# 实验11—寄存器及寄存器堆设计实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名： | 赵冰骞 | | 学号： | 3090103420 | | | 专业： | 理科试验班 | | |
| 课程名称： | | 逻辑与计算机设计基础实验 | | | 同组学生姓名： | 朱里 | | | |
| 实验时间： | | 2010-12-17 | 实验地点： | | 紫金港东4-509 | | 指导老师： | | 王总辉 | |

# 一、实验目的和要求

1. 掌握寄存器、锁存器的工作原理及设计方法
2. 掌握寄存器堆的工作原理及设计方法
3. 了解计算机中寄存器及寄存器堆的概念

# 二、实验内容和原理

## 2.1 寄存器

* 寄存器是计算机常用的基本器件
  + 由同一信号控制的一组相互并联的触发器构成
  + 包含存储、处理和传输，可构成寄存器组使用
  + 边沿D触发器构成的8位基本寄存器的结构如下



图表1 8位基本寄存器结构图

## 2.2 锁存器

* 锁存器由多个一位锁存器/触发器并联构成
  + - 一般采用电平控制锁存：
    - 高电平时置入数据
    - 低电平时锁存数据，并指示数据稳定
    - 带使能端的RS触发器实现的锁存器

## 2.3 寄存器组

* 寄存器组是多个寄存器组成的集合
  + 寄存器写：寄存器地址→变量译码器
  + 寄存器读：寄存器地址→数据选择器



图表2 寄存器组

# 三、主要仪器设备

1. 装有ISE的计算机系统 1台
2. Spartan-III 开发板 1套

# 四、操作方法与实验步骤

## 4.1 设计含由16个16位寄存器构成的寄存器堆

1．寄存器模块

|  |
| --- |
| `timescale 1ns / 1ps  module regfile(wclk, rclk, address, data\_in, data\_out);  input wire wclk,rclk;  input wire[3:0] address;  input wire[15:0] data\_in;  output wire[15:0] data\_out;  wire[15:0] regQ0, regQ1, regQ2, regQ3,  regQ4, regQ5, regQ6, regQ7,  regQ8, regQ9, regQ10,regQ11,  regQ12,regQ13,regQ14,regQ15,  clk, Yi;  assign clk[15:0] = {16{wclk}} & Yi;    decoder\_4\_16 M0(address[3:0],wclk,Yi);  mux\_16\_1 M1(rclk, address[3:0],  regQ0, regQ1, regQ2, regQ3,  regQ4, regQ5, regQ6, regQ7,  regQ8, regQ9, regQ10,regQ11,  regQ12,regQ13,regQ14,regQ15,  data\_out);    register16 R0(clk[0], data\_in, regQ0),  R1(clk[1], data\_in, regQ1),  R2(clk[2], data\_in, regQ2),  R3(clk[3], data\_in, regQ3),  R4(clk[4], data\_in, regQ4),  R5(clk[5], data\_in, regQ5),  R6(clk[6], data\_in, regQ6),  R7(clk[7], data\_in, regQ7),  R8(clk[8], data\_in, regQ8),  R9(clk[9], data\_in, regQ9),  R10(clk[10], data\_in, regQ10),  R11(clk[11], data\_in, regQ11),  R12(clk[12], data\_in, regQ12),  R13(clk[13], data\_in, regQ13),  R14(clk[14], data\_in, regQ14),  R15(clk[15], data\_in, regQ15);  endmodule  module register16(clk, Di, Do );  input wire clk;  input wire [15:0] Di;  output reg [15:0] Do;    always @(posedge clk)  Do <= Di;  endmodule |

2．4-16译码器

|  |
| --- |
| `timescale 1ns / 1ps  module decoder\_4\_16(  input wire[3:0] address,  input wire wclk,  output reg[15:0] Yi  );  always @\*begin  if(wclk==1'b1)  case(address[3:0])  4'b0000:Yi=16'h0001;  4'b0001:Yi=16'h0002;  4'b0010:Yi=16'h0004;  4'b0011:Yi=16'h0008;  4'b0100:Yi=16'h0010;  4'b0101:Yi=16'h0020;  4'b0110:Yi=16'h0040;  4'b0111:Yi=16'h0080;  4'b1000:Yi=16'h0100;  4'b1001:Yi=16'h0200;  4'b1010:Yi=16'h0400;  4'b1011:Yi=16'h0800;  4'b1100:Yi=16'h1000;  4'b1101:Yi=16'h2000;  4'b1110:Yi=16'h4000;  4'b1111:Yi=16'h8000;  endcase  else  Yi=16'h0000;  end  endmodule |

3．16-1数据选择器

|  |
| --- |
| `timescale 1ns / 1ps  module mux\_16\_1(rclk, address,  regQ0, regQ1, regQ2, regQ3,  regQ4, regQ5, regQ6, regQ7,  regQ8, regQ9, regQ10,regQ11,  regQ12,regQ13,regQ14,regQ15,  Dataout  );  input wire[3:0] address;  input wire rclk;  input wire [15:0] regQ0, regQ1, regQ2, regQ3,  regQ4, regQ5, regQ6, regQ7,  regQ8, regQ9, regQ10,regQ11,  regQ12,regQ13,regQ14,regQ15;  output reg[15:0] Dataout;    always @\*begin  if(rclk==1'b1)  case(address[3:0])  4'b0000:Dataout=regQ0;  4'b0001:Dataout=regQ1;  4'b0010:Dataout=regQ2;  4'b0011:Dataout=regQ3;  4'b0100:Dataout=regQ4;  4'b0101:Dataout=regQ5;  4'b0110:Dataout=regQ6;  4'b0111:Dataout=regQ7;  4'b1000:Dataout=regQ8;  4'b1001:Dataout=regQ9;  4'b1010:Dataout=regQ10;  4'b1011:Dataout=regQ11;  4'b1100:Dataout=regQ12;  4'b1101:Dataout=regQ13;  4'b1110:Dataout=regQ14;  4'b1111:Dataout=regQ15;  endcase  else  Dataout=16'h0000;  end  endmodule |

## 4.2 将寄存器组模块增加到xCal中

1. 修改Top模块

|  |
| --- |
| `timescale 1ns / 1ps  module top(  input wire clk,  input wire [3:0]btn\_in,  input wire[7:0]switch,  output wire [3:0]anode,  output wire [7:0]segment  );    //variable definition: op1, op2,disp\_num…  reg [15:0] display\_num;  wire [15:0] op1,op2,disp\_counter,disp\_clock;  wire [3:0] btn\_out;  wire [15:0] result, reg\_out;  wire clk\_1s;  pbdebounce b0(clk, btn\_in[0], btn\_out[0]);  pbdebounce b1(clk, btn\_in[1], btn\_out[1]);  pbdebounce b2(clk, btn\_in[2], btn\_out[2]);  pbdebounce b3(clk, btn\_in[3], btn\_out[3]);  display m0(clk, 0, display\_num, 4'b1111, anode, segment); //display module  calculate\_result m1(btn\_out[3:0], op1, op2, result); //calculate result  create\_oprands m2(switch[7:0], btn\_out[3:0], result, op1, op2, reg\_out); //generate op1/op2  counter\_1s m3(clk,clk\_1s);  counter\_16bit\_rev m4((clk\_1s&btn\_out[0]),switch[2],disp\_counter);  clock\_24h m5((clk\_1s & btn\_out[1]), disp\_clock);    always @\* begin  case (switch[1:0])  2'b01:begin //operand 1  display\_num = op1;  end  2'b10:begin //operand 2  display\_num = op2;  end  2'b00:begin //result  if(btn\_out[1]==1 && switch[3] == 1)  display\_num = reg\_out;  else  display\_num = result;  end  2'b11:begin //counter  if(btn\_out[1:0] == 2'b01)  display\_num = disp\_counter;  else if(btn\_out[1:0] == 2'b10)  display\_num = disp\_clock;  end  endcase  end  endmodule |

2．修改create\_oprands模块

|  |
| --- |
| `timescale 1ns / 1ps  module create\_oprands(  input wire [7:0] switch,  input wire [3:0] btn,  input wire [15:0] reg\_in3,  output reg [15:0] op\_out1,op\_out2,  output wire [15:0] reg\_out3  );  wire [15:0]out\_shift1, out\_shift2, out\_inc1, out\_inc2, reg\_out1, reg\_out2;  reg [15:0] reg\_in1, reg\_in2;    btn\_increment m0(btn[3:0], switch[3:0], out\_inc1, out\_inc2);  shift\_reg\_16 m1(btn[2:0], switch[3:0], switch[7:4], out\_shift1, out\_shift2);  regfile\_16 m2(btn[0], btn[1], switch[1:0], switch[7:4], reg\_in1, reg\_in2, reg\_in3,  reg\_out1, reg\_out2, reg\_out3);    always @\* begin  if(switch[1:0] == 2'b01)  begin  case(switch[3:2])  2'b00:op\_out1 = out\_inc1;  2'b01:op\_out1 = out\_shift1;  default: if(btn[1] == 1'b1) op\_out1 = reg\_out1;  endcase  reg\_in1 = op\_out1;  end  else if(switch[1:0] == 2'b10)  begin  case(switch[3:2])  2'b00:op\_out2 = out\_inc2;  2'b01:op\_out2 = out\_shift2;  default: if(btn[1] == 1'b1) op\_out2 = reg\_out2;  endcase  reg\_in2 = op\_out2;  end  end  endmodule |

3．增加regfile\_16模块

|  |
| --- |
| `timescale 1ns / 1ps  module regfile\_16 (input wire wclk, rclk, input wire [1:0]switch, input wire [3:0]address,  input wire [15:0]op1\_in, input wire [15:0]op2\_in, input wire [15:0]result\_in,  output reg [15:0]op1\_out, output reg [15:0]op2\_out, output reg [15:0]result\_out  );    wire [15:0] op\_o;  wire [1:0] btn\_op1;  wire [1:0] btn\_op2;  reg [15:0] op\_in;    regfile rf(wclk, rclk, address, op\_in, op\_o);  always @\* begin  if(switch[1:0] == 2'b01) begin  op1\_out = op\_o;  op\_in = op1\_in;  end  else if(switch[1:0] == 2'b10) begin  op2\_out = op\_o;  op\_in = op2\_in;  end  else if(switch[1:0] == 2'b00) begin  result\_out = op\_o;  op\_in = result\_in;  end  end  endmodule |

1. 修改UCF文件。

|  |
| --- |
| NET "clk" LOC = "t9" ;  NET "btn\_in[0]" LOC = "m13" ;  NET "btn\_in[1]" LOC = "m14" ;  NET "btn\_in[2]" LOC = "l13" ;  NET "btn\_in[3]" LOC = "l14" ;  NET "switch[0]" LOC = "f12" ;  NET "switch[1]" LOC = "g12" ;  NET "switch[2]" LOC = "h14" ;  NET "switch[3]" LOC = "h13" ;  NET "switch[4]" LOC = "j14" ;  NET "switch[5]" LOC = "j13" ;  NET "switch[6]" LOC = "k14" ;  NET "switch[7]" LOC = "k13" ;  NET "segment[0]" LOC = "e14" ;  NET "segment[1]" LOC = "g13" ;  NET "segment[2]" LOC = "n15" ;  NET "segment[3]" LOC = "p15" ;  NET "segment[4]" LOC = "r16" ;  NET "segment[5]" LOC = "f13" ;  NET "segment[6]" LOC = "n16" ;  NET "segment[7]" LOC = "p16" ;  NET "anode[0]" LOC = "D14" ;  NET "anode[1]" LOC = "G14" ;  NET "anode[2]" LOC = "F14" ;  NET "anode[3]" LOC = "E13" ; |

1. 下载到板子上验证。

# 五、实验结果与分析

* 寄存器工作正常

# 六、讨论、心得

这次逻辑实验课的最后一次实验，也是难度最大的一次实验。这次实验中需要编写好几个代码模块，然后再综合起来。老师提供了部分模块的代码，我们在此基础上进行修改完善即可。但是要弄清楚各模块之间的变量调用关系，正确设置变量类型。这样经过一番努力后，还算顺利完成了实验任务。