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 REG 寄存器实验 指导教师 王传胜

实验项目编号 0806006403 实验项目类型 综合 实验地点 N126

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 27 日 下午～10 月 27 日下午 温度 ℃湿度

一、 实验目的

- 掌握寄存器堆的工作原理和接口
- 掌握寄存器堆的实现方法
- 掌握寄存器堆在微处理器中承担的功能

二、 实验内容

1. 设计一 32*32bit 的寄存器文件，即 32 个 32 位的寄存器文件(寄存器组)
- 具备两组读端口及一组写端口;
- 通过读端口可从 0~31 号的任意地址读取数据;
- 通过写端口可向 1~31 号的任意地址写入 数据(0 号寄存器的值固定为 32’b0);
2. 利用该 ALU 完成斐波那契数 f(n)，其 中 2<n<16。(80 分)
3. 利用该 ALU 完成连加运算 f(n)=1+2+.....+n，其中 2<n<16。(90 分)
4. 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：将所有模块封装成顶层模块

```
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] test,   // 仿真检验结果用，实际用数码管显示结果
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
```

```
    // 实际上 top 会将两个问题都进行求解
    wire[31:0] fibo;
    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];
    // 仿真用，写比特流请注释掉以及上面的 test 端口
    //assign test = 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
```

RegFiles.v：寄存器堆模块

```
`timescale 1ns / 1ps
```

```
2.
3.  module RegFiles(
4.      input clk,
5.      input we,
6.      input [4:0] raddr1,
7.      output [31:0] rdata1,
8.      input [4:0] raddr2,
9.      output [31:0] rdata2,
10.     input [4:0] waddr,
11.     input [31:0] wdata
12.     );
13.     // 此寄存器堆共有 32 个 32 位寄存器
14.     // 可同时读两个寄存器的值
15.     // 支持写操作，we 控制读/写
16.
17.     // 此处只定义 31 个寄存器，原因是 0 号寄存器恒为 0
18.     // 可直接硬编码，缩减成本。
19.     reg[31:0] regs[1:31];
20.
21.     // 定义读操作电路
22.     assign rdata1 = (raddr1 == 5'b00000) ? 32'b0 : regs[raddr1];
23.     assign rdata2 = (raddr2 == 5'b00000) ? 32'b0 : regs[raddr2];
24.
25.     // 定义写操作电路，下降沿开始写
26.     always @(negedge clk)
27.         if(we)
28.             if(waddr != 5'b00000)
29.                 regs[waddr] <= wdata;
30.
31. endmodule
```

ALU.v：运算器模块

```
1.  `timescale 1ns / 1ps
2.
3.  module ALU(
4.      input [31:0] a,
5.      input [31:0] b,
6.      input [3:0] op,
7.      output reg [31:0] f,
8.      output c
9.      );
```

```
10.
11.     always @(*)
12.     begin
13.         case(op)
14.             4'b0000: f = 32'b0;
15.             4'b0001: f = a + b;
16.             4'b0010: f = a - b;
17.             4'b0011: f = a & b;
18.             4'b0100: f = a | b;
19.             4'b0101: f = a ^ b;
20.             default: f = 32'b0;
21.         endcase
22.     end
23.
24.     assign c = ~(|f);
25. endmodule
```

fibo.v：斐波那契控制器模块

```
1.  `timescale 1ns / 1ps
2.
3.  module fibo(
4.      input clk,
5.      input rst,
6.      input [4:0] n,
7.      output [31:0] result
8.      );
9.      // 利用寄存器堆和 ALU 实现 Fibonacci
10.     // 即 Fibo 是 RegFiles/ALu 的控制器
11.     // 控制器采用有限状态机(4 status)实现
12.
13.     // 变量声明
14.     reg [4:0] read_addr[1:2];       // 读端口 2 个地址寄存器
15.     wire [31:0] data[1:2], ans;     // ALU 的三端
16.     reg we;                         // 写使能端
17.     reg [4:0] write_addr;           // 写端口地址寄存器
18.     reg [31:0] write_data;          // 写端口数据寄存器
19.     reg [1:0] status;               // 状态表示
20.
21.     reg [4:0] count;                // 用于何时停止计算
22.     // 实例化 RegFiles 模块
23.     RegFiles reg_inst1(
```

```
        .clk(clk),
        .we(we),
        .raddr1(read_addr[1]),
        .rdata1(data[1]),
        .raddr2(read_addr[2]),
        .rdata2(data[2]),
        .waddr(write_addr),
        .wdata(write_data)
    );
    // 实例化 ALU 模块
    ALU alu_inst1(
        .a(data[1]),
        .b(data[2]),
        .f(ans),
        .op(4'b0001)    // 0x1 使 ALU 进行加运算
    );
    // 有 限 状 态 机
    // S0 和 S1 用于初始化
    // S2 和 S3 用于计算并且更新寄存器堆相应参数
    // 上升沿切换状态（注意，控制器不要和寄存器堆写操作同时使用
    // 上升沿，否则出现竞争，使得状态切换后，参数没能来得及赋值
    // 完成，而寄存器已经开始写，就会得到错误结果！
    always @(posedge clk)
    begin
        // 初始化状态，用于重新计算/刷新
        if(rst == 1'b1)
        begin
            count <= 5'b00010;
            status <= 2'b00;
        end
        // 进入有限状态机
        else
        begin
            case(status)
                // 状态 0：regs[1] <= 1
                2'b00:
                    begin
                        we <= 1'b1;
                        write_addr <= 5'b00001;
                        write_data <= 32'b1;
                        status <= 2'b01;
                    end
```

```
66.                 // 状态 1：regs[2] <= 1 且 初始化 read_addr
67.                 2'b01:
68.                     begin
69.                         we <= 1'b1;
70.                         write_addr <= 5'b00010;
71.                         write_data <= 32'b1;
72.                         read_addr[1] <= 5'b00001;
73.                         read_addr[2] <= 5'b00010;
74.                         status <= 2'b10;
75.                     end
76.                 // 状态 2：ALU 计算 并 写答案至下一个寄存器
77.                 2'b10:
78.                     begin
79.                         we <= 1'b1;
80.                         write_addr <= read_addr[2] + 1;
81.                         write_data <= ans;
82.                         count <= count + 1;
83.                         status <= 2'b11;
84.                     end
85.                 // 状态 3：平移选择的寄存器并返回状态 2
86.                 // 或计算完成后，啥也不干了。
87.                 2'b11:
88.                     begin
89.                         if(count < n)
90.                         begin
91.                             read_addr[1] <= read_addr[1] + 1;
92.                             read_addr[2] <= read_addr[2] + 1;
93.                             we = 1'b0;
94.                             status <= 2'b10;
95.                         end
96.                     end
97.                 default: status <= 2'b00;
98.             endcase
99.         end
100.     end
101.     // 众所周知，ALU 最后的 ans 即为答案
102.     assign result = ans;
103.     //initial
104.         //$monitor($time,,"status=%d reg1=%x reg2=%x",status,data[1],data[2
    ]);
105. endmodule
```

cumsum.v：累加求和控制器模块

```
1.  `timescale 1ns / 1ps
2.
3.  module cumsum(
4.      input clk,
5.      input rst,
6.      input [4:0] n,
7.      output [31:0] result
8.      );
9.      // 利用寄存器堆和 ALU 实现 累加求和
10.     // 即 cumsum 是 RegFiles/ALu 的控制器
11.     // 控制器采用有限状态机实现
12.
13.     // 变量声明
14.     reg [4:0] read_addr[1:2];       // 读端口 2 个地址寄存器
15.     wire [31:0] data[1:2], ans;     // ALU 的三端
16.     reg we;                         // 写使能端
17.     reg [4:0] write_addr;           // 写端口地址寄存器
18.     reg [31:0] write_data;          // 写端口数据寄存器
19.     reg [1:0] status;               // 状态表示
20.
21.     reg [4:0] count;                // 用于何时停止计算
22.     // 实例化 RegFiles 模块
23.     RegFiles reg_inst1(
24.         .clk(clk),
25.         .we(we),
26.         .raddr1(read_addr[1]),
27.         .rdata1(data[1]),
28.         .raddr2(read_addr[2]),
29.         .rdata2(data[2]),
30.         .waddr(write_addr),
31.         .wdata(write_data)
32.     );
33.     // 实例化 ALU 模块
34.     ALU alu_inst1(
35.         .a(data[1]),
36.         .b(data[2]),
37.         .f(ans),
38.         .op(4'b0001)    // 0x1 使 ALU 进行加运算
39.     );
40.     // 有 限 状 态 机
```

```
41.     // S0/S1 分别完成 regs[1]/regs[2]写 1/2 操作，并使 data[1]/data[2]
42.     // 为他俩。S2 将 ALU 计算结果写至 regs[1]。S3 递增 regs[2]。
43.     // 跳转流程：S0 -> S1 -> S2 <--> S3
44.     // 上升沿切换状态（注意，控制器不要和寄存器堆写操作同时使用
45.     // 上升沿，否则出现竞争，使得状态切换后，参数没能来得及赋值
46.     // 完成，而寄存器已经开始写，就会得到错误结果！
47.     always @(posedge clk)
48.     begin
49.         // 初始化状态，用于重新计算/刷新
50.         if(rst == 1'b1)
51.         begin
52.             count <= 5'b00010;
53.             status <= 2'b00;
54.         end
55.         // 进入有限状态机
56.         else
57.         begin
58.             case(status)
59.                 // 状态 0：regs[1] <= 1
60.                 2'b00:
61.                     begin
62.                         we <= 1'b1;
63.                         write_addr <= 5'b00001;
64.                         write_data <= 32'b1;
65.                         status <= 2'b01;
66.                     end
67.                 // 状态 1：regs[2] <= 2 且 初始化 read_addr
68.                 2'b01:
69.                     begin
70.                         we <= 1'b1;
71.                         write_addr <= 5'b00010;
72.                         write_data <= 32'b10;
73.                         read_addr[1] <= 5'b00001;
74.                         read_addr[2] <= 5'b00010;
75.                         status <= 2'b10;
76.                     end
77.                 // 状态 2：ALU 计算 并 写答案至 read_addr[1]寄存器
78.                 2'b10:
79.                     begin
80.                         we <= 1'b1;
81.                         write_addr <= read_addr[1];
82.                         write_data <= ans;
```

```
                        count <= count + 1;
                        status <= 2'b11;
                    end
                // 状态 3：递增 read_addr[2]寄存器的值
                2'b11:
                    begin
                        if(count <= n) begin
                            we <= 1'b1;
                            write_addr <= read_addr[2];
                            write_data <= data[2] + 1;
                            status <= 2'b10;
                        end
                    end
                default: status <= 2'b00;
            endcase
        end
    end
     // 众所周知，ALU 最后的 read_addr[1]的寄存器即为答案
     assign result = data[1];
     //initial
         //$monitor($time,,"status=%d reg1=%x reg2=%x",status,data[1],data[2
]);
endmodule
```

digit.v：数码管显示模块

```
`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
```

divclk.v: 分频器模块

```
`timescale 1ns / 1ps

module divclk(
    input clk,
    output new_clk
    );
    reg[24:0] data=25'b0;
```

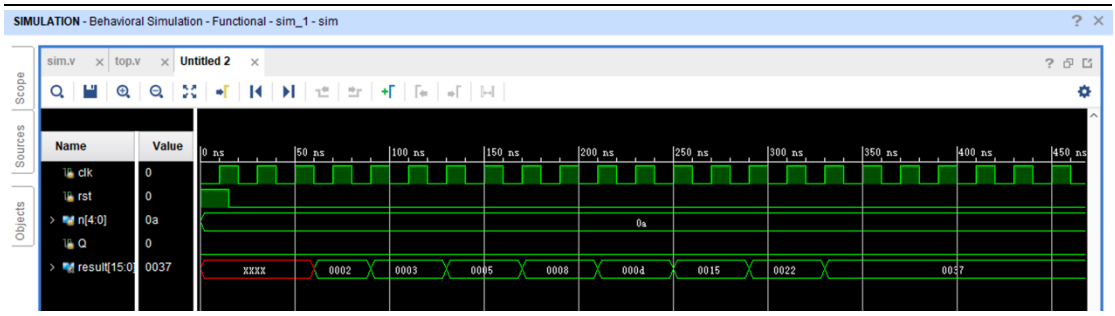
```
    always @(posedge clk)
        data<=data+1'b1;
    // 100MHz 分频至 3Hz
    assign new_clk = data[18];
endmodule
```

四、 仿真程序

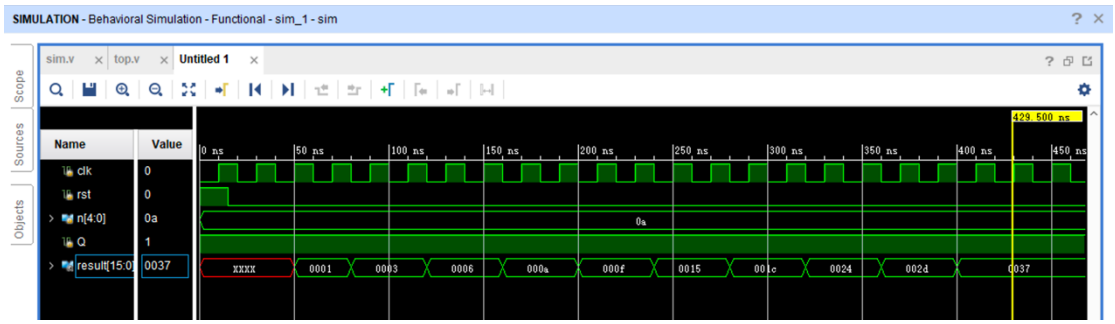
```
`timescale 1ns / 1ps

module sim(

    );
    reg clk = 1'b0;
    reg rst = 1'b1;
    // 生成时钟脉冲，周期为 20ns
    always
        #10 clk = ~clk;
    // 初始化 reset
    initial
        #15 rst = 1'b0;

    reg[4:0] n = 5'b01010;      // 待计算问题的 n
    reg Q = 1'b0;               // Fibo 为 0，CumSum 为 1
    wire[15:0] result;          // 计算结果

    // 注意：此处要将 top 中的 test 端口去掉注释！
    // 原因是实际写到板子上不需要这结果，但不分配给他输出
    // 又无法生成比特流。
    top mytop(clk,rst,n,Q,result);


endmodule
```

五、 仿真结果

斐波那契，n 为 10 时，结果 0x37，正确。

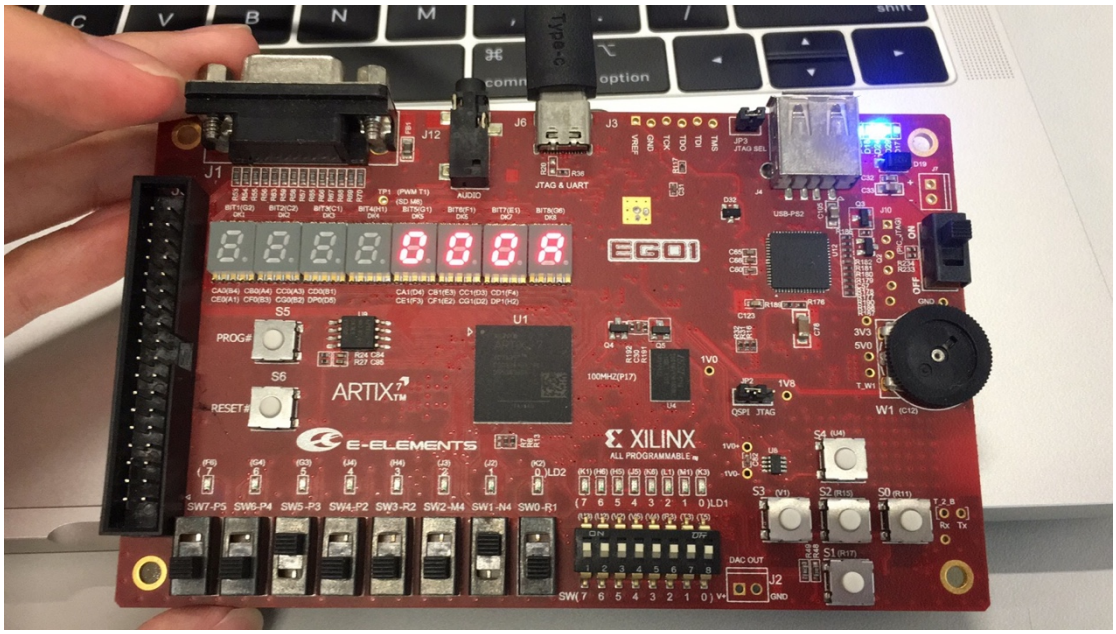


累加求和，n 为 10 时，结果 0x37，正确。

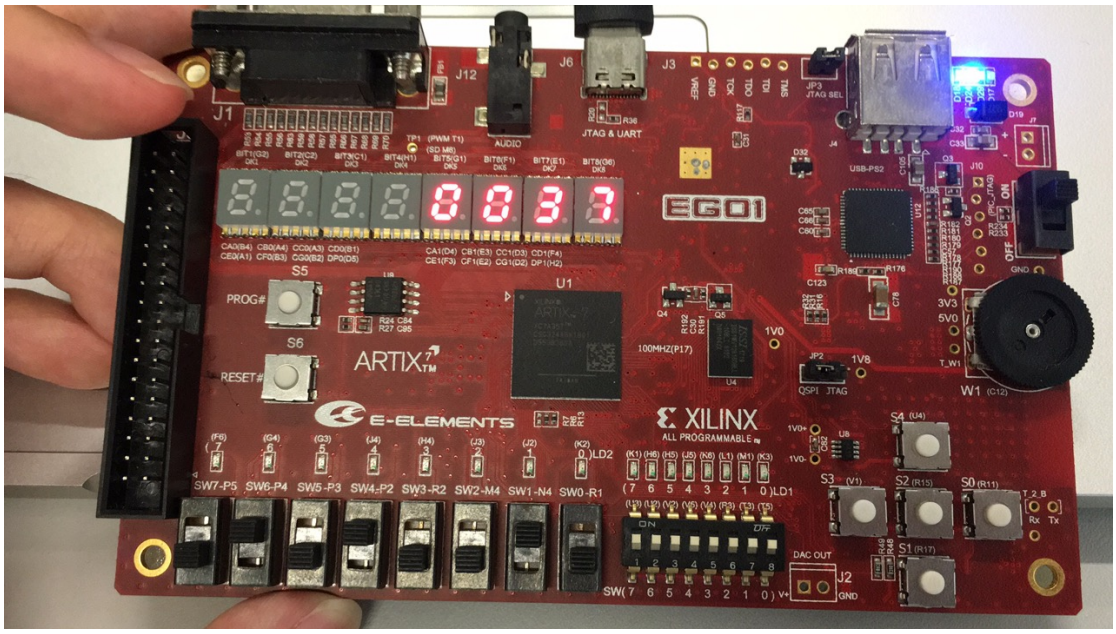


六、 实验结果

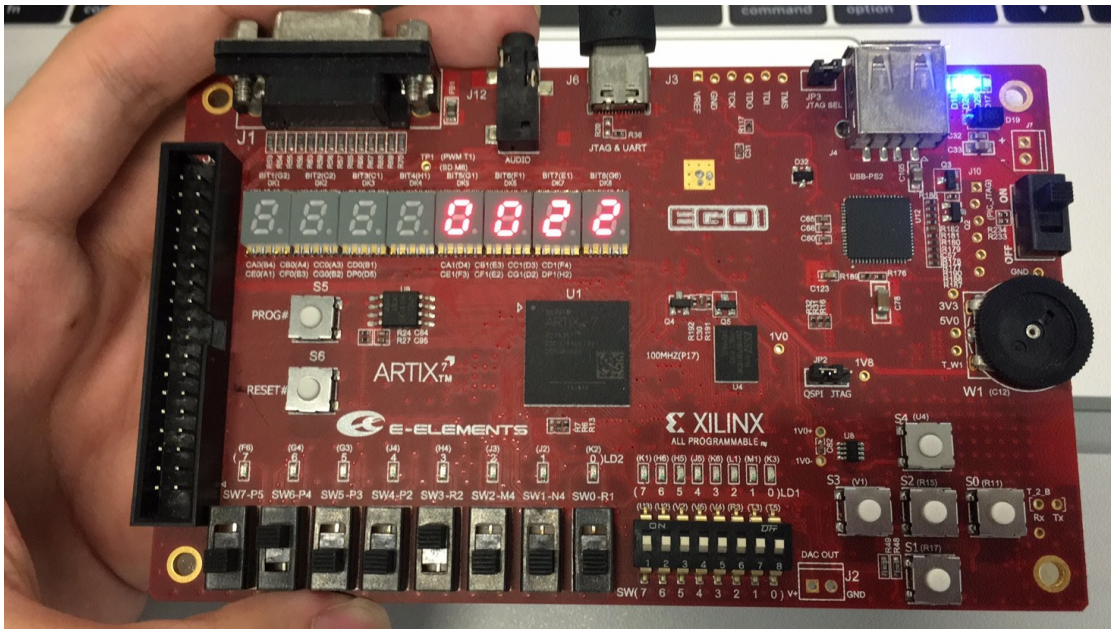
累加求和，n 为 4 时，结果 0xA，正确。



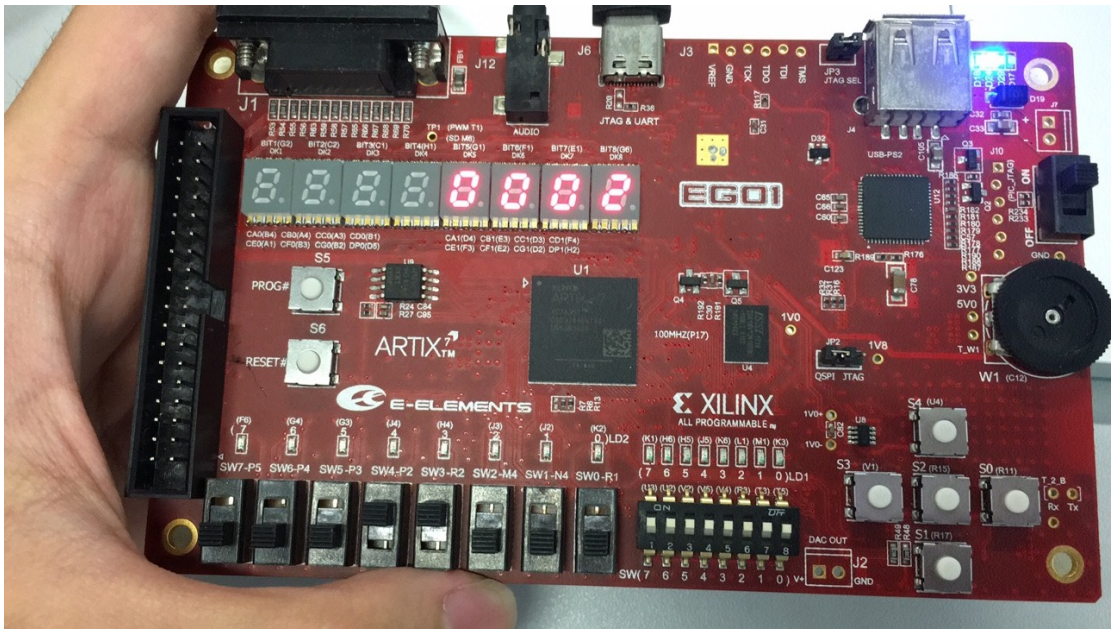
累加求和，n 为 10 时，结果 0x37，正确。



斐波那契，n 为 9 时，结果 0x22，正确。



斐波那契，n 为 3 时，结果 0x2，正确。



七、 实验体会

这次实验就是比上次多控制了寄存器堆，即控制部件（斐波那契/累加求和）内部不再使用自己的寄存器去保存结果，转而使用的是寄存器堆中的寄存器，那么控制器里就必须包含必要的控制部件，如 we 使能寄存器，地址寄存器。由于每一次下降沿只能写一个寄存器，所以就必须用到有限状态机，不像上一个实验，直接改写两个寄存器。所以本次实验难点就在于有限状态机的状态转换，以及结果应由什么指示。

老师讲的挺好的，没毛病。