合肥工業大學
HEFEI UNIVERSITY OF TECHNOLOGY

计算机组成原理 ·实验报告·

本科实验报告

课程名称：计算机组成原理

姓　　名：党存远

学　　院：计算机科学与技术

专　　业：物联网工程

学　　号：2022217587

指导教师：丁贤庆

2024　年 6 月 9 日

合肥工业大学实验报告

学生姓名：党存远 班级：物联网工程 22-2 学号：2022217587

实验地点：电气实验楼 507 实验日期：2024 年 6 月 9 日

实验五　在 vivado 中进行运算器的设计

一、实验目的及环境

1、掌握一位全加器的工作原理和逻辑功能
2、掌握串行进位加法器的工作原理和进位延迟
3、掌握减法器的实现原理
4、掌握加减法器的设计方法
5、掌握 ALU 基本原理及在 CPU 中的作用
6、掌握 ALU 的设计方法
7、装有 vivado 的计算机　　1 台
8、EGO1 开发板　　1 块

二、实验目标及任务

1、采用原理图方式设计 4 位加/减法器
2、实现 4 位 ALU 及应用设计

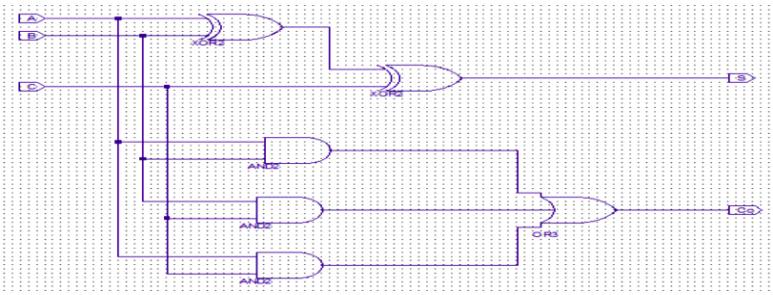
三、实验过程及记录

本节重点介绍实验的具体过程，包括：代码设计层次结构图及说明、源代码（包括注释）、PC 机上进行的关键步骤截图及说明、调试过程等，这部分的内容应当与实际操作过程类似即可（简单明了）。

1、根据实验指导书中的要求首先需要采用 4 个一位的全加器和附加逻辑设计两个四位数的加/减法器因此需要依次设计四位加/减法器

1 位全加器的设计

①电路原理图:



由上述电路原理图可得，我们需要使用 3 个与门、2 个异或门和一个或门来实现 1 位全加器，接下来按照电路原理图，使用代码将信号线连接即可。

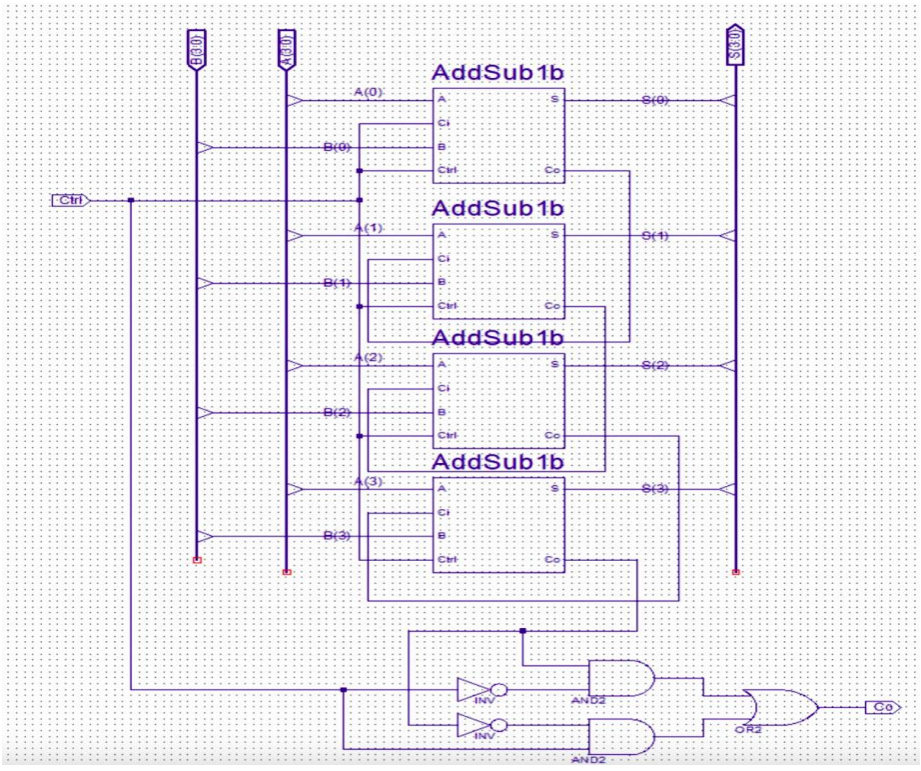
②源代码：

一位加法器：

```
`timescale 1ns / 1ps
module adder_1bit(
    input a,
    input b,
    input ci,
    output s,
    output co
    );
    and m0(c1, a, b);
    and m1(c2, b, ci);
    and m2(c3, a, ci);
    xor m3(s1, a, b);
    xor m4(s, s1, ci);
    or m5(co, c1, c2, c3);
endmodule
```

四个一位全加器和附加逻辑设计两个四位数的加/减法器：

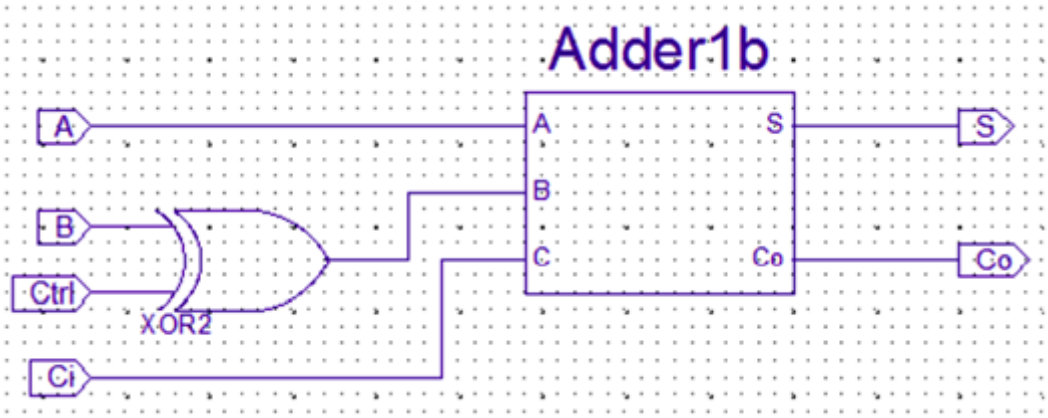
①电路原理图：



根据电路原理图可得，我们需要使用四个一位加减法器串联来实现一个四位

的加减法器，因此，我们先根据一位加减法的电路原理图来实现一位加法减器，再根据电路图，例化四个一位加法器，并将其串联，即可得到一个四位加减法器。

1 位加/减法器电路：



和上述相同，结合之前实现的一位加法器，实现一位加减法器。

②源代码：

一位加减法器：

```
`timescale 1ns / 1ps
module addsub_1bit(
    input a,
    input b,
    input ctrl,
    input ci,
    output s,
    output co
    );
    xor x(xor0, b, ctrl);
    adder_1bit adder(.a(a), .b(xor0), .ci(ci), .s(s), .co(co));
endmodule
```

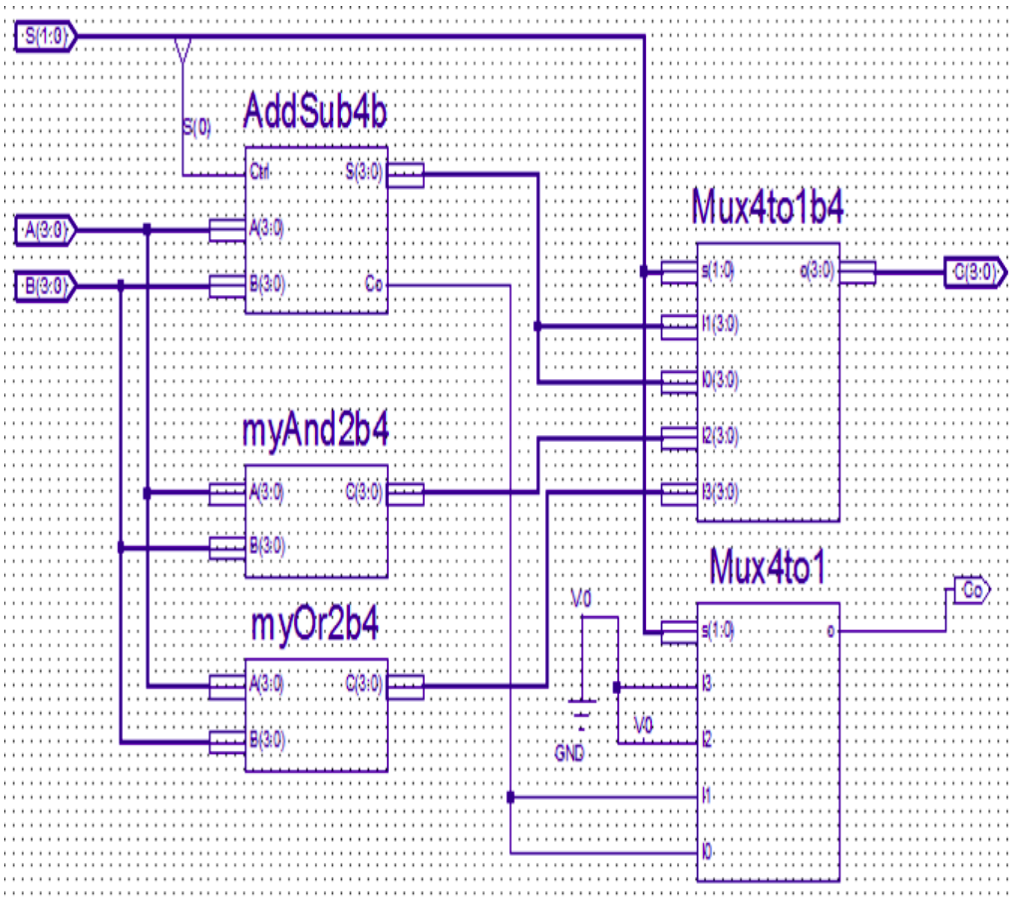
四位加减法器：

```
`timescale 1ns / 1ps
module addsub_4bit(
    input [3:0] a, b,
    input ctrl,
    output [3:0] s,
    output co
    );
    // 例化四个一位加减法器，并进行串联
    addsub_1bit
as0(.a(a[0]),.b(b[0]),.ci(ctrl),.ctrl(ctrl),.s(s[0]),.co(co0));
```

```
    addsub_1bit
as1(.a(a[1]),.b(b[1]),.ci(co0),.ctrl(ctrl),.s(s[1]),.co(co1));
    addsub_1bit
as2(.a(a[2]),.b(b[2]),.ci(co1),.ctrl(ctrl),.s(s[2]),.co(co2));
    addsub_1bit
as3(.a(a[3]),.b(b[3]),.ci(co2),.ctrl(ctrl),.s(s[3]),.co(co3));
    // 输出进位
    not not0(nc,ctrl);
    not not1(nco, co3);
    and and0(a0,co3,nc);
    and and1(a1, nco, ctrl);
    or or0(co, a0,a1);
endmodule
```

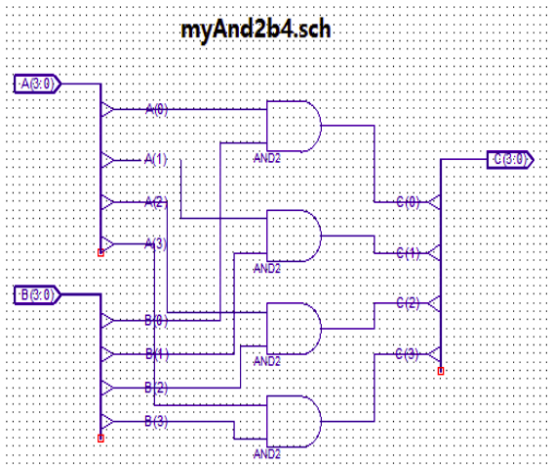
采用与逻辑、或逻辑和数据选择器设计两个四位数的加/减法器：

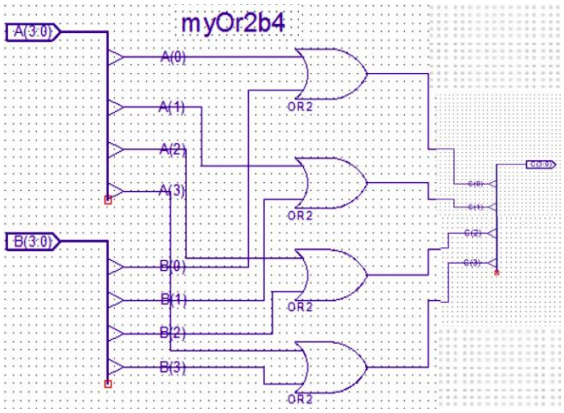
①电路原理图：



根据电路原理图，我们需要首先实现四位的与逻辑、或逻辑门和 1 位、4 位的数据选择器。

①电路原理图：





②源代码：

4 位与逻辑门：

```
`timescale 1ns / 1ps
module and_4bit(
    input [3:0] a, b,
    output reg[3:0] c
    );
    always @* begin
        c = a & b;
    end
endmodule
```

4 位或逻辑门：

```
`timescale 1ns / 1ps
module or_4bit(
    input [3:0] a, b,
    output reg[3:0] c
    );
    always @* begin
        c = a | b;
    end
endmodule
```

1 位 4 选 1 选择器：

```
`timescale 1ns / 1ps
module mux41_1bit(
  input i0, i1, i2, i3,
  input [1:0] s,
  output reg o
);
  always @* begin
    case(s)
      2'b00: o = i0;
      2'b01: o = i1;
      2'b10: o = i2;
      2'b11: o = i3;
    endcase
  end
endmodule
```

4 位 4 选 1 选择器：

```
`timescale 1ns / 1ps
module mux41_4bit(
    input [3:0] i0, i1, i2, i3,
    input [1:0] s,
    output reg [3:0] o
    );
    always @* begin
        case(s)
            2'b00: o = i0;
            2'b01: o = i1;
            2'b10: o = i2;
            2'b11: o = i3;
        endcase
    end
endmodule
```

4 位加/减法器：

```
`timescale 1ns / 1ps
module addsub_4bitmux(
    input [1:0] s,
    input [3:0] a, b,
    output [3:0]o,
    output co
    );
    // 声明数据线
    wire[3:0] st, cot, c0, c1;
    // 例化元件，并连接数据线
```

```
    addsub_4bit addsub4b(.a(a),.b(b),.ctrl(s[0]),.s(st),.co(cot));
    and_4bit and4b(.a(a),.b(b),.c(c0));
    or_4bit or4b(.a(a),.b(b),.c(c1));
    mux41_4bit mux41b4(.i0(st),.i1(st),.i2(c0),.i3(c1),.s(s),.o(o));
    mux41_1bit mux41(.i0(cot),.i1(cot),.i2(1'b0),.i3(1'b0),.s(s),.o(co));
endmodule
```

四、实验结果分析

1.这里应给出相应的实验结果。分析应有条理，要求采用规范的书面语。
2.每个实验都需要做模拟仿真，需要对仿真波形进行简单的文字说明。
3.对下载到开发板上的图片结果做分析说明。
4.原则上要求使用图片与文字结合的形式说明，因为 word 和 PDF 文档不支持视频，所以请不要使用视频文件。
5.图片请在垂直方向，不要横向。不要用很大的图片（不要超过 1M），请先做裁剪操作。

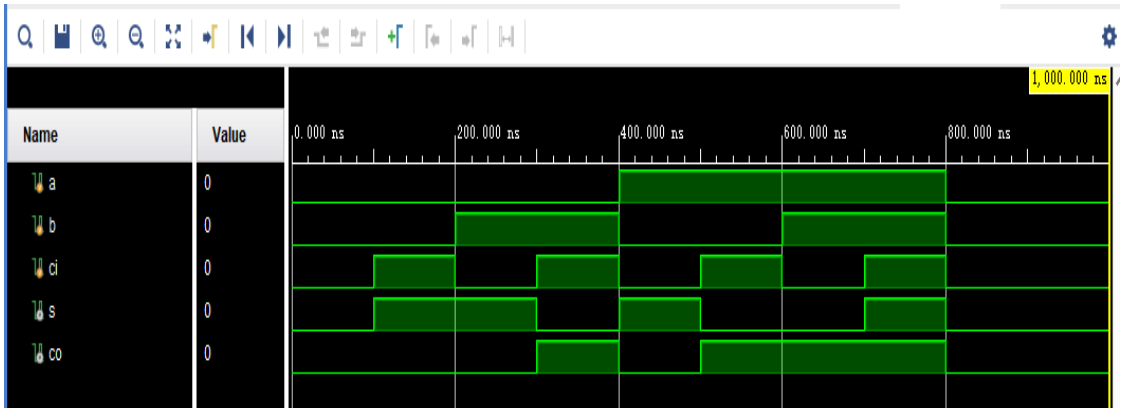
1、1 位全加器的设计:

①激励文件:

```
`timescale 1ns / 1ps
module adder_tb();
    reg a;
    reg b;
    reg ci;
    wire s;
    wire co;
    adder_1bit u0(.a(a), .b(b), .ci(ci), .s(s), .co(co));
    initial
    begin
        a = 1'b0;
        b = 1'b0;
        ci = 1'b0;
        #100;
        a = 1'b0;
        b = 1'b0;
        ci = 1'b1;
        #100;
        a = 1'b0;
        b = 1'b1;
        ci = 1'b0;
        #100;
        a = 1'b0;
        b = 1'b1;
        ci = 1'b1;
```

```
        #100
        a = 1'b1;
        b = 1'b0;
        ci = 1'b0;
        #100;
        a = 1'b1;
        b = 1'b0;
        ci = 1'b1;
        #100;
        a = 1'b1;
        b = 1'b1;
        ci = 1'b0;
        #100;
        a = 1'b1;
        b = 1'b1;
        ci = 1'b1;
        #100;
        a = 1'b0;
        b = 1'b0;
        ci = 1'b0;
        #100;
    end
endmodule
```

②仿真波形:



③仿真结果解释:

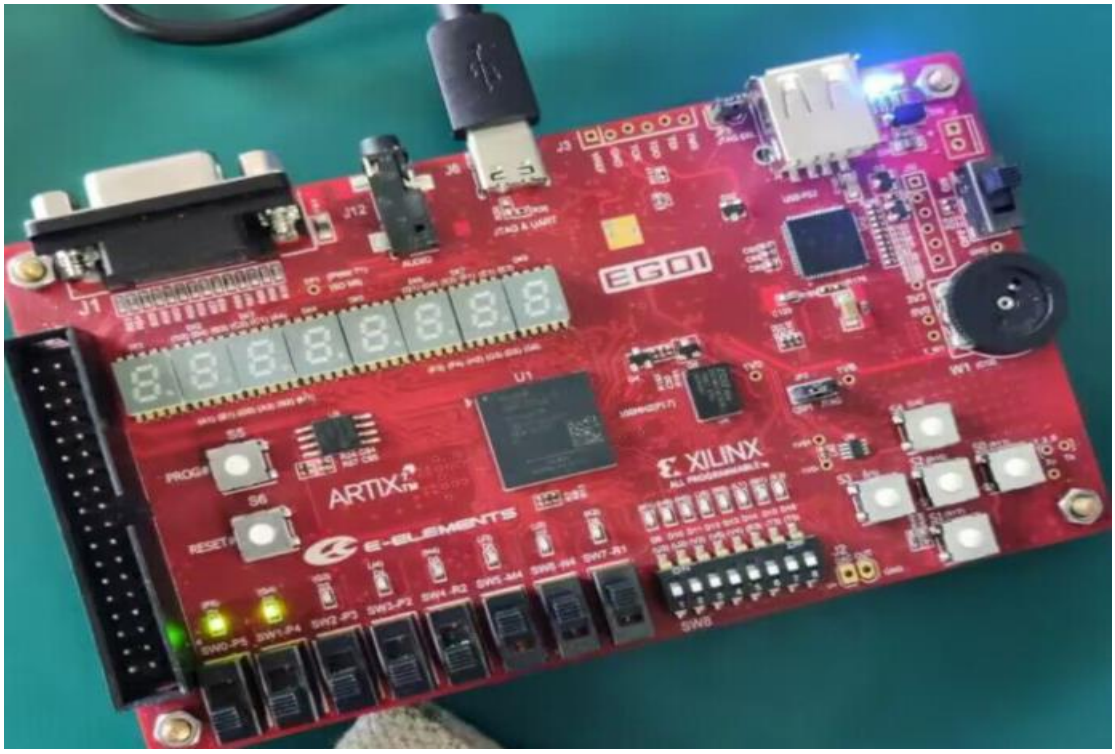
如图所示，设计 1 位全加器，0+0+0=0+0；0+0+1=1+0；0+1+0=1+0；0+1+1=0+1；1+0+0=1+0；1+0+1=0+1；1+1+1=1+1；

④约束文件:

```
set_property PACKAGE_PIN R1 [get_ports {a}]
set_property PACKAGE_PIN N4 [get_ports {b}]
set_property PACKAGE_PIN M4 [get_ports {ci}]
set_property PACKAGE_PIN G4 [get_ports {s}]
```

```
set_property PACKAGE_PIN F6 [get_ports {co}]
set_property IOSTANDARD LVCMOS33 [get_ports {a}]
set_property IOSTANDARD LVCMOS33 [get_ports {b}]
set_property IOSTANDARD LVCMOS33 [get_ports {ci}]
set_property IOSTANDARD LVCMOS33 [get_ports {s}]
set_property IOSTANDARD LVCMOS33 [get_ports {co}]
```

⑤开发板运行图片:



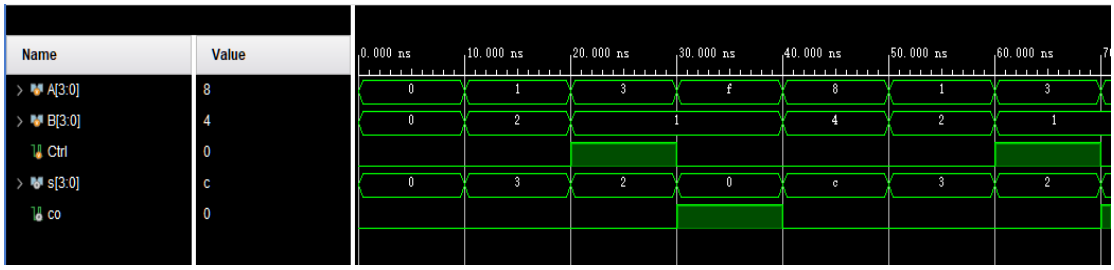
2、采用四个一位的全加器和附加逻辑设计两个四位数的加/减法器

①激励文件:

```
`timescale 1ns / 1ps
module addsub_4bit_tb();
    // Instantiate the DUT
    // Define inputs
    reg [3:0] A;
    reg [3:0] B;
    reg Ctrl;

    // Define outputs
    wire [3:0] s;
    wire co;

    addsub_4bit dut(
        .a(A),
        .b(B),
        .ctrl(Ctrl),
```

```
        .s(s),
        .co(co)
    );
    // Initialize inputs
    initial begin
        A = 4'b0000;
        B = 4'b0000;
        Ctrl = 0;
        end
    always begin
        #10 A = 4'b0001; B = 4'b0010; Ctrl = 0;
        // Subtract test case
        #10 A = 4'b0011; B = 4'b0001; Ctrl = 1;
        // Test case with carry
        #10 A = 4'b1111; B = 4'b0001; Ctrl = 0;
        // Test case with borrow
        #10 A = 4'b1000; B = 4'b0100; Ctrl = 0;
    end
```

②仿真波形:



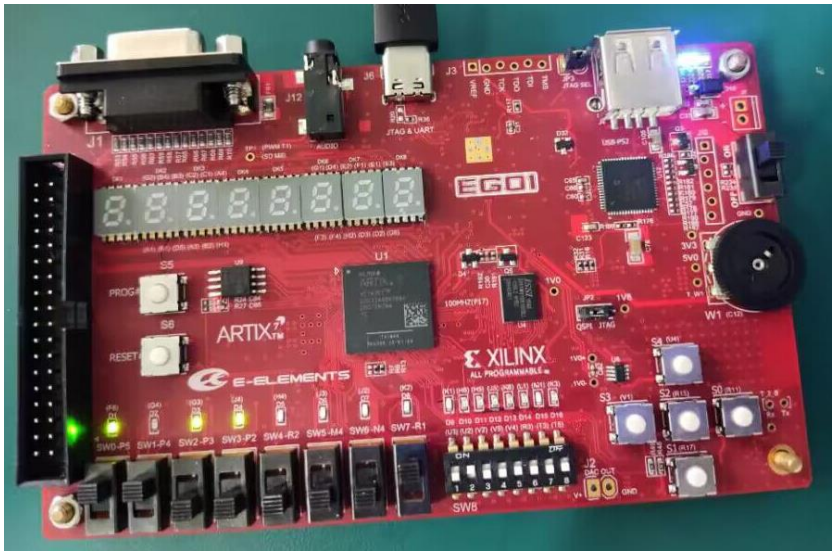
③仿真结果解释: 0+0=0+0；1+2=3+0；3-1=2；f+1=0+1（进位）；8+4=c+0；

④约束文件:

```
set_property PACKAGE_PIN P5 [get_ports {a[3]}]
set_property PACKAGE_PIN P4 [get_ports {a[2]}]
set_property PACKAGE_PIN P3 [get_ports {a[1]}]
set_property PACKAGE_PIN P2 [get_ports {a[0]}]
set_property PACKAGE_PIN R2 [get_ports {b[3]}]
set_property PACKAGE_PIN M4 [get_ports {b[2]}]
set_property PACKAGE_PIN N4 [get_ports {b[1]}]
set_property PACKAGE_PIN R1 [get_ports {b[0]}]
set_property PACKAGE_PIN U3 [get_ports {ctrl}]
set_property PACKAGE_PIN F6 [get_ports {s[3]}]
set_property PACKAGE_PIN G4 [get_ports {s[2]}]
set_property PACKAGE_PIN G3 [get_ports {s[1]}]
set_property PACKAGE_PIN J4 [get_ports {s[0]}]
set_property PACKAGE_PIN H4 [get_ports {co}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ctrl}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {co}]
```

⑤开发板运行图片:



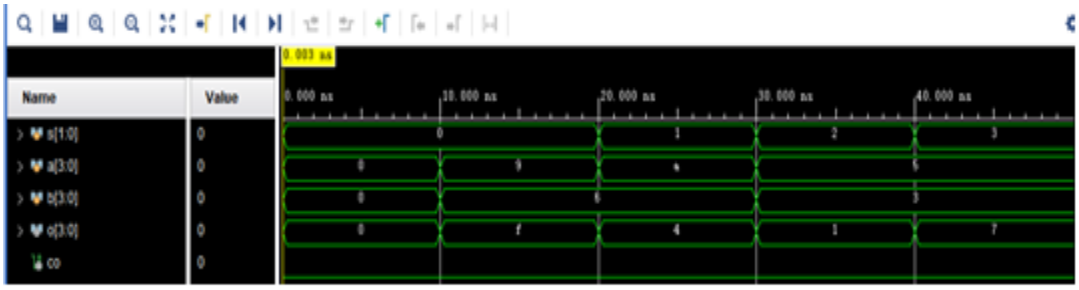
3、采用与逻辑、或逻辑和数据选择器设计两个四位数的加/减法器

①激励文件:

```
`timescale 1ns / 1ps
module addsub_4bitmux_tb();
    reg [1:0] s;
    reg [3:0] a, b;
    wire [3:0] o;
    wire co;
    addsub_4bitmux addsub4bmux(.s(s), .a(a), .b(b), .o(o), .co(co));
    initial begin
        #0
        s = 2'b00;
        a = 4'b0000;
        b = 4'b0000;
```

```
        #10
        s = 2'b00;
        a = 9;
        b = 4'b0110;
        #10
        s = 2'b01;
        a = 4'b1010;
        b = 4'b0110;
        #10
        s = 2'b10;
        a = 4'b0101;
        b = 4'b0011;
        #10
        s = 2'b11;
        a = 4'b0101;
        b = 4'b0011;
        #10
        $finish;
    end
endmodule
```

②仿真波形:



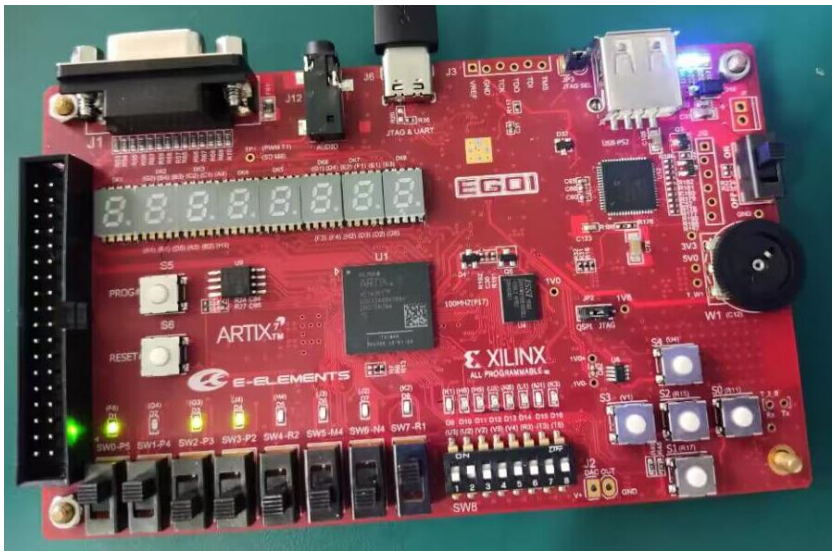
③仿真结果解释: 0+0=0；9+6=f；a-6=4；5-3-1=1；5-1+3=7；

④约束文件:

```
set_property PACKAGE_PIN P5 [get_ports {a[3]}]
set_property PACKAGE_PIN P4 [get_ports {a[2]}]
set_property PACKAGE_PIN P3 [get_ports {a[1]}]
set_property PACKAGE_PIN P2 [get_ports {a[0]}]
set_property PACKAGE_PIN R2 [get_ports {b[3]}]
set_property PACKAGE_PIN M4 [get_ports {b[2]}]
set_property PACKAGE_PIN N4 [get_ports {b[1]}]
set_property PACKAGE_PIN R1 [get_ports {b[0]}]
set_property PACKAGE_PIN F6 [get_ports {o[3]}]
set_property PACKAGE_PIN G4 [get_ports {o[2]}]
set_property PACKAGE_PIN G3 [get_ports {o[1]}]
set_property PACKAGE_PIN J4 [get_ports {o[0]}]
set_property PACKAGE_PIN U3 [get_ports {s[1]}]
```

```
set_property PACKAGE_PIN U2 [get_ports {s[0]}]
set_property PACKAGE_PIN H4 [get_ports {co}]

set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {co}]
```

⑤开发板运行图片:



五、遇到的问题与实验心得体会

简要地叙述一下实验过程中的感受，以及其他的问题描述和自己的感想。特别是实验中遇到的困难，最后如何解决的。在用 verilog 代码写程序时遇到语法或其他错误，如何修改解决的。

在本次的计算机组成实验中，我进行了一系列的工作，包括根据 4 位加/减法器的原理图，实现了 4 位 ALU 以及在 FPGA 开发板上运行代码。这个实验过程中，我积累了很多宝贵的经验，并且对计算机组成原理有了更深入的理解，了解了 ALU 是如何工作的。

在本次实验中，遇到的困难主要在于约束文件的编写，仅在参考指导书的过程中，直接的从参考书上照搬代码，却漏写了空格或引脚书写错误，导致后续的

生成比特流阶段报错，却难以找到错误。

另外，对于 verilog 语言的不熟悉，导致在编写的过程中，出现了很多语法上的错误，在编写的过程中，还需要不断的翻阅文档，了解如何对元件进行例化，还有一些变量声明上的注意事项。经过这次实验后，对于 verilog 的使用也更加熟练了。

vivado 这个软件本身就具有复杂性，在一开始，还需要不断借助指导书，才能创建一个正确的项目，在后续熟练后，也可以很快的创建项目，并向其中添加自己的代码了。

经过本次实验，我认为最重要的是要看懂电路原理图，理解了电路原理图，我们就能很快的实现出电路原理图中的每一个模块，并且最后将每一个模块相联，完成最终的功能实现。

总的来说，这个计算机组成实验给我带来了很多挑战，但也让我获得了很多宝贵的经验和技能。通过不断尝试和修正，我逐渐掌握了设计和实现加/减法器、ALU 以及在 FPGA 上运行代码的方法。我深刻认识到了实践的重要性，并且学会了如何在面对困难时保持耐心和解决问题的能力。这个实验让我更加熟悉计算机组成原理，并对我今后的学习和工作具有很大的帮助。

实验六　8 位乘法器设计与实现

一、实验目的及环境

1、了解 16 位有符号、无符号乘法器的实现原理。

2、使用 Verilog 实现 16 位无符号乘法器和有符号乘法器。

3、装有 vivado 的计算机　　1 台

4、EGO1 开发板　　1 块

二、实验目标及任务

1、设计无符号乘法器，将两个 8 位无符号数相乘,得到一个 16 位无符号数。

2、设计有符号乘法器，将两个 8 位有符号数相乘,得到一个 16 位有符号数。

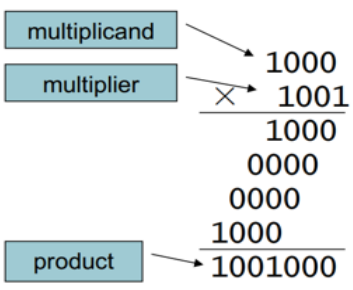
3、设计按键输入转换电路，设计七段显示器显示和控制电路：　通过 8 个按键输入两个 4 位的二进制乘数 A 和 B，两个乘数和乘积的结果，分别送给多个七段显示器进行显示。

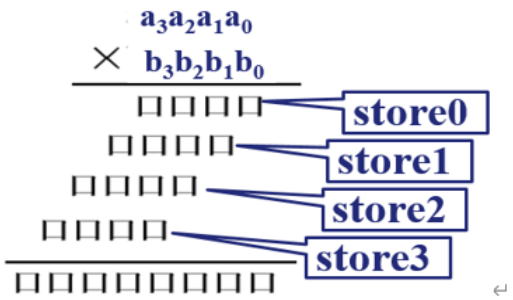
三、实验过程及记录

本节重点介绍实验的具体过程，包括：代码设计层次结构图及说明、源代码（包括注释）、PC 机上进行的关键步骤截图及说明、调试过程等，这部分的内容应当与实际操作过程类似即可（简单明了）。

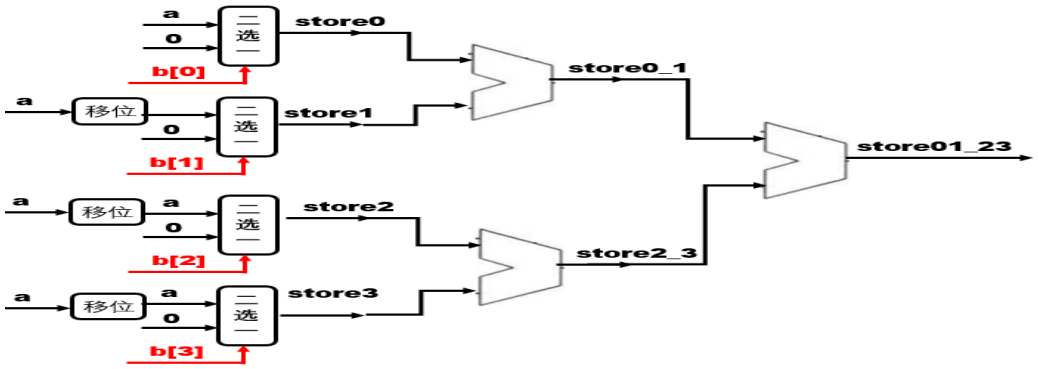
下图的乘法过程是手算的一个步骤，被乘数 x 是 1000，乘数 y 为 1001，在使用计算机实现乘法运算时，我们就可以通过模拟这个手算的过程来实现我们的乘法运算操作。

①电路原理图





根据该手工计算过程，得到电路原理图:



根据该电路原理图，我们可以相似的得到两个 16 位无符号数相乘的电路，根据该电路，我们即可通过代码来实现该乘法器。

②源代码:

16 位乘法器:

```
`timescale 1ns / 1ps
module MULTU(
    input clk,    //乘法器时钟信号
    input reset,
    input [ 15:0]  a,      //输入 a(被乘数)
    input [15:0]  b,      //输入 b(乘数)
    output [ 31:0]  z     //乘积输出 z
    );
    reg [31:0] temp;    //申请寄存器
    reg [31:0] stored [15:0];
    reg [31:0] addi_i1 [7:0];
    reg [31:0] additi1_i2ti3[3:0];
    reg [31:0] add[0:1];
    integer i;
    always @(posedge clk or negedge reset) begin
        // 初始化
        if(reset) begin
            temp <= 0;
            for (i = 0;i < 16;i = i + 1) begin
                stored[i] = 0;
            end
            for (i = 0;i < 8;i = i + 1) begin
                addi_i1[i] = 0;
            end
            for (i = 0;i < 4;i = i + 1) begin
                additi1_i2ti3[i] = 0;
            end
        end
        // 计算过程
```

```
        else begin
            stored[0] = b[0] ? {16'b0, a} : 32'b0;
            stored[1] = b[1] ? {15'b0, a, 1'b0} : 32'b0;
            stored[2] = b[2] ? {14'b0, a, 2'b0} : 32'b0;
            stored[3] = b[3] ? {13'b0, a, 3'b0} : 32'b0;
            stored[4] = b[4] ? {12'b0, a, 4'b0} : 32'b0;
            stored[5] = b[5] ? {11'b0, a, 5'b0} : 32'b0;
            stored[6] = b[6] ? {10'b0, a, 6'b0} : 32'b0;
            stored[7] = b[7] ? {9'b0, a, 7'b0} : 32'b0;
            stored[8] = b[8] ? {8'b0, a, 8'b0} : 32'b0;
            stored[9] = b[9] ? {7'b0, a, 9'b0} : 32'b0;
            stored[10] = b[10] ? {6'b0, a, 10'b0} : 32'b0;
            stored[11] = b[11] ? {5'b0, a, 11'b0} : 32'b0;
            stored[12] = b[12] ? {4'b0, a, 12'b0} : 32'b0;
            stored[13] = b[13] ? {3'b0, a, 13'b0} : 32'b0;
            stored[14] = b[14] ? {2'b0, a, 14'b0} : 32'b0;
            stored[15] = b[15] ? {1'b0, a, 15'b0} : 32'b0;
            for(i = 0;i < 8;i = i + 1)begin
                addi_i1[i] = stored[2 * i] + stored[2 * i + 1];
            end
            for(i = 0;i < 4;i = i + 1) begin
                additi1_i2ti3[i] = addi_i1[2 * i] + addi_i1[2 * i + 1];
            end
            for(i = 0;i < 2;i = i + 1) begin
                add[i] = additi1_i2ti3[2 * i] + additi1_i2ti3[2 * i + 1];
            end
            temp = add[0] + add[1];
        end
    end
    assign z = temp;
endmodule
```

时钟模块:

```
`timescale 1ns / 1ps
//数码管动态扫描所需要的时钟模块
module CLK_DIV (
    input  clk0,
    output clk
);
    reg clk = 0;
    parameter N = 32'd25000;
    parameter WIDTH = 32 - 1;
    reg [WIDTH : 0] number = 0;
    always @(posedge clk0) begin
        if (number == N - 1) begin
```

```
            number <= 0;
            clk <= ~clk;
        end else begin
            number <= number + 1;
        end
    end
endmodule
```

4 位 8421 码转换:

```
`timescale 1ns / 1ps
module Bit4_to_8421 (
    input      [3:0] a,
    output reg [7:0] bcd
);
    reg [3:0] temp1, temp2;
    always @(*) begin
        temp1 <= a / 10;
        temp2 <= a - temp1 * 10;
        bcd   <= {temp1, temp2};
    end
endmodule
```

8 位 8421 码转换:

```
`timescale 1ns / 1ps
module Bit8_to_8421 (
    input      [ 7:0] a,
    output reg [15:0] bcd
);
    reg [3:0] temp1, temp2, temp3, temp4;
    always @(*) begin
        temp1 <= a / 1000;
        temp2 <= (a - temp1 * 1000) / 100;
        temp3 <= (a - temp1 * 1000 - temp2 * 100) / 10;
        temp4 <= a - temp1 * 1000 - temp2 * 100 - temp3 * 10;
        bcd   <= {temp1, temp2, temp3, temp4};
    end
endmodule
```

数码管显示模块:

```
`timescale 1ns / 1ps
//控制 8 个数码管动态显示的模块。segn[3:0]是 8421BCD 码。
module ShowEightSeg7 (
    input            clk,
    input      [3:0] seg0,
    input      [3:0] seg1,
    input      [3:0] seg2,
```

```
    input      [3:0] seg3,
    input      [3:0] seg4,
    input      [3:0] seg5,
    input      [3:0] seg6,
    input      [3:0] seg7,
    output reg [7:0] an,
    output reg [7:0] segout_1,
    output reg [7:0] segout_2
);
    reg [2:0] state = 0;
    reg [3:0] bcd = 0;
    always @(posedge clk) begin
        state <= state + 1;
    end
    always @(*) begin
        case (state)
            0: begin
                an  = 8'b00000001;
                bcd = seg0;
            end
            1: begin
                an  = 8'b00000010;
                bcd = seg1;
            end
            2: begin
                an  = 8'b00000100;
                bcd = seg2;
            end
            3: begin
                an  = 8'b00001000;
                bcd = seg3;
            end
            4: begin
                an  = 8'b00010000;
                bcd = seg4;
            end
            5: begin
                an  = 8'b00100000;
                bcd = seg5;
            end
            6: begin
                an  = 8'b01000000;
                bcd = seg6;
            end
```

```
            7: begin
                an  = 8'b10000000;
                bcd = seg7;
            end
            default: begin
                an  = 8'b0000000;
                bcd = 4'b0000;
            end
        endcase
    end
    always @(*) begin
        if (state < 4) begin
            case (bcd)
                4'h0:    segout_1 = 8'hfc;
                4'h1:    segout_1 = 8'h60;
                4'h2:    segout_1 = 8'hda;
                4'h3:    segout_1 = 8'hf2;
                4'h4:    segout_1 = 8'h66;
                4'h5:    segout_1 = 8'hb6;
                4'h6:    segout_1 = 8'hbe;
                4'h7:    segout_1 = 8'he0;
                4'h8:    segout_1 = 8'hfe;
                4'h9:    segout_1 = 8'hf6;
                default: segout_1 = 8'h00;
            endcase
            segout_2 = 8'h00;
        end else begin
            case (bcd)
                4'h0:    segout_2 = 8'hfc;
                4'h1:    segout_2 = 8'h60;
                4'h2:    segout_2 = 8'hda;
                4'h3:    segout_2 = 8'hf2;
                4'h4:    segout_2 = 8'h66;
                4'h5:    segout_2 = 8'hb6;
                4'h6:    segout_2 = 8'hbe;
                4'h7:    segout_2 = 8'he0;
                4'h8:    segout_2 = 8'hfe;
                4'h9:    segout_2 = 8'hf6;
                default: segout_2 = 8'h00;
            endcase
            segout_1 = 8'h00;
        end
    end
endmodule
```

Top 顶层模块:

```
`timescale 1ns / 1ps

module top (
    input        clk,
    input  [3:0] data1,
    input  [3:0] data2,
    output [7:0] an,
    output [7:0] mul,
    output [7:0] result
);
    reg  reset = 0;
    // 时钟信号
    // reg        clk = 0;
    wire clk1;
    CLK_DIV u_CLK_DIV (
        .clk0(clk),
        .clk (clk1)
    );
    wire [7:0] z;
    // 乘法器模块
    MULTU_4bit u_MULTU_4bit (
        .clk  (clk1),
        .reset(reset),
        .a    (data1),
        .b    (data2),
        .z    (z)
    );
    // 将 data1, data2, result 转换为 8421BCD 码
    wire [ 7:0] data1_bcd;
    wire [ 7:0] data2_bcd;
    wire [15:0] z_bcd;
    Bit4_to_8421 u_Bit4_to_8421_1 (
        .a  (data1),
        .bcd(data1_bcd)
    );
    Bit4_to_8421 u_Bit4_to_8421_2 (
        .a  (data2),
        .bcd(data2_bcd)
    );
    Bit8_to_8421 u_Bit8_to_8421 (
        .a  (z),
        .bcd(z_bcd)
    );
```

```
    // 数码管显示被乘数、乘数和结果
    ShowEightSeg7 u_ShowEightSeg7 (
        .clk     (clk1),
        .seg0    (data1_bcd[7:4]),
        .seg1    (data1_bcd[3:0]),
        .seg2    (data2_bcd[7:4]),
        .seg3    (data2_bcd[3:0]),
        .seg4    (z_bcd[15:12]),
        .seg5    (z_bcd[11:8]),
        .seg6    (z_bcd[7:4]),
        .seg7    (z_bcd[3:0]),
        .an      (an),
        .segout_1(mul),
        .segout_2(result)
    );

    initial begin
        reset = 1;
    end
endmodule
```

四、实验结果分析

1.这里应给出相应的实验结果。分析应有条理，要求采用规范的书面语。
2.每个实验都需要做模拟仿真，需要对仿真波形进行简单的文字说明。
3.对下载到开发板上的图片结果做分析说明。
4.原则上要求使用图片与文字结合的形式说明，因为 word 和 PDF 文档不支持视频，所以请不要使用视频文件。
5.图片请在垂直方向，不要横向。不要用很大的图片（不要超过 1M），请先做裁剪操作。

①激励文件:

4 位乘法器:

```
`timescale 1ns / 1ps

module tb_MULTU_4bit;
    parameter XH = 2022217587;//学号
    // Inputs
    reg         clk = 0;
    reg         reset = 0;
    reg  [3:0] a;
    reg  [3:0] b;
    // Outputs
    wire [7:0] z;
```

```
    // 模拟使用时钟信号
    always begin
        #(XH / 2) clk <= ~clk;
    end
    // 实例化 MULTU_4bit 模块
    MULTU_4bit u_MULTU_4bit (
        .clk  (clk),
        .reset(reset),
        .a    (a),
        .b    (b),
        .z    (z)
    );
    // 初始化
    initial begin
        reset = 1;
        #XH;
        a <= 4'b0011;
        b <= 4'b0001;
        #XH;
        a <= 4'b0100;
        b <= 4'b0100;
        #XH;
        a <= 4'b0000;
        b <= 4'b0000;
        #XH;
        a <= 4'b0010;
        b <= 4'b1000;
        #XH;
        a <= 4'b1111;
        b <= 4'b0001;
        #XH;
    end
endmodule
```

16 位乘法器:

```
`timescale 1ns / 1ps
module MULTUtb();
  reg clk;
  reg reset;
  reg [15:0] a;
  reg [15:0] b;
  wire [31:0] z;
  // 实例化被测试的 MULTU 模块
  MULTU dut (
    .clk(clk),
```

```
    .reset(reset),
    .a(a),
    .b(b),
    .z(z)
);
// 时钟生成
always #5 clk = ~clk;
// 初始化
initial begin
    clk = 0;
    reset = 1;
    #10 reset = 0;  // 复位信号保持为低电平一段时间后释放

    #10;
    // 输入值设置
    a = 16'b0;  // 设置被乘数
    b = 16'b0;   // 设置乘数

    #10
    a = 16'd7;
    b = 16'd8;

    #10
    a = 16'd9;
    b = 16'd9;

    #10
    a = 16'd3;
    b = 16'd2;

    #10;
    // 输入值设置
    a = 16'b0;  // 设置被乘数
    b = 16'b1111111111111111;   // 设置乘数

    #10;
    // 输入值设置
    a = 16'b1010101010101010;  // 设置被乘数
    b = 16'b0;   // 设置乘数

    #10;
    // 输入值设置
    a = 16'b1111111111111111;  // 设置被乘数
    b = 16'b1111111111111111;   // 设置乘数
```

```
    #10;
    // 输入值设置
    a = 16'b1000000000000000;  // 设置被乘数
    b = 16'b1010101010101010;   // 设置乘数

    #10;
    // 输入值设置
    a = 16'b1010101010101010;  // 设置被乘数
    b = 16'b1000000000000000;   // 设置乘数

    #10;
    // 输入值设置
    a = 16'b1011110110111101;  // 设置被乘数
    b = 16'b1101000011010000;   // 设置乘数

    #10;
    // 输入值设置
    a = 16'b1000111111110001;  // 设置被乘数
    b = 16'b1110111001111110;   // 设置乘数
    $finish;
  end
endmodule
```

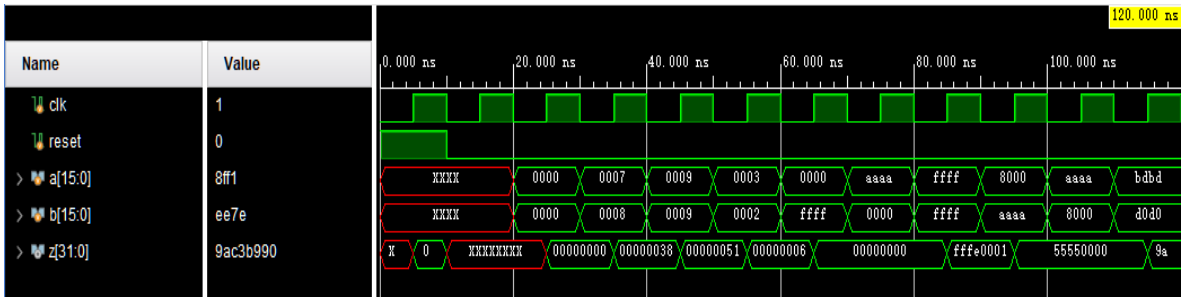
②约束文件:

```
# ============================== 时钟 ============================== #
set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports
{clk}]
# ============================== 拨码开关 sw0~sw7
============================== #
set_property -dict {PACKAGE_PIN P5 IOSTANDARD LVCMOS33} [get_ports
{data1[3]}]
set_property -dict {PACKAGE_PIN P4 IOSTANDARD LVCMOS33} [get_ports
{data1[2]}]
set_property -dict {PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports
{data1[1]}]
set_property -dict {PACKAGE_PIN P2 IOSTANDARD LVCMOS33} [get_ports
{data1[0]}]
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports
{data2[3]}]
set_property -dict {PACKAGE_PIN M4 IOSTANDARD LVCMOS33} [get_ports
{data2[2]}]
set_property -dict {PACKAGE_PIN N4 IOSTANDARD LVCMOS33} [get_ports
{data2[1]}]
```

```
set_property -dict {PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports
{data2[0]}]

# ============================== 数码管位选信号
============================== #
set_property -dict {PACKAGE_PIN G2 IOSTANDARD LVCMOS33} [get_ports
{an[0]}]
set_property -dict {PACKAGE_PIN C2 IOSTANDARD LVCMOS33} [get_ports
{an[1]}]
set_property -dict {PACKAGE_PIN C1 IOSTANDARD LVCMOS33} [get_ports
{an[2]}]
set_property -dict {PACKAGE_PIN H1 IOSTANDARD LVCMOS33} [get_ports
{an[3]}]
set_property -dict {PACKAGE_PIN G1 IOSTANDARD LVCMOS33} [get_ports
{an[4]}]
set_property -dict {PACKAGE_PIN F1 IOSTANDARD LVCMOS33} [get_ports
{an[5]}]
set_property -dict {PACKAGE_PIN E1 IOSTANDARD LVCMOS33} [get_ports
{an[6]}]
set_property -dict {PACKAGE_PIN G6 IOSTANDARD LVCMOS33} [get_ports
{an[7]}]

# ============================== 数码管段选信号
============================== #
set_property -dict {PACKAGE_PIN B4 IOSTANDARD LVCMOS33} [get_ports
{mul[7]}]
set_property -dict {PACKAGE_PIN A4 IOSTANDARD LVCMOS33} [get_ports
{mul[6]}]
set_property -dict {PACKAGE_PIN A3 IOSTANDARD LVCMOS33} [get_ports
{mul[5]}]
set_property -dict {PACKAGE_PIN B1 IOSTANDARD LVCMOS33} [get_ports
{mul[4]}]
set_property -dict {PACKAGE_PIN A1 IOSTANDARD LVCMOS33} [get_ports
{mul[3]}]
set_property -dict {PACKAGE_PIN B3 IOSTANDARD LVCMOS33} [get_ports
{mul[2]}]
set_property -dict {PACKAGE_PIN B2 IOSTANDARD LVCMOS33} [get_ports
{mul[1]}]
set_property -dict {PACKAGE_PIN D5 IOSTANDARD LVCMOS33} [get_ports
{mul[0]}]

set_property -dict {PACKAGE_PIN D4 IOSTANDARD LVCMOS33} [get_ports
{result[7]}]
```

```
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports
{result[6]}]
set_property -dict {PACKAGE_PIN D3 IOSTANDARD LVCMOS33} [get_ports
{result[5]}]
set_property -dict {PACKAGE_PIN F4 IOSTANDARD LVCMOS33} [get_ports
{result[4]}]
set_property -dict {PACKAGE_PIN F3 IOSTANDARD LVCMOS33} [get_ports
{result[3]}]
set_property -dict {PACKAGE_PIN E2 IOSTANDARD LVCMOS33} [get_ports
{result[2]}]
set_property -dict {PACKAGE_PIN D2 IOSTANDARD LVCMOS33} [get_ports
{result[1]}]
set_property -dict {PACKAGE_PIN H2 IOSTANDARD LVCMOS33} [get_ports
{result[0]}]
```

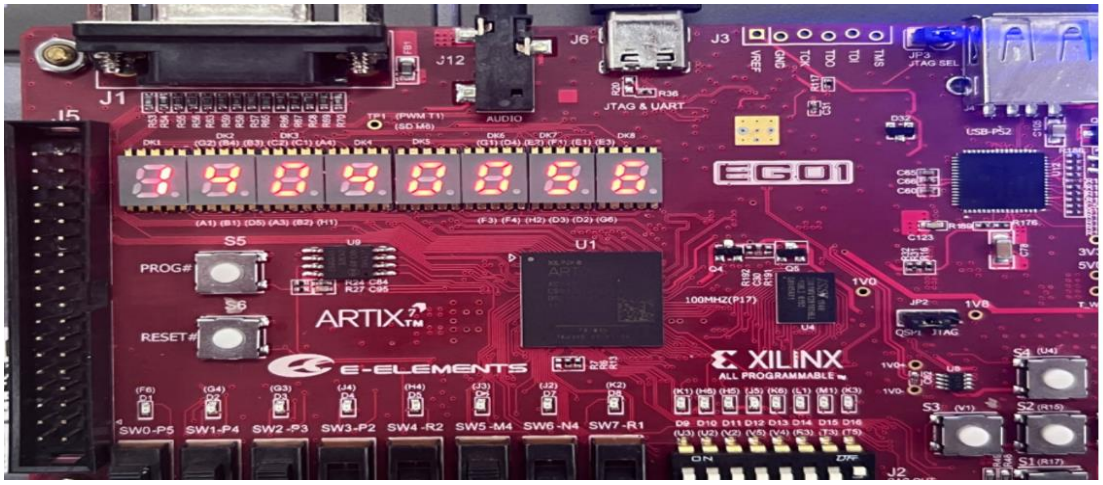
③仿真波形:



④结果解释:

在上升沿时候进行计算，在开始时候，rst 为 1，将数据复位，从第三个上升沿开始计算，计算 a*b=z；0*0=0；7h*8h=78h；9h*9h=51h；3h*2h=6h；0h*ffffh=0h；ffffh*ffffh=fffe0001h；结果均为正确。

⑤开发板运行图片:



五、遇到的问题与实验心得体会

简要地叙述一下实验过程中的感受，以及其他的问题描述和自己的感想。特别是实验中遇到的困难，最后如何解决的。在用 verilog 代码写程序时遇到语法或其他错误，如何修改解决的。

在计算机组成实验中设计和实现无符号乘法器是一项具有挑战性的任务。在本次实验中，我设计了一个无符号乘法器，用于将两个 16 位无符号数相乘并得到一个 32 位无符号数的结果。

在实验过程中，我首先深入理解了无符号乘法的原理和算法。我了解了乘法器的结构和工作原理，并明确了所需的输入和输出信号。在实现的过程中，我参考了指导书中的 2 个 8 位无符号数相乘的电路原理图和代码实现，并对其进行了拓展，拓展为 2 个 16 位无符号数相乘，因此整个实现过程也相对容易，没有遇到太大的困难。

本次实验中较为复杂的部分是 8421 码的转换以及数码管的显示，在编写这部分的时候，我通过依次计算每一位实现 8421 的转换。计算千位时计算输入值 a 除以 1000 的结果。由于 a 是 8 位的，其值最大为 255，因此在实际情况中 temp1 总是为 0。计算百位: 接下来从 a 中减去 temp1 * 1000（实际上就是 a 自身，因为 temp1 为 0），然后除以 100，得到百位数。之后计算十位: 从 a 减去千位和百位之和的结果，然后除以 10，得到十位数。最后计算个位数，就是从 a 减去前三个数位计算得到的总和。组合 BCD 输出 (bcd): 最后，四个临时寄存器的值被组合成一个 16 位的输出值，格式为{temp1, temp2, temp3, temp4}，每个临时寄存器对应 BCD 输出的一个十进制位。对于数码管显示的部分，七段显示编码是硬编码在模块内部，对应 BCD 值从 0 到 9 的每个数字的显示方式。例如，4'h0 （BCD 的 0）对应的编码是 8'hfc，这表示数码管上需要点亮的段来正确显示数字 0。

经过本次实验过后，对于乘法器在计算机中的底层实现原理，有了更深的认识。在乘法器的实现中，需要考虑乘法操作的流程、进位的处理和结果的输出。

实验七　寄存器堆的设计和实现

一、实验目的及环境

1、深入了解寄存器堆的结构和工作原理。

2、使用 Verilog HDL 语言来设计和实现寄存器堆结构，进行仿真和下载验证。

3、装有 vivado 的计算机　　1 台

4、EGO1 开发板　　1 块

二、实验目标及任务

1、设计由 16 个 4 位寄存器构成的寄存器堆，该寄存器堆是双端口输出。进行功能仿真

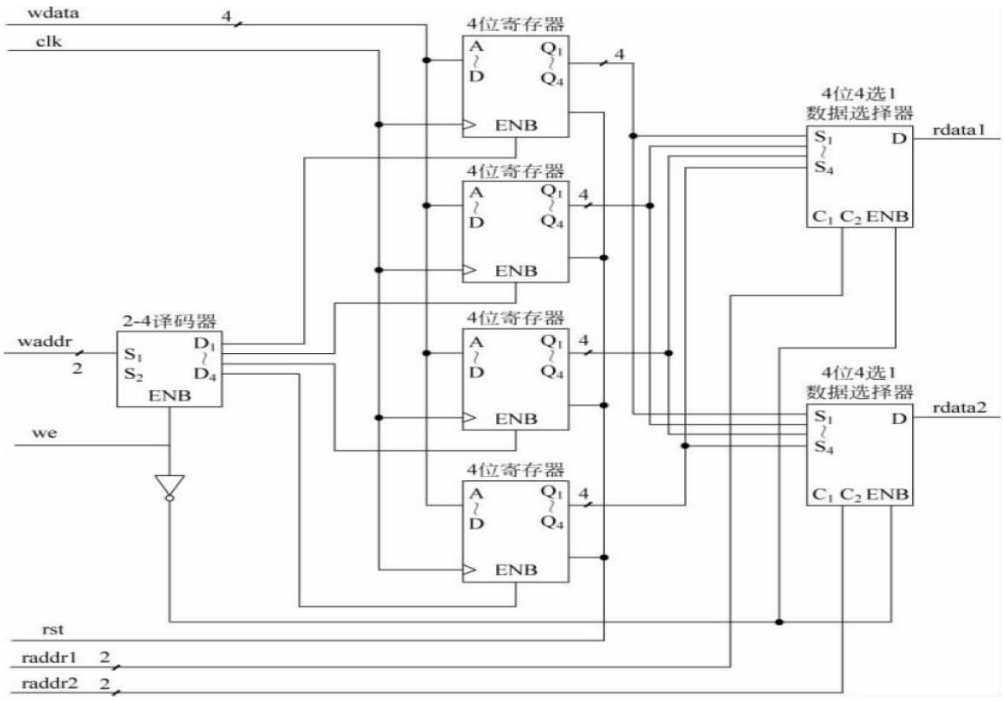
2、设计由 16 个 4 位寄存器构成的寄存器堆，引脚绑定后进行下载，在 EGO1 开发板上进行数据验证。

三、实验过程及记录

本节重点介绍实验的具体过程，包括：代码设计层次结构图及说明、源代码（包括注释）、PC 机上进行的关键步骤截图及说明、调试过程等，这部分的内容应当与实际操作过程类似即可（简单明了）。

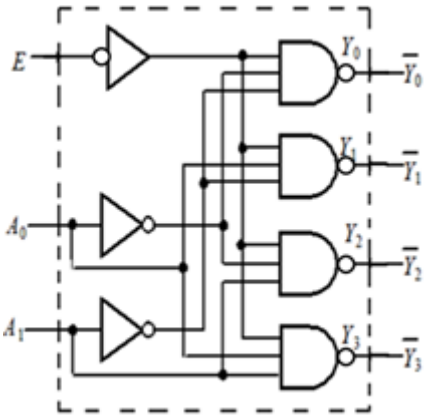
①电路原理图:

寄存器堆:



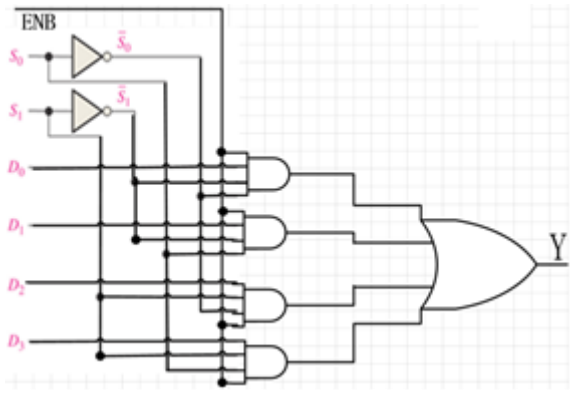
上图是 4 个 4 位寄存器构成的寄存器堆的原理图，相似的，我们可以根据这张图得到 16 个 4 位寄存器的寄存器堆：添加更多的寄存器至 16 个，将 2-4 译码器改为 4-16 译码器，将 4 位 4 选 1 数据选择器改为 4 位的 16 选 1 数据选择器。接下来的目标为：实现 4-16 译码器、16 选 1 数据选择器和 4 位寄存器。

4-16 译码器:



先观察 2-4 译码器的原理图，然后将其拓展为一个 4-16 译码器的原理图，并按照如上原理图来将其实现。

16 选 1 数据选择器:



同样的，先观察 4 选 1 数据选择器的原理图，然后将其拓展为一个 16 选 1 数据选择器，并按照上述原理图将其中的逻辑门例化。

寄存器：在寄存器中，我们需要在寄存器中先声明一个 reg_data 的寄存器类型变量，用于存储该寄存器中的值，然后响应下降沿，根据 rst 位，执行对应的操作即可。

最后是顶层文件的设计，根据寄存器堆的原理图，确定输入输出信号，然后声明一些中间的信号线，然后根据原理图，将元件一一例化，并将数据线或信号线相连。

②源代码:

4-16 译码器:

```
`timescale 1ns / 1ps
module decode(
    input we,
    input [3:0] waddr,
    output wire [15:0] D
    );
    wire not_we;
    wire [3:0] not_waddr;
    not(not_we, we);
    not(not_waddr[0], waddr[0]);
    not(not_waddr[1], waddr[1]);
    not(not_waddr[2], waddr[2]);
    not(not_waddr[3], waddr[3]);
    nand(D[0], not_we, not_waddr[3], not_waddr[2],
not_waddr[1],not_waddr[0]);
    nand(D[1], not_we, not_waddr[3], not_waddr[2], not_waddr[1],
waddr[0]);
    nand(D[2], not_we, not_waddr[3], not_waddr[2], waddr[1],
not_waddr[0]);
    nand(D[3], not_we, not_waddr[3], not_waddr[2], waddr[1], waddr[0]);
    nand(D[4], not_we, not_waddr[3], waddr[2], not_waddr[1],
not_waddr[0]);
    nand(D[5], not_we, not_waddr[3], waddr[2], not_waddr[1], waddr[0]);
    nand(D[6], not_we, not_waddr[3], waddr[2], waddr[1], not_waddr[0]);
    nand(D[7], not_we, not_waddr[3], waddr[2], waddr[1],waddr[0]);
    nand(D[8], not_we, waddr[3], not_waddr[2], not_waddr[1],
not_waddr[0]);
    nand(D[9], not_we, waddr[3], not_waddr[2], not_waddr[1], waddr[0]);
    nand(D[10], not_we, waddr[3], not_waddr[2], waddr[1], not_waddr[0]);
    nand(D[11], not_we, waddr[3], not_waddr[2], waddr[1], waddr[0]);
    nand(D[12], not_we, waddr[3], waddr[2], not_waddr[1], not_waddr[0]);
    nand(D[13], not_we, waddr[3], waddr[2], not_waddr[1], waddr[0]);
    nand(D[14], not_we, waddr[3], waddr[2], waddr[1], not_waddr[0]);
    nand(D[15], not_we, waddr[3], waddr[2], waddr[1], waddr[0]);
endmodule
```

4 位 16 选 1 数据选择器

```
`timescale 1ns / 1ps
module select(
    input [3:0] raddr,
    input [3:0] S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12,
S13, S14, S15,
```

```
    input not_we,
    output wire [3:0] rdata
    );

    wire [3:0] not_raddr;
    wire [15:0] bits0, bits1, bits2, bits3;
    not(not_raddr[0], raddr[0]);
    not(not_raddr[1], raddr[1]);
    not(not_raddr[2], raddr[2]);
    not(not_raddr[3], raddr[3]);

    and(bits0[0], not_we, S0[0], not_raddr[3],
not_raddr[2],not_raddr[1], not_raddr[0]);
    and(bits0[1], not_we, S1[0], not_raddr[3], not_raddr[2],
not_raddr[1], raddr[0]);
    and(bits0[2], not_we, S2[0], not_raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits0[3], not_we, S3[0], not_raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits0[4], not_we, S4[0], not_raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits0[5], not_we, S5[0], not_raddr[3], raddr[2], not_raddr[1],
raddr[0]);
    and(bits0[6], not_we, S6[0], not_raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits0[7], not_we, S7[0], not_raddr[3], raddr[2], raddr[1],
raddr[0]);
    and(bits0[8], not_we, S8[0], raddr[3], not_raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits0[9], not_we, S9[0], raddr[3], not_raddr[2], not_raddr[1],
raddr[0]);
    and(bits0[10], not_we, S10[0], raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits0[11], not_we, S11[0], raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits0[12], not_we, S12[0], raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits0[13], not_we, S13[0], raddr[3], raddr[2], not_raddr[1],
raddr[0]);
    and(bits0[14], not_we, S14[0], raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits0[15], not_we, S15[0], raddr[3], raddr[2], raddr[1],
raddr[0]);
```

```
    or(rdata[0], bits0[0], bits0[1], bits0[2], bits0[3], bits0[4],
bits0[5], bits0[6], bits0[7], bits0[8], bits0[9], bits0[10], bits0[11],
bits0[12], bits0[13], bits0[14], bits0[15]);

    and(bits1[0], not_we, S0[1], not_raddr[3], not_raddr[2],
not_raddr[1], not_raddr[0]);
    and(bits1[1], not_we, S1[1], not_raddr[3], not_raddr[2],
not_raddr[1], raddr[0]);
    and(bits1[2], not_we, S2[1], not_raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits1[3], not_we, S3[1], not_raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits1[4], not_we, S4[1], not_raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits1[5], not_we, S5[1], not_raddr[3], raddr[2], not_raddr[1],
raddr[0]);
    and(bits1[6], not_we, S6[1], not_raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits1[7], not_we, S7[1], not_raddr[3], raddr[2], raddr[1],
raddr[0]);
    and(bits1[8], not_we, S8[1], raddr[3], not_raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits1[9], not_we, S9[1], raddr[3], not_raddr[2], not_raddr[1],
raddr[0]);
    and(bits1[10], not_we, S10[1], raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits1[11], not_we, S11[1], raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits1[12], not_we, S12[1], raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits1[13], not_we, S13[1], raddr[3], raddr[2], not_raddr[1],
raddr[0]);
    and(bits1[14], not_we, S14[1], raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits1[15], not_we, S15[1], raddr[3], raddr[2], raddr[1],
raddr[0]);
    or(rdata[1], bits1[0], bits1[1], bits1[2], bits1[3], bits1[4],
bits1[5], bits1[6], bits1[7], bits1[8], bits1[9], bits1[10], bits1[11],
bits1[12], bits1[13], bits1[14], bits1[15]);

    and(bits2[0], not_we, S0[2], not_raddr[3], not_raddr[2],
not_raddr[1], not_raddr[0]);
    and(bits2[1], not_we, S1[2], not_raddr[3], not_raddr[2],
not_raddr[1], raddr[0]);
```

```
    and(bits2[2], not_we, S2[2], not_raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits2[3], not_we, S3[2], not_raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits2[4], not_we, S4[2], not_raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits2[5], not_we, S5[2], not_raddr[3], raddr[2], not_raddr[1],
raddr[0]);
    and(bits2[6], not_we, S6[2], not_raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits2[7], not_we, S7[2], not_raddr[3], raddr[2], raddr[1],
raddr[0]);
    and(bits2[8], not_we, S8[2], raddr[3], not_raddr[2],
not_raddr[1],not_raddr[0]);
    and(bits2[9], not_we, S9[2], raddr[3], not_raddr[2], not_raddr[1],
raddr[0]);
    and(bits2[10], not_we, S10[2], raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits2[11], not_we, S11[2], raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits2[12], not_we, S12[2], raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits2[13], not_we, S13[2], raddr[3], raddr[2], not_raddr[1],
raddr[0]);
    and(bits2[14], not_we, S14[2], raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits2[15], not_we, S15[2], raddr[3], raddr[2], raddr[1],
raddr[0]);
    or(rdata[2], bits2[0], bits2[1], bits2[2], bits2[3], bits2[4],
bits2[5], bits2[6], bits2[7], bits2[8], bits2[9], bits2[10], bits2[11],
bits2[12], bits2[13], bits2[14], bits2[15]);

    and(bits3[0], not_we, S0[3], not_raddr[3], not_raddr[2],
not_raddr[1], not_raddr[0]);
    and(bits3[1], not_we, S1[3], not_raddr[3], not_raddr[2],
not_raddr[1], raddr[0]);
    and(bits3[2], not_we, S2[3], not_raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits3[3], not_we, S3[3], not_raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits3[4], not_we, S4[3], not_raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits3[5], not_we, S5[3], not_raddr[3], raddr[2], not_raddr[1],
raddr[0]);
```

```
    and(bits3[6], not_we, S6[3], not_raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits3[7], not_we, S7[3], not_raddr[3], raddr[2], raddr[1],
raddr[0]);
    and(bits3[8], not_we, S8[3], raddr[3], not_raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits3[9], not_we, S9[3], raddr[3], not_raddr[2], not_raddr[1],
raddr[0]);
    and(bits3[10], not_we, S10[3], raddr[3], not_raddr[2], raddr[1],
not_raddr[0]);
    and(bits3[11], not_we, S11[3], raddr[3], not_raddr[2], raddr[1],
raddr[0]);
    and(bits3[12], not_we, S12[3], raddr[3], raddr[2], not_raddr[1],
not_raddr[0]);
    and(bits3[13], not_we, S13[3], raddr[3], raddr[2], not_raddr[1],
raddr[0]);
    and(bits3[14], not_we, S14[3], raddr[3], raddr[2], raddr[1],
not_raddr[0]);
    and(bits3[15], not_we, S15[3], raddr[3], raddr[2], raddr[1],
raddr[0]);
    or(rdata[3], bits3[0], bits3[1], bits3[2], bits3[3], bits3[4],
bits3[5], bits3[6], bits3[7], bits3[8], bits3[9], bits3[10], bits3[11],
bits3[12], bits3[13], bits3[14], bits3[15]);
endmodule
```

4 位寄存器:

```
`timescale 1ns / 1ps
module regfile(
    input clk, rst,
    input we,
    input [3:0] wdata,
    output wire [3:0] Q
    );
    reg [3:0] reg_data;
    initial
        reg_data = 4'b0000;
    always @(negedge clk) begin
        if (rst) begin
            reg_data <= 4'b0000;
        end
        else if(we == 0) begin
            reg_data = wdata;
        end
    end
assign Q = reg_data;
```

```
endmodule
```

寄存器堆:

```
`timescale 1ns / 1ps
module top(
    input clk,
    input rst,
    input we,
    input [3:0] raddr1,
    input [3:0] raddr2,
    input [3:0] waddr,
    input [3:0] wdata,
    output wire [3:0] rdata1,
    output wire [3:0] rdata2
    );
    wire [15:0] D;
    wire [3:0] S [15:0];
    // 例化 4-16 译码器
    decode decode(
        .we(we),
        .waddr(waddr),
        .D(D)
    );
    // 例化 16 个寄存器
    regfile
Reg0(.clk(clk), .rst(rst), .we(D[0]), .wdata(wdata), .Q(S[0]));
    regfile
Reg1(.clk(clk), .rst(rst), .we(D[1]), .wdata(wdata), .Q(S[1]));
    regfile
Reg2(.clk(clk), .rst(rst), .we(D[2]), .wdata(wdata), .Q(S[2]));
    regfile
Reg3(.clk(clk), .rst(rst), .we(D[3]), .wdata(wdata), .Q(S[3]));
    regfile
Reg4(.clk(clk), .rst(rst), .we(D[4]), .wdata(wdata), .Q(S[4]));
    regfile
Reg5(.clk(clk), .rst(rst), .we(D[5]), .wdata(wdata), .Q(S[5]));
    regfile
Reg6(.clk(clk), .rst(rst), .we(D[6]), .wdata(wdata), .Q(S[6]));
    regfile
Reg7(.clk(clk), .rst(rst), .we(D[7]), .wdata(wdata), .Q(S[7]));
    regfile
Reg8(.clk(clk), .rst(rst), .we(D[8]), .wdata(wdata), .Q(S[8]));
    regfile
Reg9(.clk(clk), .rst(rst), .we(D[9]), .wdata(wdata), .Q(S[9]));
```

```
    regfile
Reg10(.clk(clk), .rst(rst), .we(D[10]), .wdata(wdata), .Q(S[10]));
    regfile
Reg11(.clk(clk), .rst(rst), .we(D[11]), .wdata(wdata), .Q(S[11]));
    regfile
Reg12(.clk(clk), .rst(rst), .we(D[12]), .wdata(wdata), .Q(S[12]));
    regfile
Reg13(.clk(clk), .rst(rst), .we(D[13]), .wdata(wdata), .Q(S[13]));
    regfile
Reg14(.clk(clk), .rst(rst), .we(D[14]), .wdata(wdata), .Q(S[14]));
    regfile
Reg15(.clk(clk), .rst(rst), .we(D[15]), .wdata(wdata), .Q(S[15]));
    // 例化两个 4 位 16 选 1 数据选择器
    select select1(
        .raddr(raddr1),
        .S0(S[0]), .S1(S[1]), .S2(S[2]), .S3(S[3]), .S4(S[4]), .S5(S[5])
, .S6(S[6]), .S7(S[7]),
        .S8(S[8]), .S9(S[9]), .S10(S[10]), .S11(S[11]), .S12(S[12]), .S1
3(S[13]), .S14(S[14]), .S15(S[15]),
        .not_we(!we),
        .rdata(rdata1)
    );
    select select2(
        .raddr(raddr2),
        .S0(S[0]), .S1(S[1]), .S2(S[2]), .S3(S[3]), .S4(S[4]), .S5(S[5])
, .S6(S[6]), .S7(S[7]),
        .S8(S[8]), .S9(S[9]), .S10(S[10]), .S11(S[11]), .S12(S[12]), .S1
3(S[13]), .S14(S[14]), .S15(S[15]),
        .not_we(!we),
        .rdata(rdata2)
    );

endmodule
```

四、实验结果分析

1.这里应给出相应的实验结果。分析应有条理，要求采用规范的书面语。
2.每个实验都需要做模拟仿真，需要对仿真波形进行简单的文字说明。
3.对下载到开发板上的图片结果做分析说明。
4.原则上要求使用图片与文字结合的形式说明，因为 word 和 PDF 文档不支持视频，所以请不要使用视频文件。
5.图片请在垂直方向，不要横向。不要用很大的图片（不要超过 1M），请先做裁剪操作。

①激励文件:

```
`timescale 1ns / 1ps
module Regfiles_tb();
reg clk;
  reg rst;
  reg we;
  reg [3:0] raddr1;
  reg [3:0] raddr2;
  reg [3:0] waddr;
  reg [3:0] wdata;
  wire [3:0] rdata1;
  wire [3:0] rdata2;
  top regfiles (
    .clk(clk),
    .rst(rst),
    .we(we),
    .raddr1(raddr1),
    .raddr2(raddr2),
    .waddr(waddr),
    .wdata(wdata),
    .rdata1(rdata1),
    .rdata2(rdata2)
  );

  // 时钟信号生成
  always #5 clk = ~clk;

  // 初始化输入信号
  initial begin
    clk = 1'b0;
    rst = 1;
    we = 0;
    raddr1 = 0;
    raddr2 = 0;
    waddr = 0;
    wdata = 0;

    #20;
    rst = 0;

    // 写入数据
    we = 1;
    waddr = 2;
    wdata = 8;
    #10;
```

```
    we = 0;

    // 读取数据
    raddr1 = 2;
    raddr2 = 3;
    #10;

    // 更改地址
    raddr1 = 1;
    raddr2 = 2;
    #10;

    // 再次读取数据
    raddr1 = 1;
    raddr2 = 2;
    #10;

    // 写入数据
    we = 1;
    waddr = 1;
    wdata = 5;
    #10;
    we = 0;

    // 读取数据
    raddr1 = 1;
    raddr2 = 2;
    #10;

    // 异步复位
    rst = 1;
    #20;
    rst = 0;
    #10;

    // 读取数据
    raddr1 = 1;
    raddr2 = 2;
    #10;

    $finish;
  end
endmodule
```

②约束文件:

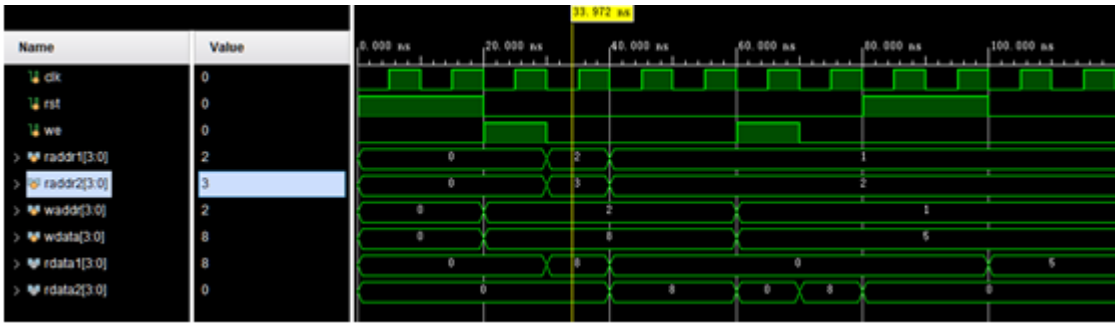
```
# ////////////////////////////系统时钟和复位
//////////////////////////////////
set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports
{clk} ]
set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports
{rst} ]
# ////////////////////////////////拨码开关
sw0~sw7//////////////////////////////
# 读地址 1
set_property -dict {PACKAGE_PIN P5 IOSTANDARD LVCMOS33} [get_ports
{raddr1[0]}]
set_property -dict {PACKAGE_PIN P4 IOSTANDARD LVCMOS33} [get_ports
{raddr1[1]}]
set_property -dict {PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports
{raddr1[2]}]
set_property -dict {PACKAGE_PIN P2 IOSTANDARD LVCMOS33} [get_ports
{raddr1[3]}]
# 读地址 2
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports
{raddr2[0]}]
set_property -dict {PACKAGE_PIN M4 IOSTANDARD LVCMOS33} [get_ports
{raddr2[1]}]
set_property -dict {PACKAGE_PIN N4 IOSTANDARD LVCMOS33} [get_ports
{raddr2[2]}]
set_property -dict {PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports
{raddr2[3]}]

# ////////////////////////////////拨码开关
sw8~sw15//////////////////# ////////////////////////////// 系统时钟和复位
//////////////////////////////////
set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports
{clk}]
# set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports
{rst}]

# ////////////////////////////////// 5 个按键
//////////////////////////////////////
set_property -dict {PACKAGE_PIN R11 IOSTANDARD LVCMOS33} [get_ports
{we}]
# set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports
{btn_pin[1]}]
# set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports
{btn_pin[2]}]
```

```
set_property -dict {PACKAGE_PIN V1  IOSTANDARD LVCMOS33} [get_ports
{rst}]
# set_property -dict {PACKAGE_PIN U4  IOSTANDARD LVCMOS33} [get_ports
{btn_pin[4]}]

# ////////////////////////////// 拨码开关 sw0~sw7
//////////////////////////
set_property -dict {PACKAGE_PIN P5 IOSTANDARD LVCMOS33} [get_ports
{raddr1[0]}]
set_property -dict {PACKAGE_PIN P4 IOSTANDARD LVCMOS33} [get_ports
{raddr1[1]}]
set_property -dict {PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports
{raddr1[2]}]
set_property -dict {PACKAGE_PIN P2 IOSTANDARD LVCMOS33} [get_ports
{raddr1[3]}]
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports
{raddr2[0]}]
set_property -dict {PACKAGE_PIN M4 IOSTANDARD LVCMOS33} [get_ports
{raddr2[1]}]
set_property -dict {PACKAGE_PIN N4 IOSTANDARD LVCMOS33} [get_ports
{raddr2[2]}]
set_property -dict {PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports
{raddr2[3]}]

# ////////////////////////////// 拨码开关 sw8~sw15
//////////////////////////
set_property -dict {PACKAGE_PIN U3 IOSTANDARD LVCMOS33} [get_ports
{waddr[0]}]
set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports
{waddr[1]}]
set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports
{waddr[2]}]
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports
{waddr[3]}]
set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports
{wdata[0]}]
set_property -dict {PACKAGE_PIN R3 IOSTANDARD LVCMOS33} [get_ports
{wdata[1]}]
set_property -dict {PACKAGE_PIN T3 IOSTANDARD LVCMOS33} [get_ports
{wdata[2]}]
set_property -dict {PACKAGE_PIN T5 IOSTANDARD LVCMOS33} [get_ports
{wdata[3]}]
```

```
# ////////////////////////////// LED0~LED15
//////////////////////////////
set_property -dict {PACKAGE_PIN F6 IOSTANDARD LVCMOS33} [get_ports
{rdata1[0]}]
set_property -dict {PACKAGE_PIN G4 IOSTANDARD LVCMOS33} [get_ports
{rdata1[1]}]
set_property -dict {PACKAGE_PIN G3 IOSTANDARD LVCMOS33} [get_ports
{rdata1[2]}]
set_property -dict {PACKAGE_PIN J4 IOSTANDARD LVCMOS33} [get_ports
{rdata1[3]}]
set_property -dict {PACKAGE_PIN H4 IOSTANDARD LVCMOS33} [get_ports
{rdata2[0]}]
set_property -dict {PACKAGE_PIN J3 IOSTANDARD LVCMOS33} [get_ports
{rdata2[1]}]
set_property -dict {PACKAGE_PIN J2 IOSTANDARD LVCMOS33} [get_ports
{rdata2[2]}]
set_property -dict {PACKAGE_PIN K2 IOSTANDARD LVCMOS33} [get_ports
{rdata2[3]}]

# set_property -dict {PACKAGE_PIN K1 IOSTANDARD LVCMOS33} [get_ports
{led_pin[8]}]
# set_property -dict {PACKAGE_PIN H6 IOSTANDARD LVCMOS33} [get_ports
{led_pin[9]}]
# set_property -dict {PACKAGE_PIN H5 IOSTANDARD LVCMOS33} [get_ports
{led_pin[10]}]
# set_property -dict {PACKAGE_PIN J5 IOSTANDARD LVCMOS33} [get_ports
{led_pin[11]}]
# set_property -dict {PACKAGE_PIN K6 IOSTANDARD LVCMOS33} [get_ports
{led_pin[12]}]
# set_property -dict {PACKAGE_PIN L1 IOSTANDARD LVCMOS33} [get_ports
{led_pin[13]}]
# set_property -dict {PACKAGE_PIN M1 IOSTANDARD LVCMOS33} [get_ports
{led_pin[14]}]
# set_property -dict {PACKAGE_PIN K3 IOSTANDARD LVCMOS33} [get_ports
{led_pin[15]}]
//////////
# 写地址1
set_property -dict {PACKAGE_PIN U3 IOSTANDARD LVCMOS33} [get_ports
{waddr[0]}]
set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports
{waddr[1]}]
set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports
{waddr[2]}]
```

```
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports
{waddr[3]}]
# 写数据
set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports
{wdata[0]}]
set_property -dict {PACKAGE_PIN R3 IOSTANDARD LVCMOS33} [get_ports
{wdata[1]}]
set_property -dict {PACKAGE_PIN T3 IOSTANDARD LVCMOS33} [get_ports
{wdata[2]}]
set_property -dict {PACKAGE_PIN T5 IOSTANDARD LVCMOS33} [get_ports
{wdata[3]}]
#
////////////////////////////////////LED0~LED17//////////////////////
////////
# 读数据 1
set_property -dict {PACKAGE_PIN F6 IOSTANDARD LVCMOS33} [get_ports
{rdata1[0]}]
set_property -dict {PACKAGE_PIN G4 IOSTANDARD LVCMOS33} [get_ports
{rdata1[1]}]
set_property -dict {PACKAGE_PIN G3 IOSTANDARD LVCMOS33} [get_ports
{rdata1[2]}]
set_property -dict {PACKAGE_PIN J4 IOSTANDARD LVCMOS33} [get_ports
{rdata1[3]}]
# 读数据 2
set_property -dict {PACKAGE_PIN H4 IOSTANDARD LVCMOS33} [get_ports
{rdata2[0]}]
set_property -dict {PACKAGE_PIN J3 IOSTANDARD LVCMOS33} [get_ports
{rdata2[1]}]
set_property -dict {PACKAGE_PIN J2 IOSTANDARD LVCMOS33} [get_ports
{rdata2[2]}]
set_property -dict {PACKAGE_PIN K2 IOSTANDARD LVCMOS33} [get_ports
{rdata2[3]}]
# ////////////////////////////////////s1 按键
////////////////////////////////
set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports
{we}]
```

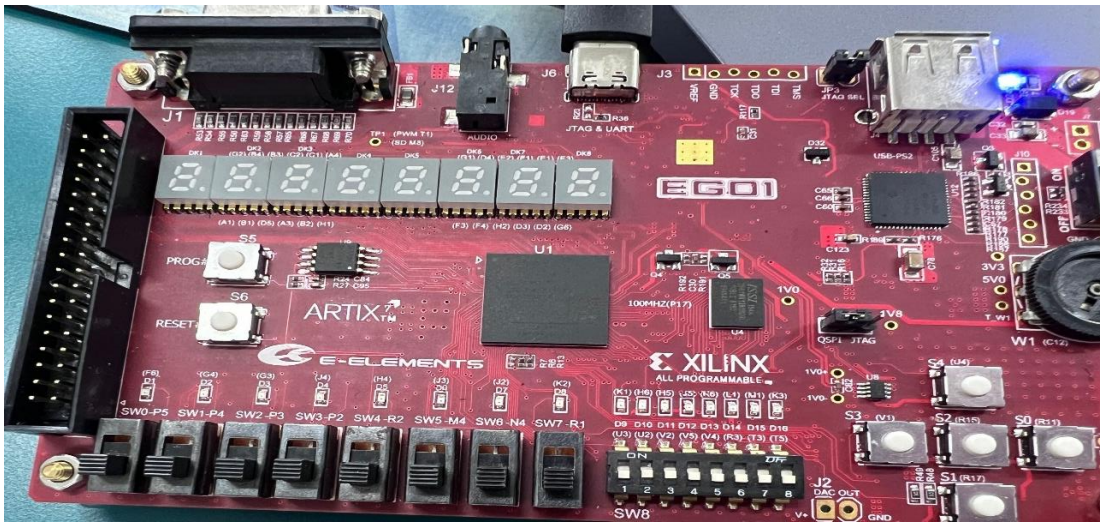
③仿真波形:



④结果解释:

在时钟的第一个下降沿和第二个下降沿时，rst 信号为高电平，此时对寄存器进行复位，并且接下来，rst 为低电平，不再执行复位，在下一个时钟下降沿时，向寄存器 2 写入数据 8，并且从寄存器 2 和 3 中读取出数据，分别为 8（刚刚写入的 8）和 0（未写入，仍为初始状态 0）；在第六个时钟下降沿时，向寄存器 1 写入数据 5，可以看到后续从寄存器 1 中读出的数据为 5。

⑤开发板运行图片:



五、遇到的问题与实验心得体会

简要地叙述一下实验过程中的感受，以及其他的问题描述和自己的感想。特别是实验中遇到的困难，最后如何解决的。在用 verilog 代码写程序时遇到语法或其他错误，如何修改解决的。

在本次的计算机组成实验中设计和实现双端口输出的寄存器堆是一项有挑战性的任务。我在实验过程中做了以下工作：设计了由 16 个 4 位寄存器构成的寄存器堆，并使用结构化的方法进行描述和实现。顶层模块通过子模块的实例化来完成整个设计和实现的过程。

一开始没有注意到需要使用结构化的方法进行描述和实现，直接使用行为级描述实现了其中的译码器和选择器模块，并没有花费多少时间。但是后续细看了实验指导书，需要使用结构化的方法进行描述和实现，需要我们从门级电路开始搭建整个译码器和选择器模块。相比之下，使用结构化的方法实现就要比行为级描述麻烦的多，我们需要根据电路原理图，来逐个例化每一个逻辑门，并且在代码量上也多了不少，但是使用结构化的方法实现，可以让我更好的了解原理。对于整个寄存器堆，使用模块化的方式，形成一个顶层的寄存器堆模块，一开始还对于这个方法非常疑惑，但是在真正实现了整个寄存器堆后，发现这样的实现方法可以简化实现，并且在实现其他模块时候，如果同样需要译码器或选择器，可以直接使用已实现的模块，非常方便。

通过这次实验，我学到了很多关于寄存器堆设计和 Verilog 编程的知识和技能。我深刻认识到了模块化设计和结构化描述的重要性，以及在面对困难时保持耐心和解决问题的能力。这个实验让我更加熟悉了计算机组成的相关概念和原理，并且提高了我的 Verilog 编程能力。这将对我今后的学习和工作有很大的帮助。

实验八　单周期的 CPU 设计

一、实验目的及环境

1、深入了解单周期 CPU 中指令控制器的结构和工作原理。

2、学习使用 Verilog HDL 设计实现单周期 CPU 中指令控制器。

3、装有 vivado 的计算机　　1 台

4、EGO1 开发板　　1 块

二、实验目标及任务

1、设计和实现单周期 CPU 中指令控制器的结构并且进行功能仿真。

三、实验过程及记录

本节重点介绍实验的具体过程，包括：代码设计层次结构图及说明、源代码（包括注释）、PC 机上进行的关键步骤截图及说明、调试过程等，这部分的内容应当与实际操作过程类似即可（简单明了）。

①电路原理图:

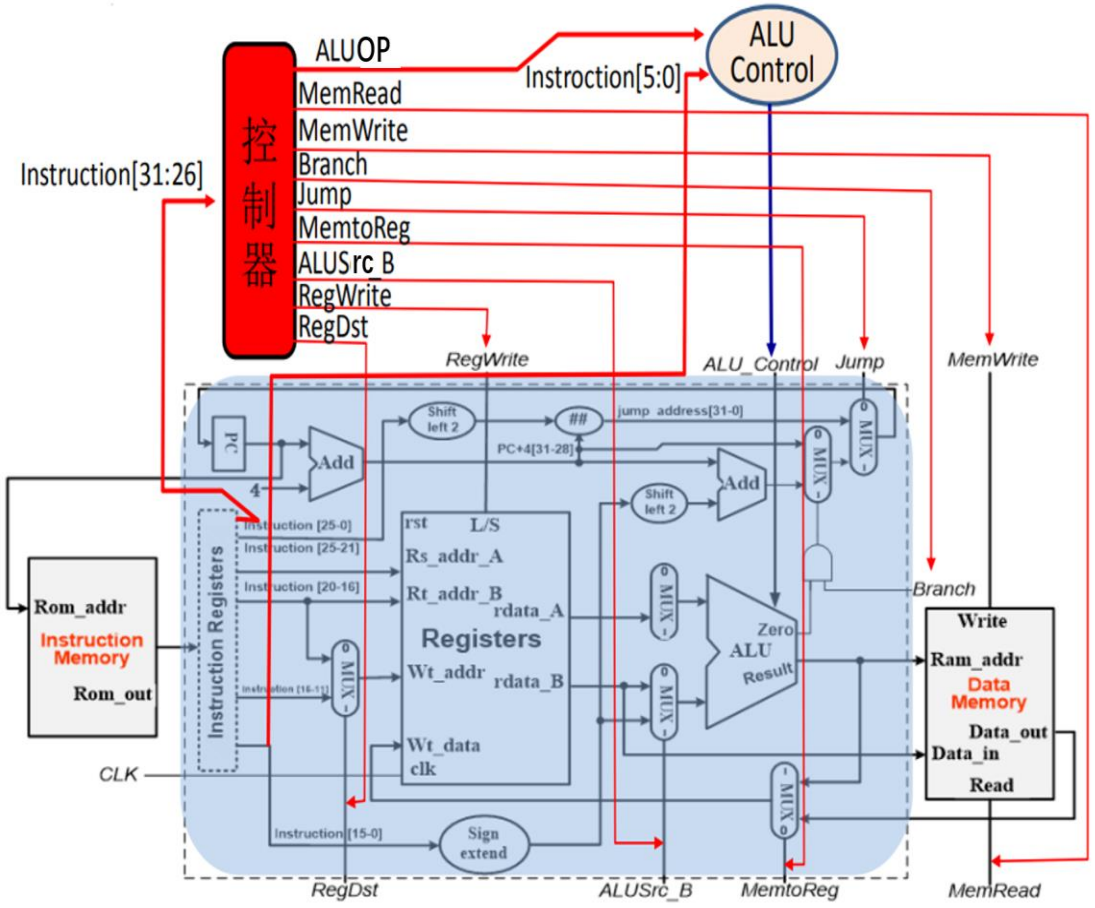
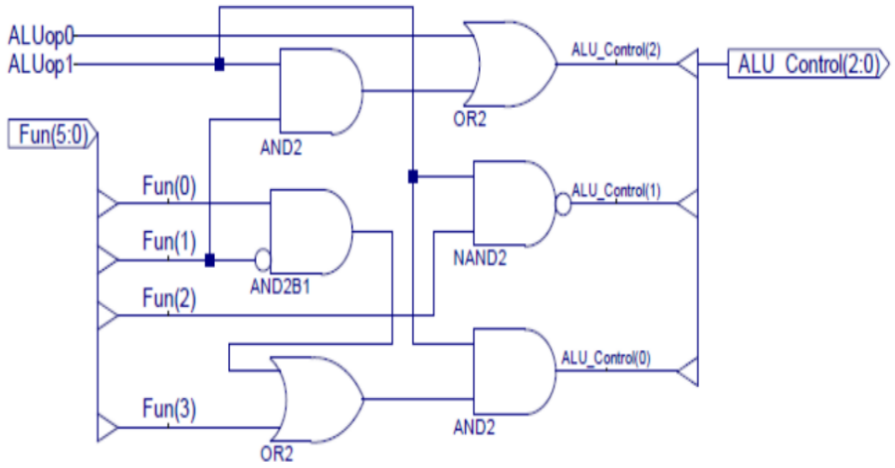


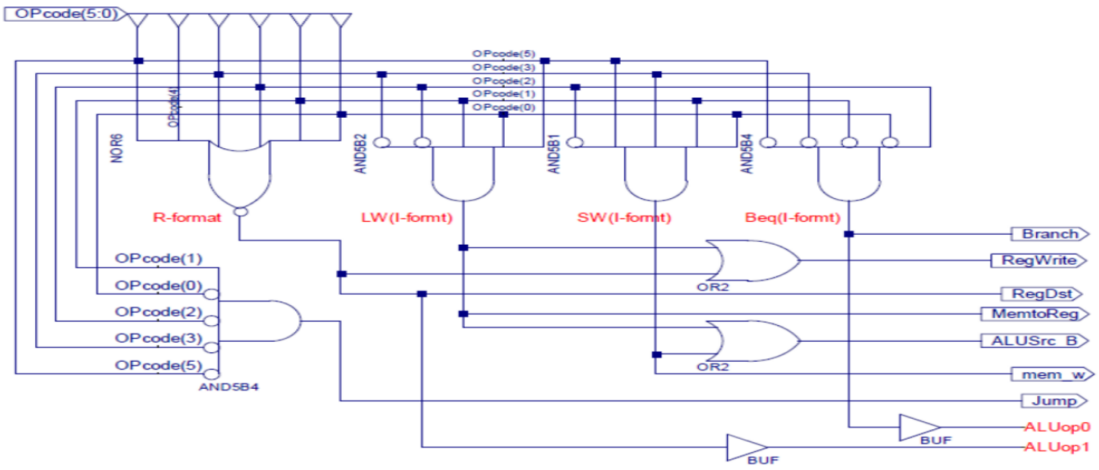
表 8.1 控制器输出信号功能定义表

信号	源数目	功能定义	赋值 0 时动作	赋值 1 时动作
ALUSrc_B	2	ALU 端口 B 输入选择	选择寄存器 B 数据	选择 32 位立即数(符号扩展后)
RegDst	2	寄存器写地址选择	选择指令 rt 域	选择指令 rs 域
MemtoReg	2	寄存器写入数据选择	选择存储器数据	选择 ALU 输出
Branch	2	Beq 指令目标地址选择	选择 PC+4 地址	选择转移地址(Zero=1)
Jump	2	J 指令目标地址选择	选择 J 目标地址	由 Branch 决定输出
RegWrite	-	寄存器写控制	禁止寄存器写	使能寄存器写
MemWrite	-	存储器写控制	禁止存储器写	使能存储器写
MemRead	-	存储器读控制	禁止存储器读	使能存储器读
ALU_Control	000-111	3 位 ALU 操作控制		

由实验原理图以及上述控制输出的表格可以得知，我们首先需要实现 ALU_Control 模块，该模块通过其中的控制信号 ALUOP 与指令机器码的低 6 位 instruction[5:0]联合起来，生成运算器运算类型相对应的控制信号 ALU_Control。



根据上面的实验原理图我们可以实现 ALU_Control 模块。之后我们需要设计 CPUControl 模块实现 Cpu 的控制。



由上述两个电路原理图可以很容易知道 CpuControl 模块的编写。最后通过顶层 Top 文件完成 CPU 的设计。

②源代码:

ALU_Control:

```
`timescale 1ns / 1ps

// ALU 控制模块
module ALU_Ctrl (
    input      [1:0] ALUOp,
    input      [5:0] Func,
    output reg [2:0] ALU_Control
);
    and and1 (temp1, ALUOp[1], Func[1]);
    and and2 (temp2, Func[0], ~Func[1]);
    or or1 (temp3, Func[3], temp2);
    or or2 (temp4, ALUOp[0], temp1);
    nand nand1 (temp5, ALUOp[1], Func[2]);
    and and3 (temp6, ALUOp[1], temp3);

    // ALU 控制信号输出
    always @(*) begin
        ALU_Control <= {temp4, temp5, temp6};
    end
endmodule
```

SCPU_Control:

```
`timescale 1ns / 1ps

// CPU 控制模块
module SCPU_Ctrl (
```

```
    input      [5:0] OPcode,       // OPcode
    input      [5:0] Func,         // Function
    output reg       RegDst,
    output reg       ALUSrc_B,
    output reg       MemtoReg,
    output reg       Jump,
    output reg       Branch,
    output reg       RegWrite,
    output reg [2:0] ALU_Control,
    output reg       MemRead,
    output reg       MemWrite
);
    wire [2:0] ALU_Control_t;

    // 控制信号生成
    nor nor6 (RFormat, OPcode[5], OPcode[4], OPcode[3], OPcode[2],
OPcode[1], OPcode[0]);
    and and5b1 (LFormat_LW, OPcode[5], ~OPcode[3], ~OPcode[2],
OPcode[1], OPcode[0]);
    and and5b2 (LFormat_SW, OPcode[5], OPcode[3], ~OPcode[2], OPcode[1],
OPcode[0]);
    and and5b3 (LFormat_Beq, ~OPcode[5], ~OPcode[3], OPcode[2],
~OPcode[1], ~OPcode[0]);
    and and5b4 (JFormat, ~OPcode[5], ~OPcode[3], ~OPcode[2], OPcode[1],
~OPcode[0]);
    // ALU 控制
    ALU_Ctrl u_ALU_Ctrl (
        .ALUOp      ({RFormat, LFormat_Beq}),
        .Func       (Func),
        .ALU_Control(ALU_Control_t)
    );

    // 控制信号输出
    always @(*) begin
        RegDst <= RFormat;
        ALUSrc_B <= LFormat_LW | LFormat_SW;
        MemtoReg <= LFormat_LW;
        Jump <= JFormat;
        Branch <= LFormat_Beq;
        RegWrite <= RFormat | LFormat_LW;
        MemRead <= LFormat_LW;
        MemWrite <= LFormat_SW;
        ALU_Control <= ALU_Control_t;
    end
```

```
    // 初始化
    initial begin
        RegDst <= 0;
        ALUSrc_B <= 0;
        MemtoReg <= 0;
        Jump <= 0;
        Branch <= 0;
        RegWrite <= 0;
        ALU_Control <= 0;
        MemRead <= 0;
        MemWrite <= 0;
    end

endmodule
```

Top 顶层模块:

```
`timescale 1ns / 1ps

module top (
    input  [15:0] Instrument_Select,
    output        RegDst,
    output        ALUSrc_B,
    output        MemtoReg,
    output        Jump,
    output        Branch,
    output        RegWrite,
    output [ 2:0] ALU_Control,
    output        MemRead,
    output        MemWrite
);
    // mem 用于读取指令
    reg [11:0] memory [0:1];

    // SCPU_Ctrl 模块输入
    reg [ 5:0] OPcode;
    reg [ 5:0] Func;

    // SCPU_Ctrl 模块实例化
    SCPU_Ctrl u_SCPU_Ctrl (
        .OPcode     (OPcode),
        .Func       (Func),
        .RegDst     (RegDst),
        .ALUSrc_B   (ALUSrc_B),
        .MemtoReg   (MemtoReg),
```

```
        .Jump       (Jump),
        .Branch     (Branch),
        .RegWrite   (RegWrite),
        .ALU_Control(ALU_Control),
        .MemRead    (MemRead),
        .MemWrite   (MemWrite)
    );

    // 读取指令到 memory
    initial begin
        $readmemb("Istructions.txt", memory);
    end

    // 指定执行第几条指令
    reg [4:0] n = 0;
    always @(Instrument_Select) begin
        case (Instrument_Select)
            1 << 0: n <= 0;
            1 << 1: n <= 1;
        endcase
        OPcode <= memory[n][11:6];
        Func   <= memory[n][5:0];
    end
endmodule
```

四、实验结果分析

1.这里应给出相应的实验结果。分析应有条理，要求采用规范的书面语。
2.每个实验都需要做模拟仿真，需要对仿真波形进行简单的文字说明。
3.对下载到开发板上的图片结果做分析说明。
4.原则上要求使用图片与文字结合的形式说明，因为 word 和 PDF 文档不支持视频，所以请不要使用视频文件。
5.图片请在垂直方向，不要横向。不要用很大的图片（不要超过 1M），请先做裁剪操作。

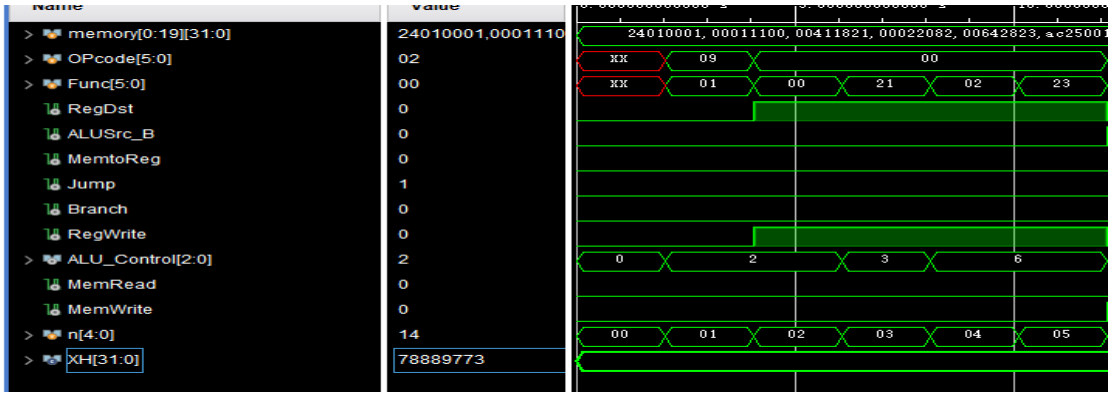
①激励文件:

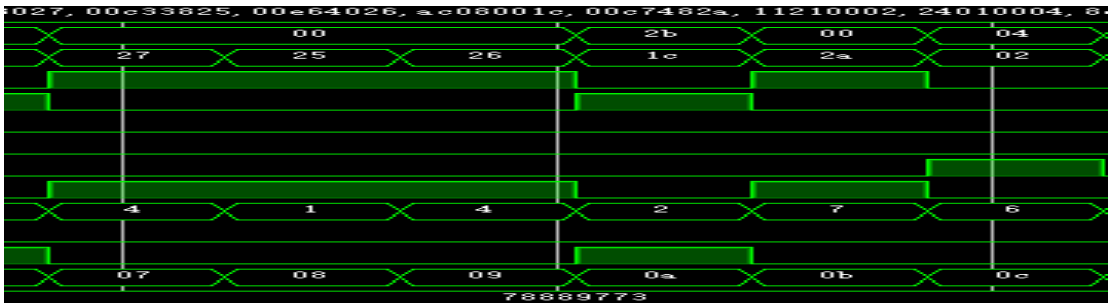
```
`timescale 1ns / 1ps

module tb_top;
    parameter XH = 2022217587;

    // top 模块输入
    reg  [15:0] Instrument_Select;

    // top 模块输出
```

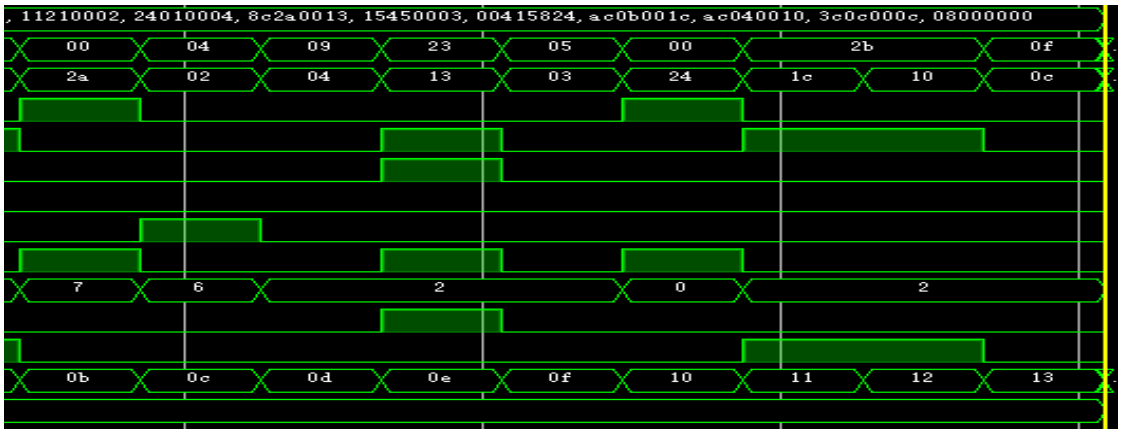
```
    wire        RegDst;
    wire        ALUSrc_B;
    wire        MemtoReg;
    wire        Jump;
    wire        Branch;
    wire        RegWrite;
    wire [ 2:0] ALU_Control;
    wire        MemRead;
    wire        MemWrite;

    // top 模块实例化
    top u_top (
        .Instrument_Select(Instrument_Select),
        .RegDst           (RegDst),
        .ALUSrc_B         (ALUSrc_B),
        .MemtoReg         (MemtoReg),
        .Jump             (Jump),
        .Branch           (Branch),
        .RegWrite         (RegWrite),
        .ALU_Control      (ALU_Control),
        .MemRead          (MemRead),
        .MemWrite         (MemWrite)
    );

    // 依次送入第 1 ~ 16 条指令
    initial begin
        Instrument_Select <= 16'b0000000000000001;
        #XH;

        Instrument_Select <= 16'b0000000000000010;
        #XH;

        Instrument_Select <= 16'b0000000000000100;
        #XH;

        Instrument_Select <= 16'b0000000000001000;
        #XH;

        Instrument_Select <= 16'b0000000000010000;
        #XH;

        Instrument_Select <= 16'b0000000000100000;
        #XH;
```

```
        Instrument_Select <= 16'b0000000001000000;
        #XH;

        Instrument_Select <= 16'b0000000010000000;
        #XH;

        Instrument_Select <= 16'b0000000100000000;
        #XH;

        Instrument_Select <= 16'b0000001000000000;
        #XH;

        Instrument_Select <= 16'b0000010000000000;
        #XH;

        Instrument_Select <= 16'b0000100000000000;
        #XH;

        Instrument_Select <= 16'b0001000000000000;
        #XH;

        Instrument_Select <= 16'b0010000000000000;
        #XH;

        Instrument_Select <= 16'b0100000000000000;
        #XH;

        Instrument_Select <= 16'b1000000000000000;
        #XH;
    end
endmodule
```

②仿真波形:







③结果解释:

在这个激励文件的仿真波形中，使用了学号作为 XH 变量，依次送入指令内容，ALU_Control 的输出结果与下面的表格输出内容一致。

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:·6]	[5:0]	功能
R 型指令							
Add	000000	rs	rt	rd	000000	100000	寄存器加
Sub	000000	rs	rt	rd	000000	100010	寄存器减
And	000000	rs	rt	rd	000000	100100	寄存器与
Or	000000	rs	rt	rd	000000	100101	寄存器或
Xor	000000	rs	rt	rd	000000	100110	寄存器异或
Sll	000000	00000	rt	rd	sa	000000	左移
Srl	000000	00000	rt	rd	sa	000010	逻辑右移
Sra	000000	00000	rt	rd	sa	000011	算术右移
Jr	000000	rs	rt	rd	000000	001000	寄存器跳

④Instructions 文件:

000000100000

000000100010

000000100100

000000100101

000000100110

000000000000

000000000010

000000000011

000000001000

该文件中的部分使用上述图片中的指令形式，通过 Top 文件进行依次读取并在开发板中进行显示。

⑤约束文件:

```
# ////////////////////////////系统时钟和复位
////////////////////////////////
# set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports
sys_clk_in ]
# set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports
sys_rst_n  ]

# ////////////////////////////////////5 个按键
////////////////////////////////////
# set_property -dict {PACKAGE_PIN R11 IOSTANDARD LVCMOS33} [get_ports
{btn_pin[0]}]
# set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports
{btn_pin[1]}]
# set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports
{btn_pin[2]}]
# set_property -dict {PACKAGE_PIN V1  IOSTANDARD LVCMOS33} [get_ports
{btn_pin[3]}]
# set_property -dict {PACKAGE_PIN U4  IOSTANDARD LVCMOS33} [get_ports
{btn_pin[4]}]

# ////////////////////////////////////拨码开关
sw0~sw7////////////////////////////////
set_property -dict {PACKAGE_PIN P5 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[15]}]
set_property -dict {PACKAGE_PIN P4 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[14]}]
set_property -dict {PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[13]}]
set_property -dict {PACKAGE_PIN P2 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[12]}]
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[11]}]
set_property -dict {PACKAGE_PIN M4 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[10]}]
```

```
set_property -dict {PACKAGE_PIN N4 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[9]}]
set_property -dict {PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[8]}]

# ////////////////////////////////拨码开关
sw8~sw15/////////////////////////////
set_property -dict {PACKAGE_PIN U3 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[7]}]
set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[6]}]
set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[5]}]
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[4]}]
set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[3]}]
set_property -dict {PACKAGE_PIN R3 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[2]}]
set_property -dict {PACKAGE_PIN T3 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[1]}]
set_property -dict {PACKAGE_PIN T5 IOSTANDARD LVCMOS33} [get_ports
{Instrument_Select[0]}]


#
////////////////////////////////////LED0~LED15/////////////////////
////////
set_property -dict {PACKAGE_PIN F6 IOSTANDARD LVCMOS33} [get_ports
{RegDst}]
set_property -dict {PACKAGE_PIN G4 IOSTANDARD LVCMOS33} [get_ports
{ALUSrc_B}]
set_property -dict {PACKAGE_PIN G3 IOSTANDARD LVCMOS33} [get_ports
{MemtoReg}]
set_property -dict {PACKAGE_PIN J4 IOSTANDARD LVCMOS33} [get_ports
{Jump}]
set_property -dict {PACKAGE_PIN H4 IOSTANDARD LVCMOS33} [get_ports
{Branch}]
set_property -dict {PACKAGE_PIN J3 IOSTANDARD LVCMOS33} [get_ports
{RegWrite}]
set_property -dict {PACKAGE_PIN J2 IOSTANDARD LVCMOS33} [get_ports
{MemRead}]
set_property -dict {PACKAGE_PIN K2 IOSTANDARD LVCMOS33} [get_ports
{MemWrite}]
```

```
set_property -dict {PACKAGE_PIN K1 IOSTANDARD LVCMOS33} [get_ports
{ALU_Control[2]}]
set_property -dict {PACKAGE_PIN H6 IOSTANDARD LVCMOS33} [get_ports
{ALU_Control[1]}]
set_property -dict {PACKAGE_PIN H5 IOSTANDARD LVCMOS33} [get_ports
{ALU_Control[0]}]
```

⑥开发板运行结果:

根据图片不难发现实验开发板与实验所编写代码运行结果一致。

五、遇到的问题与实验心得体会

简要地叙述一下实验过程中的感受，以及其他的问题描述和自己的感想。特别是实验中遇到的困难，最后如何解决的。在用 verilog 代码写程序时遇到语法或其他错误，如何修改解决的。

在计算机组成实验中设计单周期 CPU 顶层模块是一项重要的任务。在本次实验中，我完成了这两个工作。

本次实验的第一步，我实现了 ALU-Control 模块，这个模块负责根据指令的操作码输出对应的 ALU 操作控制信号，决定 ALU 将执行的具体算术或逻辑操作。这一部分是整个 CPU 中极其重要的，因为它直接影响到 ALU 的行为和最终的运算结果。之后我开始编写 CPU 模块的主体部分，这包括了指令寄存器的设计，它用于存储当前正在执行的指令。接着是寄存器堆的实现，它存储了运算所需的操作数和运算结果。译码和控制单元也是关键，译码单元负责解析指令寄存器中的内容，而控制单元则生成相应的控制信号来驱动整个 CPU 的操作，包括指令的读取、执行和数据的写回。在 CPU 的实际构建中，我遵循了指令的基本执行过程：取指、析指、执行、访存和写回。这个过程涉及从内存中读取指令、解析指令确定操作

和操作数、通过 ALU 执行运算、访问内存处理数据存取以及将结果写回到寄存器。整个过程中，我更加深入地理解了 CPU 的结构和指令的执行流程，这对于最后的实现是极为有助的。最后，将这些独立的模块通过 top 文件依次实例化，构建了一个能够执行基础指令集的单周期 CPU。通过这个实验，我不仅提升了对 CPU 内部工作原理的理解，还锻炼了我的硬件设计和调试技巧。在编写过程中也遇到了一些问题，在进行上板子的过程中，无法正常读取指令，观察 top 文件的编写后发现对于指令的读取的数量设置有误，导致在板子上不行正确的显示内容，在不断测试和优化设计的过程中，我确保了 CPU 的正确性和效率，这是一次富有成效的学习经历。

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字： 嵌入签名图片 

二、对课程实验的学术评语（教师填写）

三、对课程实验的评分（教师填写）

评分项目（分值）	报告撰写（50分）	课设验收（50分）	最终评定（100分）
得分			

指导教师签字： _____

物联网 1 班和 2 班提交实验 5-8 实验报告的链接：

https://send2me.cn/IuId3Ca1/QGSLhXdskkWmnQ

计算机 1 班、2 班和 3 班提交实验 5-8 实验报告的链接：

https://send2me.cn/zpy-PLTV/QqCUMzwR-3_EHg

计算机 4 班和 5 班提交实验 5-8 实验报告的链接：

https://send2me.cn/onBsL6Gc/QsWLLqvdK_3LqQ