浙江大学

本科实验报告

课程名称：	计算机组成与设计
姓　　名：	仇国智
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
指导教师：	刘海风
报告日期：	2024 年 2 月 28 日

浙江大学实验报告

课程名称：计算机组成与设计 实验类型：综合

实验项目名称：vivado 介绍和模块封装

学生姓名：仇国智 学号：3220102181 同组学生姓名：

实验地点：紫金港东四 509 室 实验日期：2024 年 2 月 28 日

一、操作方法与实验步骤

1 安装并使用 Vivado

1.1 项目创建：启动 Vivado 之后，选择顶部快捷栏中的 File -> Project -> New，在 Project Name 界面中设置项目名字为 project1，并设置项目路径。在 Project Type 界面，选择 RTL Project，把 Do not specify at this time 勾上（表示在新建工程时不去指定源文件）。在 Default Part 界面，进入 Boards 并在其中找到并选择 Nexys A7-100T。点击 Finish 完成工程创建。

1.2 源文件编写：工程创建后，在 Flow Navigator 界面（工作流窗口）下点击 Add Sources，选择 Add or create design sources 添加源文件。点击 Next 后，Create File 创建文件，并选择文件类型为 Verilog，输入文件名，文件保存路径选择 <Local to Project>，点击 OK，来创建源文件 top.v 和 MUX4b4to1.v。用 VScode 编写这两个源文件，结果如下

top.v

```
`timescale 1ns / 1ps
module top (
    input  wire [15:0] SW,
    output wire [ 3:0] LED
);
  MUX4b4to1 d_1 (
      .choose({SW[15], SW[14]}),
      .I({4'b0, SW[11:0]}),
      .O(LED)
  );
endmodule
```

MUX4b4to1.v

```
module MUX4b4to1 (
```

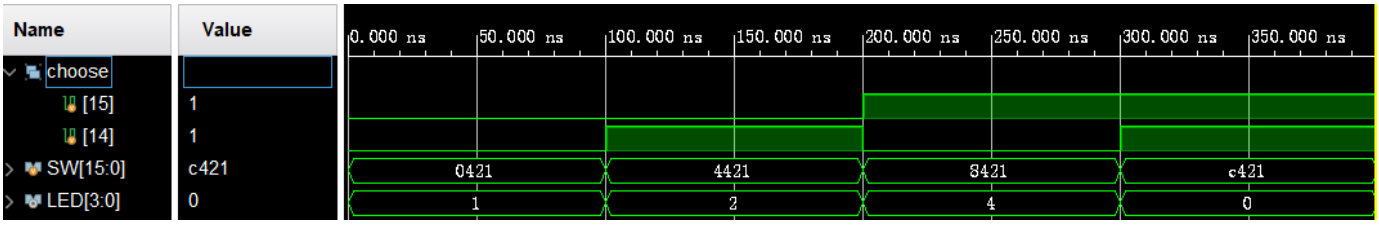
```
    input  wire [ 1:0] choose,
    input  wire [15:0] I,
    output wire [ 3:0] O
);
  assign O =
(~choose[1]&~choose[0])?I[3:0]:(~choose[1]&choose[0])?I[7:4]:(ch
oose[1]&~choose[0])?I[11:8]:I[15:12];
endmodule
```

1.3 功能仿真：工程创建后，在 Flow Navigator 界面（工作流窗口）下点击 Add Sources，选择 Add or create simulation sources 添加仿真文件文件。点击 Next 后，Create File 创建文件，并选择文件类型为 Verilog，输入文件名，文件保存路径选择 <Local to Project>，点击 OK，来创建仿真文件 top_tb.v。用 VScode 输入如下仿真代码。

top_tb.v

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

    );
    reg[15:0] SW;
    wire[3:0] LED;
    top tb1 (
        .SW(SW),
        .LED(LED)
    );
    initial begin
        SW = 16'b0000_0100_0010_0001;
        #100;
        SW[15:14]=2'b01;
        #100;
        SW[15:14]=2'b10;
        #100;
        SW[15:14]=2'b11;
        #100;
        $finish;
    end
endmodule
```

得到仿真波形如下：



可以看到我们进行如下设置 SW[3:0]:4'b0001,SW[7:4]:4'b0010,SW[11:8]:4'b0100, 当 SW[15:14]=0 时输出 1=SW[3:0]；当 SW[15:14]=1 时输出 2=SW[7:4]；当 SW[15:14]=2 时输出 4=SW[11:8]；当 SW[15:14]=3 时输出常数 0。符合设计需求

1.4 引脚约束：点击左边的工作流窗口中 Run Synthesis，进行仿真然后打开综合设计报告，接下来编写约束文件。

仿照上面创建引脚约束文件 constraints.xdc,进行编写

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
[get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
[get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
[get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 }
[get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 }
[get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 }
[get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 }
[get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 }
[get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 }
[get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 }
[get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 }
[get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
```

```
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
```

在 I/O Ports 中查看引脚约束。

1.5 生成 bit 文件烧录：修改约束文件后，点击工作流窗口的 PROGRAM AND DEBUG > Generate Bitstream 生成比特流。得到 bitstream 后，需要将下载器连接到电脑上，点击 PROGRAM AND DEBUG > Open Hardware Manager > Open Target > Auto Connect 进行识别和连接，成功连接后，点击 Program Device 选择 xc7a100t 设备，在下载程序界面选择刚刚生成的比特流文件，将其下载到板上。具体下板图像分析见第二节第一点。

2 简单模块设计（ALU / RegFile / 有限状态机）

2.1 项目创建：启动 Vivado 之后，选择顶部快捷栏中的 File -> Project -> New，在 Project Name 界面中设置项目名字为 project2，并设置项目路径。在 Project Type 界面，选择 RTL Project，把 Do not specify at this time 勾上（表示在新建工程时不去指定源文件）。在 Default Part 界面，进入 Boards 并在其中找到并选择 Nexys A7-100T。点击 Finish 完成工程创建。

2.2 源文件编写：工程创建后，在 Flow Navigator 界面（工作流窗口）下点击 Add Sources，选择 Add or create design sources 添加源文件。点击 Next 后，Create File 创建文件，并选择文件类型为 Verilog，输入文件名，文件保存路径选择 <Local to Project>，点击 OK，来创建源文件 ALU.v,Regs.v 和 TruthEvaluator.v。用 VScode 编写这三个源文件，结果如下

ALU.v:

```
`timescale 1ns / 1ps
module ALU (
    input  [31:0] A,
    input  [31:0] B,
    input  [ 3:0] ALU_operation,
    output [31:0] res,
    output        zero
);
  wire signed [31:0] A_s = $signed(A);
  wire signed [31:0] B_s = $signed(B);
  wire [31:0] A_u = $unsigned(A);
  wire [31:0] B_u = $unsigned(B);
  wire [31:0] result0= A_s+B_s;
  wire [31:0] result1= A_s-B_s;
  wire [31:0] result2= A<<B[4:0];
```

```
  wire [31:0] result3= (A_s<B_s)?32'b1:32'b0;
  wire [31:0] result4= (A_u<B_u)?32'b1:32'b0;
  wire [31:0] result5= A^B;
  wire [31:0] result6= A>>B[4:0];
  wire [31:0] result7= A_s>>>B_s[4:0];
  wire [31:0] result8= A|B;
  wire [31:0] result9= A&B;
    assign res = (ALU_operation==4'b0000)?result0:
                 (ALU_operation==4'b0001)?result1:
                 (ALU_operation==4'b0010)?result2:
                 (ALU_operation==4'b0011)?result3:
                 (ALU_operation==4'b0100)?result4:
                 (ALU_operation==4'b0101)?result5:
                 (ALU_operation==4'b0110)?result6:
                 (ALU_operation==4'b0111)?result7:
                 (ALU_operation==4'b1000)?result8:
                 (ALU_operation==4'b1001)?result9:0;
  assign zero = ~(|res)?1'b1:1'b0;
endmodule
```

Regs.v:

```
`timescale 1ns / 1ps
module Regs(
    input clk,
    input rst,
    input [4:0] Rs1_addr,
    input [4:0] Rs2_addr,
    input [4:0] Wt_addr,
    input [31:0]Wt_data,
    input RegWrite,
    output [31:0] Rs1_data,
    output [31:0] Rs2_data,
    output reg [31:0] Reg00,
    output reg [31:0] Reg01,
    output reg [31:0] Reg02,
    output reg [31:0] Reg03,
    output reg [31:0] Reg04,
    output reg [31:0] Reg05,
    output reg [31:0] Reg06,
    output reg [31:0] Reg07,
    output reg [31:0] Reg08,
    output reg [31:0] Reg09,
    output reg [31:0] Reg10,
    output reg [31:0] Reg11,
```

```
    output reg [31:0] Reg12,
    output reg [31:0] Reg13,
    output reg [31:0] Reg14,
    output reg [31:0] Reg15,
    output reg [31:0] Reg16,
    output reg [31:0] Reg17,
    output reg [31:0] Reg18,
    output reg [31:0] Reg19,
    output reg [31:0] Reg20,
    output reg [31:0] Reg21,
    output reg [31:0] Reg22,
    output reg [31:0] Reg23,
    output reg [31:0] Reg24,
    output reg [31:0] Reg25,
    output reg [31:0] Reg26,
    output reg [31:0] Reg27,
    output reg [31:0] Reg28,
    output reg [31:0] Reg29,
    output reg [31:0] Reg30,
    output reg [31:0] Reg31
);
always @(posedge clk,posedge rst) begin
    if (rst==1'b1) begin
        Reg00 <= 32'b0;
        Reg01 <= 32'b0;
        Reg02 <= 32'b0;
        Reg03 <= 32'b0;
        Reg04 <= 32'b0;
        Reg05 <= 32'b0;
        Reg06 <= 32'b0;
        Reg07 <= 32'b0;
        Reg08 <= 32'b0;
        Reg09 <= 32'b0;
        Reg10 <= 32'b0;
        Reg11 <= 32'b0;
        Reg12 <= 32'b0;
        Reg13 <= 32'b0;
        Reg14 <= 32'b0;
        Reg15 <= 32'b0;
        Reg16 <= 32'b0;
        Reg17 <= 32'b0;
        Reg18 <= 32'b0;
        Reg19 <= 32'b0;
        Reg20 <= 32'b0;
```

```
        Reg21 <= 32'b0;
        Reg22 <= 32'b0;
        Reg23 <= 32'b0;
        Reg24 <= 32'b0;
        Reg25 <= 32'b0;
        Reg26 <= 32'b0;
        Reg27 <= 32'b0;
        Reg28 <= 32'b0;
        Reg29 <= 32'b0;
        Reg30 <= 32'b0;
        Reg31 <= 32'b0;
    end
    else if(RegWrite==1'b1)begin
        case(Wt_addr)
            5'b00000: Reg00 <= 32'b0;
            5'b00001: Reg01 <= Wt_data;
            5'b00010: Reg02 <= Wt_data;
            5'b00011: Reg03 <= Wt_data;
            5'b00100: Reg04 <= Wt_data;
            5'b00101: Reg05 <= Wt_data;
            5'b00110: Reg06 <= Wt_data;
            5'b00111: Reg07 <= Wt_data;
            5'b01000: Reg08 <= Wt_data;
            5'b01001: Reg09 <= Wt_data;
            5'b01010: Reg10 <= Wt_data;
            5'b01011: Reg11 <= Wt_data;
            5'b01100: Reg12 <= Wt_data;
            5'b01101: Reg13 <= Wt_data;
            5'b01110: Reg14 <= Wt_data;
            5'b01111: Reg15 <= Wt_data;
            5'b10000: Reg16 <= Wt_data;
            5'b10001: Reg17 <= Wt_data;
            5'b10010: Reg18 <= Wt_data;
            5'b10011: Reg19 <= Wt_data;
            5'b10100: Reg20 <= Wt_data;
            5'b10101: Reg21 <= Wt_data;
            5'b10110: Reg22 <= Wt_data;
            5'b10111: Reg23 <= Wt_data;
            5'b11000: Reg24 <= Wt_data;
            5'b11001: Reg25 <= Wt_data;
            5'b11010: Reg26 <= Wt_data;
            5'b11011: Reg27 <= Wt_data;
            5'b11100: Reg28 <= Wt_data;
            5'b11101: Reg29 <= Wt_data;
```

```
            5'b11110: Reg30 <= Wt_data;
            5'b11111: Reg31 <= Wt_data;
        endcase
    end
    end
    assign Rs1_data = (Rs1_addr==5'b00000)?Reg00:
                      (Rs1_addr==5'b00001)?Reg01:
                      (Rs1_addr==5'b00010)?Reg02:
                      (Rs1_addr==5'b00011)?Reg03:
                      (Rs1_addr==5'b00100)?Reg04:
                      (Rs1_addr==5'b00101)?Reg05:
                      (Rs1_addr==5'b00110)?Reg06:
                      (Rs1_addr==5'b00111)?Reg07:
                      (Rs1_addr==5'b01000)?Reg08:
                      (Rs1_addr==5'b01001)?Reg09:
                      (Rs1_addr==5'b01010)?Reg10:
                      (Rs1_addr==5'b01011)?Reg11:
                      (Rs1_addr==5'b01100)?Reg12:
                      (Rs1_addr==5'b01101)?Reg13:
                      (Rs1_addr==5'b01110)?Reg14:
                      (Rs1_addr==5'b01111)?Reg15:
                      (Rs1_addr==5'b10000)?Reg16:
                      (Rs1_addr==5'b10001)?Reg17:
                      (Rs1_addr==5'b10010)?Reg18:
                      (Rs1_addr==5'b10011)?Reg19:
                      (Rs1_addr==5'b10100)?Reg20:
                      (Rs1_addr==5'b10101)?Reg21:
                      (Rs1_addr==5'b10110)?Reg22:
                      (Rs1_addr==5'b10111)?Reg23:
                      (Rs1_addr==5'b11000)?Reg24:
                      (Rs1_addr==5'b11001)?Reg25:
                      (Rs1_addr==5'b11010)?Reg26:
                      (Rs1_addr==5'b11011)?Reg27:
                      (Rs1_addr==5'b11100)?Reg28:
                      (Rs1_addr==5'b11101)?Reg29:
                      (Rs1_addr==5'b11110)?Reg30:
                      (Rs1_addr==5'b11111)?Reg31:32'b0;
    assign Rs2_data = (Rs2_addr==5'b00000)?Reg00:
                      (Rs2_addr==5'b00001)?Reg01:
                      (Rs2_addr==5'b00010)?Reg02:
                      (Rs2_addr==5'b00011)?Reg03:
                      (Rs2_addr==5'b00100)?Reg04:
                      (Rs2_addr==5'b00101)?Reg05:
                      (Rs2_addr==5'b00110)?Reg06:
```

```
                    (Rs2_addr==5'b00111)?Reg07:
                    (Rs2_addr==5'b01000)?Reg08:
                    (Rs2_addr==5'b01001)?Reg09:
                    (Rs2_addr==5'b01010)?Reg10:
                    (Rs2_addr==5'b01011)?Reg11:
                    (Rs2_addr==5'b01100)?Reg12:
                    (Rs2_addr==5'b01101)?Reg13:
                    (Rs2_addr==5'b01110)?Reg14:
                    (Rs2_addr==5'b01111)?Reg15:
                    (Rs2_addr==5'b10000)?Reg16:
                    (Rs2_addr==5'b10001)?Reg17:
                    (Rs2_addr==5'b10010)?Reg18:
                    (Rs2_addr==5'b10011)?Reg19:
                    (Rs2_addr==5'b10100)?Reg20:
                    (Rs2_addr==5'b10101)?Reg21:
                    (Rs2_addr==5'b10110)?Reg22:
                    (Rs2_addr==5'b10111)?Reg23:
                    (Rs2_addr==5'b11000)?Reg24:
                    (Rs2_addr==5'b11001)?Reg25:
                    (Rs2_addr==5'b11010)?Reg26:
                    (Rs2_addr==5'b11011)?Reg27:
                    (Rs2_addr==5'b11100)?Reg28:
                    (Rs2_addr==5'b11101)?Reg29:
                    (Rs2_addr==5'b11110)?Reg30:
                    (Rs2_addr==5'b11111)?Reg31:32'b0;
endmodule
```

TruthEvaluator.v:

```
module TruthEvaluator(
    input  clk,
    input  truth_detection,
    output trust_decision
);
    reg [1:0] state=2'b11;

    always @(posedge clk) begin
        case(state)
            2'b00: state <= truth_detection ? 2'b01 : 2'b00;
            2'b01: state <= truth_detection ? 2'b10 : 2'b00;
            2'b10: state <= truth_detection ? 2'b11 : 2'b01;
            2'b11: state <= truth_detection ? 2'b11 : 2'b10;
    endcase
    end
```

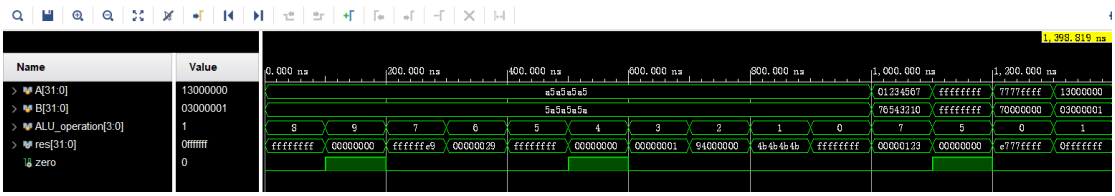
```
    assign trust_decision = state[1];
endmodule
```

2.3 功能仿真：工程创建后，在 Flow Navigator 界面（工作流窗口）下点击 Add Sources，选择 Add or create simulation sources 添加仿真文件文件。点击 Next 后，Create File 创建文件，并选择文件类型为 Verilog，输入文件名，文件保存路径选择 <Local to Project>，点击 OK，来创建仿真文件。用 VScode 输入如下仿真代码。

ALU_tb.v:

```
`timescale 1ns / 1ps

module ALU_tb;
    reg [31:0]  A, B;
    reg [3:0]   ALU_operation;
    wire[31:0]  res;
    wire        zero;
    ALU ALU_u(
        .A(A),
        .B(B),
        .ALU_operation(ALU_operation),
        .res(res),
        .zero(zero)
    );

    initial begin
        A=32'hA5A5A5A5;
        B=32'h5A5A5A5A;
        ALU_operation =4'b1000;
        #100;
        ALU_operation =4'b1001;
        #100;
        ALU_operation =4'b0111;
        #100;
        ALU_operation =4'b0110;
        #100;
        ALU_operation =4'b0101;
        #100;
        ALU_operation =4'b0100;
        #100;
        ALU_operation =4'b0011;
        #100;
        ALU_operation =4'b0010;
        #100;
        ALU_operation =4'b0001;
```

```
        #100;
        ALU_operation =4'b0000;
        #100;
        A=32'h01234567;
        B=32'h76543210;
        ALU_operation =4'b0111;
        #100;
        A=32'hffffffff;
        B=32'hffffffff;
        ALU_operation =4'b0101;
        #100;
        A=32'h7777ffff;
        B=32'h70000000;
        ALU_operation =4'b0000;
        #100;
        A=32'h13000000;
        B=32'h03000001;
        ALU_operation =4'b0001;
        #100;
        $finish;
    end
endmodule
```



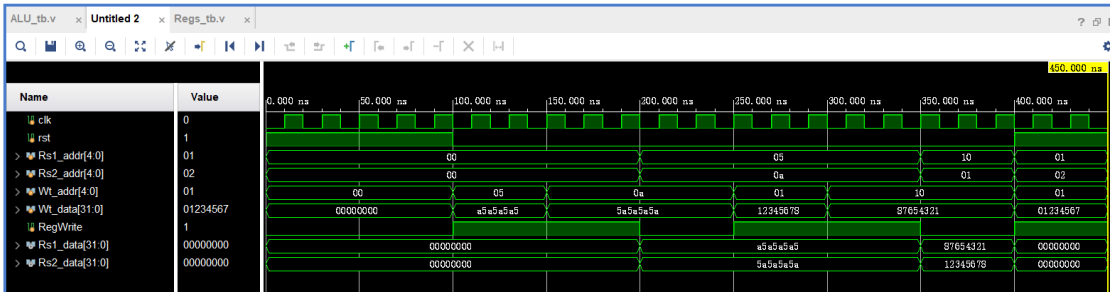
根据操作符对擦作数进行比对，结果与波形图，完全一致，测试程序主要测试了，有无符号数的右移操作与比较操作是否各自正常运行，尤其是两者结果不一值的情况

Regs_tb.v:

```
module Regs_tb;
    reg clk;
    reg rst;
    reg [4:0] Rs1_addr;
    reg [4:0] Rs2_addr;
    reg [4:0] Wt_addr;
    reg [31:0]Wt_data;
    reg RegWrite;
    wire [31:0] Rs1_data;
    wire [31:0] Rs2_data;
    Regs Regs_U(
        .clk(clk),
```

```
        .rst(rst),
        .Rs1_addr(Rs1_addr),
        .Rs2_addr(Rs2_addr),
        .Wt_addr(Wt_addr),
        .Wt_data(Wt_data),
        .RegWrite(RegWrite),
        .Rs1_data(Rs1_data),
        .Rs2_data(Rs2_data)
    );

    always #10 clk = ~clk;

    initial begin
        clk = 0;
        rst = 1;
        RegWrite = 0;
        Wt_data = 0;
        Wt_addr = 0;
        Rs1_addr = 0;
        Rs2_addr = 0;
        #100
        rst = 0;
        RegWrite = 1;
        Wt_addr = 5'b00101;
        Wt_data = 32'ha5a5a5a5;
        #50
        Wt_addr = 5'b01010;
        Wt_data = 32'h5a5a5a5a;
        #50
        RegWrite = 0;
        Rs1_addr = 5'b00101;
        Rs2_addr = 5'b01010;
        #50;
        RegWrite = 1;
        Wt_addr = 5'b00001;
        Wt_data = 32'h12345678;
        #50
        Wt_addr = 5'b10000;
        Wt_data = 32'h87654321;
        #50
        RegWrite = 0;
        Rs1_addr = 5'b10000;
        Rs2_addr = 5'b00001;
        #50;
```

```
        rst=1;
        RegWrite = 1;
        Wt_addr = 5'b00001;
        Wt_data = 32'h01234567;
        Rs1_addr = 5'b00001;
        Rs2_addr = 5'b00010;
        #50;


        $stop();
    end

endmodule
```

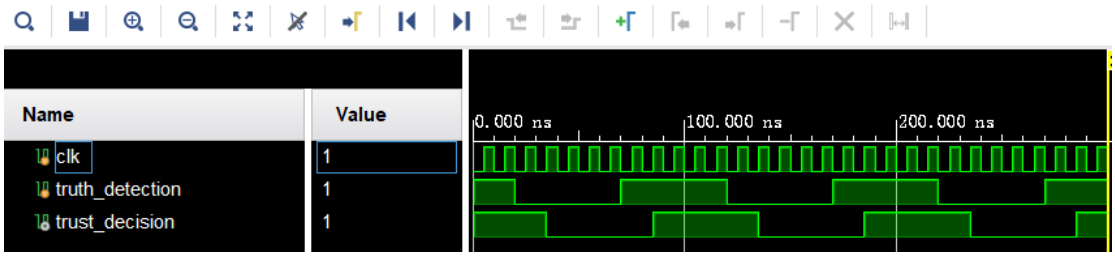


对寄存器进行了多次的读写操作，结果均符合预期

TruthEvaluator_tb.v:

```
module TruthEvaluator_tb();
    // 定义输入和输出信号
    reg clk;
    reg truth_detection;
    wire trust_decision;

    // 实例化 TruthEvaluator 模块
    TruthEvaluator uut (
        .clk(clk),
        .truth_detection(truth_detection),
        .trust_decision(trust_decision)
    );

    // 定义一个初始状态和时钟信号的行为
    always begin
        // 模拟不同的 truth_detection 输入
        truth_detection = 1; #10;
        truth_detection = 1; #10;
        truth_detection = 0; #10;
        truth_detection = 0; #10;
```

```
        truth_detection = 0; #10;
        truth_detection = 0; #10;
        truth_detection = 0; #10;
        truth_detection = 1; #10;
        truth_detection = 1; #10;
        truth_detection = 1; #10;
    end
    initial begin
        clk = 0;
        #300;
        $finish;
    end

    // 生成时钟信号
    always #5 clk = ~clk;
endmodule
```

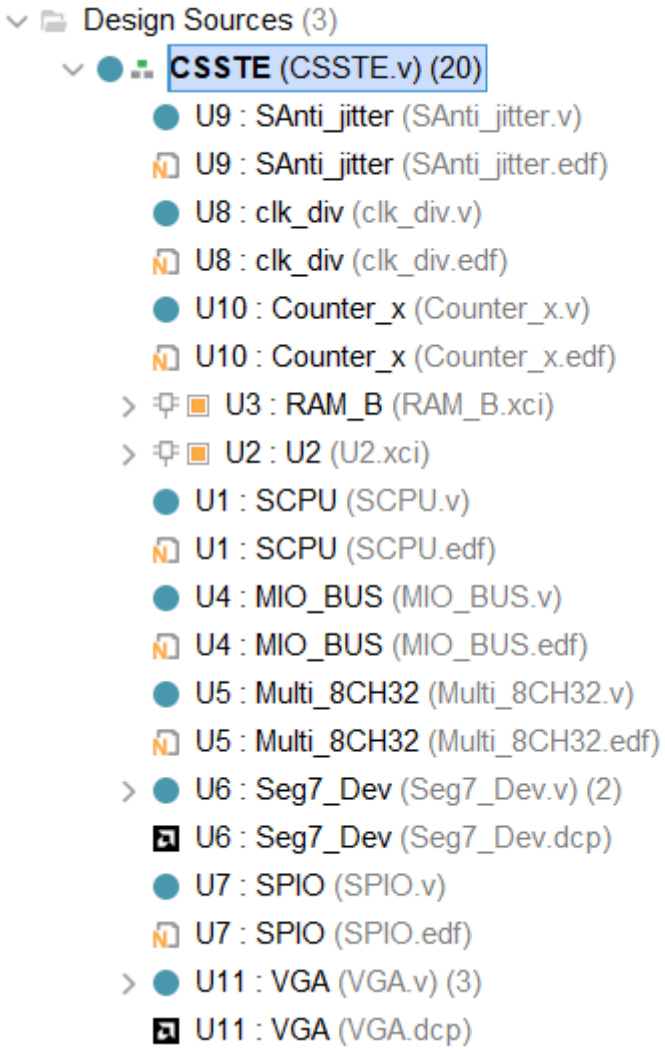


循环调高和调低信任等级，且发生上限和下限溢出，波形展现出循环的图像且符合预期。

3 使用提供的 IP 核搭建测试框架

3.1 项目创建：启动 Vivado 之后，选择顶部快捷栏中的 File -> Project -> New，在 Project Name 界面中设置项目名字为 project3，并设置项目路径。在 Project Type 界面，选择 RTL Project，把 Do not specify at this time 勾上（表示在新建工程时不去指定源文件）。在 Default Part 界面，进入 Boards 并在其中找到并选择 Nexys A7-100T。点击 Finish 完成工程创建。

3.2 导入 IP 核：源文件编写：工程创建后，在 Flow Navigator 界面（工作流窗口）下点击 Add Sources，选择 Add or create design sources 添加源文件。点击 Next 后，Add Directories，选择 IP 核所在目录。



3.3 创建源文件：同上新建文件 CSSTE.v。依据 CSSTE.pdf 中的电路图，编写该文件。

```
`timescale 1ns / 1ps
module CSSTE(
    input         clk_100mhz,
    input         RSTN,
    input  [3:0]  BTN_y,
    input  [15:0] SW,
    output [3:0]  Blue,
    output [3:0]  Green,
    output [3:0]  Red,
    output        HSYNC,
    output        VSYNC,
    output [15:0] LED_out,
    output [7:0] AN,
    output [7:0] segment
);
```

```
wire [3:0] U9_BTN_OK;
wire [15:0] U9_SW_OK;
wire U9_rst;
SAnti_jitter U9 (
    .clk(clk_100mhz),
    .RSTN(RSTN),
    .Key_y(BTN_y),
    .SW(SW),
    .BTN_OK(U9_BTN_OK),
    .SW_OK(U9_SW_OK),
    .rst(U9_rst)
);
wire [31:0] U8_clkdiv;
wire U8_Clk_CPU;
clk_div U8 (
    .clk(clk_100mhz),
    .rst(U9_rst),
    .SW2(U9_SW_OK[2]),
    .SW8(U9_SW_OK[8]),
    .STEP(U9_SW_OK[10]),
    .clkdiv(U8_clkdiv),
    .Clk_CPU(U8_Clk_CPU)
);
wire U10_counter0_OUT, U10_counter1_OUT, U10_counter2_OUT;
wire [31:0] U10_counter_out;
wire U4_counter_we;
wire [31:0] U4_Peripheral_in;
wire [1:0] U7_counter_set;
Counter_x U10 (
    .clk(~U8_Clk_CPU),
    .rst(U9_rst),
    .clk0(U8_clkdiv[6]),
    .clk1(U8_clkdiv[9]),
    .clk2(U8_clkdiv[11]),
    .counter_we(U4_counter_we),
    .counter_val(U4_Peripheral_in),
    .counter_ch(U7_counter_set),
    .counter0_OUT(U10_counter0_OUT),
    .counter1_OUT(U10_counter1_OUT),
    .counter2_OUT(U10_counter2_OUT),
    .counter_out(U10_counter_out)
);
wire [31:0] U3_douta;
wire [9:0] U4_ram_addr;
```

```
wire [31:0] U4_ram_data_in;
wire U4_data_ram_we;
RAM_B U3 (
    .clka (~clk_100mhz),
    .wea  (U4_data_ram_we),
    .addra(U4_ram_addr),
    .dina (U4_ram_data_in),
    .douta(U3_douta)
);
wire [31:0] U1_PC_out;
wire [31:0] U2_spo;
U2 U2 (
    .a  (U1_PC_out[11:2]),
    .spo(U2_spo)
);
wire [31:0] U4_Cpu_data4bus, U1_Addr_out, U1_Data_out;
wire U1_MemRW;
SCPU U1 (
    .clk(U8_Clk_CPU),
    .rst(U9_rst),
    .Data_in(U4_Cpu_data4bus),
    .inst_in(U2_spo),
    .MemRW(U1_MemRW),
    .Addr_out(U1_Addr_out),
    .Data_out(U1_Data_out),
    .PC_out(U1_PC_out)
);
wire [15:0] U7_LED_out;
assign LED_out=U7_LED_out;
wire U4_GPIOf0000000_we, U4_GPIOe0000000_we;
MIO_BUS U4 (
    .clk(clk_100mhz),
    .rst(U9_rst),
    .BTN(U9_BTN_OK),
    .SW(U9_SW_OK),
    .mem_w(U1_MemRW),
    .Cpu_data2bus(U1_Data_out),
    .addr_bus(U1_Addr_out),
    .ram_data_out(U3_douta),
    .led_out(U7_LED_out),
    .counter_out(U10_counter_out),
    .counter0_out(U10_counter0_OUT),
    .counter1_out(U10_counter1_OUT),
    .counter2_out(U10_counter2_OUT),
```

```
    .Cpu_data4bus(U4_Cpu_data4bus),
    .ram_addr(U4_ram_addr),
    .ram_data_in(U4_ram_data_in),
    .data_ram_we(U4_data_ram_we),
    .counter_we(U4_counter_we),
    .Peripheral_in(U4_Peripheral_in),
    .GPIOe0000000_we(U4_GPIOe0000000_we),
    .GPIOf0000000_we(U4_GPIOf0000000_we)
);
wire [7:0] U5_point_out, U5_LE_out;
wire [31:0] U5_Disp_num;
Multi_8CH32 U5 (
    .clk(~U8_Clk_CPU),
    .rst(U9_rst),
    .EN(U4_GPIOe0000000_we),
    .point_in({U8_clkdiv[31:0], U8_clkdiv[31:0]}),
    .Test(U9_SW_OK[7:5]),
    .LES(64'b0),
    .Data0(U4_Peripheral_in),
    .data1({2'b0, U1_PC_out[31:2]}),
    .data2(U2_spo),
    .data3(U10_counter_out),
    .data4(U1_Addr_out),
    .data5(U1_Data_out),
    .data6(U4_Cpu_data4bus),
    .data7(U1_PC_out),
    .point_out(U5_point_out),
    .LE_out(U5_LE_out),
    .Disp_num(U5_Disp_num)
);
Seg7_Dev U6 (
    .scan({U8_clkdiv[18], U8_clkdiv[17], U8_clkdiv[16]}),
    .les(U5_LE_out),
    .point(U5_point_out),
    .disp_num(U5_Disp_num),
    .AN(AN),
    .segment(segment)
);
SPIO U7 (
    .clk(~U8_Clk_CPU),
    .rst(U9_rst),
    .Start(U8_clkdiv[20]),
    .EN(U4_GPIOf0000000_we),
    .P_Data(U4_Peripheral_in),
```

```
    .counter_set(U7_counter_set),
    .LED_out(U7_LED_out)
);
VGA U11 (
    .clk_25m(U8_clkdiv[1]),
    .clk_100m(clk_100mhz),
    .rst(U9_rst),
    .pc(U1_PC_out),
    .inst(U2_spo),
    .alu_res(U1_Addr_out),
    .dmem_addr(U1_Addr_out),
    .mem_wen(U1_MemRW),
    .dmem_o_data(U3_douta),
    .dmem_i_data(U4_ram_data_in),
    .hs(HSYNC),
    .vs(VSYNC),
    .vga_r(Red),
    .vga_g(Green),
    .vga_b(Blue)
);


endmodule
```

3.4 修改 VGA 文件，修改附件 VGADisplay.v 中的文件路径，修该 VGA。V 中的端口描述，并连接上 vga_debugger 实例接口。

VGA.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2024/02/29 13:33:43
// Design Name:
// Module Name: VGA
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
//////////////////////////////////////////////////////////////////
//////////////////

module VGA (
    input wire clk_25m,
    input wire clk_100m,
    input wire rst,
    input wire [31:0] pc,
    input wire [31:0] inst,
    input wire [31:0] alu_res,
    input wire mem_wen,
    input wire [31:0] dmem_o_data,
    input wire [31:0] dmem_i_data,
    input wire [31:0] dmem_addr,

    output wire hs,
    output wire vs,
    output wire [3:0] vga_r,
    output wire [3:0] vga_g,
    output wire [3:0] vga_b,

    input wire [4:0] rs1,
    input wire [31:0] rs1_val,
    input wire [4:0] rs2,
    input wire [31:0] rs2_val,
    input wire [4:0] rd,
    input wire [31:0] reg_i_data,
    input wire reg_wen,
    input wire is_imm,
    input wire is_auipc,
    input wire is_lui,
    input wire [31:0] imm,
    input wire [31:0] a_val,
    input wire [31:0] b_val,
    input wire [3:0] alu_ctrl,
    input wire [2:0] cmp_ctrl,
    input wire cmp_res,
    input wire is_branch,
    input wire is_jal,
    input wire is_jalr,
    input wire do_branch,
    input wire [31:0] pc_branch,
    input wire mem_ren,
```

```
    input wire csr_wen,
    input wire [11:0] csr_ind,
    input wire [1:0] csr_ctrl,
    input wire [31:0] csr_r_data,
    input wire [31:0] x0,
    input wire [31:0] ra,
    input wire [31:0] sp,
    input wire [31:0] gp,
    input wire [31:0] tp,
    input wire [31:0] t0,
    input wire [31:0] t1,
    input wire [31:0] t2,
    input wire [31:0] s0,
    input wire [31:0] s1,
    input wire [31:0] a0,
    input wire [31:0] a1,
    input wire [31:0] a2,
    input wire [31:0] a3,
    input wire [31:0] a4,
    input wire [31:0] a5,
    input wire [31:0] a6,
    input wire [31:0] a7,
    input wire [31:0] s2,
    input wire [31:0] s3,
    input wire [31:0] s4,
    input wire [31:0] s5,
    input wire [31:0] s6,
    input wire [31:0] s7,
    input wire [31:0] s8,
    input wire [31:0] s9,
    input wire [31:0] s10,
    input wire [31:0] s11,
    input wire [31:0] t3,
    input wire [31:0] t4,
    input wire [31:0] t5,
    input wire [31:0] t6,
    input wire [31:0] mstatus_o,
    input wire [31:0] mcause_o,
    input wire [31:0] mepc_o,
    input wire [31:0] mtval_o,
    input wire [31:0] mtvec_o,
    input wire [31:0] mie_o,
    input wire [31:0] mip_o
```

```
);
  wire [9:0] vga_x;
  wire [8:0] vga_y;
  wire video_on;
  VgaController vga_controller (
      .clk     (clk_25m),
      .rst     (rst),
      .vga_x   (vga_x),
      .vga_y   (vga_y),
      .hs      (hs),
      .vs      (vs),
      .video_on(video_on)
  );
  wire display_wen;
  wire [11:0] display_w_addr;
  wire [7:0] display_w_data;
  VgaDisplay vga_display (
      .clk     (clk_100m),
      .video_on(video_on),
      .vga_x   (vga_x),
      .vga_y   (vga_y),
      .vga_r   (vga_r),
      .vga_g   (vga_g),
      .vga_b   (vga_b),
      .wen     (display_wen),
      .w_addr  (display_w_addr),
      .w_data  (display_w_data)
  );
  //modify a
  VgaDebugger vga_debugger (
      .clk           (clk_100m),
      .display_wen   (display_wen),
      .display_w_addr(display_w_addr),
      .display_w_data(display_w_data),
      .pc            (pc),
      .inst          (inst),
      .rs1           (rs1),
      .rs1_val       (rs1_val),
      .rs2           (rs2),
      .rs2_val       (rs2_val),
      .rd            (rd),
      .reg_i_data    (reg_i_data),
      .reg_wen       (reg_wen),
      .is_imm        (is_imm),
```

```
    .is_auipc     (is_auipc),
    .is_lui       (is_lui),
    .imm          (imm),
    .a_val        (a_val),
    .b_val        (b_val),
    .alu_ctrl     (alu_ctrl),
    .cmp_ctrl     (cmp_ctrl),
    .alu_res      (alu_res),
    .cmp_res      (cmp_res),
    .is_branch    (is_branch),
    .is_jal       (is_jal),
    .is_jalr      (is_jalr),
    .do_branch    (do_branch),
    .pc_branch    (pc_branch),
    .mem_wen      (mem_wen),
    .mem_ren      (mem_ren),
    .dmem_o_data  (dmem_o_data),
    .dmem_i_data  (dmem_i_data),
    .dmem_addr    (dmem_addr),
    .csr_wen      (csr_wen),
    .csr_ind      (csr_ind),
    .csr_ctrl     (csr_ctrl),
    .csr_r_data   (csr_r_data),
    .x0           (x0),
    .ra           (ra),
    .sp           (sp),
    .gp           (gp),
    .tp           (tp),
    .t0           (t0),
    .t1           (t1),
    .t2           (t2),
    .s0           (s0),
    .s1           (s1),
    .a0           (a0),
    .a1           (a1),
    .a2           (a2),
    .a3           (a3),
    .a4           (a4),
    .a5           (a5),
    .a6           (a6),
    .a7           (a7),
    .s2           (s2),
    .s3           (s3),
    .s4           (s4),
```

```
      .s5           (s5),
      .s6           (s6),
      .s7           (s7),
      .s8           (s8),
      .s9           (s9),
      .s10          (s10),
      .s11          (s11),
      .t3           (t3),
      .t4           (t4),
      .t5           (t5),
      .t6           (t6),
      .mstatus_o    (mstatus_o),
      .mcause_o     (mcause_o),
      .mepc_o       (mepc_o),
      .mtval_o      (mtval_o),
      .mtvec_o      (mtvec_o),
      .mie_o        (mie_o),
      .mip_o        (mip_o)
  );
endmodule
```

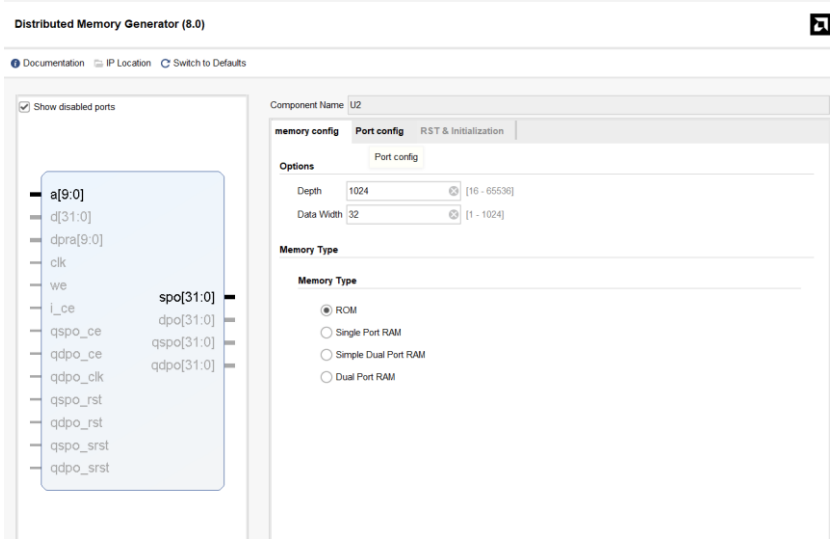
VgaDisplay.v

```
module VgaDisplay(
    input wire clk,
    input wire video_on,
    input wire [9:0] vga_x,
    input wire [8:0] vga_y,
    output wire [3:0] vga_r,
    output wire [3:0] vga_g,
    output wire [3:0] vga_b,
    input wire wen,
    input wire [11:0] w_addr,
    input wire [7:0] w_data
);

    (* ram_style = "block" *) reg [7:0] display_data[0:4095];
    initial
$readmemh("..//..//..//..//..//..//lab0//project_3//vga_debugg
er.mem", display_data);

    wire [11:0] text_index = (vga_y / 16) * 80 + vga_x / 8;
    // I don't know why I need this '- (vga_y / 16)' ...
```

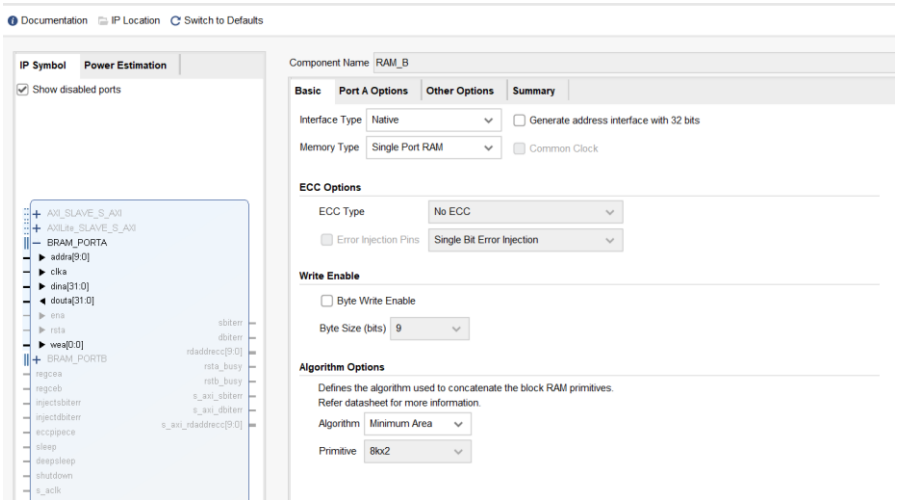
```
    wire [7:0] text_ascii = display_data[text_index] - (vga_y /
16);
    wire [2:0] font_x = vga_x % 8;
    wire [3:0] font_y = vga_y % 16;
    wire [11:0] font_addr = text_ascii * 16 + font_y;

    (* ram_style = "block" *) reg [7:0] fonts_data[0:4095];
    initial
$readmemh("..//..//..//..//..//..//lab0//project_3//font_8x16.
mem", fonts_data);
    wire [7:0] font_data = fonts_data[font_addr];

    assign { vga_r, vga_g, vga_b } = (video_on & font_data[7 -
font_x]) ? 12'hfff : 12'h0;

    always @(posedge clk) begin
        if (wen) begin
            display_data[w_addr] <= w_data;
        end
    end

endmodule
```

3.5 生成 Rom 和 Ram 核, 在工作流窗口选择 IP Catalog，将在工作区弹出 IP Catalog 的窗口,点击搜索栏，输入 memory generator，选择 Distributed Memory Generator,创建 Rom,接入 I_mem.coe 文件,该文件内的指令实现了生成了一个斐波那契数列并储存寄存器中的任务。同样，选择 Block Memory Generator，生成一个 Single Port RAM，载入 D_mem.coe 文件。

Rom



Ram



3.6 仿照上述操作导入约束文件 A7.xdc，点击工作流窗口的 PROGRAM AND DEBUG > Generate Bitstream 生成比特流。得到 bitstream 后，需要将下载器连接到电脑上，点击 PROGRAM AND DEBUG > Open Hardware Manager > Open Target > Auto Connect 进行识别和连接，成功连接后，点击 Program Device 选择 xc7a100t 设备，在下载程序界面选择刚刚生成的比特流文件，将其下载到板上。具体下板图像分析见第二节第三点。

4 实现乘法器 / 除法器

4.1 项目创建：启动 Vivado 之后，选择顶部快捷栏中的 File -> Project -> New，在 Project Name 界面中设置项目名字为 project3，并设置项目路径。在 Project Type 界面，选择 RTL Project，把 Do not specify at this time 勾上（表示在新建工程时不去指定源文件）。在 Default Part 界面，进入 Boards 并在其中找到并选择 Nexys A7-100T。点击 Finish 完成工程创建。

4.2 源文件编写：工程创建后，在 Flow Navigator 界面（工作流窗口）下点击 Add Sources，选择 Add or create design sources 添加源文件。点击 Next 后，Create File 创建文件，并选择文件类型为 Verilog，输入文件名，文件保存路径选择 <Local to Project>，点击 OK，来创建源文件 multiplier.v 等三个文件。用 VScode 编写这三个源文件，结果如下

multiplier.v

```
module multiplier (
    input clk,
    input start,
    input [31:0] A,
    input [31:0] B,
    output reg finish,
```

```
    output reg [63:0] res
);

  reg state;  // 记录 multiplier 是不是正在进行运算
  reg [31:0] multiplicand;  // 保存当前运算中的被乘数

  reg [5:0] cnt;  // 记录当前计算已经经历了几个周期（运算与移位）
  wire [5:0] cnt_next = cnt + 6'b1;

  reg sign = 0;  // 记录当前运算的结果是否是负数

  initial begin
    res <= 0;
    state <= 0;
    finish <= 0;
    cnt <= 0;
    multiplicand <= 0;
  end

  always @(posedge clk) begin
    if (~state && start) begin
      sign <= A[31] ^ B[31];
      multiplicand <= B[31] ? ~B + 1 : B;
      state <= 1;
      cnt <= 0;
      res <= {32'b0, (A[31]) ? ~A + 1 : A};
      finish <= 0;
    end else if (state & cnt[5]) begin
      cnt <= 0;
      finish <= 1;
      state <= 0;
      res = sign ? ~res + 1 : res;
    end else if (state) begin
      cnt <= cnt_next;
      res <= {1'b0, (res[0] ? res[63:32] + multiplicand :
res[63:32]), res[31:1]};
    end
  end

endmodule
```

divider.v

```
module divider (
    input             clk,
```

```
    input             rst,
    input             start,        // 开始运算
    input      [31:0] dividend,     // 被除数
    input      [31:0] divisor,      // 除数
    output reg        divide_zero,  // 除零异常
    output reg        finish,       // 运算结束信号
    output     [31:0] res,          // 商
    output     [31:0] rem           // 余数
);


  reg state;  // 记录 multiplier 是不是正在进行运算
  reg [31:0] divisor_temp;  // 保存当前运算中的除数

  reg [5:0] cnt;  // 记录当前计算已经经历了几个周期（运算与移位）
  wire [5:0] cnt_next = cnt + 6'b1;
  reg [63:0] temp = 0;  // 保存当前运算的中间结果
  assign rem = temp[63:32];  // 余数
  assign res = temp[31:0];  // 商
  wire more = temp[62:31] >= divisor_temp;  //判断 temp 够除
  initial begin
    temp <= 0;
    state <= 0;
    finish <= 0;
    cnt <= 0;
    divisor_temp <= 0;
  end

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      temp <= 0;
      state <= 0;
      finish <= 0;
      cnt <= 0;
      divisor_temp <= 0;
    end else if (~state && start) begin
      divide_zero <= ~(|divisor);
      divisor_temp <= divisor;
      state <= (|divisor);
      cnt <= 0;
      temp <= {32'b0, dividend};
      finish <= ~(|divisor);
    end else if (state & cnt[5] & ~divide_zero) begin
      cnt <= 0;
      finish <= 1;
```

```
      state <= 0;
    end else if (state & ~divide_zero) begin
      cnt  <= cnt_next;
      temp <= {(more ? (temp[62:31] - divisor_temp[31:0]) :
temp[62:31]), temp[30:0], more};
    end
  end

endmodule
```

floatadder.v

```
`timescale 1ns / 1ps
module floatadder (
    input wire clk,
    input wire [31:0] adder1,
    input wire [31:0] adder2,
    input wire start,
    output wire [31:0] sum,
    output reg finish,
    output reg overflow,
    output reg error
);
  reg state;  // 记录 multiplier 是不是正在进行运算
  reg [4:0] cnt;  // 记录当前计算已经经历了几个周期（运算与移位）
  wire [4:0] cnt_next = cnt + 5'b1;  // 下一个周期
  wire exp_more = adder1[30:23] > adder2[30:23];  // 比较指数大小
  wire [22:0] exp_diff = exp_more ? (adder1[30:23] -
adder2[30:23]) : (adder2[30:23] - adder1[30:23]);// 指数差
  wire[25:0] adder1_ext=(~|adder1[30:23])?(adder1[31]?{1'b1,-
{1'b0,adder1[22:0],1'b0}}:{1'b0,{1'b0,adder1[22:0],1'b0}}):(adde
r1[31]?{1'b1,-
{2'b01,adder1[22:0]}}:{1'b0,{2'b01,adder1[22:0]}});// 扩展尾数,防
止运算过程中溢出
  wire [25:0] adder1_rl = $signed(adder1_ext) >>> exp_diff;  //
预先对齐
  wire[25:0] adder2_ext=(~|adder2[30:23])?(adder2[31]?{1'b1,-
{1'b0,adder2[22:0],1'b0}}:{1'b0,{1'b0,adder2[22:0],1'b0}}):(adde
r2[31]?{1'b1,-
{2'b01,adder2[22:0]}}:{1'b0,{2'b01,adder2[22:0]}});
  wire [25:0] adder2_rl = $signed(adder2_ext) >>> exp_diff;
  reg [7:0] exp;  // 结果指数
  reg [25:0] res;  // 结果尾数
```

```
  assign sum[30:23] = exp;  // 输出结果指数
  assign sum[31] = res[25];  // 输出结果符号
  wire [23:0] value = res[25] ? ~res[25:2] + 1 : res[25:2];// 对
结果尾数进行符号处理,使其为正
  assign sum[22:0] = value[22:0];  // 输出结果尾数

  // 初始化
  initial begin
    state <= 0;
    cnt <= 0;
    exp <= 0;
    res <= 0;
    finish <= 0;
    overflow <= 0;
    error <= 0;
  end
  always @(posedge clk) begin
    // 初始化
    if (~state && start) begin
      state <= 1;
      cnt <= 0;
      // 指数对齐
      exp <= exp_more ? adder1[30:23] : adder2[30:23];
      res <= 0;
      finish <= 0;
      overflow <= 0;
      error <= 0;
      // 尾数对齐并计算
      res <= (exp_more ? adder1_ext : adder1_rl) + (~exp_more ?
adder2_ext : adder2_rl);
    // 运算结束操作,cnt==24 时计算结束
    end else if (state & cnt[4] & cnt[3]) begin
      // 溢出
      overflow <= &exp;
      // 根据溢出,为 0,近 0 值,规范化数进行处理
      res[24:0] <= ~|res[23:0] ? 25'b0 : &exp ? 0 : ~|exp ?
res[24:0] : res[24:0] << 1'b1;
      // 尾数为 0 指数置为 0
      exp <= ~|res[23:0] ? 0 : exp;
      //w 位数为零,则符号位为 0
      res[25] <= ~|res[23:0] ? 0 : res[25];
      // 结束,输出完成信号
      state <= 0;
      finish <= 1;
```

```
      cnt <= 0;
      // 进位计算
    end else if (state & ~(|cnt[4:0])) begin
      // 判断输入是否为 NAN 或者 INF
      if (&exp) begin
        error <= 1;
        finish <= 1;
        state <= 0;
        cnt <= 0;
        // 判断是否进位
      end else if (res[25] ^ res[24]) begin
        // 进位无需退位操作,直接跳转到结束
        cnt <= 24;
        exp <= exp + 1;
      end else begin
        // 无进位可能需要退位操作
        cnt <= cnt_next;
        // 退位操作
        res[24:0] = res[24:0] << 1'b1;
      end
      // 退位操作
    end else if (state) begin
      // 判断是否为近零值,即退无可退
      if (~|exp) begin
        cnt <= 24;
        // 判断是否退位完成
      end else if (res[25] ^ res[24]) begin
        // 退位完成,进入结束
        cnt <= 24;
      end else begin
        // 退位未完成,继续退位
        cnt <= cnt_next;
        res[24:0] = res[24:0] << 1'b1;
        exp <= exp - 1;
      end
    end
  end
endmodule
```

这是一个浮点数加法器，它接收两个 32 位的 IEEE 754 格式的浮点数作为输入，并输出它们的和。这个加法器的工作原理如下：

1. 指数对齐：首先，加法器会比较两个输入数的指数部分，然后通过右移尾数部分的方式，将指数较小的数对齐到指数较大的数，同时依据符号位将无符号数转化为有符号数。这样做的目的是为了使两个数可以直接相加并且不发生溢出。

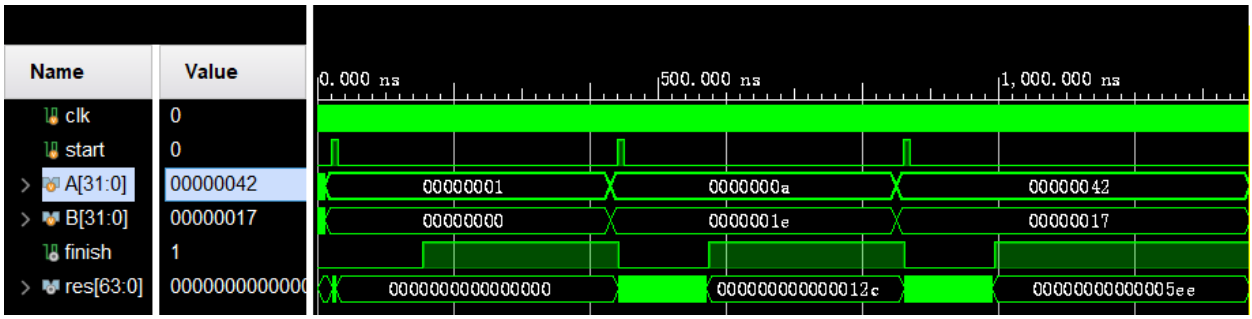
2. 尾数相加：对齐后，加法器会将两个数的尾数部分相加，得到一个初步的结果。
3. 结果规范化：加法器会检查相加得到的结果是否需要进行进位或退位操作，以保证结果的尾数是规范化的。
4. 错误检查：加法器会检查输入的数是否是非法的（如 NaN 或 INF）。如果发生，那么 error 信号会被置为 1。如果在运算过程中发生了溢出，那么 overflow 信号会被置、为 1。
5. 结果输出：最后，加法器会将计算得到的结果的符号位、指数位和尾数位（转换为无符号数后）分别赋值给 sum 信号的相应位，得到最终的加法结果。

4.3 功能仿真：工程创建后，在 Flow Navigator 界面（工作流窗口）下点击 Add Sources，选择 Add or create simulation sources 添加仿真文件文件。点击 Next 后，Create File 创建文件，并选择文件类型为 Verilog，输入文件名，文件保存路径选择 <Local to Project>，点击 OK，来创建仿真文件。用 VScode 输入如下仿真代码。

multipler_tb.v:

```
module multiplier_tb;

  reg clk, start;
  reg[31:0] A;
  reg[31:0] B;

  wire finish;
  wire[63:0] res;

  multiplier
m0(.clk(clk), .start(start), .A(A), .B(B), .finish(finish), .res
(res));

  initial begin
    $dumpfile("multiplier_signed.vcd");
    $dumpvars(0, multiplier_tb);

    clk = 0;
    start = 0;
    #10;
    A = 32'd1;
    B = 32'd0;
    #10 start = 1;
    #10 start = 0;
    #400;

    A = 32'd10;
```

```
    B = 32'd30;
    #10 start = 1;
    #10 start = 0;
    #400;

    A = 32'd66;
    B = 32'd23;
    #10 start = 1;
    #10 start = 0;
    #500;
    $finish();
  end

  always begin
    #2 clk = ~clk;
  end



endmodule
```



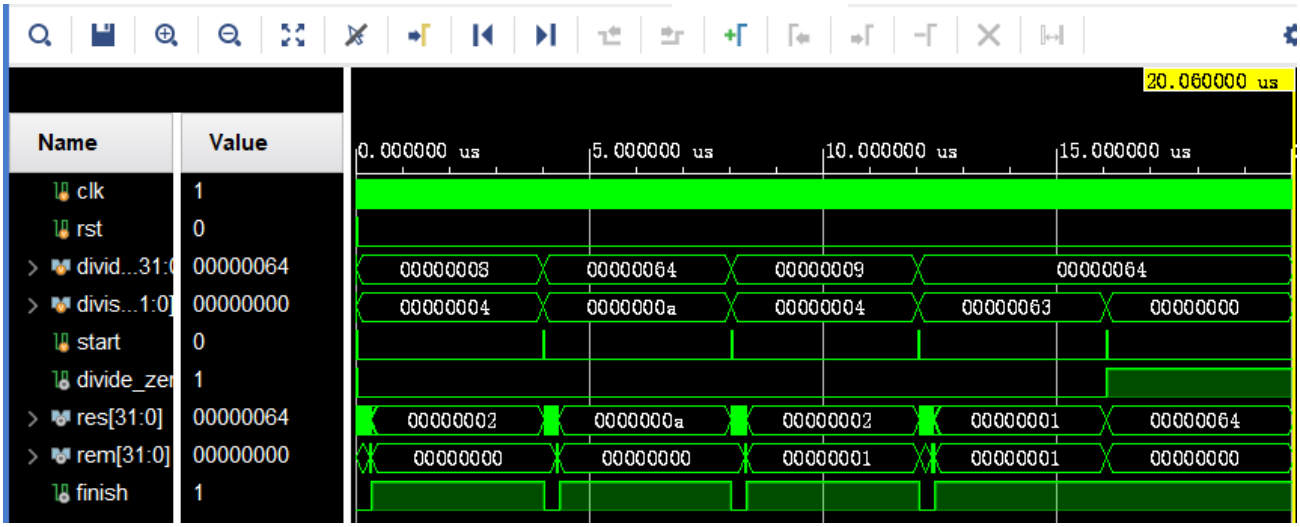
这是乘法器仿真的结果，已知（16 进制）1*0=0，a*1e=12c，42*17=5ee。和上面的仿真结果相同。

divider_tb.v

```
`timescale 1ns / 1ps
module divider_tb ();
  reg clk;
  reg rst;
  reg [31:0] dividend;
  reg [31:0] divisor;
  reg start;

  wire divide_zero;
  wire [31:0] res;
  wire [31:0] rem;
  wire finish;
```

```
divider u_div (
    .clk(clk),
    .rst(rst),
    .dividend(dividend),
    .divisor(divisor),
    .start(start),
    .divide_zero(divide_zero),
    .res(res),
    .rem(rem),
    .finish(finish)
);
always #5 clk = ~clk;

initial begin
  clk   = 0;
  rst   = 1;
  start = 0;
  #10 rst = 0;
  start = 1;
  dividend = 32'd8;
  divisor = 32'd4;
  #10 start = 0;
  #4000;
  start = 1;
  dividend = 32'd100;
  divisor = 32'd10;
  #10 start = 0;
  #4000;
  start = 1;
  dividend = 32'd9;
  divisor = 32'd4;
  #10 start = 0;
  #4000;
  start = 1;
  dividend = 32'd100;
  divisor = 32'd99;
  #10 start = 0;
  #4000;
  start = 1;
  dividend = 32'd100;
  divisor = 32'd0;
  #10 start = 0;
  #4000 $stop();
```

```
  end
endmodule
```

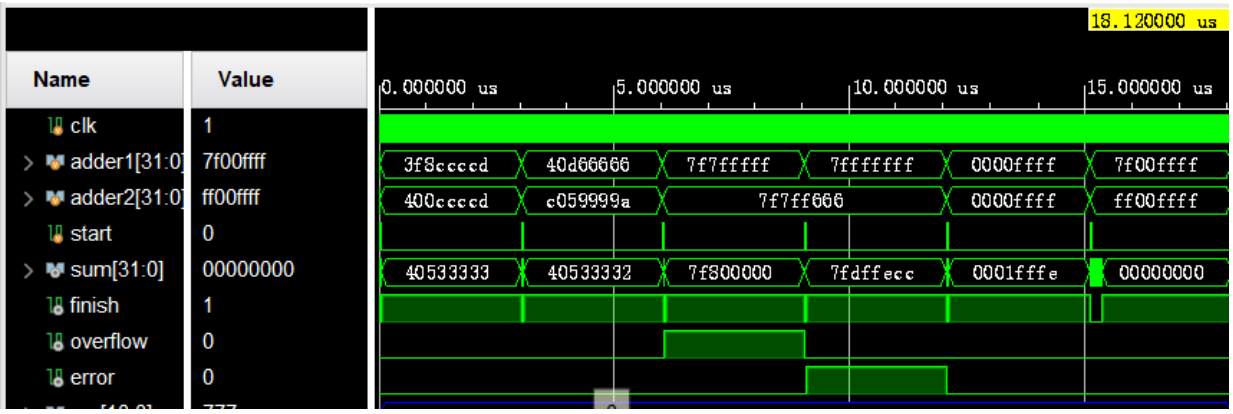


这是除法器的仿真的波形图，已知（16 进制）8/4=2…0 ，64/a=a…0，9/4=2…1，64/63=1…1。仿真波形符合预期，且 64/0 报了除零异常。

floatadder.v

```
`timescale 1ns / 1ps
module floatadder_tb ();
  reg clk;
  reg [31:0] adder1;
  reg [31:0] adder2;
  reg start;
  wire [31:0] sum;
  wire finish;
  wire overflow;
  wire error;
  wire [10:0] res;
  wire [10:0] adder1_ext;
  wire [10:0] adder2_ext;
  wire [10:0] adder1_rl;
  wire [10:0] adder2_rl;
  always begin
    #5 clk = ~clk;
  end
  floatadder floatadder_0 (
      .clk(clk),
      .adder1(adder1),
      .adder2(adder2),
      .start(start),
      .sum(sum),
```

```
      .finish(finish),
      .overflow(overflow),
      .error(error)
  );
  initial begin
    clk = 0;
    adder1 = 32'b0;
    adder2 = 32'b0;
    start = 0;
    #10 adder1 = 32'h3f8ccccd;
    adder2 = 32'h400ccccd;
    start  = 1;
    #10 start = 0;
    #3000;
    #10 adder1 = 32'h40d66666;
    adder2 = 32'hc059999a;
    start  = 1;
    #10 start = 0;
    #3000;
    #10 adder1 = 32'h7f7fffff;
    adder2 = 32'h7f7ff666;
    start  = 1;
    #10 start = 0;
    #3000;
    #10 adder1 = 32'h7fffffff;
    adder2 = 32'h7f7ff666;
    start  = 1;
    #10 start = 0;
    #3000;
    #10 adder1 = 32'h0000ffff;
    adder2 = 32'h0000ffff;
    start  = 1;
    #10 start = 0;
    #3000;
    #10 adder1 = 32'h7f00ffff;
    adder2 = 32'hff00ffff;
    start  = 1;
    #10 start = 0;
    #3000;
    $finish;
  end
endmodule
```



这是浮点数加法器的仿真波形,我编写了一个 c 语言程序来输出计算后的结果(由于 c 的截断方式和我的不一样,会导致在最低位上会有变动,此外对于 NAN 和溢出的处理页不一致)

```c
int main()
{
    float a, b, c;
    unsigned int *p = (unsigned int *)&a, *q = (unsigned int *)&b, *r = (unsigned int *)&c;
    // 3f8ccccd
    // 400ccccd
    *p = 0x3f8ccccd;
    *q = 0x400ccccd;
    c = a + b;
    printf("%x + %x = %x\n", *p, *q, *r);

    // 40d66666
    // c059999a
    *p = 0x40d66666;
    *q = 0xc059999a;
    c = a + b;
    printf("%x + %x = %x\n", *p, *q, *r);
    // 7f7fffff
    // 7f7ff666
    *p = 0x7f7fffff;
    *q = 0x7ffff666;
    c = a + b;
    printf("%x + %x = %x\n", *p, *q, *r);
    // 7fffffff
    // 7f7ff666
    *p = 0x7fffffff;
    *q = 0x7ffff666;
    c = a + b;
    printf("%x + %x = %x\n", *p, *q, *r);
```

```
    // 0000ffff
    // 0000ffff
    *p = 0x0000ffff;
    *q = 0x0000ffff;
    c = a + b;
    printf("%08x + %08x = %08x\n", *p, *q, *r);
    // 7f00ffff
    // ff00ffff
    *p = 0x7f00ffff;
    *q = 0xff00ffff;
    c = a + b;
    printf("%08x + %08x = %08x\n", *p, *q, *r);
}
```

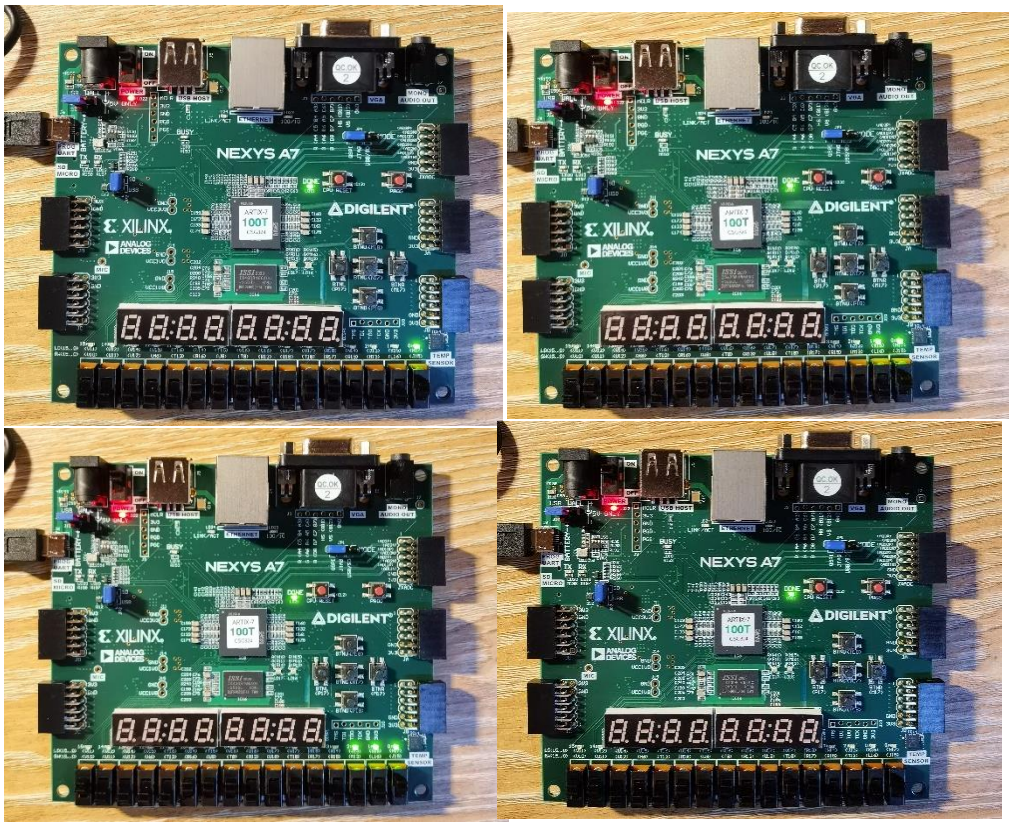
```
3f8ccccd + 400ccccd = 40533334
40d66666 + c059999a = 40533332
7f7fffff + 7ffff666 = 7ffff666
7fffffff + 7ffff666 = 7ffff666
0000ffff + 0000ffff = 0001fffe
7f00ffff + ff00ffff = 00000000
```

波形图与预期结果一致,且溢出和 NAN 均做出了正确输出.

二、实验结果与分析

1　安装并使用 Vivado

1.1　实验图片



1.2 分析：进行如下设置 SW[3:0]:4’b0001,SW[7:4]:4’b0011,SW[11:8]:4’b0111，可以看到讲 SW[15:14]从 0 拨到 3,最左侧三个二极管依次显示 b0001,b0011,b0111,b0000 符合预期.

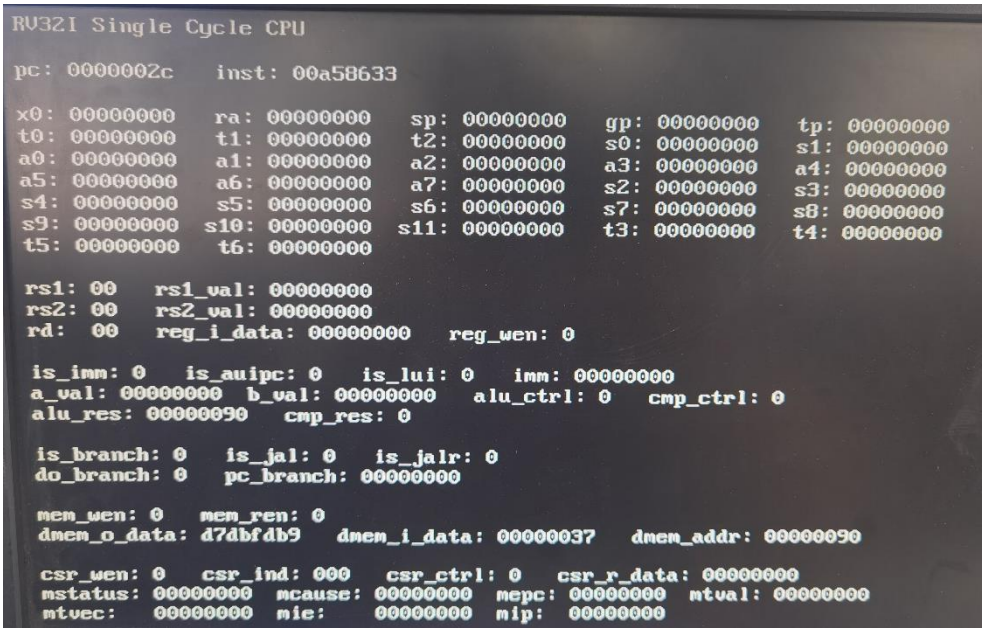
2 简单模块设计（ALU / RegFile / 有限状态机）

2.1 本次实验无需下板验证，只需仿真，结果结果分析进第一部分对应实验二仿真位置

3 使用提供的 IP 核搭建测试框架

3.1 本次下板运行的是 I_mem.coe 中的指令，这些指令实现了计算一个斐波那契数列并将其储存的任务

3.2 VGA 显示：



显示屏输出无模糊的情况，正常输出，对照 I_mem.pdf 中的样例，找到当前情况对应输出。

PC	Machine Code	Basic Code	Original Code	Result
0x2c	0x00A58633	Add x12 x11 x10	Add x12,x11,x10	#x12=00000090

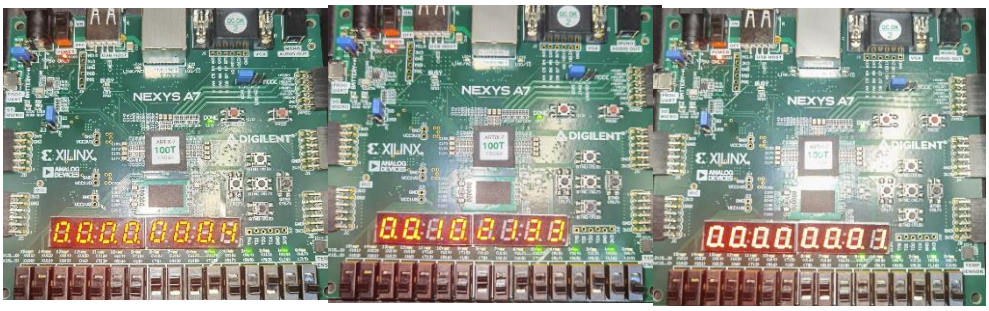
该结果与屏幕上的 pc 等输出均符合。

3.3 第一条指令执行情况

PC	Machine Code	Basic Code	Original Code	Result
0x0	0x00100093	addi x1 x0 1	loop:addi x1,x0,1	#x1=00000001

把开关 SW[7:5]拨到 111，010，100，可以观察到 PC，Machine Code，Result 与上述 I_mem.pdf 中的样例相同。

3.4 第二条指令执行情况



PC	Machine Code	Basic Code	Original Code	Result
0x4	0x00102133	slt x2 x0 x1	slt x2,x0,x1	#x2=00000001

把开关 SW[7:5]拨到 111，010，100，可以观察到 PC，Machine Code，Result 与上述 I_mem.pdf 中的样例相同

3.5 多个循环后指令执行情况

PC	Machine Code	Basic Code	Original Code	Result
0x54	0x014A8B33	add x22 x21 x20	add x22,x21,x20	#x22=0000452f

把开关 SW[7:5]拨到 111，010，100，可以观察到 PC，Machine Code，Result 与上述 I_mem.pdf 中的样例相同

3.6

4 实现乘法器 / 除法器

4.1 该实验无需下板实验,结果见上面第四节的仿真波形分析部分

三、讨论、心得

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。

思考题

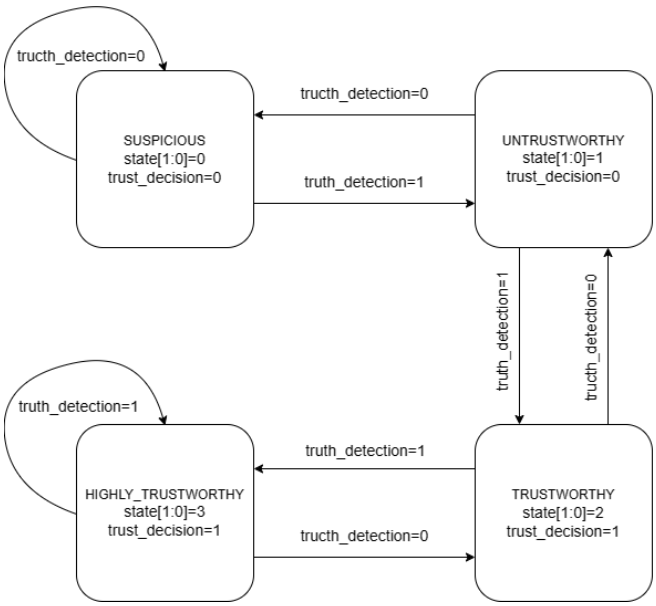
（1） 助教在做《计算机体系结构》实验时，在 Message Window 中看到了下列错误提示：……

阶段：Generate Bitstream

解决方式：修改引脚约束文件为 BTN_X[4:0], BTN_Y[3], BTN_Y[0], SW[15], SW[14], SW[13], SW[7], SW[6], SW[5], SW[4], SW[3], SW[2], SW[1], SW[0], VGA_B[3:0]等引脚添加 I/O 电平标准（IOSTANDARD）。

途径：在上学期课程中遇到过类似的报错

（2） 请根据以上要求，完成信任评估器的状态转移图，你可以纸笔书写并拍照，或使用 drawio 等工具绘图。



（3） 小 Q 同学在 2023 年春夏学期做计算机组成与设计的 ALU 时，为了实现 SRA 指令使用了下面的代码：。。。

来源：查看 verilog 标准 5.5 节

原因：ALU_operation == 4'd7 为无符号数，导致整个三元运算符组表达式的类型变为无符号数，该类型向下推广，导致$signed(A) >>> $signed(B)的两个操作数变为无符号数，>>>进行无符号右移操作，造成错误。

（4） 请结合理论课所学，回答以下问题：

- 双精度浮点数 x, y, z，若 x = -1.5e38, y = 1.5e38, z=1.0

。。。

两者有区别，在浮点数的加法中存在截断，若异号的 x，y 的指数很大且相近且 x+y 比 z 的指数大一点，z 的指数很小，那么先计算 x+y 的结果再加上 z 有可能就不会发生截断损失，但是若先计算 y+z,则必然会发生较大的截断误差（甚至 y+z=y）。

假设使用单精度浮点数，编写以下代码

。。。

-0.000954

0.000000

0.125=b0.001，在转换为浮点数时不会发生截断误差，而 0.1 的二进制为无限小数，转换为浮点数时必然会发生截断误差。

心得

（1） 在查看仿真波形时，可以自定义增加输出分组，调整数据进制，便于展示结果

（2） 安装板卡文件时，需要自己创建没有的文件夹