《计算机组成》课程实验之单周期 CPU 的 IO 接口模块设计仿真

要求说明

202004

本课程实验教学目的，是要求同学们利用 Verilog 硬件描述语言、基于大规模可编程逻辑阵列器件 FPGA，进行计算机 CPU 的逻辑功能设计，包括 RISC 风格的单周期 CPU 设计、5 级流水 CPU 设计、以及在其上的输入输出部件扩展和功能扩展。

以下为第二部分单周期 CPU 的 IO 接口模块设计的实验要求、实验步骤、设计指导、仿真要求。

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、实验步骤

1、采用 I/O 统一编址方式，即将输入输出的 I/O 地址空间，作为数据存取空间的一部分，实现 CPU 与外部设备的输入输出端口设计。实验中可采用高端地址。

2、编写 CPU 对 IO 地址空间的访问测试代码，利用设计的 I/O 端口，通过 lw 指令，输入模拟的按键等输入设备信息。即将外部设备状态，读到 CPU 内部寄存器。

3、利用设计的 I/O 端口，通过 sw 指令，输出对模拟的 LED 灯等输出设备的控制信号（或数据信息）。即将对外部设备的控制数据，从 CPU 内部的寄存器，写入到外部设备的相应控制寄存器（或可直接连接至外部设备的控制输入信号）。

4、利用自己编写的程序代码，在自己设计改写的 CPU 上，实现对模拟的板载输入开关或按键的状态输入，并将判别或处理结果，利用模拟的 7 段 LED 数码管显示出来。

5、例如，将一路 5bit 二进制输入与另一路 5bit 二进制输入相加，利用两组分别 2 个 LED 数码管以 10 进制形式显示“被加数”和“加数”，另外一组 LED 数码管以 10 进制形式显示“和”等。（具体任务形式不做严格规定，同学可自由创意，本文档后续的设计指导给出一个这样的设计仿真结果例子供参考）。

6、设计一条指令，自己定义该指令要执行的操作，例如可以是求两个数的汉明距离，改写

CPU 代码完成对指令的支持。改写仿真测试程序，通过对添加的新指令进行仿真，确认设计的正确性。（选做）

四、设计指导

本部分实验不再给完整详细的示例程序，仅给出部分程序和设计思路指导。

之前提供的单周期 CPU 模块示例程序源代码顶层文件 sc_computer.v 包含以下三个主要模块，CPU 模块 sc_cpu.v、指令存储器 sc_instmem.v 以及数据存储器 sc_datamem.v。

可在 sc_datamem.v 模块文件中，通过对输入的 lw 或 sw 指令的地址的判断，实现对数据 RAM 和 IO 控制寄存器组的区分、和分别控制。

增加对 IO 的扩展，一是要增加 IO 端口模块，二是要改写数据存储器模块的内部控制逻辑，增加对内存映射的 IO 地址段的访问功能。

sc_datamem.v 的参考代码、及其实现原理图如下图 1 及随后的源码所示。

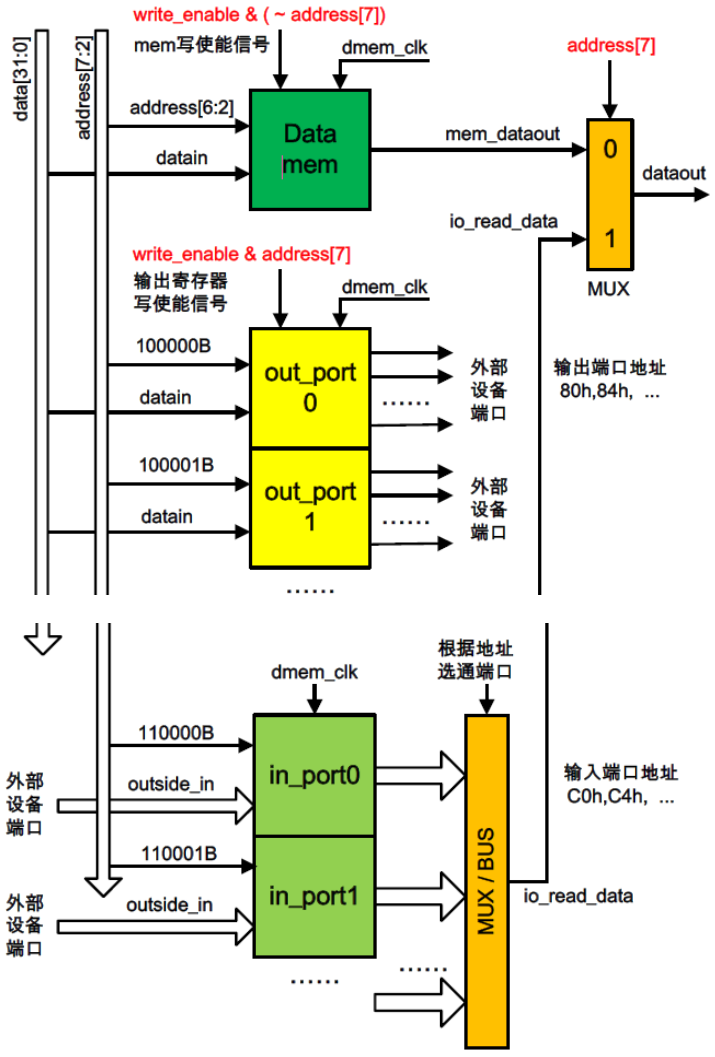

图 1　统一编址方式 I/O 端口扩展原理图解

图 2 给出了一种实现方式的 Quartus II 工程截图。其中，增加的 IO 端口模块为 in_port 和 out_port_seg。in_port 接受按键的输入，将小于 32bit 输入端口转换为 32 位。out_port_seg 将输出数据转成十进制并经过七段译码器输出到 LED。

而对 sc_datamem.v 的改写，一是增加了 IO 端口声明，二是内部添加了子模块 io_input，io_output，用于完成对 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。另外还增加了一个选择数据来源的 mux 模块。

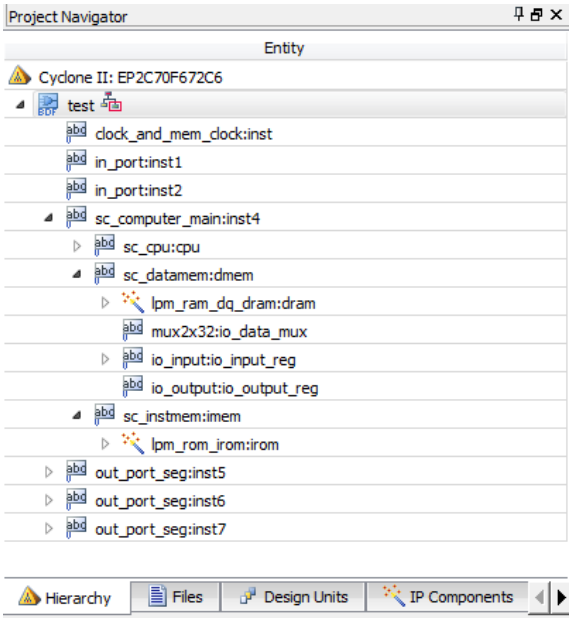

图 2 一种实现方式的 Quartus II 工程截图

改写后的 **sc_datamem.v 代码示例**如下，供参考。

```
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
); //添加了 2 个 32 位输入端口，3 个 32 位输出端口

    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;
    output [31:0] 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; //注意
```

```
        assign              dmem_clk = mem_clk & ( ~ clock) ; //注意
        //100000-111111:I/O ; 000000-011111:data
        assign write_data_enable = write_enable & (~addr[7]); //注意
        assign write_io_enable = write_enable & addr[7]; //注意
      mux2x32 io_data_mux(mem_dataout,io_read_data,addr[7],dataout); //添加子模块，用于选择输出数据来源于内部数据存储器还是 IO，
    //module mux2x32 (a0,a1,s,y);
    // when address[7]=0, means the access is to the datamem.
    // that is, the address space of datamem is from 000000 to 011111 word(4 bytes)
    // when address[7]=1, means the access is to the I/O space.
    // that is, the address space of I/O is from 100000 to 111111 word(4 bytes)
      lpm_ram_dq_dram      dram (addr[6:2], dmem_clk, datain, write_data_enable, mem_dataout );
      io_output io_output_reg (addr, datain, write_io_enable, dmem_clk, out_port0, out_port1,out_port2);
    //添加子模块，用于完成对 IO 地址空间的译码，以及构成 IO 和 CPU 内部之间的数据输出通道
    //module io_output(addr,datain,write_io_enable,io_clk,out_port0,out_port1,out_port2 );
      io_input io_input_reg(addr, dmem_clk, io_read_data, in_port0, in_port1);
    //添加子模块，用于完成对 IO 地址空间的译码，以及构成 IO 和 CPU 内部之间的数据输入通道
    //module io_input( addr,io_clk,io_read_data,in_port0,in_port1);
    Endmodule
```

io_output.v 模块参考代码：

```
    module io_output(
        addr,datain,write_io_enable,io_clk,out_port0,out_port1,out_port2
); //本例中没有添加reset信号，可以自行添加. if necessary,can use reset signal to reset the output to 0.

        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
            // more ports，可根据需要设计更多的输出端口
                endcase
    end
endmodule
```

io_input.v 模块参考代码：

```
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

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 = a0;
            6'b100001: y = a1;
          // more ports，可根据需要设计更多的端口
            default: y = 32'h0;
        endcase
endmodule
```

相应的，顶层模块 sc_computer 也要做对应的修改，增加对新加的 IO 的端口声明，并在例化 sc_datamem 模块的时候对新加的 IO 参数要添加与之连接的信号。

同时可见图一中还增加了一个 clk_and_mem_clk，其功能是采用一个 clk 作为主时钟输

入，在内部给出二分频后作为 CPU 模块的工作时钟，并产生 mem_clk，用于对 CPU 模块内含的存储器读写控制。

五、仿真验证

提供了一个用于功能仿真的激励信号波形文件 :sc_computer_test_wave_02.vwf。配合同时提供的测试 IO 功能的 CPU 程序 RAM 和数据 RAM 初始化文件，可用于仿真验证。仿真调试过程中可根据调试需要基于该文件进行修改。另外注意，如果调试代码超过原来设置的 ROM 空间大小，需要更改.mif 文件里 DEPTH 参数以适应容量改变。也可以自行根据实验一里的说明，定义一个新的更大的 ROM。

sc_code_iotest.txt 文件列出了用于测试 CPU 读写 IO 的汇编程序代码，该程序对应的 sc_instmem.mif 文件为指令存储器初始值文件，该段示例的 CPU 程序代码，完成从 IO 口读取两个数据并将求和结果输出显示的功能。

文件里每行对应了存储单元字顺序号、机器指令、%（PC 值）、汇编指令、#语句注释%。具体内容如下：

```
DEPTH = 16;                 % Memory depth and width are required %
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 : 20030088;           % (08)           addi $3, $0, 136 # 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    # add inport0 with inport1 to $6 %
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          #
```

4、sc_datamem.mif 文件内包含了数据存储器初始值，这个例子里，初值为全 0：

```
DEPTH = 1;                  % Memory depth and width are required %
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 %
CONTENT                       % otherwise specified, radixes = HEX %

BEGIN

End ;
```

IO 模块设计补充完整后用提供的 sc_computer_test_wave_02.vwf 对改写后的 CPU 模块执行数据求和示例程序的过程进行仿真，能够得到如下仿真结果。

图 3 中通过 SW4~0 和 SW9~5 设置两组输入数据，代表两组五位二进制数。通过图 4，可查看到运行示例程序后的执行过程。Reset 信号上升沿后 CPU 执行第一条指令，R1 寄存器被写入输入端口地址。随后每个时钟执行一条指令，两组数被 CPU 读到第 4、5 号寄存器后进行加运算，结果存在寄存器 6 里，之后输出到 out_port2，并经过七段译码后从 HEX0、HEX1 信号输出。第 10 个 clock 时，求和结果被送到 outport2，程序接着循环执行 7 条指令的 loop 代码段。注意关注 CPU 主时钟 clk_out 和指令存储器读时钟 imem_clk、数据存储器读写时钟 dmem_clk 的关系参见图 4。

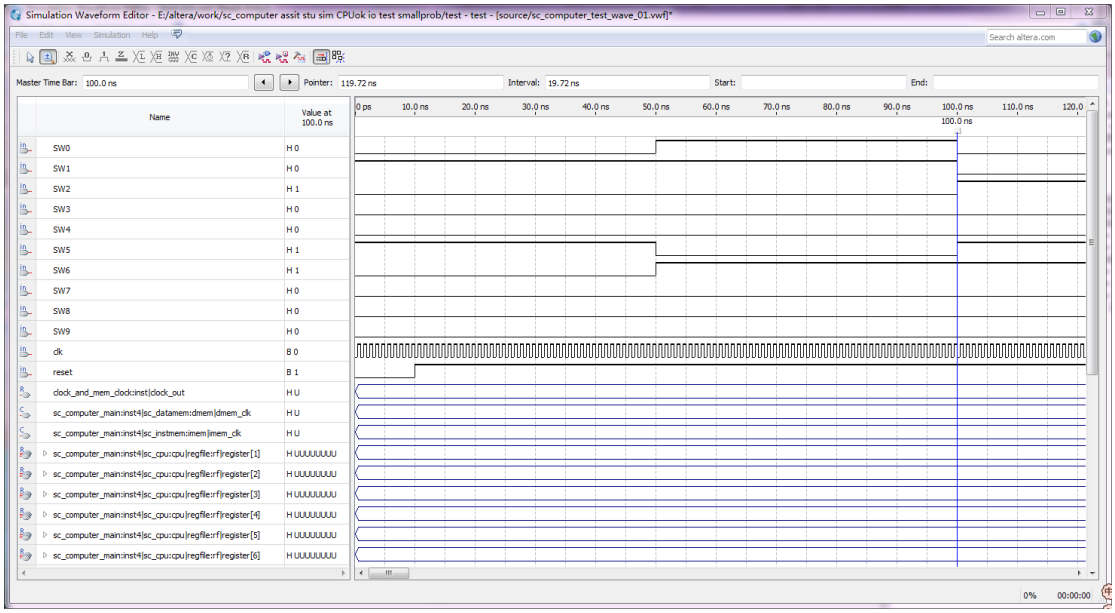

图 3 设置激励信号 SW4-0 和 SW9-5 模拟两个五位的输入数据

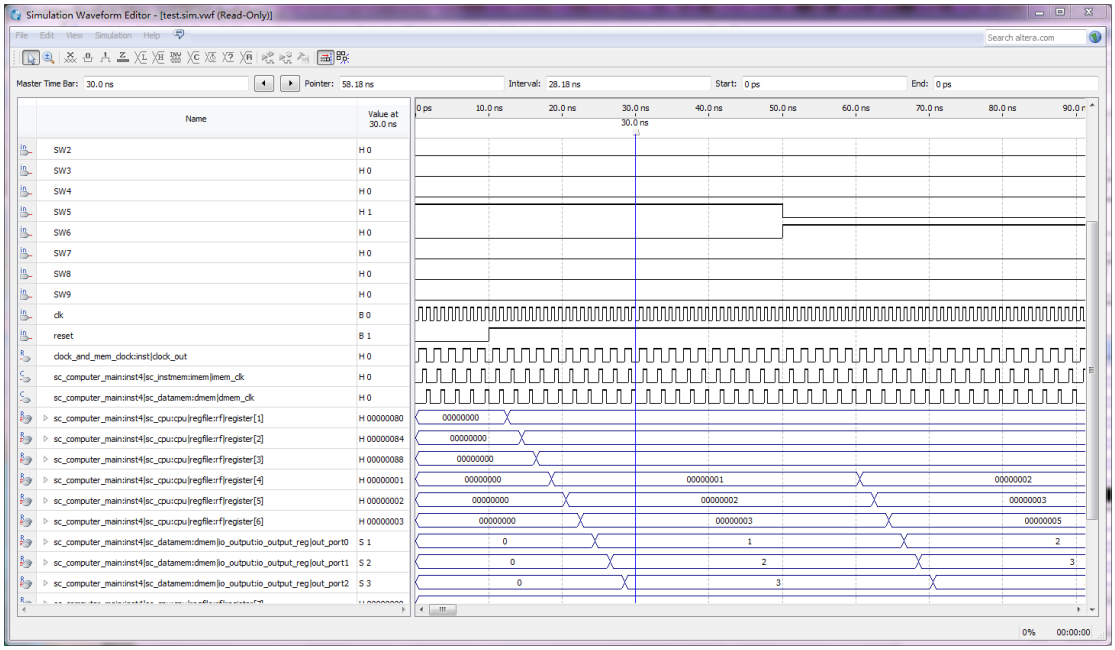

图 4 对应的仿真输出结果

附 IO 接口控制器原理简介

外部设备，如显示器、打印机、键盘、鼠标、以太网等，都是通过相应的各种“接口”和计算机进行连接的，如 VGA 接口、USB 接口、RS232 串行接口、以太网接口、modem 调制解调器等。“接口”是设备和计算机 I/O 总线之间的连接和控制部件，本质上是挂在 I/O 总线上的一种通信控制器。

接口的工作原理示意图如图 5 所示。CPU 通过 I/O 总线和接口控制器内的若干寄存器进行数据交换，这些寄存器有两类，相对于 CPU，一种是作为 CPU 的输入端口，一种是作为 CPU 的输出端口。这些寄存器按照其功能，可以分为数据寄存器和控制寄存器，数据寄存器里缓冲的是接口通讯线路上真正传输的数据，控制寄存器中放置的是对“通讯电路控制逻辑”的控制数据（或称控制信号），起配置通讯线路数据传输协议（或电信号格式）的作用。

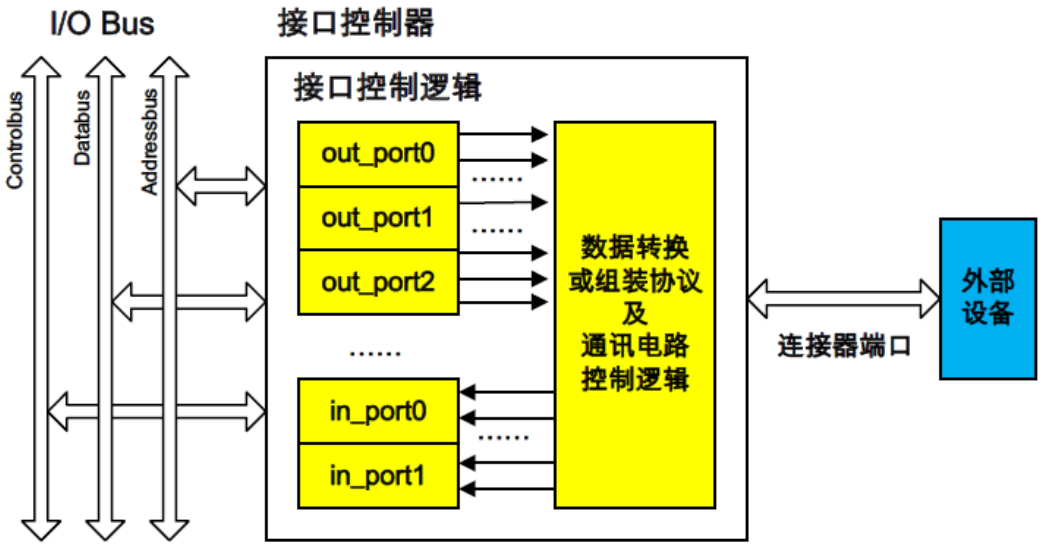

图 5 接口控制器原理

接口对于 CPU 可见的是一组输入输出端口寄存器，CPU 也是通过对这一组端口寄存器的“读”和“写”，达到控制“接口”、以及和“接口”交互数据的功能，亦即达到和外设进行交互的功能。

实验中，可通过设计相应的 I/O 端口和相应的接口控制逻辑，达到和多种外设利用其“接口”进行交互的目的。