计算机组成实验2

单周期CPU模块IO设计仿真

518030910283 王航宇

一、 实验目的

1、 在理解计算机5大组成部分的协调工作原理，理解存储程序自动执行的原理和掌握运算器、存储器、控制器的设计和实现原理 基础上，掌握 I/O 端口的设计方法，理解 I/O 地址空间的设计方法。

2、 通过设计 I/O 端口与外部设备进行信息交互。

3、 通过设计并实现新的自定义指令拓展CPU功能，深入理解CPU对指令的译码 、执行原理和实现方式。（选做）

二、 实验要求

1、采用Verilog硬件描述语言在Quartus II EDA 设计平台中，基于 Intel cyclone II 系列 FPGA完成具有执行20 条MIPS基本指令的单周期 CPU 的输入输出部件亦即 IO 接口扩展模块设计。在之前提供并自行补充已完成的单周期 CPU模块基础上添加对IO 的内部相应处理并添加 IO 接口模块，实现 CPU 对外设的 IO 访问。

2、 利用实验提供的标准测试程序代码， 对单周期 CPU 的 IO 模块进行功能仿真测试， 要求CPU能够采用查询方式模拟接收输入按键或开关的状态，并产生相应的输出状态驱动数码管显示结果从而验证 CPU 可正确执行 IO 读写指令。

3、 自行设计添加一条新的 CPU 指令，修改 CPU 控制部件和执行部件模块代码，支持新指令的操作，并通过 仿真验证功能的正确性。（选做）

三、 实验过程及结果

改写sc_datamem.v，增加了IO端口声明，内部添加了子模块io_input、io_output，用来对IO地址空间的译码以及构成IO和CPU内部之间的数据传输通道，另外增加了一个选择数据来源的mux模块

```
module sc_datamem (addr,datain,dataout,we,clock,mem_clk,dmem_clk,
      out_port0,out_port1,out_port2,in_port0,in_port1,mem_dataout,io_read_data
);

   input  [31:0]  addr;
   input  [31:0]  datain;
   input  [31:0]  in_port0,in_port1;
   input          we, clock,mem_clk;

   output [31:0]  dataout;
   output         dmem_clk;
   output [31:0]  out_port0,out_port1,out_port2;
   output [31:0]  mem_dataout,io_read_data;

   wire           dmem_clk;
   wire           write_enable;
   wire [31:0]    dataout;
   wire [31:0]    mem_dataout;
   wire           write_data_enable;
   wire           write_io_enable;

   assign         write_enable = we & ~clock;
   assign         dmem_clk = mem_clk & ( ~ clock) ;
   assign         write_data_enable = write_enable & (~addr[7]);
   assign         write_io_enable = write_enable & addr[7];

   lpm_ram_dq_dram  dram(addr[6:2],dmem_clk,datain,write_data_enable,mem_dataout );//从mem中取出mem_dataout

   io_output io_output_reg (addr,datain, write_io_enable, dmem_clk, out_port0,out_port1,out_port2);//把数据送到外部设备

   io_input io_input_reg(addr,dmem_clk,io_read_data,in_port0,in_port1);//从io中取出io_read_data

   mux2x32 io_data_mux(mem_dataout,io_read_data,addr[7],dataout);//选择io_read_data、mem_dataout哪个作为dataout

endmodule
```

io_output.v实现从CPU到IO的数据传输

```
module io_output (addr,datain,write_io_enable,io_clk,out_port0,out_port1,out_port2);
   input [31:0] addr,datain;
   input        write_io_enable,io_clk;
   output [31:0] out_port0,out_port1,out_port2;

   reg    [31:0] out_port0;
   reg    [31:0] out_port1;
   reg    [31:0] out_port2;

   always@(posedge io_clk)
   begin
      if(write_io_enable == 1)
         case(addr[7:2])
            6'b100000:out_port0=datain;//80h
            6'b100001:out_port1=datain;//84h
            6'b100010:out_port2=datain;//88h
         endcase
   end
endmodule
```

io_input.v实现从IO到CPU的数据传输

```
module io_input(addr,io_clk,io_read_data,in_port0,in_port1);
   input [31:0] addr;
   input io_clk;
   input [31:0] in_port0,in_port1;

   output [31:0] io_read_data;

   reg [31:0] in_reg0; // input port0
   reg [31:0] in_reg1; // input port1

   io_input_mux io_imput_mux2x32(in_reg0,in_reg1,addr[7:2],io_read_data);
   always @(posedge io_clk)
   begin
      in_reg0 <= in_port0; // 输入端口在 io_clk 上升沿时进行数据锁存
      in_reg1 <= in_port1; // 输入端口在 io_clk 上升沿时进行数据锁存
// more ports 可根据需要设计更多的输入端口
   end
endmodule
```

io_input_mux.v选择数据来源

```
module io_input_mux(a0,a1,sel_addr,y);
    input [31:0] a0,a1;
    input [5:0] sel_addr;

    output [31:0]  y;

    reg [31:0] y;

    always @ *
        case (sel_addr)
            6'b100000: y = a1;
            6'b100001: y = a0;
// more ports 可根据需要设计更多的端口
            default: y = 32'h0;
        endcase
endmodule
```

sc_computer_main.v在CPU中完成计算，并把结果输出到IO端口

```
module sc_computer_main (resetn,clock,mem_clk,pc,inst,aluout,memout,imem_clk,dmem_clk,mem_dataout,io_read_data,
                         out_port0,out_port1,out_port2,in_port0,in_port1);
   input resetn,clock,mem_clk;
   input  [31:0]  in_port0,in_port1;

   output [31:0]  out_port0,out_port1,out_port2;
   output [31:0]  pc,inst,aluout,memout;
   output         imem_clk,dmem_clk;
   output [31:0]  mem_dataout,io_read_data;


   wire   [31:0] data;
   wire          wmem; // all these "wire"s are used to connect or interface the cpu,dmem,imem and so on.

   sc_cpu cpu (clock,resetn,inst,memout,pc,wmem,aluout,data);          // CPU module.
   sc_instmem  imem (pc,inst,clock,mem_clk,imem_clk);                  // instruction memory.
   sc_datamem  dmem (aluout,data,memout,wmem,clock,mem_clk,dmem_clk,
                     out_port0,out_port1,out_port2,in_port0,in_port1,mem_dataout,io_read_data );   // data memory.

endmodule
```

顶层文件sc_computer.v，增加对新加的IO的端口声明，并在例化模块时对新加的IO参数添加与之连接的信号

```
module sc_computer (reset,clk,
    SW0,SW1,SW2,SW3,SW4,SW5,SW6,SW7,SW8,SW9,
    HEX0,HEX1,HEX2,HEX3,HEX4,HEX5);

    input reset,clk;
    input SW0,SW1,SW2,SW3,SW4,SW5,SW6,SW7,SW8,SW9;

    output[6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5;

    wire [31:0] pc,inst,aluout,memout;
    wire        imem_clk,dmem_clk;

    wire        clock_out;
    wire[31:0]  mem_dataout,io_read_data;
    wire[31:0]  in_port0,in_port1;
    wire[31:0]  out_port0,out_port1,out_port2;

    clock_and_mem_clock inst3(clk,reset,clock_out);//用clk产生clock_out

    in_port inst1(SW4,SW3,SW2,SW1,SW0,in_port0);//把五位2进制数存入io
    in_port inst2(SW9,SW8,SW7,SW6,SW5,in_port1);

    sc_computer_main inst4(reset,clock_out,clk,pc,inst,aluout,memout,imem_clk,dmem_clk,mem_dataout,io_read_data,
                    out_port0,out_port1,out_port2,in_port0,in_port1);//

    out_port_seg inst5(out_port0,HEX1,HEX0);//将输出数据转换到十进制然后七段译码器
    out_port_seg inst6(out_port1,HEX3,HEX2);
    out_port_seg inst7(out_port2,HEX5,HEX4);
endmodule
```

新增clock_and_mem_clock模块，用clk生成二分频clock_out

```
module clock_and_mem_clock (clk,reset,clock_out);
    input clk,reset;
    output clock_out;

    reg clock_out;

    initial
        clock_out <= 0;//初始为零

    always @ (posedge clk)
        begin
            if (~reset)
                clock_out <= 0; // 复位置零
            else
                clock_out <= ~clock_out; // 否则clock_out信号翻转
        end
endmodule
```

新增in_port模块，存储输入的五位二进制数

```
module in_port(SW4,SW3,SW2,SW1,SW0,in_port_num);//把五位二进制数加入到inport中
    input SW4,SW3,SW2,SW1,SW0;
    output[31:0] in_port_num;
    assign in_port_num={27'b0,SW4,SW3,SW2,SW1,SW0};


endmodule
```

新增out_port模块，将输出转换为十进制并用七段译码器显示在LED上

```
module out_port_seg(out_port_num,HEX1,HEX0);
    input[31:0]    out_port_num;
    output[6:0]    HEX1,HEX0;


    wire[3:0]      num1,num0;

    assign num1 = out_port_num/10;//获取十位数
    assign num0 = out_port_num%10;//获取个位数

    seven_segment_decoder high_digit(num1,HEX1);//十位数保存在HEX1
    seven_segment_decoder low_digit(num0,HEX0);//个位数保存在HEX0


endmodule
```

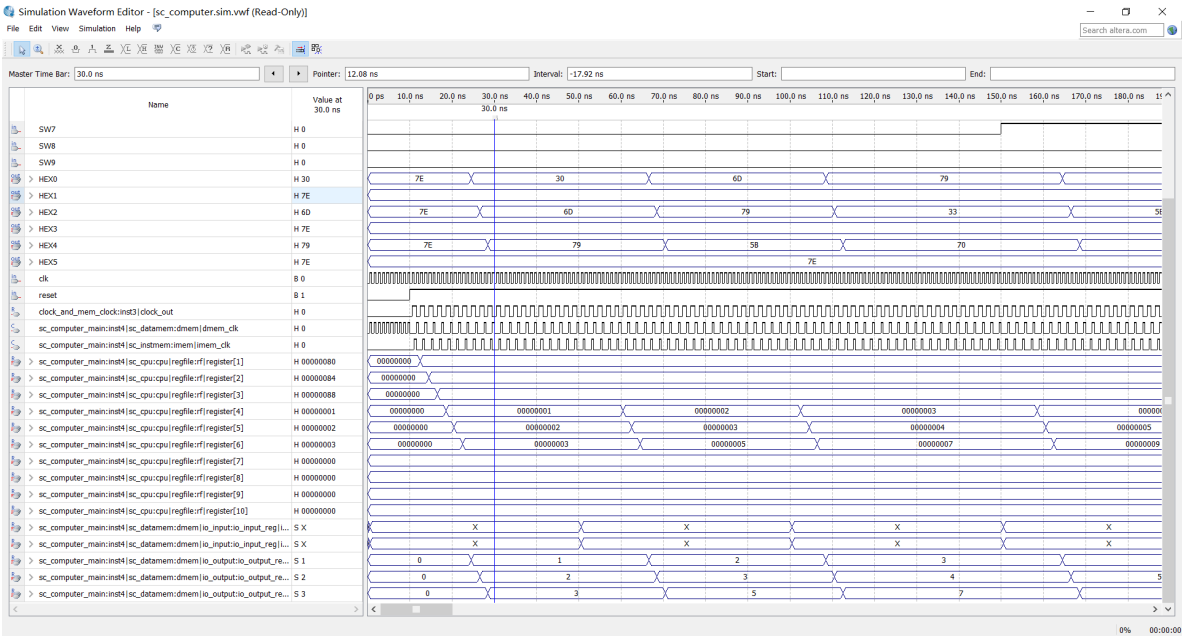
seven_segment_decoder.v是七段译码器的具体实现

```
module seven_segment_decoder (num,HEX);
    input[3:0] num;
    output[6:0] HEX;

    reg [6:0] HEX;

    always@(*)
        begin
            case(num)
                4'b0000: HEX = 7'b1111110; // 0
                4'b0001: HEX = 7'b0110000; // 1
                4'b0010: HEX = 7'b1101101; // 2
                4'b0011: HEX = 7'b1111001; // 3
                4'b0100: HEX = 7'b0110011; // 4
                4'b0101: HEX = 7'b1011011; // 5
                4'b0110: HEX = 7'b1011111; // 6
                4'b0111: HEX = 7'b1110000; // 7
                4'b1000: HEX = 7'b1111111; // 8
                4'b1001: HEX = 7'b1111011; // 9
                default: HEX = 7'b0000000;
            endcase
        end
endmodule
```

最终的波形图如下



从图中可知，每次输入的两组数SW4~0，SW9~5被CPU读到第4、5号寄存器后进行加运算，结果保存在第6号寄存器中，之后输出到out_port2，并经过七段译码后从HEX0，HEX1信号输出。

四、 选做实验

新增加一条hads指令，该指令用于计算两个数之间的汉明距离

在ALU中添加hads的功能

```
   reg[31:0]    res;
   always @ (a or b or aluc)
      begin                                  // event
         casex (aluc)
             4'bx000: s = a + b;             //x000 ADD
             4'bx100: s = a - b;             //x100 SUB
             4'bx001: s = a & b;             //x001 AND
             4'bx101: s = a | b;             //x101 OR
             4'bx010: s = a ^ b;             //x010 XOR
             4'bx110: s = {b[15:0],16'b0};   //x110 LUI: imm << 16bit
             4'b0011: s = b << a;            //0011 SLL: rd <- (rt << sa)
             4'b0111: s = b >> a;            //0111 SRL: rd <- (rt >> sa) (logical)
             4'b1111: s = $signed(b) >>> a;  //1111 SRA: rd <- (rt >> sa) (arithmetic)
             4'b1011:                        //1011 hads
               begin
                  res = a ^ b;
                  s = res[0] + res[1] + res[2] + res[3] + res[4] + res[5] + res[6] + res[7] +
                      res[8] + res[9] + res[10] + res[11] + res[12] + res[13] + res[14] + res[15] +
                      res[16] + res[17] + res[18] + res[19] + res[20] + res[21] + res[22] + res[23] +
                      res[24] + res[25] + res[26] + res[27] + res[28] + res[29] + res[30] + res[31] ;
               end
            default: s = 0;
         endcase
         if (s == 0 )  z = 1;
            else z = 0;
      end
endmodule
```

设置hads的func是111111

```
wire i_hads = r_type & func[5] & func[4] & func[3] &
              func[2] &  func[1] & func[0];          //111111
```

设置hads所需要的控制信号，hads的aluc是1011，wreg是1

```
assign wreg = i_add | i_sub | i_and | i_or   | i_xor  |
              i_sll | i_srl | i_sra | i_addi | i_andi |
              i_ori | i_xori | i_lw | i_lui  | i_jal | i_hads;

assign aluc[3] = i_sra | i_hads;
assign aluc[2] = i_sub | i_beq | i_bne | i_or | i_ori | i_lui | i_srl | i_sra;
assign aluc[1] = i_xor | i_sll | i_srl | i_sra | i_xori | i_lui | i_hads;
assign aluc[0] = i_and | i_andi | i_or | i_ori | i_sll | i_srl | i_sra |i_hads;
assign shift   = i_sll | i_srl | i_sra ;
```

修改instmem.mif。模拟的操作是把128和132分别放入1号和2号寄存器，计算他们之间的汉明距离，并把结果放到三号寄存器

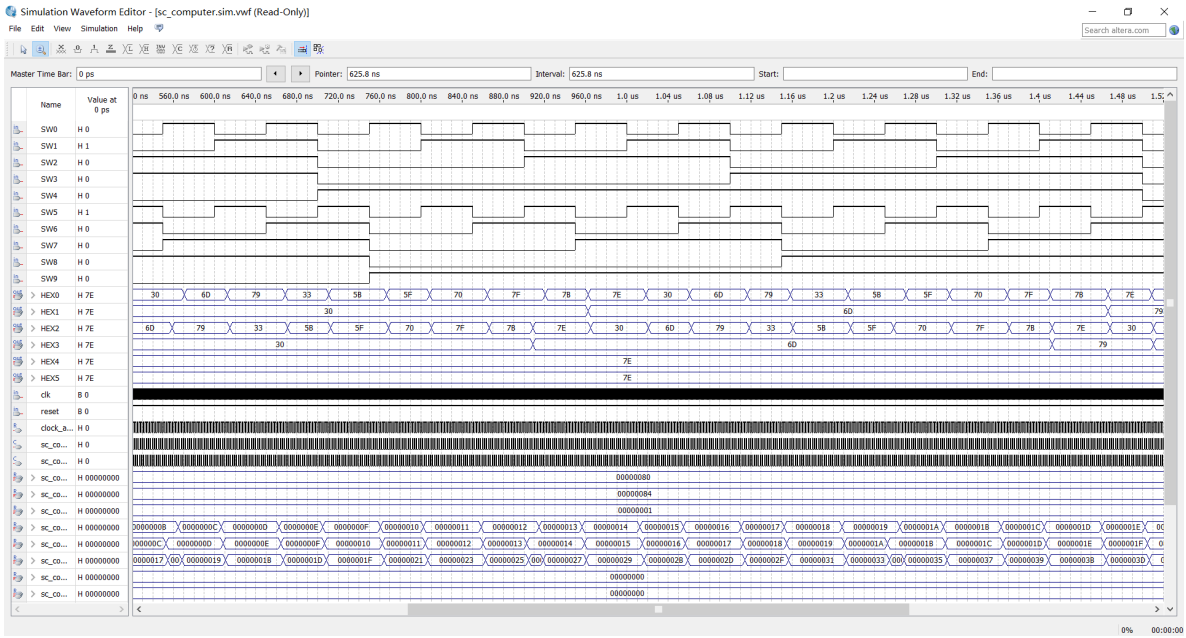
sc_instmem.mif - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
WIDTH = 32;           % Enter a decimal number %
ADDRESS_RADIX = HEX;  % Address and value radixes are optional %
DATA_RADIX = HEX;     % Enter BIN, DEC, HEX, or OCT; unless %
                      % otherwise specified, radixes = HEX %
CONTENT
BEGIN

0 : 20010080;       % (00) main: addi $1, $0, 128 # outport0, inport0
 %
1 : 20020084;       % (04)      addi $2, $0, 132 # outport1, inport1
%
2 : 0022183f;       % (08)      hads $3, $1, $2 # outport2                  %
3 : 8c240000;       % (0c) loop: lw   $4, 0($1)   # input inport0 to $4
%
4 : 8c450000;       % (10)      lw   $5, 0($2)   # input inport1 to $5        %
5 : 00853020;       % (14)      add  $6, $4, $5  #                          %
6 : ac240000;       % (18)      sw   $4, 0($1)   # output inport0 to outport0
 %
7 : ac450000;       % (1c)      sw   $5, 0($2)   # output inport1 to outport1
%
8 : ac660000;       % (20)      sw   $6, 0($3)   # output result to outport2
%
9 : 08000003;       % (24)      j loop          #                          %
END ;
```

最终结果图



1号寄存器里存储00000080，2号寄存器存储00000084，3号寄存器存储他们之间的汉明距离00000001

五、实验收获感想

通过这次实验，我简单的了解了 I/O 端口和 I/O 地址空间的设计方法，了解了 I/O 端口与外部设备进行信息交互的方式；并且通过设计并实现新的自定义指令拓展CPU功能，了解了CPU对指令的译码 、执行原理和实现方式