实验 1 -- 多路选择器与 CPU 辅助模块设计

实验报告

——数字逻辑实验输出模块扩展一

姓名：林逸竹　学号：3160104229　专业：计算机科学与技术

课程名称：计算机组成与设计实验　同组学生姓名：无

实验时间：2018-3-5　实验地点：紫金港东 4-509　指导老师：　施青松，黎金洪

一、实验目的和要求

1.熟练掌握 **EDA** 开发工具和开发流程
2.复习数字逻辑设计实现方法
3.扩展优化逻辑实验基本模块
4.优化计算机系统实现的辅助模块
5.了解计算机硬件系统构成的最基本元件模块

二、实验内容和原理

2.1 实验任务

1. 整理设计逻辑实验输出模块
   多路选择器、基本算术逻辑运算模块等
2. 整理逻辑实验输出的辅助模块
   消除机械抖动模块、通用分频模块
3. 设计存储器 IP 模块
   32 位 ROM、32 位 RAM
4. 设计 **CPU** 调试测试显示通道模块
   在逻辑实验 Framework 基础上重建

2.2 多路器及算术函数、逻辑函数、位扩展

本课程将用到的多路选择器：

2 选 1：5 位，32 位，8 位
4 选 1：5 位，32 位
8 选 1：8 位，32 位

2.3 八数据通路模块

功能：多路信号显示选择控制
用于 CPU 等各类信号的调试和测试
由 1 个或多个 8 选 1 选择器构成

2.4 通用分频模块优化

功能：用于计算机组成实验辅助模块 -> 32 位计数分频输出

2.5 只读存储器 IP 核优化及随机存储器 IP 核优化

只读存储器基本功能：用于 CPU 应用的代码存储器
容量：1024×32bit
使用 FPGA 内部存储器实现 –> Block Memory Generator/ Distributed Memory Generator
随机存储器基本功能：用于 CPU 应用的数据或代码存储器
容量：1024×32bit
用 FPGA 内部存储器实现 -> Block Memory Generator

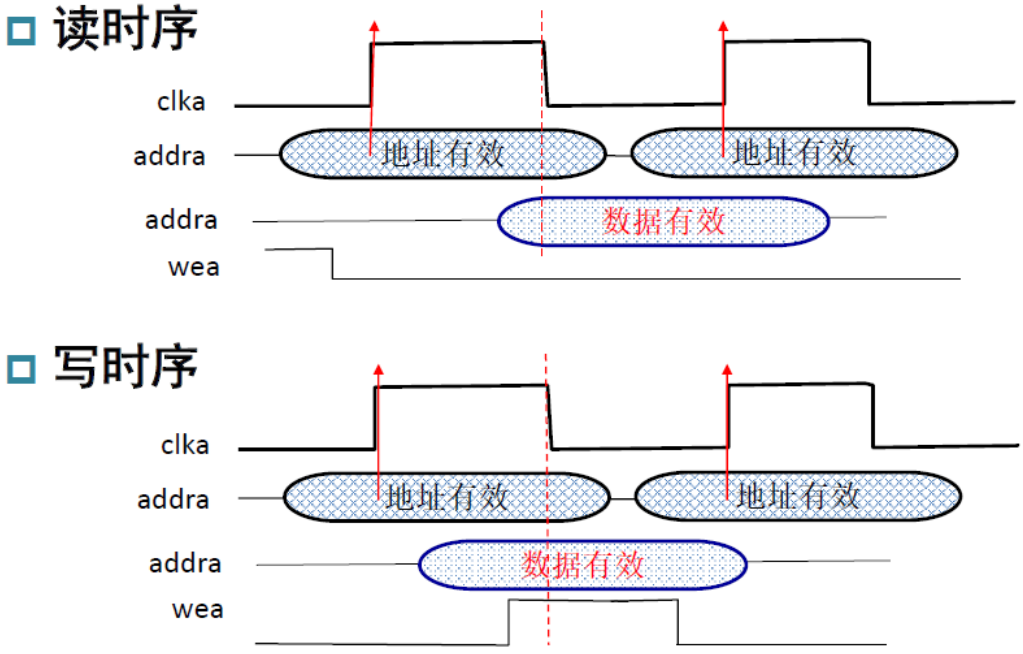


Figure 1 Block Memory 时序

三、主要仪器设备

3.1 实验设备

1. 计算机（Intel Core i5 以上，4GB 内存以上）系统
2. 计算机软硬件课程贯通教学实验系统
3. Xilinx ISE14.4 及以上开发工具

3.2 材料

无

四、实验实现方法、步骤与调试

4.1 OExp01-Element

设计、整理和优化逻辑课实验输出基本逻辑模块

4.1.1 MUX2T1

Notes：2T1 选择器代码基本相同，以 MUX2T1_32 为例并进行仿真测试。

代码如下：

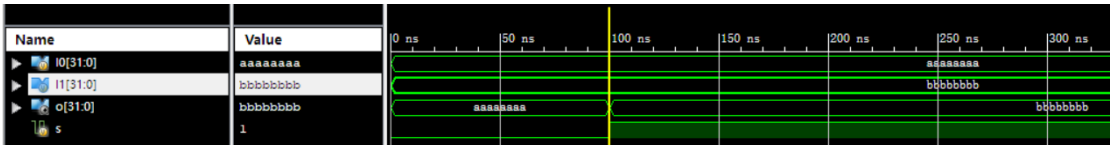
```
module  MUX2T1_32(input[31:0]I0,
                  input[31:0]I1,
                  input s,
                  output[31:0]o
                   );

    assign o = s?I1:I0;          ////32 位 2 选一,I0、I1 对应选择通道 0、1

endmodule
```

测试代码如下：

```
module MUX2T1_32_Test;

    // Inputs
    reg [31:0] I0;
    reg [31:0] I1;
    reg s;

    // Outputs
    wire [31:0] o;

    // Instantiate the Unit Under Test (UUT)
    MUX2T1_32 uut (
        .I0(I0),
        .I1(I1),
        .s(s),
        .o(o)
    );

    initial begin
        // Initialize Inputs
        I0 = 32'hAAAAAAAA;
        I1 = 32'hBBBBBBBB;
```

```
        s = 0;
        #100;
        s = 1;
    end

endmodule
```

测试结果如图：



4.1.2 MUX4T1

Notes：4T1 选择器代码基本相同，以 MUX4T1_32 为例并进行仿真测试。

代码如下：

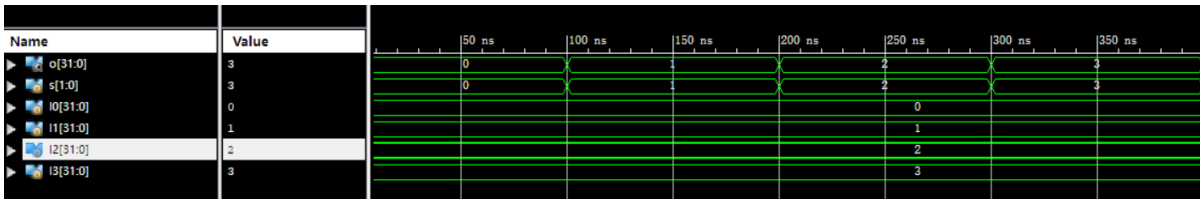
```
module   MUX4T1_32(input [1:0]s,
                    input [31:0]I0,
                    input [31:0]I1,
                    input [31:0]I2,
                    input [31:0]I3,
                    output reg[31:0]o
                    );

        always@*              //32 位 4 选一,I0、I1、I2、I3 对应选择通道 0、
1、2、3
           case(s)
               2'b00: o<=I0;
               2'b01: o<=I1;
               2'b10: o<=I2;
               2'b11: o<=I3;
           endcase

endmodule
```

测试代码如下：

```
module MUX4T1_32_Test;

    // Inputs
    reg [1:0] s;
    reg [31:0] I0;
    reg [31:0] I1;
    reg [31:0] I2;
    reg [31:0] I3;

    // Outputs
    wire [31:0] o;

    // Instantiate the Unit Under Test (UUT)
    MUX4T1_32 uut (
        .s(s),
        .I0(I0),
        .I1(I1),
        .I2(I2),
        .I3(I3),
        .o(o)
    );
    initial begin
        // Initialize Inputs
        s = 0;
```

```
        I0 = 0;
        I1 = 1;
        I2 = 2;
        I3 = 3;
        #100;
        s = 1;
        #100;
        s = 2;
        #100;
        s = 3;
        #100;
        s = 0;
    end

endmodule
```

测试结果如图：



4.1.3 MUX8T1

Notes：8T1 选择器代码基本相同，以 MUX8T1_32 为例并进行仿真测试。

代码如下：

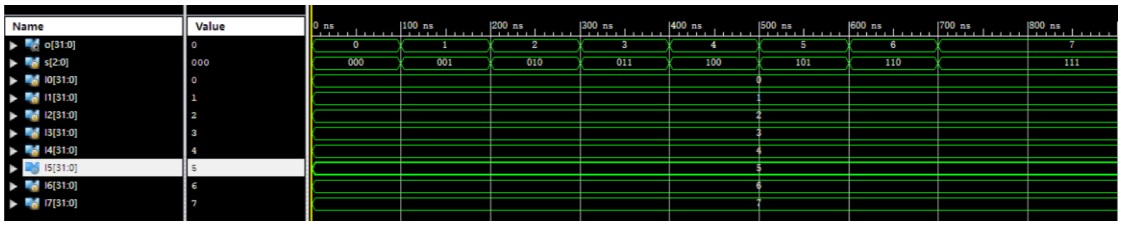
```
module  MUX8T1_32(input [2:0]s,
                  input [31:0]I0,
                  input [31:0]I1,
                  input [31:0]I2,
                  input [31:0]I3,
                  input [31:0]I4,
                  input [31:0]I5,
                  input [31:0]I6,
                  input [31:0]I7,

                  output reg[31:0]o
                  );

        always@*          //32 位 8 选一,I0、I1、I2、……对应选择通道 0、1、
2、……
           case(s)
               3'b000: o<=I0;
               3'b001: o<=I1;
               3'b010: o<=I2;
               3'b011: o<=I3;
               3'b100: o<=I4;
               3'b101: o<=I5;
               3'b110: o<=I6;
               3'b111: o<=I7;
           endcase

endmodule
```

测试代码如下：

```
module MUX8T1_32_Test;

    // Inputs
    reg [2:0] s;
    reg [31:0] I0;
```

```
    reg [31:0] I1;
    reg [31:0] I2;
    reg [31:0] I3;
    reg [31:0] I4;
    reg [31:0] I5;
    reg [31:0] I6;
    reg [31:0] I7;

    // Outputs
    wire [31:0] o;

    // Instantiate the Unit Under Test (UUT)
    MUX8T1_32 uut (
        .s(s),
        .I0(I0),
        .I1(I1),
        .I2(I2),
        .I3(I3),
        .I4(I4),
        .I5(I5),
        .I6(I6),
        .I7(I7),
        .o(o)
    );

    initial begin
        // Initialize Inputs
        s = 0;
        I0 = 0;
        I1 = 1;
        I2 = 2;
        I3 = 3;
        I4 = 4;
        I5 = 5;
        I6 = 6;
        I7 = 7;
        #100;
        s = 1;
        #100;
        s = 2;
        #100;
        s = 3;
        #100;
        s = 4;
        #100;
        s = 5;
        #100;
        s = 6;
        #100;
        s = 7;
    end

endmodule
```

测试结果如图：



4.1.4 32 位加法器 add32(无进位)

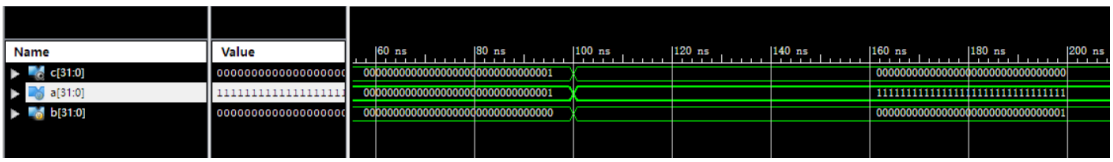
代码如下：

```
module add_32(input [31:0] a,
                input [31:0] b,
                output [31:0]c
                );

    assign c = a + b;         //无进位

endmodule
```

测试代码如下：

```
module add_32_Test;

    // Inputs
    reg [31:0] a;
    reg [31:0] b;

    // Outputs
    wire [31:0] c;

    // Instantiate the Unit Under Test (UUT)
    add_32 uut (
        .a(a),
        .b(b),
        .c(c)
    );

    initial begin
        // Initialize Inputs
        a = 1;
        b = 0;
        #100;
        a = 32'hFFFFFFFF;
        b = 1;
        #100;
    end

endmodule
```

测试结果：



4.1.5 32 位加减器 ADC

代码如下：

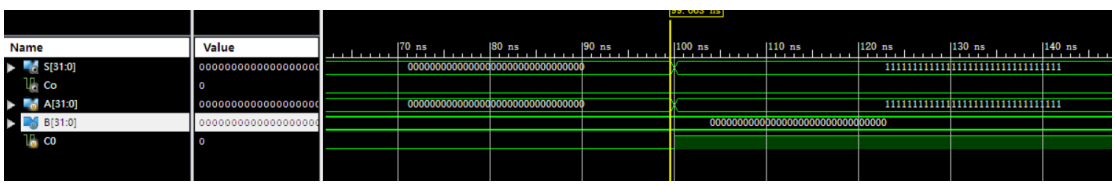
```
module ADC32(input [31:0] A,              //带进位的 32 位加减器，考虑无符号
数兼容。在设计 ALU 的 slt 指令上有区别
                input [31:0] B,
```

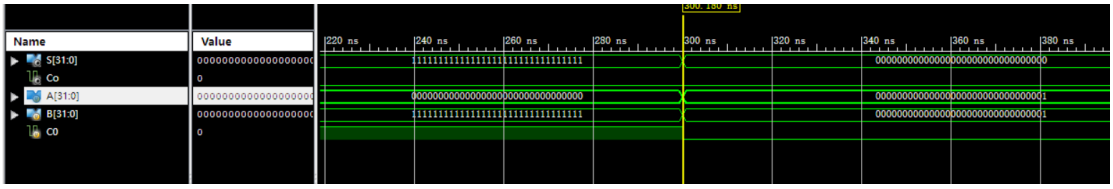
```
                input C0,     //最低进位输入
                output [31:0] S,
                output Co     //修改逻辑符号，将进位分开

                 );

    wire B_Notation = C0 ^ 1'b0;

    assign {Co, S} = B_Notation? A + B : A - B;

endmodule
```

测试代码如下：

```
module ADC32_test;

    // Inputs
    reg [31:0] A;
    reg [31:0] B;
    reg C0;

    // Outputs
    wire [31:0] S;
    wire Co;

    // Instantiate the Unit Under Test (UUT)
    ADC32 uut (
        .A(A),
        .B(B),
        .C0(C0),
        .S(S),
        .Co(Co)
    );

    initial begin
        // Initialize Inputs
        A = 0;
        B = 0;
        C0 = 0;
        #100;
        C0 = 1;
        A = 32'hFFFFFFFF;
        #100;
        A = 0;
        B = 32'hFFFFFFFF;
        #100;
        A = 1;
        B = 1;
        C0 = 0;

    end

endmodule
```

测试结果：





4.1.6 与运算

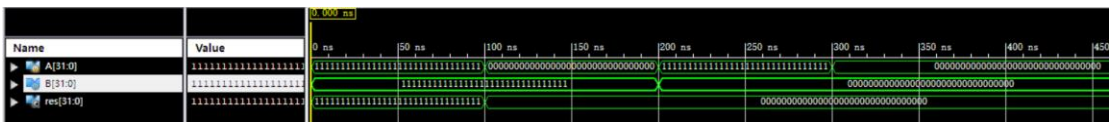
代码如下：

```
module and32(input [31:0] A,
                input [31:0] B,
                output [31:0] res
                );

    assign res = A & B;      //32 位与

endmodule
```

测试代码如下：

```
module and32_test;

    // Inputs
    reg [31:0] A;
    reg [31:0] B;

    // Outputs
    wire [31:0] res;

    // Instantiate the Unit Under Test (UUT)
    and32 uut (
        .A(A),
        .B(B),
        .res(res)
    );

    initial begin
        // Initialize Inputs
        A = 32'hFFFFFFFF;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'b0;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'hFFFFFFFF;
        B = 32'b0;
        #100;
        A = 32'b0;
        B = 32'b0;

        // Add stimulus here

    end

endmodule
```

测试结果：



4.1.7 或运算

代码如下：

```
module or32(input [31:0] A,
              input [31:0] B,
              output [31:0] res
              );

    assign res = A | B;      //32 位数或

endmodule
```

测试代码如下：

```
module or Test;

    // Inputs
    reg [31:0] A;
    reg [31:0] B;

    // Outputs
    wire [31:0] res;

    // Instantiate the Unit Under Test (UUT)
    or32 uut (
        .A(A),
        .B(B),
        .res(res)
    );

    initial begin
        // Initialize Inputs
        A = 32'hFFFFFFFF;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'b0;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'hFFFFFFFF;
        B = 32'b0;
        #100;
        A = 32'b0;
        B = 32'b0;
    end

endmodule
```

测试结果：

4.1.8 或非运算

代码如下：

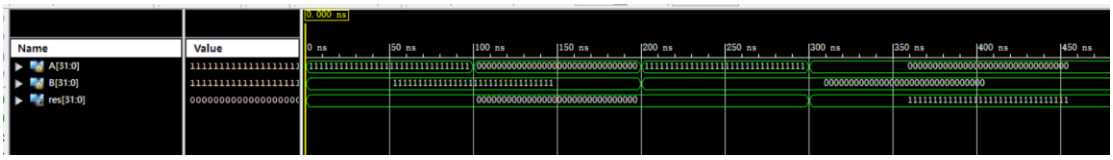
```
module nor32(input [31:0] A,
               input [31:0] B,
               output [31:0] res
               );
       assign res = ~(A|B);         //32 位或非
```

```
endmodule
```

测试代码如下：

```
module nor_Test;

    // Inputs
    reg [31:0] A;
    reg [31:0] B;

    // Outputs
    wire [31:0] res;

    // Instantiate the Unit Under Test (UUT)
    nor32 uut (
        .A(A),
        .B(B),
        .res(res)
    );

    initial begin
        // Initialize Inputs
        A = 32'hFFFFFFFF;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'b0;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'hFFFFFFFF;
        B = 32'b0;
        #100;
        A = 32'b0;
        B = 32'b0;

    end

endmodule
```

测试结果：



4.1.9 异或运算

代码如下：

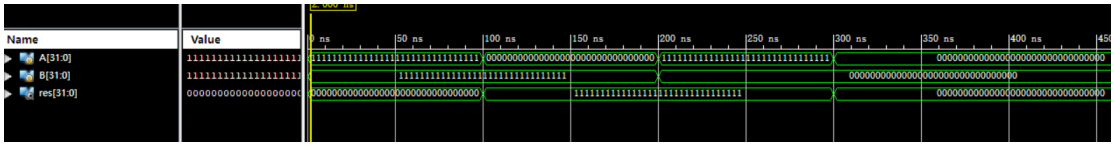
```
module xor32(input [31:0] A,
               input [31:0] B,
               output [31:0] res
              );

    assign res = A ^ B;      //32 位或

endmodule
```

测试代码如下：

```
module xor_Test;

    // Inputs
    reg [31:0] A;
    reg [31:0] B;

    // Outputs
```

```
    wire [31:0] res;

    // Instantiate the Unit Under Test (UUT)
    xor32 uut (
        .A(A),
        .B(B),
        .res(res)
    );

    initial begin
        // Initialize Inputs
        A = 32'hFFFFFFFF;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'b0;
        B = 32'hFFFFFFFF;
        #100;
        A = 32'hFFFFFFFF;
        B = 32'b0;
        #100;
        A = 32'b0;
        B = 32'b0;

    end

endmodule
```

测试结果：



4.1.10 位或运算

代码如下：

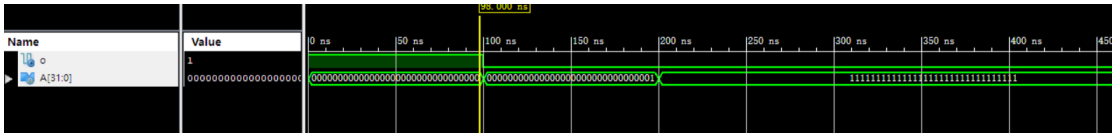
```
module  or_bit_32(input [31:0] A,
                      output o
                    );

    assign o = ~(|A);     //32 位数“位或”输出（32 位数=0，输出 o=1）

endmodule
```

测试代码如下：

```
module or_bit_32_Test;

    // Inputs
    reg [31:0] A;

    // Outputs
    wire o;

    // Instantiate the Unit Under Test (UUT)
    or_bit_32 uut (
        .A(A),
        .o(o)
    );

    initial begin
        // Initialize Inputs
        A = 0;
```

```
        #100;
        A = 1;
        #100;
        A = 32'hFFFFFFFF;
    end

endmodule
```

测试结果：



4.1.11 右移

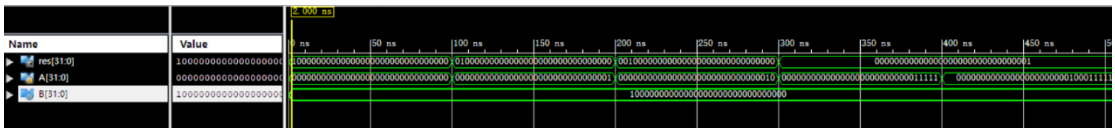
代码如下：

```
module srl32(input [31:0] A,
                input [31:0] B,
                output [31:0] res
               );

    assign res = B>>A[4:0];  //32 位数右移。移动位数为 A 的后五位

endmodule
```

测试代码如下：

```
module srl32_Test;

    // Inputs
    reg [31:0] A;
    reg [31:0] B;

    // Outputs
    wire [31:0] res;

    // Instantiate the Unit Under Test (UUT)
    srl32 uut (
        .A(A),
        .B(B),
        .res(res)
    );

    initial begin
        // Initialize Inputs
        A = 0;
        B = 32'h80000000;
        #100;
        A = 1;
        #100;
        A = 2;
        #100;
        A = 32'h0000001F;
        #100;
        A = 32'h0000011F;

    end

endmodule
```

测试结果：



4.1.12 符号位扩展

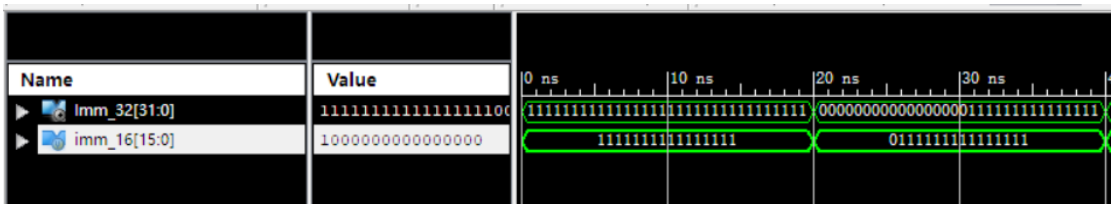
代码如下：

```
module Ext_32(input [15:0] imm_16,
              output[31:0] Imm_32
             );

    assign Imm_32 = {{16{imm_16[15]}},imm_16} ;          //扩展为32位
符号数

endmodule
```

测试代码如下：

```
module Ext_32_Test;

    // Inputs
    reg [15:0] imm_16;

    // Outputs
    wire [31:0] Imm_32;

    // Instantiate the Unit Under Test (UUT)
    Ext_32 uut (
        .imm_16(imm_16),
        .Imm_32(Imm_32)
    );

    initial begin
        // Initialize Inputs
        imm_16 = 16'hFFFF;
        #20;
        imm_16 = 16'h7FFF;
        #20;
        imm_16 = 16'h8000;

    end

endmodule
```

测试结果：



4.2 OExp01-MUX

功能：多路信号显示选择控制。

代码如下：

```
module Multi_8CH32(
    input clk,
    input rst,
```

```
    input EN,
    input [2:0] Test,
    input [63:0] point_in,
    input [63:0] blink_in,
    input [31:0] Data0,
    input [31:0] Test_data1,
    input [31:0] Test_data2,
    input [31:0] Test_data3,
    input [31:0] Test_data4,
    input [31:0] Test_data5,
    input [31:0] Test_data6,
    input [31:0] Test_data7,
    output [7:0] point_out,
    output [7:0] blink_out,
    output [31:0] Disp_num
    );
reg  [31:0] disp_data= 32'hAA5555AA;
reg [7:0] cpu_blink = 8'b11111111, cpu_point = 8'b00000000;

MUX8T1_32 MUX1_DispData(
                            .I0(disp_data),
                            .I1(Test_data1),
                            .I2(Test_data2),
                            .I3(Test_data3),
                            .I4(Test_data4),
                            .I5(Test_data5),
                            .I6(Test_data6),
                            .I7(Test_data7),
                            .S(Test),
                            .O(Disp_num)
                            );

MUX8T1_8 MUX2_Blink(
                          .I0(cpu_blink),
                          .I1(blink_in[15:8]),
                          .I2(blink_in[23:16]),
                          .I3(blink_in[31:24]),
                          .I4(blink_in[39:32]),
                          .I5(blink_in[47:40]),
                          .I6(blink_in[55:48]),
                          .I7(blink_in[63:56]),
                          .S(Test),
                          .O(blink_out)
                          );
MUX8T1_8 MUX3_Point(.I0(cpu_point),
                        .I1(point_in[15:8]),
                        .I2(point_in[23:16]),
                        .I3(point_in[31:24]),
                        .I4(point_in[39:32]),
                        .I5(point_in[47:40]),
                        .I6(point_in[55:48]),
                        .I7(point_in[63:56]),
                        .S(Test),
                        .O(point_out)
                        );
always @(posedge clk) begin
if (EN) begin
disp_data <= Data0;
cpu_blink<=blink_in[7:0];
cpu_point<=point_in[7:0];
end
```

```
else begin
disp_data <= Disp_num;
cpu_blink<=cpu_blink;
cpu_point<=cpu_point;
end
end

endmodule
```

时钟代码如下：

```
module clk_div(clk,
             rst,
             SW2,
             clkdiv,
             Clk_CPU);

   input clk;
   input rst;
    input SW2;
   output reg [31:0] clkdiv;
   output Clk_CPU;

    always @ (posedge clk or posedge rst) begin
        if (rst) clkdiv <= 0;
        else clkdiv <= clkdiv + 1'b1;
    end

    assign Clk_CPU=(SW2)?clkdiv[24]:clkdiv[2];

endmodule
```

五、实验结果与分析

完成逻辑实验输出各个小模块的优化。
重新搭建 Framework 框架，实现顶层模块的构建，学会导入 RAM 和 ROM。

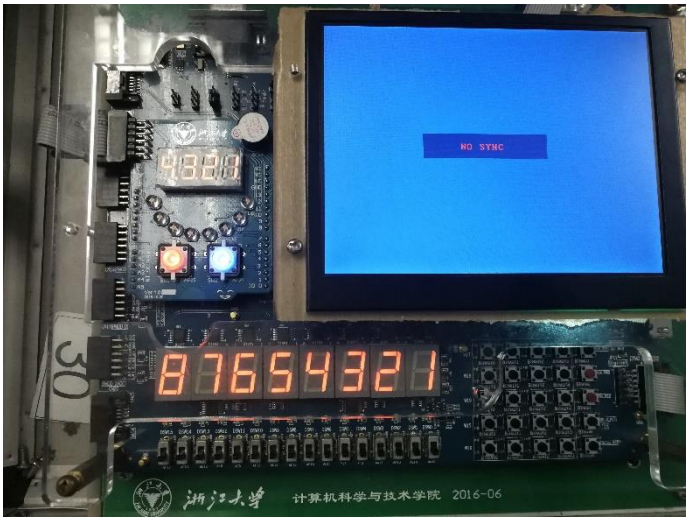


Figure 2 结果图 1(同实验二)

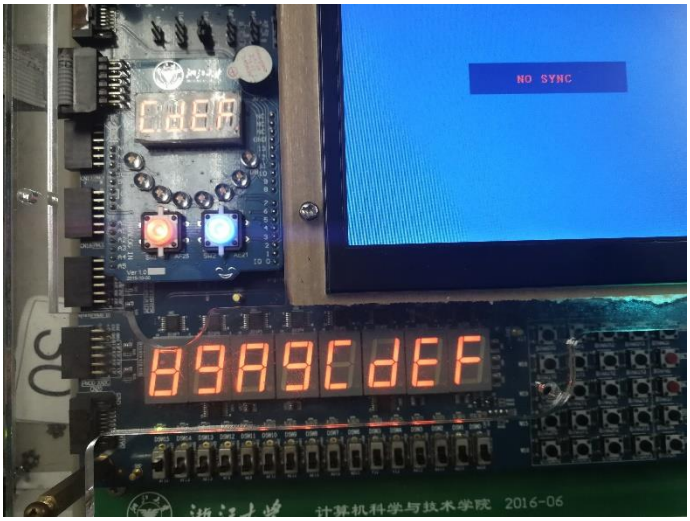


Figure 3 结果图 2(同实验二)

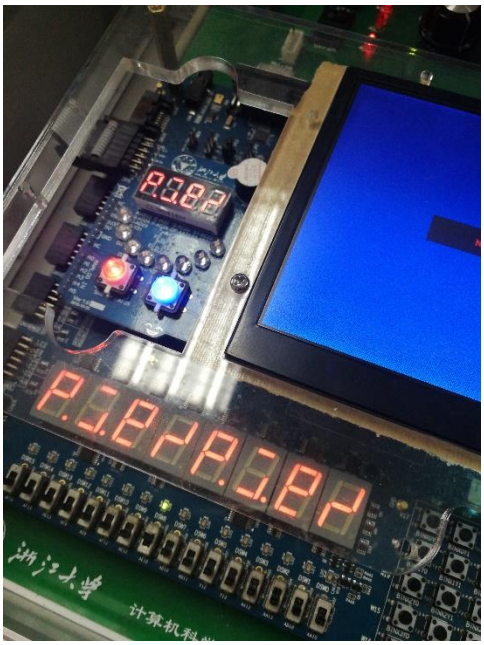


Figure 4 结果图 3(同实验二)

六、讨论、心得

这个实验相对简单，但在将以前使用原理图表示的模块转换为 Verilog 语言来书写还是存在有一些值得思考的地方，会更加注重原理的理解。