

# Linux 平台下 Mini210s 裸机程序开发指南

版本：2012-12-10

(本手册正在不断更新中，建议您到网站下载最新版本)



Copyright@2012



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 版 权 声 明

本手册版权归属广州友善之臂计算机科技有限公司(以下简称“友善之臂”)所有，并保留一切权力。未经友善之臂同意(书面形式)，任何单位及个人不得擅自摘录本手册部分或全部，违者我们将追究其法律责任。

敬告：

在售开发板的手册会经常更新，请在 <http://www.arm9home.net> 网站查看最近更新，并下载最新手册，不再另行通知。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

更新说明：

日期	说明
2012-12-10	本手册第一次发布，任何问题请在论坛中跟贴反馈。

---

地址：广州市天河区龙口西路龙苑大厦A1栋1705 网址：<http://www.arm9.net>  
电话：+86-20-85201025(售前、售后咨询) 技术支持(Tel): 13719442657 传真：+86-20-85261505  
E-Mail: capbily@163.com(商务或项目合作) dev\_friendlyarm@163.com (技术支持)



第一章 简介	7
第一节 简介	7
第二节 关于开发环境	7
第三节 文档涉及的裸机程序	7
第二章 汇编点亮LED	9
第一节 查阅原理图	9
第二节 程序相关讲解	9
第三节 编译代码和烧写运行	11
第四节 实验现象	12
第三章 关于S5PV210 的启动过程	13
第一节 初步认识IROM和IRAM	13
第二节 完整的启动序列	13
第四章 关闭看门狗和调用C函数	15
第一节 看门狗背景知识	15
第二节 程序相关讲解	15
第三节 编译代码和烧写运行	15
第四节 实验现象	16
第五章 设置栈和C语言点亮LED	17
第一节 为什么调用C函数要设置栈	17
第二节 程序相关讲解	18
第三节 编译代码和烧写运行	20
第四节 实验现象	20
第六章 控制icache	21
第一节 什么是cache	21
第二节 程序相关讲解	21
第三节 编译代码和烧写运行	21
第四节 实验现象	22
第七章 重定位代码到IRAM+0x4000	23
第一节 重定位	23
第二节 程序相关讲解	23
第三节 编译代码和烧写运行	25
第四节 实验现象	25
第八章 重定位代码到DRAM	27
第一节 关于DRAM	27
第二节 程序相关讲解	28
第三节 编译代码和烧写运行	33
第四节 实验现象	34
第九章 使用MiniTools烧写裸机程序	35
第一节 什么是MiniTools	35
第二节 如何使用MiniTools烧写裸机程序	35
第三节 程序相关讲解	37



第四节	实验现象 .....	37
第十章	控制蜂鸣器 .....	38
第一节	查阅原理图 .....	38
第二节	程序相关讲解 .....	38
第三节	编译代码和烧写运行 .....	39
第四节	实验现象 .....	40
第十一章	查询方式检测按键 .....	41
第一节	查看原理图 .....	41
第二节	程序相关讲解 .....	41
第三节	编译代码和烧写运行 .....	42
第四节	实验现象 .....	43
第十二章	初始化系统时钟 .....	44
第一节	S5PV210 时钟体系 .....	44
第二节	程序相关讲解 .....	46
第三节	编译代码和烧写运行 .....	52
第四节	实验现象 .....	53
第十三章	串口设置之输入输出字符 .....	54
第一节	S5PV210 UART相关说明 .....	54
第二节	程序相关讲解 .....	55
第三节	编译代码和烧写运行 .....	61
第四节	实验现象 .....	62
第十四章	移植printf和scanf功能 .....	64
第一节	移植的途径 .....	64
第二节	移植步骤 .....	64
第三节	程序相关讲解 .....	64
第四节	编译代码和烧写运行 .....	66
第五节	实验现象 .....	67
第十五章	NAND Flash的读写擦除 .....	69
第一节	关于NAND Flash .....	69
第二节	程序相关讲解 .....	69
第三节	编译代码和烧写运行 .....	79
第四节	实验现象 .....	80
第十六章	S5PV210 中断体系 .....	81
第一节	关于S5PV210 的中断体系结构 .....	81
第二节	程序相关讲解 .....	81
第三节	编译代码和烧写运行 .....	86
第四节	实验现象 .....	87
第十七章	PWM定时器 .....	89
第一节	S5PV210 的PWM定时器 .....	89
第二节	程序相关讲解 .....	89
第三节	编译代码和烧写运行 .....	93



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

第四节	实验现象 .....	93
第十八章	看门狗定时和复位 .....	95
第一节	S5PV210 的看门狗定时器 .....	95
第二节	程序相关讲解 .....	95
第三节	编译代码和烧写运行 .....	98
第四节	实验现象 .....	98
第十九章	RTC读写时间 .....	100
第一节	S5PV210 的RTC .....	100
第二节	程序相关讲解 .....	100
第三节	编译代码和烧写运行 .....	103
第四节	实验现象 .....	104
第二十章	LCD描点画线 .....	105
第一节	S5PV210 LCD控制器 .....	105
第二节	程序相关讲解 .....	105
第三节	编译代码和烧写运行 .....	116
第四节	实验现象 .....	117
第二十一章	测试ADC转换 .....	118
第一节	S5PV210 的ADC .....	118
第二节	程序相关讲解 .....	118
第三节	编译代码和烧写运行 .....	121
第四节	实验现象 .....	121
第二十二章	增加命令功能 .....	123
第一节	关于命令功能 .....	123
第二节	程序详细讲解 .....	123
第三节	编译代码和烧写运行 .....	124
第四节	实验现象 .....	125
第二十三章	WM8960 音频播放 .....	127
第一节	音频播放原理 .....	127
第二节	程序详细讲解 .....	127
第三节	编译代码和烧写运行 .....	131
第四节	实验现象 .....	132
第二十四章	LCD显示字符和图片 .....	133
第一节	LCD显示字符和图片 .....	133
第二节	程序详细讲解 .....	133
第三节	编译代码和烧写运行 .....	133
第四节	实验现象 .....	134



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

# 第一章 简介

## 第一节 简介

随着移动互联网的飞速发展，国内安卓智能手机和平台的电脑也得以迅速普及，性能优越的 Cortex-A8 CPU 主控无疑成为这些智能终端的入门之选，从国产全志 A10 到三星 S5PV210，无不采用了 Cortex A8 架构；当然，我们现在已经在拼“核”的浪涛之中竞逐，从双核小米到四核魅族 MX2，纷纷“你方唱罢我登场”，但是，从嵌入式开发角度而言，这些 CPU 目前都是采用了 ARMV7 指令集；鉴于较低的性价比，采用 ARM9, ARM11 等架构 CPU 的产品已经渐渐走下了舞台。

对于很多嵌入式开发者和爱好者，特别是初学者，如何从底层开始了解和学习 Cortex A8，绝非是一件容易的事！

为此，友善之臂的工程师，花了很多时间和心血，基于 Mini210S 开发板编写了这份教程，以帮助广大嵌入式爱好者更深入了解 S5PV210 的详细启动过程，同时让更多初学者能从裸机程序开始学习 Cortex A8。鉴于每个人的认知水平不同，以及我们平时的开发任务比较紧张，我们并不提供关于该教程的任何技术支持。如果你对本教程的内容有任何疑问，可以到 arm9 之家论坛 (<http://www.arm9home.net>) 反馈，并和其他网友交流讨论。

需要说明的是，本教程也适用于友善之臂出品的 Tiny210/Tiny210V2 等其他型号基于 S5PV210 的开发板平台。

友善之臂将对本教程作不定期的维护和补充，请时常留意论坛的更新信息，不再另行通知。

友善之臂 (<http://www.arm9.net>) 保留本教程的一切解释权。

## 第二节 关于开发环境

1) 学习前提：学习过简单的 C 语言和 ARM 汇编语言

2) 开发平台：windows xp + 虚拟机 fedora15，使用 source insight 编写代码

3) 交叉编译器：arm-linux-gcc-4.5.1

注：交叉编译器的安装方法可参考开发板相关的用户手册。

4) 配套开发板：所有程序均在 Mini210S 上成功测试运行过，对于友善之臂其他 210 开发板，大部分代码仍然适用。

## 第三节 文档涉及的裸机程序

首先前面的 2~8 章会从最基础的点亮 LED 讲起，采用从 sd 卡加载的方式，逐步讲解 S5PV210 的启动过程以及如何重定位代码，让用户了解 S5PV210 上电后运行的每一个步骤。有了这些必要的知识后，后面的章节我们就可以采用 USB 下载工具 MiniTools 直接将裸机程序下载到 DRAM，从而编写功能更复杂的裸机程序。用户可以通过查看书签可以了解本文档所涉及的硬件模块，本文档



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

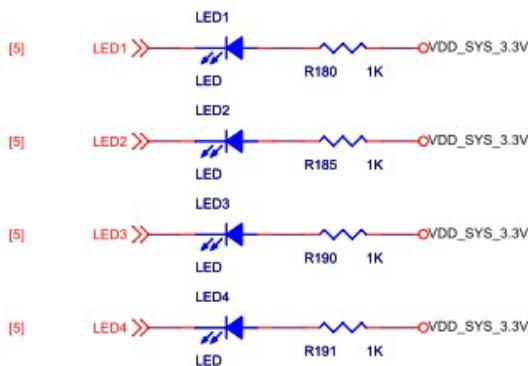
一共涉及 23 个裸机程序，并且以后将不断丰富出更多的裸机程序，也请众网友发挥各自的才智，在本文档的基础上写出更多简单实用的裸机程序并反馈给我们。

## 第二章 汇编点亮 LED

### 第一节 查阅原理图

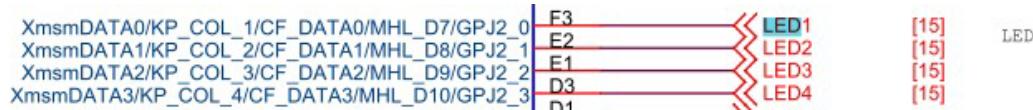
Mini210S 板上提供了 4 个可编程用户 LED，原理图如下：

**LED x 4**



**LED 原理图**

在原理图中搜索引脚 “LED1”，可得：



**LED 引脚图**

可见，LED1, 2, 3, 4 分别使用的 CPU 端口资源为 GPJ2\_0, 1, 2, 3

### 第二节 程序相关讲解

完整代码见目录 1. led\_s。 (注：为方便用户阅读，本教程涉及的所有代码本身均含相当丰富的注释)。

#### 1. start.S

由原理图可知，点亮 Mini210S 的 4 个 LED 需如下 2 个步骤：

第一步：设置寄存器 GPJ2CON，使 GPJ2\_0/1/2/3 四个引脚为输出功能；

第二步：往寄存器 GPJ2DAT 写 0，使 GPJ2\_0/1/2/3 四个引脚输出低电平，4 个 LED 会亮；相反，往寄存器 GPJ2DAT 写 1，使 GPJ2\_0/1/2/3 四个引脚输出高电平，4 个 LED 会灭；

以上两个步骤即为 start.S 中的核心内容，start.S 里面涉及的汇编指令请自行学习 GNU 汇编指令集，这里不再进行赘述。



## 2. Makefile

代码如下：

```
led.bin: start.o
    arm-linux-ld -Ttext 0x0 -o led.elf $^
    arm-linux-objcopy -O binary led.elf led.bin
    arm-linux-objdump -D led.elf > led_elf.dis
    gcc mkv210_image.c -o mkmini210
    ./mkmini210 led.bin 210.bin

%.o : %.S
    arm-linux-gcc -o $@ $< -c
%.o : %.c
    arm-linux-gcc -o $@ $< -c
clean:
    rm *.o *.*elf *.bin *.dis -f
```

关于 Makefile 网上的相关资料相当多，这里推荐《跟我一起写 makefile》（光盘中有），以方便用户后续的学习和查阅，下面我们只是进行粗略地讲解 Makefile。当用户在 Makefile 所在目录下执行 make 命令时，系统会进行如下操作：

**第一步** 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；

**第二步** 执行 arm-linux-ld -Ttext 0x0 -o led.elf \$^ 将所有 .o 文件链接成 elf 文件，-Ttext 0x0 表示程序的运行地址是 0x0，由于目前我们编写的代码是位置无关码，所以程序能在任何一个地址上运行；

**第三步** 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；

**第四步** 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；

**第五步** mkmini210 处理 led.bin 文件，mkmini210 由 mkv210\_image.c 编译而来，具体解释请看 mkv210\_image.c 相关讲解；

## 3. mkv210\_image.c

从三星提供的 S5PV210 文档《S5PV210\_iROM\_ApplicationNote\_Preliminary\_20091126.pdf》以及芯片手册《S5PV210\_UM\_REV1.1.pdf》得知，S5PV210 启动时，会先运行内部 IROM 中的固化代码进行一些必要的初始化，执行完后硬件上会自动读取 NAND Flash 或 sd 卡等启动设备的前 16K 的数据到 IRAM 中，这 16K 数据中的前 16byte 中保存了一个叫校验和的值，拷贝数据时 S5PV210 会统计出待运行的 bin 文件中含 ‘1’ 的个数，然后和校验和做比较，如果相等则继续运行程序，否则停止运行。所以所有在 S5PV210 上运行的 bin 文件都必须具有一个 16byte 的头部，该头部



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

中需包含校验和信息，mkv210\_image.c 由 arm9 论坛上的网友提供，它的作用就是用来给原始的 bin 文件添加头部。

**mkv210\_image.c 的核心工作如下：**

**第一步 分配 16k 的 buffer；**

**第二步 将 led.bin 读到 buffer 的第 16byte 开始的地方；**

**第三步 计算校验和，并将校验和保存在 buffer 第 8~11byte 中；**

**第四步 将 16k 的 buffer 拷贝到 210.bin 中；**

16byte 头部	led.bin
-----------	---------

16k 的 buffer(即 210.bin)

**校验和统计的关键代码如下：**

```
a = Buf + SPL_HEADER_SIZE;  
for(i = 0, checksum = 0; i < IMG_SIZE - SPL_HEADER_SIZE; i++)  
    checksum += (0x000000FF) & *a++;
```

代码 mkv210\_image.c 已经包含相当丰富的注释，很容易就可以读懂，用户可以自行阅读。

### 第三节 编译代码和烧写运行

**将 sd 卡插入 PC，在 Fedora 终端执行如下命令：**

```
# cd 1.led_s  
# make  
# chmod 777 write2sd  
# ./write2sd
```

执行 make 后会生成 210.bin 文件，执行 ./write2sd 后 210.bin 文件会被烧写到 sd 卡的扇区 1 中，sd 卡的起始扇区为 0，一个扇区的大小为 512byte，sd 启动时，IROM 里的固化代码是从扇区 1 开始拷贝代码的。

**write2sd 是一个脚本文件，内容如下：**

```
#!/bin/sh  
sudo dd iflag=dsync oflag=dsync if=210.bin of=/dev/sdb seek=1  
dd 是一个读写命令，if 是输入，of 是输出，seek 表示从扇区 1 开始读写。
```

**注：sdb 为本文档编写时 sd 卡的设备节点，请根据自己的实际情况进行修改，后面将不再提及该注意事项。**



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第四节 实验现象

将 sd 卡插入 Mini210S 中，选择 sd 卡启动，然后上电，可以看到以下现象：

LED 正常闪烁，这说明我们的第一个程序汇编点亮所有 LED 已经成功。这里留下了一个疑问，我们知道 CPU 刚启动时，由于看门狗的复位功能，CPU 会不断的进行复位，但是在代码 1. led\_s 中我们并没有手动关闭看门狗，为什么程序却也能正常运行，对于这个问题，我们将在第二章中进行解释。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第三章 关于 S5PV210 的启动过程

### 第一节 初步认识 IROM 和 IRAM

S5PV210 含有一个内 64K 的 IROM 和 96K 的 IRAM，系统启动时主要依靠它们，IROM 和 IRAM 所处的存储空间见下图：

Address	Size	Description	Note
0x0000_0000	0xFFFF_FFFF	512MB	Boot area Mirrored region depending on the boot mode.
0x2000_0000	0x3FFF_FFFF	512MB	DRAM 0
0x4000_0000	0x7FFF_FFFF	1024MB	DRAM 1
0x8000_0000	0x87FF_FFFF	128MB	SROM Bank 0
0x8800_0000	0x8FFF_FFFF	128MB	SROM Bank 1
0x9000_0000	0x97FF_FFFF	128MB	SROM Bank 2
0x9800_0000	0x9FFF_FFFF	128MB	SROM Bank 3
0xA000_0000	0xA7FF_FFFF	128MB	SROM Bank 4
0xA800_0000	0xAFEE_FFFF	128MB	SROM Bank 5
0xB000_0000	0xBFFF_FFFF	256MB	OneNAND/NAND Controller and SFR
0xC000_0000	0xCFFF_FFFF	256MB	MP3_SRAM output buffer
0xD000_0000	0xD000_FFFF	64KB	IROM
0xD001_0000	0xD001_FFFF	64KB	Reserved
0xD002_0000	0xD003_7FFF	96KB	IRAM
0xD800_0000	0xDFFF_FFFF	128MB	DMZ ROM
0xE000_0000	0xFFFF_FFFF	512MB	SFR region

### 第二节 完整的启动序列

系统刚启动时，会运行 IROM 中的固化代码，进行一些通用的初始化，具体步骤包括：

第一步 关闭看门狗；

第二步 初始化 icache；

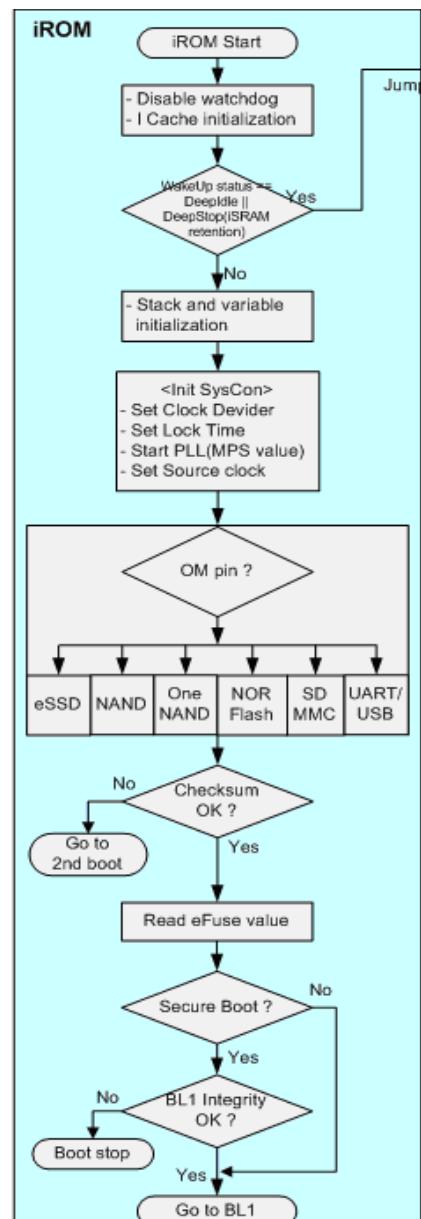
第三步 初始化堆栈；

第四步 设置时钟；

第五步 判断启动设备(nand/sd/onenand 等)，检查校验和，然后从启动设备中拷贝前 16K 的代码到 IRAM 的 **0xD0020000** 处；

第六步 若是安全模式启动，则进行完整性检查；

第七步 跳转到 IRAM 的 **0xD0020010** 地址上继续运行；



iROM 启动序列

在第一章汇编点亮 LED 中，我们的程序 210.bin 最终就是被拷贝到 IRAM 的起始地址 0xD0020000 处，因为 bin 文件中包含了 16byte 的头部，代码的真实起始地址是 0xD0020010，所以上述启动序列的第七步是跳转到 IRAM 的 0xD0020010 地址上继续运行。同时，上述启动序列中，iROM 里的固化代码已经关闭了看门狗，这也就解释了第一章中为什么无需手动关闭看门狗程序也能正常运行。第三章中我们将手动关闭看门狗，并且通过调用 C 函数来测试 iROM 中的固化代码是否已经设置了栈。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第四章 关闭看门狗和调用 C 函数

### 第一节 看门狗背景知识

看门狗的作用在于监控 CPU 的运行，保证在诸如噪音和系统错误等这样的故障干扰情况下能尽快恢复正常工作。看门狗与 PWM 定时器都能实现定时的功能，两者的区别在于看门狗能发出复位信号而 PWM 定时器不能，后面将会有具体的章节讲解看门狗的定时和复位功能，这里我们只是简单的将其关闭。

### 第二节 程序相关讲解

完整代码见目录 2. led\_s\_wtd。

#### 1. start.S

与代码 1. led\_s\_wtd 相比，在代码 2. led\_s\_wtd 中，start.S 多了两点不一样的地方：

- 1) 手动关闭了看门狗，只需往寄存器 WTCON 写入 0 即可；
- 2) 调用了 C 函数实现延时的功能，以测试 IROM 中的固化代码是否设置了栈；

#### 2. delay.c

内含一个普通的 C 语言延时函数，代码如下：

```
void delay(int r0)
{
    volatile int count = r0;
    while (count--);
}
```

汇编调用 C 函数时，当参数个数不超过 4 个，使用 r0~r3 这 4 个寄存器来传递参数；如果参数个数超过 4 个，剩余的参数通过栈来传递，delay() 只有 1 个参数，所以用 r0 来传递。另外，volatile 是为了避免编译器自动帮我们优化掉这段代码造成无法延时。Makefile 和 write2sd 与上一个程序并没有差异，以后若代码与前一个目录的代码相比没有发生变化则将不再赘述。

### 第三节 编译代码和烧写运行

将 sd 卡插入 PC，在 Fedora 终端执行如下命令：

```
# cd 2. led_s_wtd
# make
# chmod 777 write2sd
# ./write2sd
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

执行 make 后会生成 210.bin 文件，执行 ./write2sd 后 210.bin 文件会被烧写到 sd 卡的扇区 1 中。

## 第四节 实验现象

将 sd 卡插入 Mini210S 中，选择 sd 卡启动，然后上电，可以看到以下现象：

LED 正常闪烁，说明手动关闭看门狗成功，同时证明 IROM 中的固化代码已经设置了栈。为什么程序成功调用 C 函数就说明了 IROM 的固化代码设置了栈呢？是因为汇编中调用 C 函数时，参数的传递、现场的保存和恢复、临时变量的保存等都需要使用到栈，在 delay() 函数中变量 count 就是临时变量，程序运行成功说明栈已经设置好了，更详细的解释将会留到下一章。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第五章 设置栈和 C 语言点亮 LED

### 第一节 为什么调用 C 函数要设置栈

#### 1. 栈的整体作用

- 1) 保存现场;
- 2) 传递参数:汇编代码调用 C 函数时, 需传递参数;
- 3) 保存临时变量:包括函数的非静态局部变量以及编译器自动生成的其他临时变量;

#### 2. 详细解释

##### 1) 保存现场

现场, 意思就相当于案发现场, 总有一些现场的情况, 要记录下来的, 否则被别人破坏掉之后, 你就无法恢复现场了。而此处说的现场, 就是指 CPU 运行的时候, 用到了一些寄存器, 比如 r0, r1 等等, 对于这些寄存器的值, 如果你不保存而直接跳转到子函数中去执行, 那么很可能就被其破坏了, 因为其函数执行也要用到这些寄存器。因此, 在函数调用之前, 应该将这些寄存器等现场, 暂时保持起来(入栈 push), 等调用函数执行完毕返回后(出栈 pop), 再恢复现场。这样 CPU 就可以正确的继续执行了。

保存寄存器的值, 一般用的是 push 指令, 将对应的某些寄存器的值, 一个个放到栈中, 把对应的值压入到栈里面, 即所谓的压栈。然后待被调用的子函数执行完毕的时候, 再调用 pop, 把栈中的一个个的值, 赋值给对应的那些你刚开始压栈时用到的寄存器, 把对应的值从栈中弹出去, 即所谓的出栈。

其中保存的寄存器中, 也包括 lr 的值 (因为用 bl 指令进行跳转的话, 那么之前的 PC 的值是存在 lr 中的), 然后在子程序执行完毕的时候, 再把栈中的 lr 的值 pop 出来, 赋值给 PC, 这样就实现了子函数的正确的返回。

##### 2) 传递参数



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

C 语言进行函数调用的时候，常常会传递给被调用的函数一些参数，对于这些 C 语言级别的参数，被编译器翻译成汇编语言的时候，就要找个地方存放一下，并且让被调用的函数能够访问，否则就没法实现传递参数了。对于找个地方放一下，分两种情况。一种情况是，本身传递的参数不多于 4 个，就可以通过寄存器 r0~r3 传送参数。因为在前面的保存现场的动作中，已经保存好了对应的寄存器的值，那么此时，这些寄存器就是空闲的，可以供我们使用的了，那就可以放参数。另一种情况是，参数多于 4 个时，寄存器不够用，就得用栈了。

### 3) 临时变量保存在栈中

包括函数的非静态局部变量以及编译器自动生成的其他临时变量。

## 第二节 程序相关讲解

完整代码见目录 3. led\_c\_sp。

### 1. start.S

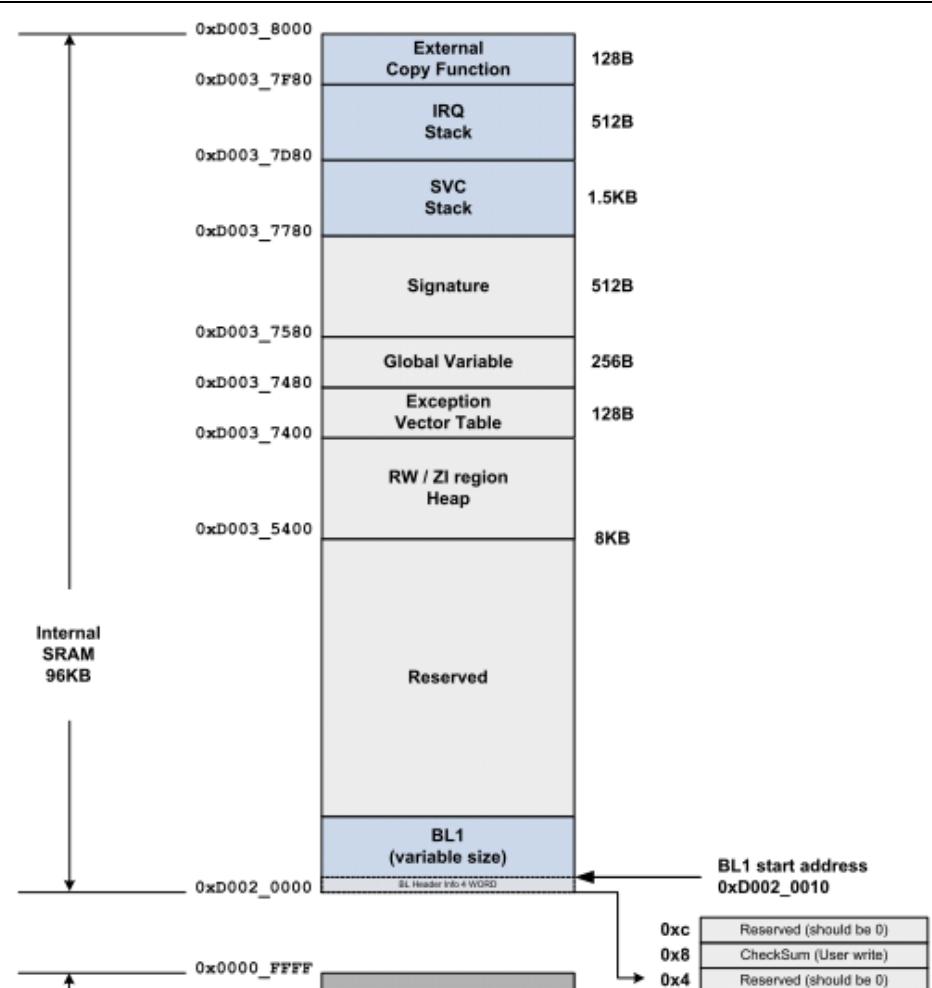
本章我们将使用 C 函数来实现点灯和延时的功能，在代码 3. led\_c\_sp 中，start.S 的作用如下：

**第一步** 关闭看门狗；

**第二步** 设置栈；

**第三步** 调用 C 函数 led\_blink()，实现 LED 闪烁；

设置栈，其实就是设置 sp 寄存器，让其指向一块可用的内存，我们将其指向 0xD003\_7D80，原因如下：



IROM 内部空间分配使用图

IROM 里的固定代码设置的 sp 就等于 0xD003\_7D80，我们可以遵从三星的旨意，也可以自己随便设置，只要指向的是一块可用的内存且不会覆盖我们自己的代码即可。

## 2. led.c

led.c 中编写了两个 C 函数，led\_blink() 实现 LED 闪烁，delay() 实现延时功能，代码如下：

```

void delay(int r0)           // 延时
{
    volatile int count = r0;
    while (count--)
        ;
}

void led_blink()             // LED 闪烁
{
    GPJ2CON = 0x00001111;    // 配置引脚

```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
while(1)
{
    GPJ2DAT = 0;           // LED on
    delay(0x100000);
    GPJ2DAT = 0xf;         // LED off
    delay(0x100000);
}
}
```

代码里的注释已经写得非常清楚了。相比起汇编，使用 C 语言来操作 LED 显得简洁容易了许多，所以后面的代码我们尽可能低使用 C 语言编写。

### 第三节 编译代码和烧写运行

将 sd 卡插入 PC，在 Fedora 终端执行如下命令：

```
# cd 3.led_c_sp
# make
# chmod 777 write2sd
# ./write2sd
```

执行 make 后会生成 210.bin 文件，执行 ./write2sd 后 210.bin 文件会被烧写到 sd 卡的扇区 1 中。

### 第四节 实验现象

将 sd 卡插入 Mini210S 中，选择 sd 卡启动，然后上电，可以看到以下现象：

LED 正常闪烁，设置栈以后，我们的程序就能调用 C 函数，这样将大大提升我们编写代码的速度。下一章我们将学习如何打开和关闭 icache，体验一下 icache 的威力。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第六章 控制 icache

### 第一节 什么是 cache

基于程序访问的局限性，在主存和 CPU 通用寄存器之前设置了一类高速的、容量较小的存储器，把正在执行的指令地址附件的一部分指令或数据从主存调入这类存储器，供 CPU 在一段时间内使用，这对提高程序的运行速度有很大的作用。这类介于主存和 CPU 之间的高速小容量存储器称作高速 cache。

比较常见的 cache 包括 icache 和 dcache。icache 的使用比较简单，系统刚上电时，icache 中的内容是无效的，并且 icache 的功能是关闭的，往 CP15 协处理器中的寄存器 1 的 bit[12]写 1 可以启动 icache，写 0 可以停止 icache。icache 关闭时，CPU 每次取指都要读主存，性能非常低。因为 icache 可随时启动，越早开 icache 越好。

与 icache 相似，系统刚上电时，dcache 中的内容是无效的，并且 dcache 的功能是关闭的，往 CP15 协处理器中的寄存器 1 的 bit[2]写 1 可以启动 dcache，写 0 可以停止 dcache。因为 dcache 必须在启动 mmu 后才能被启动，而对于裸机而言，没必要开 mmu，所以本教程的程序将不会启动 dcache。

### 第二节 程序相关讲解

完整代码见目录 4. led\_c\_icache，相比代码 3. led\_c\_sp，代码 4. leds\_c\_icache 与其的唯一区别在于在 start.S 中增加了控制 icache 的代码。

相关代码如下：

```
#ifdef CONFIG_SYS_ICACHE_OFF  
bic r0, r0, #0x00001000          @ clear bit 12 (I) I-cache  
#else  
orr r0, r0, #0x00001000          @ set bit 12 (I) I-cache  
#endif  
mcr p15, 0, r0, c1, c0, 0
```

如果没有定义 CONFIG\_SYS\_ICACHE\_OFF 则打开 icache，否则关闭 icache。至于协处理器的相关指令，可查阅 s3c2410 的芯片手册或者《arm 体系结构与编程》一书。

### 第三节 编译代码和烧写运行

将 sd 卡插入 PC，在 Fedora 终端执行如下命令：

```
# cd 4. led_c_icache  
# make  
# chmod 777 write2sd  
# ./write2sd
```

执行 make 后会生成 210.bin 文件，执行 ./write2sd 后 210.bin 文件会被烧写到 sd 卡的扇区 1 中。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第四节 实验现象

将 sd 卡插入 Mini210S 中，选择 sd 卡启动，然后上电，可以看到以下现象：

LED 正常闪烁，只是闪烁得非常慢，这是因为代码里的延时时间被增大了 10 倍。我们知道 IROM 的固化代码已经帮我们启动了 icache，如果要体验 icache 的威力，只需通过定义宏 CONFIG\_SYS\_ICACHE\_OFF 来关闭 icache。经测试，当关闭 icache 时，LED 闪烁一次的时间大约需要 20 秒，而打开 icache 时，LED 闪烁一次只需要 10 秒。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第七章 重定位代码到 IRAM+0x4000

### 第一节 重定位

对于程序而言，我们需要理解两个概念，一是程序当前所处的地址，即程序在运行时，所处的当前地址；二是程序的链接地址，即程序运行时应该位于的运行地址。编译程序时，可以指定程序的链接地址。

对于 S5PV210 而言，启动时只会从 NAND Flash/sd 等启动设备中拷贝前 16K 的代码到 IRAM 中，那么当我们的程序超过 16K 怎么办？那就需要我们在前 16K 的代码中将整个程序完完整整地拷贝到 DRAM 等其他更大存储空间，然后再跳转到 DRAM 中继续运行我们的代码，这个拷贝然后跳转的过程就叫重定位。本章中我们主要学习如何重定位，但是并不会涉如何使用到 DRAM，而是简单地将代码从 IRAM 的 0xD0020010 处拷贝到 IRAM 的 0xD0024000 处，然后跳转到 0xD0024000 处继续运行我们的代码。

### 第二节 程序相关讲解

完整代码见目录 5.link\_0x4000，该目录下的代码与上一章的代码的差别在于 start.S 和使用了链接脚本 link.lds，我们首先分析 link.lds。

#### 1. link.lds

什么是链接脚本？链接脚本就是程序链接时的参考文件，其主要目的是描述如何把输入文件中的段（SECTION）映射到输出文件中，并控制输出文件的存储布局。链接脚本的基本命令式 SECTIONS 命令，一个 SECTIONS 命令内部包含一个或多个段，段（SECTION）是链接脚本的基本单元，它表示输入文件中的某个段是如何放置的。

链接脚本的标准格式如下：

```
SECTIONS
{
    sections-command
    sections-command
}
```

下面我们配合 link.lds 进行具体讲解：

```
SECTIONS
{
    . = 0xD0024000;
    .text : {
        start.o
        * (.text)
    }
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
.data : {  
* (.data)  
}  
bss_start = .;  
.bss : {  
* (.bss)  
}  
bss_end = .;
```

- 1) 在链接脚本中，单独的点号(.)代表了当前位置，. = 0xD0024000; 表示程序的链接地址是0xD0024000；
- 2) link.lds 中的.text、.data、.bss 分别是 text 段、data 段、bss 段的段名(这些段名并不是固定的，是可以随便起的)。.text 段包含的内容是 start.o 和其余代码中所有的 text 段；.data 段包含的内容是代码中所有的 data 段；.bss 段包含的内容是代码中所有的 bss 段。
- 3) bss\_start 和 bss\_end 保存的是 bss 段的起始地址和结束地址，在 start.S 中会被用到。

下面解释一下什么是 data、text、bss 段：

- 1) **data 段：**数据段 (datasegment) 通常是指用来存放程序中已初始化的全局变量的一块内存区域。数据段属于静态内存分配。
- 2) **text 段：**代码段通常是指用来存放程序执行代码的一块内存区域。这部分区域的大小在程序运行前就已经确定，并且内存区域通常属于只读，某些架构也允许代码段为可写，即允许修改程序。在代码段中，也有可能包含一些只读的常数变量，例如字符串常量等。
- 3) **bss 段：**指用来存放程序中未初始化的全局变量的一块内存区域。BSS 是英文 BlockStarted by Symbol 的简称。当我们的程序有全局变量时，它是放在 bss 段的，由于全局变量默认初始值都是 0，所有我们需要手动清 bss 段。

## 2. start.S

在 start.S 中，我们初始化时钟后，增加了 3 个步骤：

第一步 重定位，代码如下：

```
// _start 当前所位于的地址  
adr r0, _start  
// _start 的链接地址  
ldr r1, =_start  
ldr r2, =bss_start  
cmp r0, r1  
beq clean_bss  
copy_loop:  
ldr r3, [r0], #4  
str r3, [r1], #4
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
cmp r1, r2  
bne copy_loop
```

首先需要知道的是，**adr** 指令获取的值是代码当前位于的地址，而 **ldr** 指令获取的值是代码的链接地址。再来看代码，代码里首先获得\_start 标号的当前地址(即 0xD0020010)，然后获取\_start 标号的链接地址(即 0xD0024000)，因为 bin 文件中不需要保存 bss 段，所有拷贝的代码长度为 bss\_start 的运行地址-\_start 的运行地址，使用 copy\_loop 进行拷贝。

### 第二步 清 bss，代码如下：

```
ldr r0, =bss_start  
ldr r1, =bss_end  
cmp r0, r1  
beq run_on_dram  
mov r2, #0  
clear_loop:  
str r2, [r0], #4  
cmp r0, r1  
bne clear_loop
```

首先获得 bss 段的起始地址(即 bss\_start)，然后获得 bss 段的结束地址(即 bss\_end)，最后使用 clear\_loop 将 bss 段所位于的内存清 0，bss\_start 和 bss\_end 的定义位于 link.lds。

### 第三步 跳转，代码如下：

```
run_on_dram:  
    ldr pc, =main
```

由于 ldr 指令获取的是 main 函数的链接地址，所以执行 ldr pc, =main 后，程序就跳转到 0xD002200+main 函数的 offset 的地址处了。

## 第三节 编译代码和烧写运行

将 sd 卡插入 PC，在 Fedora 终端执行如下命令：

```
# cd 5.link_0x4000  
# make  
# chmod 777 write2sd  
# ./write2sd
```

执行 make 后会生成 210.bin 文件，执行 ./write2sd 后 210.bin 文件会被烧写到 sd 卡的扇区 1 中。

## 第四节 实验现象

将 sd 卡插入 Mini210S 中，选择 sd 卡启动，然后上电，可以看到以下现象：

---

地址：广州市天河区龙口西路龙苑大厦A1栋1705 网址：<http://www.arm9.net>

电话：+86-20-85201025(售前、售后咨询) 技术支持(Tel): 13719442657 传真：+86-20-85261505

E-Mail: [capbily@163.com](mailto:capbily@163.com)(商务或项目合作) [dev\\_friendlyarm@163.com](mailto:dev_friendlyarm@163.com) (技术支持)



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

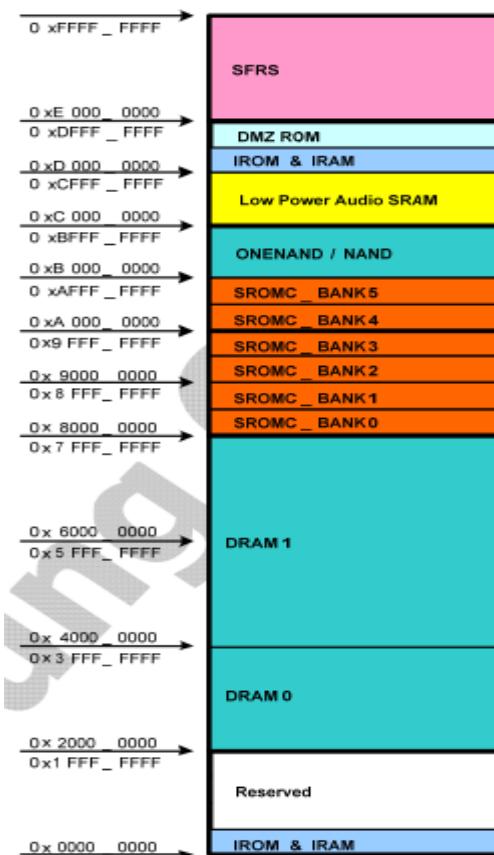
LED 正常闪烁, 该现象与前面章节的代码的运行效果一模一样, 但是程序的运行过程却有了很大的区别。通过本章的学习, 我们已经知道了如何对代码进行重定位, 这为我们下一章节将代码重定位到 DRAM 奠定了基础。

## 第八章 重定位代码到 DRAM

### 第一节 关于 DRAM

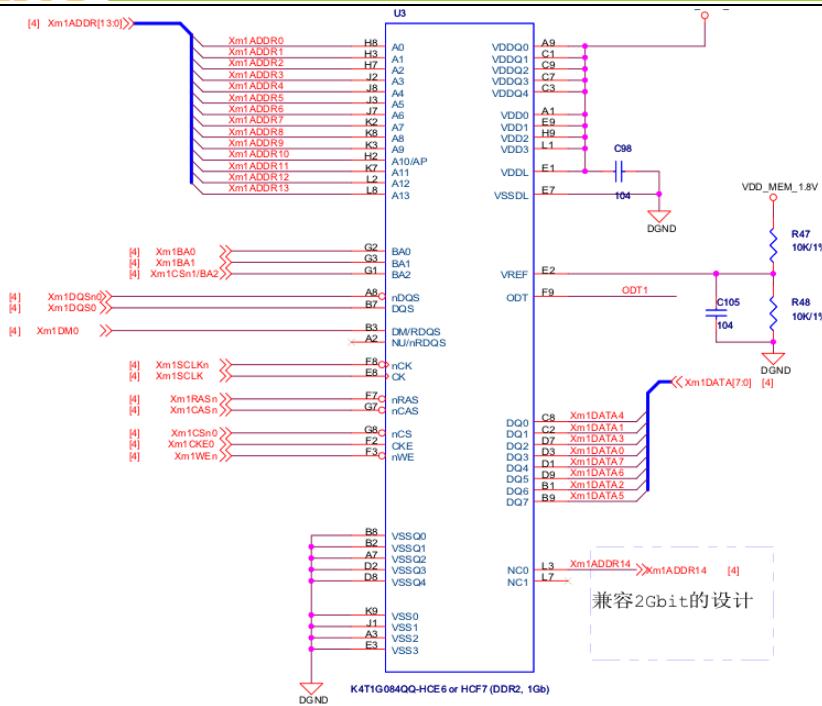
上一章我们讲解了如何对代码进行重定位，但是将代码重定位到只有 96K IRAM 中作用不大。正确的做法是将代码重定位到容量更大的主存中，即 DRAM。

S5PV210 中有两个独立的 DRAM 控制器，分别叫 DMC0 和 DMC1。DMC0 支持最大 512M 的 DRAM，DMC1 支持最大 1G 的 DRAM。它们都支持 DDR/DDR2，支持 128Mb、256Mb、512Mb、1Gb、2Gb、4Gb 的内存设备，支持 16/32bit 的位宽。通过下图能有更清晰的认识：



地址映射图

DRAM0 对应的地址是 0x2000\_0000~0x3FFF\_FFF 共 512M，DRAM1 对应的地址是 0x4000\_000~0x7FFF\_FFFF 共 1G。查阅 Mini210S 的原理图：



Mini210S DRAM 原理图

Mini210S 的 512M 的 DRAM 是由 4 片大小为 128M 的 DRAM 芯片组合而成(上图仅为其中一片)，观察片选引脚可知 4 片 DRAM 芯片都是挂接到 DMC0 处。



DRAM 连接引脚图

如何才能使用 DRAM？对应 Mini210S 而言，由于它只用到了 DMC0，所有我们只需要初始化 DMC0 和 DDR2 DRAM 芯片即可，需要注意的是，本章中对 DRAM 的初始化是实验性的，与 Superboot 对 DRAM 的初始化并不相同。

## 第二节 程序相关讲解

完整代码见目录 6. sdram，与前一章的代码相比，本章的代码有了较大的修改。首先整个工程分为 BL1 和 BL2 两个目录，目录 BL1 下的代码会被编译链接成一个名为 BL1.bin 的文件，而目录 BL2 下的代码会被编译链接成一个名为 BL2.bin 的文件。其中，BL1.bin 文件的链接地址是 0 (使用的是位置无关码，程序可以在任意可用的内存中运行)，BL2.bin 文件的链接地址是 0x23E00000 (使用的并不是位置无关码，所有程序必须位于该地址处才能正常运行)。BL1.bin 需被烧写到 sd 卡的扇区 1，BL2.bin 需被烧写到 sd 卡的扇区 49 处，为什么要这样烧写的原因将在后面的程序讲解中给出，完整的烧写过程可参考本章第四节的烧写步骤。整个程序的运行过程大致如下：系统上电后，首先将 sd 卡扇区 1 处的 BL1.bin 拷贝到 IRAM 的 0xD0020000 地址处，然后运行该部分代码，该部分代码首先会初始化 DRAM，然后把位于 sd 卡中扇区 49 处的 BL2.bin 拷贝到 DRAM 的 0x23E00000 地址处，最后跳转到该地址处继续运行。



## 1. BL1/start.S

相比上一章的代码, 本章的 start.S 在多了如下两个步骤:

**第一步** 调用 mem\_init 函数初始化内存, 函数的实现位于 memory.S, 这个文件是从 uboot 中借鉴过来的;

**第二步** 调用 copy\_code\_to\_dram() 将 BL2.bin 从 SD 卡拷贝到 DRAM 的 0x23E00000 处, copy\_code\_to\_dram() 的实现位于文件 mmc\_relocate.c;

## 2. BL1/memory.S

S5PV210 已经告诉我们如何初始化 DDR2 类型的 DRAM, 主要分为初始化 PHY DLL、初始化 DMC 和初始化 DDR2 DRAM 三大步骤, 具体细分共 27 个小步骤, 相当繁琐, 由于涉及到的寄存器过多, 如果一一介绍将需要非常大的篇幅, 所有本文档并不会一一解释寄存器的设置, 只会进行关键点的解释并且在代码中有详细注释。DDR2 类型的 DRAM 的完整初始化步骤见下图:

### 1.2.1.3 DDR2

Initialization sequence for DDR2 memory type:

1. To provide stable power for controller and memory device, the controller must assert and hold CKE to a logic low level. Then apply stable clock. **Note:** XDDR2SEL should be High level to hold CKE to low.
2. Set the **PhyControl0.ctrl\_start\_point** and **PhyControl0.ctrl\_inc** bit-fields to correct value according to clock frequency. Set the **PhyControl0.ctrl\_dll\_on** bit-field to '1' to turn on the PHY DLL.
3. DQS Cleaning: Set the **PhyControl1.ctrl\_shiftc** and **PhyControl1.ctrl\_offsetc** bit-fields to correct value according to clock frequency and memory tAC parameters.
4. Set the **PhyControl0.ctrl\_start** bit-field to '1'.
5. Set the **ConControl**. At this moment, an auto refresh counter should be off.
6. Set the **MemControl**. At this moment, all power down modes should be off.
7. Set the **MemConfig0** register. If there are two external memory chips, set the **MemConfig1** register.
8. Set the **PrechConfig** and **PwrdnConfig** registers.
9. Set the **TimingAref**, **TimingRow**, **TimingData** and **TimingPower** registers according to memory AC parameters.
10. If QoS scheme is required, set the **QosControl0~15** and **QosConfig0~15** registers.
11. Wait for the **PhyStatus0.ctrl\_locked** bit-fields to change to '1'. Check whether PHY DLL is locked.
12. PHY DLL compensates the changes of delay amount caused by Process, Voltage and Temperature (PVT) variation during memory operation. Therefore, PHY DLL should not be off for reliable operation. It can be off except runs at low frequency. If off mode is used, set the **PhyControl0.ctrl\_force** bit-field to correct value according to the **PhyStatus0.ctrl\_lock\_value[9:2]** bit-field to fix delay amount. Clear the **PhyControl0.ctrl\_dll\_on** bit-field to turn off PHY DLL.
13. Confirm whether stable clock is issued minimum 200us after power on
14. Issue a **NOP** command using the **DirectCmd** register to assert and to hold CKE to a logic high level.

15. Wait for minimum 400ns.
16. Issue a **PALL** command using the **DirectCmd** register.
17. Issue an **EMRS2** command using the **DirectCmd** register to program the operating parameters.
18. Issue an **EMRS3** command using the **DirectCmd** register to program the operating parameters.
19. Issue an **EMRS** command using the **DirectCmd** register to enable the memory DLLs.
20. Issue a **MRS** command using the **DirectCmd** register to reset the memory DLL.
21. Issue a **PALL** command using the **DirectCmd** register.
22. Issue two **Auto Refresh** commands using the **DirectCmd** register.
23. Issue a **MRS** command using the **DirectCmd** register to program the operating parameters without resetting the memory DLL.
24. Wait for minimum 200 clock cycles.
25. Issue an **EMRS** command using the **DirectCmd** register to program the operating parameters. If OCD calibration is not used, issue an **EMRS** command to set OCD Calibration Default. After that, issue an **EMRS** command to exit OCD Calibration Mode and to program the operating parameters.
26. If there are two external memory chips, perform steps 14~25 for chip1 memory device.
27. Set the **ConControl** to turn on an auto refresh counter. 28. If power down modes is required, set the **MemControl** registers.

### DRAM 的完整初始化步骤

memory.S 就是参考了上述相关知识进行了内存的初始化，它的主要进行了下面 4 个步骤的工作：

#### 第一步 设置 DRAM Driver Strength(内存访问信号的强度)

DRAM Driver Strength 数值越大，则内存访问信号的强度也越大。内存对工作频率是比较敏感的，当工作频率高于内存的标称频率时，将该选项的数值调高，可以提高电脑在超频状态下的稳定性，在这里我们使用默认值即可。

#### 第二步 初始化 PHY DLL

什么是 PHY DLL，见下图：

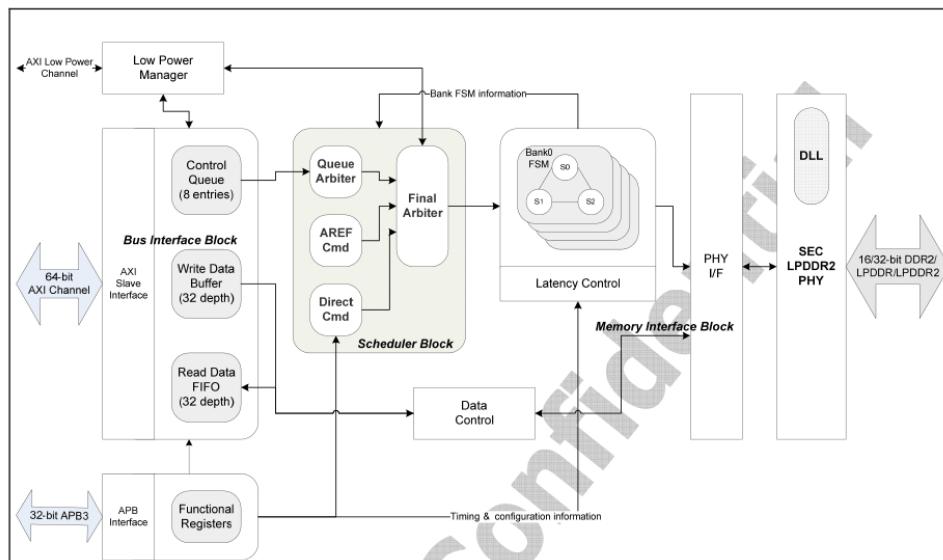


Figure 1-1 Overall Block Diagram



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

凡是使用 DDR 类型的 DRAM，都需要使用 DLL (Delay Locked Loop 延时锁定回路提供一个数据滤波信号) 技术，当数据有效时，存储控制器可使用这个数据滤波信号来精确定位数据。这里我们不用挖得太深，只需要按照上面的 27 个小步骤中步骤 2~4 来初始化 PHY DLL 就好。相关代码已经有详细的注释，这里不再粘贴代码。

### 第三步 初始化 DMC0

对应上面的 27 个小步骤中步骤 5~9，具体寄存器的设置请自行查看代码。

### 第四步 初始化 DDR2 DRAM

初始化 DRAM 只需要通过往寄存器 DIRECTCMD 写命令即可，需要写入什么命令可参考上面的 27 个小步骤中步骤 16~23。

## 3. BL1/mmc\_relocate.c

经过 mem\_init 函数对 DRAM 的初始化后，我们就可以拷贝代码到 DRAM 中然后跳转到 DRAM 中继续运行了，由 BL1 目录下的 mmc\_relocate.c 来实现这部分功能，mmc\_relocate.c 的代码如下：

```
void copy_code_to_dram(void)
{
    unsigned long ch;
    void (*BL2)(void);
    ch = *(volatile unsigned int *) (0xD0037488);
    copy_sd_sd_to_mem copy_b12 = (copy_sd_sd_to_mem) (*(unsigned int *) (0xD0037F98));

    unsigned int ret;
    // 通道 0
    if (ch == 0xEB000000)
    {
        // 0:channel 0
        // 49:源, 代码位于扇区 49, 1 sector = 512 bytes
        // 32:长度, 拷贝 32 sector, 既 16K
        // 0x23E00000:目的, 链接地址 0x23E00000
        ret = copy_b12(0, 49, 32, (unsigned int *) 0x23E00000, 0);
    }
    // 通道 2
    else if (ch == 0xEB200000)
    {
        ret = copy_b12(2, 49, 32, (unsigned int *) 0x23E00000, 0);
    }
    else
        return;

    // 跳转到 DRAM 中
    BL2 = (void *) 0x23E00000;
```

(\*BL2) () ;

}

首先我们定义了一个函数指针 copy\_BL2，将其赋值为 0xD0037F98。为什么要这么做，是因为 IROM 内部固化的代码已经帮我们实现了一类拷贝函数，其中就包括从 sd 卡拷贝内容到 DRAM 的函数，这类函数所位于的地址见下图：

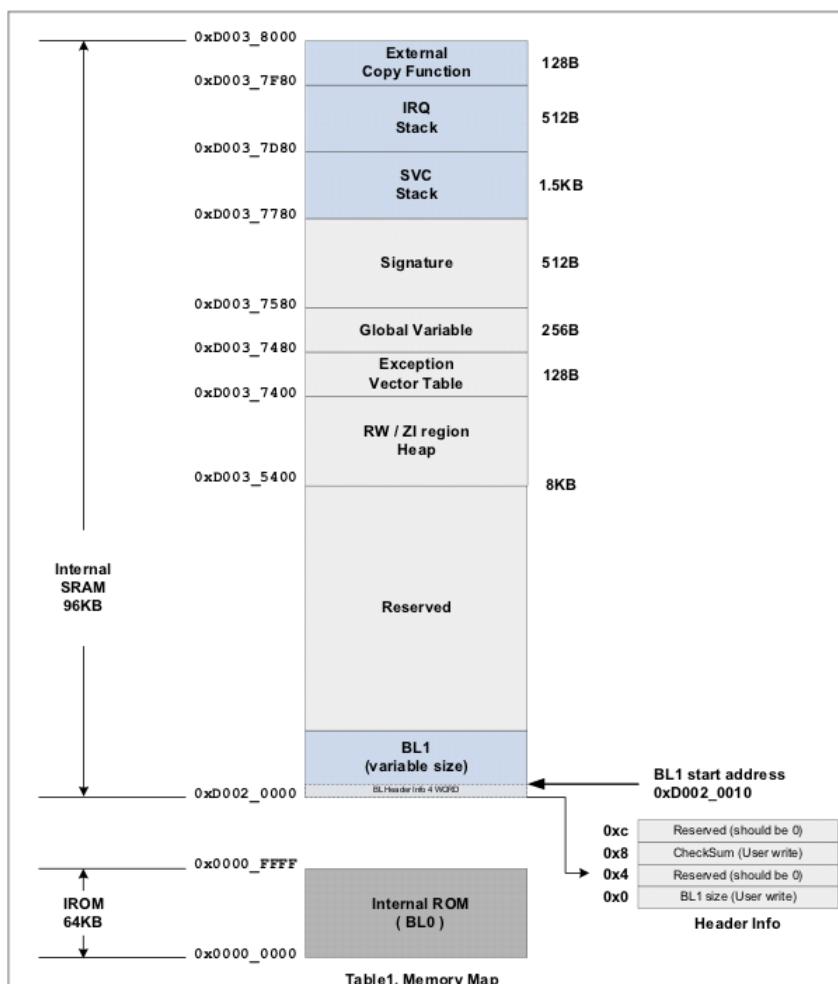


Table1. Memory Map

由上图可知，External Copy Function 位于 0xD0037F80~0xD0038000 处，其中 sd 卡拷贝内容到 DRAM 的函数就位于地址 0xD0037F98，其函数原型如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

#### ● SD/MMC Copy Function Address

External source clock parameter is used to fit EPLL source clock at 20MHz.

```
/*
 * This Function copy MMC(MoviNAND/iNand) Card Data to memory.
 * Always use EPLL source clock.
 * This function works at 20Mhz.
 * @param u32 StartBlkAddress : Source card(MoviNAND/iNand MMC) Address.(It must block address.)
 * @param u16 blockSize : Number of blocks to copy.
 * @param u32* memoryPtr : Buffer to copy from.
 * @param bool with_init : determined card initialization.
 * @return bool(u8) - Success or failure.
 */
#define CopySDMMCToMem(z,a,b,c,e)((bool(*)(int, unsigned int, unsigned short, unsigned int*, bool))(*((unsigned
int *)0xD0037F98)))(z,a,b,c,e))
```

- ⊕ StartBlkAddress: 从第几个扇区开始拷贝，一个扇区为 512byte
- ⊕ blockSize: 拷贝多少个扇区
- ⊕ memoryPtr: 拷贝到 DRAM 的哪个地址上
- ⊕ with\_init: 是否需要初始化 sd 卡

有了上面这些知识，也就很容易看懂 copy\_code\_to\_dram 函数了。通过读地址 0xD0037488 上的值来确定是使用通道 0 还是通道 1，芯片手册上明确指出“sd/MMC/eMMC boot - MMC Channel 0 is used for first boot. And Channel 2 is used for Second boot”，我们的 BL1.bin 就是 first boot，所以会使用通道 0，调用 CopysdMMCToMem 函数将 BL2.bin 从 sd 卡的扇区 49 拷贝到 DRAM 的 0x23E00000 处，拷贝的长度是 16K。最后给 BL2 这个函数指针赋值 0x23E0000，然后调用 BL2 函数即可跳转到 0x23E0000 处运行 BL2.bin 里的代码了。

## 4. BL2/start.S

BL1.bin 跳转到 0x23E00000 后，实际上运行的就是 start.S 里的代码，因为 BL2.bin 的链接脚本 sdran.lds 里指定了代码段的最开始放的是 start.o。在 BL2/start.S 中只做了一件事，就是使用一条位置相关的指令：ldr PC, =main 来调用 main 函数，main 函数的作用与上一章的代码一样化，都是 LED 闪烁。

### 第三节 编译代码和烧写运行

将 sd 卡插入 PC，在 Fedora 终端执行如下命令：

```
# cd 6.sdran
# make
# chmod 777 write2sd
# ./write2sd
```

执行 make 后，在 BL1 目录下会生成 BL1.bin，在 BL2 目录下会生成 BL2.bin。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

write2sd 的内容如下：

```
#!/bin/sh
sudo dd iflag=dsync oflag=dsync if=./BL1/BL1.bin of=/dev/sdb seek=1
sudo dd iflag=dsync oflag=dsync if=./BL2/BL2.bin of=/dev/sdb seek=49
执行./write2sd 后，BL1.bin 会被烧写到 sd 卡的扇区 1，BL2.bin 会被烧写到 sd 卡的扇区 49。
```

## 第四节 实验现象

将 sd 卡插入 Mini210S 中，选择 sd 卡启动，然后上电，可以看到以下现象：

LED 正常闪烁，该现象与上一章的代码运行效果一模一样，但是程序的运行过程却有了很大的区别。到此为止，我们就基本了解重定位的相关知识了。下面的章节，我们将介绍一个十分便捷的 USB 下载工具，来为我们的裸机程序开发锦上添花。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第九章 使用 MiniTools 烧写裸机程序

### 第一节 什么是 MiniTools

前面我们的裸机程序都是烧写到 sd 卡上，然后通过 sd 启动来运行我们的程序。这种方法缺点太多：

- 1) 频繁地拔插 sd 卡，手续繁琐；
- 2) 频繁地拔插烧写 sd 卡，严重地损害 sd 卡的寿命；
- 3) 无法将裸机程序烧写到 nand 中从 nand 启动运行裸机程序；

所以以后我们都使用友善之臂开发的全新 USB 烧写下载工具—MiniTools 来烧写裸机程序。

MiniTools 是一个烧写下载工具，其简要特性如下：

- 1) **无需串口连接**: MiniTools 完全采用 USB 线传输数据，无需串口，连接更简单，桌面更整洁。
- 2) **真正一键烧写**: MiniTools 真正实现一键烧写，烧写文件可单选，也可全选，不仅可以烧写系统(android, linux, wince)，还是烧写裸机，下载和烧写一气呵成。
- 3) **快速启动**: 以前的烧写方式需要不断来回插拔 sd 卡，这样不但容易损坏卡座，而且效率低下；现在使用 MiniTools，所有操作均在 PC 上完成，无需拔卡，无需接触开发板，立马查看到程序的运行效果，可以令开发更加快速方便。
- 4) **集成串口助手**: 借鉴了国内其他类似串口软件功能，更加方便实用，比如可自动定时发送任何数据或文件等。
- 5) **支持 32/64-bit 电脑**: MiniTools 安装程序已经包含 32/64-bit 电脑所需的 USB 下载驱动，可通吃所有 Windows 平台，如 WinXP, Win7 等；底层驱动采用 Google 官方提供的 Fastboot，因此更加稳定可靠，不会蓝屏。
- 6) **跨平台**: 采用 Qt4 开发，可以支持各种 Windows 系统或 Linux 发行版。

### 第二节 如何使用 MiniTools 烧写裸机程序

多说无益，现在我们立马来体验一下 MiniTools 的威力吧。MiniTools 的烧写功能很丰富，这里我们只是使用了其烧写裸机程序的功能。MiniTools 提供了两种烧写裸机程序的方式：**一种是直接下载到内存 DRAM，另外一种是将裸机下载到 NANDFlash**，下面以程序 7.1eds\_minitools 为例介绍如何使用 MiniTools：

方式一 下载到 DRAM，其设置方式如下：

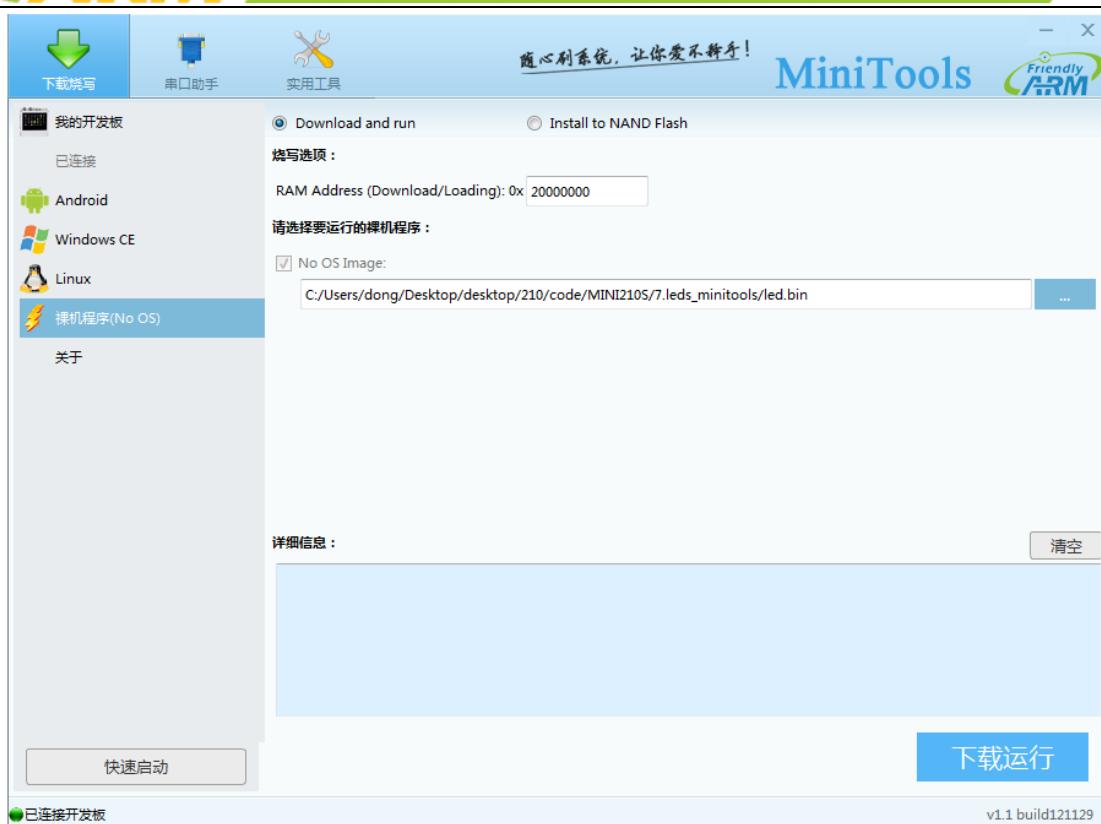


追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



先选中上方的“Download and run”，设置好下载地址“RAM Address (Download/Loading)”，选择要运行的裸机程序，再点击“下载运行”就可以了。MiniTools首先会把裸机程序下载到DRAM的地址0x20000000处，然后跳转到该地址上运行裸机程序，所以只要PC上一点击“下载运行”就可以马上看到开发板上裸机程序的运行效果了。

对于Mini210S开发板，MiniTools支持的裸机程序下载地址是：

0x20000000~0x3F5FFFFF(共502M)

剩下的0x3F600000~0x3FFFFFF的10M空间用来运行Superboot，502M的空间足够让我们运行任何裸机程序了。

**注意：不同的CPU会有不同的起始地址，在此为0x20000000**

**方式二 下载到NANDFlash，其设置方式如下：**

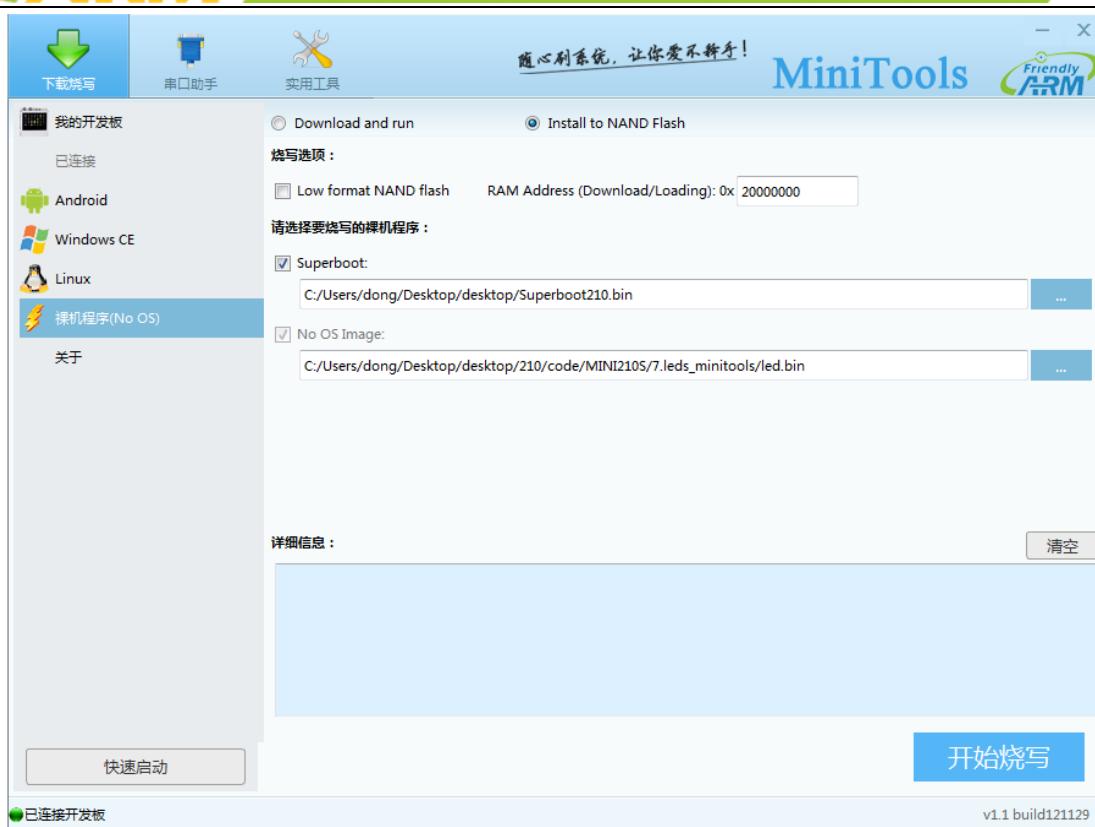


追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



选中“Install to NAND Flash”，设置好加载地址“RAM Address (Download/Loading)”，选择使用 Superboot 加载裸机程序，并选择要加载的裸机程序，点击“开始烧写”；MiniTools 首先把 Superboot 和裸机程序都烧写到 NANDFlash 中，然后点击“快速启动”，这时 NANDFlash 中的 Superboot 会把裸机程序拷贝到 DRAM 的 0x20000000 地址处，然后跳转到该地址上运行裸机程序，这样我们就可以在开发板上观察裸机的运行效果了。另外，以后只要选择 nand 启动时，NANDFlash 中的 Superboot 都会将裸机程序加载到 DRAM 中运行了。

上面两种方式都可以烧写运行我们的裸机程序，由于方式一较为便捷，并且不会损坏 NAND Flash 中的原有数据，本文档所涉及的所有的程序都将统一采用这种方式进行烧写和运行。

### 第三节 程序相关讲解

完整代码见目录 7. leds\_MiniTools，这个程序的代码跟我们的第一个程序 1. leds\_s 基本一样，唯一的差别在于 Makefile 中链接时指定的链接地址是 0x20000000，因为 MiniTools 是将裸机程序下载到 DRAM 的 0x20000000 的。

### 第四节 实验现象

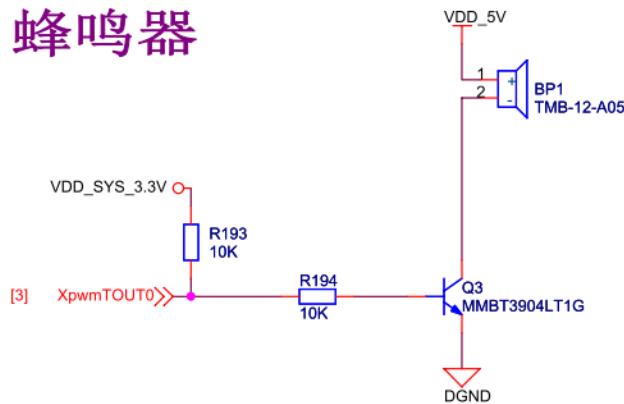
同样是 LED 闪烁，但是由于我们是通过 Superboot 将裸机程序加载到 DRAM 的，而 Superboot 已经帮我们初始化了系统时钟，所以这次我们可以明显感觉到 LED 闪烁的速度变快了，后面的章节我们将会自己来初始化系统时钟。

## 第十章 控制蜂鸣器

### 第一节 查阅原理图

Mini210S 带有一个蜂鸣器，十分吵闹，本章将学习如何控制蜂鸣器，首先查阅原理图：

**蜂鸣器**



相关引脚：



### 第二节 程序相关讲解

完整代码见目录 8. buzzer, 蜂鸣器的操作十分简单，原理跟操作 LED 一样，通过控制 GPD0\_0 这个引脚就可以达到控制蜂鸣器的目的。

#### 1. start.S

start.S 做了下面 3 件事：

第一步 关看门狗；

第二步 设置栈，由于 Superboot 帮我们初始化了 DRAM，所以我们把栈设置在 DRAM 的末端即 0x40000000 处；

第三步 调用 main 函数；

#### 2. buzzer.c

完整代码如下：

```
#define GPD0CON      (*(volatile unsigned long *)0xE02000A0)
#define GPDODAT      (*(volatile unsigned long *)0xE02000A4)
void buzzer_init(void)
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
{  
    GPD0CON |= 1<<0;  
}  
void buzzer_on(void)  
{  
    GPD0DAT |= 1<<0;  
}  
void buzzer_off(void)  
{  
    GPD0DAT &= ~(1<<0);  
}  
函数 buzzer_init() 配置 GPIO 引脚，使 GPD0_0 用于输入功能；  
函数 buzzer_on() 使引脚 GPD0_0 输出 0，蜂鸣器响；  
函数 buzzer_off() 使引脚 GPD0_0 输出 1，蜂鸣器不响；
```

### 3. main.c

在 main.c 中，首先会调用 buzzer\_init() 来初始化蜂鸣器，然后通过一个 while 循环不断的控制蜂鸣器的运行和停止。

## 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 8.buzzer  
# make
```

在 8.buzzer 目录下会生成 buzzer.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：

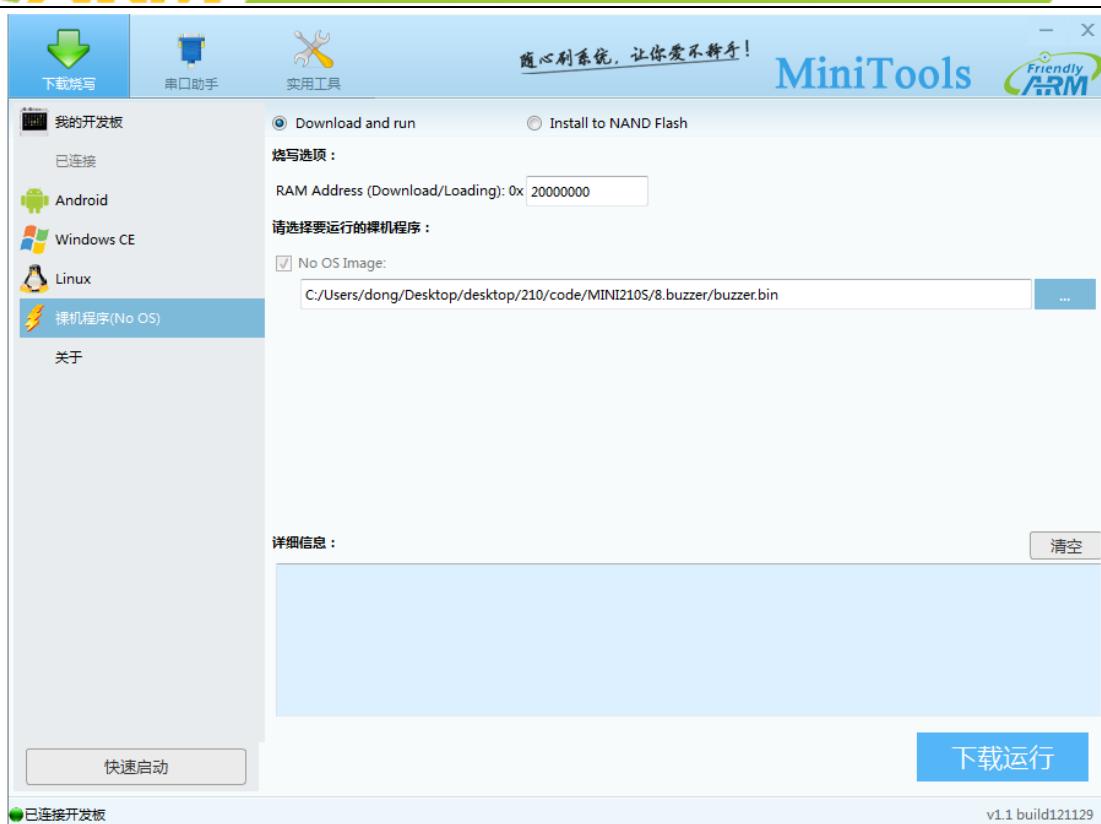


追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



点击“下载运行”，Minitools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

实验现象很简单，就是可以听到开发板欢快地滴滴声响起来了，控制蜂鸣器就是这么简单。

地址：广州市天河区龙口西路龙苑大厦A1栋1705

网址：<http://www.arm9.net>

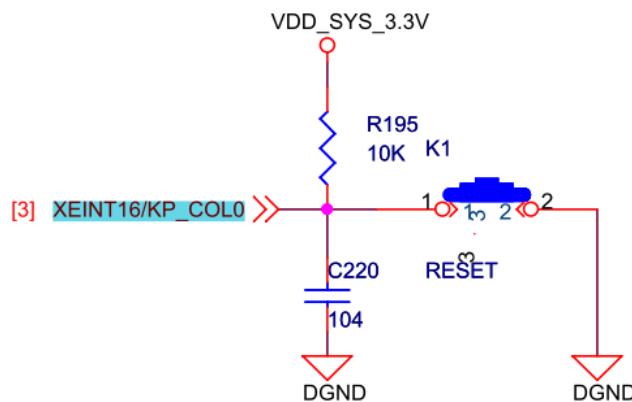
电话：+86-20-85201025(售前、售后咨询) 技术支持(Tel): 13719442657 传真：+86-20-85261505

E-Mail: [capbily@163.com](mailto:capbily@163.com)(商务或项目合作) [dev\\_friendlyarm@163.com](mailto:dev_friendlyarm@163.com) (技术支持)

# 第十一章 查询方式检测按键

## 第一节 查看原理图

Mini210S 中共有 4 个用户按键，其中按键 KEY1 的原理图如下，其余三个按键的原理图与 KEY1 的相似：



按键原理图

搜索“XEINT16/KP\_COL0”，可得



按键引脚图

## 第二节 程序相关讲解

完整代码见目录 9.key\_led，除去 main.c 中的代码，其余代码与 8.buzzer 大同小异。

### 1. main.c

核心代码如下：

```
// 轮询的方式查询按键事件
while(1)
{
    dat = GPH2DAT;
    if(dat & (1<<0))          // KEY1 被按下，则 LED1 亮，否则 LED1 灭
        GPJ2DAT |= 1<<0;       // OFF
    else
        GPJ2DAT &= ~(1<<0); // ON
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
if(dat & (1<<1))          // KEY2 被按下，则 LED2 亮，否则 LED2 灭  
    GPJ2DAT |= 1<<1;  
else  
    GPJ2DAT &= ~(1<<1);  
  
if(dat & (1<<2))          // KEY3 被按下，则 LED3 亮，否则 LED3 灭  
    GPJ2DAT |= (1<<2);  
else  
    GPJ2DAT &= ~(1<<2);  
  
if(dat & (1<<3))          // KEY4 被按下，则 LED4 亮，否则 LED4 灭  
    GPJ2DAT |= 1<<3;  
else  
    GPJ2DAT &= ~(1<<3);  
}
```

程序很简单，首先配置 GPJ2\_0/1/2/3 引脚为输出功能以及配置 GPH2\_0/1/2/3 引脚为输入功能，然后使用轮询的方式不断的读 GPH2\_0/1/2/3 引脚的值，当检测到某个按键被按下时，即对应的引脚为低，此时我们点亮对应的 LED，否者让 LED 保持熄灭的状态。

### 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 9.key_led  
# make
```

在 9.key\_led 目录下会生成 key\_led.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：

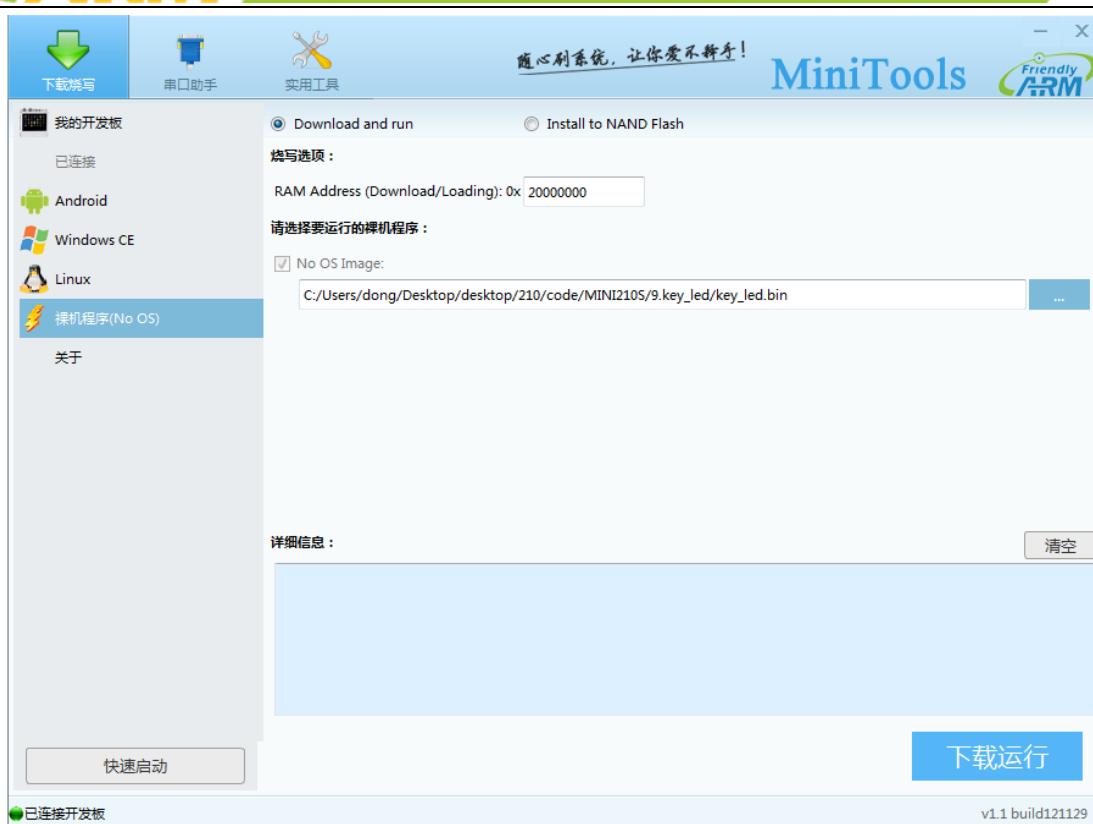


追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



点击“下载运行”，Minitools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

未按下任何按键时，所有 LED 保持熄灭状态。当按下按键 KEY1 时，LED1 被点亮，松开按键 KEY1 时，LED1 熄灭。对于其余 3 个按键，也是相同情况。用查询的方式来检测按键太占 CPU 使用率了，除了检测按键，CPU 无法进行其他工作，后面学习中断的知识后，我们将改用中断的方式来检测按键中断，这将大大降低 CPU 的使用率。

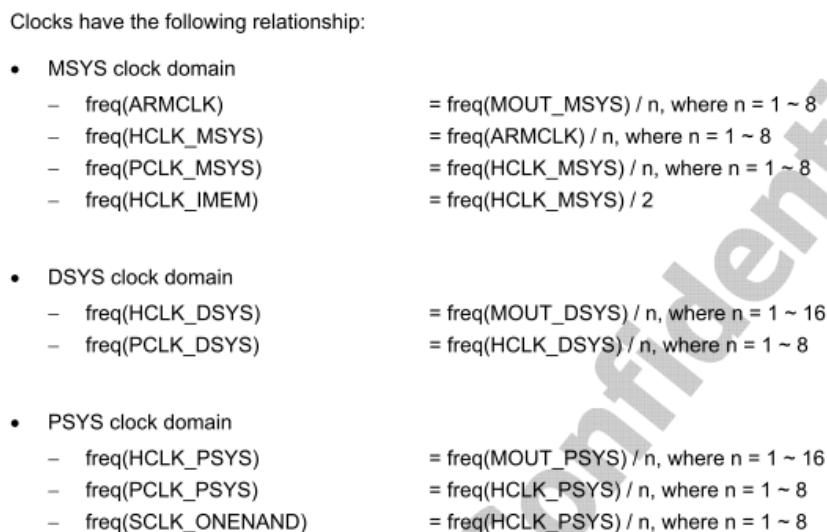
## 第十二章 初始话系统时钟

### 第一节 S5PV210 时钟体系

S5PV210 中包含 3 大类时钟 domain，分别是主系统时钟 domain (简称 MSYS，下面将使用简称来进行相关讲解)、显示相关的时钟 domain (DSYS)、外围设备的时钟 domain (PSYS)。

- 1) **MSYS:** 用来给 cortex a8 处理器, dram 控制器, 3D, IRAM, IROM, 中断控制器等提供时钟;
- 2) **DSYS:** 用来给显示相关的部件提供时钟, 包括 FIMC, FIMD, JPEG, and multimedia IPs;
- 3) **PSYS:** 用来给外围设备提供时钟, 如 i2s, spi, i2c, uart 等

Mini210S 外接的晶振频率(简称 Fin)为 24MHz，通过时钟控制逻辑 PLL 可以提高系统时钟。S5PV210 共有 4 个倍频器，即 PLL，包括 APLL(供 MSYS 使用), MPLL(供 DSYS 使用), EPLL(供 PSYS 使用), VPLL(供 video 相关的时钟使用)。3 大类时钟 domain 中，可以使用不同的分频，使其给不同部件输出所需要的时钟，各类时钟的关系如下图：



S5PV210 时钟分类图

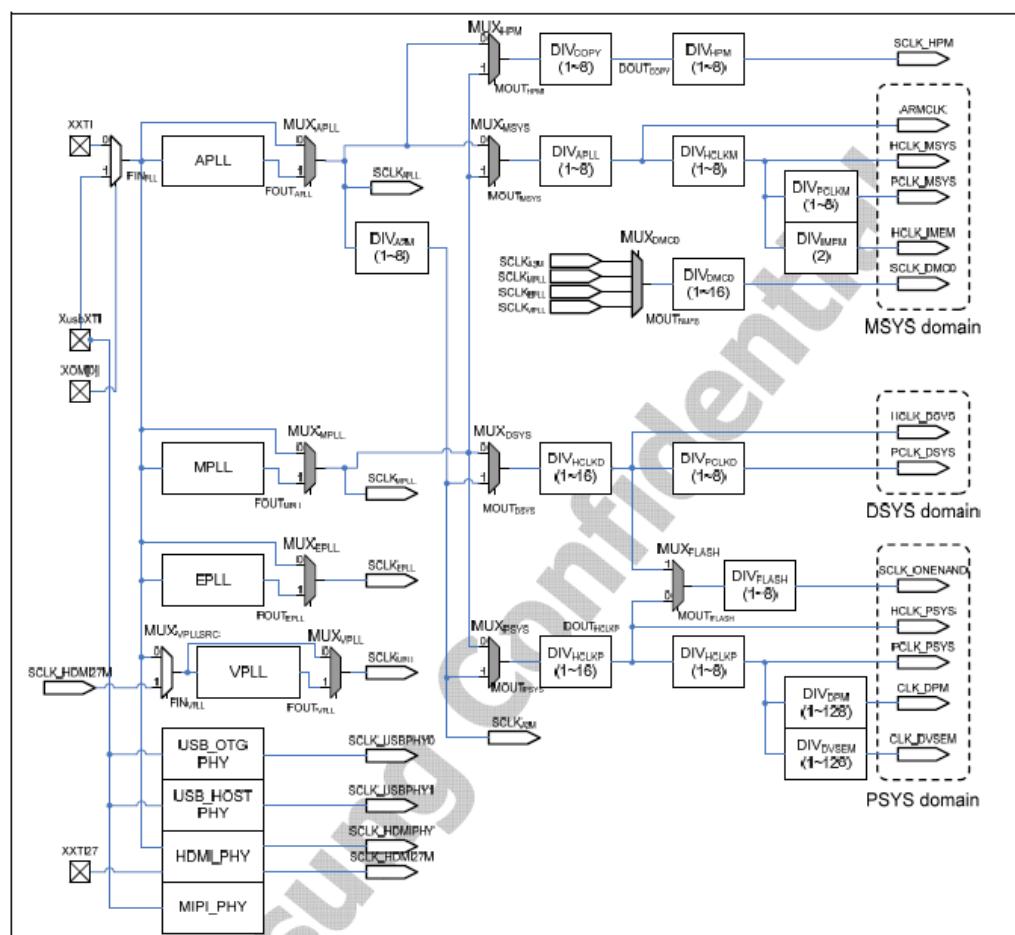
如何确定各类时钟的值，芯片手册上给出了参考值，我们按照参考值设置即可，各类时钟的参考值如下图：

Values for the high-performance operation:

- |                      |                    |
|----------------------|--------------------|
| • freq(ARMCLK)       | = 1000 MHz         |
| • freq(HCLK_MSYS)    | = 200 MHz          |
| • freq(HCLK_IMEM)    | = 100 MHz          |
| • freq(PCLK_MSYS)    | = 100 MHz          |
| • freq(HCLK_DSYS)    | = 166 MHz          |
| • freq(PCLK_DSYS)    | = 83 MHz           |
| • freq(HCLK_PSYS)    | = 133 MHz          |
| • freq(PCLK_PSYS)    | = 66 MHz           |
| • freq(SCLK_ONENAND) | = 133 MHz, 166 MHz |

### S5PV210 时钟设置参考值图

具体如何设置上述各种各样的时钟，可参考下图(芯片手册 P361)：



S5PV210 时钟设置参考图



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

该图十分重要，依据上图我们就可以设置好所有硬件部件所需的工作时钟（实际上我们并不需要设置好所有部件的工作时钟，我们只需设置好我们需要使用的硬件部件的工作时钟即可），在本章第二节中，我们将以上图为基础，通过设置时钟相关的寄存器，达到初始化时钟的目的。

## 第二节 程序相关讲解

完整代码见目录 10.clock\_s 和 11.clock\_c，本章涉及的代码有两套，包括 10.clock\_s（使用汇编初始化时钟）和 11.clock\_c（使用 c 语言初始化时钟），两者的本质是一样的，初始化时钟所达到的效果也一样，想巩固汇编指令的话，可以阅读 10.clock\_s 里的相关代码。由于 C 语言编写的代码思路更清晰，更便于阅读和理解，下面将以 11.clock\_c 里的代码为参考进行讲解。

### 1. start.S

在调用 main 函数之前，调用了时钟初始化函数 clock\_init，进行时钟相关的设置。

### 2. clock.c

clock\_init() 在 clock.c 中定义，具体代码如下：

```
void clock_init()
{
    // 1 设置各种时钟开关，暂时不使用 PLL
    CLK_SRC0 = 0x0;

    // 2 设置锁定时间，使用默认值即可
    APLL_LOCK = 0x0000FFFF;
    MPLL_LOCK = 0x0000FFFF;

    // 3 设置分频
    CLK_DIV0 = 0x14131440;

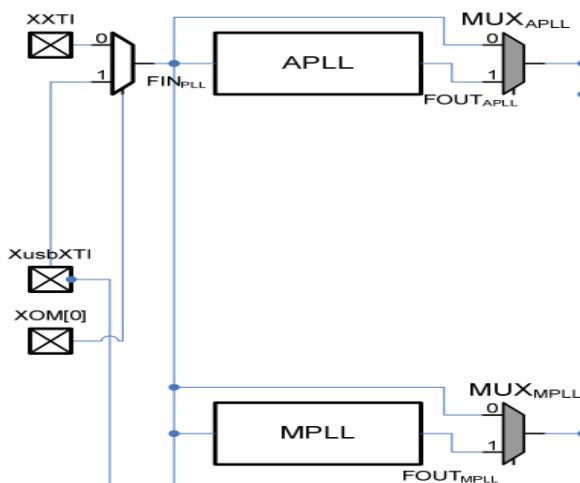
    // 4 设置 PLL
    APLL_CON0 = APLL_VAL;
    MPLL_CON  = MPLL_VAL;

    // 5 设置各种时钟开关，使用 PLL
    CLK_SRC0 = 0x10001111;
}
```

上述代码共有 5 个步骤，下面我们一一讲解每一个步骤：

#### 第一步 设置各种时钟开关，暂时不使用 PLL

根据本章第一节中所给出的时钟设置图（芯片手册 P361），下面是放大图：



首先我们需要选择使用外接 24MHz 晶振，由上图可知，APLL 和 MPLL 的时钟源由“FINPLL”决定，在芯片手册中搜索“FINPLL”，可知相关寄存器为 CLK\_SRC0，见下图：

### 3.7.3.1 Clock Source Control Registers (CLK\_SRC0, R/W, Address = 0xE010\_0200)

CLK_SRC0	Bit	Description	Initial State
Reserved	[31:29]	Reserved	0x0
ONENAND_SEL	[28]	Control MUXFLASH (0:HCLK_PSYS, 1:HCLK_DSYS)	0
Reserved	[27:25]	Reserved	0x0
MUX_PSYS_SEL	[24]	Control MUX_PSYS (0:SCLKMPPLL, 1:SCLKA2M)	0
Reserved	[23:21]	Reserved	0x0
MUX_DSYS_SEL	[20]	Control MUX_DSYS (0:SCLKMPPLL, 1:SCLKA2M)	0
Reserved	[19:17]	Reserved	0x0
MUX_MSYS_SEL	[16]	Control MUX_MSYS (0:SCLKAPLL, 1:SCLKMPPLL)	0
Reserved	[15:13]	Reserved	0x0
VPLL_SEL	[12]	Control MUXVPLL (0:FINVPLL, 1:FOUTVPLL)	0
Reserved	[11:9]	Reserved	0x0
EPLL_SEL	[8]	Control MUXEPLL (0:FINPLL, 1:FOUTEPLL)	0
Reserved	[7:5]	Reserved	0x0
MPLL_SEL	[4]	Control MUXMPPLL (0:FINPLL, 1:FOUTMPPLL)	0
Reserved	[3:1]	Reserved	0x0
APLL_SEL	[0]	Control MUXAPLL (0:FINPLL, 1:FOUTAPLL)	0

在未设置 PLL 和各种分频系数之前，我们不能使用 PLL，为了保险起见，暂时直接使用频率较低的外接的 24MHz 晶振，待设置好 PLL 和分频系数后再重新设置各种时钟开关。

### 第二步 设置锁定时间

设置 PLL 后，时钟从 Fin 提升到目标频率时，需要一定的时间，即锁定时间。

### 第三步 设置分频

与分频相关的寄存器是 CLK\_DIV0，见下图：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 3.7.4.1 Clock Divider Control Register (CLK\_DIV0, R/W, Address = 0xE010\_0300)

CLK_DIV0	Bit	Description	Initial State
Reserved	[31]	Reserved	0
PCLK_PSYS_RATIO	[30:28]	DIVPCLKP clock divider ratio, PCLK_PSYS = HCLK_PSYS / (PCLK_PSYS_RATIO + 1)	0x0
HCLK_PSYS_RATIO	[27:24]	DIVHCLKP clock divider ratio, HCLK_PSYS = MOUT_PSYS / (HCLK_PSYS_RATIO + 1)	0x0
Reserved	[23]	Reserved	0
PCLK_DSYS_RATIO	[22:20]	DIVPCLKD clock divider ratio, PCLK_DSYS = HCLK_DSYS / (PCLK_DSYS_RATIO + 1)	0x0
HCLK_DSYS_RATIO	[19:16]	DIVHCLKD clock divider ratio, HCLK_DSYS = MOUT_DSYS / (HCLK_DSYS_RATIO + 1)	0x0
Reserved	[15]	Reserved	0
PCLK_MSYS_RATIO	[14:12]	DIVPCLKM clock divider ratio, PCLK_MSYS = HCLK_MSYS / (PCLK_MSYS_RATIO + 1)	0x0
Reserved	[11]	Reserved	0
HCLK_MSYS_RATIO	[10:8]	DIVHCLKM clock divider ratio, HCLK_MSYS = ARMCLK / (HCLK_MSYS_RATIO + 1)	0x0
Reserved	[7]	Reserved	0
A2M_RATIO	[6:4]	DIVA2M clock divider ratio, SCLKA2M = SCLKAPLL / (A2M_RATIO + 1)	0x0
Reserved	[3]	Reserved	0
APLL_RATIO	[2:0]	DIVAPLL clock divider ratio, ARMCLK = MOUT_MSYS / (APLL_RATIO + 1)	0x0

现在我们来根据本章第一节中给出的时钟设置参考值来设置该寄存器。

Values for the high-performance operation:

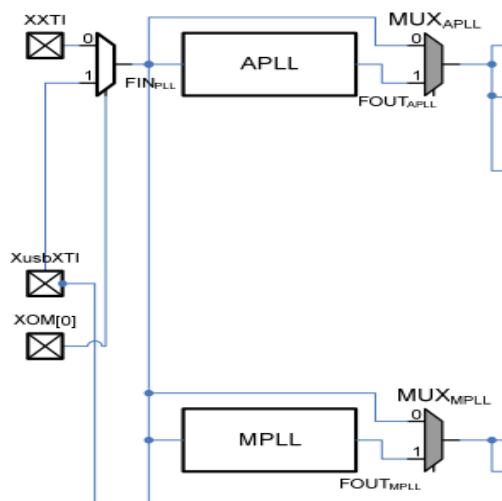
- freq(ARMCLK) = 1000 MHz
- freq(HCLK\_MSYS) = 200 MHz
- freq(HCLK\_IMEM) = 100 MHz
- freq(PCLK\_MSYS) = 100 MHz
- freq(HCLK\_DSYS) = 166 MHz
- freq(PCLK\_DSYS) = 83 MHz
- freq(HCLK\_PSYS) = 133 MHz
- freq(PCLK\_PSYS) = 66 MHz
- freq(SCLK\_ONENAND) = 133 MHz, 166 MHz

- ARMCLK = 1000MHz = MOUT\_MSYS / (APLL\_RATIO + 1), 经过在第四和第五步的设置后, MOUT\_MSYS 会被设置为 1000MHz, 所以 APLL\_RATIO=0 即可
- SCLKA2M=200MHz=SCLKAPLL / (A2M\_RATIO + 1), 由于 SCLKAPLL=1000MHz, 所以 A2M\_RATIO=4。
- HCLK\_MSYS=200MHz=ARMCLK / (HCLK\_MSYS\_RATIO + 1), 所以 HCLK\_MSYS\_RATIO=4
- PCLK\_MSYS=100MHz=HCLK\_MSYS / (PCLK\_MSYS\_RATIO + 1), 所以 PCLK\_MSYS\_RATIO=1
- HCLK\_DSYS=166MHz=MOUT\_DSYS / (HCLK\_DSYS\_RATIO + 1), 经过在第四和第五步的设置后, MOUT\_DSYS =667MHz, 所以 HCLK\_DSYS\_RATIO=3
- PCLK\_DSYS=83MHz=HCLK\_DSYS / (PCLK\_DSYS\_RATIO + 1), 所以 PCLK\_DSYS\_RATIO=1

- HCLK\_PSYS=133MHz=MOUT\_PSYS / (HCLK\_PSYS\_RATIO + 1), 经过在第四和第五步的设置后, MOUT\_PSYS =667MHz 所以 HCLK\_PSYS\_RATIO=4
- PCLK\_PSYS=66Mhz=HCLK\_PSYS / (PCLK\_PSYS\_RATIO + 1), 所以 HCLK\_PSYS\_RATIO=1  
所以 CLK\_DIV0 = 0x14131440;

#### 第四步 设置 PLL

PLL 即倍频器, 用来放大运行频率。设置好分频后, 我们就需要设置 PLL 了。APLL/MPLL 的启动是通过设置 APLL\_CON0/MPLL\_CON 寄存器, 我们逐个来设置。



- APLL\_CON0

<b>APLL_CON0</b>	<b>Bit</b>	<b>Description</b>	<b>Initial State</b>
ENABLE	[31]	PLL enable control (0: disable, 1: enable)	0
Reserved	[30]	Reserved	0
LOCKED	[29]	PLL locking indication 0 = Unlocked 1 = Locked Read Only	0
Reserved	[28:26]	Reserved	0x0
MDIV	[25:16]	PLL M divide value	0xC8
Reserved	[15:14]	Reserved	0
PDIV	[13:8]	PLL P divide value	0x3
Reserved	[7:3]	Reserved	0
SDIV	[2:0]	PLL S divide value	0x1

ALPP\_CON0 负责设置 APLL, FINPLL=24MHz, 经过 APLL 后, 我们将输出 FOUT=1000Mhz 的时钟频率, FOUT 的计算公式如下:

$$FOUT = MDIV * FIN / (PDIV * 2^{(SDIV-1)}) = 1000 \text{ MHz}$$

由于 FIN=24MHz, FOUT=1000MHz, 我们可以这样取值: MDIV= 0x7d, PDIV= 0x3, SDIV=1。这 3 个值并不是固定死的, 只要能使 FOUT=1000Mhz, 任意搭配都是可以的。

- MPLL\_CON



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 3.7.2.2 PLL Control Registers (MPLL\_CON, R/W, Address = 0xE010\_0108)

MPLL_CON	Bit	Description	Initial State
ENABLE	[31]	PLL enable control (0: disable, 1: enable)	0
Reserved	[30]	Reserved	0
LOCKED	[29]	PLL locking indication 0 = Unlocked 1 = Locked Read Only	0
Reserved	[28]	Reserved	0
VSEL	[27]	VCO frequency range selection	0x0
Reserved	[26]	Reserved	0
MDIV	[25:16]	PLL M divide value	0x14D
Reserved	[15:14]	Reserved	0
PDIV	[13:8]	PLL P divide value	0x3
Reserved	[7:3]	Reserved	0
SDIV	[2:0]	PLL S divide value	0x1

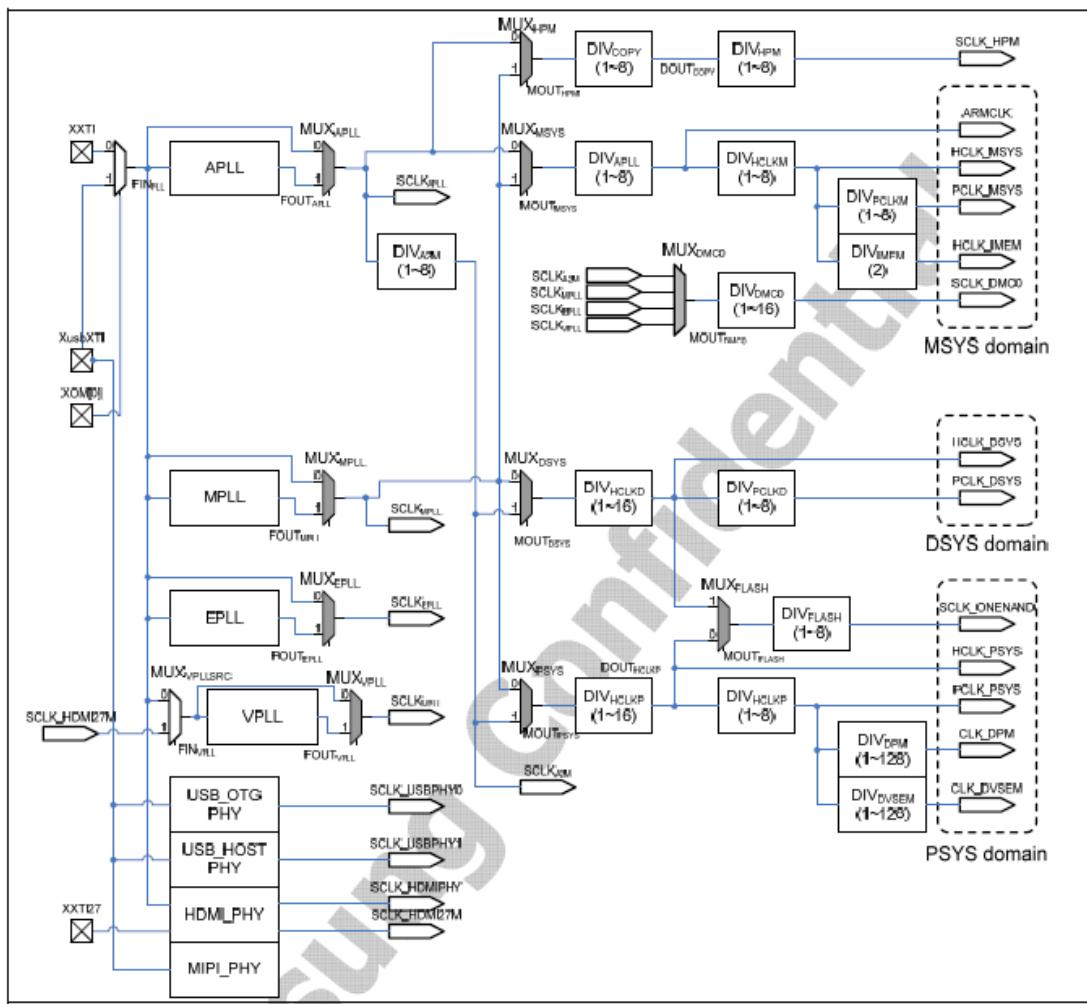
MPLL\_CON 寄存器负责设置 MPLL, 经过 MPLL 后, 我们将输出 FOUT=667Mhz 的时钟频率, FOUT 的计算公式如下:

$$FOUT = MDIV * FIN / (PDIV * 2^SDIV) = 667 \text{ MHz}$$

由于 FIN=24MHz, FOUT=667MHz, 我们可以这样取值: MDIV=0x29B,

PDIV= 0xC, SDIV=1。这 3 个值并不是固定死的, 只要能使 FOUT=667Mhz, 任意搭配都是可以的。

## 第五步 设置各种时钟开关



S5PV210 时钟设置参考图

在上图中，所有的 MUX 都是用来选择时钟的，相关寄存器是 CLK\_SRC0，见下图：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

### 3.7.3.1 Clock Source Control Registers (CLK\_SRC0, R/W, Address = 0xE010\_0200)

CLK_SRC0	Bit	Description	Initial State
Reserved	[31:29]	Reserved	0x0
ONENAND_SEL	[28]	Control MUXFLASH (0:HCLK_PSYS, 1:HCLK_DSYS)	0
Reserved	[27:25]	Reserved	0x0
MUX_PSYS_SEL	[24]	Control MUX_PSYS (0:SCLKMPPLL, 1:SCLKA2M)	0
Reserved	[23:21]	Reserved	0x0
MUX_DSYS_SEL	[20]	Control MUX_DSYS (0:SCLKMPPLL, 1:SCLKA2M)	0
Reserved	[19:17]	Reserved	0x0
MUX_MSYS_SEL	[16]	Control MUX_MSYS (0:SCLKAPLL, 1:SCLKMPPLL)	0
Reserved	[15:13]	Reserved	0x0
VPLL_SEL	[12]	Control MUXVPLL (0:FINVPLL, 1:FOUTVPLL)	0
Reserved	[11:9]	Reserved	0x0
EPLL_SEL	[8]	Control MUXEPLL (0:FINPLL, 1:FOUTEPLL)	0
Reserved	[7:5]	Reserved	0x0
MPLL_SEL	[4]	Control MUXMPPLL (0:FINPLL, 1:FOUTMPPLL)	0
Reserved	[3:1]	Reserved	0x0
APLL_SEL	[0]	Control MUXAPLL (0:FINPLL, 1:FOUTAPLL)	0

参考 S5PV210 时钟设置参考图，设置各种时钟开关：

APLL\_SEL=1，使用 FOUTAPLL

MPLL\_SEL=1，使用 FOUTMPPLL

EPLL\_SEL=1，使用 FOUTEPLL

VPLL\_SEL=1，使用 FOUTVPLL

MUX\_MSYS\_SEL=0，使用 SCLKAPLL

MUX\_DSYS\_SEL=0，使用 SCLKMPPLL

MUX\_PSYS\_SEL=0，使用 SCLKMPPLL

ONENAND\_SEL=1，使用 HCLK\_DSYS

所以 CLK\_SRC0=0x10001111；

### 3. main.c

在 main 函数中实现 LED 闪烁的功能，与前面的代码大同小异。

## 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 11.clock_c  
# make
```

在 11.clock\_c 目录下会生成 clock.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



点击“下载运行”，Minitools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

同样，我们可以看到 LED 闪烁，但是闪烁的频率并没有大幅度提升，这是因为 Superboot 已经帮我们初始化过时钟了。如果我们想体验一下不初始化系统时钟时开发板的运行速度的话，可以通过在 clock.c 中定义宏 PLL\_OFF 从而关闭 PLL 的功能，这样 LED 闪烁的频率将大大降低。本章的时钟设置主要是针对 APLL 和 MPLL，这足以让大部分硬件正常工作了，下一章我们将初始化串口以达到在终端输入输出字符的目的，这将使用到我们本章所设置好的时钟。

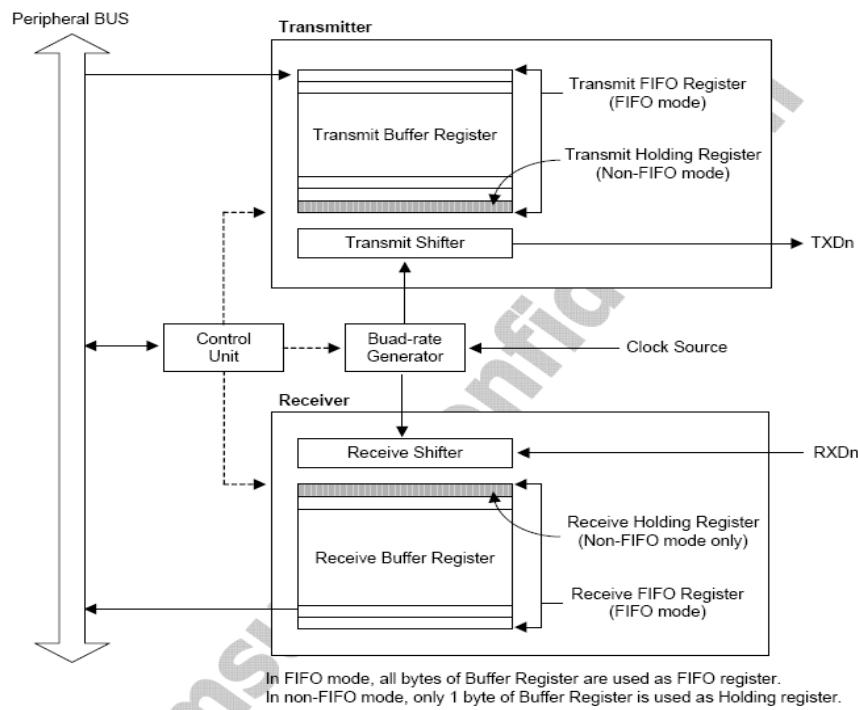
# 第十三章 串口设置之输入输出字符

## 第一节 S5PV210 UART 相关说明

通用异步收发器简称 UART，即 UNIVERSAL ASYNCHRONOUS RECEIVER AND TRANSMITTER，它用来传输串行数据。发送数据时，CPU 将并行数据写入 UART，UART 按照一定的格式在一根电线上串行发出；接收数据时，UART 检测另一根电线的信号，将串行收集在缓冲区中，CPU 即可读取 UART 获得这些数据。

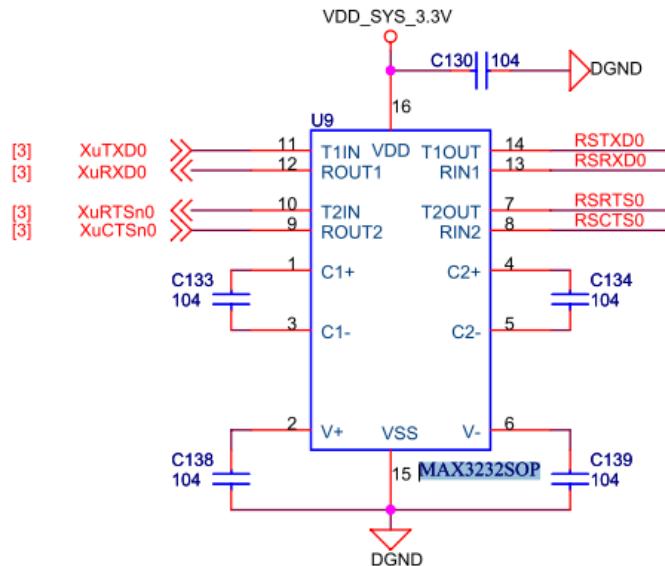
在 S5PV210 中，UART 提供了 4 对独立的异步串口 I/O 端口，有 4 个独立的通道，每个通道可以工作于 DMA 模式或者中断模式。其中，通道 0 有 256byte 的的发送 FIFO 和 256byte 的接收 FIFO，通道 1 有 64byte 的的发送 FIFO 和 64byte 的接收 FIFO，而通道 2 和 3 只有 16byte 的的发送 FIFO 和 16byte 的接收 FIFO。

S5PV210 的 uart 结构图如下：



S5PV210 的 uart 结构图

UART 使用标准的 TTL/CMOS 逻辑电平来表示数据，为了增强数据抗干扰能力和提高传输长度，通常将 TTL/CMOS 逻辑电平转换为 RS-232 逻辑电平，查看原理图可知 Mini210S 使用的是 MAX3232SOP 芯片，使用的是 TX0 和 DX0：



搜索“XuTXD0”，可知：

[9]	XuRXD0	C8	XuRXD0/GPA0_0
[9]	XuTXD0	D8	XuTXD0/GPA0_1
[9]	XuCTS0	D9	XuCTS0/GPA0_2
[9]	XuRTS0	A7	XuRTS0/GPA0_3
[9]	XuRXD1	G10	XuRXD1/GPA0_4
[9]	XuTXD1	F10	XuTXD1/GPA0_5
[9]	XuCTS1	B8	XuCTS1/GPA0_6
[9]	XuRTS1	E10	XuRTS1/GPA0_7
[9]	XuRXD2	AC20	XuRXD2/UART_AUDIO_RXD/GPA1_0
[9]	XuTXD2	AC14	XuTXD2/UART_AUDIO_TXD/GPA1_1
[9]	XuRXD3	AC13	XuRXD3/CTS02/UART_AUDIO_CTSn/GPA1_2
[9]	XuTXD3	AB13	XuTXD3/RTS02/UART_AUDIO_RTSn/GPA1_3

UART 引脚连接图

通过设置 UART 相关寄存器，我们就可以驱动 UART 工作，达到发送和接收字符的目的。

## 第二节 程序相关讲解

完整代码见目录 12. uart\_putchar，对比前一个目录 11. clock\_c，区别在于 main.c 和多了一个 uart.c 文件。

### 1. main.c

完整代码如下：

```
int main()
{
    char c;
    uart_init(); // 初始化串口

    while (1)
```



```
{  
    c = getc ();           // 接收一个字符 c  
    putc(c+1);           // 发送字符 c+1  
}  
return 0;  
}
```

在 main 函数中，先会调用 uart\_init() 初始化 UART，然后使用 getc 接收 PC 发过来的字符，再调用 putc() 将该字符+1 回复给 PC。

## 2. uart.c

uart\_init() 代码如下：

```
void uart_init()  
{  
  
    // 1 配置引脚用于 RX/TX 功能  
    GPA0CON = 0x22222222;  
    GPA1CON = 0x2222;  
  
    // 2 设置数据格式等  
    UFCON0 = 0x1;           // 使能 FIFO  
    UMCON0 = 0x0;           // 无流控  
    ULCON0 = 0x3;           // 数据位:8, 无校验, 停止位: 1  
    UCON0  = 0x5;           // 时钟: PCLK, 禁止中断, 使能 UART 发送、接收  
  
    // 3 设置波特率  
    UBRDIV0 = UART_UBRDIV_VAL;          // 35  
    UDIVSLOT0 = UART_UDIVSLOT_VAL;      // 0x1  
}
```

上述代码共有 3 个步骤，下面我们来一一讲解每一个步骤：

**第一步 配置引脚用于 RX/TX 功能**

参考 UART 引脚连接图，我们需要设置 GPA0CON 和 GPA1CON 寄存器使 GPA0 和 GPA1 引脚用于 UART 功能。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 2.2.2.1 Port Group GPA0 Control Register (GPA0CON, R/W, Address = 0xE020\_0000)

GPA0CON	Bit	Description	Initial State
GPA0CON[7]	[31:28]	0000 = Input 0001 = Output 0010 = UART_1_RTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[7]	0000
GPA0CON[6]	[27:24]	0000 = Input 0001 = Output 0010 = UART_1_CTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[6]	0000
GPA0CON[5]	[23:20]	0000 = Input 0001 = Output 0010 = UART_1_TXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[5]	0000
GPA0CON[4]	[19:16]	0000 = Input 0001 = Output 0010 = UART_1_RXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[4]	0000
GPA0CON[3]	[15:12]	0000 = Input 0001 = Output 0010 = UART_0_RTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[3]	0000
GPA0CON[2]	[11:8]	0000 = Input 0001 = Output 0010 = UART_0_CTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[2]	0000
GPA0CON[1]	[7:4]	0000 = Input 0001 = Output 0010 = UART_0_TXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[1]	0000
GPA0CON[0]	[3:0]	0000 = Input 0001 = Output 0010 = UART_0_RXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[0]	0000

GPA0CON 寄存器图

## 2.2.3.1 Port Group GPA1 Control Register (GPA1CON, R/W, Address = 0xE020\_0020)

GPA1CON	Bit	Description	Initial State
GPA1CON[3]	[15:12]	0000 = Input 0001 = Output 0010 = UART_3_TXD 0011 = UART_2_RTSn 0100 ~ 1110 = Reserved 1111 = GPA1_INT[3]	0000
GPA1CON[2]	[11:8]	0000 = Input 0001 = Output 0010 = UART_3_RXD 0011 = UART_2_CTSn 0100 ~ 1110 = Reserved 1111 = GPA1_INT[2]	0000
GPA1CON[1]	[7:4]	0000 = Input 0001 = Output 0010 = UART_2_TXD 0011 = Reserved 0100 = UART_AUDIO_TXD 0101 ~ 1110 = Reserved 1111 = GPA1_INT[1]	0000
GPA1CON[0]	[3:0]	0000 = Input 0001 = Output 0010 = UART_2_RXD 0011 = Reserved 0100 = UART_AUDIO_RXD 0101 ~ 1110 = Reserved 1111 = GPA1_INT[0]	0000

GPA1CON 寄存器图



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第二步 设置数据格式等

### <1> ULCNO 用来设置数据格式，见下图

There are four UART line control registers in the UART block, namely, ULCNO, ULCN1, ULCN2, and ULCN3.

ULCOn	Bit	Description	Initial State
Reserved	[31:7]	Reserved	0
Infrared Mode	[6]	Determines whether to use the Infrared mode. 0 = Normal mode operation 1 = Infrared Tx/Rx mode	0
Parity Mode	[5:3]	Specifies the type of parity generation to be performed and checking during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/ checked as 1 111 = Parity forced/ checked as 0	000
Number of Stop Bit	[2]	Specifies how many stop bits are used to signal end-of-frame signal. 0 = One stop bit per frame 1 = Two stop bit per frame	0
Word Length	[1:0]	Indicates the number of data bits to be transmitted or received per frame. 00 = 5-bit 01 = 6-bit 10 = 7-bit 11 = 8-bit	00

- ✚ Word Length = 11, 8bit 的数据；
- ✚ Number of Stop Bit = 0, 1bit 的停止位；
- ✚ Parity Mode = 000, 无校验；
- ✚ Infrared Mode = 0, 使用普通模式；

所以 ULCNO=0x3

### <2> 9UCONO 是 UART 的配置寄存器，见下图



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

There are four UART control registers in the UART block, namely, UCON0, UCON1, UCON2 and UCON3.

UCONn	Bit	Description	Initial State
Reserved	[31:21]	Reserved	000
Tx DMA Burst Size	[20]	Tx DMA Burst Size 0 = 1 byte (Single) 1 = 4 bytes	0
Reserved	[19:17]	Reserved	000
Rx DMA Burst Size	[18]	Rx DMA Burst Size 0 = 1 byte (Single) 1 = 4 bytes	0
Reserved	[15:11]	Reserved	0000
Clock Selection	[10]	Selects PCLK or SCLK_UART (from Clock Controller) clock for the UART baud rate.  0 = PCLK: DIV_VAL1 = (PCLK / (bps x 16)) -1 1 = SCLK_UART: DIV_VAL1 = (SCLK_UART / (bps x 16)) -1	00
Tx Interrupt Type	[9]	Interrupt request type.(2) 0 = Pulse (Interrupt is requested when the Tx buffer is empty in the Non-FIFO mode or when it reaches Tx FIFO Trigger Level in the FIFO mode.) 1 = Level (Interrupt is requested when Tx buffer is empty in the Non-FIFO mode or when it reaches Tx FIFO Trigger Level in the FIFO mode.)	0
Rx Interrupt Type	[8]	Interrupt request type. (2) 0 = Pulse (Interrupt is requested when instant Rx buffer receives data in the Non-FIFO mode or when it reaches Rx FIFO Trigger Level in the FIFO mode.) 1 = Level (Interrupt is requested when Rx buffer is receiving data in the Non-FIFO mode or when it reaches Rx FIFO Trigger Level in the FIFO mode.)	0
Rx Time Out Enable	[7]	Enables/ Disables Rx time-out interrupts if UART FIFO is enabled. The interrupt is a receive interrupt. 0 = Disables 1 = Enables	0
Rx Error Status Interrupt Enable	[6]	Enables the UART to generate an interrupt upon an exception, such as a break, frame error, parity error, or overrun error during a receive operation. 0 = Does not generate receive error status interrupt. 1 = Generates receive error status interrupt.	0

UCONn	Bit	Description	Initial State
Loop-back Mode	[5]	Setting loop-back bit to 1 trigger the UART to enter the loop-back mode. This mode is provided for test purposes only. 0 = Normal operation 1 = Loop-back mode	0
Send Break Signal	[4]	Setting this bit trigger the UART to send a break during 1 frame time. This bit is automatically cleared after sending the break signal. 0 = Normal transmit 1 = Sends the break signal	0
Transmit Mode	[3:2]	Determines which function is able to write Tx data to the UART transmit buffer register. 00 = Disables 01 = Interrupt request or polling mode 10 = DMA mode 11 = Reserved	00
Receive Mode	[1:0]	Determines which function is able to read data from UART receive buffer register. 00 = Disables 01 = Interrupt request or polling mode 10 = DMA mode 11 = Reserved	00

NOTE:

地址 : 广州市天河区龙口西路龙苑大厦A1栋1705 网址 : <http://www.arm9.net>

电话 : +86-20-85201025(售前、售后咨询) 技术支持(Tel): 13719442657 传真 : +86-20-85261505

E-Mail: [capbily@163.com](mailto:capbily@163.com)(商务或项目合作) [dev\\_friendlyarm@163.com](mailto:dev_friendlyarm@163.com) (技术支持)



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

- Receive Mode = 01 , 使用中断模式或者轮询模式;
  - Transmit Mode = 01, 使用中断模式或者轮询模式;
  - Send Break Signal = 0, 普通传输;
  - Loop-back Mode = 0, 不使用回环方式;
  - 我们采用轮询的方式接受和发送数据, 不使用中断, 所以 bit[6-9]均为 0;
  - Clock Selection = 0, 使用 PCLK 作为 UART 的工作时钟;
  - 我们不使用 DMA, 所以 bit[16]和 bit[20]均为 0;
- 所以 UCON0 = 0x5

### <3> UFCON0 和 UMCON0

这两个寄存器比较简单, UFCON0 用来使能 FIFO, UMCON0 用来设置无流控。

### 第三步 设置波特率

波特率即每秒传输的数据位数, 涉及两个寄存器: UBRDIV0 和 UDIVSLOT0

UBRDIV n	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
UBRDIVn	[15:0]	Baud rate division value (When UART clock source is PCLK, UBRDIVn must be more than 0 (UBRDIVn >0))	0x0000

UDIVSLOT n	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
UDIVSLOTn	[15:0]	Select the slot where clock generator divide clock source	0x0000

波特率设置相关公式: UBRDIVn + (num of 1's in UDIVSLOTn)/16 = (PCLK / (bps x 16)) -1  
其中, 由 Maximum Operating Frequency for Each Sub-block 图可知, UART 工作于 PSYS 下, 所以 PCLK 即 PCLK\_PSYS = 66.5MHz, 我们的波特率 bps 设置为 115200, 所以  
 $(66.5\text{MHz}/(115200 \times 16)) - 1 = 35.08 = UBRDIVn + (\text{num of 1's in UDIVSLOTn})/16$ , 所以我们设置 UBRDIV0=35, UDIVSLOT0=0x1

getc() 和 putc() 的代码如下:

```
// 接收一个字符
char getc(void)
{
    while (((UFSTAT0 & 0xff) == 0); // 如果 RX FIFO 空, 等待
    return URXH0; // 取数据
}

// 发送一个字符
void putc(char c)
{
    while ((UFSTAT0 & (1<<24)); // 如果 TX FIFO 满, 等待
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
UTXH0 = c; // 写数据  
}
```

There are four UART transmit buffer registers in the UART block, namely, UTXH0, UTXH1, UTXH2 and UTXH3. UTXHn contains 8-bit data for transmission data.

UTXHn	Bit	Description	Initial State
Reserved	[31:8]	Reserved	-
UTXHn	[7:0]	Transmit data for UARTn	-

### UART 数据发送寄存器

There are four UART receive buffer registers in the UART block, namely, URXH0, URXH1, URXH2 and URXH3. URXHn contains 8-bit data for received data.

URXHn	Bit	Description	Initial State
Reserved	[31:8]	Reserved	0
URXHn	[7:0]	Receive data for UARTn	0x00

### UART 数据接收寄存器

UTRSTATn	Bit	Description	Initial State
Reserved	[31:3]	Reserved	0
Transmitter empty	[2]	This bit is automatically set to 1 if the transmit buffer register has no valid data to transmit, and the transmit shift register is empty. 0 = Not empty 1 = Transmitter (which includes transmit buffer and shifter register) empty	1
Transmit buffer empty	[1]	This bit is automatically set to 1 if transmit buffer register is empty. 0 = Buffer register is not empty 1 = Buffer register is empty (In Non-FIFO mode, Interrupt or DMA is requested. In FIFO mode, Interrupt or DMA is requested, if Tx FIFO Trigger Level is set to 00 (Empty)) If UART uses FIFO, check Tx FIFO Count bits and Tx FIFO Full bit in UFSTAT register instead of this bit.	1
Receive buffer data ready	[0]	This bit is automatically set to 1 if receive buffer register contains valid data, received over the RXDn port. 0 = Buffer register is empty 1 = Buffer register has a received data (In Non-FIFO mode, Interrupt or DMA is requested) If UART uses the FIFO, check Rx FIFO Count bits and Rx FIFO Full bit in UFSTAT register instead of this bit.	0

### 发送/接收状态寄存器

通过读 UTRSTAT0 发送/接收状态寄存器，当 Receive buffer data ready= 1 时说明接收到数据，读 URXH0 寄存器可以得到 8bit 的数据；当 Transmitter empty = 1 时说明可以发送数据，写 8bit 的数据到 UTXH0。

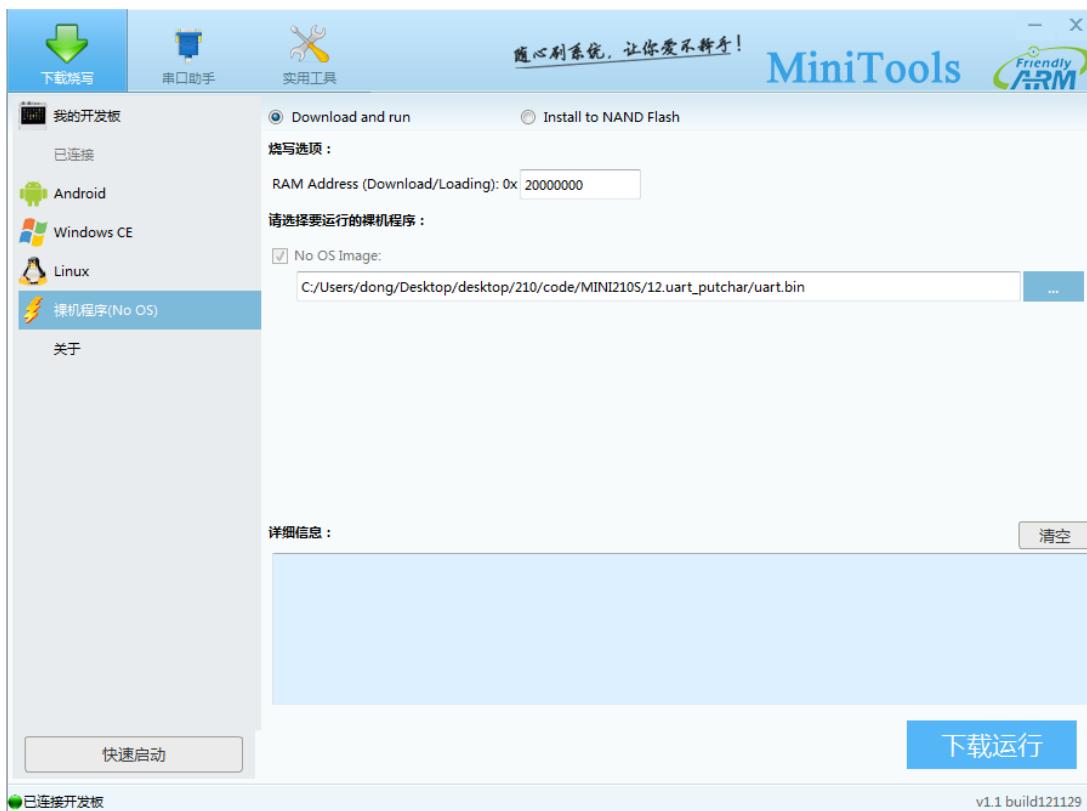
## 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 12.uart_putchar  
# make
```

在 12.uart\_putchar 目录下会生成 uart.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，Minitools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

先连接好串口线，并通过 MiniTools 自带的串口助手打开串口（**注意要设置好波特率等参数，如图**），然后从 PC 键盘中敲入一个字符，则串口终端会显示该字符在 ASCII 表中的下一字符，如输入‘a’，串口终端会出现‘b’，效果如下：

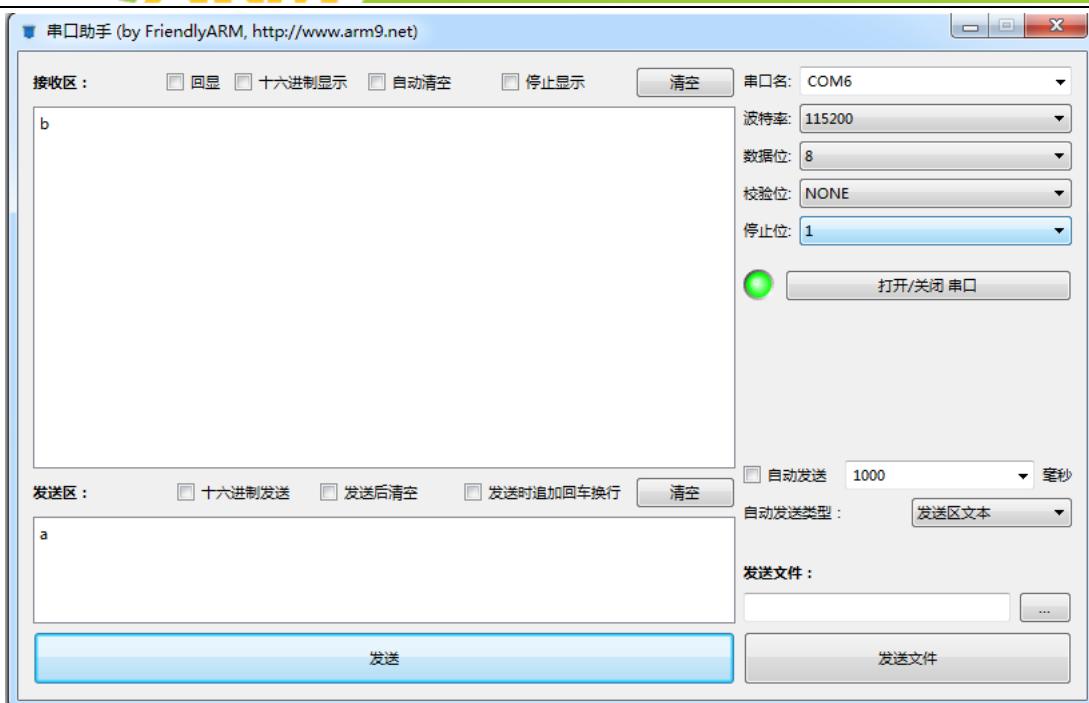


追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



通过本章对 UART 的初始化，我们已经能在终端上打印字符了，这对我们调试代码有极大的帮助。但是目前代码的功能太单一，只能简单地输入或者输出一个字符，在后面的章节中，我们将利用 `putc()` 和 `getc()` 这两个函数，为程序添加功能强大的 `printf()` 和 `scanf()` 功能。



# 第十四章 移植 printf 和 scanf 功能

## 第一节 移植的途径

对于如何移植 printf 和 scanf，我们有许多选择：

- 1) 移植 linux 的 printk 功能，版本越新越难移植，但是功能也越强大；
- 2) 移植 uboot 的 printf 和 scanf 功能，实际 uboot 也是从 linux 内核中移植而来的；
- 3) 完全自己编写，但是功能比较弱；

在保证整个裸机其他代码部分没有任何问题，且编译器也没有任何问题的情况下，上述三种方法都是可行的。下面我们只是直接利用网友从 linux 中移植好的 printk，为我们的裸机代码增加上该部分功能。

## 第二节 移植步骤

**第一步** 解压 printf.rar 到 13 uart\_stdio 目录，解压成功后会多出 include 和 lib 两个目录，其中 include 放的是相关头文件，lib 放的是 printf 和 scanf 相关的代码。

**第二步** 修改 13 uart\_stdio 目录下的 makefile，将 lib 目录下的代码编译链接成 lib.a，然后将 lib.a 编译进 bin 中，具体修改见源码。

**第三步** 编写 main 函数进行测试。

## 第三节 程序相关讲解

完整代码见目录 13 uart\_stdio，与前一章的代码相比，BL1 目录的代码没有任何变化，BL2 目录的代码多了 include 和 lib 目录以及 main.c 的内容被修改了。

### 1. /lib/printf.c

<1> printf 的定义如下：

```
int printf(const char *fmt, ...)  
{  
    int i;  
    int len;  
    va_list args;          // va_list 即 char *  
  
    va_start(args, fmt);  
    len = vsprintf(g_PCOutBuf, fmt, args); // 内部使用了 va_arg()
```



```
va_end(args);
for (i = 0; i < strlen(g_PCOutBuf); i++)
{
    putc(g_PCOutBuf[i]);
}
return len;
}
```

#### <2> printf 函数是个变参函数，什么是变参函数：

可变参数函数的原型声明为 type VAFunction(type arg1, type arg2, ... )；参数可以分为两部分：个数确定的固定参数和个数可变的可选参数。函数至少需要一个固定参数，固定参数的声明和普通函数一样；可选参数由于个数不确定，声明时用“...”表示。固定参数和可选参数共同构成一个函数的参数列表。

#### <3> printf 函数涉及了 3 个十分重要的宏：

宏1: #define va\_start(ap, A) (void) ((ap) = (((char \*) &(A)) + (\_bnd (A, \_AUPBND))) )

宏2: #define va\_arg(ap, T) (\* (T \*) ((ap) += (\_bnd (T, \_AUPBND))) - (\_bnd (T, \_ADNBND))) )

宏3: #define va\_end(ap) (void) 0

在这些宏中，va 就是 variable argument(可变参数)的意思；

ap:是指向可变参数表的指针；

A:指可变参数表的前一个固定参数；

T:可变参数的类型。

va\_list 也是一个宏，其定义为 typedef char \* va\_list，实质上是一 char 型指针。

#### <4> 三个宏的作用：

##### 1) va\_start 宏

作用：

根据 v 取得可变参数表的首指针并赋值给 ap，方法：最后一个固定参数 A 的地址 + 第一个变参对 A 的偏移地址，然后赋值给 ap，这样 ap 就是可变参数表的首地址。

举例：

如果有变参函数的声明是 void va\_test(char a, char b, char c, ... )，则它的固定参数依次是 a, b, c，最后一个固定参数为 c，因此就是 va\_start(ap, c)。

##### 2) va\_arg 宏

作用：

指取出当前 ap 所指的可变参数并将 ap 指针指向下一可变参数。

##### 3) va\_end 宏

作用：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

结束可变参数的获取。va\_end (list) 实际上被定义为空，没有任何真实对应的代码，用于代码对称，与 va\_start 对应。

#### <5> 得到可变参数个数的三种办法：

- 1) 函数的第一个参数, 指定后续的参数个数, 如 func(int num, ...);
- 2) 根据隐含参数, 判断参数个数, 如 printf 系列的, 通过字符串中%的个数判断;
- 3) 特殊情况下(如参数都是不大于 0xFFFF 的 int), 可以一直向低处访问堆栈, 直到返回地址。

有了上述知识我们就可以看懂 printf() 函数的内容了, 首先 va\_start(args, fmt); 会将可变参数的首地址保存在 args 中, 然后调用 vsprintf(g\_PCOutBuf, fmt, args) 进行处理, 在 vsprintf() 中, 会调用 va\_arg() 逐个的取出变参, 然后进行解析。如果是普通字符则无须转换, 直接保存在 g\_PCOutBuf; 如果是字符串, 则从可变参数表中拿到指向字符串的指针, 将字符串的内容拷贝到 g\_PCOutBuf; 如果是数字, 则调用 number 函数进行处理, 并把解析的结果存放在 g\_PCOutBuf。所有, 最后只调用 putC 函数把 g\_PCOutBuf 里的字符一个个的打印出来就可以了。scanf 函数的原理和 printf 类似, 这里不再进行解释

## 2. main.c

完整代码如下:

```
int main()
{
    int a = 0;
    int b = 0;
    char *str = "hello world";

    uart_init();
    printf("%s\n", str);
    while (1)
    {
        printf("please enter two number: \r\n");
        scanf("%d %d", &a, &b);
        printf("\r\n");
        printf("the sum is: %d\r\n", a+b);
    }
    return 0;
}
```

首先会打印 “hello world” , 然后从串口接收两个数字, 最后输出它们的和。

## 第四节 编译代码和烧写运行

编译代码, 在 Fedora 终端执行如下命令:

---

地址 : 广州市天河区龙口西路龙苑大厦A1栋1705 网址 : <http://www.arm9.net>

电话 : +86-20-85201025(售前、售后咨询) 技术支持(Tel): 13719442657 传真 : +86-20-85261505

E-Mail: [capbily@163.com](mailto:capbily@163.com)(商务或项目合作) [dev\\_friendlyarm@163.com](mailto:dev_friendlyarm@163.com) (技术支持)



追求卓越 创造精品

TO BE BEST

TO DO GREAT

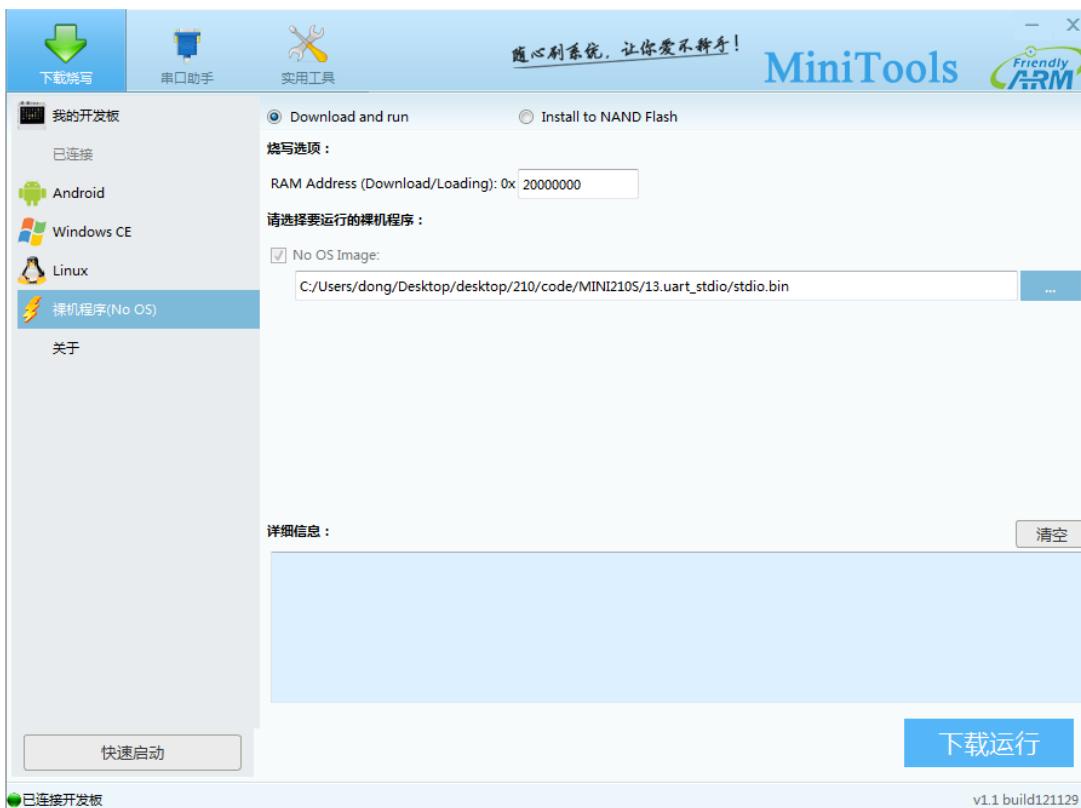
广州友善之臂计算机科技有限公司

```
# cd 13 uart_stdio
```

```
# make
```

在 13 uart\_stdio 目录下会生成 stdio.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第五节 实验现象

先连接好串口线，并通过 secure CRT 或者超级终端等串口工具打开串口，可以看到串口终端处先会打印“hello world”，接着提醒你输入两个数字，成功输入两个数字后，会打印它们的和。效果如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
hello world
please enter two number:
1 2
the sum is: 3
please enter two number:
3 4
the sum is: 7
please enter two number:
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第十五章 NAND Flash 的读写擦除

### 第一节 关于 NAND Flash

S5PV210 的 NAND Flash 控制器有如下特点：

- 1) 支持 512byte, 2k, 4k, 8k 的页大小
- 2) 通过各种软件模式来进行 NAND Flash 的读写擦除等
- 3) 8bit 的总线
- 4) 支持 SLC 和 MCL 的 NAND Flash
- 5) 支持 1/4/8/12/16bit 的 ECC
- 6) 支持以字节/半字/字为单位访问数据/ECC 寄存器，以字为单位访问其他寄存器。

注意：在此使用的 Mini210S 的 NAND Flash 类型为 SLC，大小为 1G，型号为 K9K8G08U0A。所以本章的内容是针对 SLC 类型的 NAND Flash(包括 256M/512M/1GB 等)，并不适用 MLC 类型的 NAND Flash。

### 第二节 程序相关讲解

完整代码见目录 14. nand，与上一章相比，多了 nand.c 这个文件，里面包含了对 NAND Flash 的相关操作。

#### 1. nand.c

<1> NAND Flash 初始化函数 nand\_init(), 代码如下

```
void nand_init(void)
{
    // 1. 配置 NAND Flash
    NFCNF
    = (TACLS<<12) | (TWRPH0<<8) | (TWRPH1<<4) | (0<<3) | (0<<2) | (1<<1) | (0<<0) ;
    NFCONT
    =(0<<18) | (0<<17) | (0<<16) | (0<<10) | (0<<9) | (0<<8) | (0<<7) | (0<<6) | (0x3<<1) | (1<<0) ;

    // 2. 配置引脚
    MP0_1CON = 0x22333322;
    MP0_2CON = 0x00002222;
    MP0_3CON = 0x22222222;

    // 3. 复位
    nand_reset();
}
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

}

共 3 个步骤：

### 第一步 配置 NAND Flash

主要是设置 NFCNF 和 NFCONT 两个寄存器

4.5.2.1 Nand Flash Configuration Register (NFCNF, R/W, Address = 0xB0E0\_0000)

NFCNF	Bit	Description	Initial State
Reserved	[31:26]	Reserved	0
MsgLength	[25]	0 = 512 byte Message Length 1 = 24 byte Message Length	0
ECCType0	[24:23]	This bit indicates the kind of ECC to use. 00 = 1-bit ECC 10 = 4-bit ECC 01 = 11 = Disable 1-bit and 4-bit ECC	0
Reserved	[22:16]	Reserved	0000000
TACLS	[15:12]	CLE and ALE duration setting value (0~15) Duration = HCLK x TACLS	0x1
TWRPH0	[11:8]	TWRPH0 duration setting value (0~15) Duration = HCLK x (TWRPH0 + 1) Note: You should add additional cycles about 10ns for page read because of additional signal delay on PCB pattern.	0x0
TWRPH1	[7:4]	TWRPH1 duration setting value (0~15) Duration = HCLK x (TWRPH1 + 1)	0x0
MLCFlash	[3]	This bit indicates the kind of NAND Flash memory to use. 0 = SLC NAND Flash 1 = MLC NAND Flash	0
PageSize	[2]	This bit indicates the page size of NAND Flash Memory. When MLCFlash is 0, the value of PageSize is as follows: 0 = 2048 Bytes/page 1 = 512 Bytes/page When MLCFlash is 1, the value of PageSize is as follows: 0 = 4096 Bytes/page 1 = 2048 Bytes/page	0
AddrCycle	[1]	This bit indicates the number of Address cycle of NAND Flash memory. When Page Size is 512 Bytes, 0 = 3 address cycle 1 = 4 address cycle When page size is 2K or 4K, 0 = 4 address cycle 1 = 5 address cycle	0
Reserved	[0]	Reserved	0

### NFCNF 寄存器

- AddrCycle = 1, When page size is 2K or 4K, 1 = 5 address cycle, Mini210S 的 NAND Flash 的页大小为 2k, 所有是 5 个地址周期;
- PageSize = 0, When MLCFlash is 0, the value of PageSize is as follows: 0 = 2048 Bytes/page, Mini210S 使用的是 SLC NAND Flash 且每页大小为 2k;
- MLCFlash = 0, 在此使用的是 SLC NAND Flash;
- TWRPH1/TWRPH0/TACLS 是关于访问时序的设置, 需对照 NAND Flash 芯片手册设置, 这里不再详细解释, 分别取 TWRPH1=1, TWRPH0=4, TACLS=1;
- ECCType0/MsgLength, 我们的裸机代码没有使用到 ECC, 所有不用设置这两个标志。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 4.5.2.2 Control Register (NFCONT, R/W, Address = 0xB0E0\_0004)

NFCONT	Bit	Description	Initial State
Reserved	[31:24]	Reserved	0
Reg_nCE3	[23]	NAND Flash Memory nRCS[3] signal control 0 = Force nRCS[3] to low (Enable chip select) 1 = Force nRCS[3] to High (Disable chip select)	1
Reg_nCE2	[22]	NAND Flash Memory nRCS[2] signal control 0 = Force nRCS[2] to low (Enable chip select) 1 = Force nRCS[2] to High (Disable chip select)	1
Reserved	[21:19]	Reserved	0
MLCEccDirection	[18]	4-bit, ECC encoding / decoding control 0 = Decoding 4-bit ECC, It is used for page read 1 = Encoding 4-bit ECC, It is be used for page program	0
LockTight	[17]	Lock-tight configuration 0 = Disable lock-tight 1 = Enable lock-tight, If this bit is setto 1, you cannot clear this bit. For more information, refer to the <a href="#">4.3.12 "Lock scheme for data protection".</a>	0
LOCK	[16]	Soft Lock configuration 0 = Disable lock 1 = Enable lock Software can modify soft lock area any time. For more information, refer to the <a href="#">4.3.12 "</a> .	1
Reserved	[15:14]	Reserved	00
EnbMLCEncInt	[13]	4-bit ECC encoding completion interrupt control 0 = Disable interrupt 1 = Enable interrupt	0
EnbMLCDecInt	[12]	4-bit ECC decoding completion interrupt control 0 = Disable interrupt 1 = Enable interrupt	0
	[11]	Reserved	0
EnbIllegalAccINT	[10]	Illegal access interrupt control 0 = Disable interrupt 1 = Enable interrupt Illegal access interrupt occurs when CPU tries to program or erase locking area (the area setting in NFSBLK (0xB0E0_0020) to NFEBLK (0xB0E0_0024)-1.	0
EnbRnBINT	[9]	RnB status input signal transition interrupt control 0 = Disable RnB interrupt 1 = Enable RnB interrupt	0
RnB_TransMode	[8]	RnB transition detection configuration 0 = Detect rising edge 1 = Detect falling edge	0



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

NFCONT	Bit	Description	Initial State
MECClock	[7]	Lock Main area ECC generation 0 = Unlock Main area ECC 1 = Lock Main area ECC  Main area ECC status register is NFMECC0/NFMECC1(0xB0E0_0034/0xB0E0_0038),	1
SECClock	[6]	Lock Spare area ECC generation. 0 = Unlock Spare ECC 1 = Lock Spare ECC  Spare area ECC status register is NFSECC(0xB0E0_003C),	1
InitMECC	[5]	1 = Initialize main area ECC decoder/encoder (write-only)	0
InitSECC	[4]	1 = Initialize spare area ECC decoder/encoder (write-only)	0
HW_nCE	[3]	Reserved (HW_nCE)	0
Reg_nCE1	[2]	NAND Flash Memory nRCS[1] signal control	1
Reg_nCE0	[1]	NAND Flash Memory nRCS[0] signal control 0 = Force nRCS[0] to low (Enable chip select) 1 = Force nRCS[0] to High (Disable chip select)  Note: The setting all nCE[3:0] zero can not be allowed. Only one nCE can be asserted to enable external NAND flash memory. The lower bit has more priority when user set all nCE[3:0] zeros.	1
MODE	[0]	NAND Flash controller operating mode 0 = Disable NAND Flash Controller 1 = Enable NAND Flash Controller	0

### NFCONT 寄存器

- MODE = 1, 使能 NAND Flash 控制器;
- Reg\_nCE0 = 1, 取消片选, 需要操作 NAND Flash 时再发片选;
- Reg\_nCE1 = 1, 取消片选, 需要操作 NAND Flash 时再发片选;
- InitMECC/InitSECC/SECClock/MECClock, 我们的裸机代码不涉及 ECC, 这 4 个标志位随便设置即可;
- RnB\_TransMode = 0, Detect rising edge, RnB 是 NAND Flash 的状态探测引脚, 我们使用上升沿触发;
- EnbRnBINT = 0 , 禁止 RnB 中断;
- EnbIllegalAccINT = 0, 禁止 Illegal access 中断 ;
- EnbMLCDecInt/EnbMLCEncInt 为 MCL 相关, 不用设置;
- LOCK = 0, 我们没有用到 Soft Lock, 所以禁止 Soft Lock;
- LockTight = 0, 我们没有用到 Lock-tight, 所有禁止 Lock-tight;
- MLCEccDirection, MLC 相关, 可不用设置

### 第二步 配置引脚

用于 NAND Flash 相关功能;

### 第三步 复位

复位函数 nand\_reset 的相关代码如下:

```
static void nand_reset(void)
{
    nand_select_chip();
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
nand_send_cmd(NAND_CMD_RES);  
nand_wait_idle();  
nand_deselect_chip();  
}
```

NAND Flash 的复位操作共 4 个步骤:

- 1) 发片选, 实质就是 NFCONT &= ~(1<<1); 往 NFCONT 的 bit[1]写 0;
- 2) 发命令复位命令 NAND\_CMD\_RES (0xff); 实质就是 NFCMMD = cmd; 将命令写到 NFCMMD 寄存器; 完整的 NAND Flash 命令信息见下图:

Table 1. Command Sets

Function	1st Cycle	2nd Cycle	Acceptable Command during Busy
Read	00h	30h	
Read for Copy Back	00h	35h	
Read ID	90h	-	
Reset	FFh	-	O
Page Program	80h	10h	
Two-Plane Page Program <sup>(4)</sup>	80h--11h	81h--10h	
Copy-Back Program	85h	10h	
Two-Plane Copy-Back Program <sup>(4)</sup>	85h--11h	81h--10h	
Block Erase	60h	D0h	
Two-Plane Block Erase	60h--60h	D0h	
Random Data Input <sup>(1)</sup>	85h	-	
Random Data Output <sup>(1)</sup>	05h	E0h	
Read Status	70h		O
Read EDC Status <sup>(2)</sup>	7Bh		O
Chip1 Status <sup>(3)</sup>	F1h		O
Chip2 Status <sup>(3)</sup>	F2h		O

- 3) 等待 NAND Flash 就绪; 实质就是 while( !(NFSTAT & (BUSY<<4)) ), 读 NFSTAT 的 bit[4] 检查 NAND Flash 是否就绪;
- 4) 取消片选, 实质就是 NFCONT |= (1<<1); 往 NFCONT 的 bit[1]写 1;

<2> NAND Flash 读 ID 函数 nand\_read\_id(), 代码如下

```
void nand_read_id(void)  
{  
    nand_id_info nand_id;  
    // 1. 发片选  
    nand_select_chip();  
  
    // 2. 读 ID  
    nand_send_cmd(NAND_CMD_READ_ID);  
    nand_send_addr(0x00);  
    nand_wait_idle();  
    nand_id.IDm = nand_read();  
    nand_id.IDd = nand_read();
```

```

nand_id.ID3rd = nand_read();
nand_id.ID4th = nand_read();
nand_id.ID5th = nand_read();
    
```

```

printf("NANDFlash: makercode = %x, devicecode = %x\r\n", nand_id.IDm, nand_id.IDd);
nand_deselect_chip();
}
    
```



Device	Device Code(2nd Cycle)	3rd Cycle	4th Cycle	5th Cycle
K9K8G08U0A	D3h	51h	95h	58h
K9WAG08U1A			Same as K9K8G08U0A in it	
K9NBBG08U5A				

### NAND Flash 读 ID 操作

根据上图, NAND Flash 的读 ID 操作共 4 个步骤:

第一步 发片选;

第二步 发读 ID 命令 NAND\_CMD\_READ\_ID(0x90);

第三步 发地址 0x00; 调用函数 nand\_send\_addr();

第四步 等待 NAND Flash 就绪;

第五步 读 ID; 调用了 nand\_read() 函数, 实质就是读 NFDATA 寄存器;

下面解释一下函数 nand\_send\_addr(), 核心代码如下:

```

{
    // 列地址, 即页内地址
    col = addr % NAND_PAGE_SIZE;
    // 行地址, 即页地址
    row = addr / NAND_PAGE_SIZE;

    // Column Address A0~A7
    NFADDR = col & 0xff;
    for(i=0; i<10; i++);

    // Column Address A8~A11

    NFADDR = (col >> 8) & 0x0f;
    for(i=0; i<10; i++);

    // Row Address A12~A19
    NFADDR = row & 0xff;
}
    
```

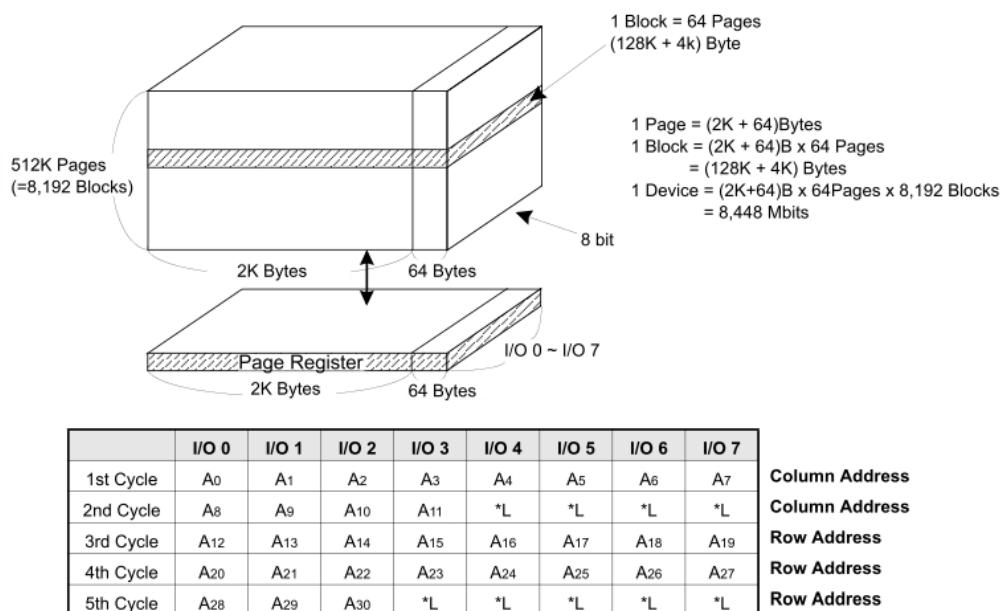
```

for(i=0; i<10; i++);

// Row Address A20~A27
NFADDR = (row >> 8) & 0xff;
for(i=0; i<10; i++);

// Row Address A28~A30
NFADDR = (row >> 16) & 0xff;
for(i=0; i<10; i++);
}
    
```

首先根据页大小来获取页地址和页内偏移地址，然后通过 5 个周期将地址发送出去，实质就是写 NFADDR 寄存器，具体每个周期如何发送，查阅 NAND Flash 芯片手册可知，见下图：



发送地址后，就可以连续读出 5 个 ID 了，其中第一个是 MAKDER CODE，第二个是 DEVICE CODE。

<3> NAND Flash 擦除函数 nand\_erase()，核心代码如下：

```

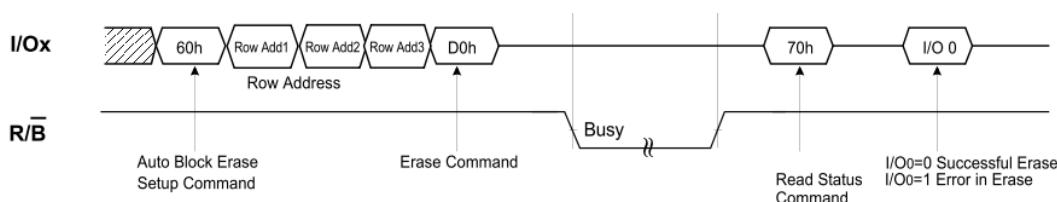
{
    // 获得 row 地址，即页地址
    unsigned long row = block_num * NAND_BLOCK_SIZE;

    // 1. 发出片选信号
    nand_select_chip();
    // 2. 擦除：第一个周期发命令 0x60，第二个周期发块地址，第三个周期发命令 0xd0
    nand_send_cmd(NAND_CMD_BLOCK_ERASE_1st);
}
    
```

```

for(i=0; i<10; i++);
// Row Address A12~A19
NFADDR = row & 0xff;
for(i=0; i<10; i++);
// Row Address A20~A27
NFADDR = (row >> 8) & 0xff;
for(i=0; i<10; i++);
// Row Address A28~A30
NFADDR = (row >> 16) & 0xff;
NFSTAT = (NFSTAT) | (1<<4);
nand_send_cmd(NAND_CMD_BLOCK_ERASE_2st);
for(i=0; i<10; i++);
// 3. 等待就绪
nand_wait_idle();

// 4. 读状态
unsigned char status = read_nand_status();
}
    
```



根据上图, NAND Flash 的擦除操作共 6 个步骤:

第一步 发片选;

第二步 发擦除命令 1 NAND\_CMD\_BLOCK\_ERASE\_1 (0x60);

第三步 发页地址, 只需发页地址;

第四步 发擦除命令 2 NAND\_CMD\_BLOCK\_ERASE\_2st (0xD0);

第五步 等待 NAND Flash 就绪;

第六步 读状态, 判断擦除是否成功。若擦除失败, 则打印是坏块再取消片选; 否则直接直接取消片选即可。读状态调用了函数 read\_nand\_status(), 它的实质就是 nand\_send\_cmd(NAND\_CMD\_READ\_STATUS);ch = nand\_read(); 先发读状态命令 NAND\_CMD\_READ\_STATUS, 然后再读状态值。

<4> NAND Flash 读函数 copy\_nand\_to\_sdram(), 从 NAND Flash 中读数据到 DRAM, 核心代码如下:

```

{
    // 1. 发出片选信号
    nand_select_chip();
}
    
```

```

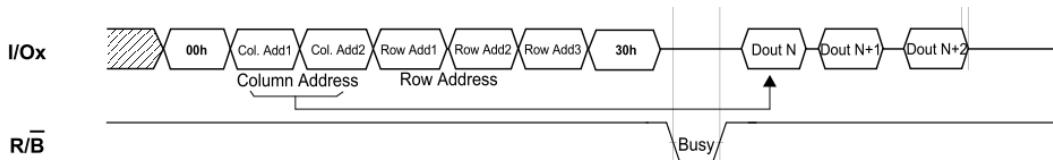
// 2. 从 nand 读数据到 sdram, 第一周期发命令 0x00, 第二周期发地址 nand_addr, 第三个
// 周期发命令 0x30, 可读一页(2k)的数据
while(length)
{
    nand_send_cmd(NAND_CMD_READ_1st);
    nand_send_addr(nand_addr);
    NFSTAT = (NFSTAT) | (1<<4);
    nand_send_cmd(NAND_CMD_READ_2st);

    nand_wait_idle();

    // 列地址, 即页内地址
    unsigned long col = nand_addr % NAND_PAGE_SIZE;
    i = col;
    // 读一页数据, 每次拷 1byte, 共拷 2048 次(2k), 直到长度为 length 的数据拷贝完毕
    for(; i<NAND_PAGE_SIZE && length!=0; i++, length--)
    {
        *sdram_addr = nand_read();
        sdram_addr++;
        nand_addr++;
    }
}

// 3. 读状态
unsigned char status = read_nand_status();
}

```



NAND Flash 读操作

根据上图, NAND Flash 的读操作共 7 个步骤:

- 第一步 发片选;
- 第二步 发读命令 1 NAND\_CMD\_READ\_1st (0x00);
- 第三步 发地址, 调用函数 nand\_send\_cmd(), 发 5 个地址周期;
- 第四步 发读命令 2 NAND\_CMD\_READ\_2st (0xD0);
- 第五步 等待 NAND Flash 就绪;
- 第六步 从页内偏移地址开始读, 读到页结尾即结束, 每次读 1byte;
- 第七步 读状态, 判断是否读成功。

<5> NAND Flash 写函数 copy\_sdram\_to\_nand (), 从 DRAM 写数据到 NAND Flash, 核心代码如下:

```

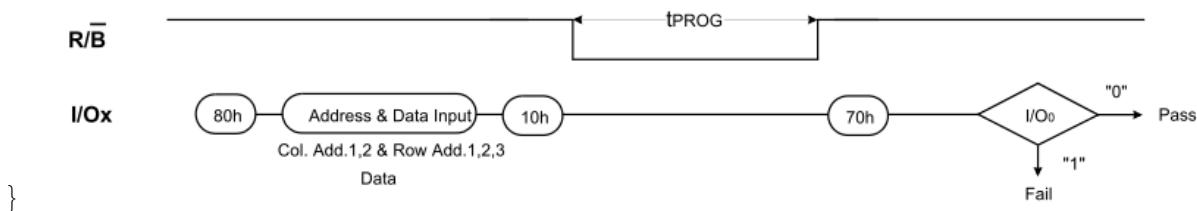
{
    // 1. 发出片选信号
    nand_select_chip();

    // 2. 从 sdram 读数据到 nand, 第一周期发命令 0x80, 第二周期发地址 nand_addr, 第三个
    // 周期写一页(2k)数据, 第四周期发 0x10
    while(length)
    {
        nand_send_cmd(NAND_CMD_WRITE_PAGE_1st);
        nand_send_addr(nand_addr);
        // 列地址, 即页内地址
        unsigned long col = nand_addr % NAND_PAGE_SIZE;
        i = col;
        // 写一页数据, 每次拷 1byte, 共拷 2048 次(2k), 直到长度为 length 的数据拷贝完毕
        for(; i<NAND_PAGE_SIZE && length!=0; i++, length--)
        {
            nand_write(*sdram_addr);
            sdram_addr++;
            nand_addr++;
        }
        NFSTAT = (NFSTAT) | (1<<4);
        nand_send_cmd(NAND_CMD_WRITE_PAGE_2st);
        nand_wait_idle();
    }

    // 3. 读状态
    unsigned char status = read_nand_status();
}

```

**Figure 8. Program & Read Status Operation**



根据上图, NAND Flash 的写操作共 7 个步骤:

**第一步** 发片选;

**第二步** 发写命令 1 NAND\_CMD\_WRITE\_PAGE\_1st (0x80);

**第三步** 发地址地址, 调用函数 nand\_send\_cmd(), 发 5 个地址周期;

**第四步** 发读命令 2 NAND\_CMD\_WRITE\_PAGE\_2st (0x10);

**第五步** 等待 NAND Flash 就绪;

**第六步** 从页内偏移地址开始写, 读到页结尾即结束, 每次写 1byte;



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

第七步 读状态，判断是否读成功。

## 2. main.c

在 main.c 中，首先会调用 nand\_init() 来初始化 NAND Flash，然后打印一个菜单，提供 4 种选择测试 NAND Flash：

读 ID 功能(nand\_read\_id());  
擦除功能(nand\_erase());  
读功能(copy\_nand\_to\_sdram());  
写功能(copy\_sdram\_to\_nand());

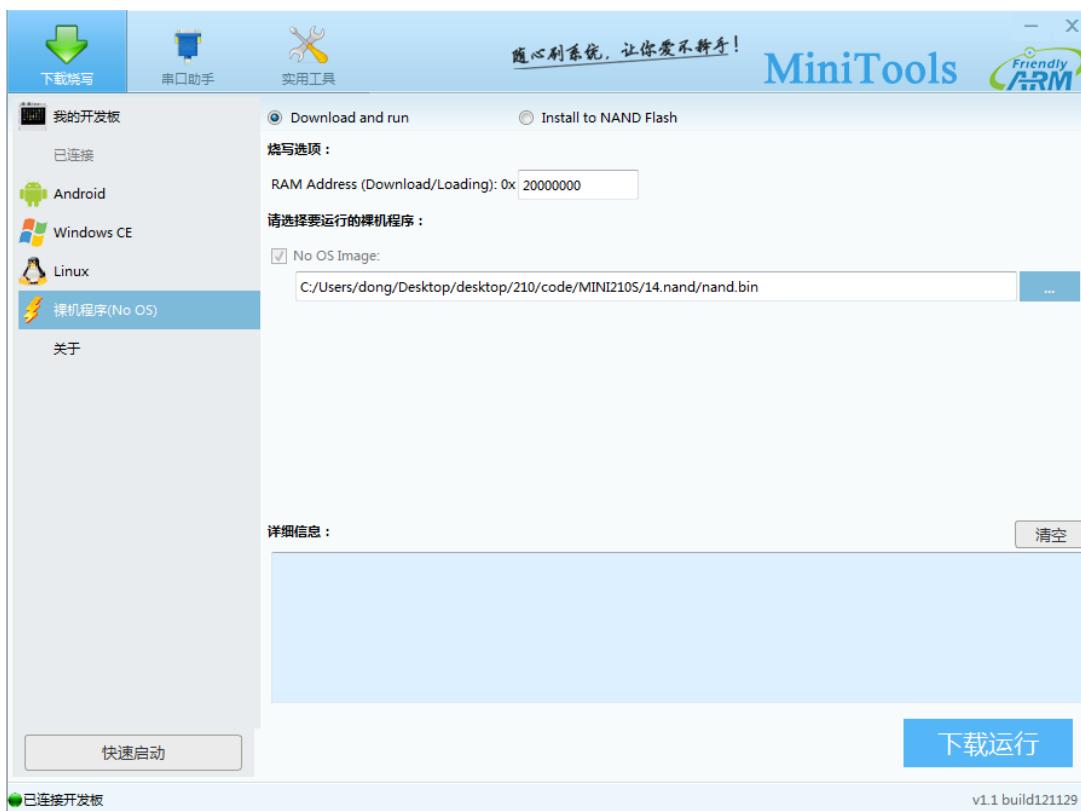
## 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 14.nand  
# make
```

在 14.nand 目录下会生成 nand.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第四节 实验现象

首先会打印 NAND Flash 的操作菜单，菜单的意图很明显，输入 i 可以读 ID，输入 e 可以擦除 NAND Flash 的某一块，输入 w 可以写一个字符串到 NAND Flash。输入 r 可以将字符串从 NAND Flash 中读出来，NAND Flash 的测试效果如下：

```
*****Nandflash R/W Test*****
[i] Read ID
[e] Erase Nandflash
[w] Write Nandflash
[r] Read Nandflash
Enter your choice: i
nandflash: makercode = ec,devicecode = 58

*****Nandflash R/W Test*****
[i] Read ID
[e] Erase Nandflash
[w] Write Nandflash
[r] Read Nandflash
Enter your choice: e
Block # to erase: 0
Block 0 erase ok

*****Nandflash R/W Test*****
[i] Read ID
[e] Erase Nandflash
[w] Write Nandflash
[r] Read Nandflash
Enter your choice: w
Enter a string:
FriendlyARM
Erase block 0 ok
write FriendlyARM ok

*****Nandflash R/W Test*****
[i] Read ID
[e] Erase Nandflash
[w] Write Nandflash
[r] Read Nandflash
Enter your choice: r
Read buffer ok:FriendlyARM

*****Nandflash R/W Test*****
[i] Read ID
[e] Erase Nandflash
[w] Write Nandflash
[r] Read Nandflash
Enter your choice: ■
```

在代码 nand.c 中，还有两个随机读和随机写的函数，可以任意读写 NAND Flash 上的地址，这里没有给出详细解释，请自行阅读。



## 第十六章 S5PV210 中断体系

### 第一节 关于 S5PV210 的中断体系结构

S5PV210 的中断控制器是由 4 个向量中断控制器(VIC)、ARM PrimeCell PL192 和 4 个 TrustZone Interrupt Controller (TZIC)共同组成。

S5PV210 共支持 93 个中断源，待会我们将使能其中的一个外部中断，让大家了解中断处理的完整过程。

### 第二节 程序相关讲解

完整代码见目录 15. int。

#### 1. start.S

共 4 个步骤，其中第 2、4 步和中断相关：

第一步 清 bss；

第二步 开中断，设置 CPSR 寄存器，允许中断发生，代码如下：

```
mov r0, #0x53
```

```
msr CPSR_cxsf, r0
```

第三步 跳转到 main；

第四步 中断处理；程序正常执行时，只会运行到第三步就跳转到 main 而不会执行该部分代码。当有中断发生时，PC 才会跳转到该部分代码，进行中断相关的处理。别急，后面会详细解释该段代码。

#### 2. main.c

共 4 个步骤，其中第 2、3、4 步和中断相关：

第一步 初始化串口；

第二步 中断相关初始化，调用了 system\_initexception()；

代码如下：

```
void system_initexception( void )
{
    // 设置中断向量表
    pExceptionUNDEF      = (unsigned long)exceptionundef;
    pExceptionSWI         = (unsigned long)exceptionswi;
    pExceptionPABORT     = (unsigned long)exceptionpabort;
    pExceptionDABORT     = (unsigned long)exceptiondabort;
    pExceptionIRQ         = (unsigned long)IRQ_handle;
```

```

pExceptionFIQ = (unsigned long) IRQ_handle;

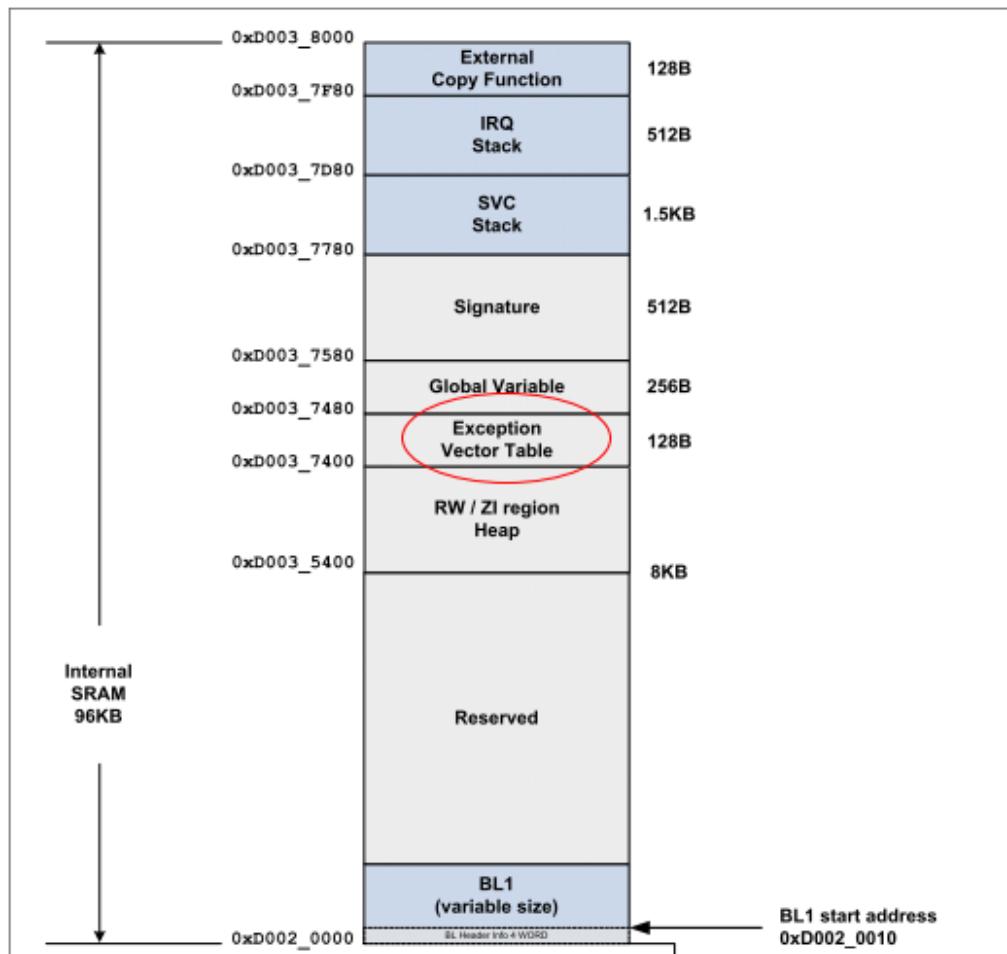
// 初始化中断控制器
intc_init();

}
    
```

这段代码主要做了下面两件事：

1) 设置中断向量表

当发生各种异常时，PC 会自动跳转到相应的异常向量；S5PV210 的异常向量表的起始地址是 0xD0037400，原因见下图：



我们需要特别注意的是，当发生 IRQ 中断异常时，对应的处理函数是 IRQ\_handle()，即我们在 start.S 中第四步所设置的代码，如下：

IRQ\_handle:

```

// 设置中断模式的栈
ldr sp, =0xD0037F80
// 保存现场
sub lr, lr, #4
    
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
stmfd sp!, {r0-r12, lr}
// 跳转到中断处理函数
bl irq_handler
// 恢复现场
ldmfd sp!, {r0-r12, PC} ^
```

注释已经写得很清楚了，当发生 IRQ 中断异常时，会根据中断向量表里的设置，跳转到该部分代码，然后设置中断模式下的栈，保存现场，再调用中断处理函数 `irq_handler()`，处理完中断后再恢复现场。`irq_handler()` 里会怎么做，后面再解释。

## 2) 初始化中断控制器

调用了函数 `intc_init()`，其代码如下：

```
void intc_init(void)
{
    // 禁止所有中断
    VIC0INTENCLEAR = 0xffffffff;
    VIC1INTENCLEAR = 0xffffffff;
    VIC2INTENCLEAR = 0xffffffff;
    VIC3INTENCLEAR = 0xffffffff;

    // 选择中断类型为 IRQ
    VIC0INTSELECT = 0x0;
    VIC1INTSELECT = 0x0;
    VIC2INTSELECT = 0x0;
    VIC3INTSELECT = 0x0;

    // 清 VICxADDR
    intc_clearvectaddr();
}
```

首先先禁止所有中断，然后选择中断类型为 IRQ，最后清寄存器 VICxARRD，VICxADDR 是用来保存当前发生的中断的处理函数的，涉及的寄存器如下：

1.4.1.6 Interrupt Enable Clear  
(VICINTENCLEAR, W, Address=0xF200\_0014, 0xF210\_0014, 0xF220\_0014, 0xF230\_0014)

VICINTENCLEAR	Bit	Description	Initial State
IntEnable Clear	[31:0]	Clears corresponding bits in the VICINTENABLE Register: 0 = No effect 1 = Disables Interrupt in VICINTENABLE Register. There is one bit of the register for each interrupt source.	-



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 1.4.1.4 Interrupt Select Register

(VICINTSELECT, R/W, Address=0xF200\_000C, 0xF210\_000C, 0xF220\_000C, 0xF230\_000C)

VICINTSELECT	Bit	Description	Initial State
IntSelect	[31:0]	Selects interrupt type for interrupt request: 0 = IRQ interrupt 1 = FIQ interrupt  There is one bit of the register for each interrupt source.	0x00000000

## 1.4.1.10 Vector Address Register

(VICADDRESS, R/W, Address=0xF200\_OF00, 0xF210\_OF00, 0xF220\_OF00, 0xF230\_OF00)

VICADDRESS	Bit	Description	Initial State
VectAddr	[31:0]	Contains the address of the currently active ISR, with reset value 0x00000000.  A read of this register returns the address of the ISR and sets the current interrupt as being serviced. A read must be performed while there is an active interrupt.  A write of any value to this register clears the current interrupt. A write must only be performed at the end of an interrupt service routine.	0x00000000

## 第三步 设置外部中断相关寄存器

代码如下：

```
// 1111 = EXT_INT[16]
GPH2CON |= 0xF;
// 010 = Falling edge triggered
EXT_INT_2_CON |= 1<<1;
// unmasked
EXT_INT_2_MASK &= ~(1<<0);
```

首先配置 GPH2\_0 引脚为中断功能；

然后设置外部中断 EINT16\_31 为下降沿触发；

EXT_INT_2_CON[0]	[2:0]	Sets the signaling method of EXT_INT[16] 000 = Low level 001 = High level 010 = Falling edge triggered 011 = Rising edge triggered 100 = Both edge triggered 101 ~ 111 = Reserved	000
------------------	-------	---	-----

最后是不屏蔽该中断；

EXT_INT_2_MASK[0]	[0]	0 = Enables Interrupt 1 = Masked	1
-------------------	-----	-------------------------------------	---

## 第四步 设置 VIC 相关寄存器

代码如下：

```
// 设置中断 EINT16_31 的处理函数
intc_setvectaddr(NUM_EINT16_31, isr_key);
// 使能中断 EINT16_31
intc_enable(NUM_EINT16_31);
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

这里调用了两个函数：

1) intc\_setvectaddr()

它的作用是设置 VICVECTADDR 这一类寄存器，作用是保存各个中断的处理函数，这里我们的外部中断 EINT16\_31 的中断处理函数是 isr\_key()。

1.4.1.12 Vector Address Registers  
(VICVECTADDR[0-31], R/W, Address=0xF200\_0100~017C, 0xF210\_0100~017C, 0xF220\_0100~017C,  
0xF230\_0100~017C)

VICVECTADDR[0-31]	Bit	Description	Initial State
VectorAddr 0-31	[31:0]	Contains ISR vector addresses.	0x00000000

前面提到，在 start.S 的第四步中，会调用函数 irq\_handler()，其代码如下：

```
void irq_handler(void)
{
    unsigned long vicaddr[4] = {VIC0ADDR, VIC1ADDR, VIC2ADDR, VIC3ADDR} ;
    int i=0;
    void (*isr) (void);

    for( ; i<4; i++)
    {
        if(intc_getvicirqstatus(i) != 0)
        {
            isr = vicaddr[i];
            break;
        }
    }
    (*isr)();
}
```

当有中断发生时，硬件上会将当前中断的中断处理函数从寄存器 VICVECTADDR 自动拷贝到寄存器 VICDDR 中，所以我们在 irq\_handler() 函数里会调用保存在寄存器 VICDDR 里的中断处理函数即可。

再来看看外部中断 EINT16\_31 的处理函数 isr\_key()，代码如下：

```
void isr_key(void)
{
    printf("we get company\r\n");
    beep();
    // clear VIC0ADDR
    intc_clearvectaddr();
    // clear pending bit
    EXT_INT_2_PEND |= 1<<0;
}
```



追 求 卓 越 创 造 精 品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

首先打印一句“we get company”，然后蜂鸣器会响一下，最后是清 VIC 相关寄存器 VICOADDR 和外部中断相关寄存器 EXT\_INT\_2\_PEND。

## 2) intc\_enable()

在 VIC 里通过设置寄存器 VICINTENABLE 使能外部中断 EINT16\_31。

### 1.4.1.5 Interrupt Enable Register

(VICINTENABLE, R/W, Address=0xF200\_0010, 0xF210\_0010, 0xF220\_0010, 0xF230\_0010)

VICINTENABLE	Bit	Description	Initial State
IntEnable	[31:0]	<p>Enables the interrupt request lines, which allows the interrupts to reach the processor.</p> <p>Read:</p> <p>0 = Disables Interrupt 1 = Enables Interrupt</p> <p>Use this register to enable interrupt. The VICINTCLEAR Register must be used to disable the interrupt enable.</p> <p>Write:</p> <p>0 = No effect 1 = Enables Interrupt.</p> <p>On reset, all interrupts are disabled.</p> <p>There is one bit of the register for each interrupt source.</p>	0x00000000

## 第五步 死循环

打印数字 1、2、3、4…，等待外部中断 EINT16\_31 的发生。

## 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 15.int  
# make
```

在 15.int 目录下会生成 int.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：

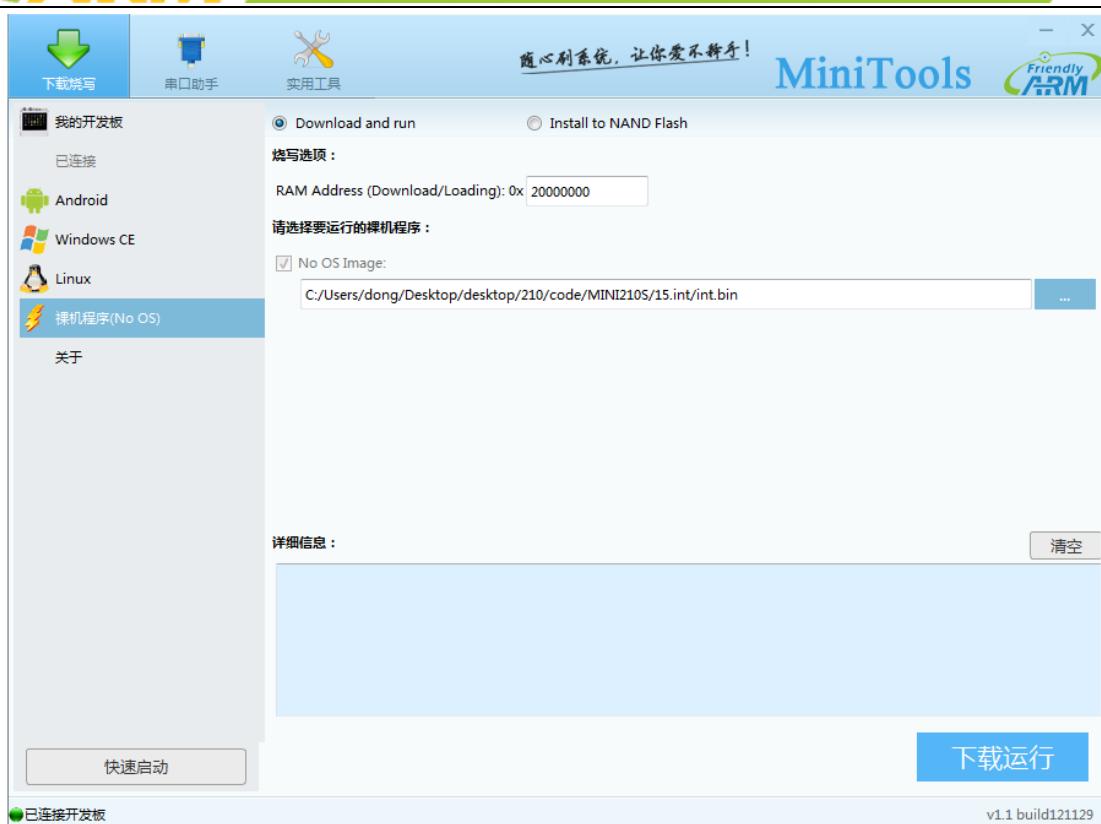


追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



点击“下载运行”，Minitools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

首先会不断的打印数字 1、2、3、4...，当我们按下 KEY1 时会产生外部中断 EINT16\_31 时，会跳转到 IRQ\_handler，然后调用 irq\_handler()，最后调用对应的中断处理函数 isr\_key()。该函数首先打印“we get company:EINT16\_31”，然后清中断。测试效果如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
***** Int test *****  
0  
1  
2  
3  
4  
5  
6  
7  
we get company:EINT16_31  
8  
9  
10  
we get company:EINT16_31  
11  
12  
13
```

# 第十七章 PWM 定时器

## 第一节 S5PV210 的 PWM 定时器

S5PV210 共有 5 个 32bit 的 PWM 定时器，其中定时器 0、1、2、3 有 PWM 功能，定时器 4 没有输出引脚。PWM 定时器使用 PCLK\_PSYS 作为时钟源，相关知识可以查阅第七章-初始化时钟，相关的结构图如下：

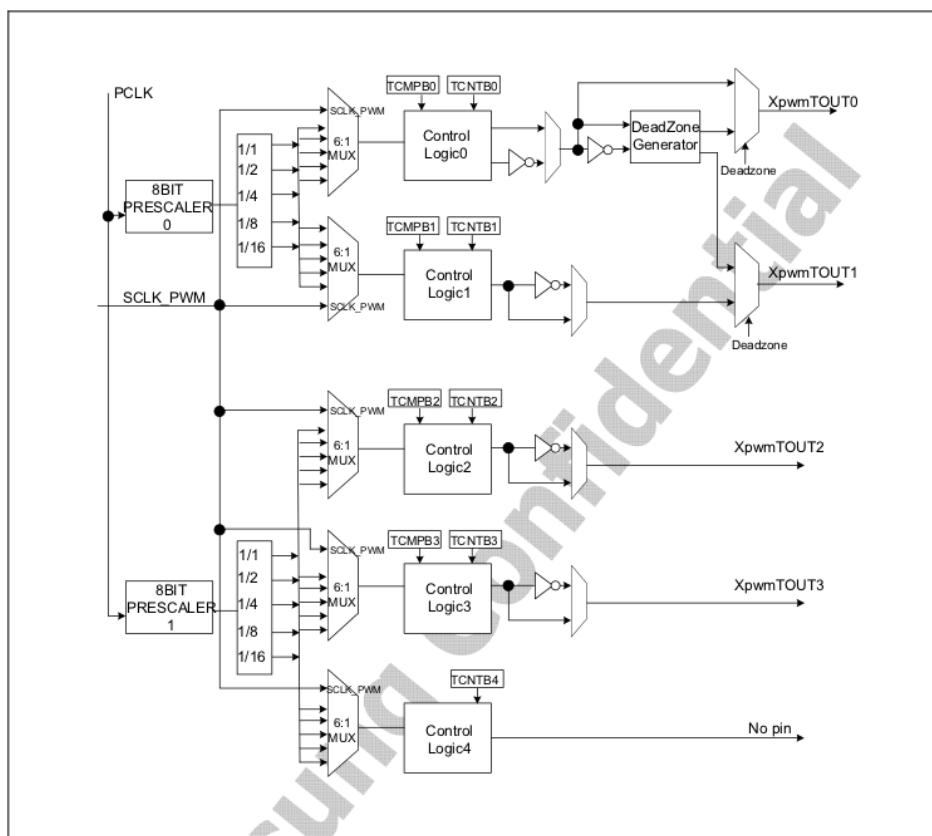


Figure 1-2 PWM TIMER Clock Tree Diagram

## 第二节 程序相关讲解

完整代码见目录 16. timer。

### 1. main.c

核心代码如下：

```
int main(void)
{
    // 初始化串口
```



```
uart_init();

// 中断相关初始化
system_initexception();

// 设置 timer
timer_request();

while(1);
}
```

共 4 个步骤，其中第 3 步与定时器相关：

第一步 初始化串口；

第二步 中断相关初始化；

第三步 设置 timer，函数 timer\_request() 的定义位于 timer.c 中；

第四步 死循环，等待 timer 中断的发生；

## 2. timer.c

```
void timer_request(void)
{
    printf("\r\n#####Timer test#####\r\n");
    // 禁止所有 timer
    pwm_stopall();
    // 设置 timer0 中断的中断处理函数
    intc_setvectaddr(NUM_TIMER0, irs_timer);
    // 使能 timer0 中断
    intc_enable(NUM_TIMER0);
    // 设置 timer0
    timer_init(0, 65, 4, 62500, 0);
}
```

共 3 个步骤：

第一步 禁止所有 timer，往寄存器 TCON 写 0 即可；

第二步 设置 VIC，先设置 timer0 中断的中断处理函数为 irs\_timer()，然后使能 timer0 中断；

第三步 设置 timer0，调用了函数 timer\_init()，其核心工作是：

### 1) 设置分频

首先设置分频系数，相关寄存器是 TCFG0，如下：

TCFG0	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Dead zone length	[23:16]	Dead zone length	0x00
Prescaler 1	[15:8]	Prescaler 1 value for Timer 2, 3 and 4	0x01
Prescaler 0	[7:0]	Prescaler 0 value for timer 0 and 1	0x01



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

然后设置为 16 分频，相关寄存器是 TCFG1，如下：

#### 1.5.1.2 Timer Configuration Register (TCFG1, R/W, Address = 0xE250\_0004)

TCFG1	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Divider MUX4	[19:16]	Selects Mux input for PWM Timer 4 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX3	[15:12]	Selects Mux input for PWM Timer 3 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX2	[11:8]	Selects Mux input for PWM Timer 2 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX1	[7:4]	Selects Mux input for PWM Timer 1 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX0	[3:0]	Selects Mux input for PWM Timer 0 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00

经过上面的设置之后，就能确定 timer 的输入时钟了，计算方式如下：

Timer Input Clock Frequency = PCLK / ( {prescaler value + 1} ) / {divider value} = 66MHz/(65+1)/16=62500hz

## 2) 设置计数

设置寄存器 TCNTB0=62500 和 TCMPB0=0，启动 timer0 后，TCNTB0 会逐渐-1，直到等于 TCMPB0 时就产生一次中断，即 1 秒产生一次 timer0 中断。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

#### 1.5.1.4 Timer0 Counter Register (TCNTB0, R/W, Address = 0xE250\_000C)

TCNTB0	Bit	Description	Initial State
Timer 0 Count Buffer	[31:0]	Timer 0 Count Buffer Register	0x0000_0000

#### 1.5.1.5 Timer0 Compare Register (TCMPB0, R/W, Address = 0xE250\_0010)

TCMPB0	Bit	Description	Initial State
Timer 0 Compare Buffer	[31:0]	Timer 0 Compare Buffer Register	0x0000_0000

### 3) 启动 timer0

设置寄存器 TCON，先设置手动更新位，然后清除手动更新位，使用自动装载，最后启动 timer0。

### 4) 使能 timer0 中断

设置寄存器 TINT\_CSTAT，使能 timer0 中断。

#### 1.5.1.18 Interrupt Control and Status Register (TINT\_CSTAT, R/W, Address = 0xE250\_0044)

TINT_CSTAT	Bit	Description	Initial State
Reserved	[31:10]	Reserved Bits	0x00000
Timer 4 Interrupt Status	[9]	Timer 4 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 3 Interrupt Status	[8]	Timer 3 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 2 Interrupt Status	[7]	Timer 2 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 1 Interrupt Status	[6]	Timer 1 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 0 Interrupt Status	[5]	Timer 0 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 4 interrupt Enable	[4]	Enables Timer 4 Interrupt. 1 = Enabled 0 = Disabled	0x0
Timer 3 interrupt Enable	[3]	Enables Timer 3 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 2 interrupt Enable	[2]	Enables Timer 2 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 1 interrupt Enable	[1]	Enables Timer 1 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 0 interrupt Enable	[0]	Enables Timer 0 Interrupt. 1 = Enables 0 = Disabled	0x0

最后再来分析 timer0 中断处理函数 irs\_timer()，共 3 个步骤：

第一步 清 timer0 的中断状态寄存器 TINT\_CSTAT；

第二步 打印 timer0 中断发生的次数，每产生一次 timer0 中断就打印一次。

第三步 VIC 相关的中断清除，调用函数 intc\_clearvectaddr()；

### 第三节 编译代码和烧写运行

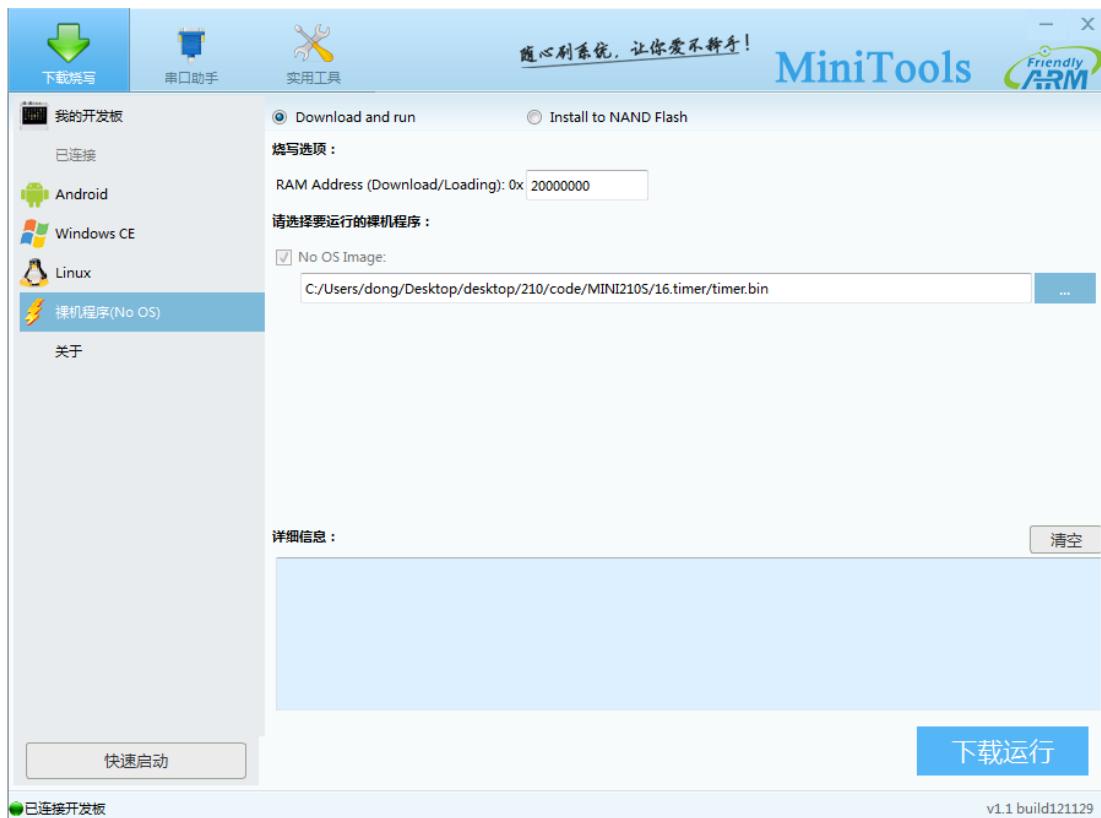
编译代码，在 Fedora 终端执行如下命令：

```
# cd 16.timer
```

```
# make
```

在 16.timer 目录下会生成 timer.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

### 第四节 实验现象

终端会不断的打印数字 1、2、3、4...，频率为每秒打印 1 次，效果如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
#####Timer test#####
Timer0IntCounter = 0
Timer0IntCounter = 1
Timer0IntCounter = 2
Timer0IntCounter = 3
Timer0IntCounter = 4
Timer0IntCounter = 5
Timer0IntCounter = 6
Timer0IntCounter = 7
Timer0IntCounter = 8
Timer0IntCounter = 9
Timer0IntCounter = 10
```

## 第十八章 看门狗定时和复位

### 第一节 S5PV210 的看门狗定时器

S5PV210 上的看门狗定时器相当于一个普通的 16bit 的定时器，它与 PWM 定时器的区别是看门狗定时器可以产生 reset 信号而 PWM 定时器不能，S5PV210 看门狗定时器的结构图如下：

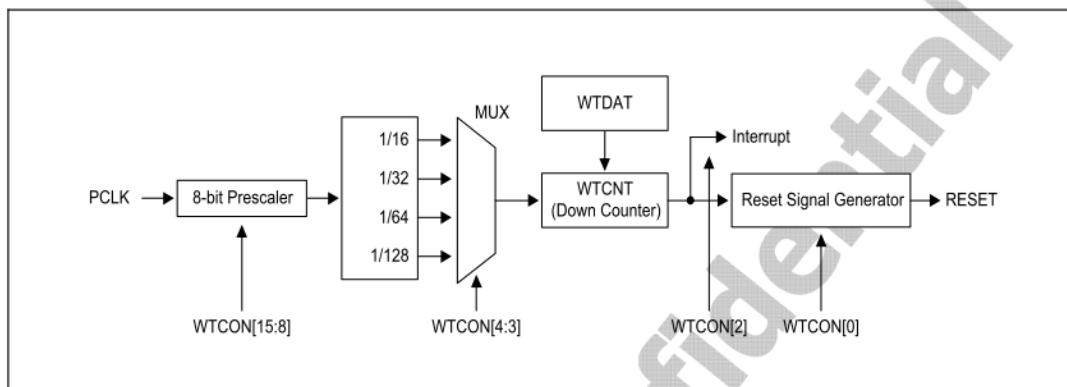


Figure 3-1 Watchdog Timer Block Diagram

### 第二节 程序相关讲解

完整代码见目录 17. watchdog。

#### 1. main.c

共 4 个步骤：

第一步 初始化串口；

第二步 中断相关初始化；

第三步 测试看门狗，调用函数 wtd\_test()，其定义位于 wtd.c 中；

第四步 死循环，等待看门狗中断的发生；

#### 2. wtd.c

wtd\_test() 的内容包括 4 个步骤：

第一步 VIC 相关的中断设置，包括设置 WTD 中断的处理函数为 isr\_wtd() 和使能中断；

第二步 测试看门狗定时的功能，调用了函数 wtd\_operate()，这里我们只是使能了看门狗定时器的定时功能，而并没有使能 reset 功能；

wtd\_operate() 的完整代码如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
void wtd_operate(unsigned long uenreset, unsigned long uenint, unsigned long uselectclk,
unsigned long uenwtd, unsigned long uprescaler, unsigned long uwtdat, unsigned long
uwtcnt)
{
    WTDAT = uwtdat;
    WTCNT = uwtcnt;

    /*uenreset: 是否使能 reset
     *uenint: 是否使能中断
     *uselectclk: 分频系数
     *uenwtd: 是否启动定时器
     *bit[8:15]: 预分频系数
     */
    WTCON= (uenreset<<0) | (uenint<<2) | (uselectclk<<3) | (uenwtd<<5) | ((uprescaler)<<8);
}
```

首先设置计数相关的寄存器 WTDAT 和 WTCNT，寄存器 WTDAT 用来决定看门狗定时器的超时周期，在看门狗定时器启动后，寄存器 WTDAT 的值会自动传入寄存器 WTCNT，当 WTCNT 计数达到 0 时：如果中断被使能的话会发出中断，如果 reset 功能被使用的话会发出复位信号，然后装载 WTDAT 的值并重新计数。

#### 3.4.1.2 Watchdog Timer Data Register (WTDAT, R/W, Address = 0xE270\_0004)

The WTDAT register specifies the time-out duration. The content of WTDAT cannot be automatically loaded into the timer counter at initial watchdog timer operation. However, using 0x8000 (initial value) drives the first time-out. In this case, the value of WTDAT is automatically reloaded into WTCNT.

WTDAT	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
Count reload value	[15:0]	Watchdog timer count value for reload.	0x8000

#### 3.4.1.3 Watchdog Timer Count Register (WTCNT, R/W, Address = 0xE270\_0008)

The WTCNT register contains the current count values for the watchdog timer during normal operation. Note that the content of the WTDAT register cannot be automatically loaded into the timer count register if the watchdog timer is enabled initially, therefore the WTCNT register must be set to an initial value before enabling it.

WTCNT	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
Count value	[15:0]	The current count value of the watchdog timer	0x8000

寄存器 WTCN 进行相关配置，用来决定是否使能 reset、是否使能中断、分频、是否启动定时器等，具体见下图：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

WTCON	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
Prescaler value	[15:8]	Prescaler value. The valid range is from 0 to (2 <sup>8</sup> -1).	0x80
Reserved	[7:6]	Reserved. These two bits must be 00 in normal operation.	00
Watchdog timer	[5]	Enables or disables Watchdog timer bit. 0 = Disables 1 = Enables	1
Clock select	[4:3]	Determines the clock division factor. 00 = 16 01 = 32 10 = 64 11 = 128	00
Interrupt generation	[2]	Enables or disables interrupt bit. 0 = Disables 1 = Enables	0
Reserved	[1]	Reserved. This bit must be 0 in normal operation.	0
Reset enable/disable	[0]	Enables or disables Watchdog timer output bit for reset signal. 1 = Asserts reset signal of the S5PV210 at watchdog time-out 0 = Disables the reset function of the watchdog timer.	1

最后再来看看门狗中断的处理函数，代码如下：

```
void isr_wtd()
{
    //记录中断发生次数
    static int wtdcounter=0;
    printf("%d\r\n", ++wtdcounter);

    // 看门狗相关中断清除
    WTCLRINT = 1;
    // VIC 相关中断清除
    intc_clearvectaddr();
    if(wtdcounter==5)
    {
        // 看门狗 reset
        printf("waiting system reset\r\n");
        wtd_operate(1, 1, 0, 1, 100, 100000000, 100000000);
    }
}
```

共 3 个步骤：

第一步 打印中断发生的次数；

第二步 中断清除；

第三步 当发生了 5 次定时中断后，使能看门狗的 reset 功能，此时系统会重启；

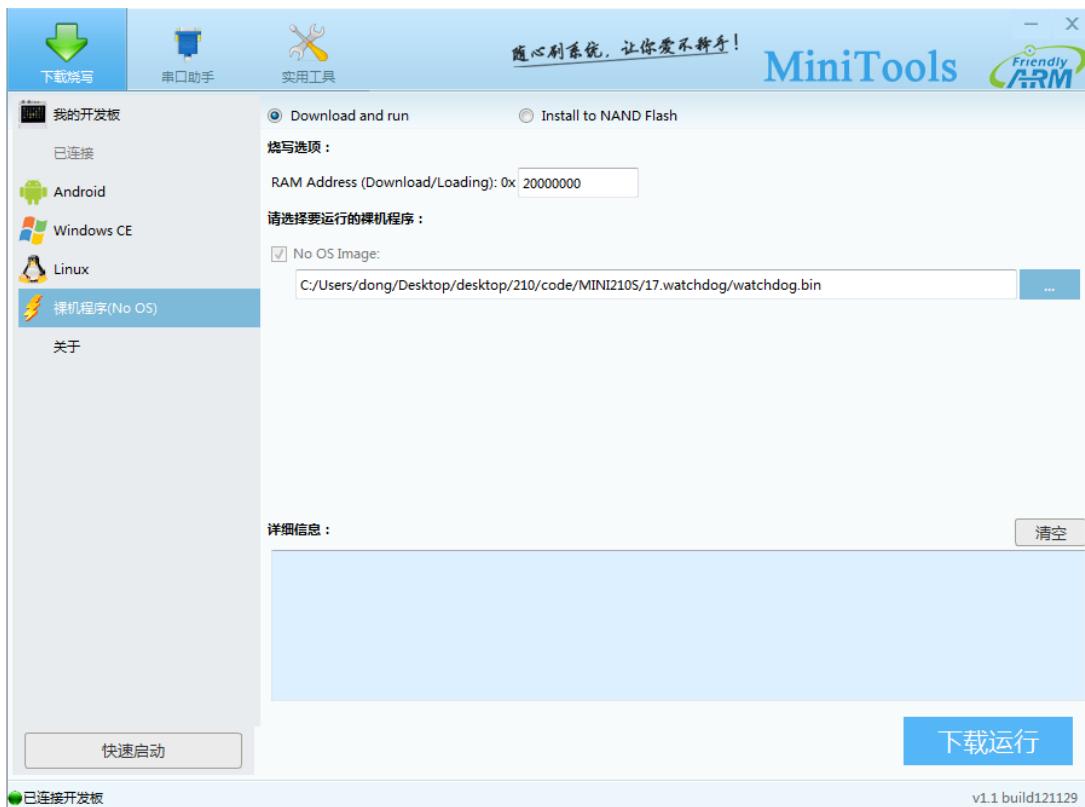
### 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 17.watchdog  
# make
```

在 17.watchdog 目录下会生成 watchdog.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

### 第四节 实验现象

首先会打印 1、2、3、4...，当打印 5 时，watchdog 的 reset 功能被使能，系统会重启，说明看门狗的复位作用生效了，效果如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
##### watchdog test#####
1
2
3
4
5
waiting system reset
```

# 第十九章 RTC 读写时间

## 第一节 S5PV210 的 RTC

RTC 就是实时时钟芯片，用来在系统断电时，利用备用的锂电池继续记录时间。S5PV210 的 RTC 主要有如下特点：

- 1) 支持 BCD 数字；
- 2) 支持 alarm 功能；
- 3) 支持 tick 功能；
- 4) 支持毫秒级的 tick time；

下面是 S5PV210 的 RTC 相关结构图：

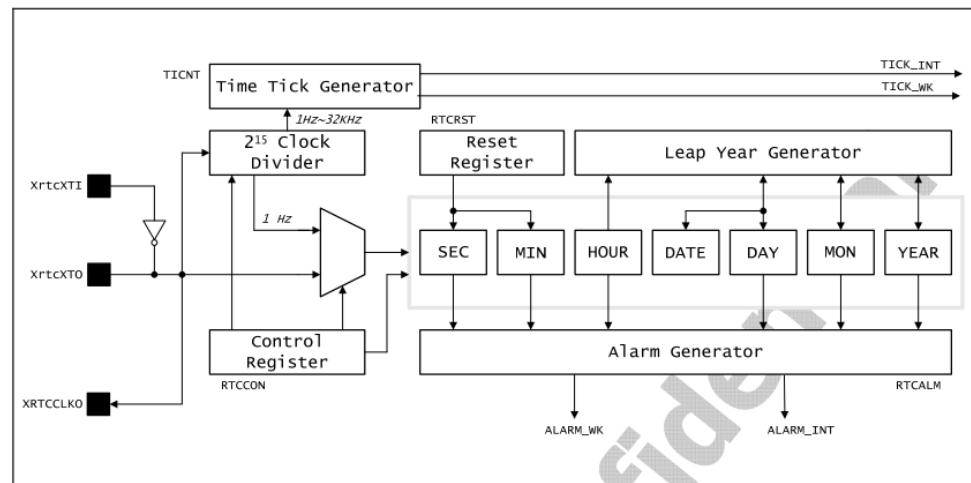


Figure 4-1 Real Time Clock Block Diagram

## 第二节 程序相关讲解

完整代码见目录 18. rtc。

### 1. main.c

共 3 个步骤，其中与 RTC 相关的是第 3 步：

第一步 初始化串口；

第二步 中断相关初始化；

第三步 打印菜单，共两个选择，输入 d 会调用函数 `rtc_realtime_display()` 显示当前的 RTC 时间，输入 s 会调用函数 `rtc_settime()` 将 RTC 时间复位，这两个函数的定义均位于 `rtc.c`；

### 2. rtc.c

<1> `rtc_realtime_display()` 的完整定义如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
void rtc_realtime_display(void)
{
    int counter = 0;

    // 使能 rtc 控制器
    rtc_enable(true);
    // 使能 rtc tick timer
    rtc_ticktime_enable(true);

    // 打印 5 次时间
    while( (counter++) < 5)
    {
        rtc_print();
        delay(0x1000000/5);
    }

    // 关闭 rtc 控制器
    rtc_ticktime_enable(false);
    // 关闭 rtc tick timer
    rtc_enable(false);
}
```

共 3 个步骤：

第一步 `rtc_enable(true)` 使能 rtc 控制器，`rtc_ticktime_enable(true)` 使能 rtc tick timer，。两个函数的实质都是设置寄存器 RTCCON，具体见下图：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

RTCCON	Bit	Description	Initial State
Reserved	[31:10]	Reserved	0
CLKOUTEN	[9]	Enables RTC clock output on XRTCCLKO pad. 0 = Disables 1 = Enables	0
TICEN	[8]	Enables Tick timer 0 = Disables 1 = Enables	0
TICKSEL	[7:4]	Tick timer sub clock selection. 4'b0000 = 32768 Hz 4'b0010 = 8192 Hz 4'b0100 = 2048 Hz 4'b0110 = 512 Hz 4'b1000 = 128 Hz 4'b1010 = 32 Hz 4'b1100 = 8 Hz 4'b1110 = 2 Hz 4'b0001 = 16384 Hz 4'b0011 = 4096 Hz 4'b0101 = 1024 Hz 4'b0111 = 256 Hz 4'b1001 = 64 Hz 4'b1011 = 16 Hz 4'b1101 = 4 Hz 4'b1111 = 1 Hz	4'b0000
CLKRST	[3]	RTC clock count reset. 0 = RTC counter ( $2^{15}$ clock divider) enable 1 = RTC counter reset and disable Note: When RTCEN is enabled, CLKRST affects RTC.	0
CNTSEL	[2]	BCD count select. 0 = Merge BCD counters 1 = Reserved (Separate BCD counters) Note: When RTCEN is enabled, CNTSEL affects RTC.	0
CLKSEL	[1]	BCD clock select. 0 = XTAL 1/ $2^{15}$ divided clock 1 = Reserved (XTAL clock only for test) Note: When RTCEN is enabled, CLKSEL affects RTC.	0
RTCEN	[0]	Enables RTC control. 0 = Disables 1 = Enables Note: When RTCEN is enabled, you can change the BCD time count setting, $2^{15}$ clock divider reset, BCD counter select, and BCD clock select can be performed.	0

**第二步** rtc\_print()用于打印时间，其作用就是读寄存器 BCODYEAR、BCDMON、BCDDATE、BCDHOUR、BCDMIN、BCDSEC、BCDDAY，分别获取年、月、日、时、分、秒、星期几，并将其打印出来；

**第三步** 关闭 rtc 控制器和 rtc tick timer，同样都是调用了函数 rtc\_enable() 和 rtc\_ticktime\_enable()，不过这次是关闭功能。

<2> rtc\_settime()的完整定义如下：

```
void rtc_settime(void)
{
    // 初始值为重置值

    unsigned long year = 12;
    unsigned long month = 5;
    unsigned long date = 1;
    unsigned long hour = 12;
    unsigned long min = 0;
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
unsigned long sec = 0;
unsigned long weekday= 3;

//将时间转化为BCD码
year = ( ((year/100)<<8) +(((year/10)%10)<<4) + (year%10) );
month = ( ((month/10)<<4)+ (month%10) );
date = ( ((date/10)<<4) + (date%10) );
weekday = (weekday%10);
hour =( ((hour/10)<<4) + (hour%10) );
min =(( (min/10)<<4) + (min%10) );
sec =( ((sec/10)<<4) + (sec%10) );

rtc_enable(true);
// 保存BCD码
BCDSEC = sec;
BCDMIN = min;
BCDHOUR = hour;
BCDDATE = date;
BCDDAY = weekday;
BCDMON = month;
BCDYEAR = year;
rtc_enable(false);

printf("reset success\r\n");
}
```

共3个步骤：

第一步 给 year、month、date 等时间变量赋初始值；

第二步 将时间转化为BCD码；

第三步 保存BCD码到寄存器BCDYEAR、BCDMON、BCDDATE等；

### 第三节 编译代码和烧写运行

编译代码，在Fedora终端执行如下命令：

```
# cd 18 rtc
```

```
# make
```

在18 rtc目录下会生成 rtc.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到DRAM，具体如下：

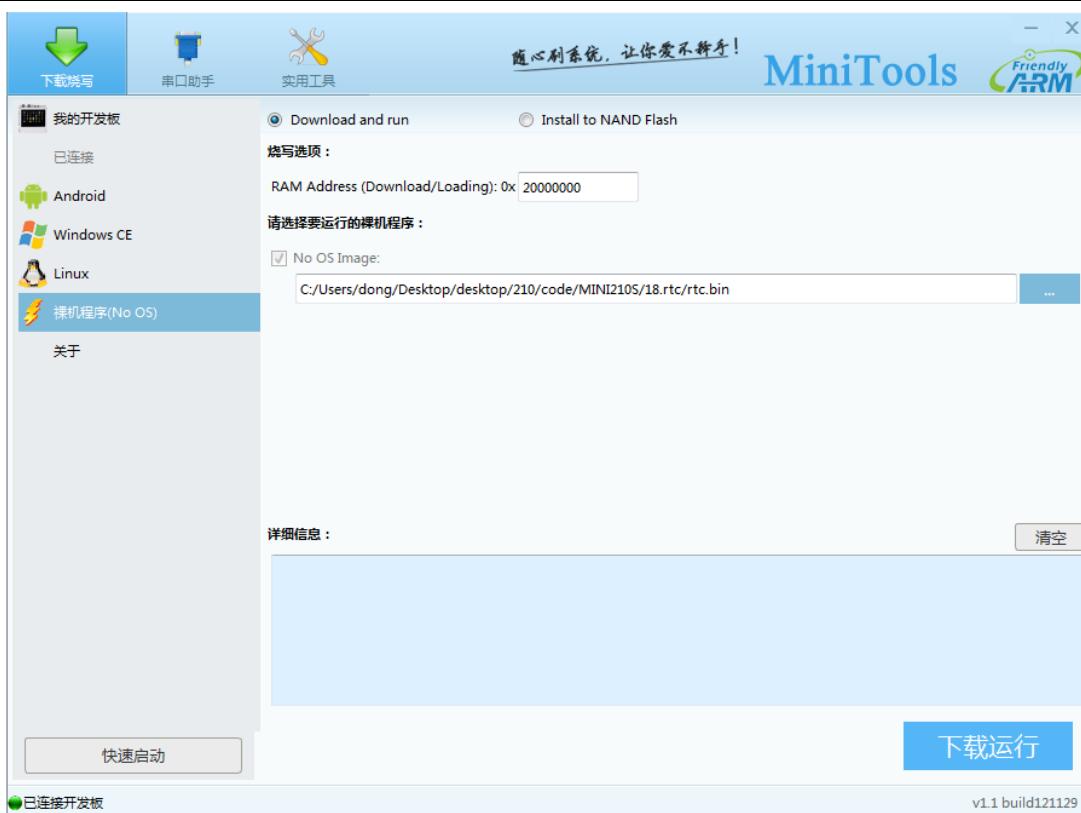


追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司



点击“下载运行”，Minitools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

首先打印功能菜单，输入 s 复位时间，输入 d 显示当前时间，效果如下：

```
#####rtc test#####
[d] Display rtc realtime(hour:min:sec:weekday date/month/year)
[s] Reset rtc realtime(12:0:0:Tuesday 1/1/2012)
Enter your choice:s
reset success

#####rtc test#####
[d] Display rtc realtime(hour:min:sec:weekday date/month/year)
[s] Reset rtc realtime(12:0:0:Tuesday 1/1/2012)
Enter your choice:d
12 : 0 : 1      Tuesday,    5/ 1/2012
12 : 0 : 2      Tuesday,    5/ 1/2012
12 : 0 : 3      Tuesday,    5/ 1/2012
12 : 0 : 4      Tuesday,    5/ 1/2012
12 : 0 : 5      Tuesday,    5/ 1/2012
```

## 第二十章 LCD 描点画线

### 第一节 S5PV210 LCD 控制器

要使一块 LCD 正常显示文字或图像，不仅需要 LCD 驱动器，还需要相应的 LCD 控制器。LCD 控制器的主要作用是将在系统存储器中的显示缓冲区中的 LCD 图像数据传送到外部 LCD 驱动器，并产生必要的控制信号，例如 VSYNC、HSYNC、VCLK。S5PV210 内部集成了 LCD 控制器，它结构图如下：

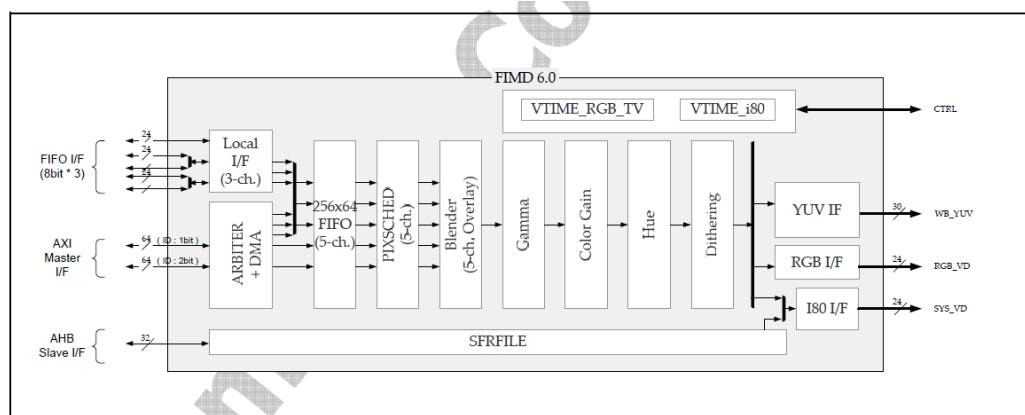


Figure 1-1 Block Diagram of Display Controller

主要有 VSFR, VDMA, VPRCS, VTIME 和视频时钟产生器几个模块组成：

- 1) VSFR 由 121 个可编程寄存器组，一套 gamma LUT 寄存器组(包括 64 个寄存器)，一套 i80 命令寄存器组(包括 12 个寄存器)和 5 块 256\*32 的调色板存储器组成，主要用于 lcd 控制器的配置；
- 2) VDMA 是 LCD 专用的 DMA 传输通道，可以自动从系统总线上获取视频数据传输到 VPRCS，无需 CPU 干涉；
- 3) VPRCS 收到数据后组成特定的格式(如 16bpp 或 24bpp)，然后通过数据接口传送到外部 LCD；
- 4) VTIME 模块又可编程逻辑组成，负责不同的 LCD 驱动器的接口时序控制需求，VTIME 模块产生 VSYNC、HSYNC、VCLK 等信号；

S5PV210 的 LCD 控制器的主要特性如下：

- 1) 支持 3 种接口:RGB/i80/YUV；
- 2) 支持可编程的 DMA；
- 3) 5 个 256\*32 bit 调色板
- 4) 虚拟屏最大可达 16MB
- 5) 支持透明叠加(overlay)
- 6) 支持多种规格和分辨率的 LCD

### 第二节 程序相关讲解

完整代码见目录 19. lcd，需要注意的是，Mini210S 标配 4.3 寸 LCD 屏，型号是 H43-Hsd043I9W1。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 1. main.c

共 3 个步骤，其中与 LCD 相关的是第 2、3 步：

第一步 初始化串口；

第二步 初始化 LCD；

第三步 打印菜单，共 5 个选择，分别是清屏、画十字、画横线、画竖线，它们的原理都是一样的，后面会讲解。

## 2. lcd.c

### (一) 函数 lcd\_init():

在 S5PV210 中，给出了 lcd 控制器的配置步骤，见下图：

#### 1.4.1 OVERVIEW OF PROGRAMMER'S MODEL

Use the following registers to configure display controller:

1. VIDCON0: Configures video output format and displays enable/disable.
2. VIDCON1: Specifies RGB I/F control signal.
3. VIDCON2: Specifies output data format control.
4. VIDCON3: Specifies image enhancement control.
5. I80IFCONx: Specifies CPU interface control signal.
6. VIDTCONx: Configures video output timing and determines the size of display.
7. WINCONx: Specifies each window feature setting.
8. VIDOSDxA, VIDOSDxB: Specifies window position setting.
9. VIDOSDxC,D: Specifies OSD size setting.
10. VIDWxAALPHA0/1: Specifies alpha value setting.
11. BLENDEQx: Specifies blending equation setting.
12. VIDWxxADDx: Specifies source image address setting.
13. WxKEYCONx: Specifies color key setting register.

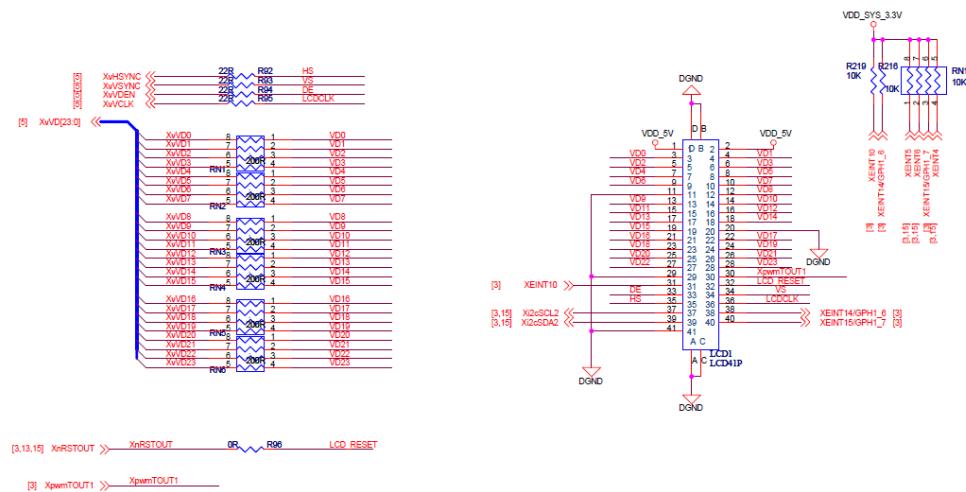
14. WxKEYALPHA: Specifies color key alpha value setting.
15. WINxMAP: Specifies window color control.
16. GAMMALUT\_xx: Specifies gamma value setting.
17. COLORGAINCON: Specifies color gain value setting.
18. HUExxx: Specifies Hue coefficient and offset value setting.
19. WPALCON: Specifies palette control register.
20. WxRTQOSCON: Specifies RTQoS control register.
21. WxPDATAXx: Specifies Window Palette Data of each Index.
22. SHDOWCON: Specifies Shadow control register.
23. WxRTQOSCON: Specifies QoS control register.

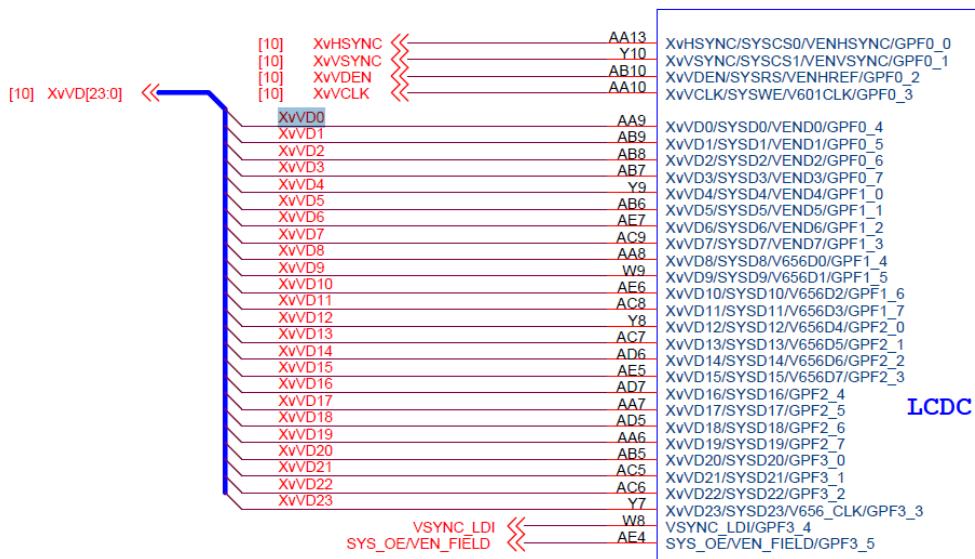
参考上述序列，在 lcd.c 里函数 lcd\_init() 对 lcd 控制器进行了初始化，共 9 个步骤：

第一步 配置相关引脚用于 LCD 功能，相关代码如下：

```
GPF0CON = 0x22222222;
GPF1CON = 0x22222222;
GPF2CON = 0x22222222;
GPF3CON = 0x22222222;
```

查看 Mini210S 的原理图可知，涉及到的 GPIO 引脚为 GPF0/1/2/3：





## 第二步 打开背光，相关代码如下：

```
// 打开背光
GPD0CON |= (1<<4);
GPD0DAT |= (1<<1);
```

相关原理图如下：



## 第三步 配置 DISPLAY\_CONTROL，设置数据输出路径；芯片手册上要求必须设置；

Using the display controller data, you can select one of the above data paths by setting DISPLAY\_PATH\_SEL[1:0] (0xE010\_7008). For more information, refer to Chapter, "Section 02.03. Clock controller".

## 第四步 配置 VIDCONx，设置接口类型、时钟、极性和使能 LCD 控制器等；

### 1) VIDCON0:



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 1.5.2.1 Video Main Control 0 Register (VIDCON0, R/W, Address = 0xF800\_0000)

VIDCON0	Bit	Description	Initial State
Reserved	[31]	Reserved (should be 0).	0
DSI_EN	[30]	Enables MIPI DSI. 0 = Disables 1 = Enables (i80 24bit data interface, SYS_ADD[1])	0
Reserved	[29]	Reserved (should be 0)	0
VIDOUT	[28:26]	Determines the output format of Video Controller. 000 = RGB interface 001 = Reserved 010 = Indirect I80 interface for LD10 011 = Indirect I80 interface for LD11 100 = WB interface and RGB interface 101 = Reserved 110 = WB Interface and i80 interface for LD10 111 = WB Interface and i80 interface for LD11	000
L1_DATA16	[25:23]	Selects output data format mode of indirect i80 interface (LD11). (VIDOUT[1:0] == 2'b11) 000 = 16-bit mode (16 bpp) 001 = 16 + 2-bit mode (18 bpp) 010 = 9 + 9-bit mode (18 bpp) 011 = 16 + 8-bit mode (24 bpp) 100 = 18-bit mode (18bpp) 101 = 8 + 8-bit mode (16bpp)	000
L0_DATA16	[22:20]	Selects output data format mode of indirect i80 interface (LD10). (VIDOUT[1:0] == 2'b10) 000 = 16-bit mode (16 bpp) 001 = 16 + 2-bit mode (18 bpp) 010 = 9 + 9-bit mode (18 bpp) 011 = 16 + 8-bit mode (24 bpp) 100 = 18-bit mode (18bpp) 101 = 8 + 8-bit mode (16bpp)	000
Reserved	[19]	Reserved (should be 0).	0
RGSPSEL	[18]	Selects display mode (VIDOUT[1:0] == 2'b00). 0 = RGB parallel format 1 = RGB serial format  Selects the display mode (VIDOUT[1:0] != 2'b00). 0 = RGB parallel format	0
PNRMODE	[17]	Controls inverting RGB_ORDER (@VIDCON3). 0 = Normal: RGBORDER[2] @VIDCON3 1 = Invert: ~RGBORDER[2] @VIDCON3  Note: This bit is used for the previous version of FIMD. You do not have to use this bit if you use RGB_ORDER@VIDCON3 register.	00
CLKVALUP	[16]	Selects CLKVAL_F update timing control. 0 = Always 1 = Start of a frame (only once per frame)	0



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

VIDCON0	Bit	Description	Initial State
Reserved	[15:14]	Reserved.	0
CLKVAL_F	[13:6]	Determines the rates of VCLK and CLKVAL[7:0]. VCLK = HCLK / (CLKVAL+1), where CLKVAL >= 1 Notes. 1. The maximum frequency of VCLK is 100Mhz(pad:50pf). 2. CLKSEL_F register selects Video Clock Source.	0
VCLKFREE	[5]	Controls VCLK Free Run (Only valid at RGB IF mode). 0 = Normal mode (controls using ENVID) 1 = Free-run mode	0
CLKDIR	[4]	Selects the clock source as direct or divide using CLKVAL_F register. 0 = Direct clock (frequency of VCLK = frequency of Clock source) 1 = Divided by CLKVAL_F	0x00
Reserved	[3]	Should be 0.	0x0
CLKSEL_F	[2]	Selects the video clock source. 0 = HCLK 1 = SCLK_FIMD HCLK is the bus clock, whereas SCLK_FIMD is the special clock for display controller. For more information, refer to Chapter, "02.03 CLOCK CONTROLLER".	0
ENVID	[1]	Enables/ disables video output and logic immediately. 0 = Disables the video output and display control signal. 1 = Enables the video output and display control signal.	0
ENVID_F	[0]	Enables/ disables video output and logic at current frame end. 0 = Disables the video output and display control signal. 1 = Enables the video output and display control signal. * If this bit is set to "on" and "off", then "H" is read and video controller is enabled until the end of current frame.	0

- ENVID\_F=1, 当前帧结束后使能 lcd 控制器;
  - ENVID=1, 使能 lcd 控制器;
  - CLKSEL\_F=1, 选择时钟源为 HCLK\_DSYS=166MHz;
  - CLKDIR=1, 选择需要分频;
  - CLKVAL\_F=14, 分频系数为 15, 即  $VCLK = 166M/(14+1) = 11M$ ;
  - RGSPSEL=0, RGB 并行;
  - VIDOUT=0, 使用 RGB 接口;
- 未设置的 bit 均使用默认值。

## 2) VIDCON1:



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 1.5.2.2 Video Main Control 1 Register (VIDCON1, R/W, Address = 0xF800\_0004)

VIDCON1	Bit	Description	Initial State
LINECNT (read only)	[26:16]	Provides the status of the line counter (read only). Up count from 0 to LINEVAL.	0
FSTATUS	[15]	Specifies the Field Status (read only). 0 = ODD Field 1 = EVEN Field	0
VSTATUS	[14:13]	Specifies the Vertical Status (read only). 00 = VSYNC 01 = BACK Porch 10 = ACTIVE 11 = FRONT Porch	0
Reserved	[12:11]	Reserved	0
FIXVCLK	[10:9]	Specifies the VCLK hold scheme at data under-flow. 00 = VCLK hold 01 = VCLK running 11 = VCLK running and VDEN disable	0
Reserved	[8]	Reserved	0
IVCLK	[7]	Controls the polarity of the VCLK active edge. 0 = Video data is fetched at VCLK falling edge 1 = Video data is fetched at VCLK rising edge	0
IHSYNC	[6]	Specifies the HSYNC pulse polarity. 0 = Normal 1 = Inverted	0
IVSYNC	[5]	Specifies the VSYNC pulse polarity. 0 = Normal 1 = Inverted	0
IVDEN	[4]	Specifies the VDEN signal polarity. 0 = Normal 1 = Inverted	0
Reserved	[3:0]	Reserved	0x0

VIDCON1 只需要设置下面两个 bit:

- ✚ IHSYNC=1, 极性反转
- ✚ IVSYNC=1, 极性反转

为什么需要反转, 原因如下:

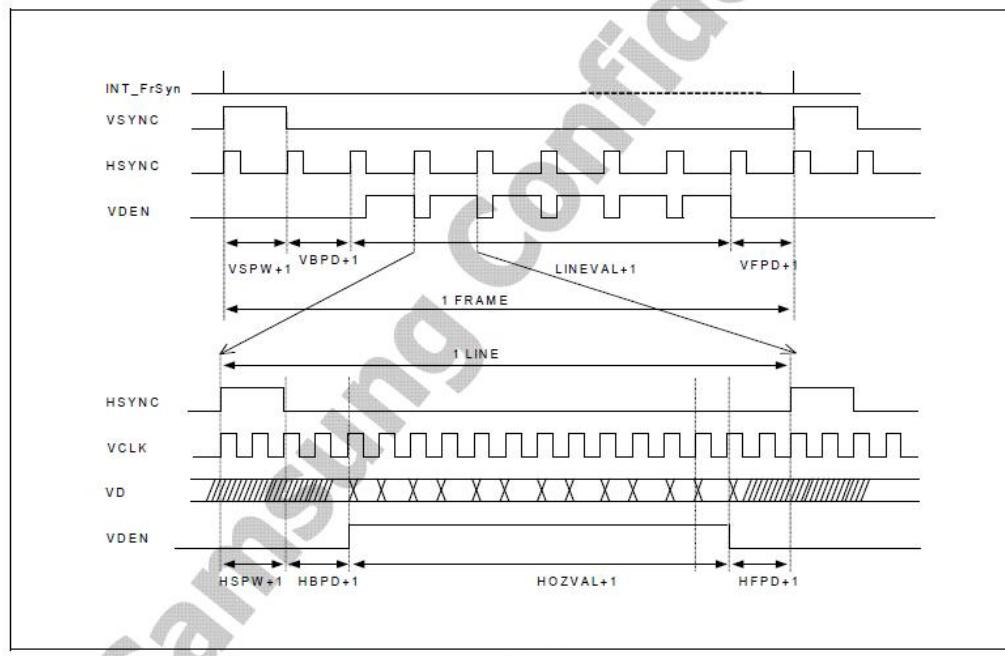
H43-Hsd043I9W1.pdf (p13) 时序图: VSYNC 和 HSYNC 都是低脉冲

S5PV210 芯片手册 (p1207) 时序图: VSYNC 和 HSYNC 都是高脉冲有效, 所以需要反转。

第五步 配置 VIDTCONx, 设置时序和长宽等;

在讲解如何设置寄存器之前, 需要先讲讲 LCD 的时序设置, 见下图:

## 1.3.11.2 LCD RGB Interface Timing



其中各个时序的含义如下图：

- **VBPD(vertical back porch):** 表示在一帧图像开始时, 垂直同步信号以后的无效的行数
- **VFBD(vertical front porch):** 表示在一帧图像结束后, 垂直同步信号以前的无效的行数
- **VSPW(vertical sync pulse width):** 表示垂直同步脉冲的宽度, 用行数计算
- **HBPD(horizontal back porch):** 表示从水平同步信号开始到一行的有效数据开始之间的 VCLK 的个数
- **HFPD(horizontal front porth):** 表示一行的有效数据结束到下一个水平同步信号开始之间的 VCLK 的个数
- **HSPW(horizontal sync pulse width):** 表示水平同步信号的宽度, 用 VCLK 计算

<1> 每一帧的传输过程如下：

- 1) VSYNC 信号有效时, 信号宽度为( $VSPW+1$ )个 HSYNC 信号周期, 即( $VSPW+1$ )个无效行;
- 2) VSYNC 信号脉冲之后, 总共还要进过( $VBPD+1$ )个 HSYNC 信号周期, 有效的行数据才出现。所以, 在 VSYNC 信号有效之后, 还要进过( $VSPW+1+VBPD+1$ )个无效的行;
- 3) 随即发出( $LINEVAL+1$ )行的有效数据;
- 4) 最后是( $VFPD+1$ )个无效的行;

<2> 每一行中像素的传输过程如下：

- 1) HSYNC 信号有效时, 表示一行数据的开始, 信号宽度为( $HSPW+1$ )个 VCLK 信号周期, 即( $HSPW+1$ )个无效像素;
- 2) HSYNC 信号脉冲之后, 还要经过( $HBPD+1$ )个 VCLK 信号周期, 有效的像素数据才出现;
- 3) 随后发出( $HOZVAL+1$ )个像素的有效数据;
- 4) 最后是( $HFPD+1$ )个无效的像素;



有了上面这些知识，设置时序相关的寄存器 VIDTCO0、VIDTCO1 和 VIDTCO2 就没什么问题了，相关代码如下：

```
#define HSPW          0
#define HBPD          (40 - 1)
#define HFDPD         (5 - 1)
#define VSPW          0
#define VBPD          (8 - 1)

#define VFPD          (8 - 1)
// 设置时序
VIDTCO0 = VBPD<<16 | VFPD<<8 | VSPW<<0;
VIDTCO1 = HBPD<<16 | HFDPD<<8 | HSPW<<0;
// 设置长宽
VIDTCO2 = (LINEVAL << 11) | (HOZVAL << 0);
```

首先是寄存器 VIDTCO0，我们参考下面两张图：

1.5.2.5 Video Time Control 0 Register (VIDTCO0, R/W, Address = 0xF800\_0010)

VIDTCO0	Bit	Description	Initial State
VBPDE	[31:24]	Vertical back porch specifies the number of inactive lines at the start of a frame after vertical synchronization period. (Only for even field of YVU interface)	0x00
VBPD	[23:16]	Vertical back porch specifies the number of inactive lines at the start of a frame after vertical synchronization period.	0x00
VFPD	[15:8]	Vertical front porch specifies the number of inactive lines at the end of a frame before vertical synchronization period.	0x00
VSPW	[7:0]	Vertical sync pulse width determines the high-level width of VSYNC pulse by counting the number of inactive lines.	0x00

### 寄存器 VIDTCO0

#### 6.3 Data Input Format

Parallel 24-bit RGB Input Timing Table

Parameters	Symbol	Min.	Typ.	Max.	Unit	Conditions
DCLK frequency	fclk	5	9	12	MHz	
VSYNC period time	Tv	277	288	400	Th	
VSYNC display area	Tvd		272		Th	
VSYNC back porch	Tvbp	3	8	31	Th	
VSYNC front porch	Tvfp	2	8	93	Th	
H SYNC period time	Th	520	525	800	DCLK	
H SYNC display area	Thd		480		DCLK	
H SYNC back porch	Thbp	36	40	255	DCLK	
H SYNC front porch	Thfp	4	5	65	DCLK	

### LCD 芯片手册时序图

- 2) **VFPD:** Vertical front porch, 单位是行。LCD 芯片手册时序图中 VSYNC front porch 就是 VFPD+1，因为时序图中 Th 表示一行需多少个 VCLK，所有 VFPD=8-1 行 (Th)；



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

- 3) **VBPD:** Vertical back porch, 单位是行。LCD 芯片手册时序图中 VSYNC back porch 就是 VBPD+1, 因为时序图中 Th 表示一行需多少个 VCLK, 所有 VBPD=8-1 行 (Th);
- 4) **VSPW:** Vertical sync pulse width, 单位是行。LCD 芯片手册时序图中没直接给出它的值, 需要自己计算: VSPW= VSYNC period time - VSYNC display area - VSYNC back porch - VSYNC front porch=288-272-8-8=0, 所有 VSPW 设为 0 即可;

然后是寄存器 VIDTCON1, 我们参考下面两张图:

1.5.2.6 Video Time Control 1 Register (VIDTCON1, R/W, Address = 0xF800\_0014)

VIDTCON1	Bit	Description	Initial State
VFPDE	[31:24]	Vertical front porch specifies the number of inactive lines at the end of a frame before vertical synchronization period. (Only for the even field of YUV interface).	0
HBPD	[23:16]	Horizontal back porch specifies the number of VCLK periods between the falling edge of HSYNC and start of active data.	0x00
HFPD	[15:8]	Horizontal front porch specifies the number of VCLK periods between the end of active data and rising edge of HSYNC.	0x00
HSPW	[7:0]	Horizontal sync pulse width determines the high-level width of HSYNC pulse by counting the number of VCLK.	0x00

### VIDTCON1 寄存器

## 6.3 Data Input Format

Parallel 24-bit RGB Input Timing Table

Parameters	Symbol	Min.	Typ.	Max.	Unit	Conditions
DCLK frequency	fclk	5	9	12	MHz	
VSYNC period time	Tv	277	288	400	Th	
VSYNC display area	Tvd		272		Th	
VSYNC back porch	Tvbp	3	8	31	Th	
VSYNC front porch	Tvfip	2	8	93	Th	
HSYNC period time	Th	520	525	800	DCLK	
HSYNC display area	Thd		480		DCLK	
HSYNC back porch	Thbp	36	40	255	DCLK	
HSYNC front porch	Thfp	4	5	65	DCLK	

### LCD 芯片手册时序图

- **HBPD:** Horizontal back porch, 单位是 VCLK。LCD 芯片手册时序图中 HSYNC back porch 就是 HBPD +1, 所有 HBPD=40-1 (VCLK);
- **HFPD:** Horizontal front porch, 单位是 VCLK。LCD 芯片手册时序图中 HSYNC front porch 就是 HFPD +1, 所有 HFPD=5-1 (VCLK);
- **HSPW:** Horizontal sync pulse width, 单位是 VCLK。LCD 芯片手册时序图中没直接给出它的值, 需要自己计算: HSPW+1 = HSYNC display area - HSYNC display area - HSYNC back porch - HSYNC front porch=525-480-40-5=0, 所有 HSPW 设为 0 即可;

最后是寄存器 VIDTCON2, 见下图:



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

#### 1.5.2.7 Video Time Control 2 Register (VIDTCON2, R/W, Address = 0xF800\_0018)

VIDTCON2	Bit	Description	Initial State
LINEVAL	[21:11]	Determines the vertical size of display. In the Interlace mode, (LINEVAL + 1) should be even.	0
HOZVAL	[10:0]	Determines the horizontal size of display.	0

NOTE: HOZVAL = (Horizontal display size) -1 and LINEVAL = (Vertical display size) -1.

- ⊕ HOZVAL: HOZVAL = (Horizontal display size) -1=480-1=479
- ⊕ LINEVAL: LINEVAL = (Vertical display size) -1=272-1=271

#### 第六步 配置 WINCON0, 设置 window0 的数据格式;

S5PV210 的 LCD 控制器有 overlay 功能，支持 5 个 window。这里我们只使用 window0，相关的代码设置如下：

```
WINCON0 |= 1<<0;  
WINCON0 &= ~(0xf << 2);  
WINCON0 |= 0xB<<2;
```

我们不需要使用很强大的功能，所有只需设置如下几个 bit 即可：

- ⊕ ENWIN\_F=1, 使能；
- ⊕ BPPMODE\_F= 1011, 24bpp；

#### 第七步 配置 VID0sd0A/B/C, 设置 window0 的坐标系;

相关代码如下：

```
VID0sd0A = (LeftTopX<<11) | (LeftTopY << 0);  
VID0sd0B = (RightBotX<<11) | (RightBotY << 0);  
VID0sd0C = (LINEVAL + 1) * (HOZVAL + 1);
```

设置 window0 的左上角和右下角的坐标和长宽。

#### 第八步 配置 VIDW00ADD0B0 和 VIDW00ADD1B0, 设置 framebuffer 的地址;

相关代码如下：

```
VIDW00ADD0B0 = FB_ADDR;  
VIDW00ADD1B0 = (((HOZVAL + 1)*4 + 0) * (LINEVAL + 1)) & (0xfffffff);
```

VIDWxxADD0	Bit	Description	Initial State
VBASEU_F	[31:0]	Specifies A [31:0] of the start address for Video frame buffer.	0

#### 第九步 配置 SHADOWCON, 使能 dma 通道 0;

相关代码如下：

```
SHADOWCON = 0x1;
```

我们使用的是 Window0，所有需要使能 DMA 通道 0；

#### (二) 函数 lcd\_draw\_pixel():



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

经过 lcd\_init() 初始化 LCD 控制器之后，我们就可以在 LCD 上描绘图形了，代码里的所有绘制图形的函数都是基于 lcd\_draw\_pixel() 这个函数，它的作用是在 LCD 上描绘一个点，通过描绘一个个的点最终会可以组成图形了。在 LCD 上描点的本质就是往 FrameBuffer 中写入颜色值而已。下面我们来分析 lcd\_draw\_pixel()，其完整代码如下：

```
void lcd_draw_pixel(int row, int col, int color)
{
    unsigned long * pixel = (unsigned long *)FB_ADDR;
    *(pixel + row * COL + col) = color;
}
```

其中 FB\_ADDR = 0x23000000，即 framebuffer 的基地址。row 和 col 用来决定偏移，color 是颜色值，只要在 framebuffer 中对应的地址处写入颜色值就可以在 LCD 上描绘出该点。

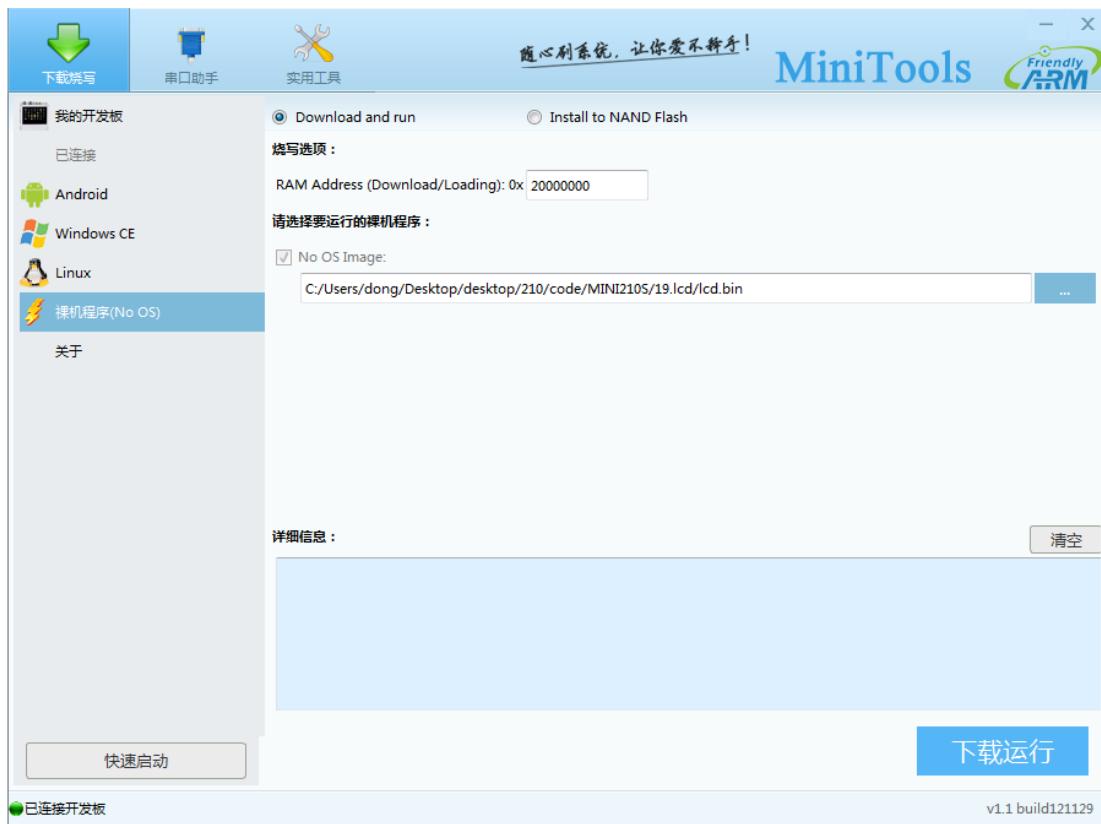
### 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 19.lcd
# make
```

在 19.lcd 目录下会生成 lcd.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：





追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

首先在串口终端会打印如下菜单：

```
#####LCD Test#####
[1] lcd_clear_screen
[2] lcd_draw_cross
[3] lcd_draw_hline
[4] lcd_draw_vline
[5] lcd_draw_circle
Enter your choice:
```

输入 1:LCD 会清屏；

输入 2: LCD 会在左上角打印一个十字；

输入 3: LCD 会在正中间画一条横线；

输入 4: LCD 会在正中间画一条竖线；

## 第二十一章 测试 ADC 转换

### 第一节 S5PV210 的 ADC

S5PV210 的 ADC 可支持 10bit 和 12bit，它支持 10 路输入，然后将输入的模拟的信号转换为 10bit 或者 12bit 的二进制数字信号。在 5MHz 的时钟下，最大转换速率是 1MSPS。本章只是涉及到初步的 ADC 转换，并不会讲解触摸屏相关知识，其结构图如下：

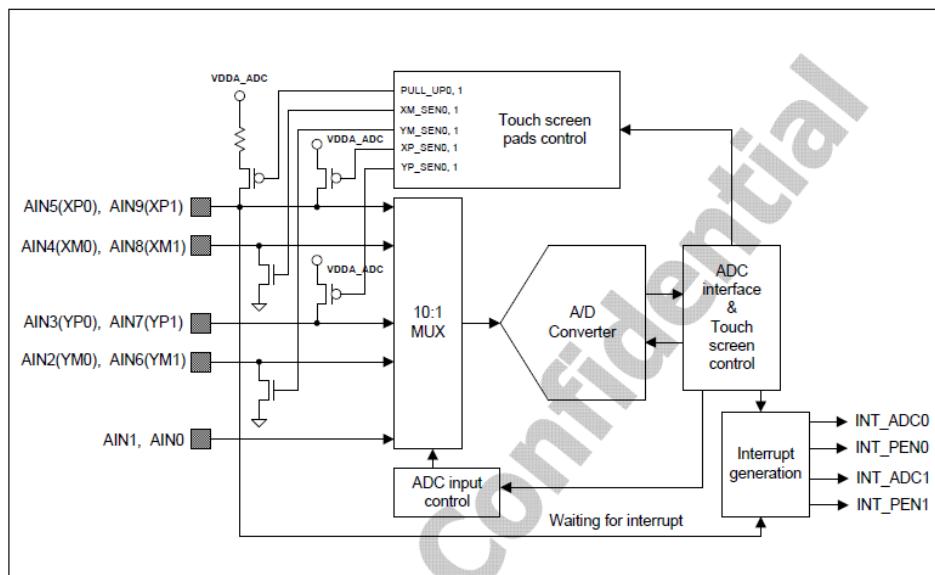
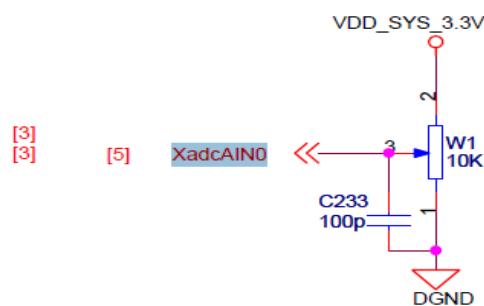


Figure 7-1 ADC and Touch Screen Interface Functional Block Diagram

在 Mini210S 中，adc 相关的原理图如下：

#### AD CONVERT



通道 0 的输入被接到可调电阻上，通过调节可调电阻，adc 能转换出不同的值。

### 第二节 程序相关讲解

完整代码见目录 20. adc。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 1. main.c

main() 函数很简单，主要是调用了 adc\_test() 函数来测试 adc，adc\_test() 的定义位于文件 adc.c 中。

## 2. adc.c

函数 adc\_test() 的代码如下：

```
void adc_test(void)
{
    printf("\r\n#####adc test#####\r\n");
    while(1)
    {
        printf("adc = %d\r\n", read_adc(0));
        delay(0x100000);
    }
}
```

通过一个 while 循环不断的读取通道 0 经过 adc 转换的值，核心函数是 read\_adc()，它主要包括 5 个步骤：

第一步 设置时钟。相关代码如下：

```
TSADCCON0 = (1<<16) | (1 << 14) | (65 << 6);
```

首先使用 12bit adc，然后使能分频，最后设置分频系数为 66



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

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 60MHz 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
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

**第二步 选择通道。代码如下：**

ADCMUX = 0;

设置寄存器 ADCMUX，选择通道 0。

**第三步 启动转换。代码如下：**

TSADCCON0 |= (1 &lt;&lt; 0);

while (TSADCCON0 &amp; (1 &lt;&lt; 0));

首先设置寄存器 TSADCCON0 的 bit[0]，启动 A/D 转换，然后读 bit[0]以确定转换已经启动。

**第四步 检查转换是否完成。代码如下：**

while (!(TSADCCON0 &amp; (1 &lt;&lt; 15)));

读寄存器 TsdACCON0 的 bit[15]，当它为 1 时表示转换结束。

**第五步 读数据，代码如下：**

return (TsdATX0 &amp; 0xffff);

由于我们使用的 12bit 的模式，所以只读寄存器 TsdATX0 的前 12bit。

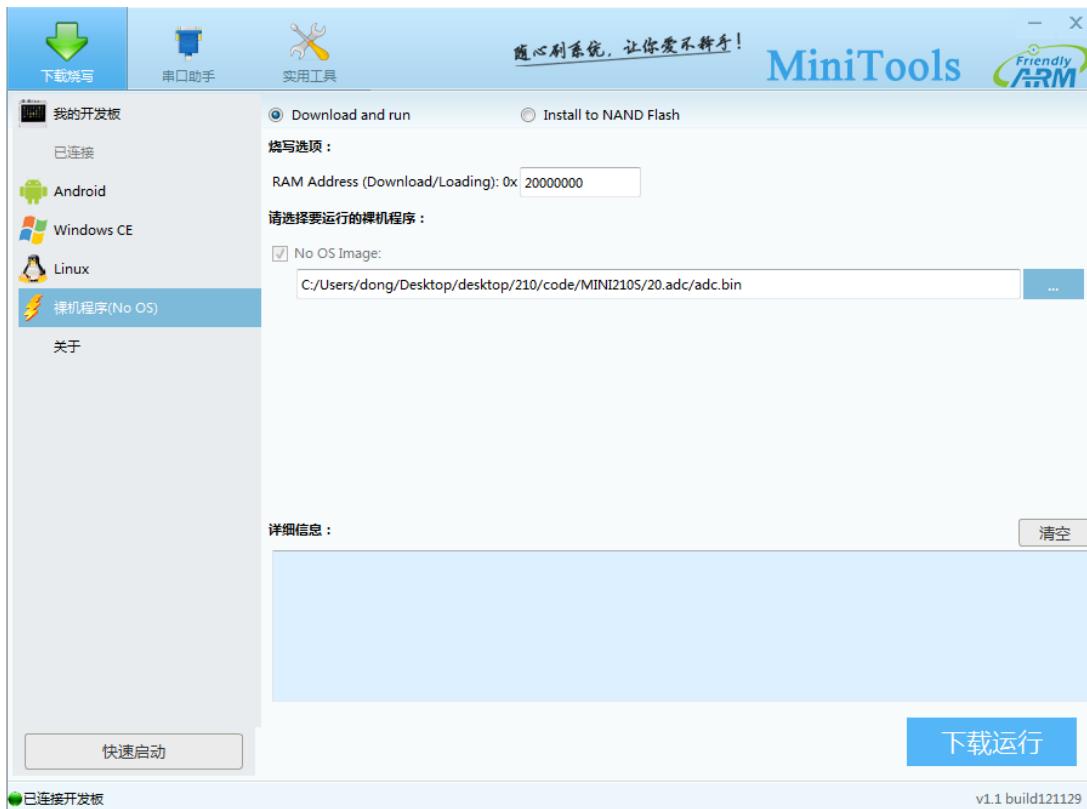
### 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 20.adc  
# make
```

在 20.adc 目录下会生成 adc.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

### 第四节 实验现象

串口终端上会不断的打印出数字，数字的范围是 0~4095，这是因为我们使用的是 12bit 的 ADC。我们通过调节可变电阻可以改变 ACD 转换值，效果如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
#####adc test#####
adc = 1853
adc = 1868
adc = 1863
adc = 1862
adc = 1860
adc = 1860
adc = 1867
adc = 1862
adc = 1858
adc = 1869
adc = 1847
adc = 1877
adc = 1858
```



## 第二十二章 增加命令功能

### 第一节 关于命令功能

这里所说的命令功能类似 linux 中的 shell，输入一个命令，然后程序开始解析运行。这里我们只是象征性的实现几个简单的命令，包括下列命令：

- 1) help: 提供帮助信息
- 2) md: memory display 显示内存
- 3) mw: memory write 写内存
- 4) loadb: 通过串口下载 bin 文件到内存
- 5) go: 运行内存中的 bin 文件

### 第二节 程序详细讲解

完整代码见目录 21. shell。

#### 1. main.c

main 函数相关代码如下：

```
while (1)
{
    printf("Mini210S: ");
    gets(buf);                                // 等待用户输入命令
    argc = shell_parse(buf, argv);             // 解析命令
    command_do(argc, argv);                   // 运行命令
}
```

注释已经一目了然了。函数 shell\_parse() 的定义位于 /BL2/shell.c，command\_do() 的定义位于 /BL2/command.c 中。

#### 2. shell.c

shell\_parse 核心代码如下：

```
while (*buf)                                // 逐个读出字符
{
    if (*buf != ' ' && state == 0)          // 获得一个单词
    {
        argv[argc++] = buf;
        state = 1;
    }
}
```



```
if (*buf == ' ' && state == 1)      // 跳过空格
{
    *buf = '\0';
    state = 0;
}
buf++;
```

逐个解析 buf 里的字符，结果保存在 argc 和 argv 中。例如输入命令：go 0x21000000，最终解析的结果是 argc = 2, argv[0] = “go”，argv[1] = “0x21000000”。

### 3. command.c

首先来看 command\_do()，其大致代码如下：

```
// 根据命令执行对应的代码
int command_do(int argc, char * argv[])
{
    if (argc == 0)
        return -1;

    if (strcmp(argv[0], "help") == 0)           // 执行 help 命令
        help(argc, argv);
    ...
    if (strcmp(argv[0], "loadb") == 0)           // 执行 loadb 命令
        loadb(argc, argv);
    ...
}
```

根据不同的命令，调用不同的执行函数。例如输入 help 命令，调用的执行函数是 help() 函数。

下面来简单地解释各个命令的执行函数：

int help(int argc, char \* argv[]): 打印帮助信息。

int md(int argc, char \* argv[]): 读内存，读写内存都只是简单的指针操作

int mw(int argc, char \* argv[]): 写内存

int loadb(int argc, char \* argv[]): 从串口下载 bin 文件到内存，先获得文件大小，再获得下载地址，最后利用 getc() 函数一个字节一个字节的接受 bin 文件。

int go(int argc, char \* argv[]): 执行内存中的 bin 文件，先定义一个函数指针，然后赋值调用。

## 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

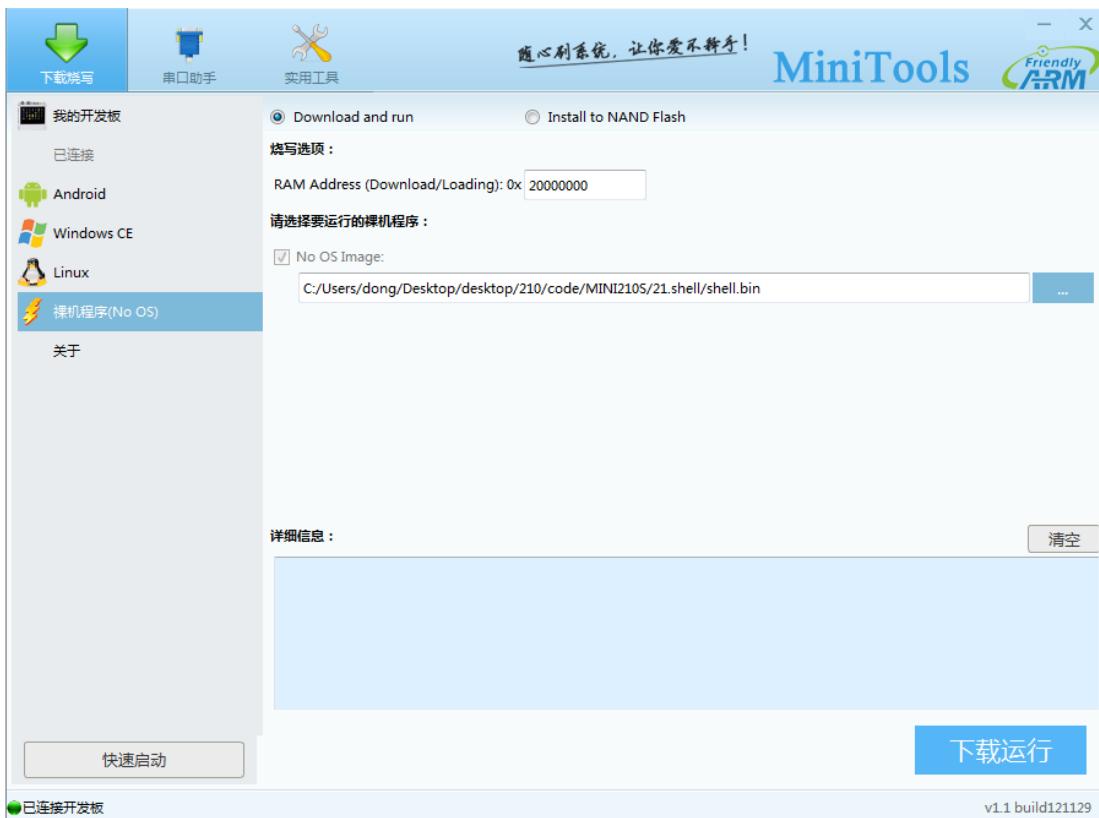
```
# cd 21.shell
```



# make

在 21.shell 目录下会生成 shell.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

#### 第四节 实验现象

串口终端上会打印出“Mini210S:”，然后输出 help 命令会打印帮助信息：



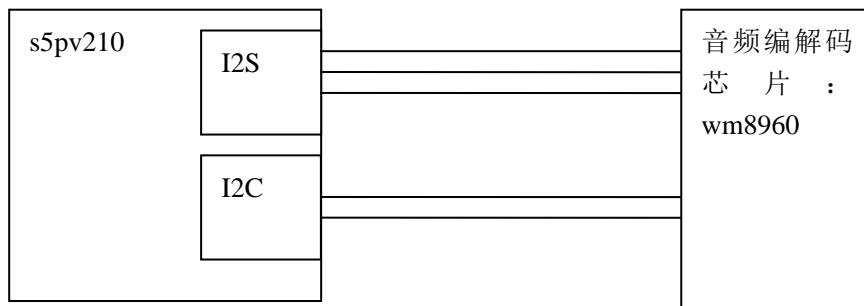
```
mini210s: help
help usage:
md - memory dispaly
mw - memory write
loadb - loadb filesize addr
go - go addr
mini210s: mw 0x21000000 0x12345678
mini210s: md 0x21000000
21000000: 12345678 ffffffff ffffffff ffffffff
21000010: ffffffff ffffffff ffffffff ffffffff
21000020: ffffffff ffffffff ffffffff ffffffff
21000030: ffffffff ffffffff ffffffff ffffffff
21000040: ffffffff ffffffff ffffffff ffffffff
21000050: ffffffff ffffffff ffffffff ffffffff
21000060: ffffffff ffffffff ffffffff ffffffff
21000070: ffffffff ffffffff ffffffff ffffffff
21000080: ffffffff ffffffff ffffffff ffffffff
21000090: ffffffff ffffffff ffffffff ffffffff
210000a0: ffffffff ffffffff ffffffff ffffffff
210000b0: ffffffff ffffffff ffffffff ffffffff
210000c0: ffffffff ffffffff ffffffff ffffffff
210000d0: ffffffff ffffffff ffffffff ffffffff
210000e0: ffffffff ffffffff ffffffff ffffffff
210000f0: ffffffff ffffffff ffffffff ffffffff
mini210s: █
```

既然实现了 loadb 和 go 命令，那么在下一章我们这两个命令来下载运行音频芯片 wm8960 的裸机程序。

## 第二十三章 WM8960 音频播放

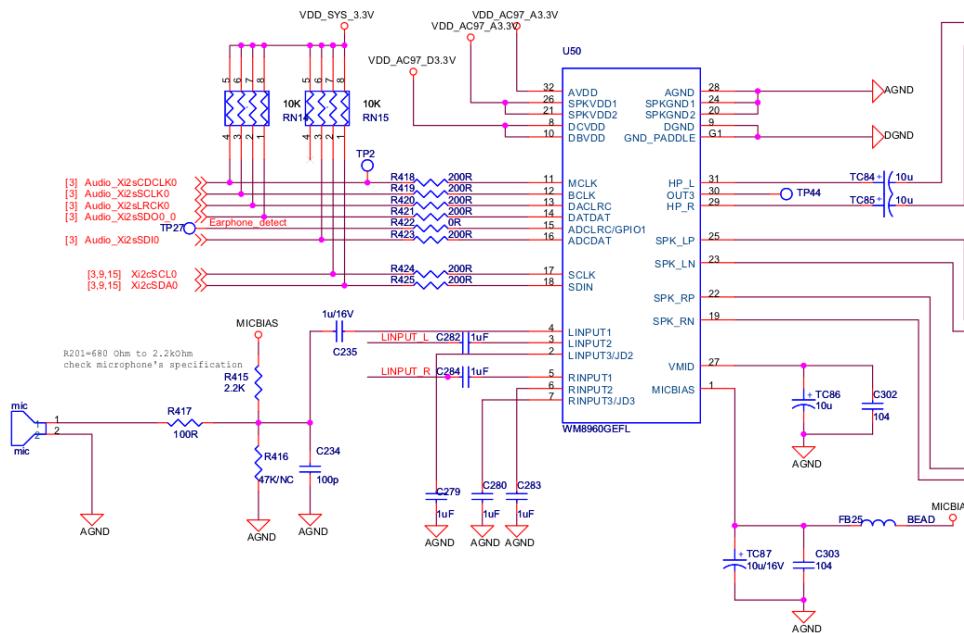
### 第一节 音频播放原理

首先需要申明一下，本章的代码来自网络，参考了亚嵌教育李明老师(论坛 ID:limingth)的帖子：  
<http://www.arm9home.net/read.php?tid=20515&page=1#175657>



S5PV210 通过 i2s 和 i2c 与音频编解码芯片 wm8960 进行交互，其中 i2s 负责只传输声音数据，而 i2c 负责传输控制信息（如音量调节、静音等），wm8960 负责编解码。要驱动 wm8960，我们需要做三件事：(1) 初始化 i2s，(2) 初始化 i2c，(3) 初始化 wm8960。

Mini210S 相关的原理图如下：



### 第二节 程序详细讲解

完整代码见目录 22. audio。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 1. Makefile

在 makefile 中，我们将程序的链接地址设置为 0x21000000，也就是说程序只有位于 0x21000000 才能正常运行，所有我们的程序一开始就必须被下载到内存 0x21000000 处。

## 2. main.c

代码如下：

```
void main(void)
{
    printf("Audio Test\r\n");
    int offset = 0x2E;                                // 音频数据开始的地方
    short * p = (short *)0x22000000;                  // 音频文件应该位于的位置
    iic_init();                                         // 初始化 i2c
    wm8960_init();                                     // 初始化 wm8960
    iis_init();                                         // 初始化 iis
    // 循环播放音频文件
    while (1)
    {
        // polling Primary Tx FIFO full status indication.
        while((IISCON & (1<<8)) == (1<<8));
        IISTXD = *(p+offset);                         // 每次发送 2byte
        offset++;
        if (offset > (882046-0x2e) /2)                // 有多少个 2byte = (文件大小-偏移)/2
            offset = 0x2E;
    }
}
```

main 函数共做了 4 件事：

- 第一步 调用 iic\_init() 初始化 i2c;
- 第二步 调用 wm8960\_init() 初始化 wm8960;
- 第三步 调用 iis\_init() 初始化 i2s;
- 第四步 用 i2s 中发出声音数据，循环播放音频文件；

## 3. audio.c

audio.c 里有几个核心的函数，下面我们来逐个分析。

函数一 iic\_init()，代码如下：

```
void iic_init(void)
{
    GPD1CON |= 0x22;                                // 配置引脚
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

GPD1PUD |= 0x5;

```
I2CCON0 = (1<<7) | (0<<6) | (1<<5) | (0xe);  
I2CSTAT0 = 0x10;
```

}

初始化 i2c 很简单，分三步：

第一步 配置引脚用于 i2c 功能；

第二步 配置 i2c 控制器，包括设置时钟，使能 ack 等；

第三步 使能接收和发送功能；

函数二 iic\_write ()，代码如下：

```
void iic_write(int slave_addr, int addr, int data)  
{  
    // 地址  
    I2CDS0 = slave_addr;  
    // 发送 s 信号和地址  
    I2CSTAT0 = 0xf0;  
  
    // 等待  
    while ((I2CCON0 & 0x10) == 0); // 等待数据发送  
    while ((I2CSTAT0 & 0x1)); // 等待从机发来 ACK  
  
    // 发 7bit 地址和 9bit 数据  
    I2CDS0 = addr<<1 | ((data>>8) & 0x0001);  
    I2CCON0 &= ~(1<<4); // 清中断  
    while ((I2CCON0 & 0x10) == 0); // 等待数据发送  
    while ((I2CSTAT0 & 0x1)); // 等待从机发来 ACK  
    I2CDS0 = (data & 0x00FF);  
    I2CCON0 &= ~(1<<4); // 清中断  
    while ((I2CCON0 & 0x10) == 0); // 等待数据发送  
    while ((I2CSTAT0 & 0x1)); // 等待从机发来 ACK  
  
    // 发 p 信息  
    I2CSTAT0 = 0xd0;  
    I2CCON0 &= ~(1<<4); // 清中断  
  
    // 延时等待  
    int i=0;  
    for(i=0; i<50; i++);  
    return;  
}
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

i2c 的写操作也不难，整体上分三步：

第一步 发 s 信号和设备地址， slave\_addr 的 bit[0:6]是 7bit 的设备地址， bit[7]=0， 表示发送；然后需等待数据发送完成和等待 ack 应答；

第二步 发 16bit 的数据，其中前 7bit 是寄存器偏移地址，后 9bit 是寄存器的值；

第三步 发 p 信号，表示结束；

第四步 延时，等待 p 信号发送成功；

函数三 `wm8960_init ()`，核心代码如下：

```
void wm8960_init(void)
{
#define WM8960_DEVICE_ADDR      0x34
    // 重置
    iic_write(WM8960_DEVICE_ADDR, 0xf, 0x0);
    // 设置电源
    iic_write(WM8960_DEVICE_ADDR, 0x19, 1<<8 | 1<<7 | 1<<6);
    iic_write(WM8960_DEVICE_ADDR, 0x1a, 1<<8 | 1<<7 | 1<<6 | 1<<5 | 1<<4 | 1<<3);
    iic_write(WM8960_DEVICE_ADDR, 0x2F, 1<<3 | 1<<2);
    // 设置时钟
    iic_write(WM8960_DEVICE_ADDR, 0x4, 0x0);
    // 设置 ADC-DAC
    iic_write(WM8960_DEVICE_ADDR, 0x5, 0x0);
    ...
}
```

`wm8960_init()`主要是调用了 `iic_write()`来初始化 `wm8960` 芯片，具体的初始化步骤需要自行详细阅读 `wm8960` 的芯片手册，这里只是简单了整理了网友 limingth 的初始化步骤：

第一步 确定 `wm8960` 的设备地址，查看 `wm8960` 的芯片手册可知其设备地址为 `0x1a`，左移 1 位且低位补 0(表示发送)后，`WM8960_DEVICE_ADDR = 0x34`，reset；

第二步 设置 power1 2 3；

第三步 设置时钟；

第四步 设置 ADC-DAC，注意设置非静音；

第五步 设置 audio interface；

第六步 设置 volume；

第七步 设置 mixer；

这些都是 `wm8960` 芯片手册里相关的内容，需自行阅读该芯片手册。

函数四 `iis_init ()`，核心代码如下：

```
void iis_init(void)
{
    int N;
    // 配置引脚用于 i2s 功能
    GPICON = 0x22222222;
```



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

```
// 设置 i2s 相关时钟
EPLL_CON0 = 0xa8430303;      // MPLL_FOUT = 67.7Mhz
EPLL_CON1 = 0xbcee;          // from linux kernel setting
CLK_SRC0 = 0x10001111;
CLK_CON = 0x1;                // 1 = FOUT_EPLL      MUXI2S_A 00 = Main CLK
// 设置 i2s 控制器
N = 5;
IISPSR = 1<<15 | N<<8;
IISCON |= 1<<0 | (unsigned)1<<31;
IISMOD = 1<<9 | 0<<8 | 1<<10;
}
```

i2s 的初始化整体上分为三个步骤：

第一步 配置引脚用于 i2s 功能；

第二步 设置 i2s 相关时钟，具体包括设置 EPLL\_CON0、1 使 EPLL 输出 67.7Mhz，设置时钟开关 CLK\_SRC0；

第三步 设置 i2s 控制器，具体包括设置分频，时钟选择和发送接收模式；

### 第三节 编译代码和烧写运行

编译代码，在 Fedora 终端执行如下命令：

```
# cd 22.audio
# make
```

在 22.audio 目录下会生成 audio.bin，我们将其烧写到开发板中。

使用第 22 章的程序 shell.bin 将 audio.bin 和音频文件烧写到 DRAM，具体如下：

```
mini210s: loadb 11056 0x21000000
load bin file to address 0x21000000
load finished!
mini210s: loadb 882046 0x22000000
load bin file to address 0x22000000
load finished!
mini210s: go
go to address 0x21000000
Audio Test
```

loadb 的用法如下：

loadb 文件大小 内存地址，每次执行 loadb 之后，就可以用串口终端软件发送文件了。

loadb 11056 0x21000000 把 audio.bin 下载到内存的 0x21000000 处， loadb 882046 0x22000000 把 WindowsXP.wav 下载到内存的 0x22000000 处。最后执行 go 命令就会跳转到 0x21000000 地址处执行 audio.bin 的代码了。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第四节 实验现象

往开发板上插入耳机，就能重复地听到 windows xp 启动时的音乐了。



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

## 第二十四章 LCD 显示字符和图片

### 第一节 LCD 显示字符和图片

在第二十章里我们讲解了如何在 LCD 上描点划线，但是这远远不能满足我们的欲望，所以这里专门用一章来讲解如何绘画字符和图片。这其中并没有什么特别的地方，都是基于描点函数 lcd\_draw\_pixel() 进行扩展，由点汇成字符，同样，也是由点汇成图片。

### 第二节 程序详细讲解

完整代码见目录 23.lcd\_picture。

#### 1. main.c

共 3 个步骤：

第一步： 初始化串口；

第二步： 调用 lcd\_init() 初始化 LCD；

第三步： 调用 lcd\_draw\_bmp() 在 LCD 上描绘图片。图片的数据保存在一个数组中，我们只需要把数组中的值一个个的读出来并写到 FrameBuffer 中即可；

第四步： 画字符，具体是调用了 printf() 打印 “FriendlyARM”。因为我们在 printf() 里不仅调用了 putc()，还调用了字符描绘函数 lcd\_draw\_char()，所以打印信息即会显示在串口终端也会显示在 LCD 上。

#### 2. lcd.c

函数 lcd\_draw\_char() 用于描绘字符，其主要步骤如下：

第一步 获得字模。以传进来的参数为下标，从字模数组 fontdata\_8x16 里取出对应的字模，数组 fontdata\_8x16 的定义位于 font\_8x16.c 中，这个文件时从 linux 内核中抽出来的；

第二步 检查是否需要回车换行。当遇到’ \n ’ 时表示换行，当遇到’ \r ’ 表示回车；

第三步 在 8x16 个像素里描绘一个字符。font\_8x16.c 里定义的每一个字模都是由 8x16bit 组成，每 1bit 对应一个像素，如果某 bit 为 1 则调用 lcd\_draw\_pixel() 将该像素描蓝，为 0 则不描；

第四步 光标移动到下一个 8x16 像素的位置；

函数 lcd\_draw\_bmp() 用于描绘图片，其主要步骤如下：

第一步： 从数组中获取像素的颜色值；

第二步： 调用 lcd\_draw\_pixel() 将数组中的值一个个地在 LCD 上描绘出来，最后组成一张图片；

### 第三节 编译代码和烧写运行

地址：广州市天河区龙口西路龙苑大厦A1栋1705

网址：<http://www.arm9.net>

电话：+86-20-85201025(售前、售后咨询) 技术支持(Tel): 13719442657 传真：+86-20-85261505

E-Mail: [capbily@163.com](mailto:capbily@163.com)(商务或项目合作) [dev\\_friendlyarm@163.com](mailto:dev_friendlyarm@163.com) (技术支持)



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

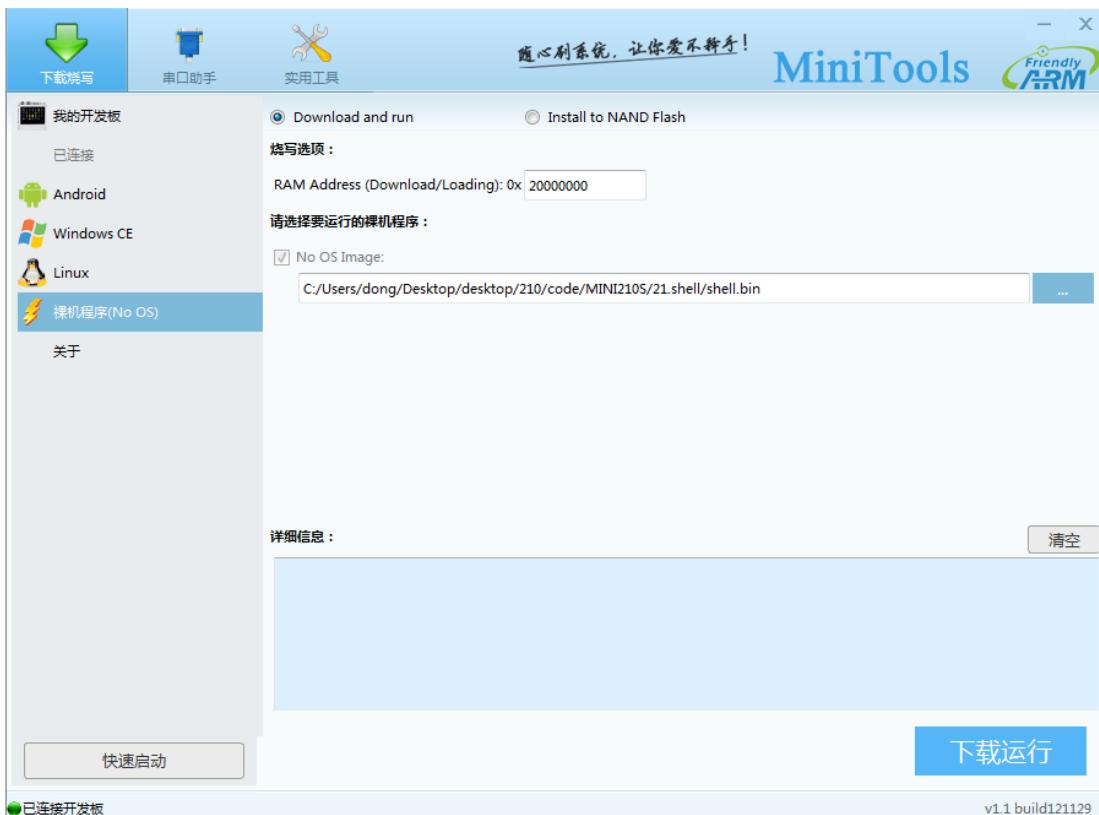
编译代码，在 Fedora 终端执行如下命令：

```
# cd 23_lcd_picture
```

```
# make
```

在 23\_lcd\_picture 目录下会生成 lcd.bin，我们将其烧写到开发板中。

使用 MiniTools 烧写到 DRAM，具体如下：



点击“下载运行”，MiniTools 会把裸机程序下载到 DRAM 然后跳转运行，立马就可以观察到程序的运行效果。

## 第四节 实验现象

LCD 上会显示绚丽的图片，并且会有“FriendlyARM”的字样，效果如下：



追求卓越 创造精品

TO BE BEST

TO DO GREAT

广州友善之臂计算机科技有限公司

