深圳市九鼎创展技术有限公司技术文档

文档名称：x210v3 开发板裸机教程

深圳市九鼎创展科技有限公司

地址：深圳市宝安区西乡街道宝源路宝安互联网产业基地 A 区 7 栋 301 室

网址：http://www.9tripod.com

论坛：http://bbs.9tripod.com/

版权声明

敬告：

在售开发板的手册会经常更新，请在 http://www.9tripod.com 网站下载最新手册，不再另行通知。

版本说明

版本号	日期	作者	描述
Rev.01	2013-9-2		原始版本

技术支持

如果您对文档有所疑问，您可以在办公时间（星期一至星期五上午 8:30~12:00；下午 1:30~6:00；星期六上午 9:00~12:00）拨打技术支持电话或 E-mail 联系。

网　　址：　www.9tripod.com
联系电话：　0755-31333436
E-mail：　phosphor88@163.com

销售与服务网络

公司：深圳市九鼎创展科技有限公司
地址：深圳市宝安区西乡街道宝源路宝安互联网产业基地 A 区 7 栋 301 室
邮编：518101
电话：0755-33133436
网址：http://www.9tripod.com
论坛：http://bbs.9tripod.com
　　http://www.xboot.org
淘宝：http://armeasy.taobao.com

QQ 群：
x6410 技术论坛：　【16073601】
x210 技术论坛 1：　【23831259】
x210 技术论坛 2：　【211127570】
x210 技术论坛 3：　【211128231】
i210 技术论坛 1：　【159144256】
i210 技术论坛 2：　【189920370】
i210 技术论坛 3：　【199358213】

目录

目录

第1章 序言

x210 开发板自上市以来，广受各界企业，工程师好评，现批量合作的企业已有几百家。但是随着三星 S3C2440 的停产，S3C6410 性能上的不足，使得 S5PV210 成为自 2012 年以来学习 linux，android 以及 WINCE 的不二之选，尤其是 android 操作系统，搭载 512M DDR2，以及强悍的 emmc 存储，相比同类型 CPU，具有得天独厚的优势。但是同时，这也给不少入行不久的工程师，或是学生一个很大的难题，就是入门很难。俗话说得好，师傅领进门，修行靠自己。但是谁来领我们入门呢？

正因为如此，九鼎创展的工程师们为大家精心编写了这套教程，让我们从裸机开始，一步步揭开嵌入式神秘的面纱！

第2章 裸机开发环境的搭建

开发裸机有很多种方法，之前在 S3C2410，S3C2440 平台上，比较常用的是 ADS1.2 或是 MDK。但是这些工具主要针对 ARM9 平台，对于后续的 cortex-A8,cortex-A9 平台，他们是心有余而力不足。也有一些朋友喜欢直接在 linux 下进行裸机开发，但是对于开发 WINCE 的朋友，可能就遇到困难了。在 linux 下开发，需要安装 linux 操作系统，需要熟悉 makefile，交叉编译工具链等。正因为如此，我们自主搭建了强大的 eclipse 开发平台，制作了四套 eclipse 开发环境，同时支持 linux32 位，linux64 位，windows32 位，windows64 位操作系统。有了这四套 eclipse 开发环境，无论您使用 ubuntu32 位，ubuntu64 位，或是 fedora32 位，fedora64 位，或是 winxp，win7 等等，都可以开发裸机。同时，烧写程序也不再局限于 linux 系统，无论您使用何种操作系统，都能方便的将映像文件写到 SD 卡。下面就开始我们裸机开发的神奇之旅吧！

2.1 安装 eclipse

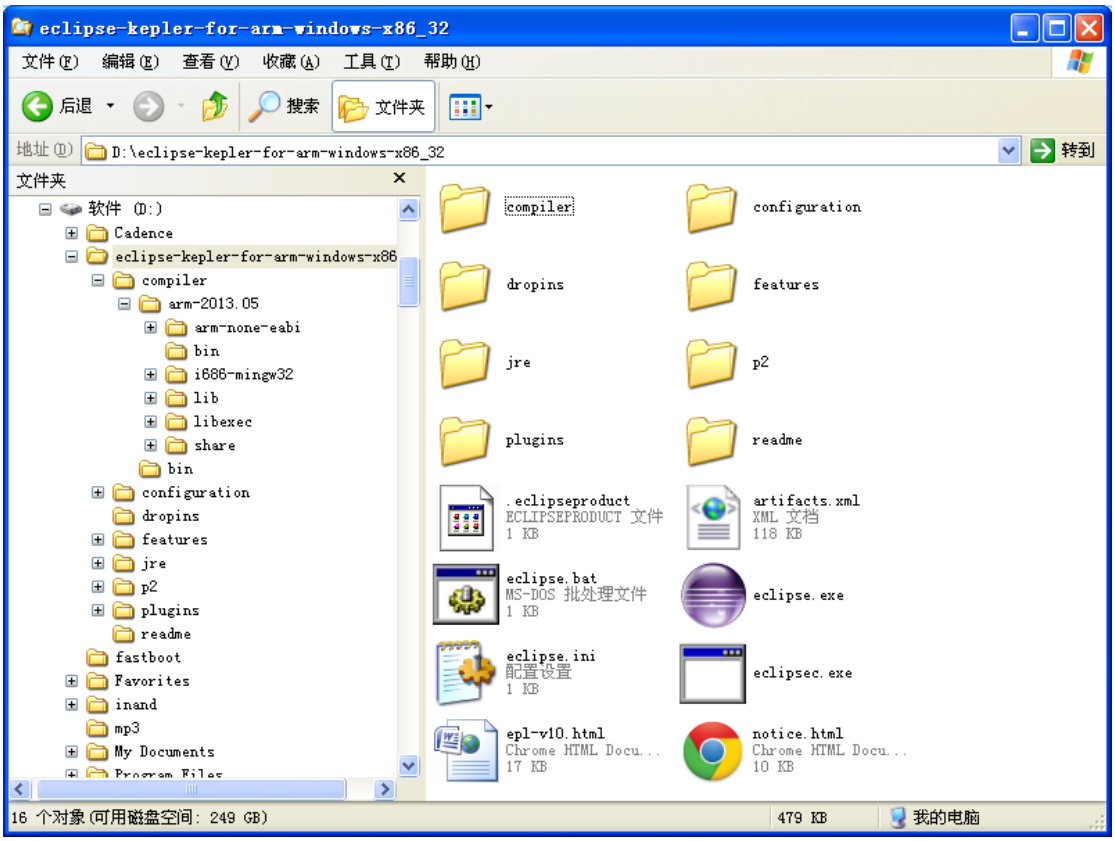
第一步：确认自己的 PC 机开发环境。开发板光盘中有如下四个 eclipse 包：

```
eclipse-kepler-for-arm-windows-x86_32.7z
eclipse-kepler-for-arm-windows-x86_64.7z
eclipse-kepler-for-arm-gtk-linux-x86_64.7z
eclipse-kepler-for-arm-gtk-linux-x86_32.7z
```

从命名方式，我们能够很清楚的差别，我们需要的开发包。这些包全部是绿色软件，无需安装，解压即可。由于笔者采用 winxp 32 位操作系统，因此选用第一个包，读者根据自己的 PC 机环境选择对应安装包，后续不再重复说明。

第二步：解压 eclipse 安装包。

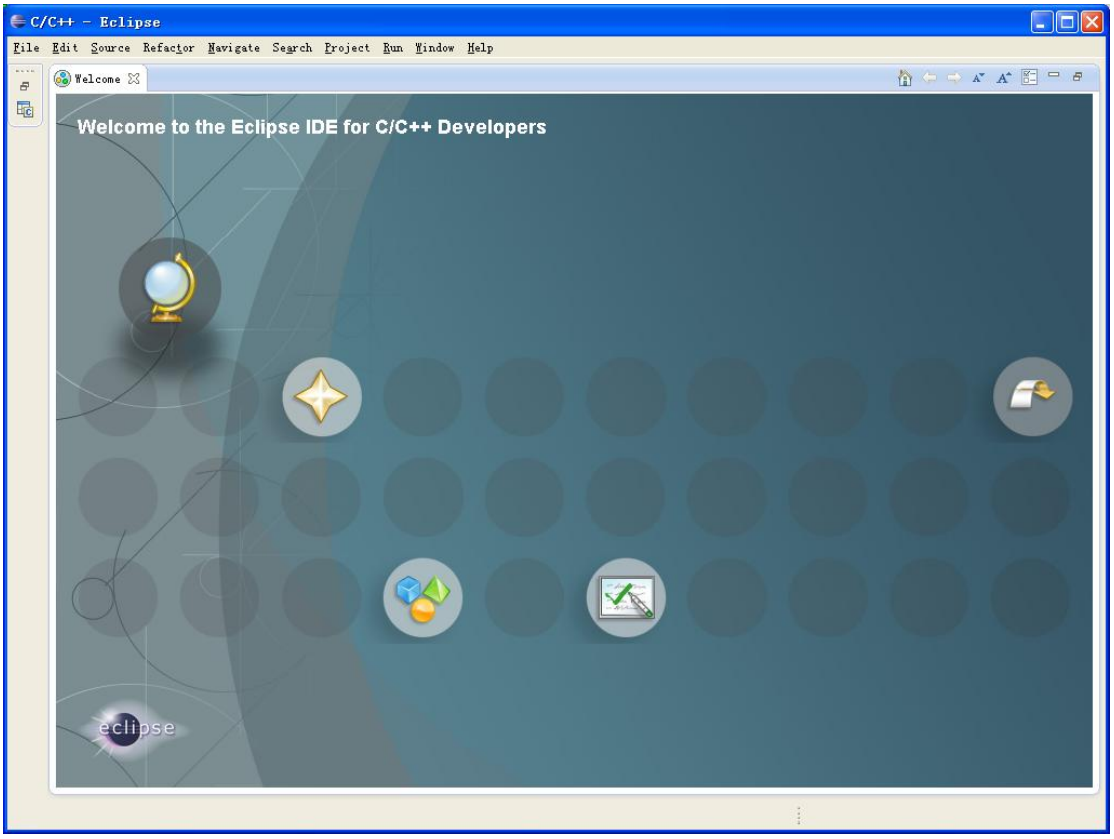
将 eclipse-kepler-for-arm-windows-x86_32.7z 拷贝到安装目录，如 D 盘并解压，如下图：



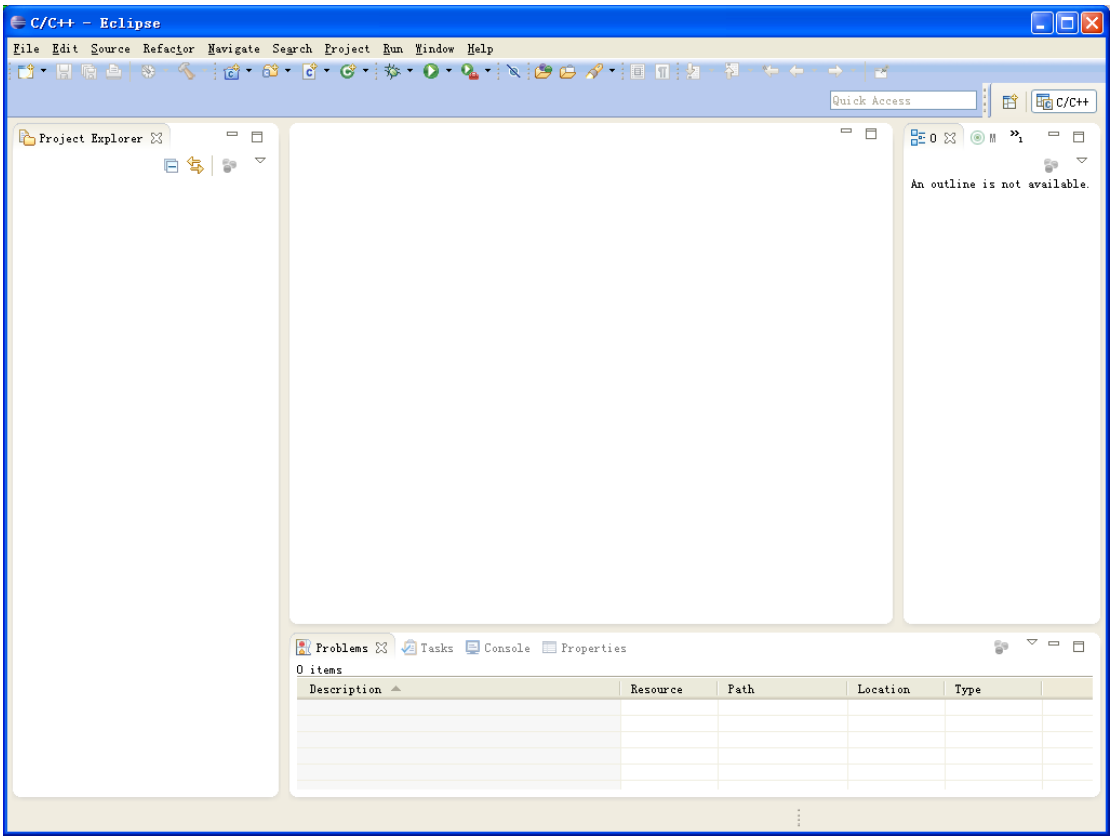
第三步：启动 eclipse

在解压的安装目录下，有 eclipse.bat 和 eclipse.exe 两个文件，双击 eclipse.bat 即可启动。由于使用 eclipse 需要一些环境变量的声明，我们专程做了个脚本 eclipse.bat，双击即可声明环境变量，同时启动 eclipse。这样，就不用再去手动设置环境变量了。当然用户也可以手动设置好环境变量，然后通过双击 eclipse.exe 来运行 eclipse，这里就不赘述了。

启动后界面如下：



这里是一幅欢迎界面，直接关掉，即进入我们需要的开发界面：

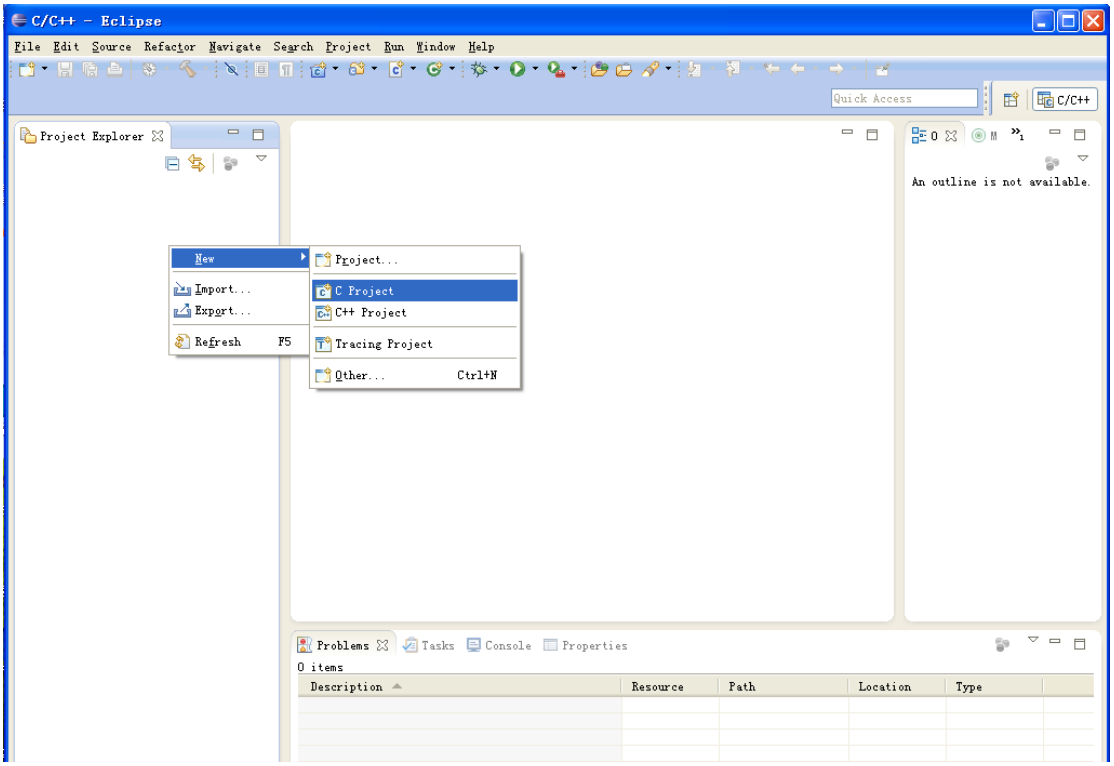


注意，第一次启动 eclipse 时，会提示我们选择工作空间路径，这里我们设置为 D:\workspace，当然用户也可以设置到自己喜爱的路径。

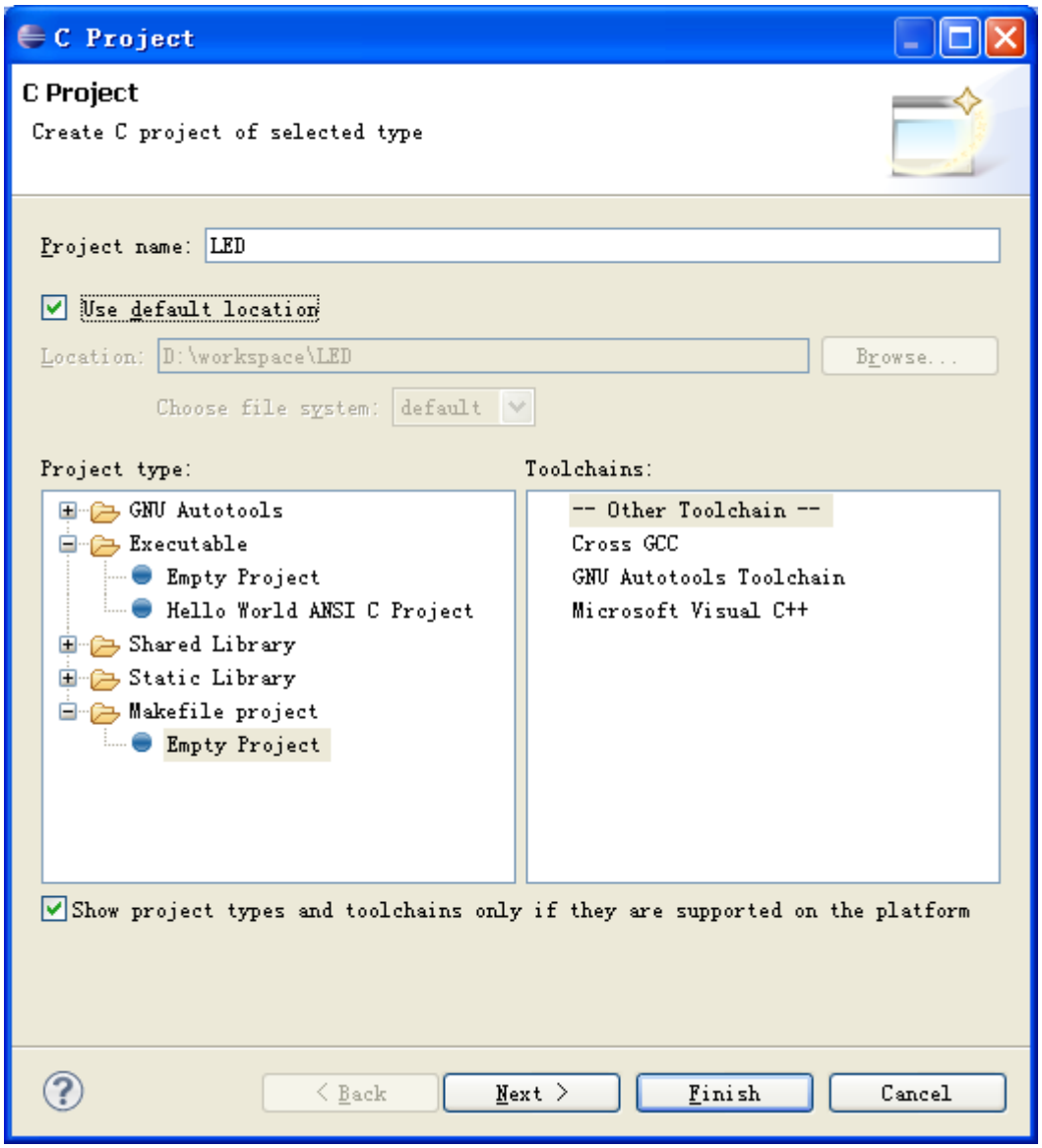
2.2 建立第一个工程

第一步：双击 eclipse.bat，打开 eclipse

第二步：在 Project Explorer 的空白栏右键单击->New->C Project，

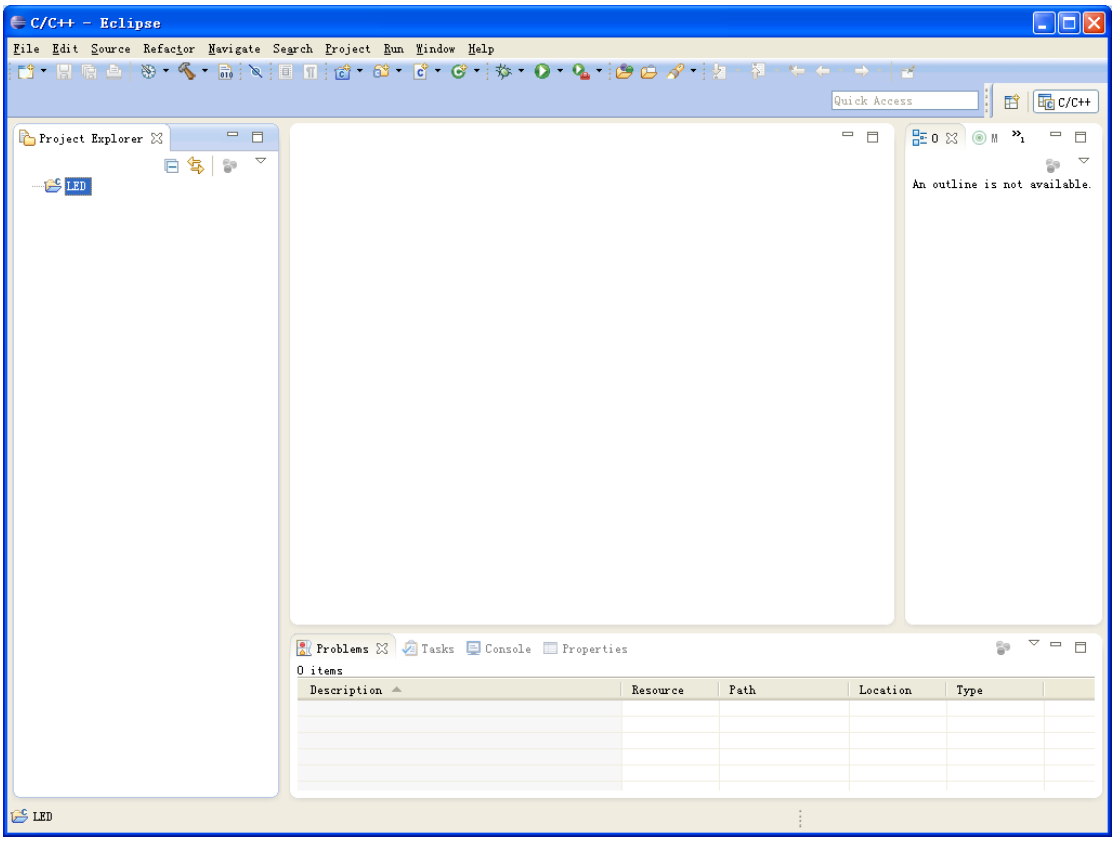


弹出新建项目的对话框：

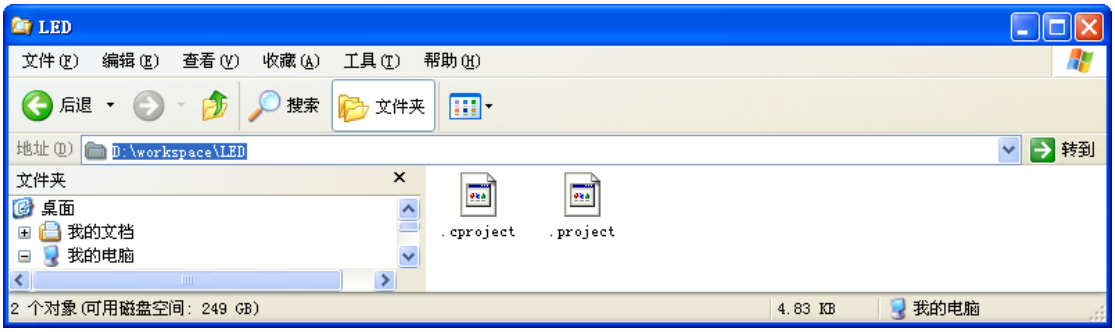


在 Project name 中输入工程名称，这里我们命名为 LED，在 Use default location 前有一个勾，默认已经勾选上，下面的 Location 一栏就是我们的工程目录。由于前面我们已经将默认工作空间设置为 D:\ workspace，因此我们建立的工程路径为：D:\workspace\LED。在 Project type 一栏中选择 Makefile project 下面的 Empty Project，Toolchains 不用管他，点击 Finish。

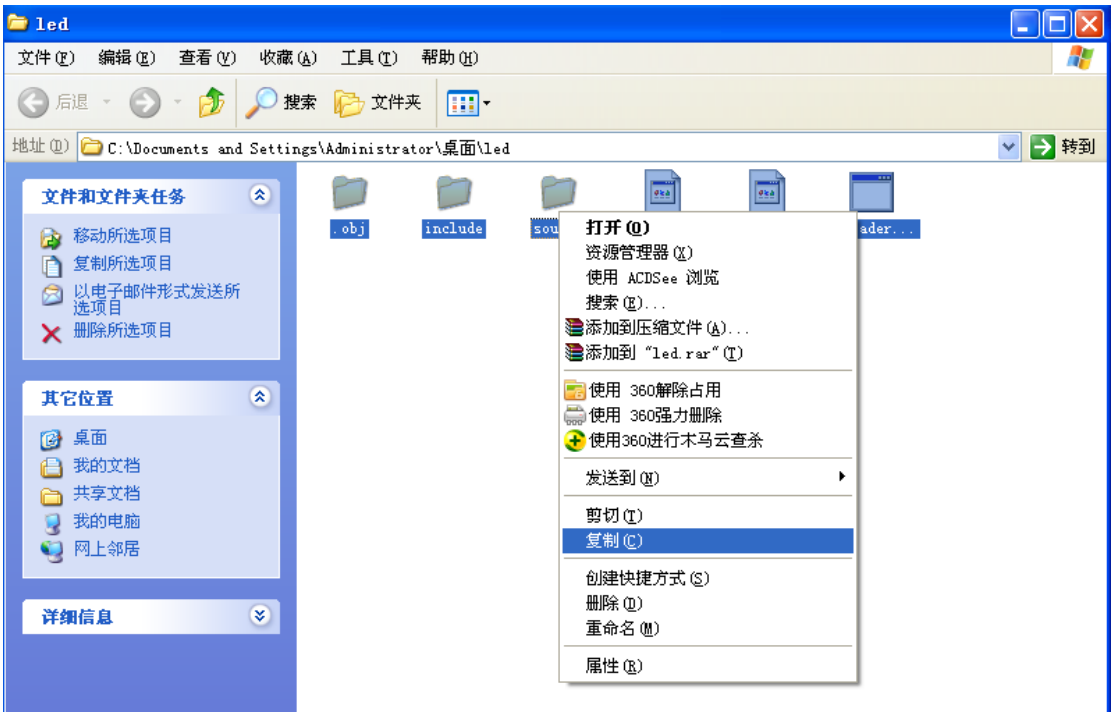
新建的工程如下：



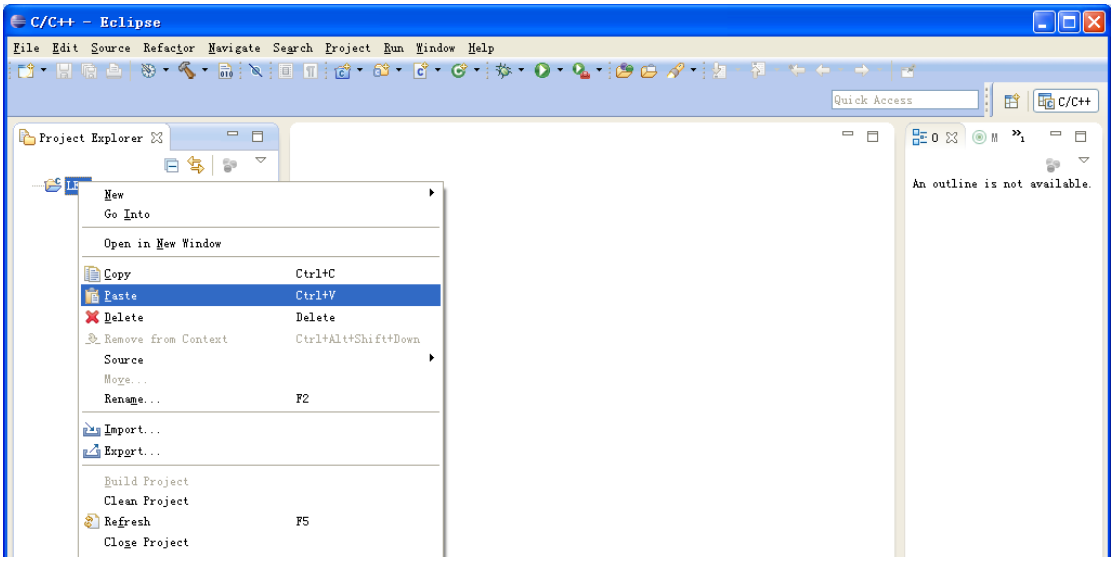
我们可以进入 D:\workspace\LED 目录，这时里面只有两个文件：



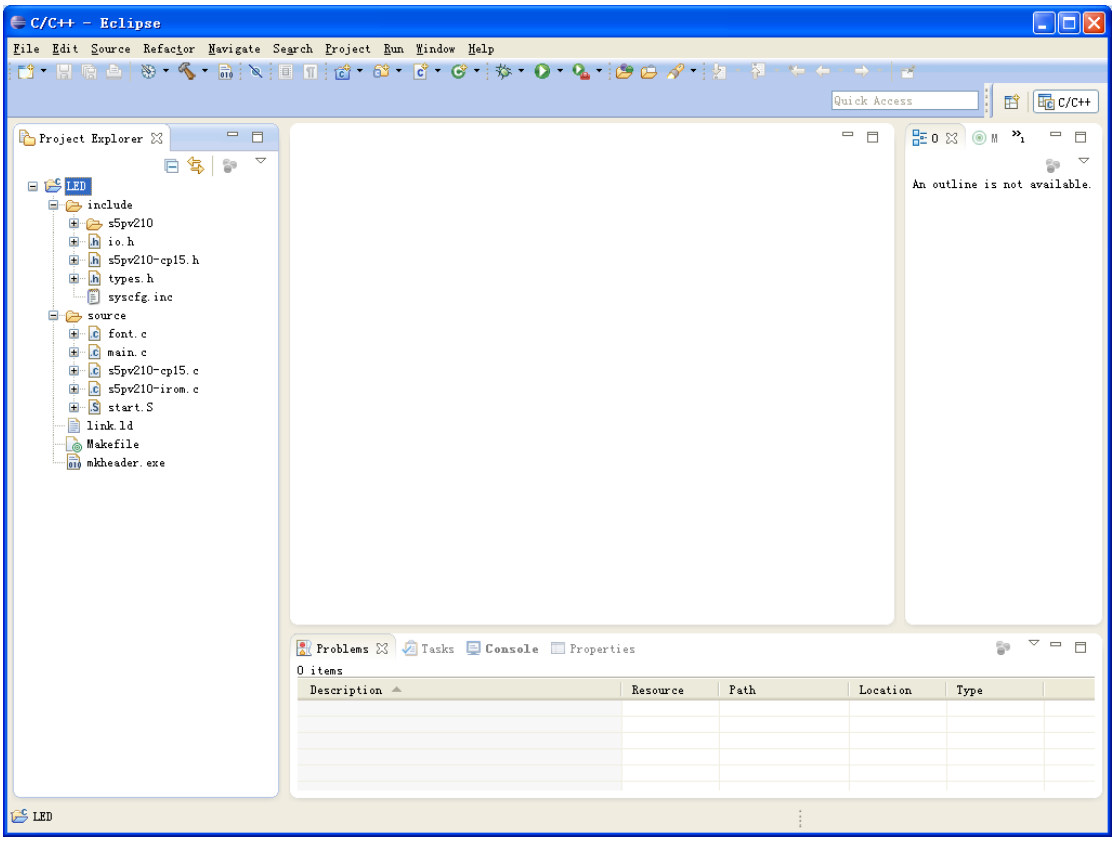
这是 eclipse 自动生成。找到我们提供的第一个裸机工程文件 led.rar 并解压，复制里面所有内容：



在 eclipse 的 Project Explorer 一栏，右键点击工程 LED，选择 Paste，即将 LED 工程拷贝过来了。



拷贝后工程目录如下：

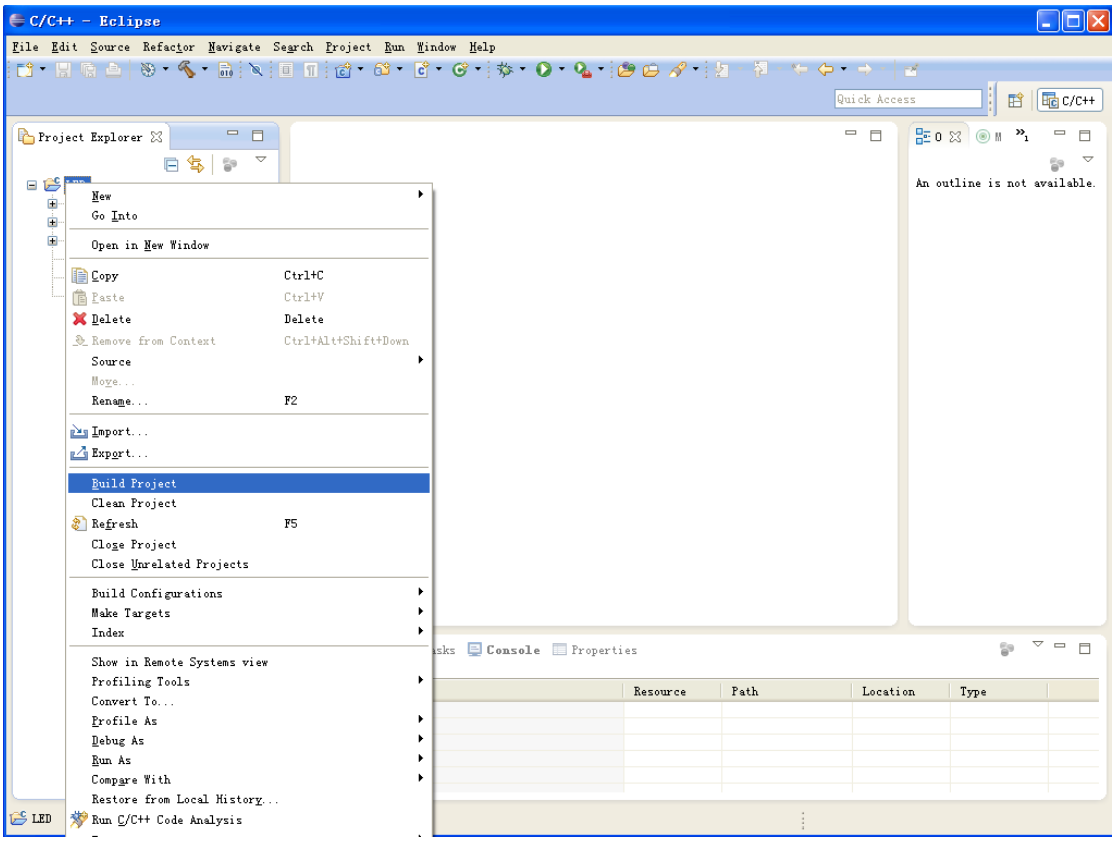


这里，include 目录是我们已经建好的基于 S5PV210 的一些头文件，用户可直接调用。soure 目录为源代码文件，程序首先会从 start.S 文件开始执行，这里面初始化了一系列 S5PV210 的寄存器，如看门狗等。之后，会通过 bl main 语句跳到 main.c 中执行 main 函数。

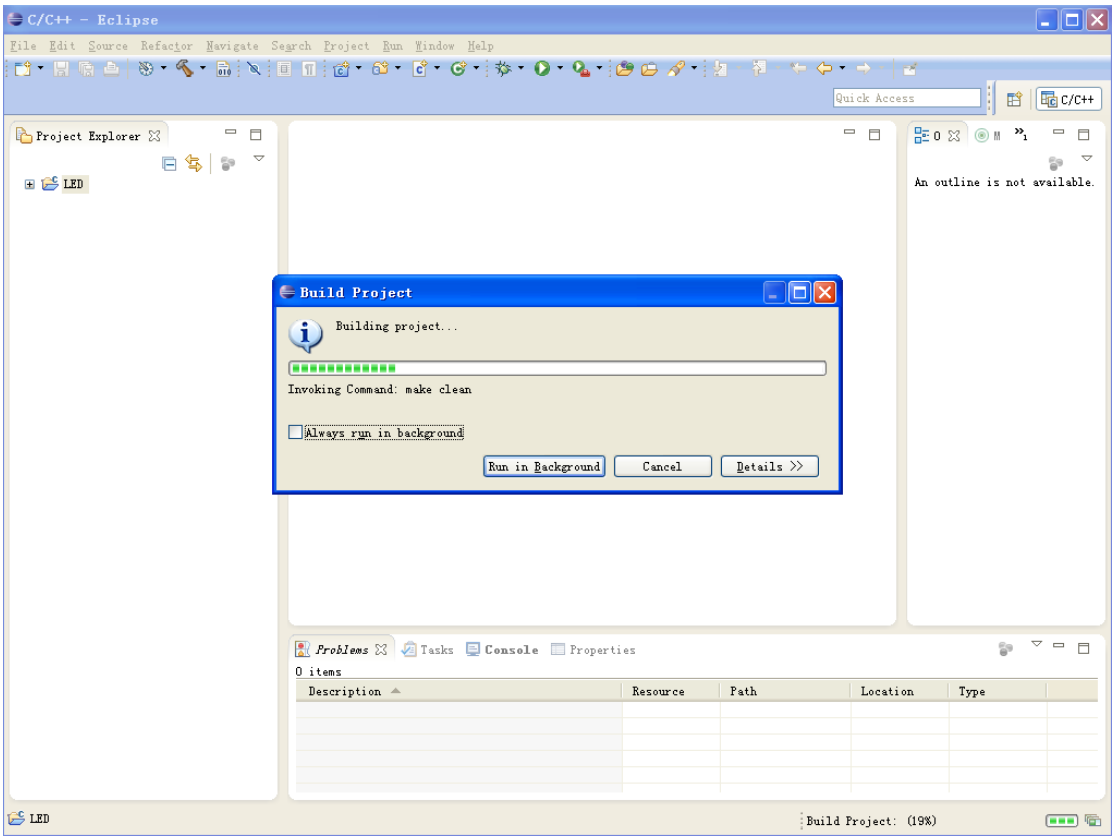
工程目录中 link.ld 为链接文件，Makefile 为编译源码需要用到的脚本文件，mkheader.exe 用于计算校验和，没有这个文件，我们后面生成的映像，下载到 SD 卡后就没法运行了。

2.3 编译源码

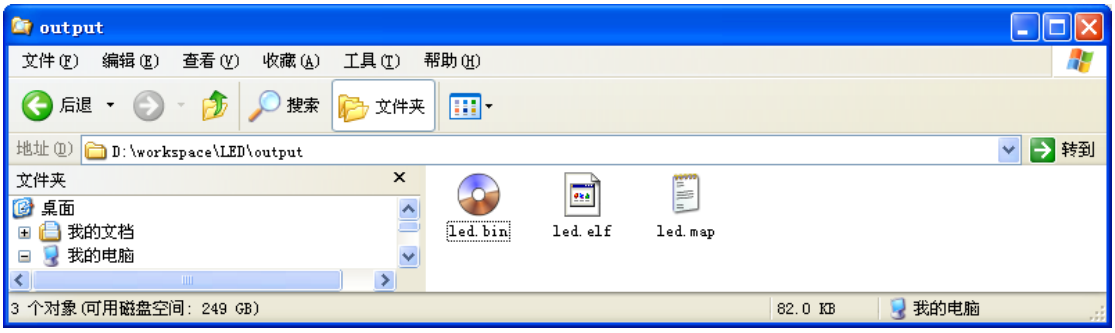
右击 Project Explorer 栏的工程名称 LED，选择 Build Project：



也可以使用快捷键 CTRL+B 编译。编译时界面如下：



编译完成后，在工程目录下会生成 output 目录：



图中 led.bin 即是我们最终需要的映像文件。

2.4 下载源码到 SD 卡

针对 x210 开发板，裸机开发我们几乎都是将裸机程序直接烧写到 SD 卡运行。在 WindowsXP 或是 WIN7 下，可以使用我们自主编写的烧写工具 x210_Fusing_Tool.exe 烧写。

双击 x210_Fusing_Tool.exe，将不小于 2GB 的 SD 卡通过读卡器插到 PC 机(注意市面上很多山寨的读卡器，推荐大家用正品行货，如飚王的读卡器，使用山寨读卡器不保证能正常将映像烧写到 SD 卡)，这时烧写软件会自动识别出 SD 卡，如下图：

点击 Browse，找到我们刚生成的 led.bin 文件，点击 Add，映像会添加到下面的路径中，这样下次如果不更改路径，就不用再指定路径了。

点击 START，开始烧写。烧写完成后，会有如下提示：

点击确定即可。

2.5 清除开发板中的 bootloader

由于 S5PV210 芯片无法直接从 SD2 通道启动，首先会从 SD0 通道启动，而 SD0 通道接了 emmc 芯片，因此我们务必将 emmc 中已存在的 bootloader 破坏掉！

2.5.1 破坏开发板 linux 平台下的 bootloader，从 SD2 启动开发板

进入 linux 系统控制台，执行如下指令：

```
busybox dd if=/dev/zero of=/dev/mmcblk0 bs=512 seek=1 count=1 conv=sync
```

为了确保执行数据有效，再执行 sync 同步一下，即可破坏掉前一个块的数据。

2.5.2 破坏开发板 android 平台下的 bootloader，从 SD2 启动开发板

进入 android 系统控制台，执行如下指令：

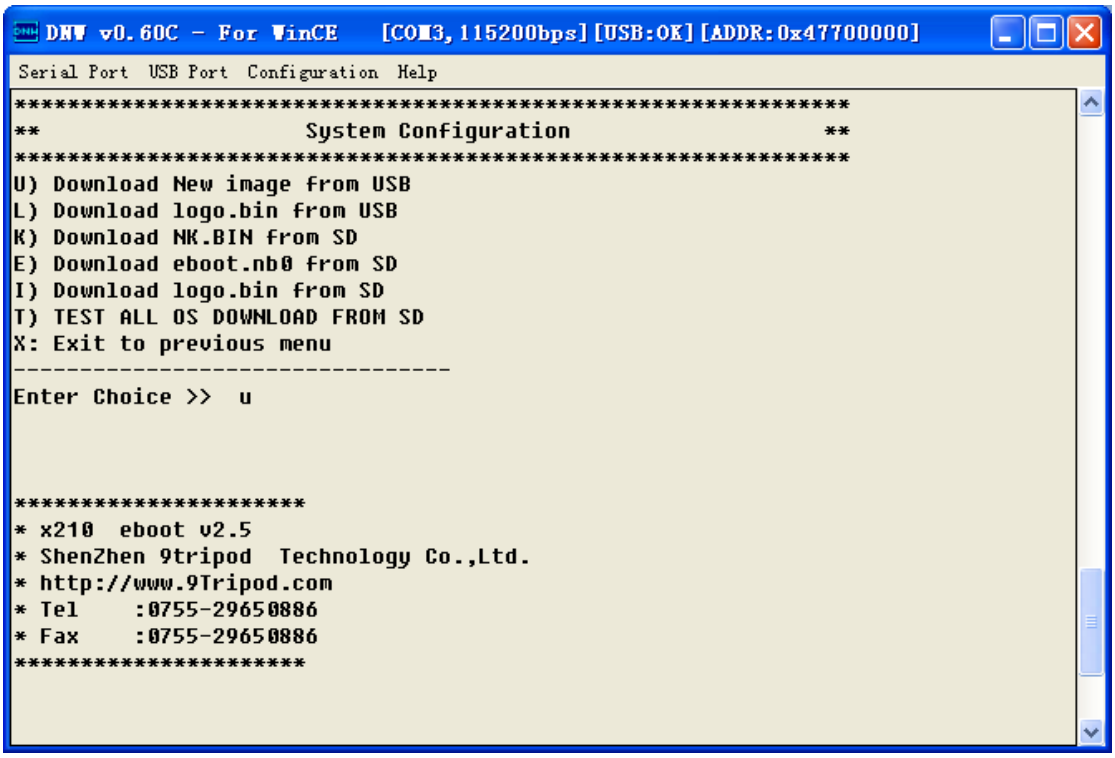
```
busybox dd if=/dev/zero of=/dev/block/mmcblk0 bs=512 seek=1 count=1 conv=sync
```

为了确保执行数据有效，再执行 sync 同步一下，即可破坏掉前一个块的数据。

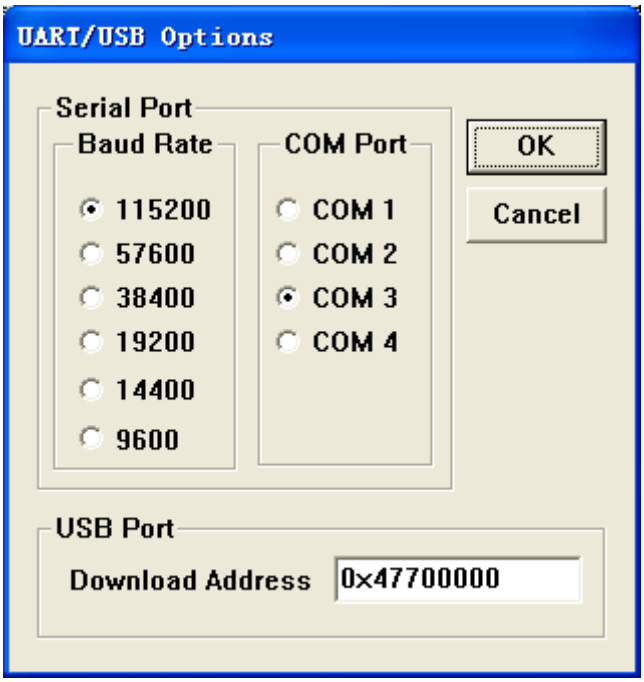
2.5.3 破坏开发板 WINCE 平台下的 bootloader，从 SD2 启动开发板

第一步：在 eboot 5 秒倒计时的时候，按下 PC 机的空格键，进入 eboot 菜单；

第二步：确保 DNW 的地址为 0x47700000，按下 u，如下图所示：



如果地址不是 0x47700000，点击 DNW 的 Configuration，选择 Options，在最下面的 Download Address 一栏设置为 0x47700000 即可：



第三步：点击 DNW 的 USB Port->Uboot，选择 erase.nb0 文件，烧写完成后，即清掉了 eboot，这时，我们就可以从 SD2 通道通过外部 SD 卡启动了。

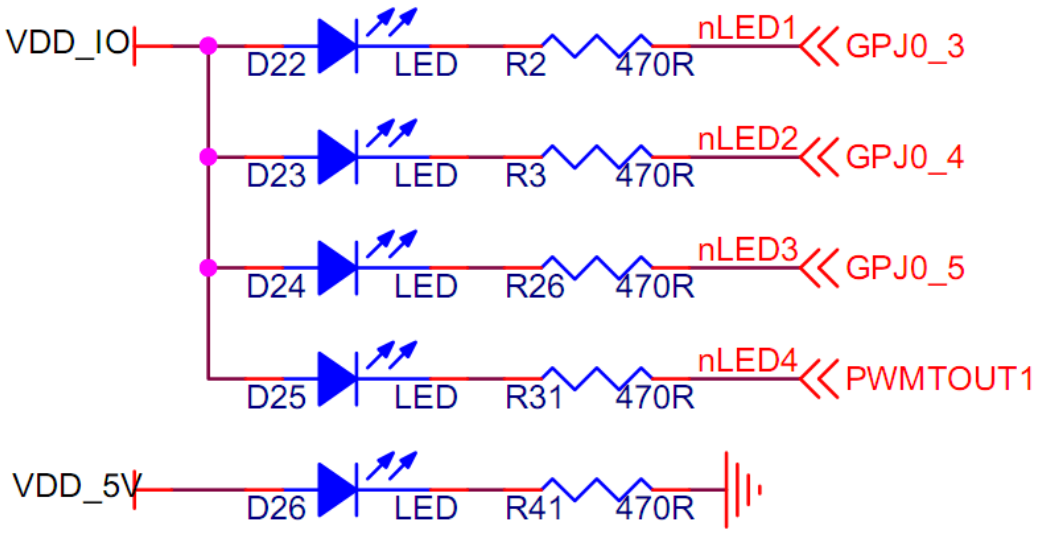
2.6 通过 SD 卡运行裸机程序

将烧有裸机程序的 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键，约 3 秒后即可松手，这时可以发现，四盏 LED 灯已经在来回闪烁了。

第3章 x210v3 裸机实例

3.1 x210v3 裸机开发 1-LED 流水灯实验

3.1.1 原理图



上图中，当按下 POWER 键后，VDD_5V 和 VDD_IO 会产生 5V 和 3.3V 的电压，其中 D26 无需 GPIO 控制，为常亮状态，即我们所说的电源指示灯。D[22:25]对应 GPIO 如下：

LED 指示灯	GPIO 口
D22	GPJ0_3
D23	GPJ0_4
D24	GPJ0_5
D25	GPD0_1

当对应的 GPIO 口为低电平时，相应的 LED 灯被点亮，反之会灭。

3.1.2 源码

说明：由于笔者的工作空间在 D 盘根目录，所以后面提及所有源码包全部在 D 盘，介绍路径时也全部按这个路径介绍，用户如果设置在其他盘，源码包路径会相应变化，特此声明，后续不再重复。

源码路径：

```
D:\workspace\LED\source\main.c
D:\workspace\LED\source\s5pv210-irom.c
D:\workspace\LED\source\s5pv210-cp15.c
D:\workspace\LED\source\start.S
D:\workspace\LED\include\io.h
D:\workspace\LED\include\s5pv210-cp15.h
D:\workspace\LED\include\
D:\workspace\LED\include\types.h
D:\workspace\LED\include\syscfg.inc
D:\workspace\LED\include\s5pv210\reg-others.h
```

D:\workspace\LED\include\s5pv210\reg-gpio.h

这里是我们学习裸机开发的第一个示例，我们会将源码中每一个文件都详细介绍。

在 main()主函数中，首先调用 led_init 函数初始化 led，再通过 led_set 函数控制相应 LED 的亮与灭。在整个 main 函数中，关键在于 while(1)这个死循环，每隔 200ms 设置一次 LED 灯状态，时间间隔通过延时函数 mdelay 实现，LED 灯的状态通过变量 index 实现。

LED 初始化函数如下：

```
void led_init(void)
{
	/* LED1 */
	writel(S5PV210_GPJ0CON, (readl(S5PV210_GPJ0CON) & ~(0xf<<12)) | (0x1<<12));
	writel(S5PV210_GPJ0PUD, (readl(S5PV210_GPJ0PUD) & ~(0x3<<6)) | (0x2<<6));
	writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<3)) | (0x1<<3));

	/* LED2 */
	writel(S5PV210_GPJ0CON, (readl(S5PV210_GPJ0CON) & ~(0xf<<16)) | (0x1<<16));
	writel(S5PV210_GPJ0PUD, (readl(S5PV210_GPJ0PUD) & ~(0x3<<8)) | (0x2<<8));
	writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<4)) | (0x1<<4));

	/* LED3 */
	writel(S5PV210_GPJ0CON, (readl(S5PV210_GPJ0CON) & ~(0xf<<20)) | (0x1<<20));
	writel(S5PV210_GPJ0PUD, (readl(S5PV210_GPJ0PUD) & ~(0x3<<10)) | (0x2<<10));
	writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<5)) | (0x1<<5));

	/* LED4 */
	writel(S5PV210_GPD0CON, (readl(S5PV210_GPD0CON) & ~(0xf<<4)) | (0x1<<4));
	writel(S5PV210_GPD0PUD, (readl(S5PV210_GPD0PUD) & ~(0x3<<2)) | (0x2<<2));
	writel(S5PV210_GPD0DAT, (readl(S5PV210_GPD0DAT) & ~(0x1<<1)) | (0x1<<1));
}
```

这里针对每一个 GPIO 口都设置了三个寄存器，以 LED4 为例讲解，其他雷同。S5PV210_GPD0CON 为 GPD0 的功能寄存器，用于设置该组 GPIO 的功能，如下图：

GPD0CON	Bit	Description	Initial State
GPD0CON[3]	[15:12]	0000 = Input 0001 = Output 0010 = TOUT_3 0011 ~ 1110 = Reserved 1111 = GPD0_INT[3]	0000
GPD0CON[2]	[11:8]	0000 = Input 0001 = Output 0010 = TOUT_2 0011 ~ 1110 = Reserved 1111 = GPD0_INT[2]	0000
GPD0CON[1]	[7:4]	0000 = Input 0001 = Output 0010 = TOUT_1 0011 ~ 1110 = Reserved 1111 = GPD0_INT[1]	0000
GPD0CON[0]	[3:0]	0000 = Input 0001 = Output 0010 = TOUT_0 0011 ~ 1110 = Reserved 1111 = GPD0_INT[0]	0000

当设置为 0 时，相应 IO 口被设置为输入；设置为 1 时，相应 IO 口被设置为输出；设置为其他数据时，参考 210 数据手册。由于 LED4 对应 GPIO 为 GPD0_1，因此我们需要设置 GPD0CON[1]，这时我们在程序中会先将 GPD0CON[7:4]清 0，然后再设置为 1，即将 GPD0_1 设置为输出了。

GPD0PUD 为硬件上下拉设置，如下表：

GPD0PUD	Bit	Description	Initial State
GPD0PUD[n]	[2n+1:2n] n=0~3	00 = Pull-up/ down disabled 01 = Pull-down enabled 10 = Pull-up enabled 11 = Reserved	0x0055

设置为 0 表示禁止上下拉；设置为 1 表示使能下拉；设置为 2 表示使能上拉；

GPD0DAT 为 GPIO 的数据寄存器，当设置为输出时，设置为 1 表示输出高电平；设置为 0 表示输出低电平。当设置为输入时，我们可以通过读取该寄存器的值判断外办输入的电平。

上述 LED 初始化函数将四组 GPIO 设置为输出，同时都设置为高电平，使能上拉。根据硬件电路分析，初始化之后，四盏 LED 灯都会熄灭。紧接着通过 LED 设置函数 led_set()控制 LED 的亮与灭，对应程序如下：

```
void led_set(int id, int on)
{
    switch(id)
    {
    case 0:
        if(on)
            writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<3)) | (0x0<<3));
        else
            writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<3)) | (0x1<<3));
        break;

    case 1:
        if(on)
```

```
                writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<4)) | (0x0<<4));
            else
                writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<4)) | (0x1<<4));
            break;

        case 2:
            if(on)
                writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<5)) | (0x0<<5));
            else
                writel(S5PV210_GPJ0DAT, (readl(S5PV210_GPJ0DAT) & ~(0x1<<5)) | (0x1<<5));
            break;

        case 3:
            if(on)
                writel(S5PV210_GPD0DAT, (readl(S5PV210_GPD0DAT) & ~(0x1<<1)) | (0x0<<1));
            else
                writel(S5PV210_GPD0DAT, (readl(S5PV210_GPD0DAT) & ~(0x1<<1)) | (0x1<<1));
            break;

        default:
            break;
        }
}
```

该函数有两个传入参数 id 和 on，id 对应第几盏灯，on 表示 LED 的亮与灭。亮时，将相应 GPIO 置低，灭时置高。本程序巧妙的运用了变量 index，通过它来实现四盏灯依次被点亮。详细机理读者可仔细琢磨源码中的 while(1)死循环程序。到此，整个 main 函数结束。

前面的源码路径中，列出了很多源文件，而真正干活的好像只有 main.c 一个文件，那么其他文件是否可以删掉不用?答案是否定的，在嵌入式平台上，并不像单片机那样简单的写一个 main 函数就完了，我们还需要实现内存，看门狗，中断等的初始化，实现源码的自拷贝等等。在使用 eclipse 编译时，还需要相应的链接文件，makefile 指定源码的编译和目标的生成。

我们先介绍整个源码的工作流程，后面再根据代码逐个解析。

第一步:将开发板的拨码开关设置成 OM[5:0]=001101,即从 IROM 的 SD/MMC 通道启动。开发板上电时，S5PV210 里面的 IROM 程序得到执行，注意 IROM 程序是固化在 S5PV210 芯片内部，而并非外面的 emmc 中。这段 IROM 程序我们完全无法修改，他会将 emmc 或外部 SD 卡的前面一段代码拷贝到 SRAM 的前 16KB，具体拷贝到大小由 start.s 的最前面的 16byte 决定,我们后面再介绍。这里拷贝的大小最大只允许 16KB,否则会出错。这段代码相当于 S5PV210 手册上介绍的 BL1，即 bootloader1，尽管 SRAM 的大小为 96KB，但是留给 BL1 使用的只有前面的 16KB，剩下的 80KB 留给 BL2 使用。

第二步：IROM 会对拷贝到 SRAM 中的前 16K 字节进行校验和计算，如果不满足校验和，则停止运行程序，如果满足，再进行下一步。

第三步：映像文件的最前面一段映像被拷贝到 SRAM 中后，校验和成功后，PC 指针会跳到 SRAM 的起始地址，拷贝的程序得到运行。这段程序会再次将 SD/MMC 中的整个映像文件拷贝到 SDRAM 的指定地址。

第四步：PC 指针跳转到 SDRAM 的指定地址，程序开始在 SDRAM 中执行。

下面我们针对这个 LED 测试程序，逐步揭开整个运行过程的谜底。

第一步：开发板设置为从 SD 通道启动，将烧有 LED 测试映像的 SD 卡插到开发板的 SD 通道 2，长按 POWER 键开机，这时，S5PV210 的内部 IROM 程序首先将 SD 中的前 8K 拷贝到 SRAM，注意，映像文件必须写到从 SD 卡的第一个扇区开始，第 0 个扇区为 SD 卡的分区表信息，我们任何时候也不要操作它！这里拷贝的长度由映像文件的最前面 16 个字节决定，在 start.s 的最前面有如下代码：

```
/*
 * bl1 header infomation for irom
 *
 * 0x0 - bl1 size
 * 0x4 - reserved (should be 0)
 * 0x8 - check sum
 * 0xc - reserved (should be 0)
 */
    .word 0x2000    /* 自拷贝文件大小 */
    .word 0x0
    .word 0x0
    .word 0x0

    .global    _start
_start:
```

前面填充了 16 个字节，其中第一个字节我们设置为 0x2000，即 8KB。第二个和第四个必须为 0，否则会出错。第三个为校验和文件。

第二步：IROM 程序通过读取 0x8 开始的一个字的内容，进行校验和比对，如果校验和通过，则继续运行，否则停止运行。因此，我们务必将生成的映像文件的前 16 个字节的第 3 个字做必要的修改，即将校验数据填充进去，否则映像无法运行。我们已经做好了填充校验和工具 mkv210.exe，用户直接使用即可，可以不关心里面的工作原理。

第三步：程序从 SRAM 的第 17 个字节开始运行，即 start.s 里面的_start 地址。注意，由于前面我们填充的自拷贝文件的大小仅为 8K，因此为 8K 程序必须保证能够将所有的映像全部搬运到 SDRAM 中。实现自拷贝的文件为 start.s 和 s5pv210-irom.c 两个文件。我们编译完 LED 源码后，会生成 led.map 文件，打开该文件，最前面有如下内容：

```
.obj/source/start.o(.text)
 .text          0x34000000      0x8c0 .obj/source/start.o
                0x34000010              _start
                0x34000754              irq
.obj/source/s5pv210-irom.o(.text)
 .text          0x340008c0      0xd4 .obj/source/s5pv210-irom.o
                0x340008c0              irom_copyself
```

可见，start.s 占用了 0x8c0，s5pv210-irom.c 占用了 0xd4，即一起占用了 0x994，即 2.39453125KB，远远小于 8KB。

自拷备程序首先跳到 reset 地址，在这里禁止看门狗，设置 CPU 工作模式，设置 cache，设置 MMU，将 GPH0_0 置高，置锁开发板电源，初始化系统时钟，禁止 MMU，再执行自拷贝函

数。自拷贝代码如下：

```
    /* copyself to ram using irom */
    adr   r0, _start
    ldr r1, =_start
    cmp  r0, r1
    beq   have_copyed
    bl     irom_copyself
```

注意，这里源码完全不同于 uboot，而且大大优于 uboot，在 uboot 中需要将整个 uboot 映像分成两块，一块用于自拷贝，另一块为真正引导内核的 bootloader，并分两次烧写，而这里我们只需要烧写一次就可以了，关键就在上面这几句汇编，第一句，通过 adr 指令获取当前_start 的地址，第二句，获取_start 应该被拷贝的地址，链接脚本 link.ld 中有如下内容：

```
MEMORY
{
    rom (rx)    : org = 0x34000000, len = 0x02000000   /* 32 MB */
    ram (rwx)  : org = 0x36000000, len = 0x0a000000   /* 160 MB */
}

SECTIONS
{
    .text :
    {
        . = ALIGN(8);
        PROVIDE (__text_start = .);
        .obj/source/start.o (.text)
        .obj/source/s5pv210-irom.o (.text)
        *(.text)
        *(.text.*)

        . = ALIGN(8);
        *(.rodata);
        *(.rodata.*);

        . = ALIGN(8);
        *(.glue_7);
        *(.glue_7t);

        . = ALIGN(8);
        PROVIDE (__text_end = .);
    } > rom
…
}
```

可以看到，start.s 和 s5pv210-irom.c 存放在代码的最前面，而且被定位到 ROM，在

MEMORY 表中可以看到 ROM 的起始地址为 0x34000000，在 start.s 的最前面填充了 16 个字节，紧接着就是_start 的地址，即_start 正常工作的地址为 0x34000010。

当程序在 SRAM 中运行时，通过 adr 指令获取的地址为 SRAM 的地址，SRAM 的起始地址为 0xD0020000，再加上 16 个字节的头文件，我们获取的地址应该为 0xD0020010。可见，当程序在 SRAM 中运行时，后面执行的比较指令判断不相等，程序进入自拷贝。后面该段程序在 SDRAM 中运行时，比较通过 adr 和 ldr 指令获取的地址都为 0x34000010，则程序不会自拷贝。

这里自拷贝函数 irom_copyself 在 s5pv210-irom.c 中，函数内容如下：

```
void irom_copyself(void)
{
	u8_t om;
	u32_t * mem;
	u32_t page, block, size;

	/*
	 * read om register, om[4..1]
	 */
	om = (u8_t)((reg_read(S5PV210_OMR) >> 1) & 0x0f);

	/* essd */
	if(om == 0x0)
	{

	}

	/* nand 2KB, 5-cycle, 8-bit ecc */
	else if(om == 0x1)
	{

	}

	/* nand 4KB, 5-cycle, 8-bit ecc */
	else if(om == 0x2)
	{

	}

	/* nand 4KB, 5-cycle, 16-bit ecc */
	else if(om == 0x3)
	{

	}

	/* onenand mux */
```

```
else if(om == 0x4)
{

}

/* onenand demux */
else if(om == 0x5)
{

}

/* sd / mmc */
else if(om == 0x6)
{
	/*
	 * the xboot's memory base address.
	 */
	mem = (u32_t *)__text_start;

	/*
	 * the size which will be copyed, the 'size' is
	 * 1 : 256KB, 2 : 512KB, 3 : 768KB, 4 : 1024KB ...
	 */
	size = (__data_shadow_end - __text_start + 0x00040000) >> 18;

	/*
	 * how many blocks the 'size' is , 512 bytes per block.
	 * size * 256 *1024 / 512 = size * 2^9 = size << 9
	 */
	size = size << 9;

	/*
	 * copy xboot to memory from sd/mmc card.
	 */
	if(irom_v210_sdmmc_base == 0xeb000000)
	{
		irom_sdmmc_to_mem(0, 1, size, mem, 0);
	}
	else if(irom_v210_sdmmc_base == 0xeb200000)
	{
		irom_sdmmc_to_mem(2, 1, size, mem, 0);
	}
	else
```

```
            {
                return;
            }
        }

        /* emmc, 4-bit */
        else if(om == 0x7)
        {

        }

        /* reserved */
        else if(om == 0x8)
        {

        }

        /* nand 2KB, 4-cycle, 8-bit ecc */
        else if(om == 0x9)
        {

        }

        /* nor flash */
        else if(om == 0xa)
        {

        }

        /* emmc, 8-bit */
        else if(om == 0xa)
        {

        }

        /* not support */
        else
        {
            return;
        }
}
```

函数首先通过读取寄存器 S5PV210_OMR 的值，判断程序从什么存储媒介启动，之后再进行相关操作。由于我们裸机全部在 SD 卡中运行，因此这里只需执行 OM=6 对应的程序。程序中，

__text_start 表示程序的起始地址，__data_shadow_end 表示程序的结束地址，二者相减，即得到了我们需要拷贝的程序的总大小。在程序中，我们先加上 256KB，再除 256KB，是为了保证我们拷贝的数据是 256KB 的整数倍。由于程序拷贝有一个对齐的过程，如果我们不进行处理，有可能会出现拷贝不全，这样程序就没法运行了。

紧接着通过 IROM 中的变量 irom_v210_sdmmc_base 判断从 SD0 还是 SD2 通道拷贝数据，执行拷贝的关键函数是 irom_sdmmc_to_mem，它是固化在 IROM 中，留给 bootloader 拷贝的函数，我们只需能够使用它就可以了。

当程序从 SD 卡中将数据搬运到 SDRAM 中后，在 start.s 中，程序接着往下执行，在 have_copyed 地址处，开始初始化栈，拷贝程序声明的变量，然后跳到 SDRAM 中执行。

```
copy_shadow_data:
    ldr   r0, _data_shadow_start
    ldr   r1, _data_start
    ldr   r2, _data_shadow_end
    bl    mem_copy
```

这里是一段内存拷贝程序，为了理解这一段话，我们先打开 link.ld 链接脚本文件，里面有如下三句话：

```
.data_shadow ALIGN(8) :
.data : AT ( ADDR (.data_shadow) )
.bss ALIGN(8) (NOLOAD) :
```

分别对应三段地址，第一个用于存放程序初始化的值，第二个用于存放程序中已经初始化值的变量的地址，第三个用于存放程序中没有初始化值的变量的地址。比如我们在程序中做如下声明：

```
int i=5,j;
```

那么， 变量 i 会在.data 段申请一个地址，变量 j 会在.bss 段申请一个地址，被赋的值 5 将被拷贝到.data_shadow 中。再回到前面的汇编，第一句将_data_shadow 的起始地址拷贝到 r0，第二句将_data 的起始地址拷贝到 r1，第三句将_data_shadow 的结束地址拷贝到 r2，第四句将调用 mem_copy 函数，将_data_shadow 中从 r0 开始，长度为 r2 的数据拷贝到从 r1 开始的地址中去。一句话概括函数实现的功能：将程序中声明的变量拷贝到相应内存。

```
clear_bss:
    ldr   r0, _bss_start
    ldr   r1, _bss_end
    mov r2, #0x00000000
    bl    mem_clear
```

这段函数用于将未赋值的变量清零。下面进入关键程序段，程序首先从 SRAM 中跳转到 SDRAM，然后从汇编跳到 C 语言的 main 函数。

```
    ldr   r1, =on_the_ram
    mov  pc, r1
```

注意，整段 start.s 中的变量，在链接脚本中已经将他们定位到了 0x34000000 到 0x40000000，因此，在程序中任何执行绝对地址跳转到这些变量，都会从 SRAM 中跳到 SDRAM 中。上面代码通过 ldr 指令，将变量 on_the_ram 的地址赋给 r1，然后将 PC 指针指向 r1，这里虽然程序在 SRAM 中执行，但是 on_the_ram 变量的地址仍然在 SDRAM 段，执行上面两句话后，程序就从 SRAM 中跳转到 SDRAM 中执行了，跳转地址也就是在 on_the_ram 开始的地址。

```
on_the_ram:
```

```
/* jump to main fuction */
mov r0, #1;
mov r1, #0;
bl    main            /* call main function */
b     on_the_ram
```

上面这段程序，是在 SDRAM 中运行的第一段程序。前面将 1 赋给 r0，将 0 赋给 r1，然后跳转到 C 语言的 main 函数。这里 r0，r1 会传递到 main 的传入参数 argc 和* argv[]，这里我们没有用到，先不管他。在 C 语言中，紧接着就是执行 LED 流水灯程序了。

3.1.3 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，可以看到四盏 LED 灯开始循环显示。我们可以通过设置 mdelay 函数的传入参数来调节流水时间间隔。

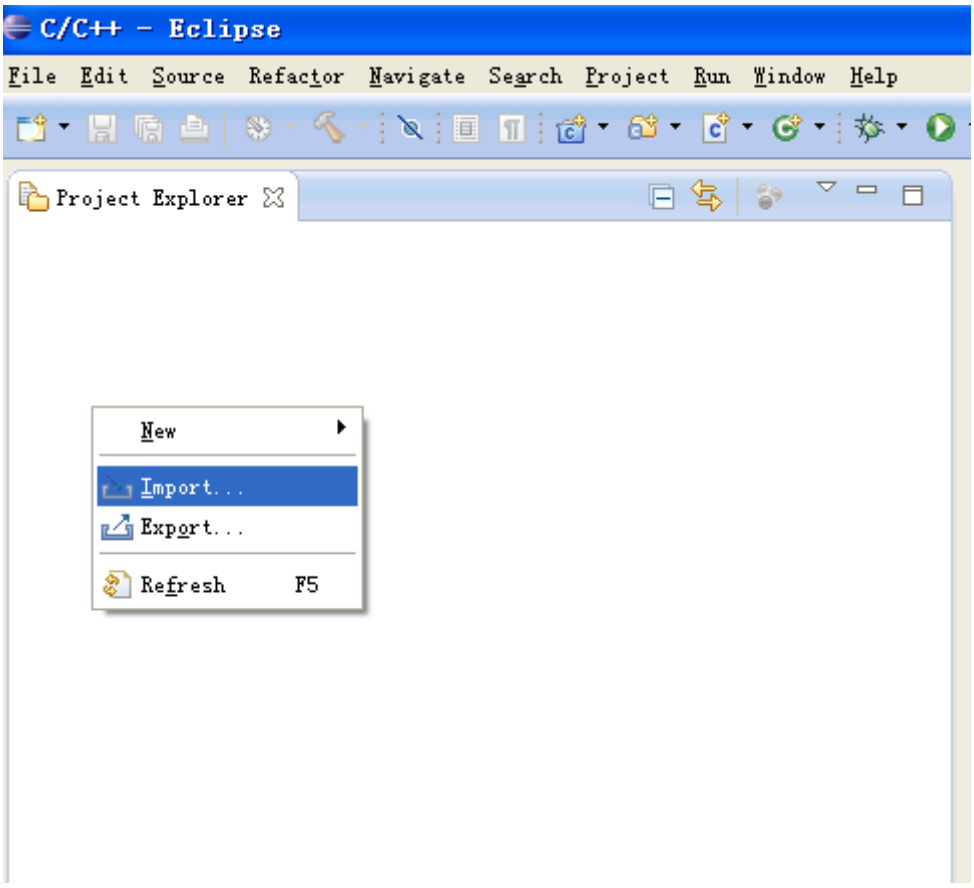
3.2 x210v3 裸机开发 2-蜂鸣器实验

3.2.1 导入已有的工程

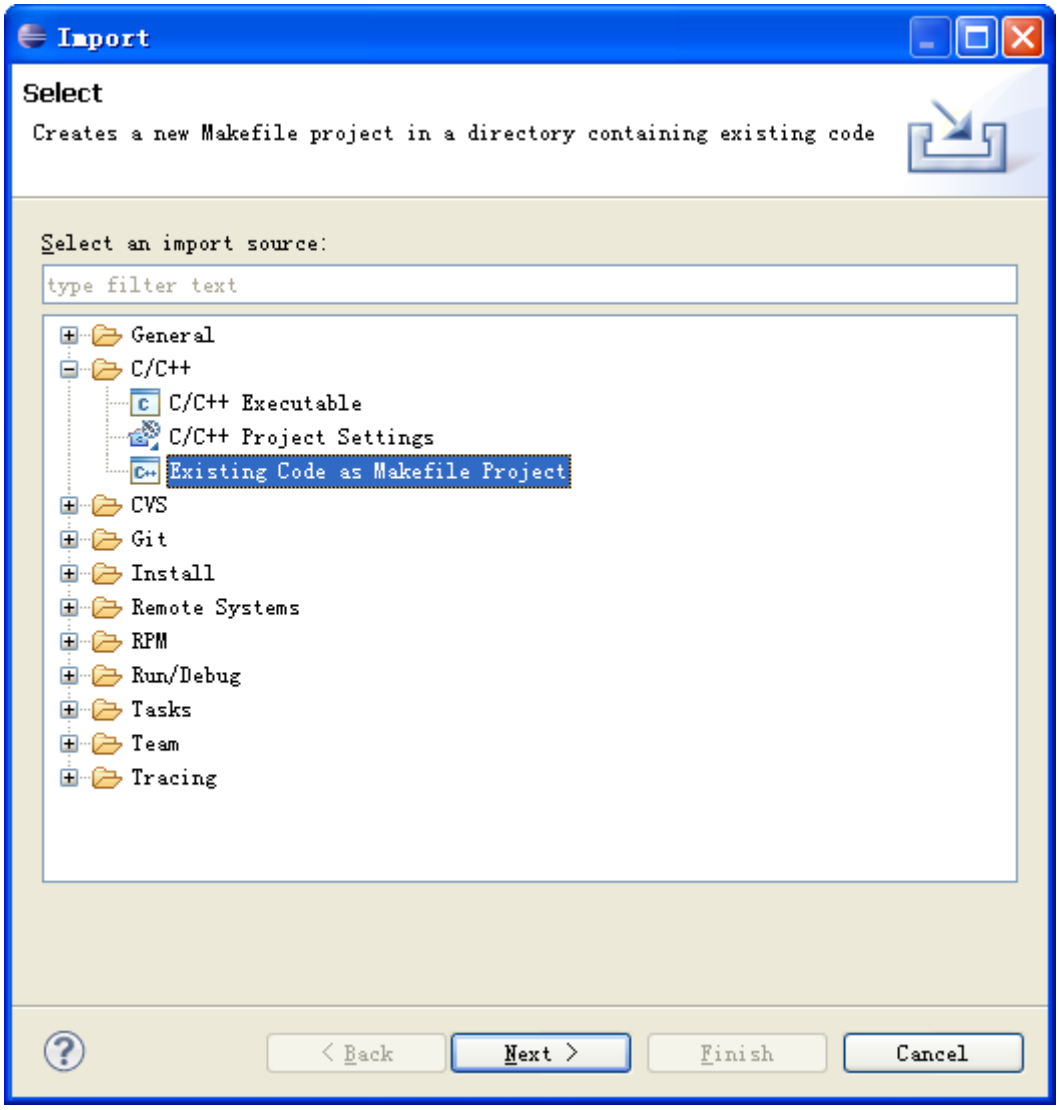
这一章节我们学习如何导入现有的工程。

第一步：将按键检测程序 buzzer.rar 拷贝到工程目录并解压；

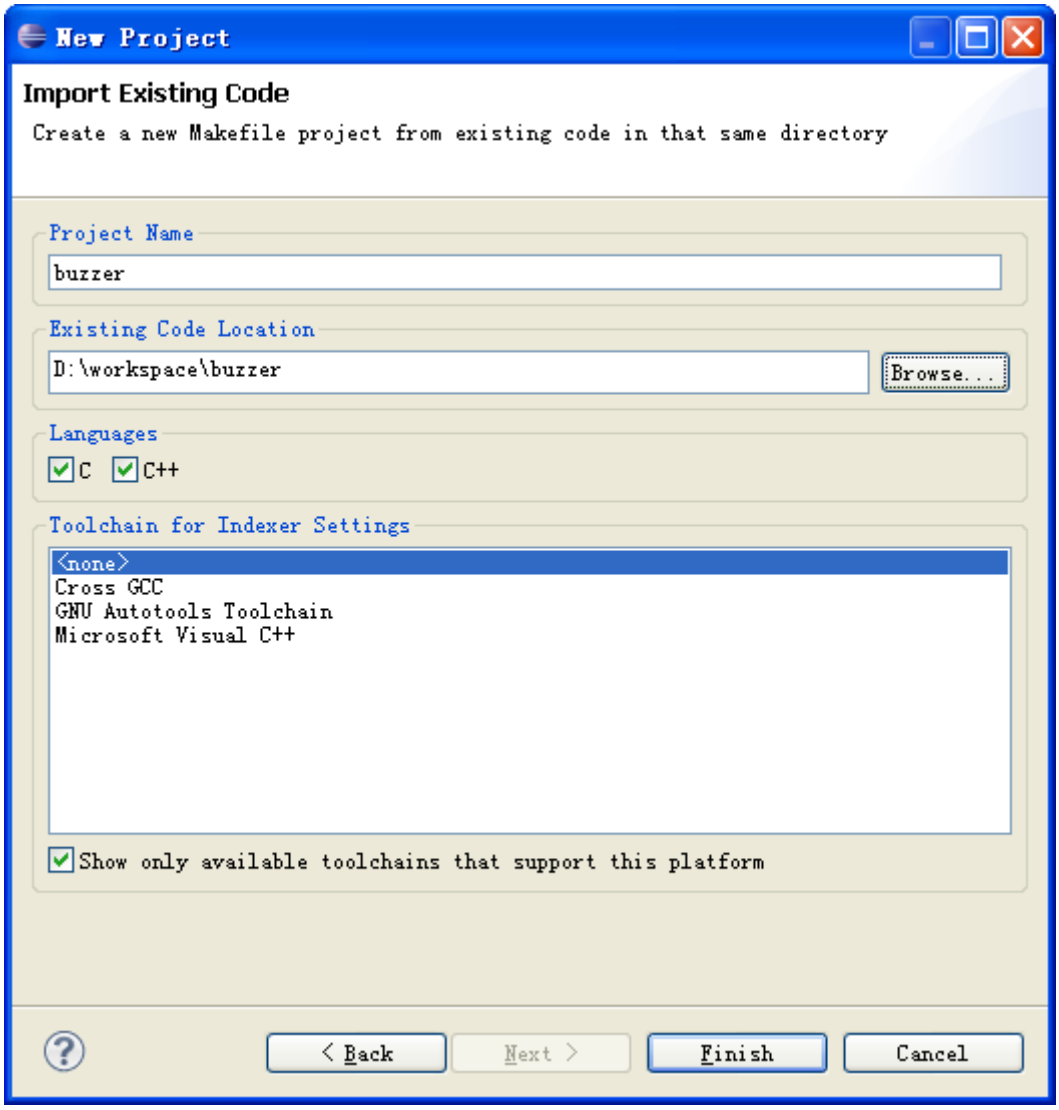
第二步：在 eclipse 的 Project Explorer 的空白处右键，点击 Implort，



弹出如下界面：

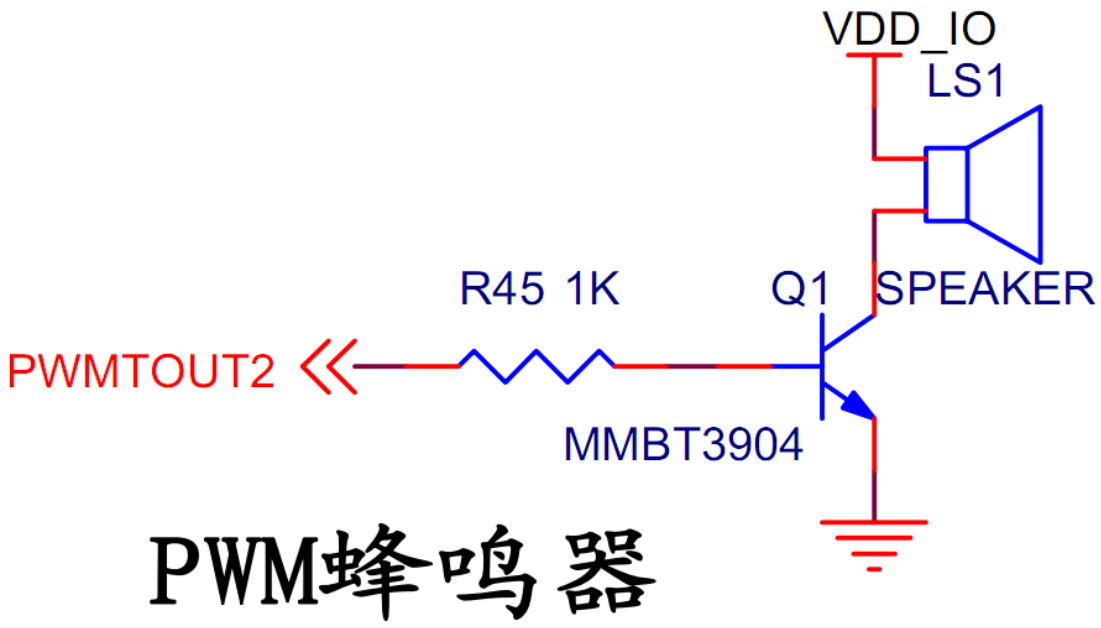


在 C/C++一栏选择 Existing Code as Makefile Project，点击 Next；



点击 Browse，选择解压出来的工程根目录，点击 Finish，即完成工程的导入。

3.2.2 原理图



上述电路通过一个 NPN 的三级管控制蜂鸣器的停止与蜂鸣。

GPIO 状态	蜂鸣器状态
GPD0_2 = 0	停止
GPD0_2 = 1	蜂鸣

3.2.3 源码

D:\workspace\buzzer\source\main.c

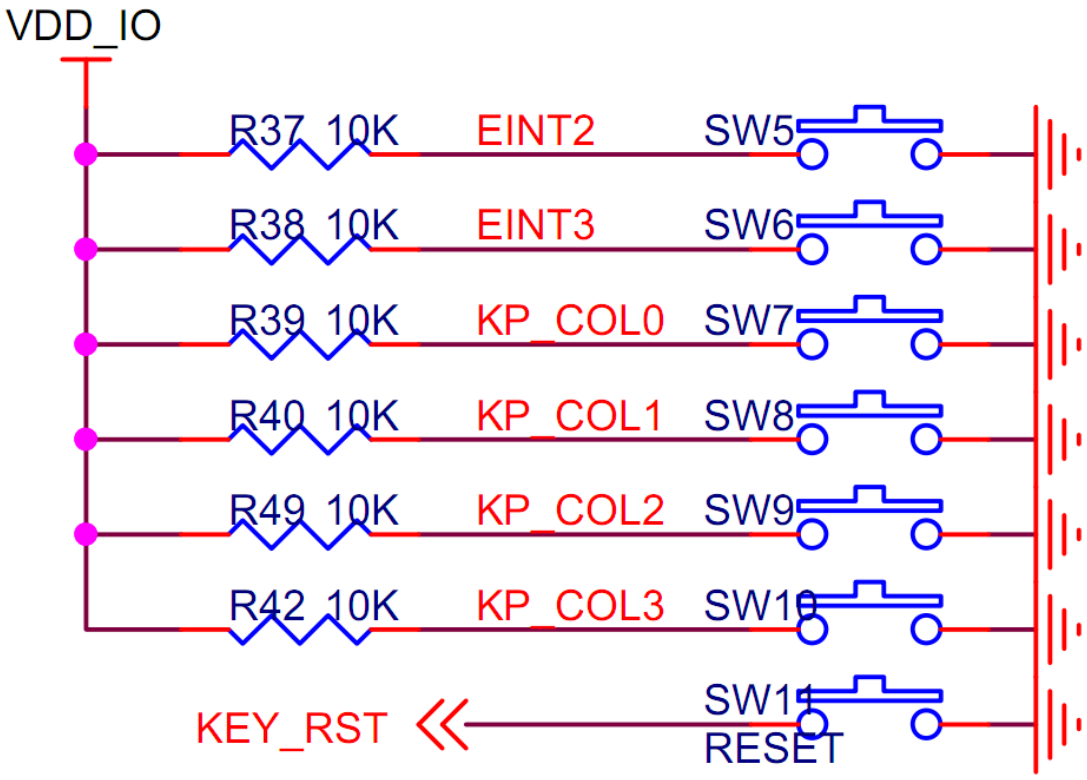
在 main 函数中，调用 `speaker_init` 函数初始化控制蜂鸣器的 `GPIO` 口，然后进入 `while(1)`死循环，每隔约 `500ms`，蜂鸣器状态改变一次，实现反复停止，蜂鸣的功能。

3.2.4 实验现象

将SD卡插到x210v3开发板的SD2通道，长按 POWER键3秒，可以听到每隔约500ms，蜂鸣器会鸣叫一次。我们可以通过设置 mdelay 函数的传入参数来调节蜂鸣时间间隔。

3.3 x210v3 裸机开发 3-按键控制 LED 灯实验

3.3.1 原理图



按键	网络标号	GPIO
SW5(LEFT)	EINT2	GPH0_2
SW6(DOWN)	EINT3	GPH0_3
SW7(UP)	KP_COL0	GPH2_0
SW8(RIGHT)	KP_COL1	GPH2_1
SW9(BACK)	KP_COL2	GPH2_2
SW10(MENU)	KP_COL3	GPH2_3

对应 GPIO 口默认硬件上拉，有按键按下时对应 IO 口变为低电平。我们可以将对应 GPIO 口设置为输入，检测对应的电平即可。

3.3.2 源码

D:\workspace\key_led\source\main.c

由于我们需要通过按键控制 LED，所以要将 LED 和按键的 GPIO 口都初始化。main 函数的开始通过 led_init 和 key_init 函数初始化对应的 IO 口，然后进入 while(1) 死循环，不断的判断 LEFT，DOWN，RIGHT，BACK 四个按键。默认这四个键没有按下时，执行 if() 下面的语句，置四个 LED 为低。一旦有键按下，则执行相应的 else 语句，点亮对应 LED 灯，从而达到按键控制 LED 的目的。

3.3.3 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，默认情况下，四盏 LED 灯都为灭的状态。按下 LEFT，DOWN，RIGHT，BACK 键，对应的 LED 会被点亮，同时按下，LED 灯也会同时被点亮，松开即灭。

3.4 x210v3 裸机开发 4-按键控制蜂鸣器实验

3.4.1 原理图

同上一章节。

3.4.2 源码

D:\workspace\key_buzzer\source

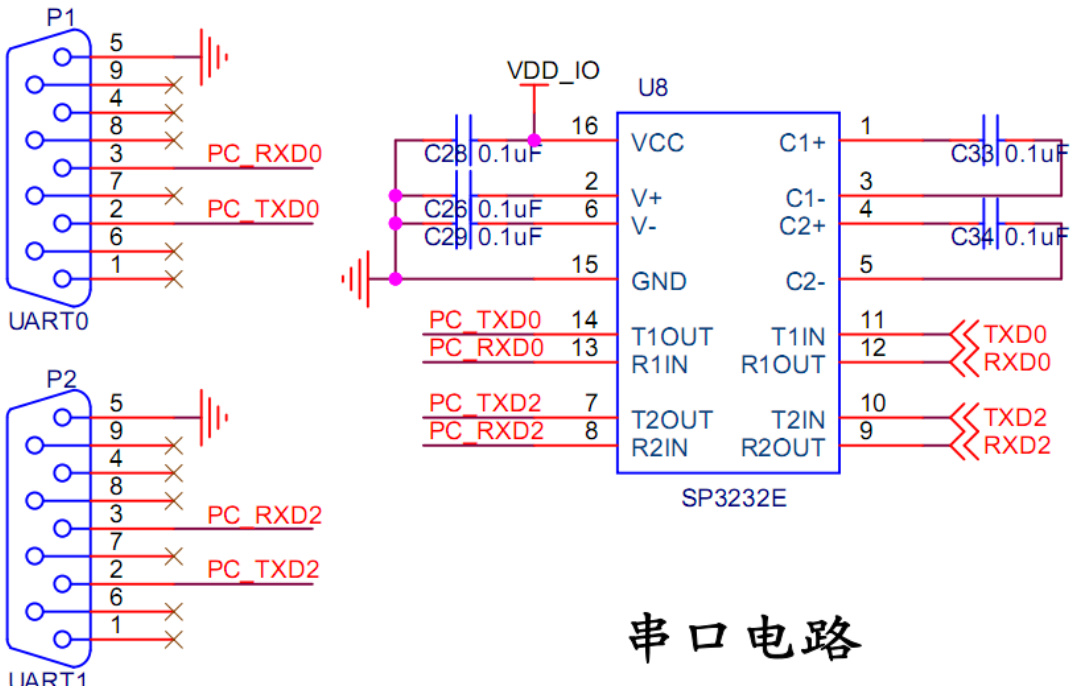
由于我们需要通过按键控制蜂鸣器，所以要将蜂鸣器和按键的 GPIO 口都初始化。这里我们保留了上一章节的控制 LED 的程序，因此，在 main 函数中，通过 `led_init`，`key_init` 以及 `speaker_init` 函数初始化对应的 IO 口。紧接着进入 `while(1)` 死循环，不断的判断 LEFT，DOWN，RIGHT，BACK，UP 以及 MENU 六个按键。默认这六个键没有按下时，执行 `if()` 下面的语句，置四个 LED 为低，同时关闭蜂鸣器。一旦有键按下，则执行相应的 `else` 语句，点亮对应 LED 灯，或是打开蜂鸣器，从而达到按键控制 LED 和蜂鸣器的目的。

3.4.3 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，默认情况下，四盏 LED 灯都为灭的状态，蜂鸣器默认不发声。按下 LEFT，DOWN，RIGHT，BACK 键，对应的 LED 会被点亮，同时按下，LED 灯也会同时被点亮，松开即灭。按下 UP 键，四盏 LED 灯会同时被点亮，同时，蜂鸣器会一直发声，直到松开 UP 键。按下 MENU 键，蜂鸣器会间隔鸣叫，同时，四盏 LED 灯会随着蜂鸣器的鸣叫而跟着节奏闪烁。松开 MENU 键，结束闪烁和鸣叫。

3.5 x210v3 裸机开发 5-串口输入输出实验

3.5.1 原理图



由于 S5PV210 的串口输出电平为 3.3V，而标准的 PC 机的串口为 RS232 电平，因此需要进行电平转换。比较常用的是 SP3232 电平转换芯片。

PIN	GPIO
TXD0	GPA0_1
RXD0	GPA0_0
TXD2	GPA1_1
RXD2	GPA1_0

3.5.2 源码

D:\workspace\uart\source\main.c

S5PV210 有四路串口，x210v3 开发板预留了两个带电平转换的串口，电路图见前面原理图章节。我们只需将对应的串口寄存器初始化，然后通过相应的寄存器读取和输出相应的字符即可。尽管串口看起来比较简单，但是如此简单的程序，在我们刚开始调试硬件时，却能起到至关重要的作用。在硬件电路板完全没有跑起来时，可以通过写一些简单的串口打印之类的测试程序来定位故障。

在 main 函数中，调用 uart_init 函数初始化串口的相关寄存器，然后通过 uart_getc 函数获取外界输入的字符，再通过 uart_putc 函数将获取的字符串打印出来。

3.5.3 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，运行串口测试程序。使用串口延长线连接 PC 机和 x210v3 开发板的 UART0，在 PC 机上打开串口调试助手 V2.2，并设置正确的串口和波特率等，如下图：

串口设置为 PC 机上使用的串口，波特率设置为 115200，校验位设置为 NONE，数据位为 8，停止位为 1，并打开串口。在发送的字符/数据一栏输入我们需要发送的字符串，点击手动发送，这时，在接收区就可以看到来自开发板的串口输出信息了。

3.6 x210v3 裸机开发 6-LED 测试程序

前面讲解了一些非常简单的裸机程序，而且源码也尽可能的少，便于读者理解。但是真

正在嵌入式系统，哪怕是 bootloader 中，源码都非常的庞大，这就需要我们有一定的阅读代码的能力。从这个实验开始，我们会将能够用到的测试程序整理成函数或是库，在 main 函数中直接调用即可。为了加深理解，后续的测试程序同样包含了前面的几个简单的测试程序。

3.6.1 源码

源码路径：

```
D:\workspace\template-led\source\main.c
D:\workspace\template-led\source\tester-led.c
D:\workspace\template-led\source\hardware\hw-led.c
```

在主函数 main()中，首先调用 do_system_initial 函数初始化一系列硬件相关寄存器，然后通过 tester_led 函数测试 LED。在 do_system_initial 函数中，包含了 LED 初始化函数 led_initial，在 hw-led.c 中。该函数用于将控制 LED 的相应 GPIO 口设置为输出，并初始化为 1，关闭 LED 灯。tester_led 函数在 tester-led.c 中，通过 while(1)死循环实现流水灯。

可以看到，在 D:\workspace\template-led\source\hardware 路径下，我们实现了很多硬件相关的函数，如蜂鸣器，按键，LED，时钟，中断，LCD，串口，延时等。读者可以直接调用这里面的函数实现自己想要的功能。

如果您比较细心，不难发现，前面的 LED 实验中，最终生成的映像文件 LED.bin 仅有 8K 大小，而本实验生成的映像文件，已经达到 118K。这是因为编译脚本默认将源码包中所有的.c 文件都编译进去了。用户可以直接将没有使用的源码删掉，减小映像大小。

3.6.2 实验现象

这里实验现象和前面的相同，将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，可以看到四盏 LED 灯开始循环显示。我们可以通过设置 mdelay 函数的传入参数来调节流水时间间隔。

3.7 x210v3 裸机开发 7-蜂鸣器实验

3.7.1 源码

```
D:\workspace\template-beep\source\main.c
D:\workspace\template-beep\source\tester-beep.c
D:\workspace\template-beep\source\hardware\hw-beep.c
```

同样，在主函数 main()中，通过 do_system_initial 函数初始化硬件相关寄存器，再调用 tester_beep 函数实现间隔蜂鸣。在 do_system_initial 函数中，调用 beep_initial 函数初始化控制蜂鸣器的 GPIO 口，在 tester_beep 函数中，通过 while(1)死循环实现蜂鸣器反复间隔鸣叫。

3.7.2 实验现象

这里实验现象和前面的相同，将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，可以听到每隔约 500ms，蜂鸣器会鸣叫一次。我们可以通过设置 mdelay 函数的传入参数来调节蜂鸣时间间隔。

3.8 x210v3 裸机开发 8-按键控制 LED 实验

3.8.1 源码

D:\workspace\template-key-with-led\source\main.c
D:\workspace\template-key-with-led\source\tester-key-with-led.c
D:\workspace\template-key-with-led\source\hardware\hw-key.c

在主函数 main 中通过 do_system_initial 函数调用 led_initial 和 key_initial 函数初始化 LED 和按键，然后调用 tester_key_with_led 函数测试按键。在 tester_key_with_led 函数中，首先熄灭四盏 LED 灯，然后调用 get_key_event 函数获取每一个按键的按下或抬起状态，进而控制相应的 LED 灯。

3.8.2 实验现象

这里实验现象和前面的相同，将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，默认情况下，四盏 LED 灯都为灭的状态。按下 LEFT，DOWN，RIGHT，BACK 键，对应的 LED 会被点亮，同时按下，LED 灯也会同时被点亮，松开即灭。

3.9 x210v3 裸机开发 9-按键控制 LED 和蜂鸣器

3.9.1 源码

D:\workspace\template-key-with-led-beep\source\main.c
D:\workspace\template-key-with-led-beep\source\tester-key-with-led-beep.c
D:\workspace\template-key-with-led-beep\source\hardware\hw-key.c
D:\workspace\template-key-with-led-beep\source\hardware\hw-beep.c
D:\workspace\template-key-with-led-beep\source\hardware\hw-led.c

在主函数 main() 中，通过 do_system_initial 函数调用 led_initial，beep_initial 以及 key_initial 函数初始化 LED，蜂鸣器以及按键的硬件 GPIO，再调用 tester_key_with_led_beep 函数测试按键控制 LED 和蜂鸣器。

3.9.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键 3 秒，默认情况下，四盏 LED 灯都为灭的状态。按下 LEFT，DOWN，RIGHT，BACK 键，对应的 LED 会被点亮，同时蜂鸣器会不断鸣叫。松开按键，LED 灯会熄灭，同时蜂鸣器也停止蜂鸣。

3.10 x210v3 裸机开发 10-移植 printf 函数

3.10.1 源码

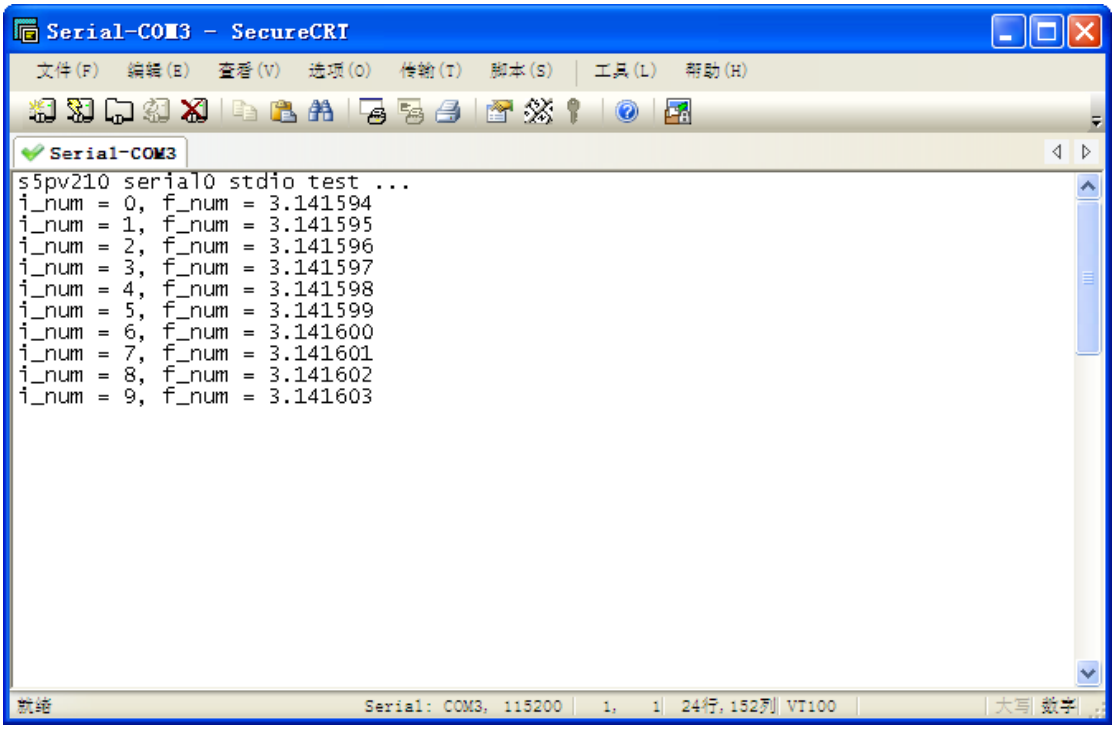
D:\workspace\template-serial-stdio\source\main.c
D:\workspace\template-serial-stdio\source\tester-serial-stdio.c
D:\workspace\template-serial-stdio\source\hardware\s5pv210-serial.c
D:\workspace\template-serial-stdio\source\hardware\s5pv210-serial-stdio.c

在主函数 main()中，通过 do_system_initial 函数调用 s5pv210_serial_initial 函数初始化四路串口，通过 tester_serial_stdio 函数测试串口 0.如果需要测试其他串口，只需在 tester_serial_stdio 函数中，将 s5pv210_serial_write_string 和 serial_printf 的第一个传入参数改为想要测试的串口即可。

3.10.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键 3 秒，默认情况下，串口

会有如下打印：



3.11 x210v3 裸机开发 11-汇编向 C 传递参数实验

3.11.1 源码

D:\workspace\template-asm-c\source\main.c

D:\workspace\template-asm-c\source\tester-asm-c.c

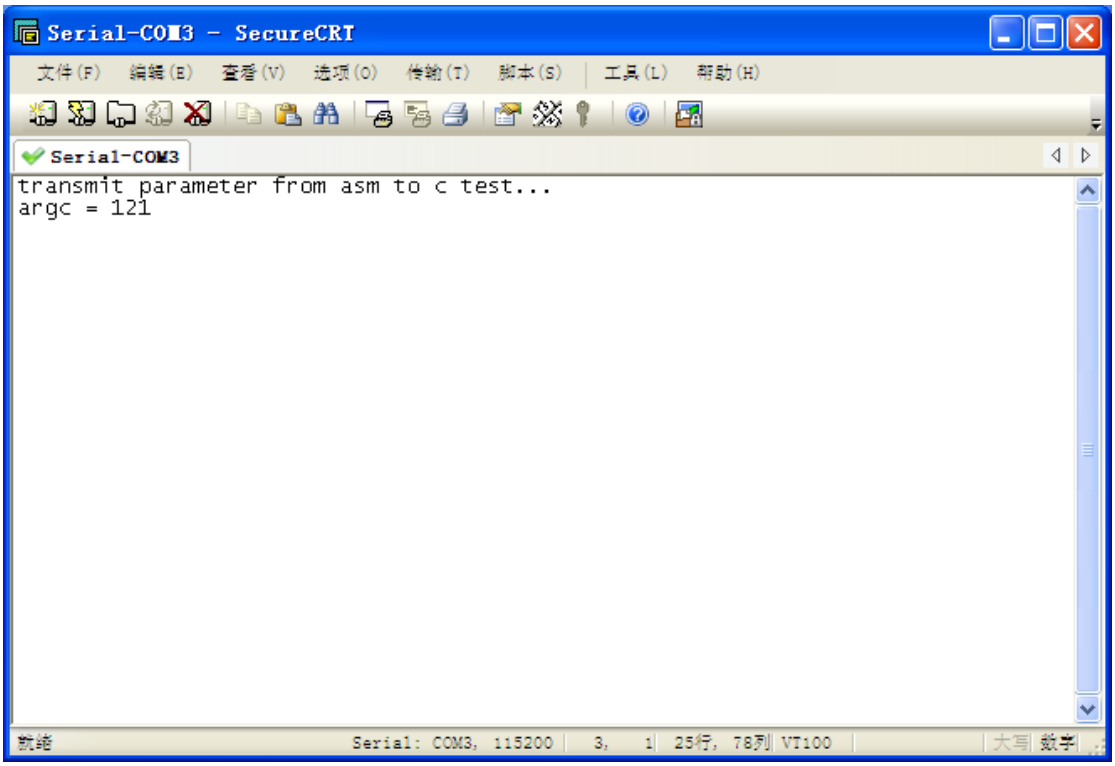
在 start.s 中，跳到 C 语言程序之前，执行如下语句：

```
mov r0, #121;
mov r1, #0;
bl    main              /* call main function */
b     on_the_ram
```

这里，将 121 赋给 r0，将 0 赋给 r1。其中，r0 将会被传递给 main 函数的传入参数 argc。在 main 函数中，tester_asm_c 函数通过 serial_printf 函数将 argc 的值打印出来。

3.11.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键 3 秒，默认情况下，串口会有如下打印：



更改汇编中 r0 的值，重新编译源码，更新映像后再重启开发板，可以看到 argc 的打印会根着变化。

3.12 x210v3 裸机开发 12-时钟分频实验

3.12.1 源码

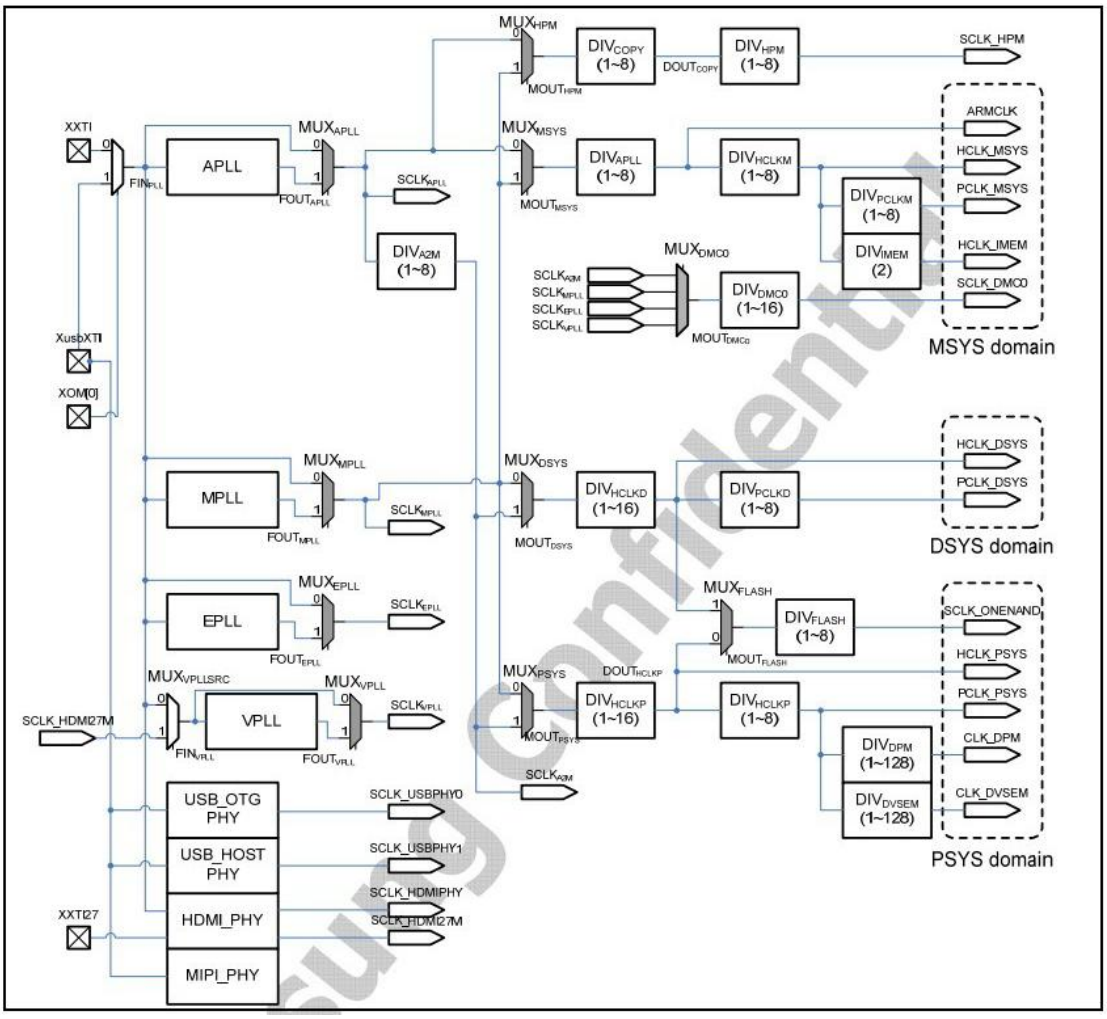
D:\workspace\template-clk\source\main.c

D:\workspace\template-clk\source\tester-clk.c

D:\workspace\template-clk\source\hardware\s5pv210-clk.c

本程序实现各种频率分频，并将频率打印出来。在 main() 主函数中，do_system_initial 函数调用 s5pv210_clk_initial 函数初始化时钟，再执行 tester_clk 函数，通过 clk_get_rate 函数获取相应的时钟频率，最后通过 serial_printf 函数将时钟打印出来。

我们跟踪 s5pv210_clk_initial 函数，不然发现，它调用的函数 s5pv210_setup_clocks 并没有真正去设置时钟相关的寄存器，仅仅是通过读取相关寄存器，然后把 armclk,hclk 以及 pclk 计算出来。事实上，真正时钟频率的设置，在 start.s 中。为了更加清晰的理解整个芯片的时钟，我们可以仔细阅读下图。



图中最左边 XXTI，XusbXTI，XXTI27 对应核心板上 24M，24M，27M 三个晶体，从图片左上脚可以看出，各个锁相环的时钟源，全部可以通过 OM[0]管脚配置成 XusbXTI，但是 USB 和 MIPI 的时钟源只能配置成 XusbXTI。可见，我们完全可以将 XXTI 这颗晶体省掉。HDMI 的时钟也可以配置成三个晶体中的任何一个，而且 27M 的晶体仅供 HDMI 使用。如果我们将 HDMI 的时钟源设置为 XusbXTI，那么外面 27M 的晶体也可以省掉了，这就是 x210cv3 核心板上省掉两个晶体的原因。同时，晶体省掉后，意味着 OM[0]在任何情况下必须设置为 1，否则无法启动。

最后面的 armclk,各 pclk,hclk 等时钟全部是通过 APLL,MPLL,EPLL,VPLL 分频而来，具体他们对应的时钟源，分频系数等，由一串寄存器决定。因此，设置各自时钟，即设置各 PLL 输出时钟，以及分频，选择寄存器即可。

在 start.s 中，以下指令用于设置时钟频率：

```
system_clock_init:
    ldr   r0, =0xe0100000

    mov  r1, #0xe00
    orr   r1, r1, #0x10
    str   r1, [r0, #0x00]        //*(0xe0100000)=0xe10
    str   r1, [r0, #0x04]        //*(0xe0100004)=0xe10
```

```
    str     r1, [r0, #0x08]          //*(0xe0100008)=0xe10
    str     r1, [r0, #0x0c]          //*(0xe010000c)=0xe10

    ldr r1, =CLK_DIV0_VAL
    str     r1, [r0, #0x300]  //*(0xe0100300)=((1<<0) | (7<<4) | (3<<8) | (1<<12) | (3<<16) | (1<<20) | (4<<24) |
(1<<28))

    ldr r1, =CLK_DIV1_VAL
    str     r1, [r0, #0x304]  //*(0xe0100304)=((1<<16) | (1<<12) | (1<<8) | (1<<4))

    ldr r1, =CLK_DIV2_VAL
    str     r1, [r0, #0x308]  //*(0xe0100308)=(1<<0)

    ldr     r1, =APLL_VAL
    str     r1, [r0, #0x0100] //*(0xe0100100)=((1<<31) | (APLL_MDIV<<16) | (APLL_PDIV<<8) |
(APLL_SDIV))

    ldr     r1, =MPLL_VAL
    str     r1, [r0, #0x0108] //*(0xe0100108)=((1<<31) | (MPLL_MDIV<<16) | (MPLL_PDIV<<8) |
(MPLL_SDIV))

    ldr     r1, =EPLL_VAL
    str     r1, [r0, #0x0110] //*(0xe0100110)=((1<<31) | (EPLL_MDIV<<16) | (EPLL_PDIV<<8) |
(EPLL_SDIV))

    ldr     r1, =VPLL_VAL
    str     r1, [r0, #0x0120] //*(0xe0100120)=((1<<31) | (VPLL_MDIV<<16) | (VPLL_PDIV<<8) |
(VPLL_SDIV))

    mov  r1, #0x10000
1:  subs  r1, r1, #1
    bne   1b
```

在 syscfg.inc 中，有如下定义：

```
    .equ APLL_MDIV,          (0x7d)
    .equ APLL_PDIV,          (0x3)
    .equ APLL_SDIV,          (0x1)
    .equ MPLL_MDIV,          (0x29b)
    .equ MPLL_PDIV,          (0xc)
    .equ MPLL_SDIV,          (0x1)
    .equ EPLL_MDIV,          (0x60)
    .equ EPLL_PDIV,          (0x6)
    .equ EPLL_SDIV,          (0x2)
    .equ VPLL_MDIV,          (0x6c)
```

```
    .equ VPLL_PDIV,         (0x6)
    .equ VPLL_SDIV,         (0x3)

    .equ CLK_DIV0_VAL,      ((0<<0) | (4<<4) | (4<<8) | (1<<12) | (3<<16) | (1<<20) | (4<<24) |
(1<<28))
    .equ CLK_DIV1_VAL,      ((1<<16) | (1<<12) | (1<<8) | (1<<4))
    .equ CLK_DIV2_VAL,      (1<<0)
.equ APLL_VAL,              ((1<<31) | (APLL_MDIV<<16) | (APLL_PDIV<<8) |
(APLL_SDIV))
.equ MPLL_VAL,              ((1<<31) | (MPLL_MDIV<<16) | (MPLL_PDIV<<8) |
(MPLL_SDIV))
.equ EPLL_VAL,              ((1<<31) | (EPLL_MDIV<<16) | (EPLL_PDIV<<8) |
(EPLL_SDIV))
.equ VPLL_VAL,              ((1<<31) | (VPLL_MDIV<<16) | (VPLL_PDIV<<8) |
(VPLL_SDIV))
```

APLL 频率输出计算公式：

```
FOUT = MDIV × FIN / (PDIV × 2SDIV-1)
```

这时我们不难计算出，

```
FOUT(APLL)=125*24M/3/1=1GHz;
```

MPLL 频率输出计算公式：

```
FOUT = MDIV × FIN / (PDIV × 2SDIV)
```

我们可以计算出，

```
FOUT(MPLL)=667*24M/12/2=667MHz;
```

EPLL 频率输出计算公式：

```
FOUT = (MDIV+K/65536) × FIN / (PDIV × 2SDIV)
```

这里 K 表示 EPLL_CON1[15:0]，初始化值为 0，我们可以计算出，

```
FOUT(EPLL)=96*24M/6/4=96MHz;
```

VPLL 频率输出计算公式：

```
FOUT = MDIV × FIN / (PDIV × 2SDIV)
```

我们可以计算出，

```
FOUT(VPLL)=108*24M/6/8=54MHz;
```

```
ARMCLK = MOUT_MSYS / (APLL_RATIO + 1);
```

APLL_RATIO 由 CLK_DIV0[2:0]决定，MOUT_MSYS 由 CLK_MUX_STAT0[18:16]决定，

CLK_MUX_STAT0[18:16]=1 时，MOUT_MSYS=SCLKAPLL;

CLK_MUX_STAT0[18:16]=2 时，MOUT_MSYS=SCLKMPLL;

在程序中，我们并没有对 CLK_MUX_STAT0[18:16]进行设置，即它为默认值 1，那么我们可以得出 MOUT_MSYS=SCLKAPLL=1GHz。

在 syscfg.inc 中，CLK_DIV0[2:0]=0，这时我们不难计算出：

```
ARMCLK=MOUT_MSYS/(APLL_RATIO+1)=1GHz
```

我们可以尝试在 syscfg.inc 中将 CLK_DIV0[2:0]设置为 1，更新映像，可以发现，ARMCLK 会被降频到 500MHz。

我们再分析 psys_hclk 和 psys_pclk 是如何得来的。从上面的表可以看出，他们的时钟源为 MOUTpsys，在程序中用变量 psys 表示，图中的二进一出选择器由寄存器

CLK_MUX_STAT0[26:24]决定。

CLK_MUX_STAT0[26:24]=1 时，MOUTpsys=SCLKMPLL；

CLK_MUX_STAT0[26:24]=2 时，MOUTpsys=SCLKA2M；

由于我们并没有对这个寄存器初始化，默认值为 1，那么 MOUTpsys=SCLKMPLL=667MHz。

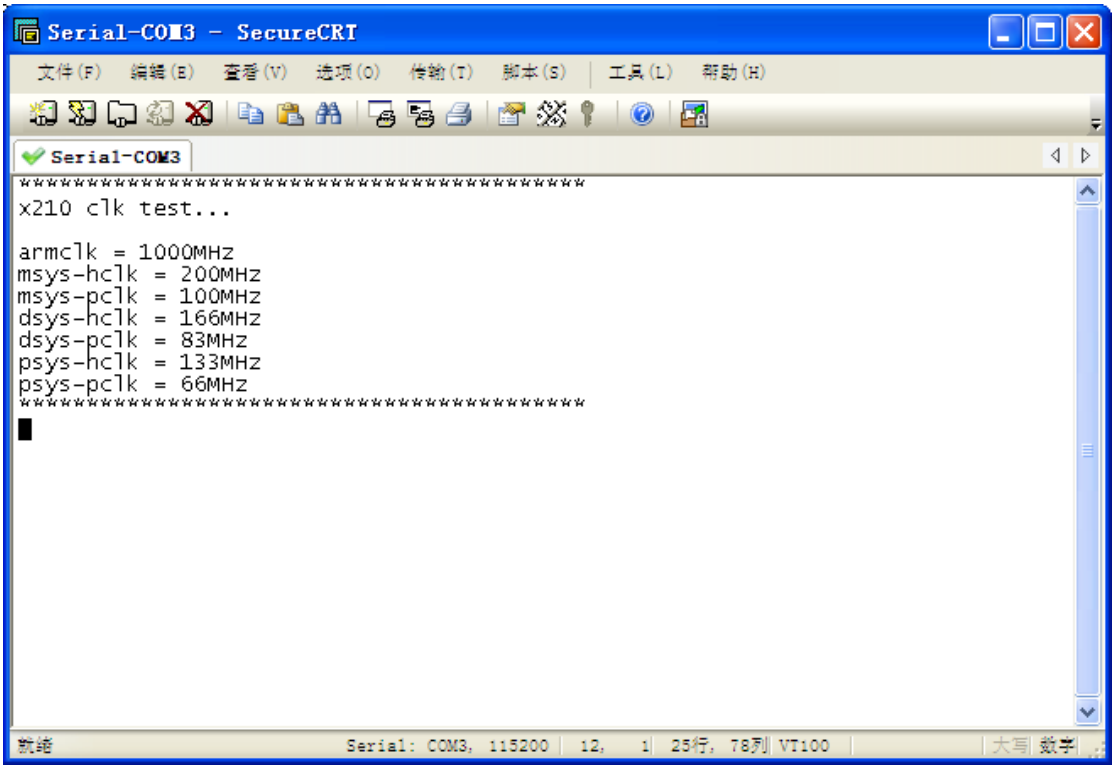
psys_hclk=MOUTpsys/(CLK_DIV0[27:24]+1)=667M/(4+1)=133M

psys_pclk=psys_hclk/(CLK_DIV0[30:28]+1)=133M/(1+1)=66M

以上各频率，和我们打印出来的频率不谋而合，其他频率，读者可以自己分析。

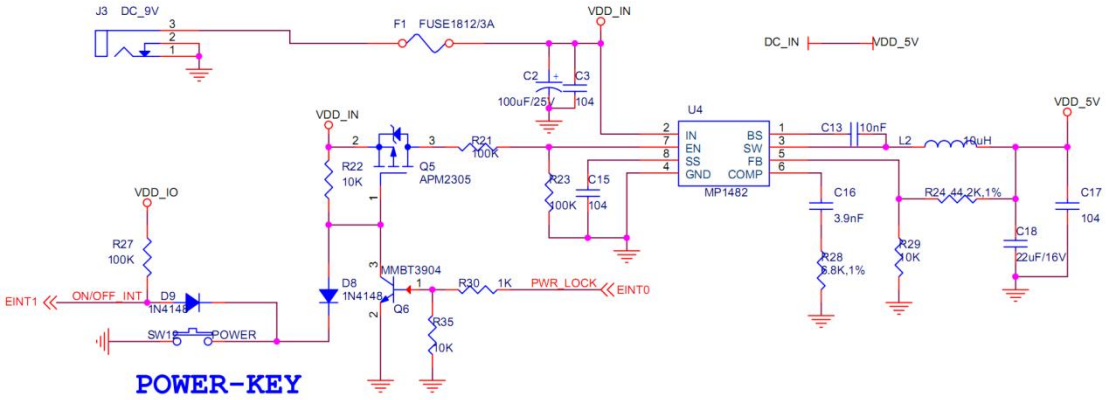
3.12.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键 3 秒，默认情况下，串口会有如下打印：



3.13 x210v3 裸机开发 13-开发板置锁

3.13.1 原理图



市面上绝大多数开发板都是给整个系统直接上电，通过拨码开关实现硬件开关机，这种设计开发板将无法休眠唤醒。x210 开发板独特的软件开关机功能，可实现轻触开关实现软件开关机，休眠唤醒等，让开发板更接近产品。

上述电路图中，EINT0 用于置锁整个开发板的 5V 电源，EINT0 为高时，5V 电源被使能，为低时 5V 电源关闭。在实现休眠唤醒时，配合 EINT1 中断，通过 POWER 键实现中断响应，继而实现休眠唤醒。

EINT0 管脚和 PS_HOLD 管脚复用，当 PS_HOLD_CONTROL 寄存器的第 0 位设置为高时，使能 PS_HOLD 管脚，这时，EINT0 的相关寄存器设置将无效，该管脚将完全由 PS_HOLD_CONTROL 寄存器控制。

PS_HOLD_CONTROL	Bit	Description	Initial State
Reserved	[31:12]	Reserved	0x00005
Reserved	[11:10]	Reserved	0
DIR	[9]	Direction (0: input, 1: output)	1
DATA	[8]	Driving value (0:low, 1:high)	0
Reserved	[7:1]	Reserved	0x00
PS_HOLD_OUT_EN	[0]	XEINT[0] pad is controlled by this register values and values of control registers for XEINT[0] of GPIO chapter is ignored when this field is '1'. (0: disable, 1: enable)	0

我们将 PS_HOLD_CONTROL 的第 8，9 位设置为 1 时，该 IO 口为高，设置为 0 时，为低。在 tester_powerlock 函数中，经 5 秒延时后，执行如下指令：

```
writel(S5PV210_PS_HOLD_CONTROL,  (readl(S5PV210_PS_HOLD_CONTROL)  & ~( 0x00000301 )) | ((0x1<<0) | (0x0<<8) | (0x0<<9)));
```

即将 PS_HOLD_CONTROL 的第 8，9 位清零，这时，EINT0 管脚被拉低，整个开发板的 5V 电源被关闭，开关板断电关机。

在整个 main 函数中，我们并没有对该寄存器做过任何设置，而上面的寄存器表格显示，在芯片的初始化状态，PS_HOLD_CONTROL 的第 0 位是 0，即默认为普通 GPIO 口，复位时电平为 0，那么开发板为什么还能运行起来呢？

在 start.s 中，有如下程序段：

```
ldr    r0, =0xe010e81c
ldr    r1, [r0]
orr    r1, r1, #0x300
orr    r1, r1, #0x1
str    r1, [r0]
```

上面程序将 PS_HOLD_CONTROL 的第 0，8，9 位置高，即将该 IO 设置为 PS_HOLD 的模式的同时将 IO 口拉高，实现 5V 电源置锁。如果在 start.s 中去掉上面语句，那么开发板只能长按住 POWER 键不放，才能维持电平。

3.13.2 源码

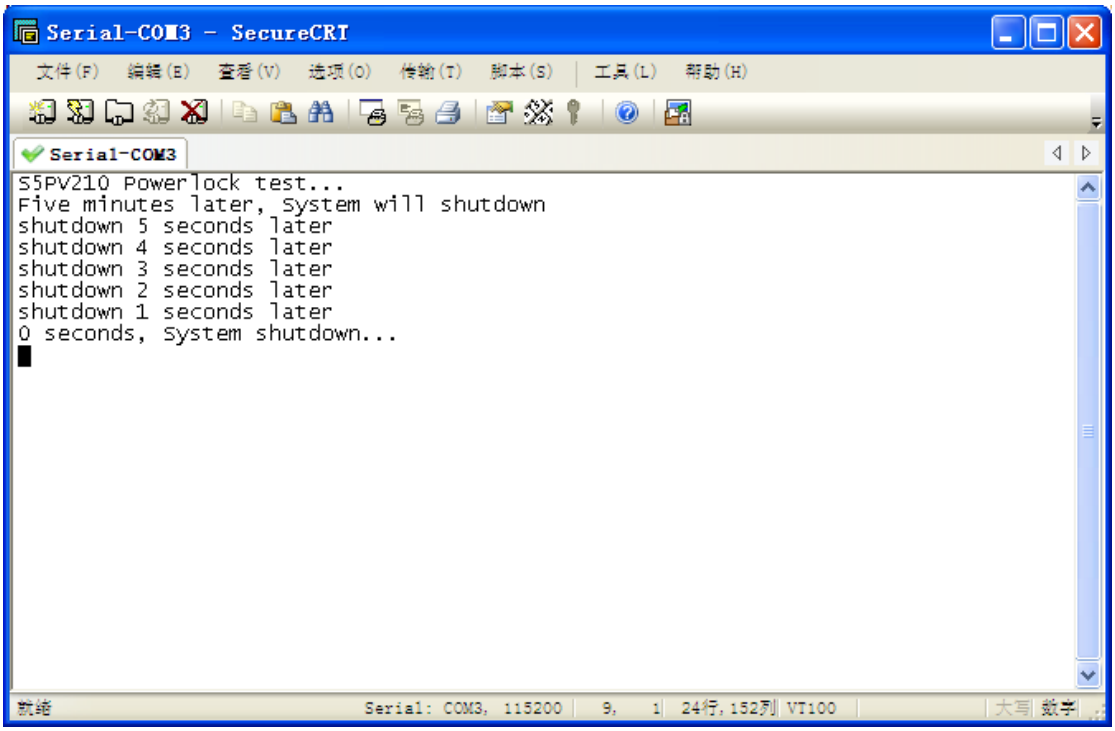
D:\workspace\template-watchdog\source\main.c

D:\workspace\template-watchdog\source\tester-watchdog.c

在主函数 main()中，通过 do_system_initial 初始化串口，通过 tester_powerlock 函数测试置锁 GPIO 口。在 tester_powerlock 函数中，调用串口驱动打印一些信息提示，5 秒后，将 PS_HOLD 管脚拉低断电。

3.13.3 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键开机，串口终端会有如下打印：



5 秒后，开发板断电关机。

3.14 x210v3 裸机开发 14-看门狗实验

3.14.1 源码

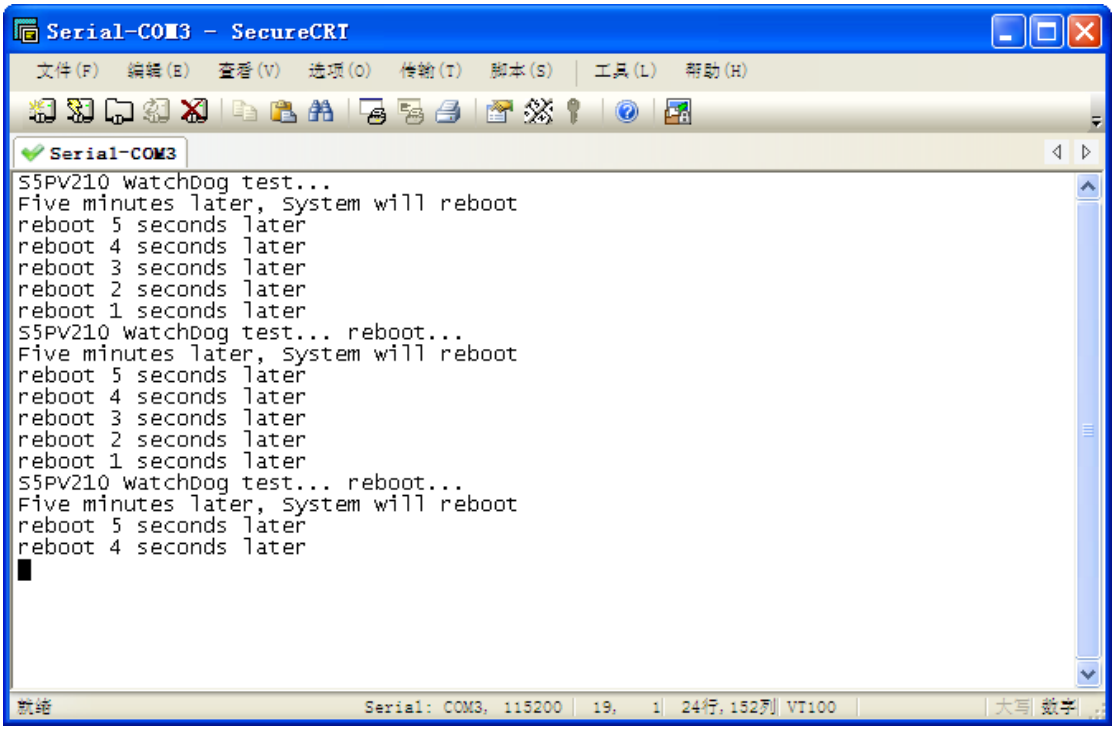
D:\workspace\template-watchdog\source\main.c

D:\workspace\template-watchdog\source\tester-watchdog.c

在 main()主函数中，do_system_initial 函数调用 s5pv210_serial_initial 函数初始化串口，然后进入 tester_watchdog 函数测试看门狗。tester_watchdog 函数中通过串口打印提示，5 秒后打开看门狗，开发板自动复位重启。

3.14.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键开机，串口终端会有如下打印：



3.15 x210v3 裸机开发 15-看门狗定时器实验

3.15.1 源码

D:\workspace\template-watchdog-timer\source\main.c

D:\workspace\template-watchdog-timer\source\tester-watchdog.c

在前一个实验中，当我们设置看门狗之后，系统马上就复位了，事实上，看门狗的真正用途，是设置一个看门狗定时器，在打开开门狗后，当看门狗定时器倒计时时间到之前，手动给看门狗喂狗，定时器又会重新计数。一旦定时器计数到之后，我们仍然没有喂狗，那么看门狗就会触发复位。它的功能就是在机器死机之后，看门狗触发复位信号自动重启。可见，前面的实验并没有达到我们想要的效果。看门狗定时器的定时周期计算公式：

t_watchdog=1/(PCLK / (Prescaler value + 1) / Division_factor)

=(Prescaler value + 1)* Division_factor/PCLK

这里，PCLK= psys_pclk=66MHz，Prescaler value=WTCON[15:8]，Division_factor 由 WTCON[4:3]决定。

看门狗总的定时时间= t_watchdog*WDCNT[15:0]，WDCNT[15:0]表示看门狗计数次数。由于前面实验中，WTCON=0x0021，WTCNT=0x0001，那么我们可以计算出看门狗的定时时间 T 如下(CNT 表示看门狗计数次数)：

T= t_watchdog*CNT

=(Prescaler value + 1)* Division_factor*CNT/PCLK

=(0+1)*16*1/66M

可见，定时时间基本上为 0。也就是说，看门狗一旦打开，系统就会复位。

这里我们将 WTCON 设置为 0xff21，WTCNT 设置为 0xffff，再计算看门狗的定时时间：

T=(255+1)*16*65535/66M
=4.067 秒

本实验测试程序源码如下：

```
int tester_watchdog_timer(int argc, char * argv[])
{
/*
        u64_t armclk,msys_hclk,msys_pclk,dsys_hclk,dsys_pclk,psys_hclk,psys_pclk;
        if(!clk_get_rate("armclk", &armclk))
                return 0;
        if(!clk_get_rate("msys-hclk", &msys_hclk))
                return 0;
        if(!clk_get_rate("msys-pclk", &msys_pclk))
                return 0;
        if(!clk_get_rate("dsys-hclk", &dsys_hclk))
                return 0;
        if(!clk_get_rate("dsys-pclk", &dsys_pclk))
                return 0;
        if(!clk_get_rate("psys-hclk", &psys_hclk))
                return 0;
        if(!clk_get_rate("psys-pclk", &psys_pclk))
                return 0;

        serial_printf(0, "armclk = %d\r\n",armclk/1000000);
        serial_printf(0, "msys-hclk = %d\r\n",msys_hclk/1000000);
        serial_printf(0, "msys-pclk = %d\r\n",msys_pclk/1000000);
        serial_printf(0, "dsys-hclk = %d\r\n",dsys_hclk/1000000);
        serial_printf(0, "dsys-pclk = %d\r\n",dsys_pclk/1000000);
        serial_printf(0, "psys-hclk = %d\r\n",psys_hclk/1000000);
        serial_printf(0, "psys-pclk = %d\r\n",psys_pclk/1000000);
*/
        serial_printf(0, "S5PV210 WatchDog test...\r\n\r\n");

        serial_printf(0, "start watchdog now!\r\n");
        serial_printf(0, "4s later, System will reboot\r\n");
        writel(S5PV210_WTCON, 0x0000);
        writel(S5PV210_WTCNT, 0xffff);
        writel(S5PV210_WTCON, 0xff21);

        serial_printf(0, "reboot 0 seconds\r\n");
        mdelay(1000);
        serial_printf(0, "reboot 1 seconds\r\n");
        mdelay(1000);
        serial_printf(0, "reboot 2 seconds\r\n");
```

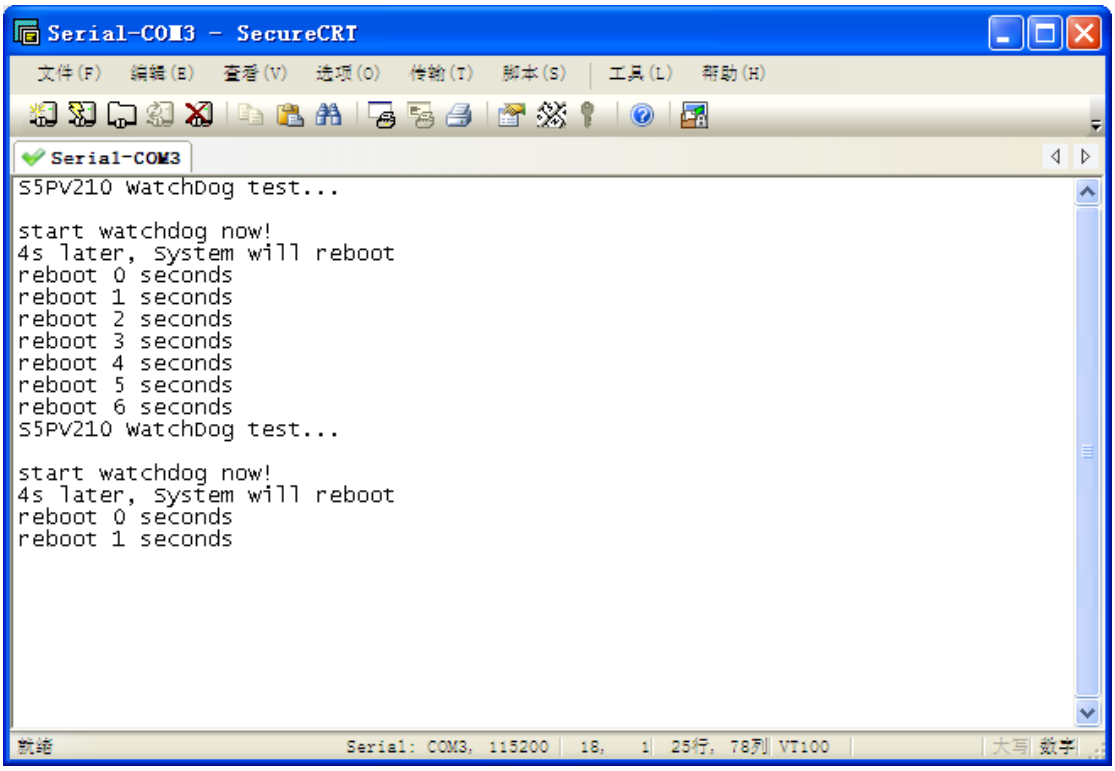
```
    mdelay(1000);
    serial_printf(0, "reboot 3 seconds\r\n");
    writel(S5PV210_WTCNT, 0xffff);
    mdelay(1000);
    serial_printf(0, "reboot 4 seconds\r\n");
    mdelay(1000);
    serial_printf(0, "reboot 5 seconds\r\n");
    mdelay(1000);
    serial_printf(0, "reboot 6 seconds\r\n");
    mdelay(1000);
    serial_printf(0, "reboot 7 seconds\r\n");
    mdelay(1000);
    serial_printf(0, "reboot 8 seconds\r\n");
    mdelay(1000);
    serial_printf(0, "reboot 9 seconds\r\n");
    while(1)
    {
    }

    return 0;
}
```

程序开始就已经打开了看门狗，然后每隔约 1 秒打印一次打印信息，在约 3 秒后，我们对倒计时次数进行了重载，这样，看门狗又会重新计时，因此，约 7 秒后，开发板才会重启。

3.15.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键开机，串口终端会有如下打印：



3.16 x210v3 裸机开发 16-中断检测按键实验

3.16.1 源码

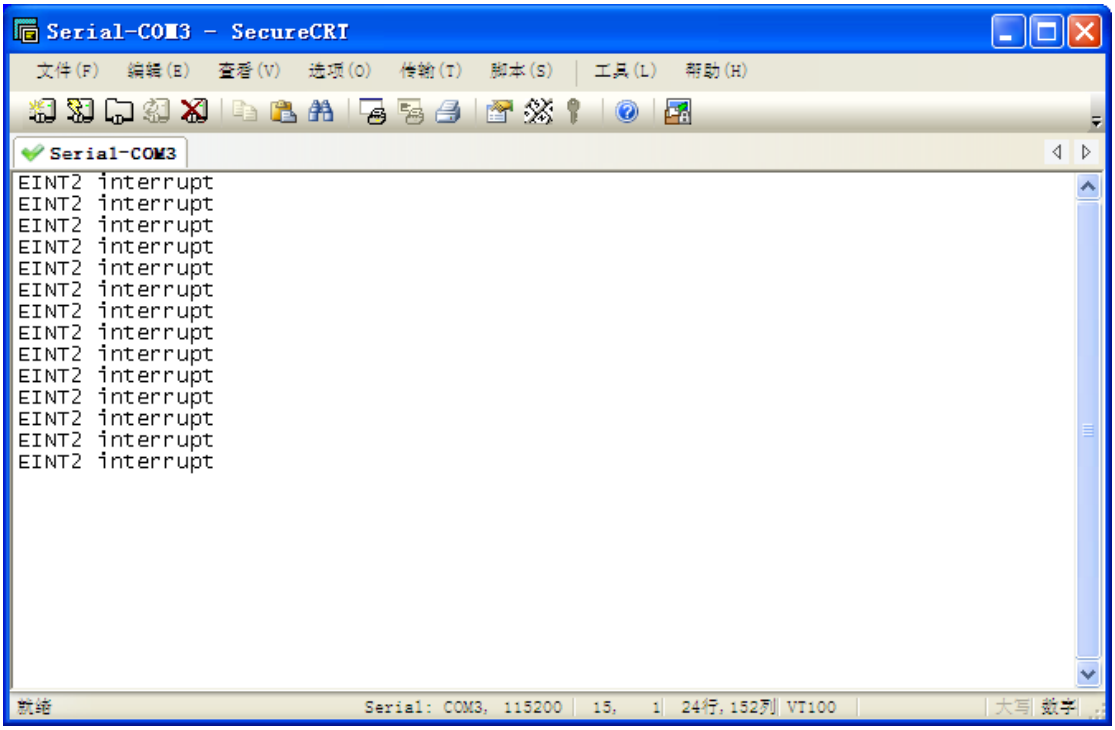
D:\workspace\template-key-interrupt\source\main.c

D:\workspace\template-key-interrupt\source\tester-key-interrupt.c

在 main()主函数中，do_system_initial 函数调用 s5pv210_irq_initial 函数初始化中断，然后进入中断测试函数 tester_key_interrupt，在该函数中将 GPH0_2 设置为中断模式，双边沿触发，同时使能中断。每当 EINT2 上检测到中断信号，中断服务函数 eint2_interrupt_func 得到执行，该函数在串口上打印中断信息，同时检测 LEFT 按键的状态，按下时点亮 D22 这盏 LED 灯，松开时熄灭 D22。之后清除中断标志位，等待响应下一次中断。

3.16.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键开机，串口终端会有如下打印：



同时，LED 灯 D22 会根据 LEFT 按键的状态点亮或熄灭。

3.17 x210v3 裸机开发 17-定时器中断点亮 LED 实验

3.17.1 源码

D:\workspace\template-timer-led-heartbeat\source\main.c

D:\workspace\template-timer-led-heartbeat\source\tester-timer-led-heartbeat.c

D:\workspace\template-timer-led-heartbeat\source\hardware\s5pv210-tick.c

在程序主函数 main()中，do_system_initial 函数调用 s5pv210_tick_initial 函数初始化定时器中断，同时请求定时器服务函数 timer_interrupt，一旦定时器中断产生，timer_interrupt 函数得到运行，led_heartbeat_task 函数得到调用，进而控制 LED。

3.17.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LED 灯 D22 会不断闪烁。

3.18 x210v3 裸机开发 18-裸机实现 shell 指令实验

3.18.1 源码

D:\workspace\template-serial-shell\source\main.c

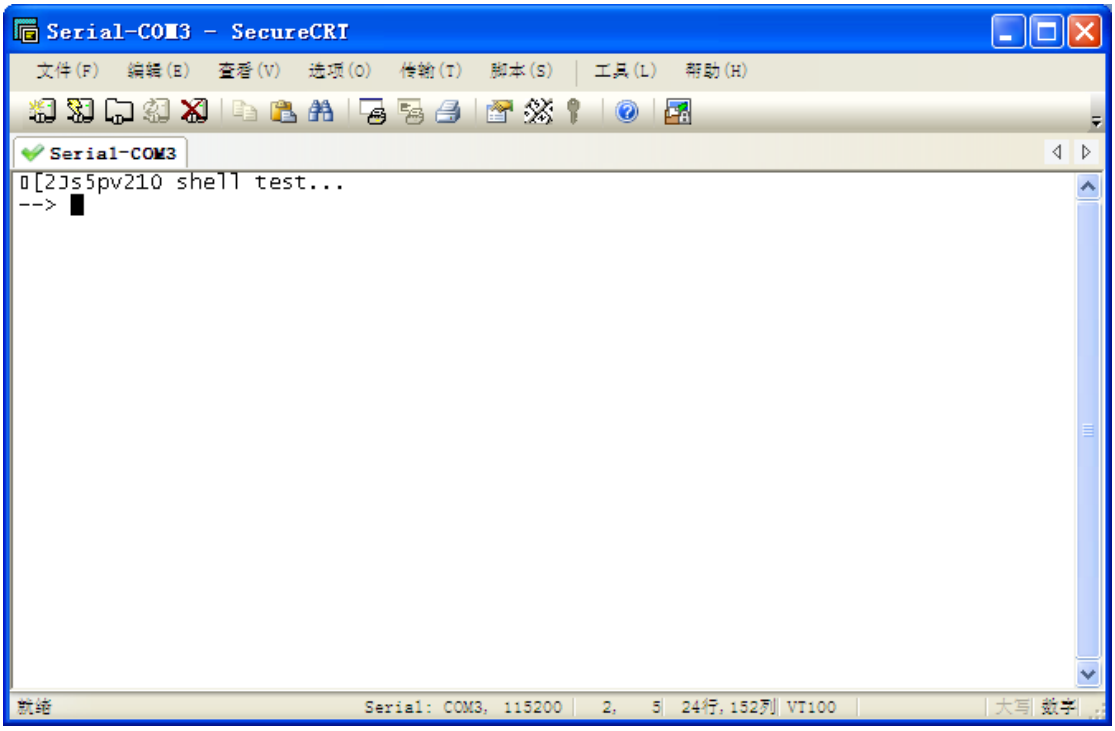
D:\workspace\template-serial-shell\source\tester-serial-shell.c

D:\workspace\template-serial-shell\source\library\readline.c

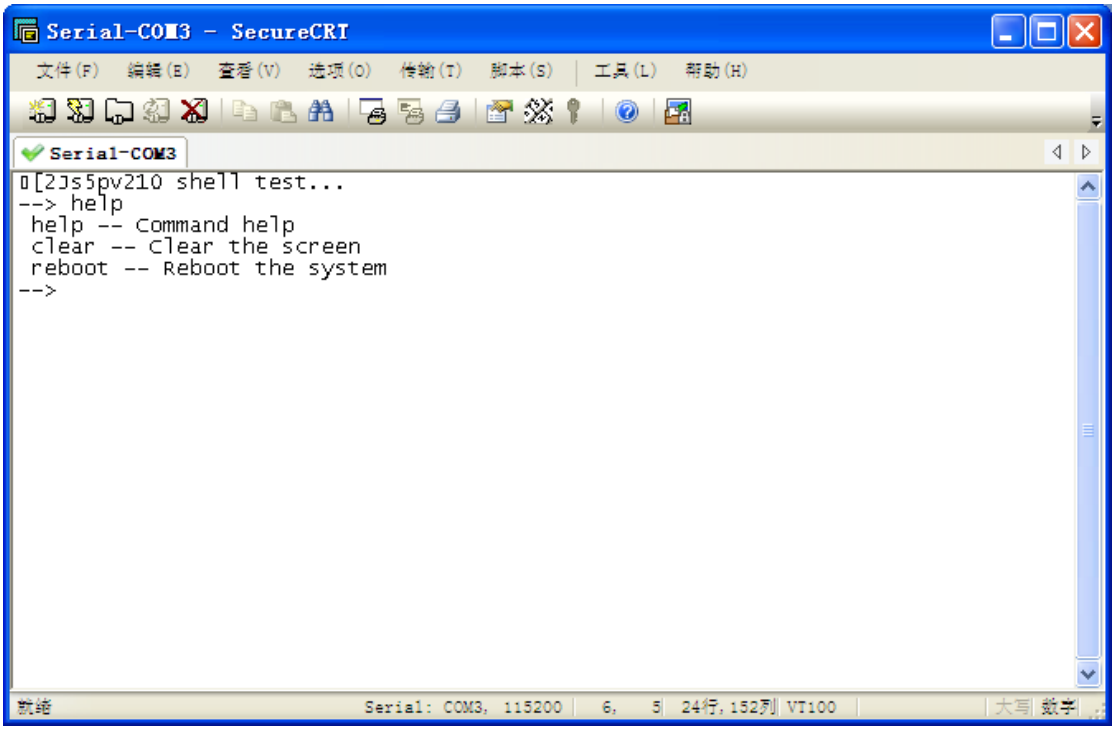
在主函数 main()中，do_system_initial 函数初始化时钟等硬件寄存器，tester_serial_shell 函数用于测试 shell。在该函数中，通过 while(1)死循环，不断调用 readline 函数检测外部输入的命令，一旦检测到有输入命令，则 exec_command 函数得到执行，解析对应的命令并做出相应命令处理。

3.18.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键开机，串口终端会有如下打印：



这时，按下回车键，提示符→不断出现，默认支持三种命令：help,clear 和 reboot。敲入 help，回车，给出帮助的打印信息：



敲入 clear，回车，会清屏；敲入 reboot，回车，开发板重启。

3.19 x210v3 裸机开发 19-串口输入实验

3.19.1 源码

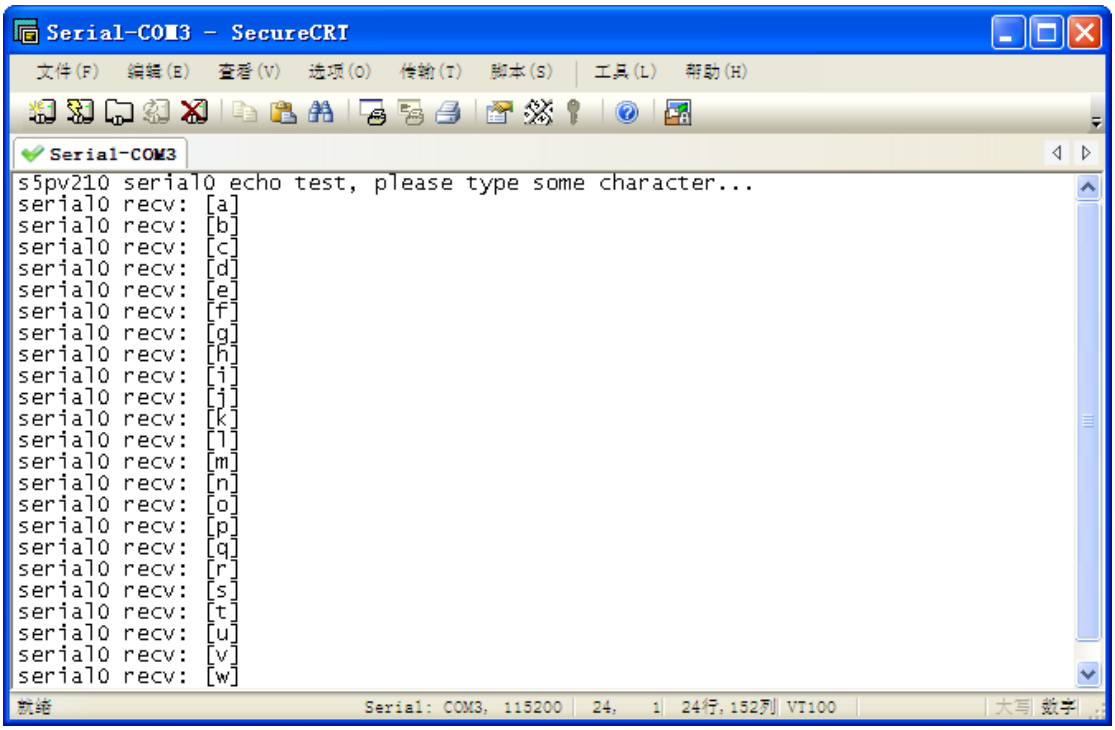
D:\workspace\template-serial-echo\source\main.c

D:\workspace\template-serial-echo\source\tester-serial-echo.c

在主函数 main 中，do_system_initial 函数初始化串口，tester_serial_echo 函数用于从 PC 机读取输入的数据，并通过串口发送出去。默认函数将四组串口全部实现了，因此任意接哪一组串口，全部会有打印提示。

3.19.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，将开发板的串口 0 接到 PC 机的串口，在 PC 机上打开串口监测终端，如超级终端，SecureCRT 等，长按 POWER 键开机，串口终端会有打印信息提示，敲击 PC 机的任意键盘，串口终端会有相应的字符打印提示：



3.20 x210v3 裸机开发 20-LCD 字符显示实验

3.20.1 源码

D:\workspace\template-framebuffer-font\source\main.c

D:\workspace\template-framebuffer-font\source\tester-framebuffer-font.c

在主函数 main()中，do_system_initial 函数调用 s5pv210_fb_initial 函数初始化 LCD 相关寄存器，然后进入 LCD 测试函数 tester_framebuffer_font，源码如下：

```
int tester_framebuffer_font(int argc, char * argv[])
{
	u32_t x, y;
	u32_t fc, bc;
	u32_t count = 0;
```

```
    while(1)
    {
        s5pv210_screen_swap();
        surface_fill(s5pv210_screen_surface(), 0, 0, BLEND_MODE_REPLACE);

        x = randomInt(0, 799 - 16);
        y = randomInt(0, 479 - 8);
        fc = randomInt(0, 0x00ffffff);
        bc = (~fc) & 0x00ffffff;

        lcd_print(x, y, fc, bc, "LCD Print [count = %d]", count++);

        s5pv210_screen_flush();
        mdelay(1000);
    }
    return 0;
}
```

surface_fill 函数将整屏黑屏，x,y,fc 三个变量通过调用 randomInt 函数获取三个随机值，分别用于 LCD 上显示横坐标，纵坐标，字符颜色以及字符背景颜色。在整个 while(1) 死循环中，每隔 1 秒刷新一次，即字符颜色位置等都会更新一次。

3.20.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LCD 上每隔 1 秒会刷新一次，字符颜色，背景颜色以及显示坐标都会相应随机改变。

3.21 x210v3 裸机开发 21-GUI 显示实验

3.21.1 源码

D:\workspace\template-framebuffer-gui\source\main.c
D:\workspace\template-framebuffer-gui\source\tester-framebuffer-gui.c

在主函数 main()中，do_system_initial 调用 s5pv210_fb_initial 函数初始化 LCD 相关寄存器，然后进入 GUI 测试函数 tester_framebuffer_gui。测试函数同样用到了随机数生成函数 randomInt，不断的在 LCD 上画随机圆，椭圆，矩形，线等。

3.21.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LCD 上每隔 1.2 秒会刷新一次，LCD 上会随机绘制圆，椭圆，矩形边框等等。

3.22 x210v3 裸机开发 22-miniGAME 实验

3.22.1 源码

D:\workspace\template-framebuffer-minigame\source\main.c
D:\workspace\template-framebuffer-minigame\source\tester-framebuffer-minigame.c

在主函数 main()中，do_system_initial 调用 s5pv210_fb_initial 函数初始化

LCD 相关寄存器，然后进入迷你游戏测试函数 tester_framebuffer_minigame，该函数再次调用 minigame 函数，这里才是程序的关键所在。同样，程序中调用了随机数生成函数 randomFloat，faces[i].x，faces[i].y，faces[i].xvel 以及 faces[i].yvel 分别对应第 i 个小球的 x 坐标，y 坐标，x 轴移动速度和 y 轴移动速度。surface_map_color 函数用于设置背景颜色，被赋值到变量 c 中，在 while(1)死循环中，首先通过 surface_fill 函数将整屏全部刷成咖啡色，然后通过 surface_blit 函数绘制 NUM_HAPPY_FACES 个小球，同时 NUM_HAPPY_FACES 个小球会不断移动，移动的速度由前面产生的随机数 faces[i].xvel，faces[i].yvel 以及变量 MILLESECONDS_PER_FRAME 决定。这样，整个 LCD 上的图像，就会显示出多个小球不断的来回碰撞的界面出来。这里只是 android,ios 上很多游戏中最简单的一个动作，读者可以自己发掘更多更复杂的动作效果。

3.22.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LCD 上会有 10 个九鼎创展的 LOGO 图片来回随机的滚动。实验效果如下：

3.23 x210v3 裸机开发 23-随机矩形框显示实验

3.23.1 源码

D:\workspace\template-framebuffer-random-rect\source\main.c
D:\workspace\template-framebuffer-random-rect\source\tester-framebuffer-random-rect.c

在主函数 main()中，do_system_initial 函数调用 s5pv210_fb_initial 函数初始化 LCD 相关寄存器，然后进入测试函数 tester_framebuffer_random_rect 随机显示矩形框。在该函数中有五个变量调用了随机数生成函数 randomInt，分别对应矩形的坐标，宽高和填充的颜色。然后调用 surface_fill 函数绘制矩形框。

3.23.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LCD 上会有无数个背景颜色随机，大小也随机的矩形框不停刷新显示。整体一看显得非常凌乱无章，真正达到了随机的效

果。

3.24 x210v3 裸机开发 24-长方体旋转实验

3.24.1 源码

D:\workspace\template-framebuffer-tingl-cube\source\main.c

D:\workspace\template-framebuffer-tingl-cube\source\tester-framebuffer-tingl-cube.c

本实验不再像上次的实验那样，显得杂乱无章，这里显示的是一个长方体，不乱的延 X 轴，Y 轴，Z 轴旋转。关键函数在 idle 中，读者可以自行分析。

3.24.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LCD 上会有一个长方体不断的延 X 轴，Y 轴，Z 轴旋转。效果如下：

3.25 x210v3 裸机开发 25-轴承旋转实验

3.25.1 源码

D:\workspace\template-framebuffer-tingl-gear\source\main.c

D:\workspace\template-framebuffer-tingl-gear\source\tester-framebuffer-tingl-gear.c

本实验显示旋转的轴承，不断旋转的同时，界面也会不断旋转，非常有立体感。这里不讲述源码实现过程，读者可以自行分析。

3.25.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LCD 上会有三个立体的轴承不断旋转，界面也会不断变化。效果如下：

3.26 x210v3 裸机开发 26-二维码扫描实验

3.26.1 源码

D:\workspace\template-framebuffer-qrencode\source\main.c
D:\workspace\template-framebuffer-qrencode\source\tester-framebuffer-qrencode.c

本实验需要借助于能够识别二维码的手机。在主函数 main 中调用 tester_framebuffer_qrencode 函数，在 LCD 上绘制出一个二维码，每隔 5 秒更新一次。当我们使用手机扫描该二维码时，可以获取二维码的信息，即 while(1)死循环中通过 sprintf 语句输出的一长串字符串。

3.26.2 实验现象

将 SD 卡插到 x210v3 开发板的 SD2 通道，长按 POWER 键开机，LCD 的左上脚会显示一幅二维码图像。找到一个支持二维码识别的手机，安装微信，进入扫一扫功能，对着开发板上的二维码扫描，扫描完毕后，手机会从该二维码读取相应的信息，并显示在手机上。效果如下：


LCD070 www.9tripod.com
4GB

第4章 其他产品介绍

4.1 核心板系列

4.1.1 6410 核心板

X6410CV10

4.1.2 210 核心板

X210CV10
X210CV3
G210CV10
I210CV10

4.2 开发板系列

4.2.1 6410 开发板

x6410 开发板

4.2.2 210 开发板

x210v3 开发板
g210 开发板
i210 开发板

说明：产品详细规格，以及更多其他产品请关注九鼎创展官方网站和论坛。