上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

实验报告

EXPERIMENT REPORT

基本单周期 CPU 设计与外部 I/O 及接口扩展

学生姓名：XXX

学生学号：XXX

班　　级：XXX

目录

实验目的

1. 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。
2. 掌握运算器、存储器、控制器的设计和实现原理。重点掌握控制器设计原理和实现方法。
3. 掌握 I/O 端口的设计方法，理解 I/O 地址空间的设计方法。
4. 会通过设计 I/O 端口与外部设备进行信息交互。

实验仪器

1. DE1-SOC Cyclone V 实验板 1 套
2. Altera Quartus II 13.1 软件
3. ModelSim-Altera 10.1d 软件

实验任务

1. 采用 Verilog HDL 在 Quartus II 中实现基本的具有 20 条 MIPS 指令的单周期 CPU 设计。
2. 利用实验提供的标准测试程序代码，完成仿真测试。
3. 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。
4. 利用设计的 I/O 端口，通过 lw 指令，输入 DE2 实验板上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。
5. 利用设计的 I/O 端口，通过 sw 指令，输出对 DE2 实验板上的 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）。
6. 利用自己编写的程序代码，在自己设计的 CPU 上，实现对板载输入开关或按键的状态输入，并将判别或处理结果，利用板载 LED 灯或 7 段 LED 数码管显示出来。
7. 例如，将一路 4bit 二进制输入与另一路 4bit 二进制输入相加，利用两组分别 2 个 LED 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）。
8. 在实现 MIPS 基本 20 条指令的基础上，掌握新指令的扩展方法。
9. 在实验报告中，汇报自己的设计思想和方法；并以汇编语言的形式，提供以上指令集全覆盖的测试应用功能的程序设计代码，并提供程序主要流程图。

实验过程

总体思路

本次实验的设计主要由两部分组成：单周期 CPU 设计与外部 IO 拓展。第一部分，单周期 CPU 设计，由于老师已经给出了 CPU 的框架以及大部分的 Verilog 代码，只需要自行设计缺失的一部分指令并实现 CU 和 ALU 的功能；第二部分，外部 IO 拓展，本次实验采用 IO 地址和主存同一编址的方式实现，因而需要对原有的数据主存进行修改，加入对外部 IO 的支持。

本次实验对应的 Quartus 项目的 Top-Level Entity 的最终结构如图 1 所示：

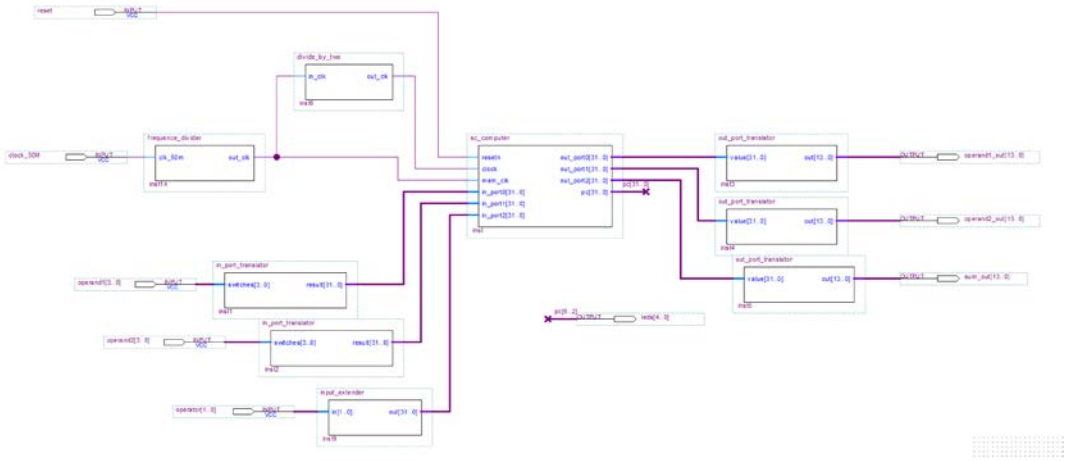

图 1：Top-Level Entity

单周期 CPU 设计

实验前，已有部分指令与 CPU 内部控制信号的对应表格，那么，我需要自行设计剩余指令与控制信号的对应关系。只要不与已有的指令发生冲突，这一部分的内容具有很大的自由度。

以 beq 指令为例，其格式为{000100，rs(5)，rt(5)，imm(16)}，那么对于 CPU 来说，当读取到 op = 000100 时，便可知这是一条 beq 指令，据此可以发出控制信号：ALU 需要执行减法操作因而 aluc = x100，无需移位因而 shift = 0，ALU 的操作数不是立即数因而 aluimm = 0，imm 需要做符号拓展因而 sext = 1，无需写主存或者寄存器堆因而 wmem = wreg = m2reg = 0，regrt = x，不为 jal 指令因而 jal = 0 等等。

指令	指令格式		op	rs	rt	rd	sa	func		pcsource [1..0]	aluc [3..0]	shift	aluimm	sext	wmem	wreg	m2reg	regrt	call jal
addi	addi	rt, rs, imm	001000	rs	rt		imm		x	0 0	0 0 0 0	0	1	1	0	1	0	1	0
andi	andi	rt, rs, imm	001100	rs	rt		imm		x	0 0	x 0 0 1	0	1	0	0	1	0	1	0
ori	ori	rt, rs, imm	001101	rs	rt		imm		x	0 0	x 1 0 1	0	1	0	0	1	0	1	0
xori	xori	rt, rs, imm	001110	rs	rt		imm		x	0 0	x 0 1 0	0	1	0	0	1	0	1	0
lw	lw	rt, imm(rs)	100011	rs	rt		imm		x	0 0	0 0 0 0	0	1	1	0	1	1	1	0
sw	sw	rt, imm(rs)	101011	rs	rt		imm		x	0 0	0 0 0 0	0	1	1	1	0	x	x	0
beq	beq	rs, rt, imm	000100	rs	rt		imm		0	0 0	x 1 0 0	0	0	1	0	0	0	x	0
									1	0 1									
bne	bne	rs, rt, imm	000101	rs	rt		imm		0	0 1	x 1 0 0	0	0	1	0	0	x	x	0
									1	0 0									
lui	lui	rt, imm	001111	00000	rt		imm		x	0 0	x 1 1 0	0	1	0	0	1	0	1	0
j	j	addr	000010			addr			x	1 1	x x x x	x	x	x	0	0	x	x	x
jal	jal	addr	000011			addr			x	1 1	x x x x	x	x	x	0	1	x	x	1

图 2：真值表的部分内容

以此方式逐一完成各条指令的控制信号设计，部分结果如图 2 所示，然后便可依照此表完成 CU 和 ALU 的设计。由于完整的代码较为冗长不便贴出，在此仅以 beq 指令和 shift 控制信号为例进行说明，相关的 Verilog 代码如下：

```
wire i_beq = ~op[5] & ~op[4] & ~op[3] & op[2] & ~op[1] & ~op[0];
assign sext = i_addi | i_lw | i_sw | i_beq | i_bne;
```

在完成了以上步骤之后，一个基本的单周期 CPU 便被粗略地实现了。

ModelSim 仿真模拟

为了确认上一步实现的单周期 CPU 能够正常地工作，采用 ModelSim 对 sc_computer 模块进行仿真模拟，通过观察模拟过程中的波形是否正确来判断单周期 CPU 是否正常工作。

那么，首先需要编写 testbench，产生仿真所需的输入信号。参考给定的 sc_computer_sim.v 文件，去除其中 IO 拓展相关的输入输出信号，如 in_port_sim 和 out_port_sim，便可进行模拟。

在 ModelSim 中，载入模拟所涉及的 Verilog 源文件并编译，以 sc_computer_sim 模块为模拟对象，观察其波形输出。

由于在实验过程中我忘记保存此时的输出波形图，在此暂时无法附上该图片。

通过对比模拟的波形输出与正确的波形输出（人脑模拟执行流程并分析得到），基本可以确认单周期 CPU 实现正确。

外部 IO 拓展

考虑到 DE1-SOC Cyclone V 开发板上可用的输入主要有 4 个按键和 10 个拨动开关，输出主要有 10 个 LED 灯和 6 个七段数码管，本次实验中我使用 10 个拨动开关及进行输入拓展（前 4 个和后 4 个用于输入运算数，中间 2 个用于指定进行何种运算），6 个七段数码管进行输出拓展（每个运算数占 2 个，运算结果占 2 个）。在此之外，还用到了按键作为 reset 信号的输入口，5 个 LED 灯作为 PC 值的指示灯。

首先，对数据主存进行修改，使其能够根据地址区分 IO 与实际内存。这一部分已给出大部分的代码，只需要根据自己的实际需求，改变输入输出的端口数量，即增加 in_port 和 out_port 的个数，较为简单在此便不多说明。

然后，对于 io_input_reg 和 io_output_reg，也只需在给出的代码基础上，增加端口个数

与对应的地址，需要注意的是用到的地址应该满足第 7 位为高，以区分实际内存中的地址。

最后，需要完成 IO 输入输出信号的转换：从实际的 4 位开关信号或 2 位开关信号转换为标准的 32 位输入信号，从标准的 32 位输出信号转换为两个七段数码管显示所需要的 14 位信号。为了实现这些功能，我额外实现了一些模块：in_port_translator，out_port_translator，input_extender 等。以其中较为复杂的 out_port_translator 模块为例，其作用是将 32 位信号转换为 14 位信号用于两个七段数码管显示（其中涉及的 sevenseg 模块在上一次实验中已定义，在此不再重复贴出）：

```
module out_port_translator (value, out);
    input [31:0] value;
    output [13:0] out;
    wire [3:0] data1,data2;
    wire [6:0] out1,out2;
    sevenseg value_display_high(data1, out1);
    sevenseg value_display_low(data2, out2);
    assign out = { out1,out2 };
    assign data1 = value / 10;
    assign data2 = value % 10;
endmodule
```

在完成上述步骤之后，将编写的模块转换为 Quartus 中的 Symbol，以图 1 的方式进行组合，便基本完成了单周期 CPU 的 IO 拓展。

编写程序

在完成硬件的设计之后，还需要编写指令，实现一个简易的计算器：通过开发板上的开关输入运算数与运算类型，通过七段数码管输出运算数与运算结果。大致的汇编指令如下所示：

```
BEGIN:  ori $1,$0,192
        ori $2,$0,196
        ori $3,$0,200
        ori $4,$0,128
        ori $5,$0,132
        ori $6,$0,136
        ori $20,$0,0
        ori $21,$0,1
        ori $22,$0,2
        ori $23,$0,3
LOOP:   lw $7,0($1)
        sw $7,0($4)
        lw $8,0($2)
        sw $8,0($5)
        lw $9,0($3)
        beq $9,$20,ADD
```

```
            beq $9,$21,SUB
            beq $9,$22,AND
            beq $9,$23,OR
ADD:        add $10, $7, $8
            j END
SUB:        sub $10,$7,$8
            j END
AND:        and $10,$7,$8
            j END
OR:         or $10,$7,$8
            j END
END:        sw $10,0($6)
            j LOOP
```

再根据 MIPS 指令的格式，将这些汇编指令转换为机器码的形式，放入 inst_mem 的初始化文件（.mif）中即可。

一些问题

虽然本次实验总的流程并不复杂，但是我在实际实验中遇到了不少问题，这些问题往往十分细节且不够典型，然而耗费了我大量的时间进行调试和排错。

第一，CPU 运行的频率问题。一开始我并未对此加以关心，而是直接将板子上的 50MHz 信号作为 CPU 的输入频率，然而这样不仅导致了运行的出错，而且使得调试格外困难。因此，为了方便调试，我将板子上的 5 个 LED 灯作为 PC 值的指示，用以实时观察 CPU 的运行状态；当然，还需要降低 CPU 的运行频率，不然无法观察 LED 灯。所以，在将板载时钟信号输入到 CPU 之前，我先对其进行了降频处理（所涉及的模块为 frequence_divider），而后把信号和信号的二分频作为 mem_clk 和 clock 信号输入。

第二，Verilog 中变量的初始化问题。在基本完成了 IO 拓展实验之后，我用 ModelSim 对其进行模拟，得到了正确的运行结果，十分满意；然而当我编译并烧录到开发板上后，始终得不到正确的结果。反复调试，反复排错，耗费了两三天的时间，最终发现了症结所在：在 datamem 中，我忘记了对某一个变量进行初始化，而 ModelSim 模拟时和实际运行时，这个变量的初始值恰恰相反，从而导致了行为的异常。

实验总结

在本次实验中，我第一次较为完整地实现了一个单周期 CPU，在调试与排错中，我对 Quartus 和 ModelSim 的使用渐渐熟练，也更加深刻地理解了 Verilog 语言用于硬件描述的功能。虽然本次实验大部分的代码已经给出，但是通过反复的调试，实际上我对大部分的代码都已仔细阅读过，并更加深入地理解了单周期 CPU 的完整运行流程，收益颇丰。