版本历史

文档更新记录	文档名:	Lab04_软件编程电子表
	版本号	V0.1
	创建人:	计算机体系结构研讨课教学组
	创建日期 :	2017-11-1

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2017/11/1	邢金璋	V0.1	初版。

文档信息反馈: xingjinzhang@loongson.cn

1 实验四 软件编程电子表

在学习并尝试本章节前，你需要具有以下环境和能力：

(1) 较为熟练使用 Vivado 工具。

(2) 一定的 MIPS 汇编编程、编译能力。

(3) 一定的自学能力。

通过本章节的学习，你将获得：

(1) MIPS 指令的使用。

(2) CPU 部分系统控制寄存器（CP0 寄存器）的知识。

(3) MIPS 中断产生、标记和处理的知识。

在本章节的学习过程中，你可能需要查阅：

(1) 文档“A05_“体系结构研讨课”MIPS 指令系统规范”。

(2) MIPS32 官方文档的卷 II 和卷 III。

(3) 书籍“See MIPS RUN”第 9 章，学习 MIPS 汇编。

在开展本次实验前，请确认自己知道以下知识：

(1) Lab3 实现的 56 条指令的明确意义。

(2) 理解 CONFREG 模块里的数码管、矩阵键盘寄存器的访问方式。

(3) MIPS 汇编指令和机器指令的区别，比如汇编指令“li”、“la”对应的机器指令是什么？你们在 lab3 里实现的 56 条指令是汇编指令，还是机器指令？（对应多个机器指令的汇编指令往往又称为宏指令）

(4) 汇编指令编程时，跳转指令后接的标号“1b”、“1f”、“1000b”等等这些“数字+b/f”的标号代表什么意思？

(5) MIPS 的汇编指示命令（又称为伪操作）的作用，比如.set、.org、.word、.global、.reorder 等等。

(6) MTC0 和 MFC0 指令的定义和使用。

(7) CP0 寄存器 COUNT 的定义和作用。

为采用中断完成本次实验，请确认自己知道以下知识：

(1) ERET 指令的定义和使用。

(2) CP0 寄存器 STATUS、CAUSE、COMPARE、EPC 的定义和作用。

(3) 特别是 STATUS.IM、STATUS.IE、STATUS.EXL、STATUS.ERL、CAUSE.TI、CAUSE.IP 域的定义和作用。

(4) MIPS 架构的硬件中断机制，特别是中断的响应、开启/关闭、入口地址。

(5) MIPS 架构的硬件中断、时钟中断的产生机制和清除机制。

1.1 实验目的

1. 从应用层次理解 MIPS 架构、CPU 指令集的定义。
2. 初步理解 MIPS 架构的硬件中断机制。

1.2 实验设备

1. 装有 Xilinx Vivado、MIPS 交叉编译环境的计算机一台。
2. 龙芯体系结构教学实验箱（Artix-7）一套。

1.3 实验任务

编译一段汇编程序，运行在 SoC_Lite 上，调用 Confreg 模块的数码管和按钮开关等外设，实现一个 12 /24 小时进制的电子表，并在实验板上予以演示。

该电子表的显示包含时、分、秒，采用实验箱开发板上的 4 组数码管显示，并通过板上的矩阵键盘完成电子表的设置功能。具体要求是：

(1) 电子表具有一个暂停/启动键，具有时、分、秒设置键。

(2) 电子表复位结束后从 0 时 0 分 0 秒开始计时，按下暂停/启动键一次则计时暂停进入设置模式，此时可以通过时、分、秒的设置键修改时、分、秒的值，再次按下暂停/启动键则推出设置模式并从设置好的时间开始继续计时。

(3) 时、分、秒设置键的设置方式是每按下一次，对应的时、分、秒值循环加 1，按住不放则按照一定频率不停地循环加 1 直至按键松开。

(4) 时、分、秒设置键仅在设置模式下操作才有效果。

(5) 矩阵键盘上非设置键被按下，应当不影响电子表的精确计时。

(6) 采用硬件中断+时钟中断完成本次实验的满分是 100%，采样硬件中断或时钟中断完成本次实验的满分是 95%，所有功能都是采用轮询完成的本次实验的满分是 80%。

(7) 时间有余的可以考虑实现闹铃功能，但这不在本次实验要求范围内，也不承诺有本课程最终分数上的加分。

1.4 实验环境

本次实验实验环境分为软件编程环境和硬件运行环境：

(1) 软件编程环境是 lab3_func_4。请将 start.S 里 60 行开始到文件尾的代码都删除，然后开始编写本次实验电子表程序。

(2) 硬件运行环境是 ucas_cde_v0.3/cpu132_gettrace。

1.5 实验检查

检查前需提交实验报告和软件程序代码，代码应当有必要的注释。本次实验在 2017 年 11 月 7 日进行检查。

现场分仿真检查和上板检查，所以请设置好仿真和上板时 wait_1s 的循环次数：

1) 仿真检查，需要准确看到每过一定的时间（比如 10us），数码管寄存器显示的时钟秒钟上加 1。最好在 testbench 里增加一定的按钮开关的激励，以模拟电子表设置的功能。

2) 上板检查，需要准确看的每过 1s，板上数码管秒钟加 1。且可以通过按键进行电子表设置。我们会检查 2~3 分钟，电子表的计时误差。

1.6 实验说明

1.6.1 快速仿真

本次实验时进行软件编程，需求进行多次仿真，确认软件程序行为正确。其实，如果我们只关心仿真，那不需要替换 inst 和 data ram 里的数据，只需要替换 mif 文件即可。下面提供一个快速开始仿真的方法：

(1) 打开 cpu132_gettrace 里工程的仿真界面，不需要管 ram 里的数据。

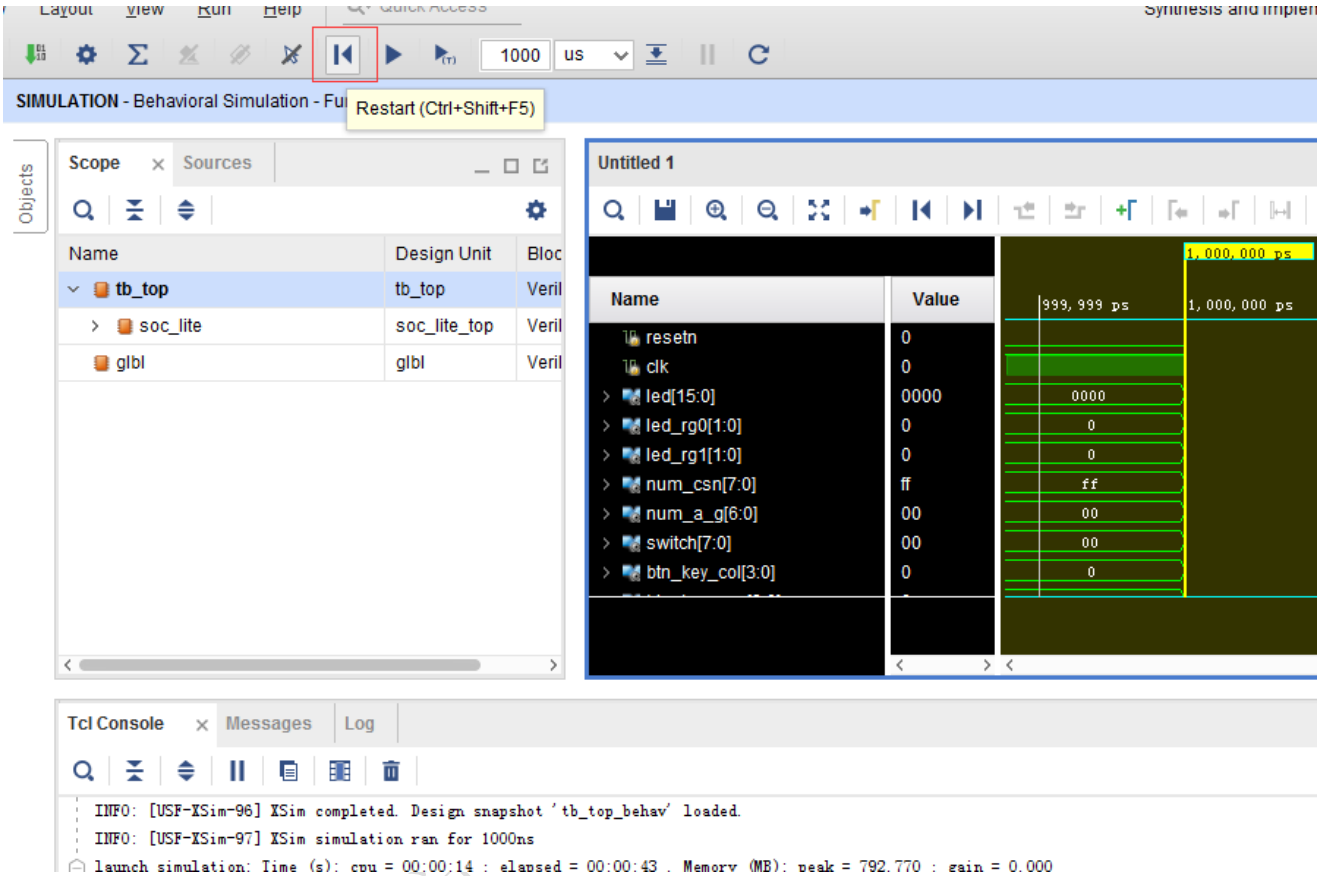
(2) 编译电子表软件程序，将编译结果 obj/ 下的 inst_ram.mif 和 data_ram.mif 拷贝到

cpu132_gettrace/run_vivado/cpu132_gettrace/cpu132_gettrace.sim/sim_1/behav/下替换里面原先的 2 个 mif 文件。

(3) 在仿真界面里，先点击 restart 回到仿真 0 时刻。如下图。

(4) 之后，再点击 Run all 或 Run for，正常运行仿真即可。此时加载到 RAM 里的数据即是新拷贝到 sim_1/behav/里的 2 个 mif 文件。

(5) 注意，千万不要点击 relaunch 按钮，这一按钮会重新生成仿真的环境，此时 sim_1/behav/里的文件会重新生成出来，也就覆盖了两个 mif 文件。如果你点击了 relaunch，请重新拷贝 mif 文件。



1.6.2 gs132 debug 指导

cpu132_gettrace 里使用的是龙芯开源的 gs132 处理器核。这个处理器核有以下特点：

(1) 三级流水：取指、译码和执行。

(2) 分支跳转指令在执行级处理：所以即使有延迟槽技术，分支跳转指令如果需要跳转，则必定给流水线引入一个空泡；如果分支跳转指令不跳转，则是连续先后取指，也就是猜测永远不 taken 的分支预测。

(3) 如果需要观察取指级的信息，请抓出 ls132r_fetch_stage.v 里的 new_inst_in、new_inst_code 和 inst_pc_r，对于就是新指令返回了、新指令的编码和对应的 PC。

(4) 如果需要观察译码级的信息，请抓出 ls132r_fetch_stage.v 里的 ir_valid_r、ir_inst_r 和 ir_pc_r，这就是译码级有有效指令、该指令编码和对应的 PC 值。

(5) 如果需要观察执行级的信息，请抓出 ls132r_execute_stage.v 里的 rs_valid_r、debug_wb_pc、debug_wb_rf_wen、debug_wb_rf_wnum 和 debug_wb_rf_wdata。

(6) 如果需要看 32 个通用寄存器里的某个寄存器号的值，请抓出 ls132r_decode_stage.v 里的 u0_gr_heap 子模块里的 heap_*，信号名 heap_00~heap_31 对应 0~31 号通用寄存器。

(7) 如果需要观察 CP0 寄存器，请抓出 ls132r_execute_stage.v 里的 status_value、cause_value、count_value 和 compare_value，还可以通过 cr_status_*抓出 status 寄存器里特定域的信号。

1.6.3 电子表设置功能仿真激励

在仿真验证设置功能时，需要自己编写激励。

所谓编写激励，就是在 testbench 里增加模拟按键的功能。这不需要我们直接控制矩阵键盘引脚上的信号。我们可以直接在 tb_top 里给 soc_lite.confreg. btn_key_r[15:0]进行赋值，对于 16 个按键。所以电子表设置功能的激励就类似下面的写法：

```
initial
begin
    #n;  //等 n ns
    force soc lite.confreg. btn key r[15:0]=16'h0001;
    #m;  //等 m ns
    release soc_lite.confreg. btn_key_r[15:0]; //强迫按键赋值持续 m ns
    #x;  //等 x ns
    force soc lite.confreg. btn key r[15:0]=16'h0008;
    #y;  //等 y ns
    release soc_lite.confreg. btn_key_r[15:0]; //强迫按键赋值按键持续 y ns
    …
end
```

1.6.4 轮询 COUNT 寄存器的关键点说明

在课上举例了轮询 COUNT 寄存器的代码，如下：

```
Li   t6 250
1:
    mfc0 t7, $9
    nop
    nop
    nop
    nop
    nop
    bne t7, t6, 1b
    nop
```

但其实上述代码存在问题，问题在于：由于 nop 和 bne 指令的存起，在第 n 次读 mfc0 后，到第 n+1 次读 mfc0，已经过了 8 拍左右的 CPU clk，而由于 cp0 寄存器 COUNT 是每过 2 个 CPU clk 加 1，所以很有可能错过了 COUNT 等于 250 的时刻，导致上诉程序陷入死循环了。有 3 个方法:

(1) 将 bne 改为 blt 指令，也就是大于 250 时即跳出循环，但这存在计时不精确的问题。

(2) 使用 mfc0 和 beq/bne 穿插的方法，也就确保 2 个 CPU clk 内记录一次 COUNT 值。这样也可以实现计时精确。

(3) 采用时钟中断。

1.6.5 中断的进入和退出。

当有中断进来时，CPU 会执行以下动作：

(1) 保存发生中断的指令的 PC 值到 CP0 寄存器 EPC 中。

(2) 置上 STATUS.EXL，以表示 CPU 进入例外级别，同时起到屏蔽中断的作用。

(3) CPU 转到 0xbfc00380 处执行。

当中断处理完退出时，软件需要完成以下动作：

(1) 清除 STATUS.EXL，表示 CPU 退出例外级别，同时可以开启中断。

(2) 如果中断源没有自动撤销中断请求，还需要清除中断源的中断请求。比如时钟中断需要执行一条对 COMPARE 寄存器的 mtc0 指令，才会清除。

(3) 将 CPU 转到需要处继续执行。这一个需要处可以是中断发生处，也可以是其他指令处。

通常退出中断处理程序时，软件会使用机器指令 ERET，该指令的作用是：

(1) 清除 STATUS.EXL。

(2) 取出 EPC 里保存的 PC 值，转到该 PC 处继续执行。

(3) 显然 ERET 会返回中断发生处继续执行。如果想转到其他指令处执行，可以先使用 mtc0 将 EPC 寄存器里

的值更改为目标指令处。这样 ERET 执行就会转到我们想要的指令处执行了。

1.6.6 时钟中断使用

MIPS 架构有 2 个 CP0 寄存器，COUNT 和 COMPARE，其中 COUNT 是自累加的，而 COMPARE 是软件指令设定后不变的。当 COUNT 与 COMPARE 相等时，触发时钟中断，置上 CAUSE.TI 位。

MIPS 架构共支持 8 个中断，分为 6 个硬件中断，2 个软件中断：

(1) CAUSE.IP[7:0]域用于采样这 8 个中断。

(2) STATUS.IM[7:0]域为对应的屏蔽为，为 0 时屏蔽。

(3) STATUS.IE 可用于开/关所有中断，为 1 时关闭。

(4) 6 个硬件中断通过 gs132 顶层的 int_n_i[5:0]接入，低电平有效，被 CAUSE.IP[7:2]采样锁存。

(5) 当 CPU 已经处于错误或例外级别时，会自动屏蔽中断。也就是当 STATUS.ERL 和 STATUS.EXL 任意一个为 1 时，会屏蔽中断。

时钟中断 CAUSE.TI 是与 int_n_i[5]作或运算，再一起被 CAUSE.IP[7]采样。

也就是说，要使用时钟中断，需要确保：

(1) STATUS.IE 为 1。

(2) STATUS.IM[7]为 1.

(3) STATUS.ERL 和 STATUS.EXL 为 0

使用时钟中断时，通常先给 COMPARE 写一个固定值，然后清零 COUNT，之后等待时钟中断到来，硬件会自动转到 0xbfc00380 处执行。

时钟中断处理完，需要需要清除时钟中断，同时要清除 STATUS.EXL（进入中断时，硬件会自动置上 STATUS.EXL）：

(1) 清除时钟中断，也就是为了清除 CAUSE.TI，方法是再写一次 COMPARE。CPU 检测到对 COMPARE 的写会自动清除 CAUSE.TI。

(2) 清除 STATUS.EXL，可以通过 mtc0 写 STATUS，也可以执行 ERET 指令。

本次实验中，可以使用时钟中断来完成电子表的精确计时。

1.6.7 硬件中断使用

本次实验中，硬件中断的使用与时钟中断类似，但可以不需要软件清除中断，因为一次按键即是一次中断，按键松下的时刻，即是中断清除的时刻。需要注意，在设置时、分、秒时，如果一直按住按键，则需要时、分、秒不停累加。

硬件中断需要在 SoC_Lite 源码里将硬件中断信号连接到 CPU 的 int_n_i 上。为了避免与时钟中断作用到同一根请求信号上，硬件中断最好不要连接到 int_n_i[5]上，该信号对应 CAUSE.IP[7]。

如果本次实验硬件中断和时钟中断同时使用了，那在进入中断时，首先要甄别到来的是时钟中断还是硬件中断。这一甄别过程可以通过读取 CAUSE.IP 来判定。

本次实验硬件中断可以用来完成以下 3 项工作：

(1) 电子表从计时模式进入设置模式。

(2) 电子表从设置模式退出到计时模式。

(3) 设置模式下设置时、分、秒。

本次实验可以任选以上一项或多项来使用硬件中断完成。