# 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

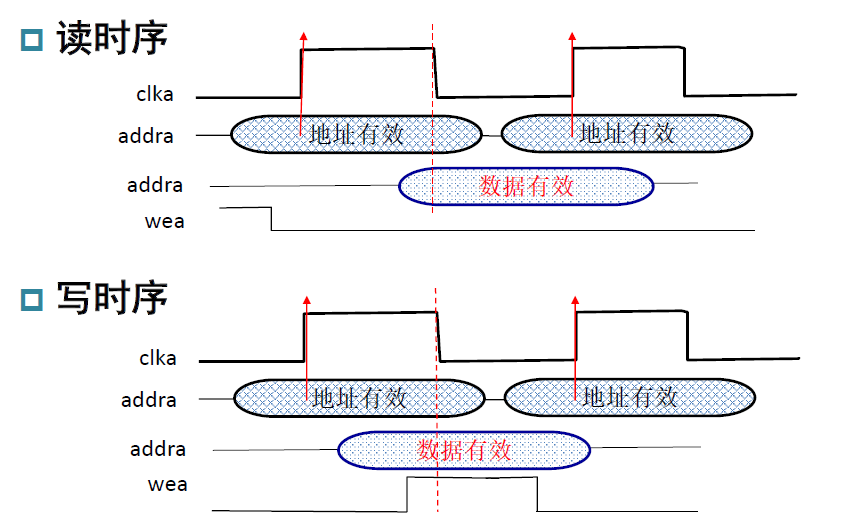


Figure 1 Block Memory时序

# 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。

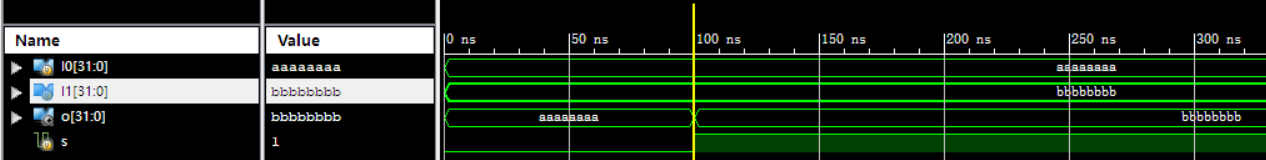
代码如下：

|  |
| --- |
| 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。

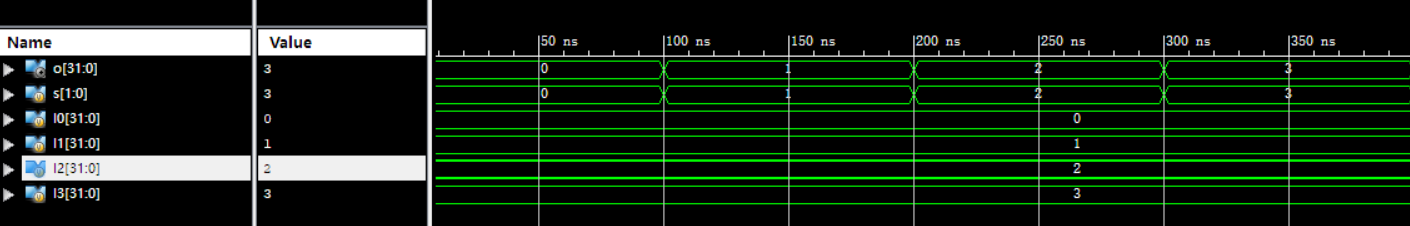
代码如下：

|  |
| --- |
| 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。

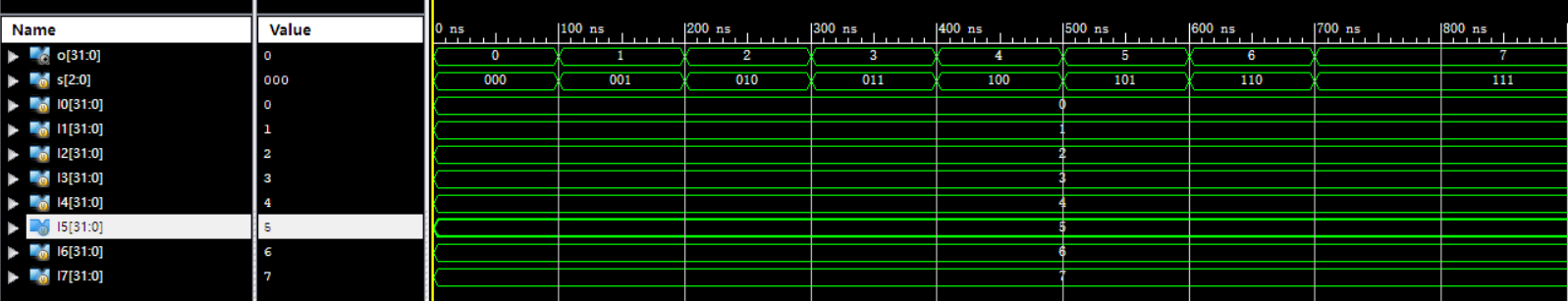
代码如下：

|  |
| --- |
| 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)

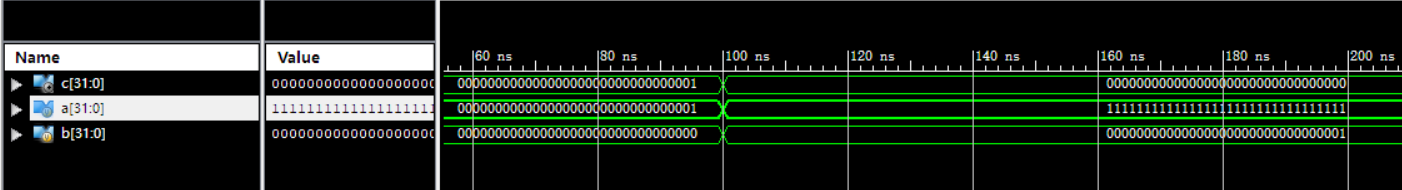
代码如下：

|  |
| --- |
| 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

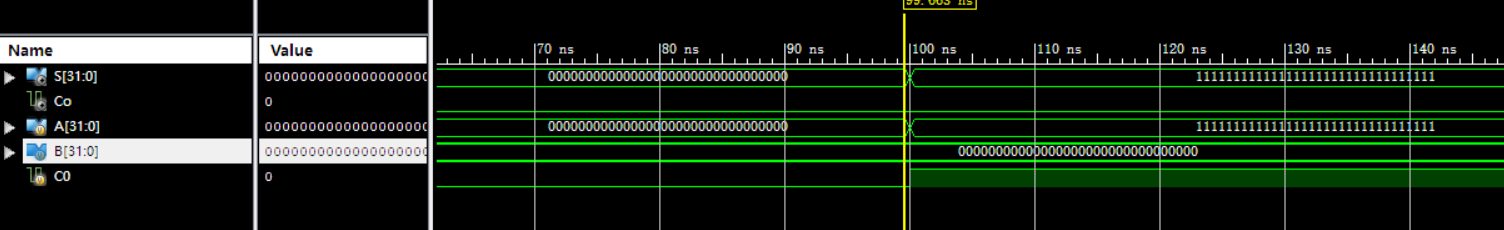
代码如下：

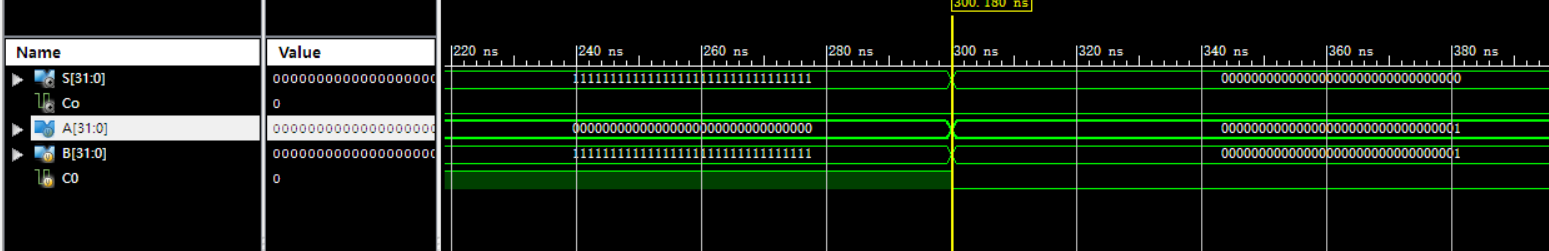
|  |
| --- |
| module ADC32(input [31:0] A, //带进位的32位加减器，考虑无符号数兼容。在设计ALU的slt指令上有区别  input [31:0] B,  input C0, //最低进位输入  output [31:0] S,  output Co //修改逻辑符号，将进位分开  );    wire B\_Notation = C0 ^ 1'b0;  assign {Co, S} = B\_Notation? A + B : A - B;    endmodule |

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

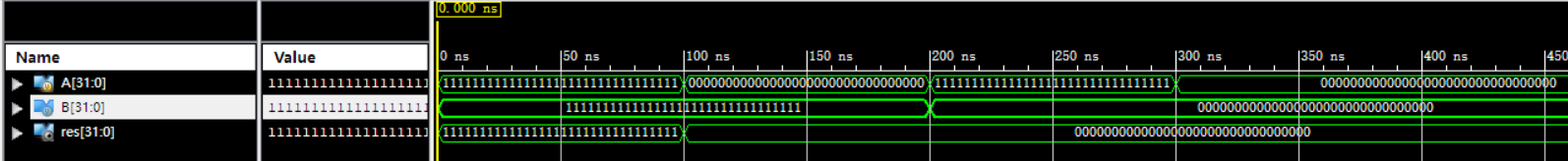
代码如下：

|  |
| --- |
| 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

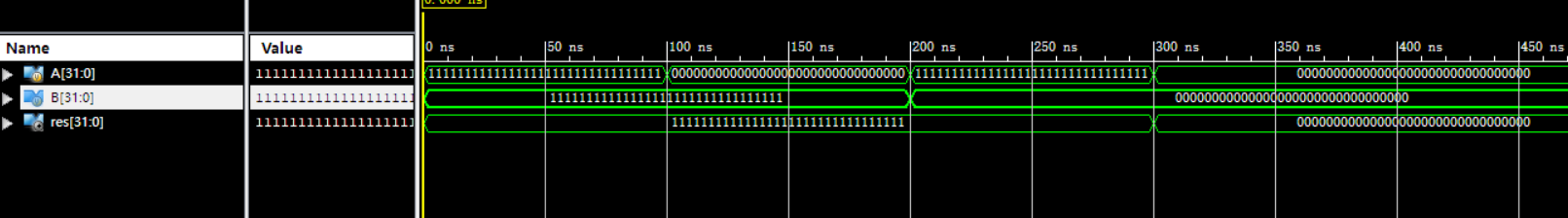
代码如下：

|  |
| --- |
| 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

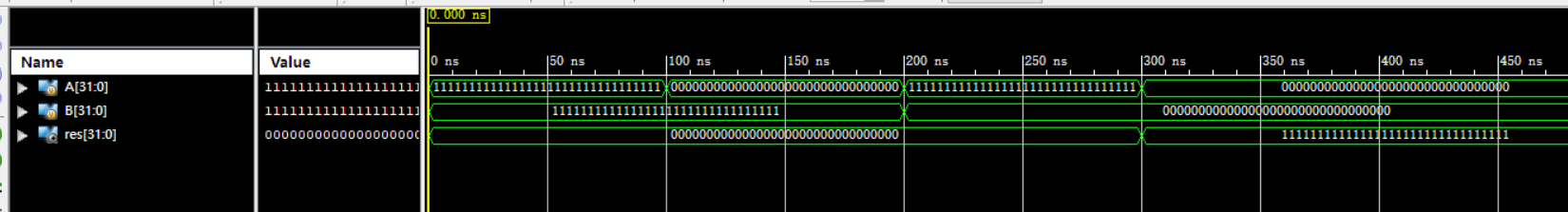
代码如下：

|  |
| --- |
| 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

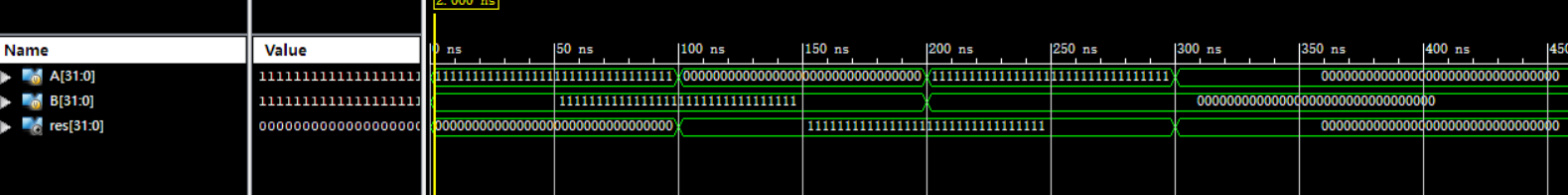
代码如下：

|  |
| --- |
| 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

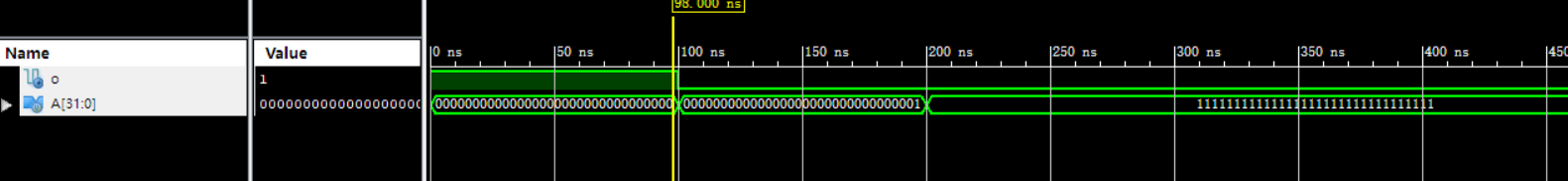
代码如下：

|  |
| --- |
| 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

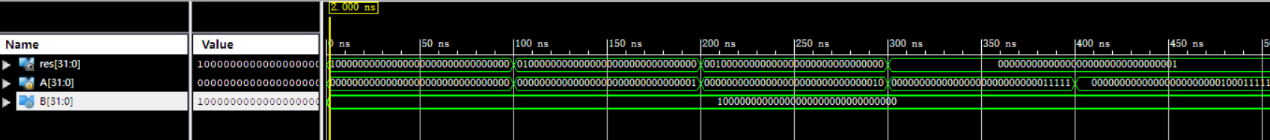
代码如下：

|  |
| --- |
| 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

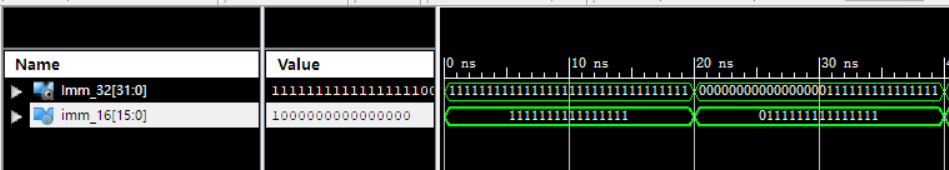
代码如下：

|  |
| --- |
| 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。

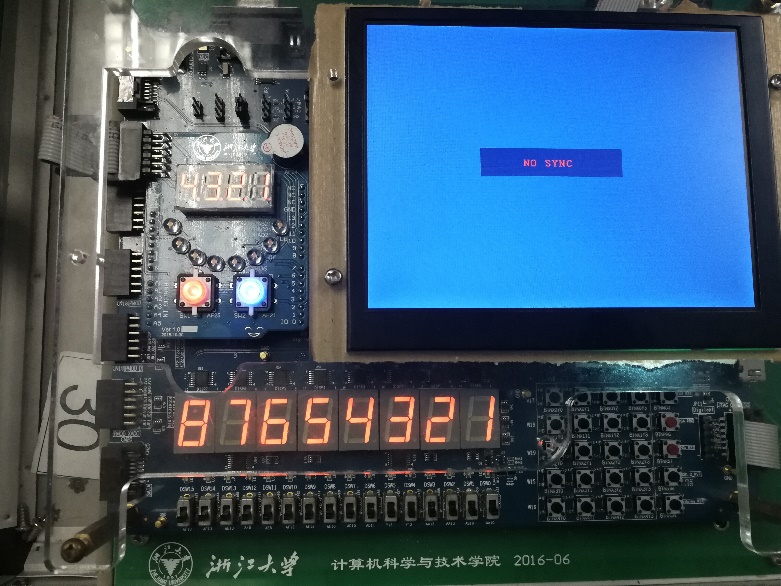


Figure 2结果图1(同实验二)

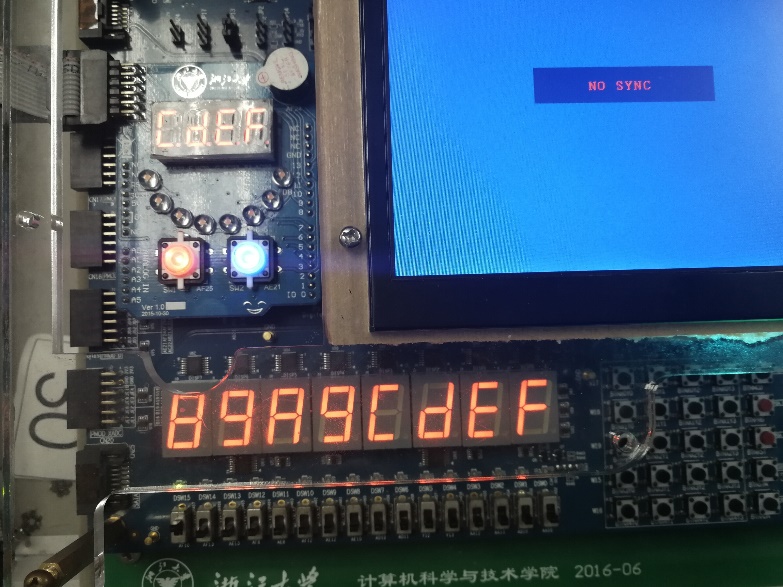


Figure 3结果图2(同实验二)

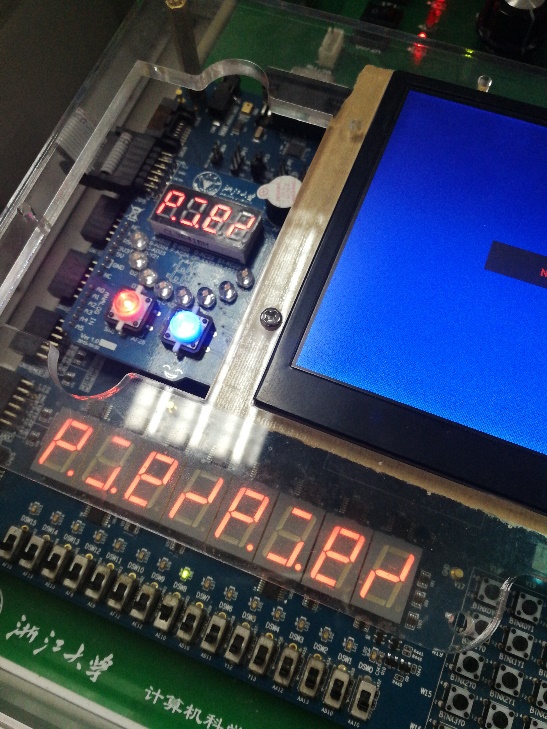


Figure 4结果图3(同实验二)

# 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。