**实验报告**

2019 年 5 月 16 日 成绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 王坤 | 学号 | 17051225 | 班级 | 17052312 |
| 专业 | 计算机科学与技术 | | 课程名称 | 计算机组成原理 | |
| 任课老师 | 章复嘉 | 指导老师 | 章复嘉 | 机位号 |  |
| 实验序号 |  | 实验名称 | 取指令与指令译码 | | |
| 实验时间 |  | 实验地点 |  | 实验设备号 |  |
| **一、实验目的与要求** | | | | | |
| 1. 实验目的：    1. 学习指令存储器的设计；    2. 掌握CPU取指令操作与指令译码的方法和过程； 2. 实验要求： 3. 在ISE中使用Memory IP核生成一个Inst\_ROM，当做指令存储器，并关联一个实验六所生成的\*.coe文件。 4. 编程实现取指令模块，调用Inst\_ROM指令存储器模块 | | | | | |
| **二、实验设计与程序代码** | | | | | |
| 1. 模块设计说明   本程序使用一个顶层模块和两个子模块(IP核模块和数码管模块)  本程序使用32位led灯输出指令,数码管输出PC值   1. 实验程序源代码及注释等   //顶层模块  `timescale 1ns / 1ps  //取指令模块  module Get\_Inst(clk,clr,cls,Output\_Data,shine,which,led);  input cls;//时钟  input clr;//清零  input clk;//    output [32:1] Output\_Data;//取出的指令  output [2:0] which; //八段位置  output [7:0] led; //八段led  output reg shine = 1;    reg [31:0] PC;  //wire [31:0]PC\_new;  wire [31:0]Inst\_code;    //assign PC\_new = PC+4;    always @(negedge cls)  begin  if(clr) begin PC<=32'h00000000; end  else begin PC <= PC + 4; end  end    //实例化模块  Inst\_Rom your\_instance\_name (  .clka(cls), // input clka  .addra(PC[7:2]), // input [5 : 0] addra  .douta(Output\_Data) // output [31 : 0] douta  );  LedTube instance\_name (  .data(PC),  .clk(clk),  .which(which),  .led(led)  );  // assign Output\_Data = Inst\_code;  Endmodule  //数码管模块  `timescale 1ns / 1ps  //////////////////////////////////////////////////////////////////////////////////  // Company:  module LedTube(data,clk,which,led  );  input[32:1]data;  input clk;  output reg [2:0]which=0;  output reg [7:0]led;    reg [3:0]digit;  reg [19:0] times=0;    always @ (posedge clk)  begin  if(times == 15'b111111) begin times <= 15'b0; end  else begin times <= times + 15'b1; end  end    always @ (posedge clk)  begin  if(which > 3'b111) begin which<=0; end  else if(times == 15'b111111) begin which<=which+1; end  end    always @(\*)  begin  case(which)  3'h0: digit <= data[4:1];  3'h1: digit <= data[8:5];  3'h2: digit <= data[12:9];  3'h3: digit <= data[16:13];  3'h4: digit <= data[20:17];  3'h5: digit <= data[24:21];  3'h6: digit <= data[28:25];  3'h7: digit <= data[32:29];  endcase  end  always @(\*)  begin  case(digit)  4'h0: led <= 8'b0000\_0011;  4'h1: led <= 8'b1001\_1111;  4'h2: led <= 8'b0010\_0101;  4'h3: led <= 8'b0000\_1101;  4'h4: led <= 8'b1001\_1001;  4'h5: led <= 8'b0100\_1001;  4'h6: led <= 8'b0100\_0001;  4'h7: led <= 8'b0001\_1111;  4'h8: led <= 8'b0000\_0001;  4'h9: led <= 8'b0000\_1001;  4'hA: led <= 8'b0001\_0001;  4'hB: led <= 8'b1100\_0001;  4'hC: led <= 8'b0110\_0011;  4'hD: led <= 8'b1000\_0101;  4'hE: led <= 8'b0110\_0001;  4'hF: led <= 8'b0111\_0001;  endcase  end  endmodule | | | | | |
| **三、实验仿真** | | | | | |
| 1. 仿真代码   `timescale 1ns / 1ps  ////////////////////////////////////////////////////////////////////////////////  // Company:  // Engineer:  //  // Create Date: 14:52:46 05/09/2019  // Design Name: Get\_Inst  // Module Name: E:/ISE/Term\_PC/Get\_Inst/demo.v  // Project Name: Get\_Inst  // Target Device:  // Tool versions:  // Description:  //  // Verilog Test Fixture created by ISE for module: Get\_Inst  //  // Dependencies:  //  // Revision:  // Revision 0.01 - File Created  // Additional Comments:  //  ////////////////////////////////////////////////////////////////////////////////  module demo;  // Inputs  reg clk;  reg clr;  reg cls;  // Outputs  wire [32:1] Output\_Data;  wire shine;  wire [2:0] which;  wire [7:0] led;  // Instantiate the Unit Under Test (UUT)  Get\_Inst uut (  .clk(clk),  .clr(clr),  .cls(cls),  .Output\_Data(Output\_Data),  .shine(shine),  .which(which),  .led(led)  );  initial begin  // Initialize Inputs  clk = 0;  clr = 0;  cls = 0;    // Wait 100 ns for global reset to finish  #100;  cls= 1;#10;  clr = 1;#10;  cls = 0;#10;  clr = 0;#10;  cls =1;#10; cls = 0;#10;  cls =1;#10; cls = 0;#10;  cls =1;#10; cls = 0;#10;  cls =1;#10; cls = 0;#10;cls =1;#10; cls = 0;#10;  cls =1;#10; cls = 0;#10;  cls =1;#10; cls = 0;#10;  cls =1;#10; cls = 0;#10;  // Add stimulus here    end    endmodule   1. 仿真波形      1. 仿真结果分析 | | | | | |
| **四、电路图** | | | | | |
|  | | | | | |
| **五、引脚配置（约束文件）** | | | | | |
| NET "Output\_Data[31]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[30]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[29]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[28]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[27]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[26]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[25]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[24]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[23]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[22]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[21]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[20]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[19]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[18]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[17]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[16]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[15]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[14]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[13]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[12]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[11]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[10]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[9]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[8]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[7]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[6]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[5]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[4]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[3]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[2]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[1]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[32]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[32]" LOC = R1;  NET "Output\_Data[31]" LOC = P2;  NET "Output\_Data[30]" LOC = P1;  NET "Output\_Data[29]" LOC = N2;  NET "Output\_Data[28]" LOC = M1;  NET "Output\_Data[27]" LOC = M2;  NET "Output\_Data[26]" LOC = L1;  NET "Output\_Data[25]" LOC = J2;  NET "Output\_Data[24]" LOC = G1;  NET "Output\_Data[23]" LOC = E1;  NET "Output\_Data[22]" LOC = D2;  NET "Output\_Data[21]" LOC = A1;  NET "Output\_Data[20]" LOC = L3;  NET "Output\_Data[19]" LOC = G3;  NET "Output\_Data[18]" LOC = K4;  NET "Output\_Data[17]" LOC = G4;  NET "Output\_Data[16]" LOC = K1;  NET "Output\_Data[15]" LOC = J1;  NET "Output\_Data[14]" LOC = H2;  NET "Output\_Data[13]" LOC = G2;  NET "Output\_Data[12]" LOC = F1;  NET "Output\_Data[11]" LOC = E2;  NET "Output\_Data[10]" LOC = D1;  NET "Output\_Data[9]" LOC = B1;  NET "Output\_Data[8]" LOC = B2;  NET "Output\_Data[7]" LOC = N3;  NET "Output\_Data[6]" LOC = M3;  NET "Output\_Data[5]" LOC = K3;  NET "Output\_Data[4]" LOC = H3;  NET "Output\_Data[3]" LOC = N4;  NET "Output\_Data[2]" LOC = L4;  NET "Output\_Data[1]" LOC = J4;  NET "led[7]" IOSTANDARD = LVCMOS18;  NET "led[6]" IOSTANDARD = LVCMOS18;  NET "led[5]" IOSTANDARD = LVCMOS18;  NET "led[4]" IOSTANDARD = LVCMOS18;  NET "led[3]" IOSTANDARD = LVCMOS18;  NET "led[2]" IOSTANDARD = LVCMOS18;  NET "led[1]" IOSTANDARD = LVCMOS18;  NET "led[0]" IOSTANDARD = LVCMOS18;  NET "led[7]" LOC = H19;  NET "led[6]" LOC = G20;  NET "led[5]" LOC = J22;  NET "led[4]" LOC = K22;  NET "led[3]" LOC = K21;  NET "led[2]" LOC = H20;  NET "led[1]" LOC = H22;  NET "led[0]" LOC = J21;  NET "which[2]" IOSTANDARD = LVCMOS18;  NET "which[1]" IOSTANDARD = LVCMOS18;  NET "which[0]" IOSTANDARD = LVCMOS18;  NET "which[2]" LOC = M22;  NET "which[1]" LOC = M21;  NET "which[0]" LOC = N22;  NET "clr" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "clr" IOSTANDARD = LVCMOS18;  NET "clr" LOC = V8;  NET "cls" LOC = AA8;  NET "cls" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "cls" IOSTANDARD = LVCMOS18;  NET "clk" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "clk" IOSTANDARD = LVCMOS18;  NET "clk" LOC = H4;  NET "shine" IOSTANDARD = LVCMOS18;  NET "shine" LOC = L21; | | | | | |
| **六、思考与探索** | | | | | |
| 1. 实验结果记录：   实验结果与预期完全一致   1. 实验结论：   实验结果与预期完全一致   1. 问题与解决方案：   问题一:PC值的选择  解决方案;开始的时候,IP核的模块的地址输入直接是PC,然后出来的结果就是跳跃式的出现,然后改成PC[5:0],结果还是一样的,仔细想想后,发现程序执行的是PC+4,而不是循序的PC+1,但是在coe文件里面每次读取的时候,地址只是自加一,所以,这个地方输入的PC值应该是PC[7:2]  问题二:rst与clk的先后与位置混乱  解决方案刚刚开始打算把rst单独放到一个always语句里面,然后发现这样做PC值就会在两个驱动里面被赋值,语法不通过,然后就把rst放到了clk下跳沿触发的一个always语句里面,然后使用一个if语句去检测clr,但是这样做就会出现一个新的问题,就是如果我提前按住了rst,但是clk’即将盗来的并不是上跳沿,这样就会存在周期浪费,目前还没有解决!   1. 思考题： 2. 实验结果与COE文件里面完全一致 3. 第一个clk到来时读出来的是0号单元 | | | | | |