

****

**实 验 报 告**

**课 程 名 称 嵌入式系统原理**

**专 业 班 级 物联网工程22-2班**

**学生姓名及学号 党存远 2022217587**

**指 导 教 师 毕翔、张本宏**

**实 验 地 点 综合楼312实验室**

**2023 ～ 2024 学年第 二 学期**

《嵌入式系统原理》课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | **实验一 汇编点亮LED灯** | | |
| 实验日期 | 2024.6.12 | 实验成绩 |  |
| **一、实验目的和要求**  **目的:**学会Linux系统中开发C程序的步骤和方法。在此基础上，掌握通过汇编程序实现Tiny6410初始化及通过C程序实现通用输入输出端口（GPIO）数据操作的方法。  **要求**:在延时不变的条件下，将源程序中“四个LED全亮，然后全灭”的显示状态，修改为循环跑马灯效果，即“LED1亮→LED2亮→LED3亮→LED4亮→LED3亮→LED2亮→LED1亮→LED2亮→……” | | | |
| **二、实验内容（包括实验步骤和代码）**  本次实验使用Fedora（合肥校区）/CentOS（宣城校区）操作系统环境，安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程，即创建一个新目录leds\_s，使用编辑器建立start.S和Makefile文件，并使用汇编语言编写LED控制程序。编译程序，并下载文件到目标开发板上运行。   1. 建立工作目录leds。   首先将预先提供的实验源码复制到Windows系统桌面上，再点击【虚拟机】菜单中的【设置】，选择【选项】中的“共享文件夹”，添加Windows系统中的桌面路径为共享文件夹，然后鼠标右键复制Windows系统桌面上的key\_led文件夹（内含源代码和Makefile文件），接着进入虚拟机当前用户的Home（合肥校区）/root（宣城校区）目录，使用鼠标右键进行粘贴，从而将文件夹从Windows系统复制到虚拟机的系统中。   1. 编写程序源代码。 2. start.S修改前汇编代码源程序如下:      1. Makefile文件如下:     在Makefile所在目录下执行make命令时，系统会执行如下操作：  ● 执行arm-Linux-gcc -o $@ $< -c命令，将当前目录下存在的汇编文件和C文件编译  成.o文件；  ● 执行arm-Linux-ld -Ttext 0x50000000 -o led.elf $^，将所有.o文件链接成elf  文件，-Ttext 0x50000000表示程序的运行地址是0x50000000，即程序只有位于该  地址上才能正常运行；  ● 执行arm-Linux-objcopy -O binary led.elf led.bin，将elf文件抽取为可在开发  板上运行的bin文件；  ● 执行arm-Linux-objdump -D led.elf > led\_elf.dis，将elf文件反汇编后保存在dis  文件中，调试程序时可能会用到。   1. 编译及下载运行程序 2. 编译代码   确保当前用户为root用户（可使用su root命令切换到root用户）的条件下，在Fedora/CentOS的终端中执行如下命令：  # cd leds  # make  执行make后会生成led.bin文件。   1. 下载（烧写）和运行程序   在6410开发板断电的情况下，先用MicroUSB线连接开发板与PC机，并将领取（已烧写了Superboot-6410.bin）的SD卡插入开发板的SD卡槽，同时确认启动模式拨动开关S2拨在“SDBOOT”位置，然后打开开发板的电源。此时开发板将进入USB下载模式，LCD显示屏上显示“USB Mode: Waiting...”。接着，在Windows系统中以管理员身份运行MiniTools。若USB连接成功，则开发板LCD屏幕上显示“USB Mode: Connected”，当显示连接成功或者发现USB连接断开时可以先尝试关闭开发板电源，然后重新上电进行连接。   1. 修改要求   在延时不变的条件下，将源程序中“四个LED全亮，然后全灭”的显示状态，修改为循环跑马灯效果，即“LED1亮→LED2亮→LED3亮→LED4亮→LED3亮→LED2亮→LED1亮→LED2亮→……”   1. 实现方式   修改汇编文件代码依次对GPKDAT中传入不同值来表示不同的灯的亮与灭，同时设置延时时间。对于Led灯的亮灭，硬件原理如下:    由上述硬件原理图不难看出发光二极管是共阳极，因此当GPIO端口设置为输出同时为低电平时表示LED灯亮，而输出为高电平表示灯光灭。基于此，需要设置在不同时间下GPKDAT中的值不断变化以此满足要求。首先当开始后LED1灯亮，对应结果为GPK4的输出为低电平，但同时需要保证其它的端口不能输出为低电平因此需要设置其余端口为高电平。因此不同情况下GPKDAT中的数值设置为:LED1(0XE0),LED2(0XD0), LED3(0XB0), LED2(0X70)对于流水灯操作，我们需要保证每次一种情况结束后延时一段时间后重新显示灯光情况，因此需要设置延时时间，示例代码中已经给出延时效果的编写，采用对数值进行减减的操作来完成延时的效果。对于循环效果使用手动设置，当LED4亮后设置GPKDAT的值变为LED3亮的情况，以此类推实现全部操作。   1. 修改后代码:   .global \_start  \_start:  *; 设置外设的基地址*      ldr r0, =0x70000000      orr r0, r0, #0x13      mcr p15, 0, r0, c15, c2, 4  *; 关闭看门狗*      ldr r0, =0x7E004000      mov r1, #0      str r1, [r0]  *; 设置GPKCON0寄存器*  *; r2表示显示程序的运行次数*      ldr r1, =0x7F008800      ldr r0, =0x11110000      str r0, [r1]      mov r2, #0x1000  led\_blink:  *;  LED1亮，其余熄灭*      ldr r1, =0x7F008808      mov r0, #0xE0      str r0, [r1]      bl delay  *; LED2亮，其余熄灭*      ldr r1, =0x7F008808      mov r0, #0xD0      str r0, [r1]      bl delay  *; LED3亮，其余熄灭*      ldr r1, =0x7F008808      mov r0, #0xB0      str r0, [r1]      bl delay  *; LED4亮，其余熄灭*      ldr r1, =0x7F008808      mov r0, #0x70      str r0, [r1]      bl delay  *; LED3亮，其余熄灭*      ldr r1, =0x7F008808      mov r0, #0xB0      str r0, [r1]      bl delay  *; LED2亮，其余熄灭*      ldr r1, =0x7F008808      mov r0, #0xD0      str r0, [r1]      bl delay  *; 循环控制*      sub r2, r2, #1      cmp r2, #0      bne led\_blink  halt:      b halt  *; 延时时间的设置*  delay:      mov r0, #0x1000000  delay\_loop:      cmp r0, #0      sub r0, r0, #1      bne delay\_loop      mov pc, lr | | | |
| **三、实验验收结果及其分析（可附实验结果图片）**  **1.实验结果显示**   1. LED1灯亮，其余灯均灭。      1. LED2亮，其余灯灭。      1. LED3亮，其余灯灭      1. LED4亮，其余灯灭      1. LED3亮，其余灯灭      1. LED2亮，其余灯灭。      1. LED1灯亮，其余灯均灭。     **2.实验结果分析**  如上面所列的图片可以看到，要求是将灯由全亮然后全灭的状态改变为循环跑马灯效果，因此只需要在汇编文件中更改灯的状态也就是GPKDAT的每一位来显示操作，特别注意，GPKDAT中当某一位为0时此时表示该位置的灯是亮的状态。在编写代码中还要注意最终要的是如何进行延时，采用汇编中的代码，通过设置比较r2为一个所需要延时的时间的数值，每次进行-1操作直至r2为0时表示显示结束，此时表示延时循环结束。对于流水灯的循环设置，我需要将灯的对应位打开后延迟一段时间之后重新更换状态就可以改变其状态。上图中可以看出实验结果与要求相同。 | | | |

《嵌入式系统原理》课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | **实验三 按键中断实验** | | |
| 实验日期 | 2024.6.12 | 实验成绩 |  |
| **一、实验目的和要求**  **目的**:学会Linux系统中开发C程序的步骤和方法。在次基础上，掌握通过汇编程序实现Tiny6410中断控制器初始化以及通过C程序实现中断处理的方法。  **要求**: 当K1按下时，LED1到LED4依次被点亮；当K2按下时，LED4到LED1依次被点亮；当K3按下时，LED1到LED4依次被点亮，且每个时刻只有一个LED亮；当K4按下时，LED4到LED1依次被点亮，且每个时刻只有一个LED亮。 | | | |
| **二、实验内容（包括实验步骤和代码）**  你好本次实验使用Fedora（合肥校区）/CentOS（宜城校区）操作系统环境，安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程，即创建一个新目录irq，使用编辑器建立start.S、main.c、irq.c和Makefile等文件，并使用C语言编写中断处理程序。编译程序，并下载文件到目标开发板上运行。   1. 建立工作目录irq。   首先将预先提供的实验源码复制到Windows系统桌面上，再点击【虚拟机】菜单中的【设置】，选择【选项】中的“共享文件夹”，添加Windows系统中的桌面路径为共享文件夹，然后鼠标右键复制Windows系统桌面上的irq文件夹（内含源代码和Makefile文件），接着进入虚拟机当前用户的Home（合肥校区）/root（宜城校区）目录，使用鼠标右键进行粘贴，从而将文件夹从Windows系统复制到虚拟机的系统中。   1. 编写程序源代码   start.S的汇编源程序如下:  .global \_start  .global asm\_k1\_irq  .extern do\_irq  start:  reset:  *; 初始化并告诉CPU外设的基地址*      ldr r0, =0x70000000      orr r0, r0, #0x13      mcr p15, 0, r0, c15, c2, 4  *; 关闭看门狗*      ldr r0, =0x7E004000      mov r1, #0      str r1, [r0]  *; 初始化VIC*      mcr p15, 0, r0, c1, c0, 0      orr r0, r0, #(1<<24)      mcr p15, 0, r0, c1, c0, 0  *; 栈初始化*      ldr sp, =8\*1024  *; 初始化时钟*      bl clock\_init  *; 初始化DDR*      bl sdram\_init  *; 初始化NAND Flash*      bl nand\_init  *; 初始化IRQ*      bl irq\_init  *; 开始用户程序*  *; mov r0, #0x53*  *; msr CPSR\_cxsf, r0*      mrs r0, cpsr      bic r0, r0, #0x80      msr cpsr\_c, r0  *; 清理和拷贝数据段*      adr r0, \_start      ldr r1, =\_start      ldr r2, =bss\_start      sub r2, r2, r1      cmp r1, r1      beq clean\_bss      bl copy2ddr      cmp r0, #0      bne halt  clean\_bss:  *; 清除BSS段*      ldr r0, =\_bss\_start      ldr r1, =\_bss\_end      mov r3, #0      cmp r0, r1      beq on\_ddr  clean\_loop:      str r3, [r0], #4      cmp r0, r1      bne clean\_loop  on\_ddr:  *; 跳转到主程序*      ldr pc, =main  asm\_k1\_irq:  *; 中断服务程序*      .word irq  irq:  *; 保存寄存器*      ldr sp, =0x54000000      sub lr, lr, #4      stmfd sp!, {r0-r12, lr}  *; 处理中断*      bl do\_irq  *; 恢复寄存器*      ldmfd sp!, {r0-r12, pc}^  *; ^表示把spsr恢复到cpsr*  halt:      b halt  irq.c文件如下:  #include "stdio.h"  #define GPKCON0 (\*((volatile unsigned long \*)0x7F008800))  #define GPKDATA (\*((volatile unsigned long \*)0x7F008808))  #define GPNCON (\*((volatile unsigned long \*)0x7F008830))  #define GPNDAT (\*((volatile unsigned long \*)0x7F008834))  #define EINT0CON0 (\*((volatile unsigned long \*)0x7F008900))  #define EINT0MASK (\*((volatile unsigned long \*)0x7F008920))  #define EINT0PEND (\*((volatile unsigned long \*)0x7F008924))  #define PRIORITY (\*((volatile unsigned long \*)0x7F008280))  #define SERVICE (\*((volatile unsigned long \*)0x7F008284))  #define SERVICEPEND (\*((volatile unsigned long \*)0x7F008288))  #define VIC0IRQSTATUS (\*((volatile unsigned long \*)0x71200000))  #define VIC0FIQSTATUS (\*((volatile unsigned long \*)0x71200004))  #define VIC0RAWINTR (\*((volatile unsigned long \*)0x71200008))  #define VIC0INTSELECT (\*((volatile unsigned long \*)0x7120000c))  #define VIC0INTENABLE (\*((volatile unsigned long \*)0x71200010))  #define VIC0INTENCLEAR (\*((volatile unsigned long \*)0x71200014))  #define VIC0PROTECTION (\*((volatile unsigned long \*)0x71200020))  #define VIC0SWPRIORITYMASK (\*((volatile unsigned long \*)0x71200024))  #define VIC0PRIORITYDAISY (\*((volatile unsigned long \*)0x71200028))  #define VIC0ADDRESS (\*((volatile unsigned long \*)0x71200f00))  typedef void(isr)(void);  extern void asm\_k1\_irq(); *// 汇编写的中断处理函数，该函数具体实现中断时需要执行的操作。*  void irq\_init(void)  {  */\* 配置GPN0~4引脚为中断功能 \*/*    GPNCON &= ~(0xff);    GPNCON |= 0xaa; *// 1010 1010*  */\* 设置中断触发方式为: 下降沿触发 \*/*    EINT0CON0 &= ~(0xff);    EINT0CON0 |= 0x33; *// 0 011 0 011  01x表示下降沿触发*  */\* 禁止屏蔽中断 \*/*    EINT0MASK &= ~(0x0f); *// 0000 1111*  *// Select INT\_EINT0 mode as irq*    VIC0INTSELECT = 0;  */\* 在中断控制器里使能这些中断 \*/*    VIC0INTENABLE |= (0x1); *// 0001  /\* bit0: eint0~3, bit1: eint4~11 \*/*    isr \*\*isr\_array = (isr \*\*)(0x71200100);    isr\_array[0] = (isr \*)asm\_k1\_irq;  *//*  */\*将GPK4-GPK7配置为输出口\*/*    GPKCON0 = 0x11110000;  */\*熄灭四个LED灯\*/*    GPKDATA = 0xf0;  }  void dely1(volatile int *n*)  {    while (*n*--)    {    }  }  void dely(volatile int *n*)  {    while (*n*--)    {      dely1(1000);    }  }  void do\_irq(void)  {    int i = 0;  *// GPKDATA = 0x00;*  */\* 分辨是哪个中断 \*/*    if (EINT0PEND & (1 << 0))    {      GPKDATA = 0xe0; *// 1110 0000*      dely(1000);      GPKDATA = 0xc0; *// 1100 0000*      dely(1000);      GPKDATA = 0x80; *// 1000 0000*      dely(1000);      GPKDATA = 0x00; *// 0000 0000*      dely(1000);    }    if (EINT0PEND & (1 << 1))    {      GPKDATA = 0x70; *// 0111 0000*      dely(1000);      GPKDATA = 0x30; *// 0011 0000*      dely(1000);      GPKDATA = 0x10; *// 0001 0000*      dely(1000);      GPKDATA = 0x00; *// 0000 0000*      dely(1000);    }    if (EINT0PEND & (1 << 2))    {      GPKDATA = 0xe0; *// 1110 0000*      dely(1000);      GPKDATA = 0xd0; *// 1101 0000*      dely(1000);      GPKDATA = 0xb0; *// 1011 0000*      dely(1000);      GPKDATA = 0x70; *// 0111 0000*      dely(1000);    }    if (EINT0PEND & (1 << 3))    {      GPKDATA = 0x70; *// 0111 0000*      dely(1000);      GPKDATA = 0xb0; *// 1011 0000*      dely(1000);      GPKDATA = 0xd0; *// 1101 0000*      dely(1000);      GPKDATA = 0xe0; *// 1110 0000*      dely(1000);    }  */\* 清中断 \*/*    EINT0PEND = 0x3f; *// 0011 1111 写入0表示清除中断*    VIC0ADDRESS = 0; *// 中断向量地址寄存器*  }  main.c文件如下:  #include "stdio.h"  int main()  {    while (1)    {    }    return 0;    }  Makefile文件如下:     1. 编译及下载运行程序   编译代码  确保当前用户为root用户(可使用su root命令切换到root用户)的条件下，在Fedora/CentOS的终端中执行如下命令:  # cd  # make  执行make后会生成irq.bin文件  下载(烧写)和运行程序  按实验一给出的方法下载程序。   1. 修改要求:   当K1按下时，LED1到LED4依次被点亮；当K2按下时，LED4到LED1依次被点亮；当K3按下时，LED1到LED4依次被点亮，且每个时刻只有一个LED亮；当K4按下时，LED4到LED1依次被点亮，且每个时刻只有一个LED亮。   1. 实现方法:   通过外部中断挂起寄存器来判断时哪一个中断源产生了中断，由于没有设置其优先级因此每个中断源之间的优先级是相同的，判断完是哪一个中断源产生的中断之后，就可以根据对应的中断去实现不同的逻辑，对于灯的亮灭调整，我只需要改变GPKDAT的数值便可以修改每个灯的显示状态，以此实现所需要求。 | | | |
| **三、实验验收结果及其分析（可附实验结果图片）**  **1.实验结果显示**   1. K1按下后（LED1到LED4依次被点亮）        1. K2按下后（LED4到LED1依次被点亮）        1. K3按下后（LED1到LED4依次被点亮,且每一时刻只有一个灯亮）        1. K4按下后（LED4到LED1依次被点亮,且每一时刻只有一个灯亮）       **2.实验结果分析**  通过中断方式实现定时器时，需要在主程序中循环，知道外部条件触发中断时会触发相应的操作，每当出现中断时会由汇编代码调用C语言程序中的do\_irq的函数来处理中断。在do\_irp中通过外部中断挂起寄存器，每当发生中断时会先对中断挂起寄存器中的值进行判断，用于判断是哪一种中断，每当触发该种中断时，外部中断挂起寄存器中的数值将会被置为1.对于K1中断，要求实现LED1->LED4依次被点亮，通过更改GPKDAT的状态即可实现，但需要注意状态循环之后要保证延时，来显示变化的效果。K2与K1实现方式类似；对于K3中断，同样我需要设置状态位为不同的值来显示不同的效果 | | | |

《嵌入式系统原理》课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | **实验四 PWM定时器实验** | | |
| 实验日期 | 2024.6.23 | 实验成绩 |  |
| **一、实验目的和要求**  **目的**:学会Linux系统中开发C程序的步骤和方法。在此基础上，掌握掌握通过C程序实现Tiny6410定时器初始化及中断处理的方法。  **要求:** 每间隔2秒，四个LED跑马灯方式循环显示(LED1亮→LED2亮→LED3亮→LED4亮→LED1亮→LED2亮→LED3亮→LED4亮→……) | | | |
| **二、实验内容（包括实验步骤和代码）**  本次实验使用Fedora（合肥校区）/CentOS（宣城校区）操作系统环境，安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程，即创建一个新目录timer，使用编辑器建立start.S、main.c、timer.c和Makefile文件。编译程序，并下载文件到目标开发板上运行。   1. 建立工作目录leds。   首先将预先提供的实验源码复制到Windows系统桌面上，再点击【虚拟机】菜单中的【设置】，选择【选项】中的“共享文件夹”，添加Windows系统中的桌面路径为共享文件夹，然后鼠标右键复制Windows系统桌面上的timer文件夹（内含源代码和Makefile文件），接着进入虚拟机当前用户的Home（合肥校区）/root（宣城校区）目录，使用鼠标右键进行粘贴，从而将文件夹从Windows系统复制到虚拟机的系统中。   1. 编写程序源代码。   start.S的汇编代码源程序如下:  .global \_start *; 程序入口*  .global asm\_timer\_irq *; 定义全局变量用于中断处理*  .extern do\_irq *; 声明外部函数用于中断处理*  \_start:  reset:  *; 外设地址告诉cpu*      ldr r0, =0x70000000      orr r0, r0, #0x13      mcr p15, 0, r0, c15, c2, 4 *; 用于设置外设地址*  *; 关看门狗*      ldr r0, =0x7E004000      mov r1, #0      str r1, [r0]  *; 使能 VIC*      mrc p15, 0, r0, c1, c0, 0      orr r0, r0, #(1<<24)      mcr p15, 0, r0, c1, c0, 0  *; 设置栈*      ldr sp, =8\*1024  *; 初始化时钟*      bl clock\_init  *; 初始化ddr*      bl sdram\_init  *; 初始化nandflash*      bl nand\_init  *; 初始化中断*      bl irq\_init  *; 开中断*      mrs r0, cpsr  *; 清理和拷贝数据段*      adr r0, \_start      ldr r1, =\_start      ldr r2, =bss\_start      sub r2, r2, r1      cmp r0, r1      beq clean\_bss      bl copy2ddr      cmp r0, #0      bne halt  clean\_bss:  *; 清BSS，把BSS段对应的内存清零*      ldr r0, =bss\_start      ldr r1, =bss\_end      mov r3, #0      cmp r0, r1      beq on\_ddr  clean\_loop:      str r3, [r0], #4      cmp r0, r1      bne clean\_loop  on\_ddr:  *; 跳转*      ldr pc, =main *; 执行C语言所写的main函数*  *; 中断异常*  asm\_timer\_irq:      .word irq  irq:  *; 1. 保存现场*      ldr sp, =0x54000000      sub lr, lr, #4      stmf sp!, {r0-r12, lr} *; lr就是swi的下一条指令的地址*  *; 2. 处理异常*      bl do\_irq  *; 3. 恢复现场*      ldmfd sp!, {r0-r12, pc}^ *; ^表示把spsr恢复到cpsr*  halt:      b halt  timer.c文件如下:  #include "stdio.h"  #define GPKCON0 (\*((volatile unsigned long \*)0x7F008800))  #define GPKDATA (\*((volatile unsigned long \*)0x7F008808))  #define EINT0CON0 (\*((volatile unsigned long \*)0x7F008900))  #define EINT0MASK (\*((volatile unsigned long \*)0x7F008920))  #define EINT0PEND (\*((volatile unsigned long \*)0x7F008924))  #define PRIORITY (\*((volatile unsigned long \*)0x7F008280))  #define SERVICE (\*((volatile unsigned long \*)0x7F008284))  #define SERVICEPEND (\*((volatile unsigned long \*)0x7F008288))  #define VIC0IRQSTATUS (\*((volatile unsigned long \*)0x71200000))  #define VIC0FIQSTATUS (\*((volatile unsigned long \*)0x71200004))  #define VIC0RAWINTR (\*((volatile unsigned long \*)0x71200008))  #define VIC0INTSELECT (\*((volatile unsigned long \*)0x7120000c))  #define VIC0INTENABLE (\*((volatile unsigned long \*)0x71200010))  #define VIC0INTENCLEAR (\*((volatile unsigned long \*)0x71200014))  #define VIC0PROTECTION (\*((volatile unsigned long \*)0x71200020))  #define VIC0SWPRIORITYMASK (\*((volatile unsigned long \*)0x71200024))  #define VIC0PRIORITYDAISY (\*((volatile unsigned long \*)0x71200028))  #define VIC0ADDRESS (\*((volatile unsigned long \*)0x71200f00))  #define PWMTIMER\_BASE (0x7F006000)  *// 定时器配置寄存器0，用于设置定时器的预分频系数。*  #define TCFG0 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x00)))  *// 定时器配置寄存器1，用于选择PWM输出的分频系数。*  #define TCFG1 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x04)))  *// 定时器控制寄存器，用于启动、停止和控制定时器操作。*  #define TCON (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x08)))  *// 定时器计数缓冲寄存器，用于设置定时器的计数初值。*  #define TCNTB0 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x0C)))  *// 定时器比较缓冲寄存器，用于设置定时器的比较值。*  #define TCMPB0 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x10)))  *// 定时器计数监视寄存器，用于读取定时器的当前计数值。*  #define TCNTO0 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x14)))  #define TCNTB1 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x18)))  #define TCMPB1 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x1C)))  #define TCNTO1 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x20)))  #define TCNTB2 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x24)))  #define TCMPB2 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x28)))  #define TCNTO2 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x2C)))  #define TCNTB3 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x30)))  #define TCMPB3 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x34)))  #define TCNTO3 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x38)))  #define TCNTB4 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x3C)))  #define TCNTO4 (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x40)))  *// 定时器4没有比较缓冲寄存器，只有计数缓冲寄存器和计数监视寄存器，无输出引脚。*  #define TINT\_CSTAT (\*((volatile unsigned long \*)(PWMTIMER\_BASE + 0x44)))  *// Timer 0 的中断使能位的功能和设置。*  *// 位于寄存器的第 0 位，可读可写（R/W）。*  *// 当设置为 1 时，中断被使能；当设置为 0 时，中断被禁用。*  typedef void(isr)(void);  extern void asm\_timer\_irq(); *// 声明汇编写的中断处理函数，该函数具体实现中断时需要执行的操作。*  void irq\_init(void)  {  */\* 在中断控制器里使能timer0中断 \*/*    VIC0INTENABLE |= (1 << 23); *// 使能定时器0中断 23位置为定时器0*    VIC0INTSELECT = 0; *// 设置中断类型为IRQ*    isr \*\*isr\_array = (isr \*\*)(0x7120015C); *// 获取中断服务程序数组地址*  *// 获取一个中断向量表的地址，这个地址是一个指针数组，每个指针指向一个中断服务程序。*    isr\_array[0] = (isr \*)asm\_timer\_irq; *// 将定时器0的中断服务程序设置为asm\_timer\_irq*  *// 设置中断服务程序，即当中断发生时，执行的函数。*  */\*将GPK4-GPK7配置为输出口\*/*    GPKCON0 = 0x11110000; *// 配置GPK4-GPK7为输出*  */\*熄灭四个LED灯\*/*    GPKDATA = 0xff; *// 1表示灭，0表示亮*  }  int t = 0;  *// timer0中断的中断处理函数*  void do\_irq()  {    unsigned long uTmp;    t++;    switch (t)    {    case 1:      GPKDATA = 0xef; *// 1110 1111*      break;    case 2:      GPKDATA = 0xdf; *// 1101 1111*      break;    case 3:      GPKDATA = 0xbf; *// 1011 1111*      break;    case 4:      GPKDATA = 0x7f; *// 0111 1111*      t = 0;      break;    }  *// 清timer0的中断状态寄存器*    uTmp = TINT\_CSTAT;    TINT\_CSTAT = uTmp;  *// 查找用户手册发现，要对*    VIC0ADDRESS = 0x0; *// 在中断处理结束时，*  *// 写入任意值（通常为0）到这个寄存器，用于通知中断控制器当前中断服务程序（ISR）已经处理完毕，可以接收新的中断请求。*  }  *// 初始化timer*  *// 用于初始化硬件定时器。它配置定时器的预分频系数、分频系数、计数值、比较值，并使能定时器中断。*  void timer\_init(unsigned long *utimer*, unsigned long *uprescaler*, unsigned long *udivider*, unsigned long *utcntb*, unsigned long *utcmpb*)  {    unsigned long temp0;  *// 定时器工作频率 = PCLK / ( {预分频值 + 1} ) / {分频值} = PCLK/(65+1)/16=62500hz*  *// PCLK的设置是由MPLLCON寄存器决定的 PCLK为66Mhz*  *// 设置预分频系数为66*  *// TCFG0是用于设置预分频系数的寄存器*    temp0 = TCFG0; *// 读取当前 TCFG0 的值。*  *// 仅仅更改定时器0和1的预分频系数*    temp0 = (temp0 & (~(0xff00ff))) | (*uprescaler* << 0); *// 0xff00ff = 1111 1111 0000 0000 1111 1111*    TCFG0 = temp0; *// 将修改后的值写回 TCFG0，设置预分频系数。*  *// 16分频*  *// TCFG1 是用于设置分频系数的寄存器*    temp0 = TCFG1; *// 读取当前 TCFG1 的值 由于在main函数中设置udivider=0100 因此表示16分频*    temp0 = (temp0 & (~(0xf << 4 \* *utimer*)) & (~(1 << 20))) | (*udivider* << 4 \* *utimer*);  *//         清除定时器 utimer 的分频系数，即将 TCFG1 的第 4\*utimer 位到第 4\*utimer+3 位清零。*    TCFG1 = temp0; *// 将修改后的值写回 TCFG1，设置分频系数*  *// 1s = 62500hz*  *// TCNTB0：设置定时器的计数初值。*  *// TCMPB0：设置定时器的比较值。*    TCNTB0 = *utcntb*; *// 将 utcntb 的值写入 TCNTB0，设置定时器的计数初值。*    TCMPB0 = *utcmpb*; *// 将 utcmpb 的值写入 TCMPB0，设置定时器的比较值。*  *// 如果设置为0时候，则表示和结束的时候相同也就是没有意义*  *// 手动更新*  *// TCON 是定时器控制寄存器，用于启动、停止和控制定时器操作。*    TCON |= 1 << 1; *// 设置 TCON 的第 1 位为 1，触发手动更新，将 TCNTB0 和 TCMPB0 的值加载到计数器和比较器中。*  *// 执行结束后立即送入实际的计数器和比较器中*  *// 清手动更新位*  *// 防止下一次不必要的手动更新*    TCON &= ~(1 << 1); *// 清除 TCON 的第 1 位，将手动更新位清零。*  *// 启动timer0和自动加载*    TCON |= (1 << 0) | (1 << 3);  */\**  *设置 TCON 的第 0 位为 1，启动定时器。*  *设置 TCON 的第 3 位为 1，使能自动重装载模式，定时器在达到比较值时会自动重载计数初值。*  *\*/*  *// 使能timer0中断*    temp0 = TINT\_CSTAT; *// 读取当前 TINT\_CSTAT 的值。*  *// 清除timer位置上的中断使能位后，再将其设置为1，使能timer的中断。*    temp0 = (temp0 & (~(1 << *utimer*))) | (1 << (*utimer*));    TINT\_CSTAT = temp0; *// 将修改后的值写回 TINT\_CSTAT，使能timer的中断。*  }  main.c文件如下:  #include 'stdio.h'  void timer\_init(unsigned long *utimer*, unsigned long *uprescaler*, unsigned long *udivider*, unsigned long *utcntb*, unsigned long *utcmpb*);  int main()  {    timer\_init(0, 65, 4, 125000, 0);    while (1)    {      }    return 0;  }  Makefile文件如下:     1. 编译及下载运行程序   编译代码  确保当前用户为root用户（可使用su root命令切换到root用户）的条件下，在Fedora/CentOS的终端中执行如下命令：  # cd timer  # make  执行make后会生成timer.bin文件。  下载（烧写）和运行程序  ①～③按实验一给出的方法操作  ④使用Minitools软件下载并运行程序的方法如下    首先在软件界面左侧选择“Linux”，接着勾选“Linux Bootloader”复选框，并选择需要下载的裸机程序(bin文件)，表示将当前逻辑程序以Linux系统启动引导程序的方式下载到NAND Flash的0地址开始的区域。最后点击下方的【Start flashing】按钮，执行所选则的裸机程序的烧写。  关闭开发板电源，将拨动开关S2拨动至“NAND”位置，然后重新开启开发板电源，则开发板运行刚刚烧入的裸机程序。初始状态，四个LED灯都为灭。每当定时器0的1秒定时达到后，四个LED切换状态   1. 修改要求   每间隔2秒，四个LED跑马灯方式循环显示(LED1亮→LED2亮→LED3亮→LED4亮→LED1亮→LED2亮→LED3亮→LED4亮→……)   1. 实现方式   设置一个标志用于表示当前时间，根据标记的不同结果来执行不同的操作，注意标记在每一次循环后要进行清除操作保证能够循环执行程序。 | | | |
| **三、实验验收结果及其分析（可附实验结果图片）**  **1.实验结果显示**            **2.实验结果分析**  如上图所示，开发板会以0.2秒的间隔进行状态的变换。编写程序中首先需要注意的是对于间隔时间的设置，由上课所讲的很容易计算出每间隔两秒表示PWM的输出频率为0.5因此只需要更改计数初值就可以完成定时器间隔两秒的操作，每当计数值到达0时便会触发中断，此时会改变灯的状态。实现要求主要是需要设置状态位也就是每当我中断一次将会对这个状态位进行++,根据这个状态来判断应该执行何种代码，同时需要注意的是由于本题目实现的要求是LED1->LED4循环亮而并不是跑马灯的形式，因此我在执行完最后的代码之后会将状态位置零此时将会重新执行代码。 | | | |

《嵌入式系统原理》课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | **实验六 熟悉Linux开发环境**  **实验六 PWM定时器实验** | | |
| 实验日期 | 2024.6.23 | 实验成绩 |  |
| **一、实验目的和要求**  **目的**:熟悉Linux 开发环境，学会基于Tiny6410 的Linux 开发环境的配置和使用。使用arm-Linux-gcc编译，并使用minicom实现串口方式下载和调试程序。  **要求**:将原程序中显示的“Hello, World!”修改为“Hello, HFUT!”。然后分别通过命令行输入自己的学号和当前日期，提示符分别是“Please enter your student ID:”和“Please enter the current date:”。若输入的学号的最后一位是偶数，则按格式“2019123456||2021-05-08”输出信息；若是奇数，则按格式“2019123457|2021-05-08”输出信息。  注意：所有输出的信息必须单独显示，不能与其它字符显示在同一行。 | | | |
| **二、实验内容（包括实验步骤和代码）**  本次实验使用Fedora（合肥校区）/CentOS（宣城校区）操作系统环境，安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程，即创建一个新目录hello，并在其中编写hello.c和Makefile文件。学习在Linux下的编程和编译过程，以及ARM开发板的使用和开发环境的设置。下载好已经编译好的文件到目标开发板上运行。   1. 登录Windows系统，使用管理员权限打开VMware虚拟机软件。确认虚拟机中已安装Fedora(合肥校区)/CentOS(宣城校区)系统，否则请通过镜像文件安装。 2. 确认虚拟机串口已打开（虚拟机右下角显示 图标且不为灰色），否则通过【编辑此虚拟机】选项，使用“添加“功能添加串行通信端口，并确认使用物理串行端口中为“自动选择串口“选项。 3. 通过【启动此虚拟机】，启动Fedora（合肥校区）/CentOS\_QT\_6410（宣城校区）系统。合肥校区请选择knight用户，并输入登录密码knight;宣城校区启动后自动以root用户登录。 4. 打开终端（terminal），若当前不是root用户，需要切换为root用户。建立hello工作目录。 5. 编写程序源代码   #include <stdio.h>  #include <string.h>  int main()  {    char student\_id[20];    char current\_date[20];    int last\_digit;    printf("Hello, HFUT!\n");    printf("Please enter your student ID: ");    scanf("%s", student\_id);    printf("Please enter the current date: ");    scanf("%s", current\_date);  *// 获取学号的最后一位数字*    last\_digit = student\_id[strlen(student\_id) - 1] - '0';  *// 判断最后一位数字是偶数还是奇数，并按要求输出信息*    if (last\_digit % 2 == 0)    {      printf("%s||%s\n", student\_id, current\_date);    }    else    {      printf("%s|%s\n", student\_id, current\_date);    }    return 0;  }   1. 编写Makefile文件。   要使上面的hello.c程序能够运行，必须要编写一个Makefile文件，Makefile文件定义了一系列的规则，它指明了哪些文件需要编译，哪些文件需要先编译，哪些文件需要重新编译等等更为复杂的命令。使用它带来的好处就是自动编译，只需要敲一个“make”命令整个工程就可以实现自动编译，当然本次实验只有一个文件，它还不能体现出使用Makefile的优越性，但当工程比较大文件比较多时，不使用Makefile几乎是不可能的。  下面介绍本次实验用到的Makefile文件。     1. 编译应用程序   在上面步骤完成后，就可以在hello目录下运行“make”命令来编译程序，如果进行了修改，则重新编译运行。   1. 下载调试   ①首先使用串口线连接开发板的串口（COM0）和PC机的RS232串口（或使用USB转线虚拟出的串口）;然后将Tiny6410开发板的启动模式拨动开关S2拨至“Nand”位置，表从NandFlash启动系统;再将电源拨动开关S1拨至“ON”，启动嵌入式Linux系统运行，此开发板LCD屏幕左上角显示企鹅图标，土系统初始化完毕显示主页面。（备注：若上电后屏幕黑屏， 且模式开关位置正确，则联系实验才导教师为开发板烧写系统）。  ②确认当前为root 用户后，在虚拟机的终端中输入minicom 命令，若提示Device/dev/ttyUSB0 access failed: No such file or directory错误信息，则命令行输入minicom–s，用键盘方向键选择Serial port setup后回车，然后按“A”键将ttyUSB0（合肥校区）/tty0（宣城校区）修改为ttySO（合肥校区）/ttyS1（宣城校区）（＊注意：字母“S”需要大写，且前面的/dev/路径不能删除），然后回车返回主界面，再选择Save setup as dfl 后回车，最后选择Exit退出。执行minicom命令成功打开串口后，出现下图：    回车后，出现下图：    ③ 按Ctrl+A，再按S，出现下图：    选择第一1zmodem，回车后，出现下图：    回车后，出现下图：    输入绝对路径或者通过键盘快捷键（双击空格键进入下一级H录，单击空格键选中文件）选择文件后，出现下图：    回车，出现下图：    从上图可以看出，文件未被下载，原因是Tiny6410板子已经有了hello可执行文件（之前下载的），若需要下载，则需要删除之前的hello文件，执行rm hello命令即可。可通过ls 命令查看是否删除成功。  删除之后，继续下载，出现下图，表示下载成功。    ⑤运行程序。  在电脑终端（当前路径是开发板嵌入式Linux的路径）中输入./hello，或者在Tiny6410 开发板LCD屏幕上输入hello都可。 | | | |
| **三、实验验收结果及其分析（可附实验结果图片）**  **1.实验结果显示**      **2.实验结果分析**  当输入学号以及当前时间信息之后，则会显示结果如上。 | | | |

**表1. 实验成绩评定表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 评价内容 | | 权重 | 得分 |
| **验收** | 实验原理是否理解；程序能否运行；实验结果是否正确；任务是否全部完成。 | | 0.4 |  |
| **实验报告** | 1 | 报告格式是否规范，语言使用是否规范，行文是否流畅，是否图文并茂； | 0.2 |  |
| 2 | 实验原理、实验步骤描述是否正确、详实；  程序流程图是否规范，代码实现是否正确；  实验数据记录是否完整，实验结果是否正确；  实验结果的分析、对比是否充分； | 0.3 |  |
| 3 | 实验体会是否正确，是否提出了自己独到见解。 | 0.1 |  |
| 合计 |  | | | |
| 指导教师（签章）： 2023 年 月 日 | | | | |