嵌入式考试范围详解

（本资料为嘉豪教育旗下会员独享资料）

主讲人：赵泽雨
2021 年 6 月 29 日星期二

目录

一、简答题：2 道，每题 10 分 2
1.makefile 2
2.S5PV210 的启动流程； 3
3.DNW 下载的软硬件流程； 4
二、程序填空题：3 道，一共 35 分 5
1.各种的初始化+ADC 5
GPIO 初始化 5
串口初始化 6
中断初始化 7
定时器初始化 8
定时器中断初始化 9
AD 编程（主要是触摸屏 AD 转换） 10
三、编程题：2 道，一共 45 分 12

（？加起来怎么只有 90 分）

一、简答题：2 道，每题 10 分

1.makefile

要讲出每行代码的意思

简单讲解：【视频链接】[教程]Makefile 的写法_哔哩哔哩_bilibili

arm-linux-ld -Ttext 0x00000000 crt0.o led_on_c.o -o led_on_c_tmp.o

这条指令的作用就是将 crt0.o 和 led_on_c.o 连接成 led_on_c_mp.o 可执行文件，此可执行文件的代码段起始地址为 0x00000000（即从这里开始执行）。

arm-linux-objcopy –o binary –S elf_file bin_file.elf

arm-linux-objcopy 被用来复制一个目标文件的内容到另一个文件中.此选项可以进行格式的转换.在实际编程的,用的最多的就是将 ELF 格式的可执行文件转换为二进制文件.

arm-linux-objdump –D –b binary –m arm bin_file > dis_file

arm-linux-objdump 常用来显示二进制文件信息,常用来查看反汇编代码

-b bfdname 指定目标码格式

—disassemble 或者-d 反汇编可执行段

—disassemble-all 或者-D 反汇编所有段

-EB,-EL 指定字节序

—file-headers 或者-f 显示文件的整体头部摘要信息

—section-headers,--headers 或者-h 显示目标文件中各个段的头部摘要信息

—info 或者-I 显示支持的目标文件格式和 CPU 架构

—section=name 或者-j name 显示指定 section 的信息

—architecture=machine 或者-m machine 指定反汇编目标文件时使用的架构

--Ttext 0x00000000 设置代码段的起始地址为 0x00000000；

-o 选项设置输出文件的名字

-c 只激活预处理,编译,和汇编,也就是他只把程序做成 obj 文件

-O0: 不做任何优化，这是默认的编译选项。

-O0 和-O1: 对程序做部分编译优化，对于大函数,优化编译占用稍微多的时间和相当大的内存。使用本项优化，编译器会尝试减小生成代码的尺寸，以及缩短执行时间，但并不执行需要占用大量编译时间的优化。

-O2: 是比 01 更高级的选项，进行更多的优化。Gcc 将执行几乎所有的不包含时间和空间折中的优化。当设置 02 选项时，编译器并不进行循环打开（）loop unrolling 以及函数内联。与 01 比较而言，02 优化增加了编译时间的基础上，提高了生成代码的执行效率。

-fno-builtin 即不使用 C 语言的内建函数

-fno-builtin-function （其中 function 为要冲突的函数名）某一函数不使用 C 语言的内建函数

-nostdlib 不连接系统标准启动文件和标准库文件，只把指定的文件传递给连接器。

$@ 表示目标文件

$^ 表示所有的依赖文件

$< 表示第一个依赖文件

$? 表示比目标还要新的依赖文件列表

2.S5PV210 的启动流程;

【视频链接】六、s5pv210 启动流程_哔哩哔哩_bilibili
结合书 P74-78 背流程可以背书上的

第一步：iROM 可以做最初的启动：初始化系统时钟，驱动特殊控制器和启动设备（MMC/OneNand/Nand/eSSD/NOR）。
第二步：iROM 可以引导代码可以把 boot-loader 载入到 SRAM，这个 boot-loader 被称为 BL1。之后 iROM 会在安全启动（引导）的模式下检查验证 BL1 的完整性。
第三步：BL1 将被执行：BL1 会载入剩下的叫做 BL2 的 boot-loader。之后 BL1 会在安全模式下检查验证 BL2 的完整性。
第四步：BL2 将被执行：BL2 在初始化 DRAM（内存）控制器后把 OS 数据载入到 SDRAM。
第五步：最终，跳转到操作系统开始的地址。会给使用系统创建良好的环境。

3.DNW 下载的软硬件流程;

参考实验课上 DNW 的使用方法

2. 安装USB驱动

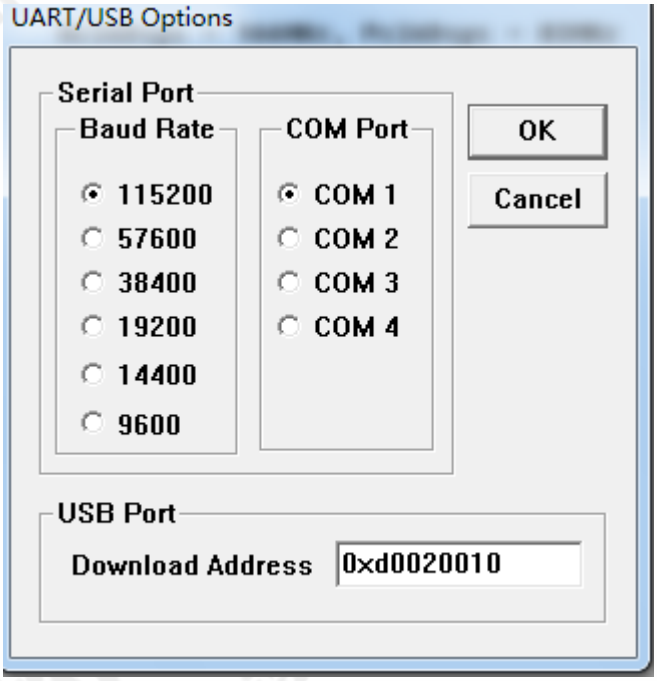
将实验箱中的拨码开关 2 拨到 on，长按 Power 键直至电脑提示安装驱动。打开计算机 设备管理器，选择下图所示硬件安装驱动。

右键选择更新驱动程序，手动添加 USB 驱动程序路径“D:\新 509\04-常用工具\DNW”。 在图 4 所示驱动安装过程中，将拨码开关 2 重新置为 OFF 状态，然后关闭开发板的电源，等待驱动安装完毕。

3. 下载文件、启动系统

1）用 USB device 线连接电脑和开发板，设置开发板为 nandflash 启动(拨码开关全部拨至 OFF 状态)。

2)在目录“D:\新 509\04-常用工具\DNW”中打开 DNW.exe。设置串口：波特率为 115200，USB Port 为 Download，Address 为 0xd0020010。



3）在菜单栏开启 DNW 串口连接（Serial Port ->Connect）。启动开发板后立即在 DNW 窗口迅速敲击空格键进入 Uboot 状态，可见图 11 启动界面

4）在 DNW 窗口中输入“dnw 0xd0020010”设置下载地址。如果 DNW 驱动安装失败 或首次使用 DNW，会提示安装驱动，请正确安装驱动，等到提示硬件可使用从进行下一步。
5）在 DNW 菜单中，选择 usbport->Transmit->Transmit 发送生成的 210.bin 文件，DNW 自动下载 210.bin 文件至开发板。
（用MV指令将linux下的210.bin，移动到/mnt/hgfs/forlinux，方便DNW传输，/mnt/hgfs/forlinux 文件夹就是window下的forlinux文件夹）
6）在 DNW 窗口中输入“go 0xd0020010”，即可开始运行 210.bin 程序。

二、程序填空题：3 道，一共 35 分

1.各种的初始化+ADC

GPIO 初始化、串口初始化、中断初始化、定时器初始化、定时器中断初始化、AD 编程。（系统时钟初始化不考<不要求会对 PCLK 时钟的配置>）。

GPIO 初始化

结合书 P84-86
P84 GPIO 寄存器总览
P85-86 GPC0 寄存器使用规则详解

```
#define GPC0CON          *((volatile unsigned int *)0xE0200060)
#define GPC0DAT      *((volatile unsigned int *)0xE0200064)

void delay(volatile unsigned long dly)
{
    volatile unsigned int t = 0xFFFF;
    while (dly--)
        for(; t > 0; t--);
}

int main(void)
{
    unsigned long i = (1 << 3);

    GPC0CON &= ~(0xF<<(3*4) | 0xF<<(4*4));    // 清 bit[15:12]和 bit[19:16]
    GPC0CON |= (1<<(3*4) | 1<<(4*4));         // 配置 GPC0_3 和 GPC0_4 为输出引脚

    while (1)
```

```
    {
        delay(0x50000);          // 延时
        GPC0DAT &= ~(0x3 << 3); // LED1 和 LED2 都熄灭
        if (i == 0x08)
            i = (1 << 4);
        else
            i = (1 << 3);
        GPC0DAT |= i;                  // 循环点亮 LED1 和 LED2
    }

    return 0;
}
```

串口初始化

结合书 P98 要会配置红色标记的部分!

```
#define GPA0CON             *((volatile unsigned int *)0xE0200000)
#define ULCON0          *((volatile unsigned int *)0xE2900000)
#define UCON0           *((volatile unsigned int *)0xE2900004)
#define UFCON0          *((volatile unsigned int *)0xE2900008)
#define UTRSTAT0        *((volatile unsigned int *)0xE2900010)
#define UTXH0           *((volatile unsigned int *)0xE2900020)
#define URXH0           *((volatile unsigned int *)0xE2900024)
#define UBRDIV0  *((volatile unsigned int *)0xE2900028)
#define UDIVSLOT0       *((volatile unsigned int *)0xE290002C)

// 初始化UART0(COM1)
void uart0_init(void)
{
    // 配置GPA0_0为UART_0_RXD，GPA0_1为UART_0_TXD
    GPA0CON &= ~0xFF;
    //GPA0CON |= 0x22; // bit[3:0]=0b0010,bit[7:4]=0b0010
    GPA0CON |=(2<<0) | (2<<4);
    /* 配置数据传输格式
     * data:8bits    bit[1:0]=0x3
     * stop:1bit     bit[2]=0
     * parity:no     bit[5:3]=0b0xx
     * mode:normal bit[6]=0
     */
     ULCON0 = (0x3<<0) | (0<<2) | (0<<3) | (0<<6);
```

```
    /* 配置UCON0
     * Receive Mode: bit[1:0]=1
     * TransmitMode: bit[3:2]=1
     * Rx Error Status Interrupt Enable: bit[6]=1
     * Clock Selection: bit[10]=0
     */
    UCON0 = (1<<0) | (1<<2) | (1<<6) | (0<<10);

    // 不使用FIFO
    UFCON0 = 0;

    /* 计算波特率(参考手册上面的公式)，设置为115200
     * PCLK=66MHz
     * DIV_VAL = (66000000/(115200 x 16))-1 = 35.8 - 1 = 34.8
     * UBRDIV0 = 34(DIV_VAL的整数部分)
     * (num of 1's in UDIVSLOTn)/16 = 0.8
     * (num of 1's in UDIVSLOTn) = 12（向下取整）
     * UDIVSLOT0 = 0xDDDD (参考手册查表)
     */
     UBRDIV0 = 34;
     UDIVSLOT0 = 0XDDDD;

     return;
}
```

中断初始化

结合书 P105-106 中断控制寄存器介绍
书 P108-109 中断初始化

```
// 配置中断引脚
void init_key(void)
{
    // 配置 GPIO 引脚为中断功能
    GPH0CON &= ~(0xFF<<0);
    GPH0CON |= (0xFF<<0); // key1:bit[3:0];key2:bit[7:4]
    // 配置 EXT_INT[0]、EXT_INT[1]中断为下降沿触发
    EXT_INT_0_CON &= ~(0xFF<<0);
    EXT_INT_0_CON |= 2|(2<<4);
    // 取消屏蔽外部中断 EXT_INT[0]、EXT_INT[1]
    EXT_INT_0_MASK &= ~0x3;
}
// 清中断挂起寄存器
```

```
void clear_int_pend()
{
    EXT_INT_0_PEND |= 0x3; // EXT_INT[0]、EXT_INT[1]
}
// 初始化中断控制器
void init_int(void)
{
    // 选择中断类型为 IRQ
    VIC0INTSELECT &= ~0x3; // 外部中断 EXT_INT[0]、EXT_INT[1]为 IRQ
    // 清 VIC0ADDRESS
    VIC0ADDRESS = 0x0;
    // 设置 EXT_INT[0]、EXT_INT[1]对应的中断服务程序的入口地址
    VIC0VECTADDR0 = (int)IRQ_handle;
    VIC0VECTADDR1 = (int)IRQ_handle;
    // 使能外部中断 EXT_INT[0]、EXT_INT[1]
    VIC0INTENABLE |= 0x3;
}
```

定时器初始化

结合书 P120-122 寄存器介绍
书 P128

```
/*
 * File：timer.c
 * 功能：初始化 Timer0
 */
//#define    GPD0CON      (*(volatile unsigned int *)0xE02000A0) // PWM 输出引脚
#define  TCFG0        (*(volatile unsigned int *)0xE2500000)
#define  TCFG1        (*(volatile unsigned int *)0xE2500004)
#define  TCON         (*(volatile unsigned int *)0xE2500008)
#define  TCNTB0       (*(volatile unsigned int *)0xE250000C)
#define  TCMPB0       (*(volatile unsigned int *)0xE2500010)
//#define    TCNTO0       (*(volatile unsigned int *)0xE2500014)
/*
 * Timer input clock Frequency = PCLK / {prescaler value+1} / {divider value}
 * {prescaler value} = 1~255
 * {divider value} = 1, 2, 4, 8, 16, TCLK
 * 本实验的 Timer0 的时钟频率=66.7MHz/(65+1)/(16)=63162Hz( 即 1s 计数 63162 次 )
 * 设置 Timer0 1 秒钟触发一次中断:
 */
void timer0_init(void)
{
    TCNTB0 = 63162;      /* 1 秒钟触发一次中断 */
```

```
    TCMPB0 = 31581;             /* PWM 占空比=50% */
  TCFG0 |= 65;                  /* timer 0 Prescaler value = 65 */
  TCFG1   = 0x04;          /* 选择 16 分频 */
  TCON   |= (1<<1);     /* 手动更新 */
  TCON    = 0x09;          /* 自动加载，清“手动更新”位，启动定时器 0 */
}
```

定时器中断初始化

结合书 P128-129

```
/*
功能：使能 TIMER0 中断、初始化中断向量控制器
 */
#define TINT_CSTAT          *((volatile unsigned int *)0xE2500044)

#define   VIC0IRQSTATUS                 *((volatile unsigned int *)0xF2000000)
#define VIC0INTSELECT          *((volatile unsigned int *)0xF200000C)
#define VIC0INTENABLE          *((volatile unsigned int *)0xF2000010)
#define VIC0VECTADDR21           *((volatile unsigned int *)0xF2000154)
#define VIC0ADDRESS            *((volatile unsigned int *)0xF2000F00)

extern void IRQ_handle(void);

// 使能 TIMER0 中断
void init_irq(void)
{
    TINT_CSTAT |= 1;
}
// 清中断
void clear_irq(void)
{
    TINT_CSTAT |= (1<<5);
}
// 初始化中断控制器
void init_int(void)
{
    // 选择中断类型为 IRQ
    VIC0INTSELECT |= ~(1<<21); // TIMER0 中断为 IRQ
    // 清 VIC0ADDRESS
    VIC0ADDRESS = 0x0;
    // 设置 TIMER0 中断对应的中断服务程序的入口地址
```

```
    VIC0VECTADDR21 = (int)IRQ_handle;
    // 使能 TIMER0 中断
    VIC0INTENABLE |= (1<<21);
}
// 清除需要处理的中断的中断处理函数的地址
void clear_vectaddr(void)
{
    VIC0ADDRESS = 0x0;
}
// 读中断状态
unsigned long get_irqstatus(void)
{
    return VIC0IRQSTATUS;
}
```

AD 编程（主要是触摸屏 AD 转换）

- **TSADCCON0 寄存器的位定义如下图所示：**

TSADCCONn	Bit	Description	Initial State
TSSEL	[17]	Touch screen selection 0 = Touch screen 0 (AIN2~AIN5) 1 = Touch screen 1 (AIN6~AIN9) This bit exists only in TSADCCON0. Note: An access to TSADCCON1 bits is prohibited when TSSEL bit is 0, and an access to TSADCCON0 bits except TSSEL is prohibited when TSSEL bit is 1. An access to TSSEL bit is always permitted.	0
RES	[16]	ADC output resolution selection 0 = 10bit A/D conversion 1 = 12bit A/D conversion	0
ECFLG	[15]	End of conversion flag(Read only) 0 = A/D conversion in process 1 = End of A/D conversion	0
PRSCEN	[14]	A/D converter prescaler enable 0 = Disable 1 = Enable	0
PRSCVL	[13:6]	A/D converter prescaler value Data value: 5 ~ 255 The division factor is (N+1) when the prescaler value is N. For example, ADC frequency is 3.3MHz if PCLK is 66MHz and the prescaler value is 19. Note: This A/D converter is designed to operate at maximum 5MHz clock, so the prescaler value should be set such that the resulting clock does not exceed 5MHz.	0xFF

触摸屏接口选择

AD转换精度选择

AD转换完成标志位

分频器使能控制位

分频系数设置

Reserved	[5:3]	Reserved	0
STANDBY	[2]	Standby mode select 0 = Normal operation mode 1 = Standby mode Note: In standby mode, prescaler should be disabled to reduce more leakage power consumption.	1
READ_START	[1]	A/D conversion start by read 0 = Disables start by read operation 1 = Enables start by read operation	0
ENABLE_START	[0]	A/D conversion starts by enable. If READ_START is enabled, this value is not valid. 0 = No operation 1 = A/D conversion starts and this bit is automatically cleared after the start-up.	0

模式控制

使能start by read控制位

开始一次AD转换

```
void adc_init()
{
 TSADCCON0 |=1 <<17;  //选择触摸屏 1
 TSADCCON1=1<<14 | 19<<6 | 0<<2;  //分频器使能、3.3Mhz、正常模式。
 TSDLY1 =0XFFFF;    //最大延时
 TSCON1=0xd3;            // 11010011,
uart_send_string("\rinitadc finsh\n");
}
void irq_int()
{      VIC2INTENCLEAR |=0x3<<9;
VIC3INTENCLEAR |=0x3<<9;      //清中断。
VIC2INTSELECT |=~(0x3<<9);       //中断选择寄存器，我们在这里选择 FIQ 模式还是 IRQ 模式.
VIC3INTSELECT |=~(0x3<<9);
VIC2ADDRESS =0;  //清中断向量寄存器（基地址）
VIC3ADDRESS =0;
 VIC2VECTADDR0 = (int)key_isr;
 VIC3VECTADDR1 = (int)key_isr;
VIC3INTENABLE |= 0x3<<9;
}
void key_handle( )
{
uart_send_string("test key_handle");
VIC2ADDRESS =0;  //清中断向量寄存器（基地址）
VIC3ADDRESS =0;
TSCON1=0x5c;      //x、y 设置为自动转换
TSADCCON1 |=1;                    //启动 ADC
while(TSADCCON1 & 0x1);          //
while(!TSADCCON1 & (1<<15)); //等待 ADC 转换完成

uart_send_string("\rX:");
```

```
uart_send_num(TSDATX1 & & 0x3ff);
uart_send_string("Y:");
uart_send_num(TSDATY1 & 0x3ff);
uart_send_byte("\n");
TSCON1 = 0xd3;
}
```

1）.VIC0ADDR ： 一共有四个寄存器，VIC(0-3)一人一个，用来存放我们想要的执行的中断处理程序（isr）的地址，它里面的地址是从 VIC0VECTADDRn 这个寄存器里面来的，当中断发生之后，VIC0VECTADDRn 里面的地址就会硬件自动刷到这个寄存器上。（中断处理完成之后，我们要清除这个寄存器）

2）.VIC0INTENCLEAR ： 清中断寄存器，也就是中断处理完成之后，我们要往这个寄存器里面写 1，把中断清理掉。我们也可以通过相应的中断号，来禁止那个中断。

3）.VIC0INTSELECT ： 中断选择寄存器，我们在这里选择 FIQ 模式还是 IRQ 模式

4）.VIC0VECTADDRn ： 一共有 128 个这样的寄存器，一个中断号对应一个这样的寄存器，我们可以通过中断号和 VICnADDR 这个基地址来计算 VIC0VECTADDRn 这个寄存器的地址，然后把我们的执行程序放到这个寄存器中，这样我们就不需要定义太多的宏了。

5）.VIC0IRQSTATUS ： IRQ 模式下的中断状态寄存器（一共有 4 个），中断发生后，这个寄存器就会自动置 1 了，然后我们是通过判断这 4 个寄存器中哪个寄存器写了 1，然后得知我们的中断执行程序的地址是放到了哪个 VICnADDR 寄存器上，最后在相应的 VICADDR 寄存器上找到中断执行程序的地址。

三、编程题：2 道，一共 45 分

类似：
定时器+PWM+中断服务程序
串口+定时器+中断服务程序

流程：
①初始化 GPIO 端口，使 GPIO 为外部中断状态；（寄存器：GPxxCON）
②配置外部中断触发模式，上升沿触发，下降沿触发，双边触发，高电平触发，低电平触发（寄存器：EXT_INT_x_CON）
③取消屏蔽外部中断（寄存器：EXT_INT_x_MASK）
④通过向量中断控制器（VIC）使能中断（寄存器：VIC0INTENABLE，VIC1INTENABLE，VIC2INTENABLE，VIC3INTENABLE）（注：几乎每一个中断都有其对应的 VIC）
注：VICINTENABLE 寄存器分别包括 VIC0，IVC1，VIC2，VIC3 四个，每一个中断控制器有 32 位，每一位对应一个中断源（假如你要使能外部中断 1，那么首先要先找到外部中断 1 对应哪一个中断控制器，然后找到这个中断控制器的相应位为设置）
⑤设置中断程序入口（寄存器：VICxVECTADDRx）
⑥开总中断
⑦编写中断处理程序（清除中断寄存器：EXT_INT_x_PEND）