计算机组成原理(H) 实验

PB20111699 吴骏东

一、实验题目

Lab01 运算器及其应用

二、实验目的

1. 熟练掌握算术逻辑单元 (ALU) 的功能；
2. 掌握数据通路和控制器的设计方法；
3. 掌握组合电路和时序电路，以及参数化和结构化的Verilog描述方法；
4. 了解查看电路性能和资源使用情况。

三、实验平台

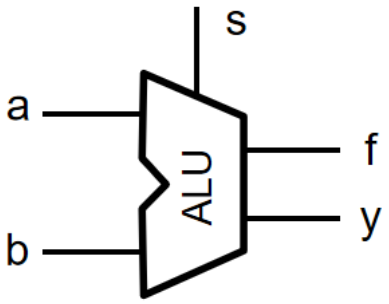
- Vivado 2019
- NEXYS4 DDR 开发板

四、实验过程

4.1 ALU 模块设计

算术逻辑单元 ALU 是计算机中进行算数运算的核心单元。本实验中 ALU 模块的相关功能如下：

- s：功能选择，加、减、与、或、异或、逻辑左移、逻辑右移、算术右移等运算
- a, b：两个操作数
- y：运算结果，和、差 ……
- f：标志，相等(eq)，小于(lt, ltu)



ALU 模块的参数描述如下所示：

```
module ALU
#(
    parameter WIDTH = 32                                //数据宽度
)
(
    input [WIDTH-1 : 0] num1, num2,                     // [数据] 两操作数
    input [3:0] sel,                                    // [信号] 功能选择
                                                        //改成了四位
    output reg [WIDTH-1 : 0] ans,                       // [数据] 运算结果
    output [2:0] sub_flag,                              // [信号] 减法大小标志
    output reg error                                    // [信号] 错误标志，为 1 代表运算
出错
);
```

在 PPT 所做规范的基础上，对部分变量名称与位宽进行了调整，使模块更加具有拓展性。以下是部分实现细节。

4.1.1 模式约定

为了便于后续功能添加与区分，实验中对所有可能的功能模式进行了编号区分。本实验中所设计的 ALU 模块模式表为：

```
    localparam SUB = 4'd00;
    localparam ADD = 4'd01;
    localparam AND = 4'd02;
    localparam OR = 4'd03;
    localparam XOR = 4'd04;
    localparam RMV = 4'd05;       //逻辑右移（符号位置0） 移位时对 num2 >= WIDTH 的情况
处理为置0
    localparam LMV = 4'd06;
    localparam ARMV = 4'd07;      //算数右移（保留符号位）
    // 4'd08 ~ 4'd14 为保留模式代码
    localparam TEST = 4'd15;      //调试模式
```

4.1.2 结果输出标志

为了便于对输入的数值进行大小比较， ALU 模块设计了专门的大小比较结果输出端口，包括：无符号数小于、有符号数小于、等于。模块以`sub_flag[2:0]`信号作为输出，其中`sub_flag[0]`对应相等（num1 = num2），`sub_flag[1]`对应有符号数小于（num1 < num2），`sub_flag[2]`对应无符号数小于（num1 < num2）。代码实现如下：

```
    assign sub_flag[0] = (num1 == num2 ? 1 : 0);          //等于
    assign sub_flag[1] = ((num1[WIDTH-1] == 1 && num2[WIDTH-1] == 1 && num1 >
num2) ||
                          (num1[WIDTH-1] == 0 && num2[WIDTH-1] == 0 && num1 <
num2) ||
                          (num1[WIDTH-1] == 1 && num2[WIDTH-1] == 0) )? 1 : 0;

    //有符号数小于(lt), a < b
    assign sub_flag[2] = ((num1[WIDTH-1] == 0 && num2[WIDTH-1] == 1) ||
                          (num1[WIDTH - 1] == num2[WIDTH-1] && num1 < num2)) ? 1
: 0;
    //无符号数小于(ltu), a < b     //认为是无符号原码
```

4.1.3 数值计算

为了实现不同的计算过程，程序根据输入的`sel`信号进行判断，执行对应的计算过程。代码如下：

```
always @(*) begin
    error = 0;
    case(sel)
        SUB: begin
            ans = num1 - num2;
        end

        ADD: begin
            ans = num1 + num2;
        end

        AND: begin
            ans = num1 & num2;
        end

        OR: begin
            ans = num1 | num2;
        end

        XOR: begin
            ans = num1 ^ num2;
        end

        RMV: begin
            ...
        end

        LMV: begin
            ...
        end

        ARMV: begin
            ...
        end

        default : begin
            ans = 0;
            error = 1;
```

```
        end

    endcase
end
```

加、减、按位与、按位或、按位异或的实现相对较为简单。逻辑左移与逻辑右移在 Verilog 中也有对应的运算符。实验中为了防治可能的意外情况，对 `num2` （位移位数）的情况进行了额外判断。以逻辑右移为例，当移位位数超出当前位宽时，运算结果将被自动置零。

```
if (num2 >= WIDTH) begin
    ans = {WIDTH{1'b0}};
end
else begin
    ans = num1 >> num2;
end
```

对于算数右移过程，程序在开始对 `num1` 的正负进行了判断，并根据结果进行相应的补位操作。

```
temp = 'b0;
counter = num1[WIDTH-1];        //记录最高位正负
if (num2 >= WIDTH) begin
    if (counter == 0)
        ans = {WIDTH{1'b0}};
    else
        ans = {WIDTH{1'b1}};
end
else begin
    temp = num1 >> num2;        //先执行逻辑右移

    if (counter == 1)           //针对负数的情况单独补1
        ans = temp | ({WIDTH{1'b1}} << num2);
end
```

4.1.4 ALU top 模块设计

设计完 ALU 模块主体后，本实验设计了其对应的 6-bit 数据主模块。参数描述如下：

```
/*
 * 6-bit算数逻辑单元的测试程序
 * 由开关进行数据输入， LED进行数据输出，七段数码管显示当前计算模式。
 * sw[15]~sw[12] 模式选择
 * sw[11]~sw[6] num1
 * sw[5]~sw[0] num2
 * button 控制输入
 * led[15] error
 * led[14]~led[9] 置零
 * led[8]~led[6] sub_flag
 * led[5]~led[0] ans
*/


module topmodule(
    input sys_clk,
    input sys_but,          //高电平有效按钮信号
```

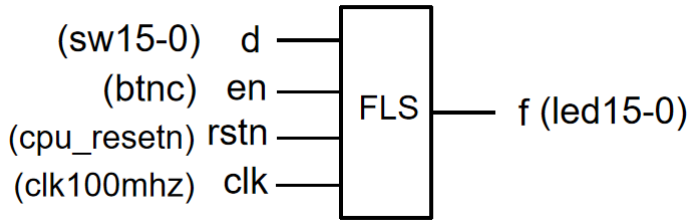
```
    input [15:0] sw,          //开关信号
    input rst_n,              //低电平有效异步复位信号    //本实验中尚未实装
    output reg [15:0] led,
    output reg [6:0] seg,
    output reg [7:0] an
);
```

其中包含了对 ALU 模块的例化：

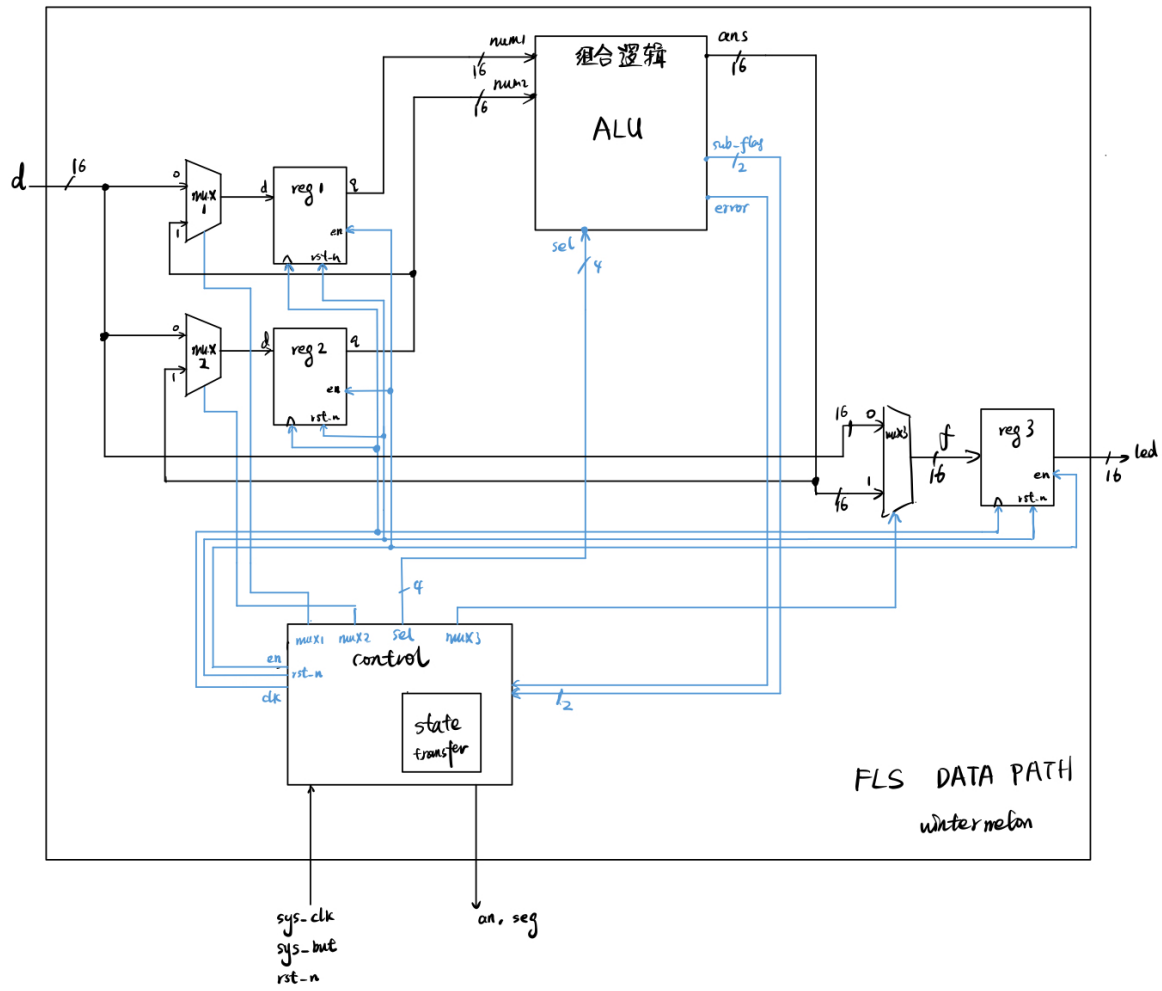
```
ALU #(6) alu(num1, num2, sel, ans, sub_flag, error);
```

4.2 FLS 模块设计

FLS 模块为斐波那契数列计算器。复位后，其可以以外部输入的前两个数为数列前两项，按照要求逐步输出后续项。具体描述为：前 2 次 `en` 有效时，`f` 分别输出 `f0` 和 `f1 (= d)`。 随后，每当 `en` 有效时， `f` 依次输出 `fn = fn-2 + fn-1,  n > 1`。



为了便于模块化设计与综合，我先设计了 FLS 对应的数据通路。如下图所示，其中**黑色代表数据线**，**蓝色代表信号线。**整个 FLS 模块包括 MUX 、 REG 、ALU 、Control 四个子模块。以下是对 FLS 模块的拆解介绍。



4.2.1 MUX 数据选择器

是一个十分基本的可变位宽二选一数据选择器。实现如下：

```
module MUX2
#(
    parameter DATA_WIDTH = 32           // 数据信号宽度
)
(
    input [DATA_WIDTH-1 : 0] data1, data2,  // sel 置 1 则选择 data1
    input  sel,
    output [DATA_WIDTH-1 : 0] out
);

    assign out = (sel) ? data1 : data2;
endmodule
```

4.2.2 REG 数据寄存器

是一个十分基本的可变位宽数据寄存器。其本质上是一个扩展的门控 D 触发器。实现如下：

```
module Dff
#(
    parameter DATA_WIDTH = 32           // 数据信号宽度
)
```

```
(
    input [DATA_WIDTH-1 : 0] d,
    input  clk, rst_n, en,
    output reg [DATA_WIDTH-1 : 0] q
);
    initial begin
        q <= 'b0;
    end

    always @(posedge clk or negedge rst_n) begin
        if (~rst_n) begin
            q <= 0;
        end
        else if (en) begin
            q <= d;
        end
        else
            q <= q;
    end
endmodule
```

4.2.3 Ctrl 控制器

控制器负责**外部信号接收、状态机控制、内部信号发送**三个核心功能。其参数列表如下：

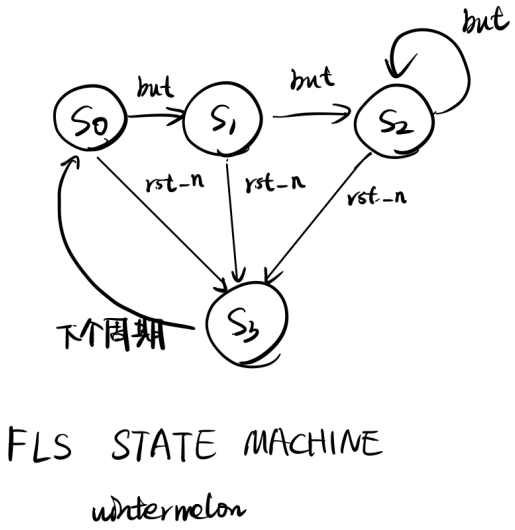
```
/*
 * lab1 控制单元
   根据按钮控制状态转换
   发出相应的控制信号
   rst_n 为复位信号(低电平有效)
   sys_but 为使能信号(高电平有效)
   sys_clk 为系统时钟
   seg、an 为七段数码管信号
   ctrl 为内部控制信号
*/

module Lab1_ctrl(
    input sys_clk,
    input sys_but,
    input rst_n,
    input [2:0]sub_flag,
    input error,
    output reg [6:0] seg,
    output reg [7:0] an,
    output reg [12:0] ctrl
);
```

外部信号接收

控制器模块时刻检测外部的按钮信号和复位信号。当按钮信号发出时，控制器操控状态机进行状态跳转，程序对应输出数列的某一项。复位信号发出时，控制器会让状态机跳转至复位状态，清空 FLS 内部全部缓存后跳转回初始状态。

状态机控制

本实验采用Moore型状态机，采用两段式进行状态机的描述。状态机状态跳转图如下所示：



其中，S0为初始状态，获取用户输入数列的第一项并将其输出；S1为预备状态，获取用户输入数列的第二项并将其输出；S2为循环状态，每当用户按下按钮后程序输出数列的下一项；S3为复位状态，将所有寄存器清零、信号重置，并在下一个时钟周期自动跳转会初始状态S0。

下面是状态机的相关实现细节：

```
//状态机设定
localparam S0 = 2'b00;
localparam S1 = 2'b01;
localparam S2 = 2'b10;
localparam S3 = 2'b11;

always @(posedge sys_clk or posedge rst_flag) begin   //part1 更改 cs
    if (rst_flag) begin
        current_state = S3;
    end
    else
        current_state = next_state;
end

always @(*) begin   //part2 确定 ns 和 out
        if (but_flag) begin
            case (current_state)
                S0: next_state = S1;
                S1: next_state = S2;
                S2: next_state = S2;
                S3: next_state = S0;
            endcase
        end
        else if (current_state == S3)
                next_state = S0;
        else
            next_state = current_state;

        case (current_state)        //对 ctrl 信号的描述
            S0: ctrl <= ...;
            S1: ctrl <= ...;
```

```
            S2: ctrl <= ...;
            S3: ctrl <= ...;
        endcase
    end
```

内部信号发送

本实验中，控制器会根据当前状态调整 ctrl 信号的值。设计细节如下：

```
/* 控制信号一览：
    ctrl[0] - mux1
    ctrl[1] - mux2
    ctrl[2] - mux3
    ctrl[3] - reg1_enable
    ctrl[4] - reg1_rst_n
    ctrl[5] - reg2_enable
    ctrl[6] - reg2_rst_n
    ctrl[7] - reg3_enable
    ctrl[8] - reg3_rst_n
    ctrl[12: 9] - ALU_sel
*/
case (current_state)
    S0: ctrl <= {{5'b0001_1},{but_flag},{7'b00_11_000}};
    //选择器选择外部输入，寄存器1载入，寄存器2清零，寄存器3根据按钮信号进行数据载入
    S1: ctrl <= {{5'b0001_1},{but_flag},{7'b11_10_000}};
    //选择器选择外部输入，寄存器1保持，寄存器2载入，寄存器3根据按钮信号进行数据载入
    S2: ctrl <= {{5'b0001_1},{but_flag},{1'b1},{but_flag},{1'b1},{but_flag},
{3'b111}};
    //选择器选择循环输入，寄存器根据按钮信号进行数据载入
    S3: ctrl <= 13'b0001_00_00_00_000;
    //寄存器全部清0
endcase
```

此外，控制器模块中还设置了七段数码管输出以显示当前状态机状态。实现细节如下：

```
/七段数码管计时器
always @(posedge sys_clk) begin      //循环扫描
    if (clk400 > 'd49999) begin
        clk400 = 'b0;
        if (seg_cnt == 'd7) begin
            seg_cnt <= 'b0;
        end
        else
            seg_cnt <= seg_cnt + 'b1;
    end
    else begin
        clk400 <= clk400 + 'b1;
    end
end

always @(*) begin
    case (seg_cnt)       //选择激活数码管，并确定各位数据
        'd0: begin an <= 8'b11111110; seg_data <= current_state; end
```

```
        'd1: begin an <= 8'b11111101; seg_data <= 'b0; end
        'd2: begin an <= 8'b11111011; seg_data <= 'b0; end
        'd3: begin an <= 8'b11110111; seg_data <= 'b0; end
        'd4: begin an <= 8'b11101111; seg_data <= 'b0; end
        'd5: begin an <= 8'b11011111; seg_data <= 'b0; end
        'd6: begin an <= 8'b10111111; seg_data <= 'b0; end
        'd7: begin an <= 8'b01111111; seg_data <= 'b0; end
    endcase
    case (seg_data)     //七段数码管数字显示
        'd0: seg <= 7'b0000001;  //0
        'd1: seg <= 7'b1001111;  //1
        'd2: seg <= 7'b0010010;  //2
        'd3: seg <= 7'b0000110;  //3
        'd4: seg <= 7'b1001100;  //4
        'd5: seg <= 7'b0100100;  //5
        'd6: seg <= 7'b0100000;  //6
        'd7: seg <= 7'b0001111;  //7
        'd8: seg <= 7'b0000000;  //8
        'd9: seg <= 7'b0000100;  //9
       default: seg <= 7'b0000001;
    endcase
end
```

4.2.4 FLS top 模块设计

FLS 的顶层模块主要负责上述子模块的实例化。设计代码如下：

```
module FLS_top(
    input sys_clk,
    input sys_but,          //高电平有效按钮信号
    input rst_n,            //高电平有效复位信号
    input [15:0] sw,        //开关信号（数据输入）
    output [6:0] seg,
    output [7:0] an,
    output [15:0] led   //LED (数据输出)
);

wire [2:0] sub_flag;
wire error;
wire [15:0] d1, d2, q1, q2; //寄存器信号
wire [15:0] ans;    //ALU输出信号
wire [15:0] f;
wire [12:0] ctrl;

Lab1_ctrl ctrl_unit(sys_clk, sys_but, rst_n, sub_flag, error, seg, an, ctrl);

MUX2 #(16) mux_1(q2, sw[15:0], ctrl[0], d1);
MUX2 #(16) mux_2(ans, sw[15:0], ctrl[1], d2);
MUX2 #(16) mux_3(ans, sw[15:0], ctrl[2], f);

Dff #(16) reg_1(d1, sys_clk, ctrl[4], ctrl[3], q1);
Dff #(16) reg_2(d2, sys_clk, ctrl[6], ctrl[5], q2);
Dff #(16) reg_3(f, sys_clk, ctrl[8], ctrl[7], led[15:0]);

ALU #(16) alu(q1, q2, ctrl[12:9], ans, sub_flag, error);
```

```
endmodule
```

4.3 引脚约束文件设计

本实验采用了所附的 NEXYS4 DDR 引脚约束文件，按照需要对端口进行了修改。设计文件如下：

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports {
sys_clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{sys_clk}];

##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { sw[0]
}];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sw[1]
}];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sw[2]
}];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { sw[3]
}];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { sw[4]
}];
#IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { sw[5]
}];
#IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { sw[6]
}];
#IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { sw[7]
}];
#IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { sw[8]
}];
#IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { sw[9]
}];
#IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { sw[10]
}]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { sw[11]
}];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { sw[12]
}];
#IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { sw[13]
}];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
```

```
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { sw[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { sw[15] }];
#IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
#IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { led[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { led[9] }];
#IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { led[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { led[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { led[12] }];
#IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { led[13] }];
#IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { led[14] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { led[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { seg[6] }];
#IO_L24N_T3_A00_D16_14 Sch=ca
```

```
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { seg[5]
}];
#IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { seg[4]
}];
#IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { seg[3]
}];
#IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { seg[2]
}];
#IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { seg[1]
}];
#IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { seg[0]
}];
#IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { an[0]
}];
#IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { an[1]
}];
#IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { an[2]
}];
#IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { an[3]
}];
#IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { an[4]
}];
#IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { an[5]
}];
#IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { an[6]
}];
#IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { an[7]
}];
#IO_L23N_T3_A02_D18_14 Sch=an[7]


##Buttons
set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { rst_n
}];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports {
sys_but }];
#IO_L9P_T1_DQS_14 Sch=btnc
```

五、实验结果

5.1 ALU 仿真

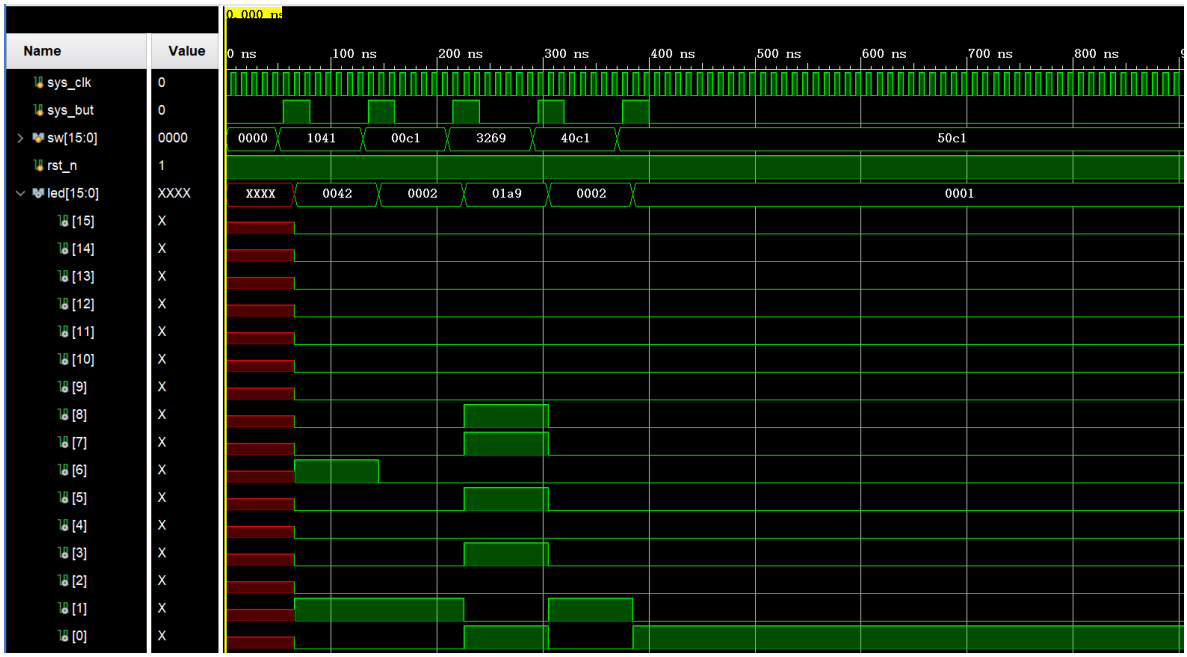
ALU top 的仿真文件如下：

```
initial begin
    sw <= 'b0;
    sys_but <= 'b0;
    sys_clk <= 'b0;
    rst_n <= 'b1;


    #50 sw <= 16'b0001_000001_000001;
    #5 sys_but <= 'b1;
    #25 sys_but <= 'b0;

    #50 sw <= 16'b0000_000011_000001;
    #5 sys_but <= 'b1;
    #25 sys_but <= 'b0;

    #50 sw <= 16'b0011_001001_101001;
    #5 sys_but <= 'b1;
    #25 sys_but <= 'b0;


    #50 sw <= 16'b0100_000011_000001;
    #5 sys_but <= 'b1;
    #25 sys_but <= 'b0;

    #50 sw <= 16'b0101_000011_000001;
    #5 sys_but <= 'b1;
    #25 sys_but <= 'b0;

end

always #5 sys_clk <= ~sys_clk;
```

这是一个简单的 6-bit ALU。按照先后顺序，ALU计算的内容为：

```
1. sw = 0x1041: 1+1
2. sw = 0x00c1: 3 - 1
3. sw = 0x3269:   001001 | 101001
4. sw = 0x40c1: 3 xor 1
5. sw = 0x50c1: 3 >> 1
```

仿真结果如下图所示：



我们可以读出`led[5:0]`的内容为：

```
1. 1 + 1 = 2
2. 3 - 1 = 2
3. 001001 | 101001 = 101001
4. 3 xor 1 = 2
5. 3 >> 1 = 1
```

由此可见，算数计算结果完全正确。同时，我们可以读出`sub_flag`的输出结果为：

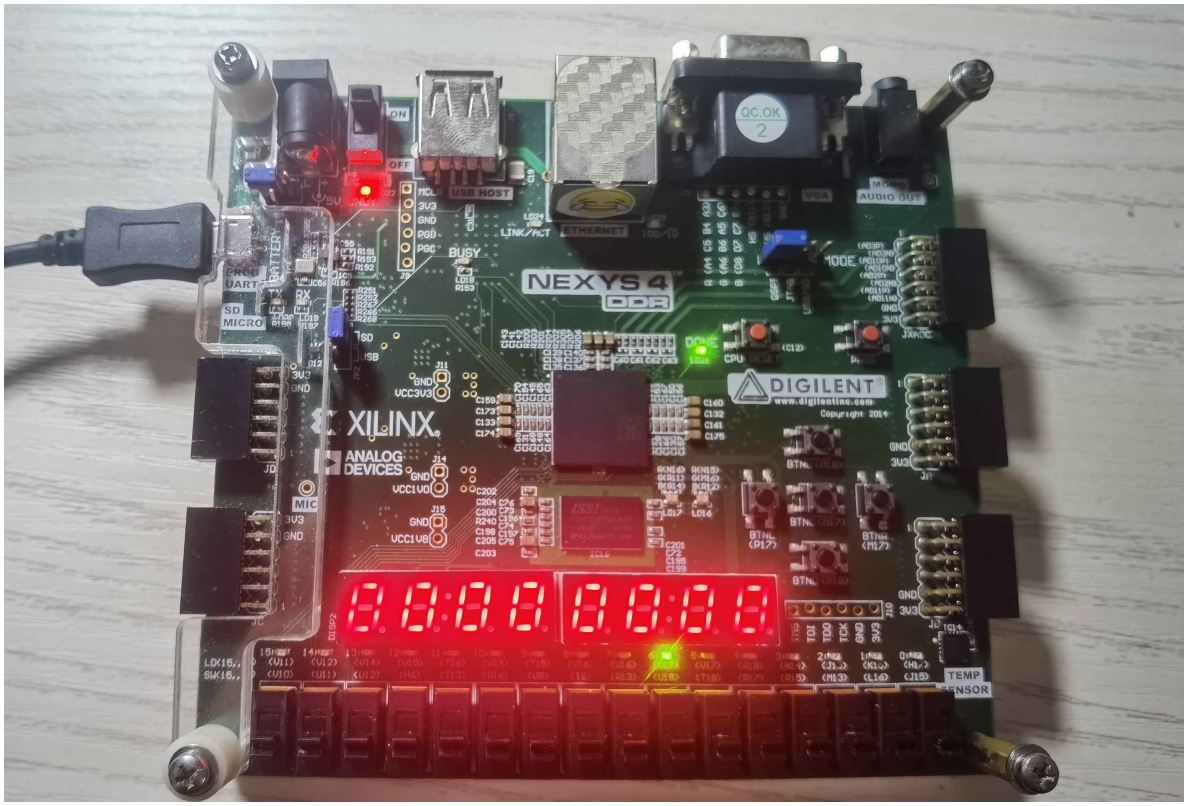
```
1. 相等
2. 无
3. 小于
4. 无
5. 无
```

由此可见，减法输出标志的结果也是完全正确的。

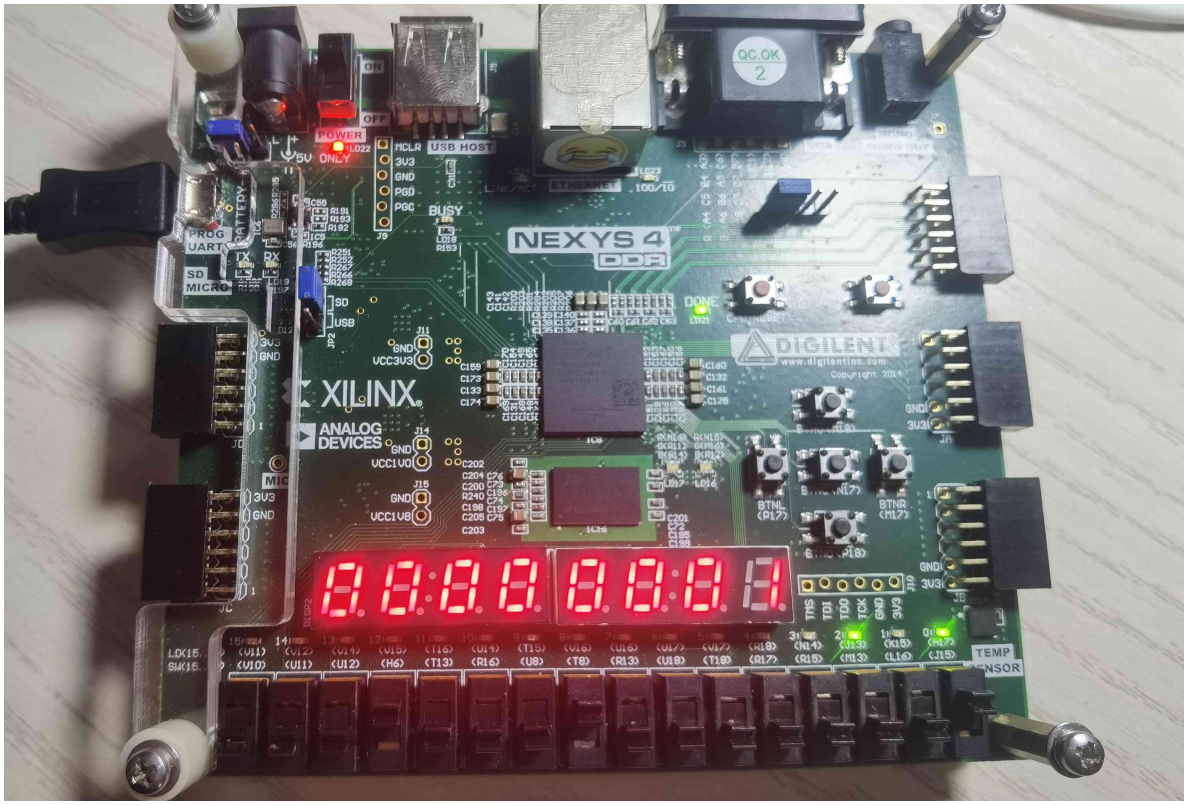
5.2 6-bit ALU 下载测试

5.2.1 实际运行展示

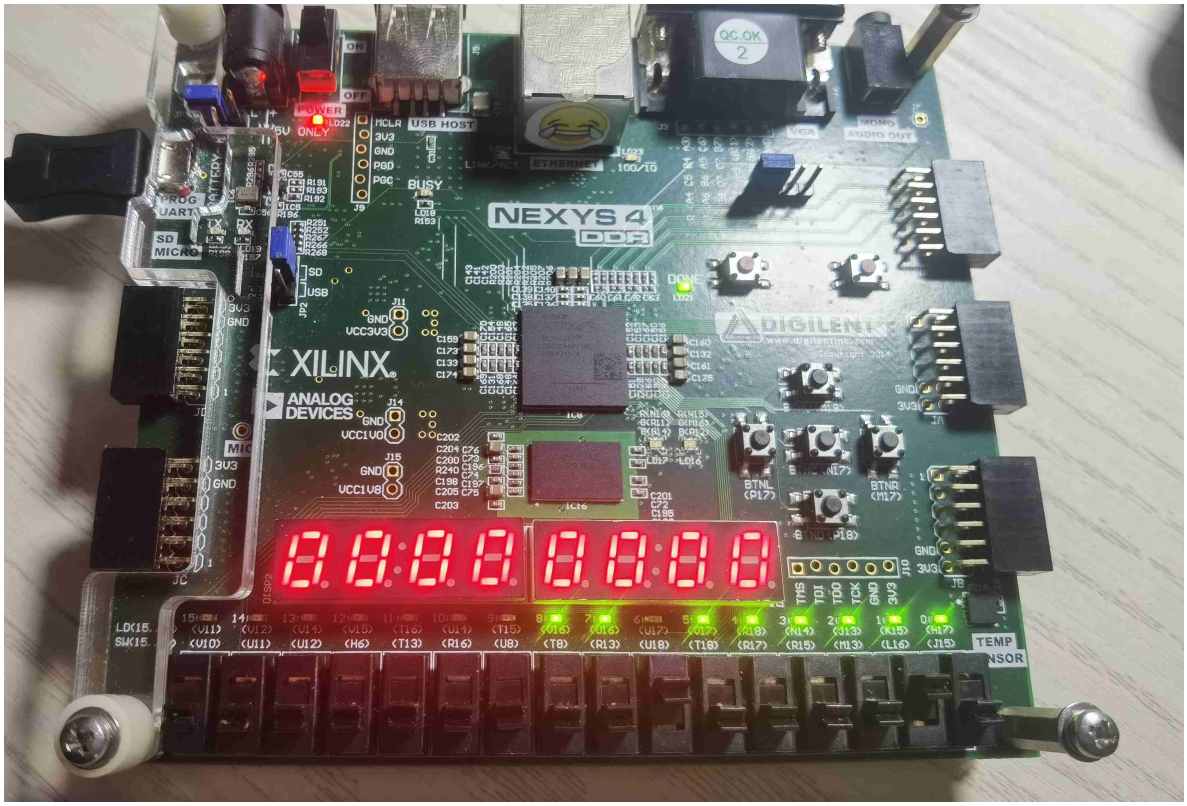
使用 NEXYS4 DDR 开发板，我们将烧好的bit文件下载上去。部分运行结果如下：



初始状态，视作 0 - 0 = 0 。此时只有相等标志为1。



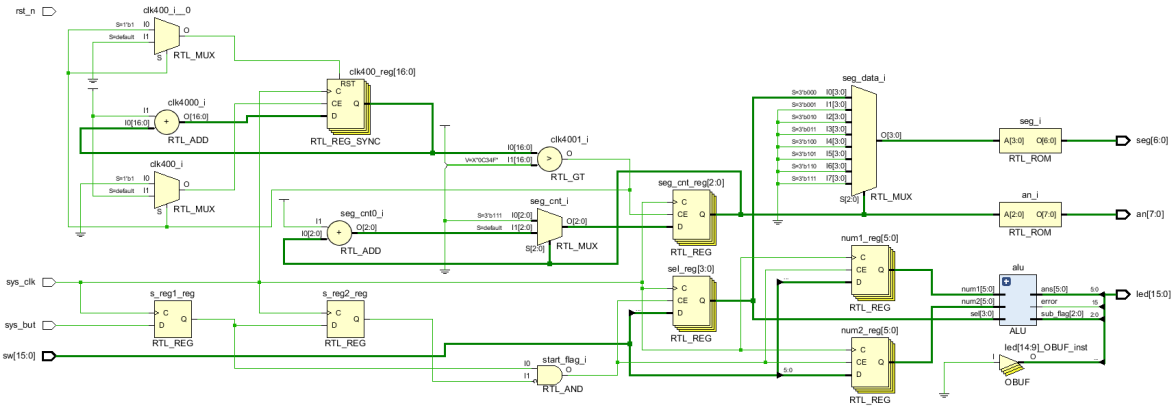
加法示例： 4 + 1 = 5 。此时结果输出 5 ， 所有标志均为0。



> 减法示例：1 - 2 = -1 。此时结果输出 -1 (111111)，小于标志为1。

5.2.2 RTL 电路图

本实验的 RTL 电路图如下：



5.2.3 资源与性能报告

本实验的电路资源使用如下：....

Name	Slice LUTs (63400)	Slice Registers (126800)	Bonded IOB (210)	BUFGCTRL (32)
topmodule	75	62	50	1
alu (ALU)	8	6	0	0
reg1 (Dff)	14	6	0	0
reg2 (Dff_0)	38	6	0	0
reg3 (Dff_1)	1	6	0	0

电路性能如下：

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 11	0.152	0	1	1	num1_reg[0]/C	reg1/q_reg[0]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 12	0.152	0	1	1	num1_reg[1]/C	reg1/q_reg[1]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 13	0.152	0	1	1	num1_reg[2]/C	reg1/q_reg[2]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 14	0.152	0	1	1	num1_reg[3]/C	reg1/q_reg[3]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 15	0.152	0	1	1	num1_reg[4]/C	reg1/q_reg[4]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 16	0.152	0	1	1	num1_reg[5]/C	reg1/q_reg[5]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 17	0.152	0	1	1	num2_reg[0]/C	reg2/q_reg[0]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 18	0.152	0	1	1	num2_reg[1]/C	reg2/q_reg[1]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 19	0.152	0	1	1	num2_reg[2]/C	reg2/q_reg[2]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin
Path 20	0.152	0	1	1	num2_reg[3]/C	reg2/q_reg[3]/D	0.288	0.147	0.141	0.0	sys_clk_pin	sys_clk_pin

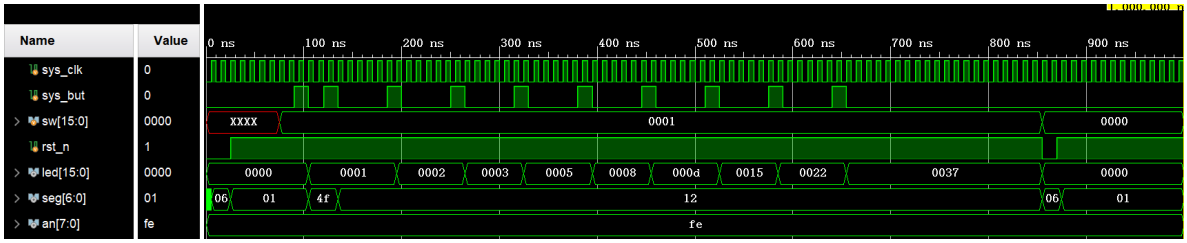
5.3 FLS 仿真

FLS top 模块的仿真文件如下：

```
initial begin

    sys_but <= 'b0;
    sys_clk <= 'b0;
    rst_n <= 'b0;
    #25 rst_n <= 'b1;

    #50 sw <= 16'b0_000_0000_0000_0001;
    #15 sys_but <= 'b1;
    #15 sys_but <= 'b0;


    #15 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
    #50 sys_but <= 'b1;
    #15 sys_but <= 'b0;
```

```
    #200 sw <= 16'b0_000_0000_0000_0000;
    rst_n <= 'b0;
    #15 rst_n <= 'b1;
end
always #5 sys_clk <= ~sys_clk;
```

仿真文件中，我们保持输入一直是 1 。仿真波形如下：



由此可见，按下按钮后，程序的输出结果顺次为1、1、2、3、5、8、13、21……，结果符合斐波那契数列的关系。按下`rst_n`后，所有缓存清零，程序回到初始状态。

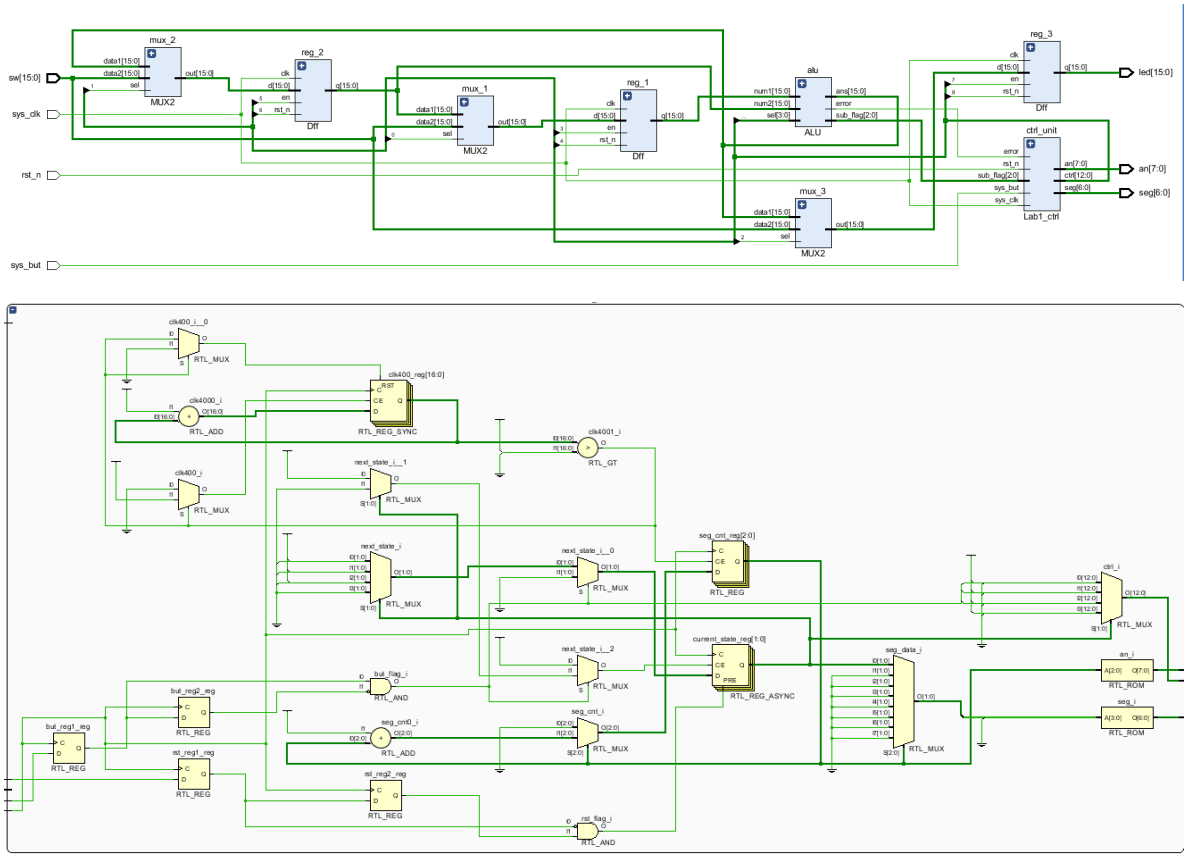
5.4 FLS 下载测试

5.4.1 实际运行展示

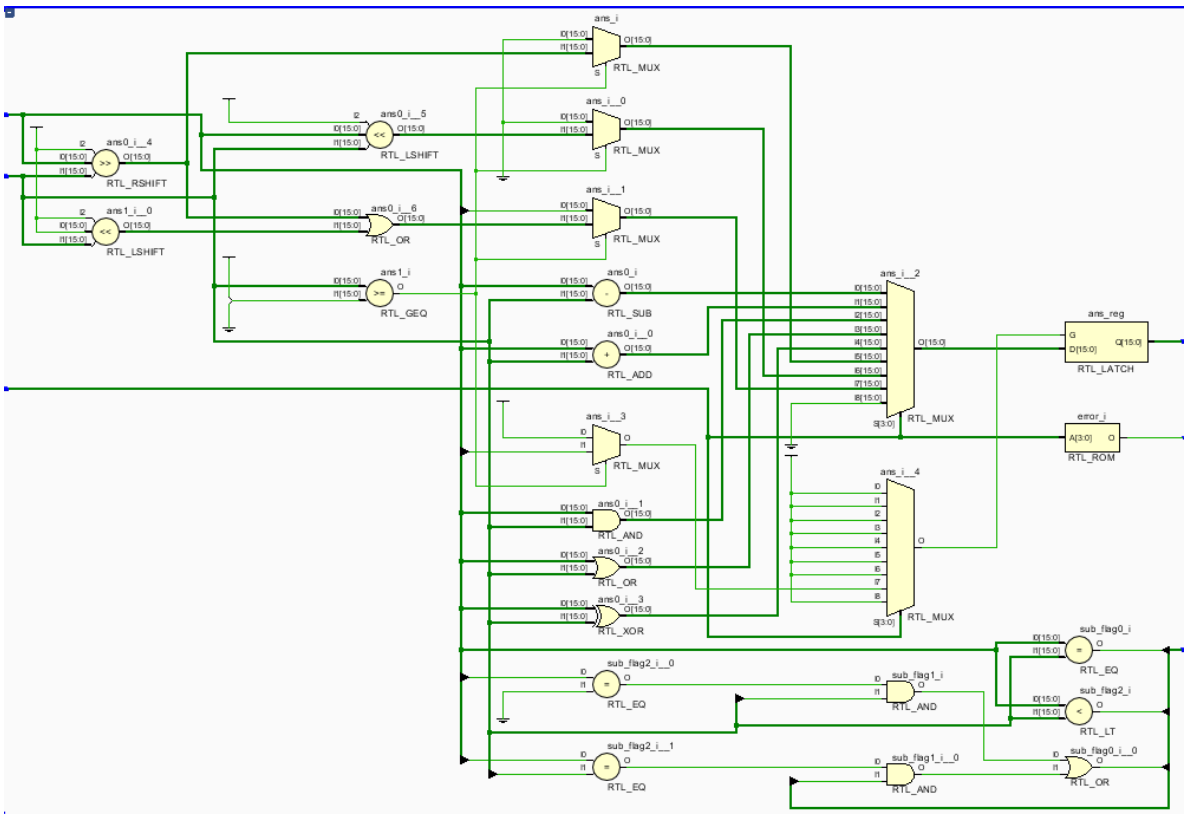
略。见验收细节。

5.4.2 RTL 电路图

本实验的 RTL 电路图如下：



控制器单元



ALU 单元

5.4.3 资源与性能报告

本实验的电路资源使用如下：

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Block RAM Tile (135)	Bonded IPADs (2)	BUFIO (24)
N FLS_top	57	74	25	57	74	50	1
alu (ALU)	0	0	4	0	0	0	0
ctrl_unit (Lab1_ctrl)	32	26	19	32	0	0	0
mux_1 (MUX2)	8	0	3	8	0	0	0
reg_1 (Dff)	0	16	3	0	0	0	0
reg_2 (Dff_0)	17	16	6	17	0	0	0
reg_3 (Dff_1)	0	16	5	0	0	0	0

电路性能如下：

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	5.298	4	4	3	reg_2/q_reg[5]/C	reg_2/q_reg[9]/D	4.576	1.905	2.671	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 2	5.356	5	5	3	reg_2/q_reg[5]/C	reg_3/q_reg[13]/D	4.477	2.019	2.458	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 3	5.361	5	5	3	reg_2/q_reg[5]/C	reg_2/q_reg[12]/D	4.499	1.903	2.596	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 4	5.379	5	5	3	reg_2/q_reg[5]/C	reg_2/q_reg[13]/D	4.481	2.019	2.462	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 5	5.388	5	5	3	reg_2/q_reg[5]/C	reg_2/q_reg[14]/D	4.446	1.923	2.523	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 6	5.417	4	4	3	reg_2/q_reg[5]/C	reg_3/q_reg[11]/D	4.420	1.887	2.533	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 7	5.482	5	5	3	reg_2/q_reg[5]/C	reg_3/q_reg[12]/D	4.355	1.903	2.452	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 8	5.506	4	4	3	reg_2/q_reg[5]/C	reg_2/q_reg[11]/D	4.334	1.887	2.447	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 9	5.527	4	4	3	reg_2/q_reg[5]/C	reg_3/q_reg[9]/D	4.353	1.905	2.448	10.0	sys_clk_pin	sys_clk_pin		0.035
Path 10	5.555	5	5	3	reg_2/q_reg[5]/C	reg_3/q_reg[14]/D	4.302	1.923	2.379	10.0	sys_clk_pin	sys_clk_pin		0.035

六、心得体会

本实验作为学期的开端之作，在保持了一定难度的同时让我充分复习了一次 Verilog 硬件描述方式。这也是我第一次按照数据通路的方法对电路进行模块化设计，过程中感觉条理清晰，思路明确。希望后续实验能够和本次实验一样流程合理且难度适中！