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 RAM 存储器实验 指导教师 王传胜

实验项目编号 0806006404 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 11 月 17 日 下午～11 月 17 日 午 温度 ℃湿度

一、 实验目的

- 掌握存储器的工作原理和接口
- 掌握存储器的实现方法和初始化方法
- 掌握 RISC-V 中存储器的存取方式

二、 实验内容

- **实验一**

用 Verilog 实现 PC 和 ROM，ROM 保存数据。如下:

0x34020002 0x8c030020 0x34040001 0x34050401 0x00853020 0x00052026

1. 要求在 EGO-1 板卡中读出对应数据的后 12bit(90 分)
2. 用 4 个七段数码管读出对应数据的后 16bit (+10 分)

注意:用通用按钮 s2 来产生脉冲信号

- **实验二**

将数据 0x123487ab 按下列方式保存到 ram 的对应单元,并将数据的[31:28][15:12][7:4]显示出来。如果做数码管则显示: [31:28][23:20][15:12][7:4]

code	指令	地址
0000	lb	0
0001	lh	0
0010	lw	0
0011	lbu	0
0100	lhu	0
0101	sb	4
0110	sh	8
0111	sw	0
1000	lw	4
1001	lw	8

三、 实验程序

本实验的 ROM 模块使用 IP Core 的 Distributed Memory Generator v8.0 实现。

init.coe: ROM 系数文件

```
memory_initialization_radix = 16;
memory_initialization_vector =
34020002 8c030020 34040001 34050401
00853020 00052026;
```

top.v: 顶层模块

```
`timescale 1ns / 1ps

module top(
    input chip_clk,     // P17 时钟脉冲 100MHz
    input clk,          // S2 手动脉冲
    input rst,          // PC 复位端
    input [3:0] code,   // RAM_cntr 代码端
    input Q,            // 用于选择显示 ROM(=0)或 RAM(=1)的数据
    //output [31:0] test, // 仿真测试输出端口(综合前请注释)
    output [6:0] a2g,   // 数码管 a2g 端
    output [3:0] an     // 选择数码管端
    );
    // rom 使用 IP core(Distributed LUT)
    // pc 应使用 clk
    // 七段数码管/RAM 芯片 使用 chip_clk

    wire [31:0] pc_addr;
    // 实例化 PC
    pc my_pc(
        .rst(rst),
        .clk(clk),
        .addr(pc_addr)
    );

    wire [31:0] rom_data;
    // 实例化 ROM
    rom my_rom(
        .a(pc_addr[5:0]),
        .spo(rom_data)
    );
```

```
31.
32.     wire [31:0] rdata;  // 保存读出数据
33.     // 实例化 RAM controller
34.     ram_cntr ins_ram_cntr(
35.         .clk(chip_clk),
36.         .code(code),
37.         .wdata(32'h123487ab),   // 直接写 0x123487ab(实验要求)
38.         .rdata(rdata)
39.     );
40.
41.     // 实例化分频器
42.     wire clk_3hz;
43.     divclk my_divclk(
44.         .clk(chip_clk),
45.         .new_clk(clk_3hz)
46.     );
47.
48.     wire [15:0] disp_data;
49.     // Q = 0 显示 ROM 输出
50.     // Q = 1 显示 RAM 输出
51.     assign disp_data = Q ?
52.                         {
53.                             rdata[31:28],
54.                             rdata[23:20],
55.                             rdata[15:12],
56.                             rdata[7:4]
57.                         } :
58.                         rom_data[15:0];
59.     // 实例化数码管
60.     digit my_digit(
61.         .data(disp_data),
62.         .clk(clk_3hz),
63.         .a2g(a2g),
64.         .an(an)
65.     );
66.
67.     // 仅用于仿真查看 ROM 和 RAM 的数据
68.     // 综合前必须注释掉 test 端口和下面这行
69.     //assign test = {{rdata[31:28], rdata[23:20], rdata[15:12], rdata[7:4]},
    rom_data[15:0]};
70. endmodule
```

pc.v：程序计数器模块

```
1.  `timescale 1ns / 1ps
2.
3.  module pc(
4.      input rst,
5.      input clk,
6.      output reg [31:0] addr
7.      );
8.      always @(posedge clk)
9.      begin
10.         if(rst == 1'b0)
11.             addr <= 32'b0;
12.         else
13.             addr <= addr + 1;
14.     end
15. endmodule
```

ram.v：RAM 存储器模块

```
1.  `timescale 1ns / 1ps
2.
3.  module ram(
4.      input clk,
5.      input we,
6.      input [5:0] addr,
7.      output [31:0] rdata,
8.      input [31:0] wdata
9.      );
10.     // 用寄存器堆模拟 RAM
11.     reg [31:0] data[0:63];
12.     // read
13.     assign rdata = data[addr];
14.     // write
15.     always @(posedge clk) begin
16.         if(we) begin
17.             data[addr] <= wdata;
18.         end
19.     end
20. endmodule
```

ram_cntr.v：RAM 控制器模块

```
1.  `timescale 1ns / 1ps
2.
3.  module ram_cntr(
4.      input clk,
5.      input [3:0] code,
6.      input [31:0] wdata,
7.      output reg [31:0] rdata
8.      );
9.      // 地址(由相应指令信号确定)
10.     wire [5:0] addr;
11.     // 读数据的中间形态(未处理)
12.     wire [31:0] im_rdata;   // intermedian read_data
13.     // 处理后的写数据
14.     wire [31:0] fn_wdata;   // final write_data
15.     // 指令信号 高电平有效
16.     wire lb0, lh0, lw0, lbu0, lhu0, sb4, sh8, sw0, lw4, lw8;
17.
18.     // 译码 code,生成指令信号
19.     assign lb0 = ~(|code);                                  // 0000
20.     assign lh0 = ~(|code[3:1]) & code[0];                   // 0001
21.     assign lw0 = ~(|code[3:2]) & code[1] & ~code[0];        // 0010
22.     assign lbu0 = ~(|code[3:2]) & (&code[1:0]);             // 0011
23.     assign lhu0 = ~code[3] & code[2] & ~(|code[1:0]);       // 0100
24.     assign sb4 = ~code[3] & code[2] & ~code[1] & code[0];   // 0101
25.     assign sh8 = ~code[3] & (&code[2:1]) & ~code[0];        // 0110
26.     assign sw0 = ~code[3] & (&code[2:0]);                   // 0111
27.     assign lw4 = code[3] & ~(|code[2:0]);                   // 1000
28.     assign lw8 = code[3] & ~(|code[2:1]) & code[0];         // 1001
29.
30.     // 根据指令生成地址字段
31.     assign addr = (sh8 | lw8) ?
32.                     6'b10 : ((sb4 | lw4) ?
33.                             6'b01 : 6'b0);
34.
35.     // 根据指令信号加工写数据
36.     assign fn_wdata = (
37.                         {24'b0, {8{sb4}}}  |
38.                         {16'b0, {16{sh8}}} |
39.                         {32{sw0}}
40.                       ) &
41.                       wdata;
42.
```

```
43.     // 根据指令类型生成 we 信号
44.     wire we;
45.     assign we = (sb4 | sh8 | sw0) ? 1'b1 : 1'b0;
46.
47.     // RAM 实例
48.     ram ins_ram(
49.         .clk(clk),
50.         .we(we),
51.         .addr(addr),
52.         .rdata(im_rdata),
53.         .wdata(fn_wdata)
54.     );
55.     initial
56.         $monitor("code = %x\taddr = %x\twdata = %x", code, addr,
    wdata);
57.     // 根据指令信号加工读出数据
58.     always @(*) begin
59.         if(lb0)
60.             rdata <= {{24{im_rdata[7]}}, im_rdata[7:0]};
61.         else if(lh0)
62.             rdata <= {{16{im_rdata[15]}}, im_rdata[15:0]};
63.         else if(lw0 | lw4 | lw8)
64.             rdata <= im_rdata;
65.         else if(lbu0)
66.             rdata <= {24'b0, im_rdata[7:0]};
67.         else if(lhu0)
68.             rdata <= {16'b0, im_rdata[15:0]};
69.         else
70.             rdata <= 32'b0;
71.     end
72. endmodule
```

digit.v：数码管显示模块

```
1.  `timescale 1ns / 1ps
2.
3.  module digit(
4.      input [15:0] data,
5.      input clk,
6.      output reg[6:0] a2g,
7.      output reg[3:0] an
8.      );
```

```
    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：分频模块

```
`timescale 1ns / 1ps

module divclk(
    input clk,
    output new_clk
    );
    reg[18:0] data=19'b0;
    always @(posedge clk)
        data<=data+1'b1;
    // 100MHz 分频至 3Hz
    assign new_clk = data[18];
endmodule
```

四、 仿真程序

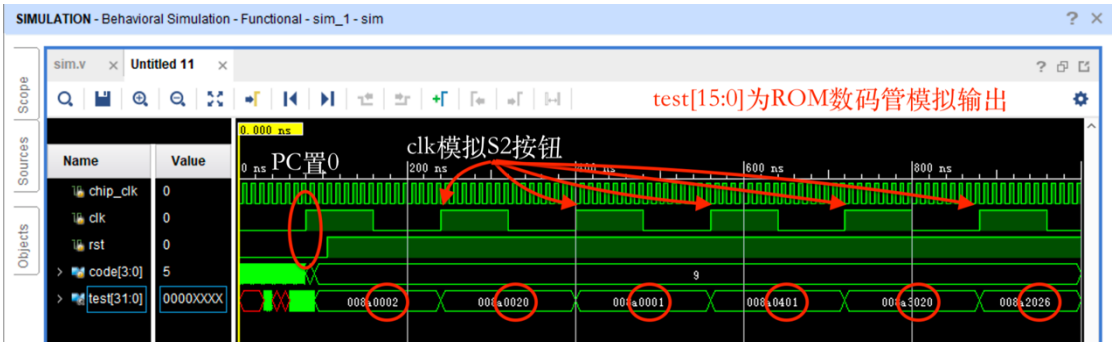
sim.v: 仿真脚本

```
`timescale 1ns / 1ps

module sim(

    );
    reg chip_clk = 1'b0;
    reg clk = 1'b0;
    reg rst = 1'b0;
    reg [3:0] code;
    // 模拟 clk T = 160ns PC 可以加 6 次 1
    always
          #80 clk <= ~clk;
    // 保证 set PC addr 0
    initial
          #105 rst <= ~rst;

    // 模拟芯片时钟脉冲
    always
          #5 chip_clk = ~chip_clk;

    // 模拟代码执行，先 s 再 l

    initial begin
        code = 4'b0101;     // sb4
          #10 code = 4'b0110; // sh8
```

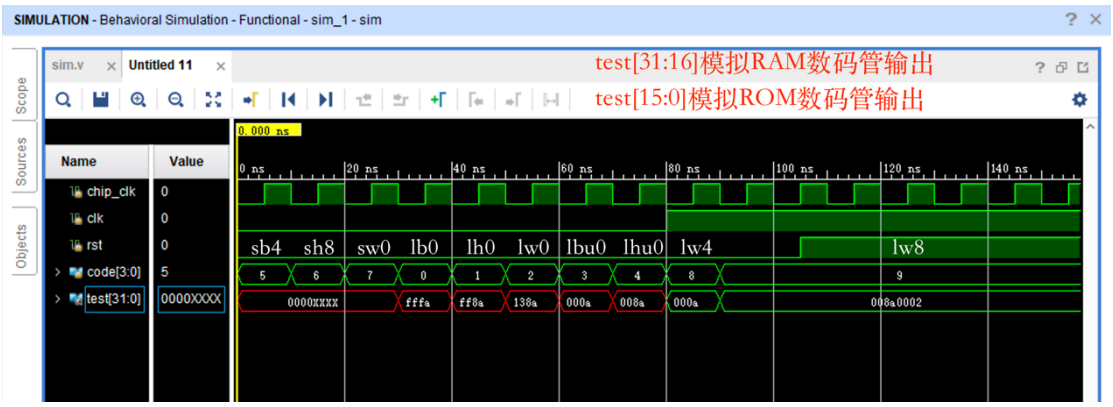
```
            #10 code = 4'b0111; // sw0
            #10 code = 4'b0000; // lb0
            #10 code = 4'b0001; // lh0
            #10 code = 4'b0010; // lw0
            #10 code = 4'b0011; // lbu0
            #10 code = 4'b0100; // lhu0
            #10 code = 4'b1000; // lw4
            #10 code = 4'b1001; // lw8
    end

    wire [31:0] test;
    top my_top(
        .chip_clk(chip_clk),
        .clk(clk),
        .rst(rst),
        .code(code),
        .test(test)
    );
endmodule
```

五、 仿真结果

ROM 的模拟结果：



RAM 的模拟结果：



六、 团队分工

没有团队，一个人完成所有工作。

七、 实验结果

Q = 1 时（N4 打开），数码管显示 RAM 的内容，执行 sb/sw/sh 后：

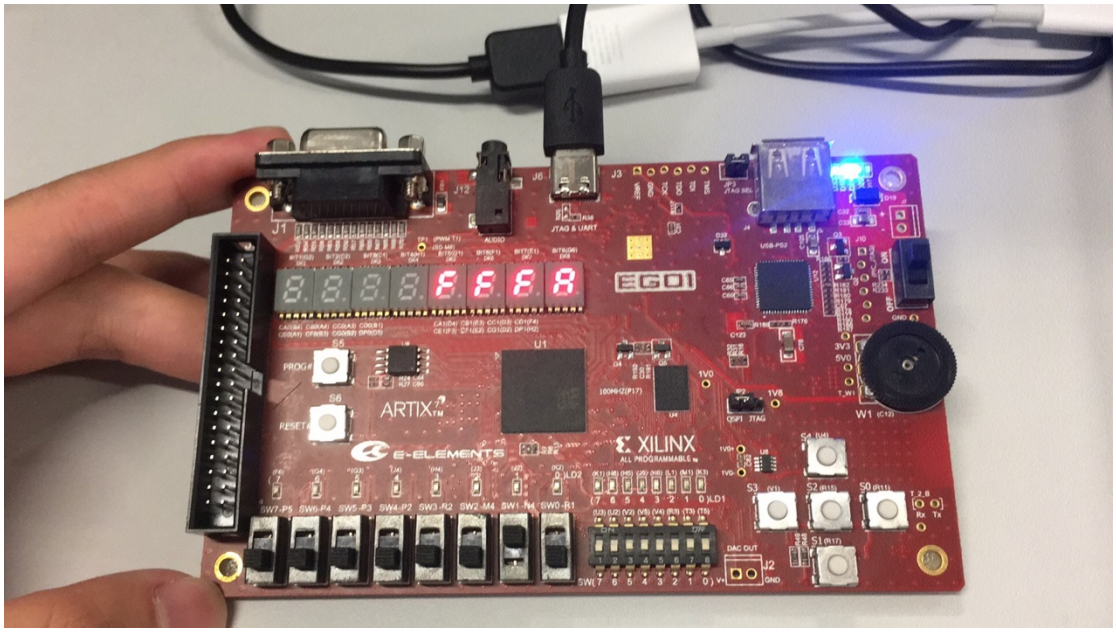


图 1 code = 0000

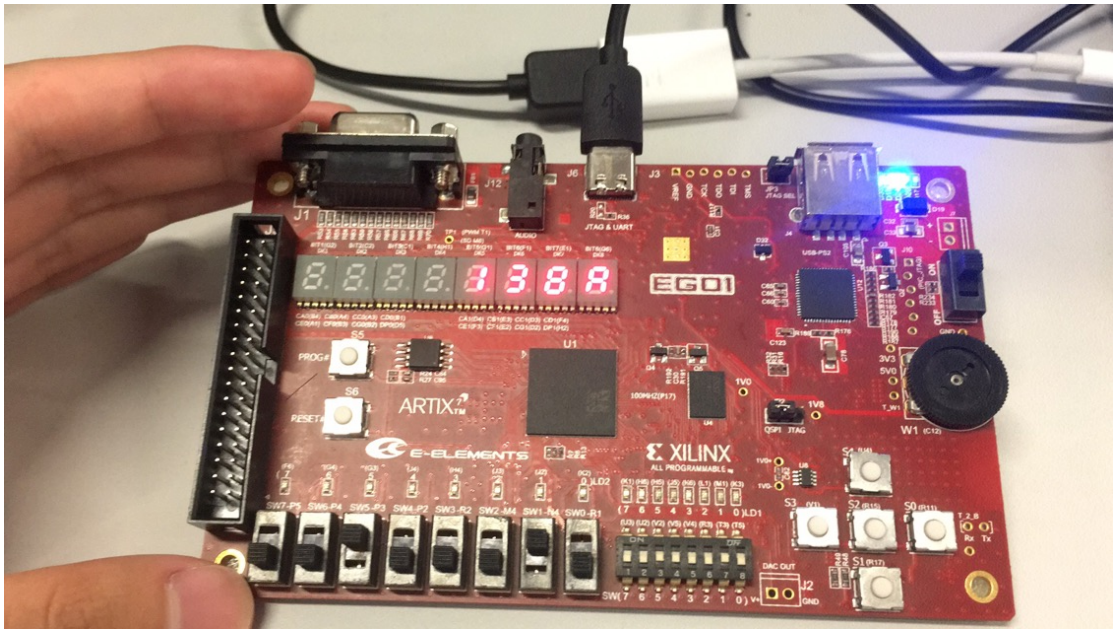


图 2 code = 0010

Q = 0 时（N4 关闭），数码管显示 ROM 的内容：

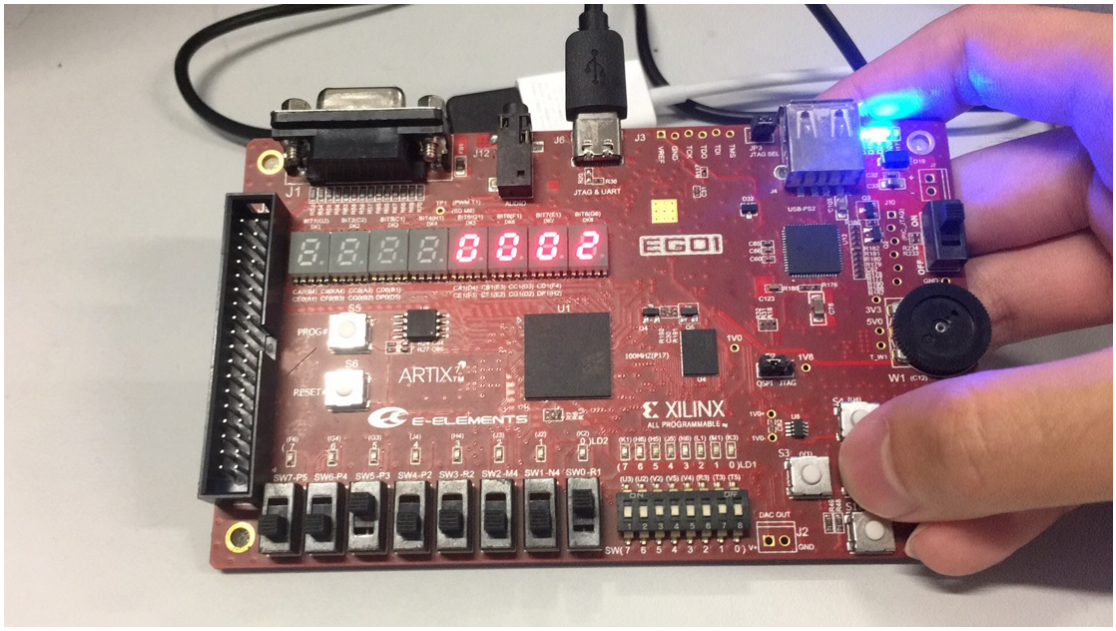


图 3 rst（R1）关闭按下 S2 按钮

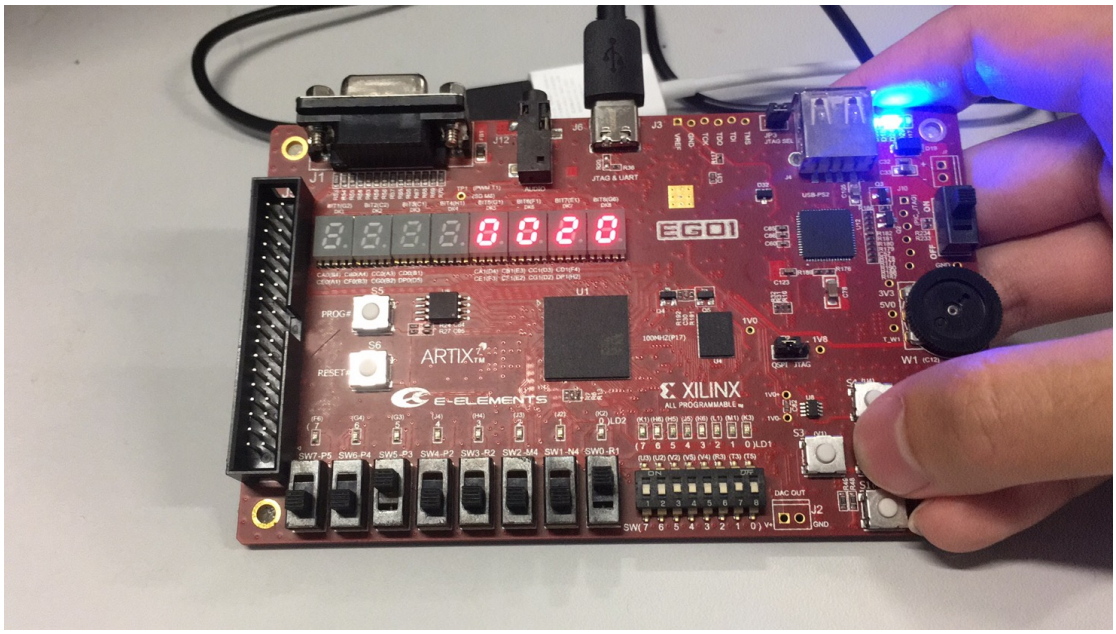


图 4 再次按下 S2 按钮

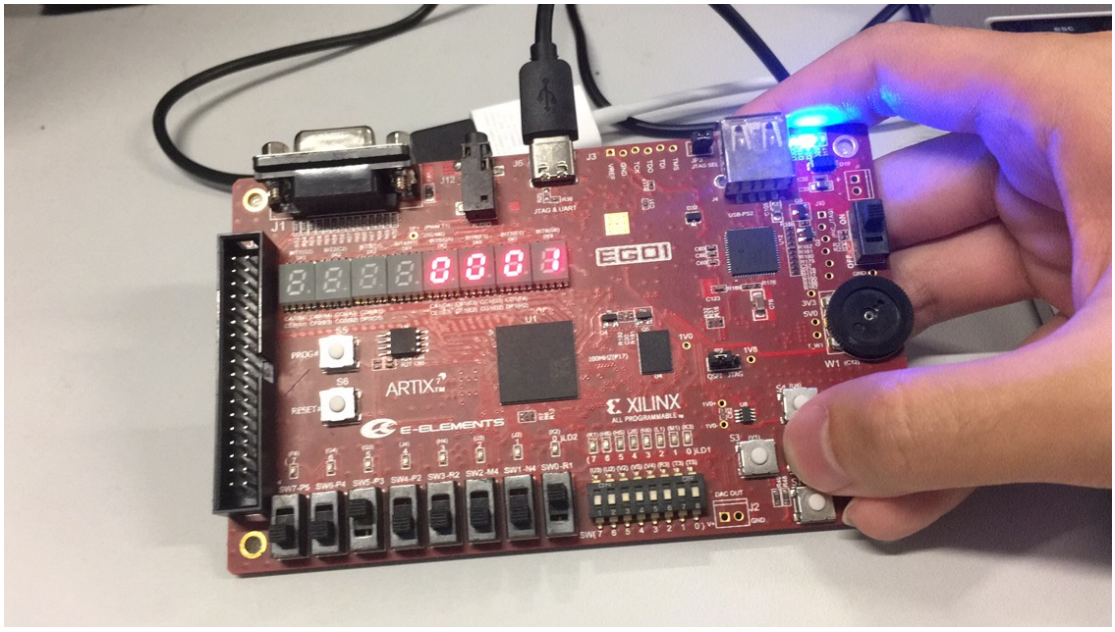


图 4 再次按下 S2 按钮

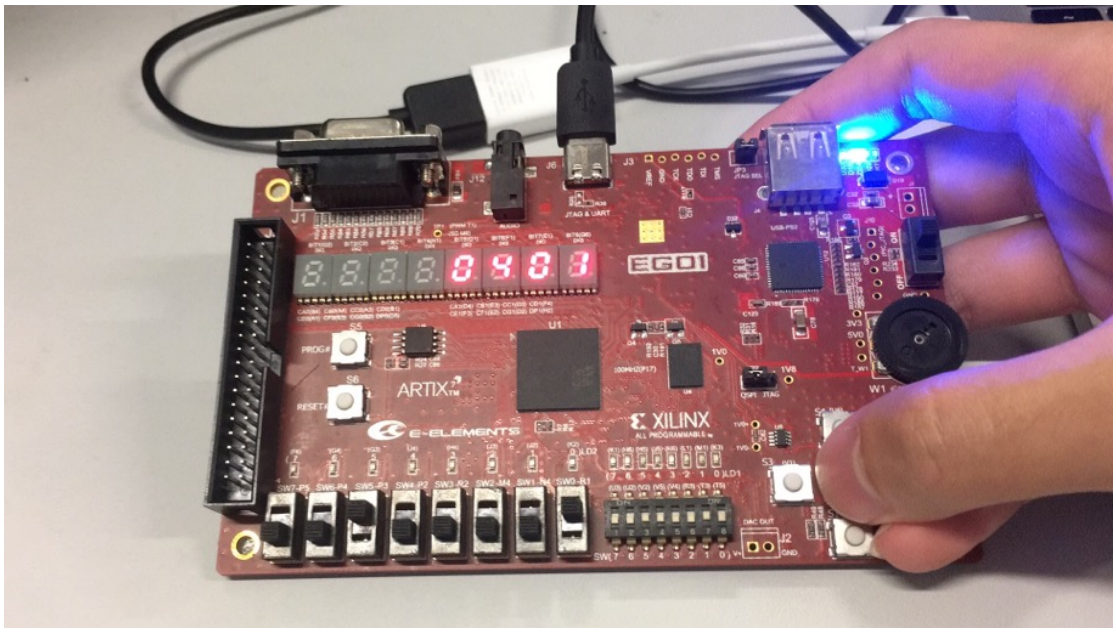


图 4 再次按下 S2 按钮

八、 实验体会

本次实验的 ROM 特意用了一下 IP 核，学习一下。原来就是一个别人写好的模块，也没什么特别的。ROM 的实验不难，RAM 的实验一开始让我没搞懂什么意思。后面看了一下原来就和之前一样，还是控制器-核心这样的架构，即控制器通过 4 位的代码来控制 RAM 芯片，并作出相应输出，理解后就不难了。

不过老师可以将 PPT 写完善一点，预习很多地方看不懂，只有到上课才知道怎么回事。其他都没毛病，不知道说啥好了。。