暨南大学本科实验报告专用纸

课程名称 计算机组成原理 成绩评定_____

实验项目名称 ALU 指导教师_____

实验项目编号 0806006402 实验项目类型 综合 实验地点_____

学生姓名 甄洛生 学号 2018054625

学院 信息科学技术 系 计算机科学 专业计算机科学与技术

实验时间 2020 年 10 月 13 日 下午～10 月 13 日 午 温度__℃湿度

一、 实验目的

- 掌握 ALU 模块的组成和接口，理解 ALU 的功能。
- 通过编程调用 ALU 模块计算斐波那契数。
- 掌握 Verilog 中多模块编程方法和实现。

二、 实验内容

用 Verilog 设计一个算术运算单元 ALU，采用纯组合逻辑设计，32bit 宽。

1. 利用该 ALU 完成斐波那契数 f(n)，其 中 2<n<16。（80 分）
2. 利用该 ALU 完成连加运算 f(n)=1+2+……+n，其中 2<n<16。（90 分）
3. 用 4 个七段数码管显示 16 位输出。（+10 分）(2<n<32)

三、 实验程序

Top.v

```
1.  `timescale 1ns / 1ps
2.
3.  module top(
4.      input clk,
5.      input rst,
6.      input [4:0] n,          // 2 < n < 32
7.      input Q,                // Q = 0 为 Fibo; Q = 1 为累加求和
8.      //output [15:0] result,   // 4 个七段数码管暗示需要 16bits
9.      output [6:0] a2g,       // 用于显示第 an 个数码管的 a2g 值
10.     output [3:0] an
11.     );
12.     // 保存模块运行结果
13.     // 注意：Q 用于选择显示哪个问题的结果
14.     // 实际上 top 会将两个问题都进行求解
15.     wire[31:0] fibo;
16.     wire[31:0] sum;
```

```
    wire[15:0] result;

    // 实例化 fibo 控制器模块
    fibo my_fib(
        .clk(clk),
        .rst(rst),
        .n(n),
        .result(fibo)
    );
    // 实例化 cumsum 控制器模块
    cumsum my_cumsum(
        .clk(clk),
        .rst(rst),
        .n(n),
        .result(sum)
    );

    // 根据 Q 保存结果至 result
    assign result = Q ? sum[15:0] : fibo[15:0];

    // 降低时钟频率至合适，否则烧坏数码管
    wire clk_digit;
    divclk my_divclk(
        .clk(clk),
        .new_clk(clk_digit)
    );
    // 实例化七段数码管控制器模块，显示 result（16 进制）
    digit my_digit(
        .data(result),
        .clk(clk_digit),
        .a2g(a2g),
        .an(an)
    );
endmodule
```

Fibo. v

```
`timescale 1ns / 1ps

module fibo(
    input clk,
    input rst,
```

```
    input [4:0] n,
    output [31:0] result
    );
    // 利用 ALU 实现 Fibonacci，即 Fibo 是 ALu 的控制器

    // 变量声明
    reg[31:0] regA, regB;
    wire[31:0] ans;
    reg[4:0] count;
    // 实例化 ALU 模块
    alu myalu(.a(regA),.b(regB),.op(4'b0001),.f(ans),.c());
    // 每个 clk 上升沿改变寄存器 A/B
    // 导致 ans 立即发生改变
    always @(posedge clk)
    begin
        // 从 fibonacci(3)开始计算
        if(rst == 1'b1)
        begin
            regA <= 32'b1;
            regB <= 32'b1;
            count <= 5'b00011;
        end
        // 开始迭代计算
        else
        begin
            if(count < n)
            begin
                regA <= regB;
                regB <= ans;
                count <= count + 1'b1;
            end
        end
    end

    // 每当 ans 变化，result 则会更新
    assign result = ans;
endmodule
```

Alu.v

```
`timescale 1ns / 1ps
module alu(
```

```
    input [31:0] a,
    input [31:0] b,
    input [3:0] op,
    output reg [31:0] f,
    output c
    );

    always @(*)
    begin
        case(op)
            4'b0000: f = 32'b0;
            4'b0001: f = a + b;
            4'b0010: f = a - b;
            4'b0011: f = a & b;
            4'b0100: f = a | b;
            4'b0101: f = a ^ b;
            default: f = 32'b0;
        endcase
    end

    assign c = ~(|f);
endmodule
```

Cumsum.v

```
`timescale 1ns / 1ps

module cumsum(
    input clk,
    input rst,
    input [4:0] n,
    output [31:0] result
    );
    // 利用 ALU 实现累加求和，即 cumsum 是 ALu 的控制器

    // 变量声明
    reg[31:0] regA, regB;   // 输入端寄存器
    wire[31:0] ans;         // 保存 Alu 运行结果

    // 实例化 ALU 模块
    alu myalu(
        .a(regA),
```

```
        .b(regB),
        .op(4'b0001),
        .f(ans),
        .c()
    );
    // 每个 clk 上升沿改变寄存器 A/B
    // 导致 ans 立即发生改变
    always @(posedge clk)
    begin
        // 从 1 + 2 开始算起
        if(rst == 1'b1)
        begin
            regA <= 32'b1;
            regB <= 32'b10;
        end
        // 现在不断递增 regB，直到 regB == n
        else
        begin
            if(regB < n)
            begin
                regA <= ans;
                regB <= regB + 1'b1;
            end
        end
    end

    // 每当 ans 变化，result 则会更新
    assign result = ans;
endmodule
```

Divclk.v

```
`timescale 1ns / 1ps

module divclk(
    input clk,
```

```
    output new_clk
    );
```

```
    reg[24:0] data=25'b0;
    always @(posedge clk)
        data<=data+1'b1;
    assign new_clk = data[18];
endmodule
```

Digit.v

```
`timescale 1ns / 1ps

module digit(
    input [15:0] data,
    input clk,
    output reg[6:0] a2g,
    output reg[3:0] an
    );
    reg[1:0] status = 2'b00;
    reg[3:0] digit;

    // 调度数码管
    always @(posedge clk)
        status<=status+1'b1;

    // 根据不同状态来确定显示高 4 位还是低 4 位
    always @(*)
    case(status)
        2'b00:begin digit = data[15:12]; an=4'b1000;end
        2'b01:begin digit = data[11:8];  an=4'b0100;end
        2'b10:begin digit = data[7:4];   an=4'b0010;end
        2'b11:begin digit = data[3:0];   an=4'b0001;end

        default:begin digit=data[3:0]; an=4'b0001;end
    endcase

    //根据数字 digit 来设置不同的 a~g 段
    always @(*)
    case(digit)
            4'h0:a2g=7'b1111110;
            4'h1:a2g=7'b0110000;
            4'h2:a2g=7'b1101101;
            4'h3:a2g=7'b1111001;
            4'h4:a2g=7'b0110011;
```

```
            4'h5:a2g=7'b1011011;
            4'h6:a2g=7'b1011111;
            4'h7:a2g=7'b1110000;
            4'h8:a2g=7'b1111111;
            4'h9:a2g=7'b1111011;
            4'hA:a2g=7'b1110111;
            4'hB:a2g=7'b0011111;
            4'hC:a2g=7'b1001110;
            4'hD:a2g=7'b0111101;
            4'hE:a2g=7'b1001111;
            4'hF:a2g=7'b1000111;
            default:a2g=7'b1111110;
    endcase
endmodule
```

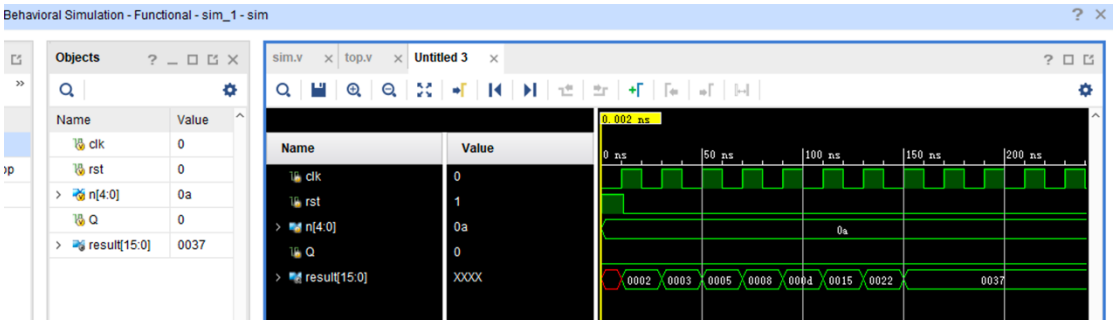
四、 仿真程序

Sim.v

```
`timescale 1ns / 1ps

module sim(

    );
    // 时钟信号仿真
    reg clk = 1'b0;
    always #10
        clk = ~clk;

    reg rst = 1'b1;
    reg[4:0] n = 5'b01010;
    reg Q = 1'b0;
    wire[15:0] result;

    top mytop(clk,rst,n,Q,result);

    initial
          #11 rst = 1'b0;
endmodule
```

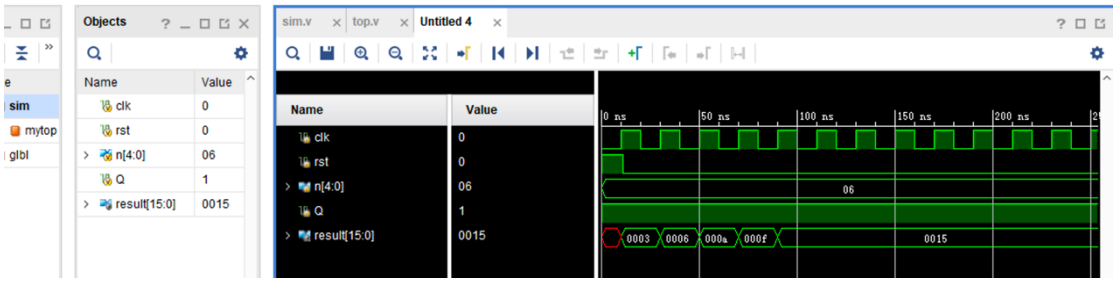
五、 仿真结果

斐波那契 f(10)仿真：



结果 = 0x37 = 55，正确。

累加求和 g(6)仿真：



结果 = 0x15 = 21，正确。

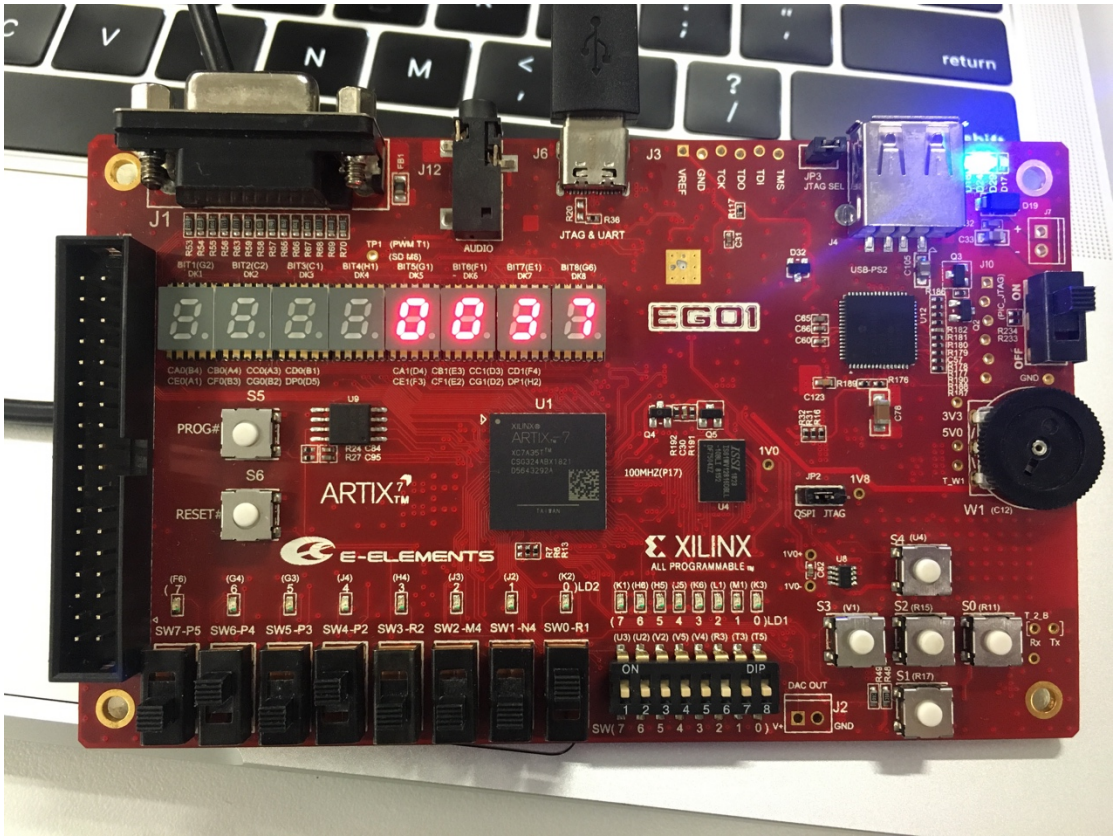
六、 实验结果

变量 n 从高位至低位的各个引脚为：P5/P4/P3/P2/R2。（即左边 5 个开关）

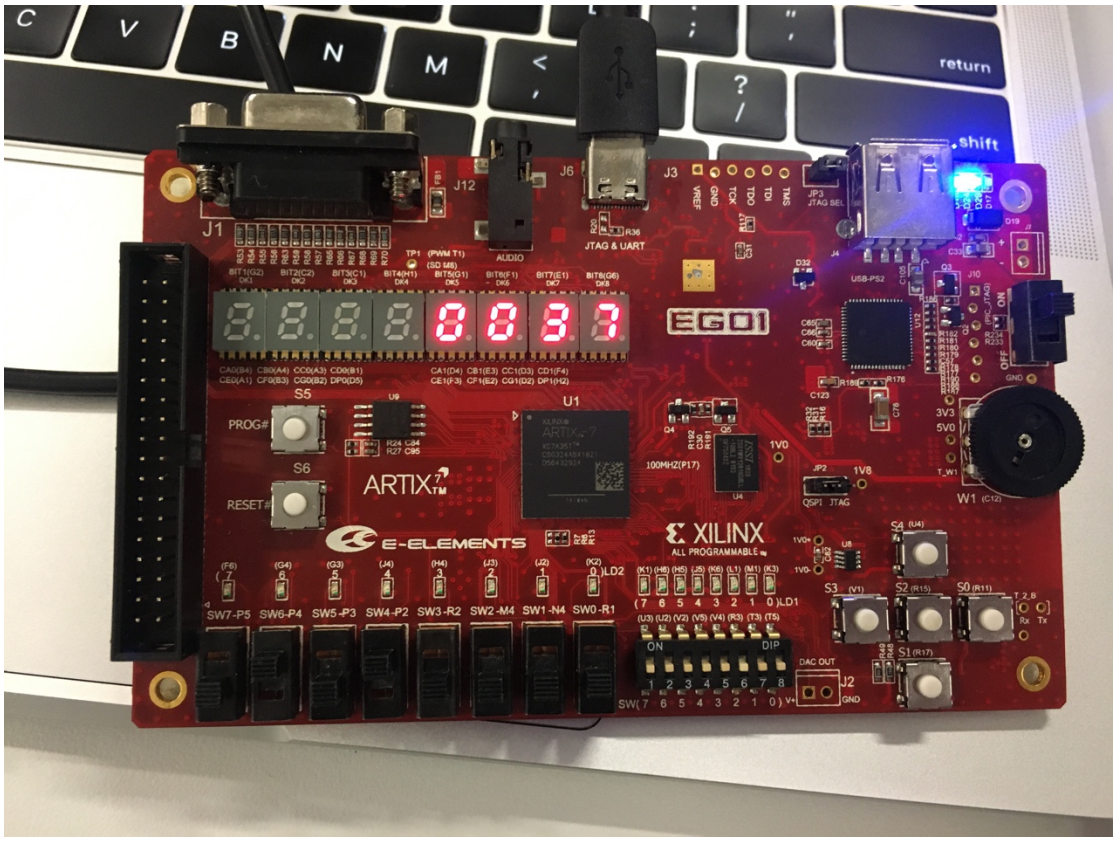
函数选择 Q 为 R1 引脚。（右边数第一个开关）

刷新 rst 引脚为 N4 引脚。（右边数第二个开关）

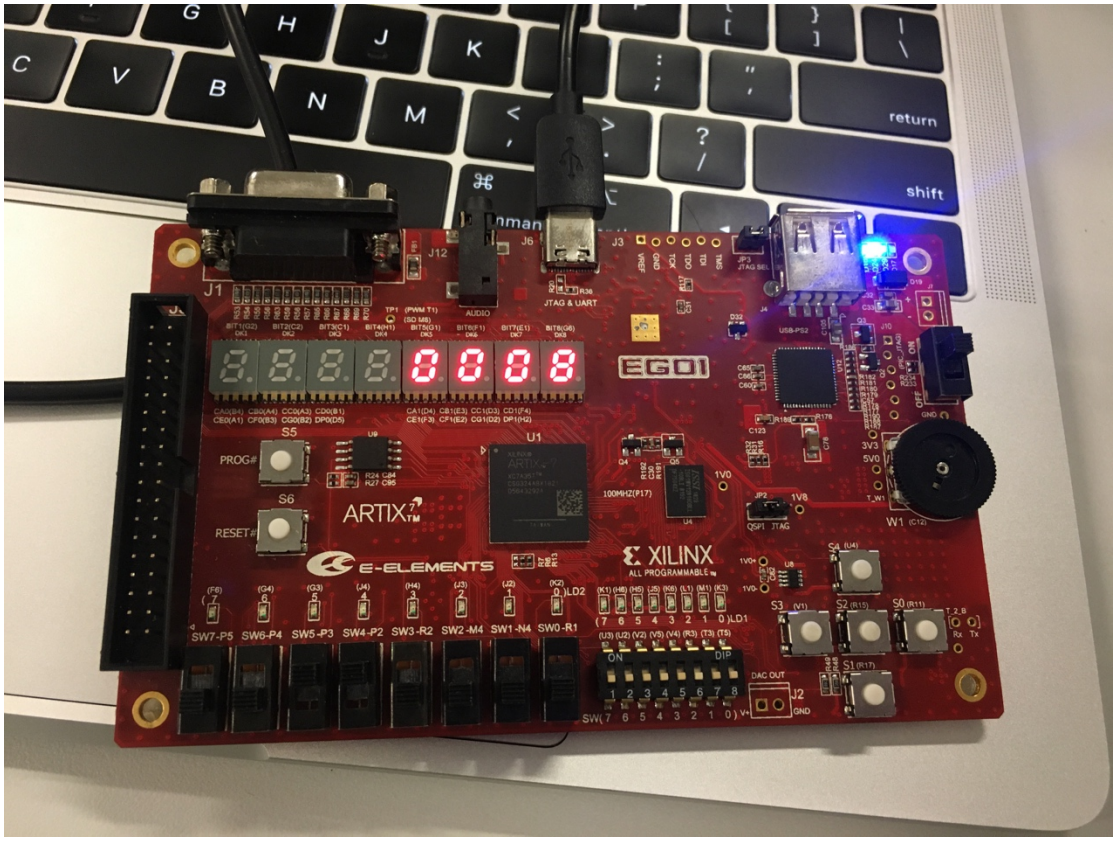
累加求和 g(10) = 0x37 = 55,结果正确



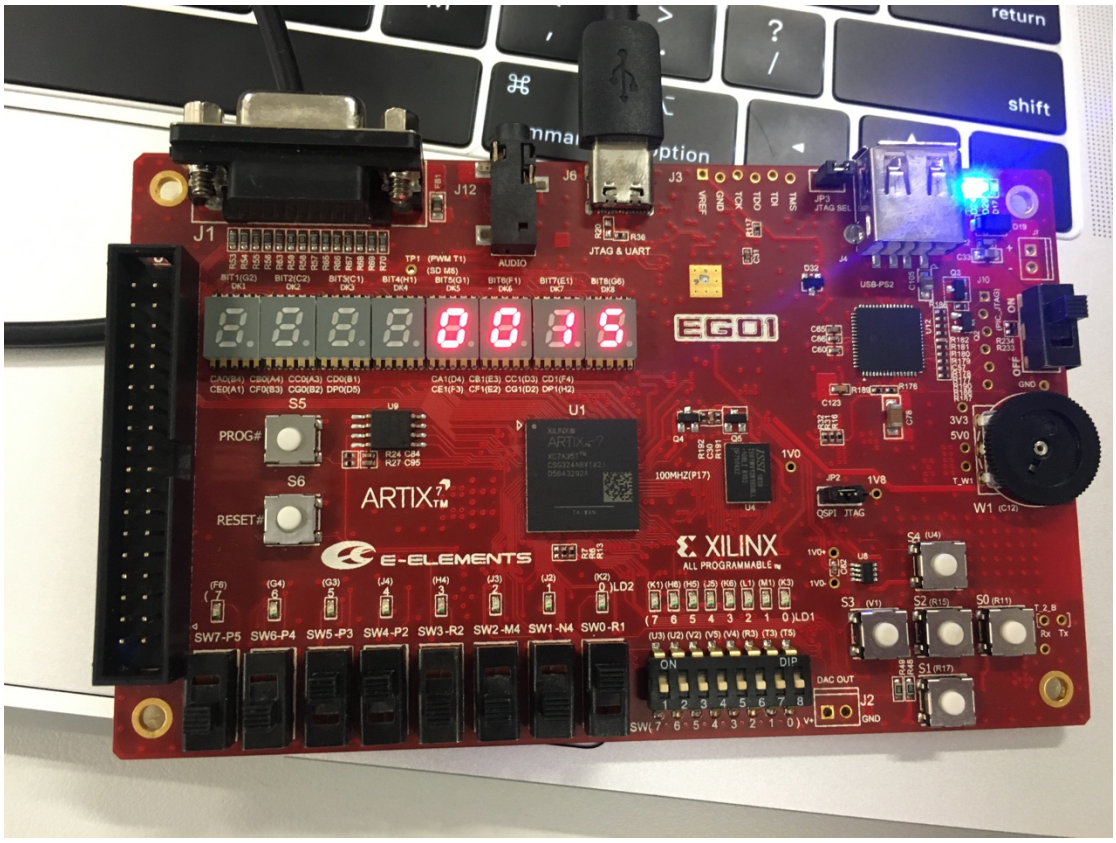
斐波那契 f(10) = 0x37 = 55,结果正确。



斐波那契 f(6) = 0x8 =8,结果正确。



累加求和 g(6) = 0x15 = 21，结果正确。



七、 实验体会

老师，上课时可以多讲一下电路代码的模块思想，或者说电路模块设计套路，我觉得这个很重要，一个好的电路应该具备好的模块设计，好的模块设计可以使得代码更容易理解，而且便于调试。