
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { y[2] }];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { y[1] }];
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { y[0] }];
```

3.4 电路资源和时间性能

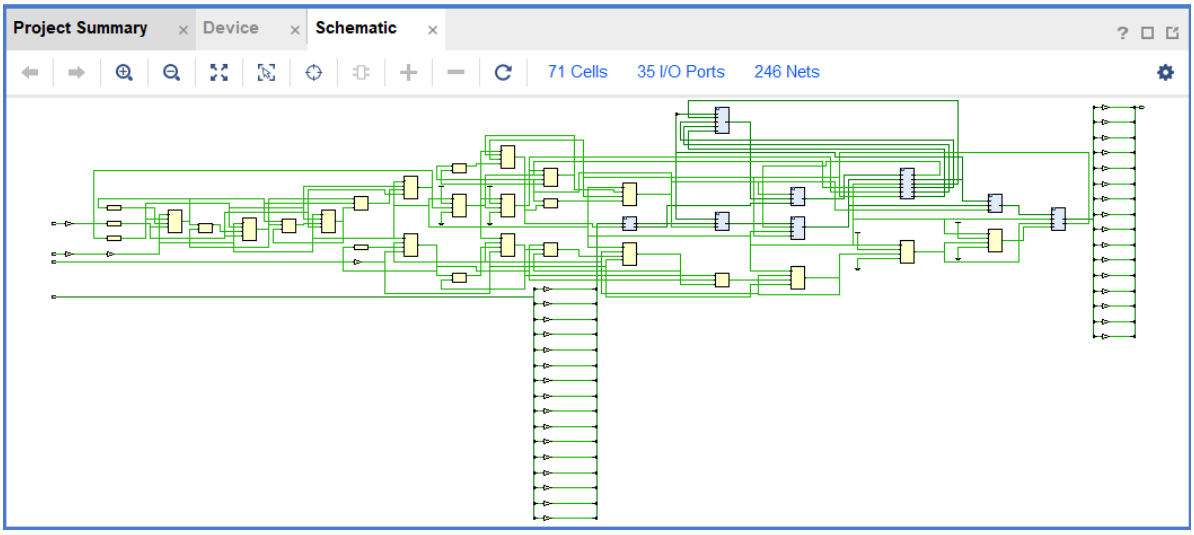
3.4.1 RTL电路图


en_cnt_i
RTL_MUX
en
d[15:0]
en_cnt0_i
RTL_ADD
en_cnt0_i__0
RTL_LT
en_cnt_reg[3:0]
RTL_REG_SYNC
en_btn1_reg
RTL_REG
en_btn2_reg
RTL_REG
rstn
clk
rstn_cnt0_i__0
RTL_LT
rstn_cnt_reg[3:0]
RTL_REG_SYNC
rstn_clean_i
RTL_INV
rstn_btn1_reg
RTL_REG
rstn_btn2_reg
RTL_REG
rstn_cnt0_i
RTL_ADD


reg_cnt0_i
RTL_ADD
reg_cnt_i
RTL_MUX
DMUX_d
DMUX2
MUX_b
MUX2
ALU_test
ALU
reg_cnt_i__0
RTL_MUX
reg_cnt_reg[1:0]
RTL_REG_ASYNC
en_edge_i
RTL_AND
rstn_edge0_i
RTL_AND
rstn_edge_i
RTL_INV


MUX_a
MUX2
DFF_b
DFF
MUX_y
MUX2
DFF_a
DFF
DFF_y
DFF
y[15:0]

3.4.2 综合电路



3.4.3 电路资源

Reports | Design Runs | **Utilization**

Hierarchy

Name	Slice LUTs (63400)	Slice Registers (126800)	Bonded IOB (210)	BUFGCTRL (32)
FLS	51	94	35	1
ALU_test (ALU)	0	0	0	0
DFF_a (DFF)	0	16	0	0
DFF_b (DFF_0)	17	16	0	0
DFF_y (DFF_1)	1	16	0	0
DMUX_d (DMUX2)	0	32	0	0
MUX_a (MUX2)	8	0	0	0
MUX_b (MUX2_2)	8	0	0	0
MUX_y (MUX2_3)	8	0	0	0

3.4.4 电路性能

Design Runs | **Timing** | Utilization

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.752 ns	Worst Hold Slack (WHS):	0.115 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	154	Total Number of Endpoints:	154	Total Number of Endpoints:	63

All user specified timing constraints are met.

4. 实验总结

1. 本次实验进行了 ALU 模块的编写，FLS 的应用，对 Verilog 设计语言起了一定的复习作用。同时温故了几种常见的信号处理技巧，例如信号整形去除毛刺、取信号上下边缘技巧。并且巩固了状态机

的设计方法，最终能够将实现功能在开发板上体现出来，本次实验采用的就是摩尔型电路来对 FLS 进行设计。最后还学习了一些新的知识。包括数据通路和控制器的设计方法和了解查看电路性能和资源使用情况。在实现 FLS 过程中，需要自己凭借状态机来实现数据通路的设计。然后可以通过查看电路性能和资源使用情况，了解自己设计的电路图的优良性以及可行性。

2. 建议：本次实验内容其实和数电实验内容并没有什么差别，希望可以提前几周开始实验，然后为后面实验多留点时间。