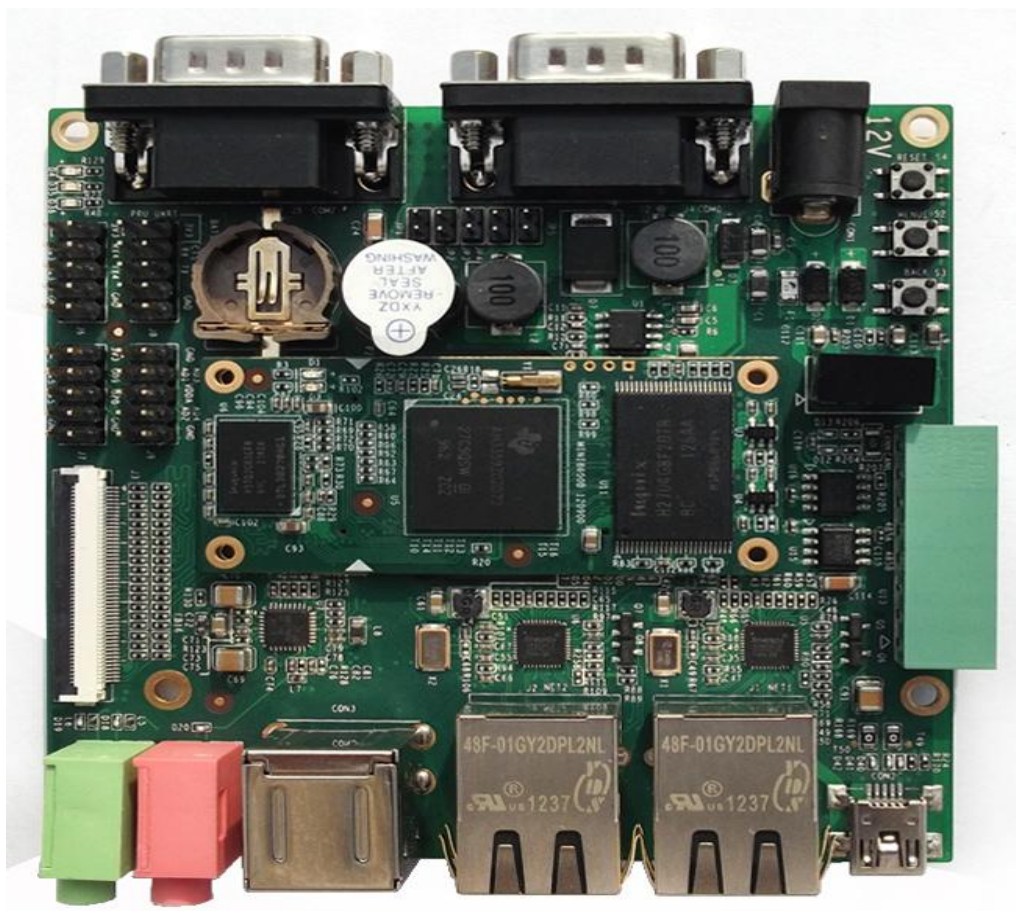


SOM860E 单板机



用户手册

版本 V1.0
2019 年 07 月 12 日

版权声明：

- Microsoft, MS-DOS, Windows 操作系统, Windows Embedded Compact 7 由微软公司授权使用。

版本更新记录：

版本	更新日期	描述
1.0	2019-07-12	初始版本

目录

第 1 章	概述	1
1.1	产品介绍	1
1.2	硬件特性	1
1.2.1	核心板	1
1.2.2	扩展板	3
1.3	硬件尺寸	5
第 2 章	LINUX 操作系统	7
2.1	软件资源	7
2.1.1	软件资源的位置	7
2.1.2	BSP 软件包	8
2.2	嵌入式 LINUX 的组成	9
2.3	开发环境搭建	9
2.3.1	交叉编译工具安装	10
2.3.2	添加环境变量	10
2.4	准备源代码	10
2.4.1	从产品光盘获取源代码	10
2.4.2	用网络工具获取源代码	12
2.5	编译	12
2.6	系统定制	13
2.6.1	U-Boot LOGO 制作	13
2.6.2	进入内核配置菜单	14
2.6.3	内核配置	14
2.6.4	编译内核	15
2.6.5	驱动测试	15
2.7	驱动介绍	16
2.7.1	BSP 的所有驱动源码路径	16

2.7.2	SD/MMC	18
2.7.3	LCDC	19
2.7.4	Audio In/Out	20
2.8	驱动开发	21
2.8.1	GPIO_keys 驱动	21
2.8.2	GPIO_leds 驱动	27
2.9	系统更新	29
2.9.1	TF 卡系统映像更新	30
2.9.2	eMMC 更新/恢复	33
2.10	显示模式配置	36
2.11	测试和演示	36
2.11.1	LED 测试	36
2.11.2	KEYPAD 测试	37
2.11.3	触摸屏测试	37
2.11.4	背光测试	38
2.11.5	ADC 测试	38
2.11.6	RTC 测试	39
2.11.7	TF 卡测试	40
2.11.8	USB DEVICE 测试	41
2.11.9	USB HOST 测试	43
2.11.10	AUDIO 测试	43
2.11.11	VIDEO 测试	45
2.11.12	网络测试	45
2.11.13	CAN 测试	47
2.11.14	RS485 测试	49
2.11.15	串口测试	50
2.11.16	蜂鸣器测试	50
2.11.17	休眠唤醒测试	50

2.11.18	Unique ID	51
2.11.19	GPIO 测试	52
2.11.20	SPI 测试	53
2.11.21	CAMERA 测试	54
2.11.22	WIFI 测试	54
2.11.23	BLUETOOTH 测试	57
2.11.24	Debian 配置	57
2.12	上层开发	59
2.12.1	LED 应用程序开发示例	59
2.12.2	CAN 应用程序开发示例	60
2.12.3	串行接口应用程序开发	68

第 1 章 概述

1.1 产品介绍

SOM860E 一款基于 AM335x 的嵌入式单板机，它采用金手指式核心板加底板的分离式结构设计，方便用户二次开发。主板板载 6 路串口（其中 1 路带隔离 RS485 接口）、1 路带隔离 CAN2.0 接口、2 路千兆以太网口、2 路 USB Host 和 1 路 USB OTG、LCD 触摸屏、TF 卡等接口。支持 Linux-4.1 操作系统。资料提供包括用户手册、PDF 原理图、外扩接口驱动、BSP 源码包、开发工具等，为开发者提供完善的软件开发环境，缩短开发时间，实现面向包括便携式导航系统、数字视频机顶盒、便携式教育/游戏设备、工业自动化、楼宇自动化、人机界面、教学/医疗设备等行业应用产品快速上市。

1.2 硬件特性

SOM860E 评估板是基于 AM335x 处理器，同时也是集成了此芯片主要功能与特性的评估板，以下是板子的特性：

1.2.1 核心板

电气参数

- 工作温度：0 °C~ 70°C
- 环境温度：20% ~ 90%，非冷凝
- 机械尺寸：60mm x 27mm
- 输入电压：3.3V

处理器

- 1GHz ARM Cortex™-A8 32-Bit RISC Microprocessor
 - NEON™ SIMD Coprocessor
 - 32KB/32KB of L1 Instruction/Data Cache with Single-Error Detection (parity)
 - 256KB of L2 Cache with Error Correcting Code (ECC)

- SGX530 Graphics Engine
- Programmable Real-Time Unit Subsystem


存储器

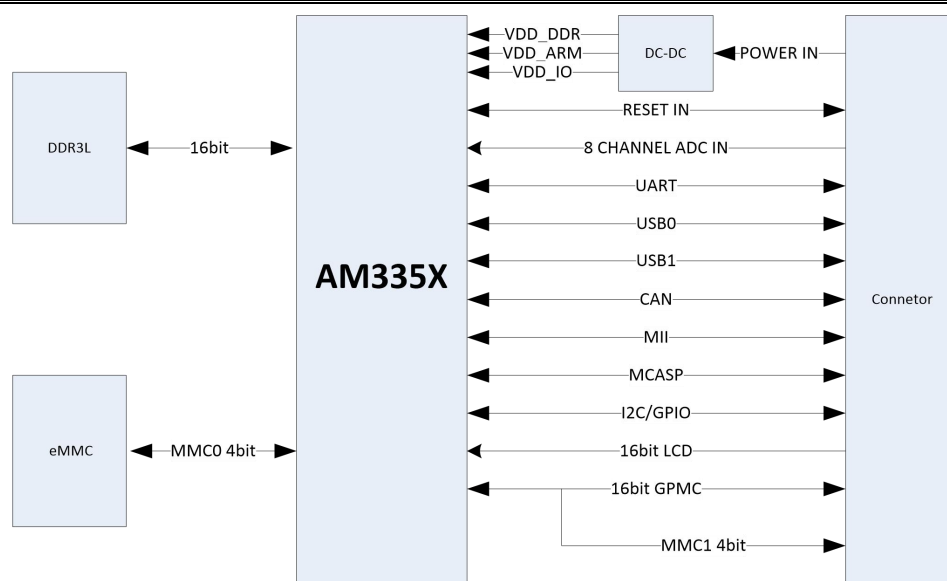
- 8GByte eMMC
- 2*256MB DDR3 SDRAM

板对板连接器和引出接口信号

- 两个0.4mm间距2*40-pin排针
- TFT LCD 信号(支持24-bpp并行RGB接口LCD)
- 2路USB2.0 High-Speed OTG信号
- 6路UART信号
- 1路SPI信号
- 2路10/100/1000Mbps 以太网MAC(EMAC), 带管理数据输入/输出模块(MDIO)
- 1路McASP信号
- 8路12bit ADC接口
- 3路IIC总线信号
- 1路4线SDMMC信号
- GPMC信号

注意:

 UART、IIC、SPI、CAN 存在部分引脚复用，详细情况请参考芯片手册和附带原理图



SOM-860-E

1.2.2 扩展板

电气参数

- 工作温度：0 °C~ 70°C
- 环境湿度：20% ~ 90%，非冷凝
- 机械尺寸：95m x 95m
- 输入电压：12V

音频/视频接口

- LCD/4线电阻触摸屏接口 (24位数据RGB全彩色输出, 50-pin FPC连接器)
- 一个音频输入接口(3.5mm音频接口)
- 一个双声道音频输出接口(3.5mm音频接口)

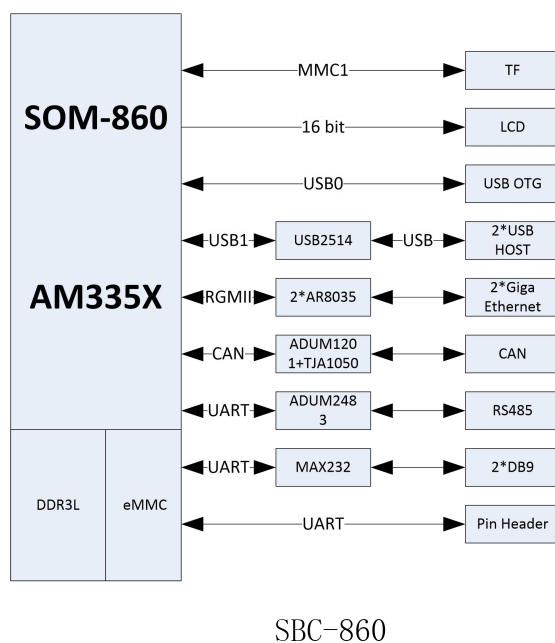
数据传输接口

- 两个10/100/1000Mbps以太网接口(WinCE 7仅支持一个以太网口)
- 一个CAN 2.0接口和一个RS485接口(8 Pin 凤凰端子连接器)
- 一个USB 2.0 High-Speed OTG Ports with Integrated PHY (480Mbps , Mini USB接口)
- 两个USB 2.0 High-Speed HOST Ports with Integrated PHY (480Mbps, USB-A接口)

- 一个TF卡接口（兼容SD/MMC通信，3.3V逻辑）
- 串口
 - UART0, 3线RS232电平，DB9调试串口
 - UART2, 3线RS232电平，DB9普通串口
 - UART3, 3线TTL电平，排针引出
 - UART4, 3线TTL电平，排针引出
 - UART5, 3线TTL电平，排针引出
- GPIO接口

输入接口及其他

- 二个自定义按键（MENU、BACK）
- 一个复位按键
- 一个蜂鸣器
- 一个电源指示灯
- 两个用户自定义灯



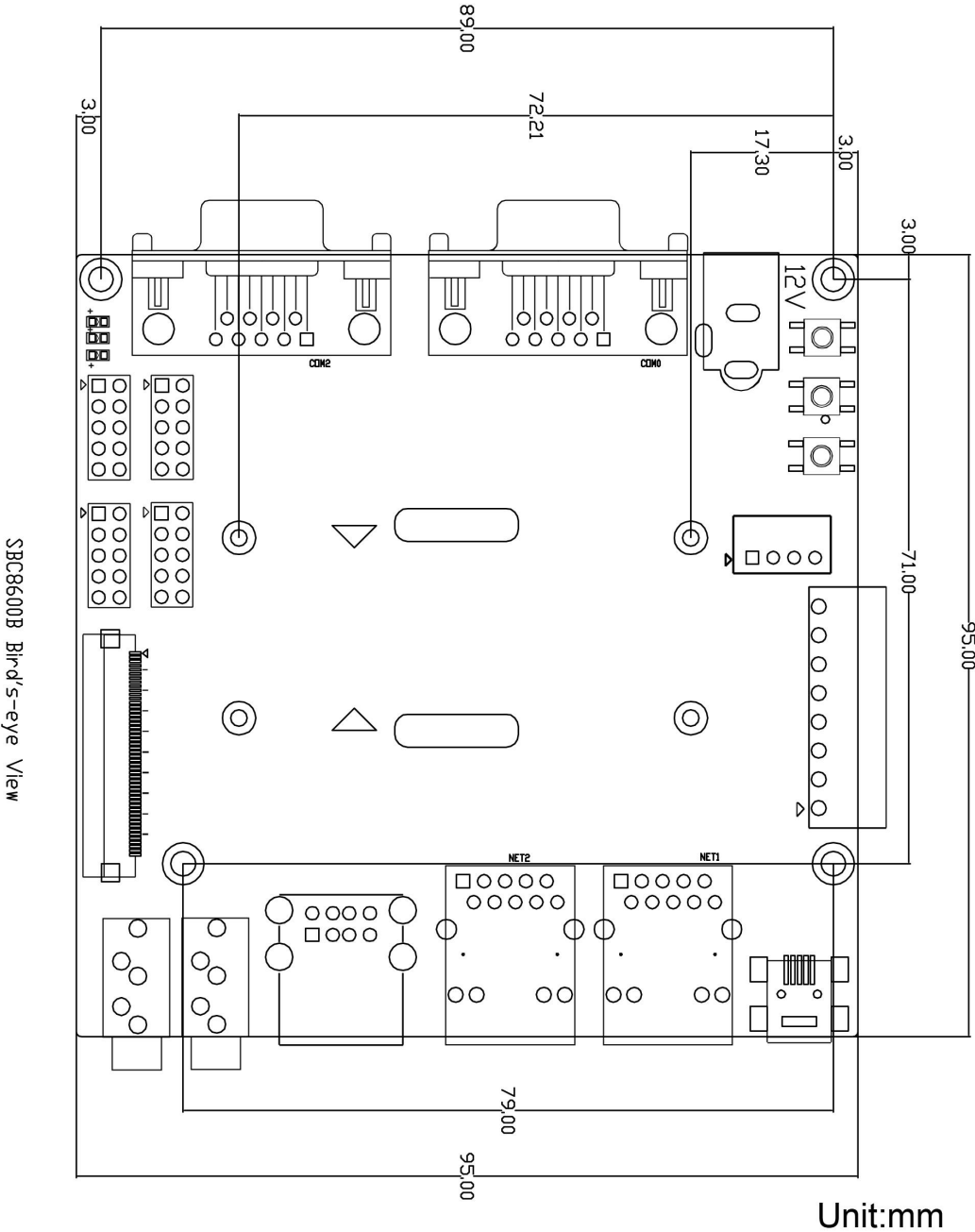



图 1-2 扩展板硬件尺寸

第 2 章 Linux 操作系统

本章节将简要介绍产品附带的 DVD 光盘中的 Linux 软件资源，并且会详细讲解嵌入式 Linux 系统开发的过程、驱动程序及开发、系统更新操作、功能测试、应用程序开发实例等内容。

注意：
 本文档使用 Ubuntu Linux 系统作为操作系统。如果您的 PC 尚未安装 Linux 系统，请参考章节 附录一安装 Ubuntu Linux 系统的内容。

2.1 软件资源

产品附带光盘中包含了 Demo 程序、应用程序、Linux 源代码和工具等，便于用户轻松地使用 SOM860E 开发套件进行 Linux 应用和系统的开发。

2.1.1 软件资源的位置

您可以通过下列表格中的内容，在产品附带的 DVD-ROM 中找到相应的程序和代码；

表 2-1 程序和代码

类别	位置
应用程序	CD\Source\App\uart.tar.xz
源代码	CD\Source\linux-am335x.tar.xz
	CD\Image\rootfs.tar.xz
	CD\Source\u-boot-am335x.tar.xz
工具	CD\Tools
映像	CD\Image

2.1.2 BSP 软件包

下方的表格列出了 BSP 软件包所包含的内容以及文件格式：

表 2-2 BSP软件包内容

名称	备注	源码/二进制文件
Bootloader	SPL	MMC/SD
		FAT
	u-boot	MMC/SD
		FAT
		NET
Kernel	linux-4.1	支持ROM/CRAM/EXT4/FAT/NFS/JFFS2/UBIFS等多种文件系统
Device Driver	Serial	串口驱动
	RTC	硬件时钟驱动
	NET	10/100M/1000M以太网驱动
	CAN	CAN总线驱动
	SPI	SPI驱动
	I2C	I2C驱动
	LCD	TFT LCD 驱动
	Touch Screen	4 线触摸屏控制器驱动
	ADC	4 路普通ADC通道
	MMC/SD	MMC/SD控制器驱动
	USB OTG	USB OTG驱动
	Audio	声卡驱动(支持录/放音)
	Keypad	GPIO键盘驱动
	LED	用户LED灯驱动
	UVC Camera	USB摄像头驱动
	VGA	VGA8000-A模块驱动
	WIFI	USB WIFI模块驱动
	Bluetooth	USB蓝牙模块驱动
Ramdisk	BusyBox	精简文件系统
Rootfs	Debian 8	精简版，不带桌面系统

2.2 嵌入式 Linux 的组成

SOM860E 出厂默认在 eMMC 中写入了 Linux-4.1 操作系统，支持 4.3 寸触摸屏。该系统的基本组成包括 SPL（MLO）、u-boot、kernel、ramdisk 和 rootfs 五个部分。以下为系统结构示意图：



图 2-2 嵌入式Linux系统结构

- 1) SPL是一级引导程序，系统上电后由CPU内部ROM自动拷贝到内部RAM并执行。主要作用为初始化CPU，拷贝u-boot到内存中，然后把控制权交给u-boot；
- 2) u-boot是二级引导程序，主要用于和用户进行交互，提供映像更新、引导内核等功能；
- 3) kernel使用Linux 4.1 内核，根据SOM860E的硬件定制驱动程序；
- 4) ramdisk采用busybox系统，用于支持系统升级；
- 5) rootfs采用开源文件系统EXT4 格式的arm Debian8。

2.3 开发环境搭建

用户使用 SOM860E 进行软件开发之前，必须先搭建 Linux 交叉开发环境，并安装到电脑的 Linux 系统。下面以 Ubuntu 操作系统为例，介绍如何搭建交叉开发环境。

新安装的 Ubuntu 系统建议联网执行下列指令安装必要软件工具，以便接下来的操作顺利进行：

- `sudo apt-get update; sudo apt-get install -y xz-utils ncurses-dev autoconf libtool automake texinfo bison flex libc6:i386 libncurses5:i386 libstdc++6:i386`

注意：

📖 每条指令前都加上了符号“•”，以免指令换行导致误解。

2.3.1 交叉编译工具安装

将产品附带的光盘放入PC的光盘驱动器，Ubuntu会自动将其挂载到/media/cdrom目录下，然后在Ubuntu的终端窗口中执行以下命令来将/media/cdrom/Tools目录下的交叉编译工具解压到\$HOME目录下：

- `mkdir $HOME/tools`
- `cd /media/cdrom/Tools`
- `tar -xvf gcc-linaro-arm-linux-gnueabi-4.9-2014.09_linux.tar.xz -C $HOME/tools`

2.3.2 添加环境变量

执行以下命令来将之前安装的工具添加到临时环境变量中：

- `export PATH=$HOME/tools/gcc-linaro-arm-linux-gnueabi-4.9-2014.09_linux/bin:$HOME/tools:$PATH`
- `export ARCH=arm`
- `export CROSS_COMPILE=arm-linux-`

注意：

📖 您可以将添加环境变量的命令复制到用户目录下的.bashrc 文件中，以便让系统启动时自动添加环境变量。

📖 通过 `echo $PATH` 命令可以查看路径。

2.4 准备源代码

Linux 源代码可从产品附带的光盘获取，也可通过网络工具获取，具体方法如下：

2.4.1 从产品光盘获取源代码


系统所有组成部分的源码位于光盘的 linux/source 目录下，用户在进行开发前需要把它们解压至 Ubuntu 系统：

- `mkdir $HOME/work`
- `cd $HOME/work`
- `tar -xvf /media/cdrom/Source/u-boot-am335x.tar.xz`

- `tar -xvf /media/cdrom/Source/linux-am335x.tar.xz`
- `mkdir rootfs`
- `sudo tar -xvf /media/cdrom/Image/rootfs.tar.xz -C rootfs`

执行完以上操作后，当前目录下会生成 u-boot-am335x、linux-am335x 和 rootfs 目录。

注意：

 源码文件请不要解压到其他位置，以免编译时出错。

2.4.2 用网络工具获取源代码

用户还可以在 ubuntu 系统下通过网络工具下载源码：

1) 通过以下命令获取项目源码和映像等资料：

- `$ cd ~`
- `$ svn co svn://47.107.111.205/TI/SOM860E`

账户名和密码均为 guest

2.5 编译

1) 编译启动代码

SOM860E 支持 MMC/SD 启动与 eMMC 启动，如果不短接 SOM860E 底板上的 **JP5** 跳线，系统优先选择 eMMC 启动，如果短接 **JP5** 跳线，系统优先选择 SD 启动。

下面介绍启动代码映像文件的生成方法：

- `cd u-boot-am335x`
- `make distclean`
- `make am335x_som860e_defconfig`
- `make`

执行完以上操作后，当前目录下会生成我们需要的启动代码映像 MLO 和 u-boot.img。

2) 编译内核

对于 Linux 系统，在 Ubuntu 终端输入如下命令：

- `cd linux-am335x`
- `make distclean`
- `make am335x_som860e_defconfig`
- `make zImage am335x-som860e.dtb`

执行完以上操作后，**arch/arm/boot** 目录下会生成我们需要的 zImage 文件，**arch/arm/boot/dts** 下生成 am335x-som860e.dtb。

2.6 系统定制

Linux内核有很多内核配置选项，用户可以在默认配置的基础上，增加或裁减驱动和一些内核特性，以更适合用户的需要。下面举例说明系统定制的一般流程。

2.6.1 U-Boot LOGO 制作

以 Photoshop 为例简要说明制作 u-boot LOGO 的基本步骤和要点。

- 新建图像



图 2-3 RGB颜色-8 位

- 保存图像，另存为 logo.bmp，弹出对话框：

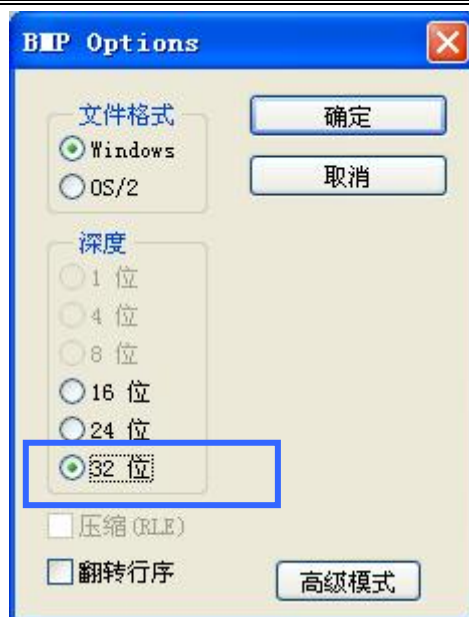


图 2-4 保存为 32 位色格式

2.6.2 进入内核配置菜单

出厂内核源码中提供有默认配置文件：

linux-am335x/arch/arm/configs/am335x_som860e_defconfig

执行以下命令来进入系统配置菜单：

- `cd linux-am335x`
- `make am335x_som860e_defconfig`
- `make menuconfig`

注意：

若输入 `make menuconfig` 系统出错，Ubuntu 系统是需要安装 `ncurses`，`ncurses` 库是字符图形库，用于 `kernel` 的 `make menuconfig`，具体的安装指令：

`sudo apt-get install ncurses-dev。`

2.6.3 内核配置

进入配置菜单后根据定制要求进行修改，下面以 `usb gadget` 模拟 `USB Serial Device` 为例：

进入配置菜单

-> Device Drivers

-> USB support

-> USB Gadget Support

-> USB Gadget Drivers

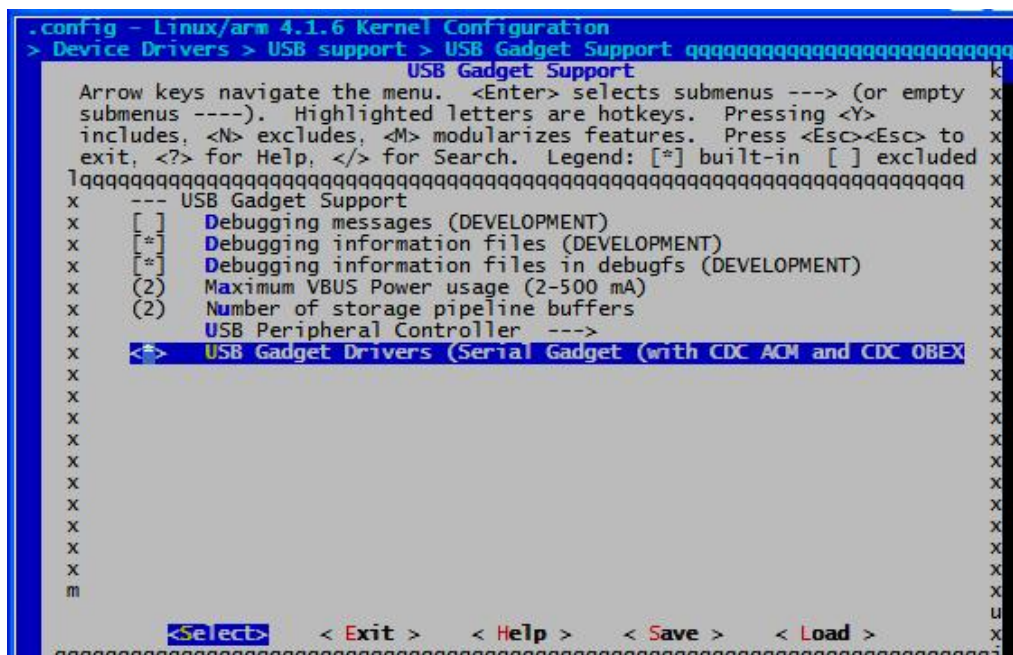


图 2-5 USB Gadget Driver Configuration

选择 “USB Gadget Drivers (Serial Gadget……)” 为<*>, 然后退出并保存配置。

2.6.4 编译内核

请执行以下命令重新编译内核：

- `make am335x-som860e.dtb zImage`

执行完以上操作后，目录下生成新的内核映像 [arch/arm/boot/zImage](#) 更新到 ARM 板上。

2.6.5 驱动测试

连接 miniUSB 端口和 PC 的 USB 插口，设备管理器中将提示发现 Gadget Serial v2.4 硬件设备。安装驱动 **linux-cdc-acm.inf** [Win7 系统会自动搜索服务器中的驱动程序并自动安装]。

linux-cdc-acm.inf 取自内核源码：[linux-am335x/Documentation/usb/linux-cdc-ac](#)

m.inf



图 2-6 Gadget Serial驱动安装完成

PC 上运行串口工具，选择连接对应的串口号（本机上为 COM47）；ARM 板上执行串口测试程序，对应的串口为/dev/ttyGS0，即可看到双方收发的数据。

2.7 驱动介绍

2.7.1 BSP 的所有驱动源码路径:

表 2-3 驱动路径

类别	名称	说明	驱动源码路径
Bootloader	SPL	MMC/SD	drivers/mmc/omap_hsmmc.c
		FAT	fs/
	u-boot	MMC/SD	drivers/mmc/omap_hsmmc.c
		FAT	fs/
		NET	drivers/net/cpsw.c
内核	Linux-4.1	支持ROM/CRAM/EXT4/ FAT/NFS/JFFS2/UBIFS 等多种文件系统	fs/

设备驱动	Serial	串口驱动	drivers/tty/serial/8250/8250_omap.c
	RTC	硬件时钟驱动	drivers/rtc/rtc-omap.c
	NET	10/100M/1000M 以太网驱动	drivers/net/ethernet/ti/ti_cpsw.c
	CAN	CAN总线驱动	drivers/net/can/c_can/c_can_platform.c
	SPI	SPI 驱动	drivers/spi/spi-omap2-mcspi.c
	LCD	TFT LCD 驱动	drivers/gpu/drm/tilcdc/tilcdc_drv.c drivers/video/of_display_timing.c
	Touch Screen	4 线触摸屏控制器驱动	drivers/input/touchscreen/ti_tscadc.c
	ADC	4 路普通ADC通道	drivers/iio/adc/ti_am335x_adc.c
	MMC/SD	MMC/SD控制器驱动	drivers/mmc/host/omap_hsmmc.c
	USB	USB控制器驱动	drivers/usb/musb/musb_am335x.c
	Audio	声卡驱动(支持录/放音)	sound/soc/codecs/sgtl5000.c
	Keypad	GPIO键盘驱动	drivers/input/keyboard/gpio_keys.c
	LED	用户LED灯驱动	drivers/leds/leds-gpio.c
	UVC Camera	USB摄像头驱动	drivers/media/usb/uvc
	VGA	VGA8000-A模块驱动	drivers/video/of_display_timing.c
	WIFI	USB WIFI模块驱动	drivers/net/wireless/rt2x00
	Bluetooth	USB蓝牙模块驱动	drivers/bluetooth/btusb.c

2.7.2 SD/MMC

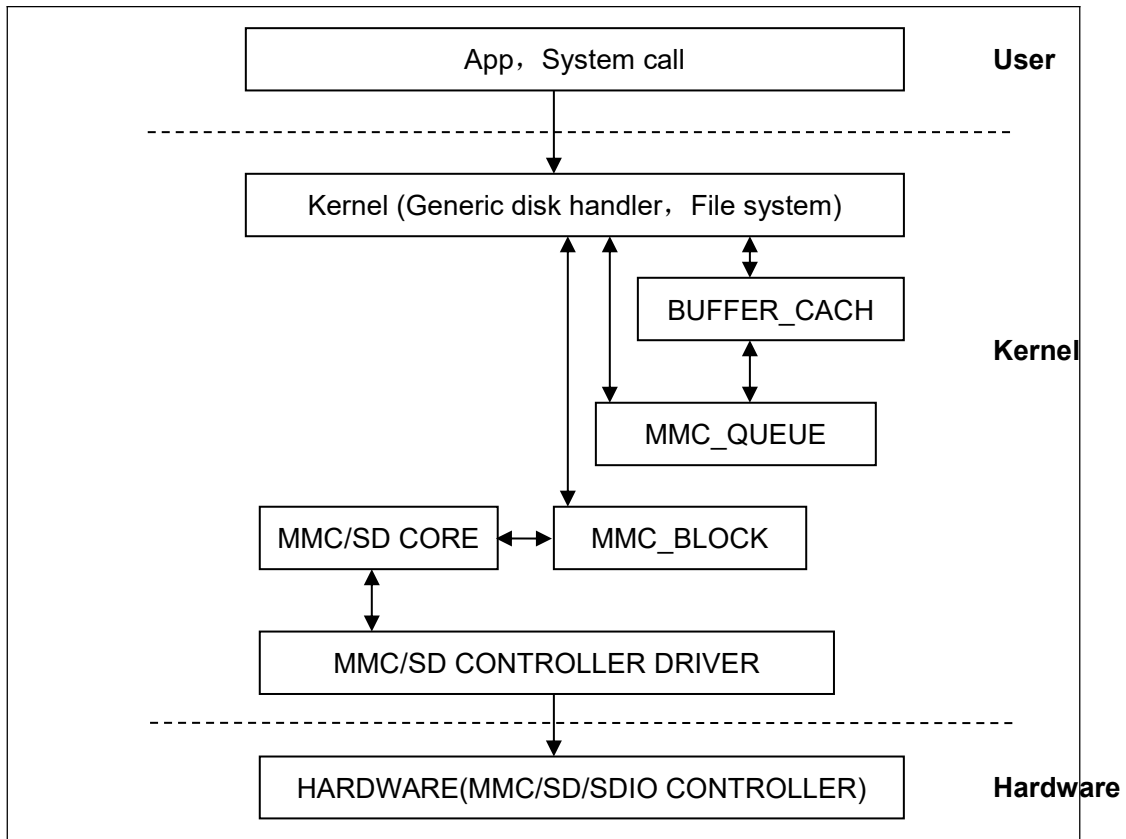


图 2-7 Modular structure for SD/MMC

Linux 下 SD/MMC 卡驱动主要分为 SD/MMC core、mmc_block、mmc_queue、SD/MMC driver 四大部分：

- 1) SD/MMC core 实现 SD/MMC 卡操作中与结构无关的核心代码。
- 2) mmc_block 实现 SD/MMC 卡作为块设备使用时的驱动结构。
- 3) mmc_queue 实现请求队列的管理。
- 4) SD/MMC driver 实现具体的控制器驱动。

驱动参考文件：

linux-am335x/drivers/mmc/

linux-am335x/drivers/mmc/host/omap_hsmmc.c

2.7.3 LCDC

AM335x 下的 LCD 控制器 (LCDC) 是 OMAP-L138 SoC 中 LCDC 的更新版本, 与 OMAP-L138 比较具有如下特点:

- 1) 中断配置和状态寄存器是不同的
- 2) 分辨率最高可支持 2048*2048
- 3) 每像素 24 位有源 TFT 光栅配置

因此 Linux LCD 驱动可用于 LCD_VERSION2 代码下的改进。通过读 PID 寄存器可以检测到 LCDC 版本的更新。

驱动参考文件:

linux-am335x/drivers/video/

linux-am335x/drivers/gpu/drm/tilcdc/tilcdc_panel.c

linux-am335x/drivers/video/of_display_timing.c

2.7.4 Audio In/Out

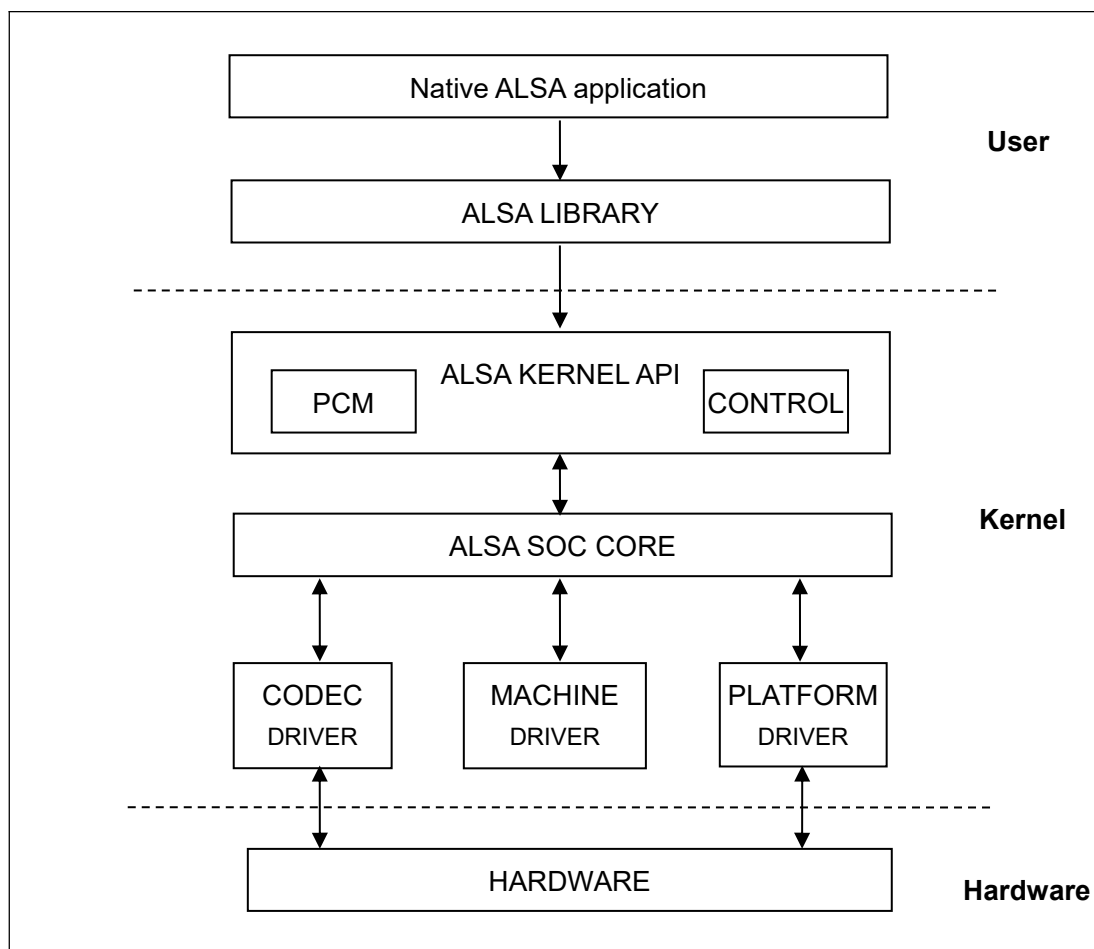


图 2-8 Modular structure for Audio

ASoC 嵌入式音频系统基本分割以下三部分：

- 1) 编解码器驱动：编解码器驱动是一个平台无关，包括 audio controls, audio interface capabilities, codec dapm definition and codec IO functions;
- 2) 平台驱动：平台驱动包括平台相关的 audio dma engine and audio interface drivers (e.g. I2S, AC97, PCM);
- 3) Machine 驱动：Machine 驱动管理任何 machine 相关的 controls and audio events i.e. turning on an amp at start of playback;

驱动参考文件：

linux-am335x/sound/soc/

linux-am335x/sound/soc/davinci/davinci-evm.c

linux-am335x/sound/soc/codecs/sgtl5000.c

2.8 驱动开发

2.8.1 GPIO_keys 驱动

1) 设备定义

linux-am335x/arch/arm/boot/dts/am335x-som860e.dts

配置 gpio0.20 为"menu"键，返回键值"KEY_F1"，低电平触发； gpio2.1 为"back"键，返回键值"KEY_ESC"，低电平触发。

```
gpio_keys {
    compatible = "gpio-keys";
    pinctrl-names = "default";
    pinctrl-0 = <&button_pins>;

    key@0 {
        label = "MENU";
        linux,code = <KEY_F1>;
        gpios = <&gpio0 20 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
    };

    key@1 {
        label = "BACK";
        linux,code = <KEY_ESC>;
        gpios = <&gpio2 1 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
    };
};
```

2) GPIO pinmux 配置

在文件 linux-am335x/arch/arm/boot/dts/am335x-som860e.dts

下配置 GPIO0.20 和 GPIO2.1 为 MODE7(gpio 模式)、AM33XX_PIN_INPUT（配置输入）

```

button_pins: pinmux_button_pins {
    pinctrl-single,pins = <
        0x1B4 (PIN_INPUT_PULLUP | MUX_MODE7) /* xdma_event_intr1.gpio
o0_20 */
        0x08C (PIN_INPUT_PULLUP | MUX_MODE7) /* gpmc_clk.gpio2_1 */
    >;
};

```

3) 驱动设计

linux-am335x/drivers/input/keyboard/gpio_keys.c

a) 调用 platform_driver_register 注册 gpio_keys 驱动

```

static struct platform_driver gpio_keys_device_driver = {
    .probe      = gpio_keys_probe,
    .remove     = gpio_keys_remove,
    .driver     = {
        .name    = "gpio-keys",
        .pm      = &gpio_keys_pm_ops,
        .of_match_table = gpio_keys_of_match,
    }
};

static int __init gpio_keys_init(void)
{
    return platform_driver_register(&gpio_keys_device_driver);
}

static void __exit gpio_keys_exit(void)
{
    platform_driver_unregister(&gpio_keys_device_driver);
}

late_initcall(gpio_keys_init);
module_exit(gpio_keys_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Phil Blundell <pb@handhelds.org>");
MODULE_DESCRIPTION("Keyboard driver for GPIOs");
MODULE_ALIAS("platform:gpio-keys");

```

b) 调用 input_register_device 注册 input 驱动

```

static int __devinit gpio_keys_probe(struct platform_device *pdev)
{
    if (!pdata) {
        pdata = gpio_keys_get_devtree_pdata(dev);
        if (IS_ERR(pdata))
            return PTR_ERR(pdata);
    }
    ...

    input = devm_input_allocate_device(dev);
    ...

    for (i = 0; i < pdata->nbuttons; i++) {
        const struct gpio_keys_button *button = &pdata->buttons[i];
        struct gpio_button_data *bdata = &ddata->data[i];

        error = gpio_keys_setup_key(pdev, input, bdata, button);

        error = gpio_keys_setup_key(pdev, bdata, button);
        if (error)
            return error;

        if (button->wakeup)
            wakeup = 1;
    }
    error = sysfs_create_group(&pdev->dev.kobj, &gpio_keys_attr_group);
    if (error) {
        dev_err(dev, "Unable to export keys/switches, error: %d\n",
            error);
        goto fail2;
    }

    error = input_register_device(input);
    if (error) {
        dev_err(dev, "Unable to register input device, error: %d\n",
            error);
        goto err_remove_group;
    }
    ...
}

```

c) 申请 gpio，配置 gpio 为输入，注册 gpio 中断

```

static int gpio_keys_setup_key(struct platform_device *pdev,
    struct input_dev *input,
    struct gpio_button_data *bdata,

```

```
const struct gpio_keys_button *button)
{
    const char *desc = button->desc ? button->desc : "gpio_keys";
    struct device *dev = &pdev->dev;
    irq_handler_t isr;
    unsigned long irqflags;
    int irq;
    int error;

    bdata->input = input;
    bdata->button = button;
    spin_lock_init(&bdata->lock);

    if (gpio_is_valid(button->gpio)) {

        error = devm_gpio_request_one(&pdev->dev, button->gpio,
                                      GPIOF_IN, desc);
        if (error < 0) {
            dev_err(dev, "Failed to request GPIO %d, error %d\n",
                    button->gpio, error);
            return error;
        }

        if (button->debounce_interval) {
            error = gpio_set_debounce(button->gpio,
                                      button->debounce_interval * 1000);
            /* use timer if gpiolib doesn't provide debounce */
            if (error < 0)
                bdata->software_debounce =
                    button->debounce_interval;
        }

        if (button->irq) {
            bdata->irq = button->irq;
        } else {
            irq = gpio_to_irq(button->gpio);
            if (irq < 0) {
                error = irq;
                dev_err(dev,
                        "Unable to get irq number for GPIO %d, error %d\n",
                        button->gpio, error);
                return error;
            }
        }
    }
}
```

```
    }
    bdata->irq = irq;
}

INIT_DELAYED_WORK(&bdata->work, gpio_keys_gpio_work_func);

isr = gpio_keys_gpio_isr;
irqflags = IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING;

} else {
    if (!button->irq) {
        dev_err(dev, "No IRQ specified\n");
        return -EINVAL;
    }
    bdata->irq = button->irq;

    if (button->type && button->type != EV_KEY) {
        dev_err(dev, "Only EV_KEY allowed for IRQ buttons.\n");
        return -EINVAL;
    }

    bdata->release_delay = button->debounce_interval;
    setup_timer(&bdata->release_timer,
                gpio_keys_irq_timer, (unsigned long)bdata);

    isr = gpio_keys_irq_isr;
    irqflags = 0;
}

input_set_capability(input, button->type ?: EV_KEY, button->code);

/*
 * Install custom action to cancel release timer and
 * workqueue item.
 */
error = devm_add_action(&pdev->dev, gpio_keys_quiesce_key, bdata);
if (error) {
    dev_err(&pdev->dev,
            "failed to register quiesce action, error: %d\n",
            error);
    return error;
}
```

```
/*
 * If platform has specified that the button can be disabled,
 * we don't want it to share the interrupt line.
 */
if (!button->can_disable)
    irqflags |= IRQF_SHARED;

error = devm_request_any_context_irq(&pdev->dev, bdata->irq,
                                     isr, irqflags, desc, bdata);

if (error < 0) {
    dev_err(dev, "Unable to claim irq %d; error %d\n",
            bdata->irq, error);
    return error;
}

return 0;
}
```

d) 中断处理

按键被按下，产生中断，汇报键值：

```
static irqreturn_t gpio_keys_gpio_isr(int irq, void *dev_id)
{
    ...

    mod_delayed_work(system_wq,
                      &bdata->work,
                      msecs_to_jiffies(bdata->software_debounce));
    ...
}

static void gpio_keys_gpio_work_func(struct work_struct *work)
{
    ...

    gpio_keys_gpio_report_event(bdata);
    ...
}

static void gpio_keys_gpio_report_event(struct gpio_button_data *bdata)
{
    const struct gpio_keys_button *button = bdata->button;
    struct input_dev *input = bdata->input;
    unsigned int type = button->type ?: EV_KEY;
    int state = (gpio_get_value_cansleep(button->gpio) ? 1 : 0) ^ button->active_low;
```

```
if (type == EV_ABS) {
    if (state)
        input_event(input, type, button->code, button->value);
    } else {
        input_event(input, type, button->code, !state);
    }
    input_sync(input);
}
```

2.8.2 GPIO_leds 驱动

1) 设备定义

linux-am335x/arch/arm/boot/dts/am335x-som860e.dts

配置 GPIO2.5 为 “sys”（系统心跳灯）为高电平有效。

```
leds {
    compatible = "gpio-leds";
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&user_leds_default>;
    pinctrl-1 = <&user_leds_sleep>;

    led@0 {
        label = "sys";
        gpios = <&gpio2 5 GPIO_ACTIVE_HIGH>;
        linux,default-trigger = "heartbeat";
        default-state = "off";
    };
};
```

2) GPIO pinmux 配置

linux-am335x/arch/arm/boot/dts/am335x-som860e.dts

配置 GPIO2.5 为 MODE7(gpio 模式)、AM33XX_PIN_OUTPUT（配置输出）

```
user_leds_default: pinmux_user_leds_default {
    pinctrl-single,pins = <
        /* leds */
        0x09c (PIN_OUTPUT_PULLUP | MUX_MODE7) /*
gpmc_be0n_cle.gpio2_5 */
    >;
};
```

3) 驱动设计

linux-am335x/drivers/leds/leds-gpio.c

a) 调用 platform_driver_register 注册 gpio_leds 驱动

```
static struct platform_driver gpio_led_driver = {
    .probe      = gpio_led_probe,
    .remove     = gpio_led_remove,
    .driver     = {
        .name    = "leds-gpio",
        .of_match_table = of_gpio_leds_match,
        .pm      = &gpio_led_pm_ops,
    },
};

module_platform_driver(gpio_led_driver);

MODULE_AUTHOR("Raphael Assenat <raph@8d.com>, Trent Piepho
<tpiepho@freescale.com>");
MODULE_DESCRIPTION("GPIO LED driver");
MODULE_LICENSE("GPL");
MODULE_ALIAS("platform:leds-gpio");
```

b) 申请 gpio，调用 led_classdev_register 注册 led_classdev 驱动

```
static int gpio_led_probe(struct platform_device *pdev)
{
    ...

    if (pdata && pdata->num_leds) {
        priv = devm_kzalloc(&pdev->dev,
                           sizeof_gpio_leds_priv(pdata->num_leds),
                           GFP_KERNEL);

        if (!priv)
            return -ENOMEM;

        priv->num_leds = pdata->num_leds;
        for (i = 0; i < priv->num_leds; i++) {
            ret = create_gpio_led(&pdata->leds[i],
                                &priv->leds[i],
                                &pdev->dev, pdata->gpio_blink_set);

            if (ret < 0) {
                /* On failure: unwind the led creations */
                for (i = i - 1; i >= 0; i--)
                    delete_gpio_led(&priv->leds[i]);

                return ret;
            }
        }
    }
}
```

```
    }  
    }  
    } else {  
        priv = gpio_leds_create(pdev);  
        if (IS_ERR(priv))  
            return PTR_ERR(priv);  
    }  
  
    platform_set_drvdata(pdev, priv);  
  
    return 0;  
}  
  
static int create_gpio_led(const struct gpio_led *template,  
    struct gpio_led_data *led_dat, struct device *parent,  
    int (*blink_set)(struct gpio_desc *, int, unsigned long *,  
        unsigned long *))  
{  
    ...  
  
    ret = devm_gpio_request_one(parent, template->gpio, flags, template->name);  
    ...  
  
    ret = gpiod_direction_output(led_dat->gpiod, state);  
    ...  
  
    return led_classdev_register(parent, &led_dat->cdev);  
}
```

- c) 用户通过访问 [/sys/class/leds/xxx/brightness](#) 文件，调用 `gpio_led_set` 函数，控制 led 灯的状态

```
static void gpio_led_set(struct led_classdev *led_cdev,  
    enum led_brightness value)  
{  
    ...  
  
    gpiod_set_value(led_dat->gpiod, level);  
}
```

2.9 系统更新

SOM860E 支持从 TF 卡和 eMMC 启动系统，下面将详细介绍两种不同的系统映像更新方式。

2.9.1 TF 卡系统映像更新

1) TF 卡格式化

请使用 **HP USB Disk Storage Format Tool 2.0.6** 格式 TF 卡。

获取软件: **CD\Tools\SDFormatter HP.zip**。

- a) 把 TF 卡插入 PC 机读卡器中
- b) 打开 **HP USB Disk Storage Format Tool**, 出现类似提示如下:

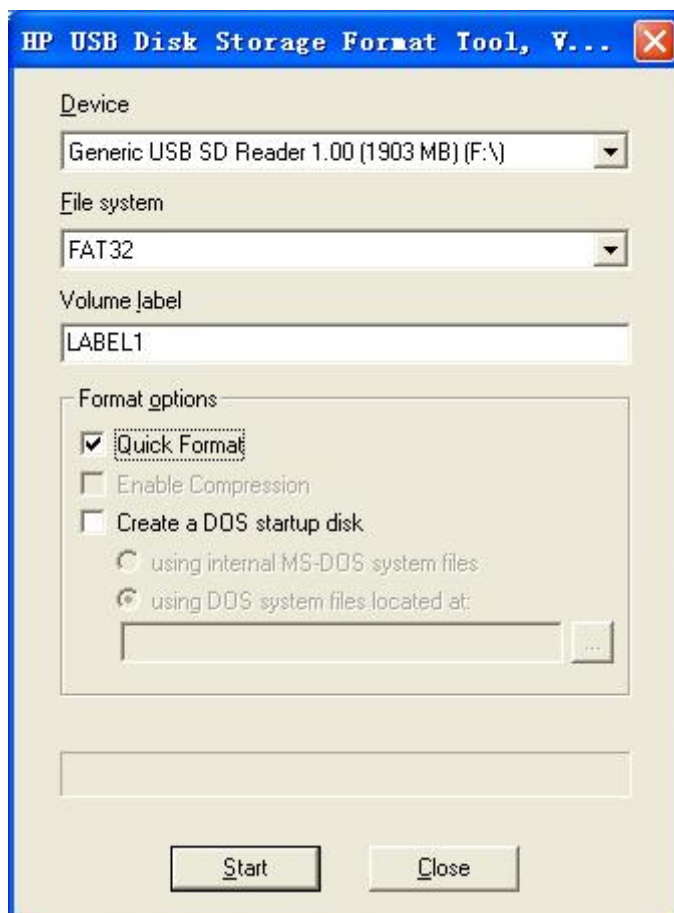


图 2-9 HP Format Tool

- c) 选择” FAT32 “系统格式
- d) 点击” Start”
- e) 等待格式化完成, 点击” OK”。

注意：

- 使用其他版本的 HP USB Disk Storage Format Tool 格式化 TF 卡时，可能会出现不能从 TF 卡启动的情况
- 使用 HP USB Disk Storage Format Tool 格式化 TF 卡时将清除 TF 存储卡的分区。

2) 映像更新

将 **CD/Image** 目录下的所有文件拷贝到 TF 卡上，将 TF 卡接入板子，短接跳线 **JP5**，上电启动，串口信息显示如下：

注意：

- 默认 800x600 LCD 显示。如想使用其他的显示设备，在启动时进入 u-boot 设置显示方式，再输入 boot 继续启动即可。显示方式的设置方法请参考【2.10 显示模式配置】。
- 必须用跳线帽短接板上的 JP5 引脚**，使 SOM860E 从 TF 卡启动升级。

```
U-Boot SPL 2015.07-g4e2a180 (Jul 11 2019 - 15:17:51)
reading args
spl_load_image_fat_os: error reading image args, err --1
reading u-boot.img
reading u-boot.img
U-Boot 2015.07-g4e2a180 (Jul 11 2019 - 15:17:51 +0800)
    Watchdog enabled
I2C:   ready
DRAM:  512 MiB
NAND:  0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
reading uboot.env
** Unable to read "uboot.env" from mmc0:1 **
Using default environment
reading logo.bmp
FAT: Misaligned buffer address (8000a002)
Net:   <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot:  1
switch to partitions #0, OK
mmc1 is current device
SD/MMC found on device 1
reading boot.scr
```

```
** Unable to read file boot.scr **
reading uEnv.txt
300 bytes read in 5 ms (58.6 KiB/s)
Loaded env from uEnv.txt
Importing environment from mmc1 ...
Running uenvcmd ...
reading /am335x-som860e.dtb
38105 bytes read in 13 ms (2.8 MiB/s)
reading /zlimage
4111720 bytes read in 326 ms (12 MiB/s)
reading ramdisk.img
14143658 bytes read in 1100 ms (12.3 MiB/s)
Booting from ramdisk ...
Kernel image @ 0x82000000 [ 0x000000 - 0x3ebd68 ]
## Loading init Ramdisk from Legacy Image at 88080000 ...
   Image Name:
   Created:      2019-06-13   6:38:03 UTC
   Image Type:   ARM Linux RAMDisk Image (gzip compressed)
   Data Size:    14143594 Bytes = 13.5 MiB
   Load Address: 81600000
   Entry Point:  81600000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 88000000
   Booting using the fdt blob at 0x88000000
   Loading Ramdisk to 8f282000, end 8ffff06a ... OK
   Loading Device Tree to 8f275000, end 8f2814d8 ... OK
Starting kernel ...
[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Initializing cgroup subsys cpuset
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Initializing cgroup subsys cpuacct
[ 0.000000] Linux version 4.1.6 (chengpg@Ubuntu18) (gcc version 4.9.2 20140904
(prerelease) (crosstool-NG linaro-1.13.1-4.9-2014.09 - Linaro GCC 4.9-2014.09) ) #1
PREEMPT Thu Jul 11 15:20:30 CST 2019
[ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c5387d
[ 0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[ 0.000000] Machine model: TI AM335x SOM860E
[ 0.000000] cma: Reserved 24 MiB at 0x9e800000
[ 0.000000] Memory policy: Data cache writeback
[ 0.000000] CPU: All CPU(s) started in SVC mode.
[ 0.000000] AM335X ES2.1 (sgx neon )
.....
```

```
[ 2.709173] RAMDISK: gzip image found at block 0
[ 5.827792] EXT4-fs (ram0): mounting ext2 file system using the ext4 subsystem
[ 5.837166] EXT4-fs (ram0): mounted filesystem without journal. Opts: (null)
[ 5.844376] VFS: Mounted root (ext2 filesystem) on device 1:0.
[ 5.851233] devtmpfs: mounted
[ 5.855765] Freeing unused kernel memory: 740K (c0a59000 - c0b12000)
[ 5.911728] EXT4-fs (ram0): re-mounted. Opts: (null)
Starting logging: OK
Populating /dev using udev: [ 6.106777] udevd[97]: starting version 2.1.1
[ 6.121516] random: udevd urandom read with 18 bits of entropy available
[ 7.269066] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may
be corrupt. Please run fsck.
[ 7.370398] FAT-fs (mmcblk1p1): Volume was not properly unmounted. Some data may
be corrupt. Please run fsck.
done
Starting watchdog...
Initializing random number generator... done.
Starting system message bus: done
Starting network...
[ 8.080994] net eth0: initializing cpsw version 1.12 (0)
[ 8.167483] net eth0: phy found : id is : 0x4dd072
ssh-keygen: generating new host keys: ED25519
Starting sshd: OK
```

超级终端显示上述信息，则代表已经成功从 TF 卡启动 Linux 系统。

2.9.2 eMMC 更新/恢复

eMMC 启动映像的更新需要借助于 ramdisk 来完成，无论 eMMC 中是否已安装系统，都可以利用 ramdisk 系统对其更新系统。

1) 准备

- a) 用 **HP USB Disk Storage Format Tool 2.0.6** 将 TF 卡格式化为 FAT 或 FAT32 文件系统；
- b) 将光盘里的 **MLO, u-boot.img, am335x-som860e.dtb, zImage, ramdisk.img, rootfs.tar.xz, logo.bmp, uEnv.txt, uEnv_eMMC.txt** 映像文件拷贝到 TF 卡中。

2) 更新

- a) 将带有系统映像的 TF 卡插入 ARM 板，用跳线帽短接 **JP5** 引脚，上电启动，

不用进入 u-boot 命令行。

```
U-Boot SPL 2015.07-gba89fddf4-dirty (Apr 09 2019 - 14:04:26)
reading args
spl_load_image_fat_os: error reading image args, err - -1
reading u-boot.img
reading u-boot.img
U-Boot 2015.07-gba89fddf4-dirty (Apr 09 2019 - 14:04:26 +0800)

  Watchdog enabled
I2C:  ready
DRAM:  512 MiB
NAND:  0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
** No partition table - mmc 0 **
Using default environment
reading logo.bmp
FAT: Misaligned buffer address (8000a002)
Net:   <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot:  1
switch to partitions #0, OK
mmc1 is current device
SD/MMC found on device 1
reading boot.scr
** Unable to read file boot.scr **
reading uEnv.txt
300 bytes read in 4 ms (73.2 KiB/s)
Loaded env from uEnv.txt
Importing environment from mmc1 ...
Running uenvcmd ...
reading /am335x-som860e.dtb
37689 bytes read in 11 ms (3.3 MiB/s)
reading /zlimage
4048952 bytes read in 318 ms (12.1 MiB/s)
reading ramdisk.img
14143658 bytes read in 1101 ms (12.3 MiB/s)
Booting from ramdisk ...
Kernel image @ 0x82000000 [ 0x000000 - 0x3dc838 ]
## Loading init Ramdisk from Legacy Image at 88080000 ...
```

```
.....
Starting logging: OK
Populating /dev using udev: [ 6.061087] udevd[96]: starting version 2.1.1
[ 6.067224] random: udevd urandom read with 27 bits of entropy available
done
Starting watchdog...
Initializing random number generator... done.
Starting system message bus: done
Starting network...
[ 7.910771] net eth0: initializing cpsw version 1.12 (0)
[ 7.997811] net eth0: phy found : id is : 0x4dd072
ssh-keygen: generating new host keys: ED25519
Starting sshd: OK
running system update...
UPDATE FLAG : TRUE
=====eMMC UPDATE=====
Warning: disk /dev/mmcblk0 will be formatted !
... ..
Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): [ 13.017622] cfg80211: Exceeded CRDA call max
attempts. Not calling CRDA
done
Writing superblocks and filesystem accounting information: done

[ 21.038910] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts:
(null)
Info: Copy /media/mmcblk1p1/MLO to /tmp/p1
Info: Copy /media/mmcblk1p1/u-boot.img to /tmp/p1
Info: Copy /media/mmcblk1p1/logo.bmp to /tmp/p1
Info: Copy /media/mmcblk1p1/uEnv_eMMC.txt to /tmp/p1/uEnv.txt
Info: Copy /media/mmcblk1p1/am335x-som860e.dtb to /tmp/p1
Info: Copy /media/mmcblk1p1/zImage to /tmp/p1
Thu Jun 13 14:56:00 UTC 2019
Info: Copy /media/mmcblk1p1/rootfs.tar.xz to /tmp/p2
/media/mmcblk1p1/rootfs.tar.xz (1/1)
100 %      46.9 MiB / 157.9 MiB = 0.297    2.2 MiB/s      1:13
UPDATE : COMPLETED
```

更新过程进行时：LED 快速闪烁；

更新过程完成时：LED 恢复心跳闪烁；蜂鸣器鸣叫；

其他状态表示异常。

拔出 TF 卡和 JP5 上的跳线帽，重启开发板，即可从 eMMC 启动 Linux 系统。

3) Uboot 参数设置

映像默认为 800x600 显示，如想使用其他显示设备，用户必须根据所使用的显示设备修改 UBOOT 参数，具体方法可参考【2.10 显示模式配置】。

2.10 显示模式配置

系统支持多种显示输出模式，用户可通过配置启动参数的方法选择不同的显示输出模式。下面的内容将介绍如何针对显示模式进行配置。

注意：

📖 暂不支持 9.7 寸电容屏！

1) 编辑 u-boot 源码文件：include/configs/am335x_evm.h:

```
#if defined(CONFIG_BOARD_SOM860E)
# define LCDMODE           "dispmode=800x600\0"
```

请根据所配的 LCD 分辨率填写相应的参数，目前支持 480x272、640x480、800x480、800x600 和 1024x768；

2) 编译 u-boot 生成新映像，更新到 ARM 板中即可。

2.11 测试和演示

本小节将对 SOM860E 上的各个设备在 Debian 系统下进行测试。

2.11.1 LED 测试

主板上的 D35 为系统心跳灯。

以下操作在超级终端中进行：

1) 控制系统心跳灯：

- `root@arm:~# echo none > /sys/class/leds/sys/trigger`
- `root@arm:~# echo 1 > /sys/class/leds/sys/brightness`
- `root@arm:~# echo 0 > /sys/class/leds/sys/brightness`

恢复心跳状态：


- `root@arm:~# echo heartbeat > /sys/class/leds/sys/trigger`

2.11.2 KEYPAD 测试

板子有两个用户键盘 BACK 和 MENU，用户可执行以下命令进行测试：

```
root@arm:~# evtest /dev/input/event1
Input driver verevdev: (EVIOCGBIT): Suspicious buffer size 511
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (Sync)
  Event type 1 (Key)
    Event code 1 (Esc)
    Event code 59 (F1)
Testing ... (interrupt to exit)
Event: time 1233046135.256046, type 1 (Key), code 1 (Esc), value 1
Event: time 1233046135.256053, ----- Report Sync -----
Event: time 1233046135.426967, type 1 (Key), code 1 (Esc), value 0
Event: time 1233046135.426970, ----- Report Sync -----
Event: time 1233046136.373255, type 1 (Key), code 59 (F1), value 1
Event: time 1233046136.373260, ----- Report Sync -----
Event: time 1233046136.548841, type 1 (Key), code 59 (F1), value 0
Event: time 1233046136.548844, ----- Report Sync -----
```

注意：

 按 CONTROL+C 退出测试，后续测试同理。

2.11.3 触摸屏测试

1) 输入以下指令执行触摸屏校准程序：

- `root@arm:~# ts_calibrate`

按照屏幕上提示，点击“+”图标 5 次完成校准。

2) 校准完成后，输入以下指令进行触摸屏测试：

- `root@arm:~# ts_test`

按照屏幕提示，可选择画点、画线测试。

2.11.4 背光测试

背光的亮度设置范围为（0 ~ 8），8 表示亮度最高。0 表示关闭背光亮度的，进入系统后在终端下输入如下命令进行背光测试。

1) 执行以下指令查看背光的亮度默认值：

- `root@arm:~# cat /sys/class/backlight/backlight/brightness`

2) 执行以下指令设置背光亮度为 0 并察看当前背光亮度：

- `root@arm:~# echo 0 > /sys/class/backlight/backlight/brightness`

此时背光被关闭，屏幕显示黑屏。

3) 执行以下指令设置背光亮度为 8 并察看当前背光亮度：

- `root@arm:~# echo 8 > /sys/class/backlight/backlight/brightness`

此时屏幕变亮。

2.11.5 ADC 测试

SOM860E 带有 4 路 ADC 通道，用于测量外部模拟信号电压。

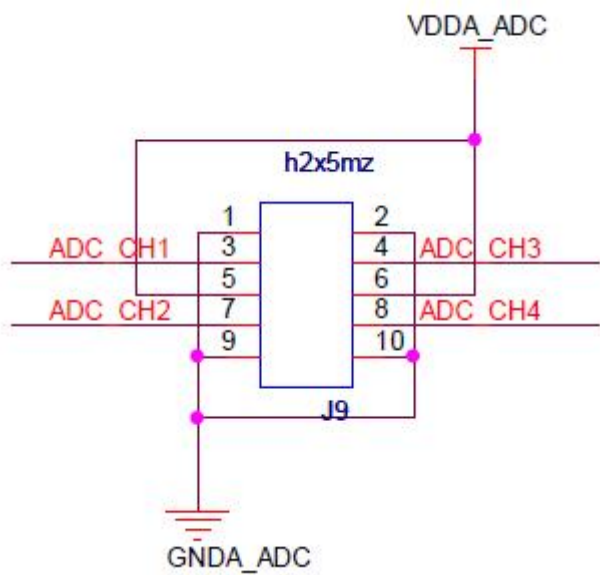


图 2-10 ADC通道原理图

表 2-4 ADC通道

ADC Pin	System node
ADC_CH1	in_voltage4_raw
ADC_CH2	in_voltage5_raw
ADC_CH3	in_voltage6_raw
ADC_CH4	in_voltage7_raw

将待测信号连接到任一 ADC 管脚上，比如 ADC_CH2，执行以下命令获取 ADC 转换值：

```
root@arm:~# cat /sys/bus/iio/devices/iio:device0/in_voltage5_raw
```

计算公式：

$$\text{Voltage Input} = \text{VDDA_ADC} * (\text{value read}) / 4096$$

其中 VDDA_ADC 为 1.8V。

2.11.6 RTC 测试

开发板带硬件时钟，用于保存并恢复系统时间，可参考如下方法进行测试：

- 1) 设置系统时间为 2019 年 6 月 30 日 12 点 10 分

```
root@arm:~# date -s "2019-06-30 12:10"
Sun Jun 30 12:10:00 UTC 2019
```

2) 把系统时钟写入 RTC

- `root@arm:~# hwclock -w`

3) 读取 RTC

```
root@arm:~# hwclock
Sun Jun 30 12:10:21 2019  0.000000 seconds
```

硬件时钟 RTC 被设置成 2019 年 6 月 30 日，系统时钟信息被保存到硬件时钟里。

4) 重启系统，输入以下命令恢复系统时钟

```
root@arm:~# hwclock -s
root@arm:~# date
Sun Jun 30 12:11:00 UTC 2019
```

系统时间被同步为硬件时间。

注意：

开发板自身未带电池（型号 CR1220），用户需自行购买。

2.11.7 TF 卡测试

1) 接入 TF 卡后，系统会自动将 TF 卡的文件系统挂载到/media 目录下：

```
root@arm:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
.....
/dev/mmcblk0p1  121M  4.6M  116M   4% /media/mmcblk0p1
/dev/mmcblk1p1  3.7G  34M   3.6G   1% /media/mmcblk1p1
```

2) 输入下述指令后，即可看到 TF 卡里面的内容：

```
root@arm:~# ls /media/mmcblk1p1
MLO                      ramdisk.img              uEnv.txt
logo.bmp                 u-boot.img
```

3) 手动卸载 TF 卡。

- `root@arm:~# umount /media/mmcblk1p1`

4) 手动挂载 TF 卡。

```
root@arm:~# mount -t vfat /dev/mmcblk1p1 /mnt
root@arm:~# df -h
```

Filesystem	Size	Used	Available	Use%	Mounted on
.....					
/dev/mmcblk1p1	3.7G	34M	3.6G	1%	/mnt
root@arm:~# ls /mnt					
MLO	ramdisk.img				uEnv.txt
logo.bmp	u-boot.img				

2.11.8 USB DEVICE 测试

USB DEVICE 测试主要是使用连接线连接开发板的 miniUSB 接口与电脑端的 USB 接口，对于电脑端，开发板被识别成一个网络设备，实现两端 ping 通讯。

- 1) 系统起来后，使用 USB mini B to USB A 转接线连接开发板（CON2 接口）与电脑端，其中 USB mini B 接口连接开发板，USB A 接口连接电脑端。此时电脑需要安装 Linux USB Ethernet 驱动；
- 2) 配置 usb 虚拟网卡的 IP 地址

```
root@arm:~# ifconfig usb0 192.168.1.115
root@arm:~# ifconfig
lo          Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            UP LOOPBACK RUNNING  MTU:16436  Metric:1
            RX packets:26 errors:0 dropped:0 overruns:0 frame:0
            TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:2316 (2.2 KiB)  TX bytes:2316 (2.2 KiB)

usb0        Link encap:Ethernet  HWaddr 5E:C5:F6:D4:2B:91
            inet addr:192.168.1.115  Bcast:192.168.1.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:253 errors:0 dropped:0 overruns:0 frame:0
            TX packets:43 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:35277 (34.4 KiB)  TX bytes:10152 (9.9 KiB)
```

- 3) 配置好开发板，点击我的电脑-网上邻居-查看网络连接，PC 端会增加一个虚拟网卡的网络连接。

- 4) 在虚拟网卡的网络连接图标上单击电脑端鼠标右键，选择 “属性”，在弹出的属性窗口，双击 “Internet 协议（TCP/IP）” 进入 “Internet 协议（TCP/IP）属性” 窗口，配置虚拟网卡的 IP 地址：

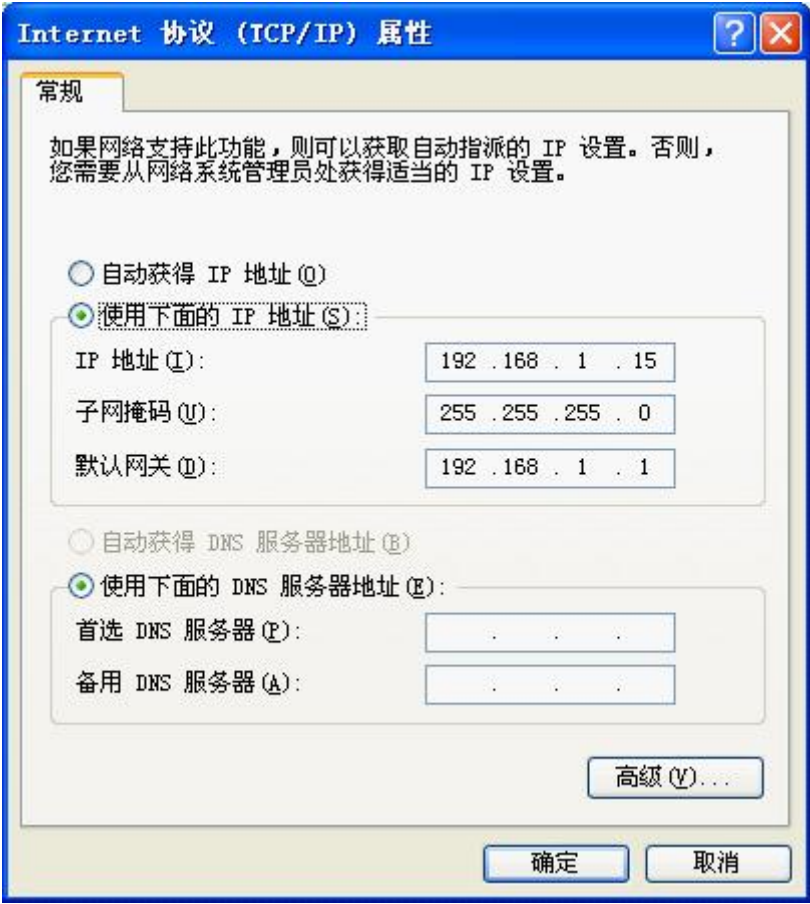


图 2-11 IP配置

- 5) 在超级终端中使用 ping 命令测试开发板是否设置成功：

```
root@arm:~# ping 192.168.1.15
PING 192.168.1.15 (192.168.1.15): 56 data bytes
64 bytes from 192.168.1.15: seq=0 ttl=128 time=0.885 ms
64 bytes from 192.168.1.15: seq=1 ttl=128 time=0.550 ms
```

- 6) 出现上述串口信息，代表测试成功。

注意：
OTG 虚拟网卡的 IP 地址不能与开发板以太网口的 IP 地址在同一个网段内。

2.11.9 USB HOST 测试

- 1) 进入 Linux 系统，将 U 盘连接到板上的 USB-HUB（CON3）接口，系统会自动将 U 盘的文件系统挂载到 /media/usbhd-sda1 目录下：

```
root@arm:~# df -h
Filesystem              Size      Used Available Use% Mounted on
.....
/dev/sda1                3.7G    74M    3.6G     2% /media/usbhd-sda1
root@arm:~# ls /media/usbhd-sda1
MLO      u-boot.img  zImage
```

- 2) 手动卸载 U 盘：
- `root@arm:~# cd`
 - `root@arm:~# umount /media/usbhd-sda1`
- 3) 查看 U 盘是否已经卸载，当输入 `df` 命令后，发现没有 /media/usbhd-sda1 目录。

```
root@arm:~# df
Filesystem      Size  Used Avail Use% Mounted on
ubi0:rootfs    206M   55M  146M   28% /
devtmpfs        237M    0   237M    0% /dev
tmpfs           249M    0   249M    0% /dev/shm
tmpfs           249M   6.6M   243M    3% /run
tmpfs           5.0M    0    5.0M    0% /run/lock
tmpfs           249M    0   249M    0% /sys/fs/cgroup
```

- 4) 手动挂载 U 盘。

```
root@arm:~# mount -t vfat /dev/sda1 /mnt/
root@arm:~# df -h
Filesystem              Size      Used Available Use% Mounted on
.....
/dev/sda1                3.7G    74M    3.6G     2% /mnt
```

2.11.10 AUDIO 测试

板上带音频输入、输出接口，支持录放音。文件系统内带 `alsa-utils` 音频播放、录制测试工具，用户可使用如下命令进行测试：

- 1) 录音测试：
- 插上麦克风，在超级终端输入以下命令即可进行录音


```
root@arm:~# arecord -t wav -c 1 -r 44100 -f S16_LE -v k
Recording WAVE 'k' : Signed 16 bit Little Endian, Rate 44100 Hz, Mono
Plug PCM: Route conversion PCM (sformat=S16_LE)
Transformation table:
    0 <- 0*0.5 + 1*0.5
Its setup is:
stream      : CAPTURE
access      : RW_INTERLEAVED
format      : S16_LE
subformat   : STD
channels    : 1
rate        : 44100
exact rate  : 44100 (44100/1)
msbits      : 16
buffer_size : 22052
period_size : 5513
period_time : 125011
tstamp_mode : NONE
period_step : 1
avail_min   : 5513
period_event : 0
start_threshold : 1
stop_threshold : 22052
silence_threshold: 0
silence_size : 0
boundary    : 1445199872
Slave: Hardware PCM card 0 'AM335x-SGTL5000' device 0 subdevice 0.....
```

2) 放音测试:

设置音量:

```
root@arm:~# amixer set Headphone 100
```

音量范围: 0 ~ 127

插上耳机, 执行以下操作, 即可听刚才的录音内容。

```
root@arm:~# aplay -t wav -c 2 -r 44100 -f S16_LE -v k
Playing WAVE 'k' : Signed 16 bit Little Endian, Rate 44100 Hz, Mono
Plug PCM: Route conversion PCM (sformat=S16_LE)
Transformation table:
    0 <- 0
    1 <- 0
Its setup is:
```

```
stream      : PLAYBACK
access      : RW_INTERLEAVED
format      : S16_LE
subformat   : STD
channels    : 1
rate        : 44100
exact rate  : 44100 (44100/1)
msbits      : 16
buffer_size : 22052
period_size : 5513
period_time : 125011
tstamp_mode : NONE
period_step : 1
avail_min   : 5513
period_event : 0
start_threshold : 22052
stop_threshold : 22052
silence_threshold: 0
silence_size : 0
boundary    : 1445199872
Slave: Hardware PCM card 0 'AM335x-SGTL5000' device 0 subdevice 0.....
```

2.11.11 VIDEO 测试

出厂映像中默认支持 mplayer 视频播放器，可支持 avi,mp4 等多种格式的视频影像文件。

- `root@arm:~# mplayer -vo fbdev2 movie.avi`

注意：

影片画面尺寸不可超过 LCD 尺寸，否则播放可能失败！

2.11.12 网络测试

SOM860E 板载两个以太网口，它们分别是 NET1（J1）和 NET2（J2），它们对应的设备节点分别为 eth0 和 eth1。用网线连接以太网口和路由器，使用以下命令进行测试：

注意：

SOM860E 两个网口的 IP 地址必须设置为不同网段，否则不能正常使用。

```
root@arm:~# ifconfig eth0 192.192.192.200
root@arm:~# ifconfig
eth0      Link encap:Ethernet  HWaddr D4:94:A1:8D:EB:25
          inet addr:192.192.192.200 Bcast:192.192.192.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:137 errors:0 dropped:4 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13792 (13.4 KiB)  TX bytes:0 (0.0 B)
          Interrupt:40

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@arm:~# ping 192.192.192.170
PING 192.192.192.170 (192.192.192.170): 56 data bytes
64 bytes from 192.192.192.170: seq=0 ttl=128 time=4.486 ms
64 bytes from 192.192.192.170: seq=1 ttl=128 time=0.336 ms
root@arm:~# ifconfig eth1 192.168.168.116
root@arm:~# ifconfig
eth1      Link encap:Ethernet  HWaddr 00:17:EA:96:34:D5
          net addr:192.168.168.116  Bcast:192.168.168.255 Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

```
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
root@arm:~# ping 192.168.168.121
PING 192.168.168.121 (192.168.168.121): 56 data bytes
64 bytes from 192.168.168.121: seq=0 ttl=64 time=7.969 ms
64 bytes from 192.168.168.121: seq=1 ttl=64 time=0.319 ms
```

出现上述串口信息，代表测试成功。

2.11.13 CAN 测试

SOM860E 可以作为一个 CAN 设备使用。按照下图所示连接原理，并参考原理图找到对应的引脚，用连接线连接 SOM860E 的 CAN 接口和另一个 CAN 设备。

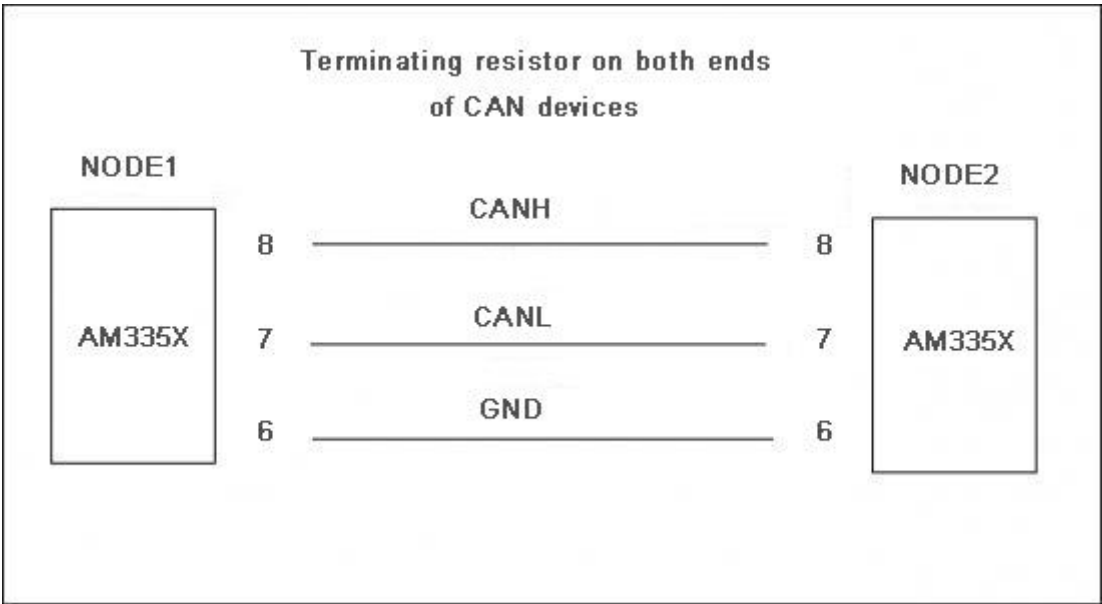


图 2-12 CAN连接

测试方法如下：

- 1) 将 SOM860E 和另一个 CAN 设备的通信波特率都设置为 125Kbps，并使能 CAN 设备；
- `root@arm:~# ifconfig can0 down`
- 设置 CAN 参数前必须确保设备关闭，否则将报错。
- `root@arm:~# /bin/ip link set can0 type can bitrate 125000`
- 设置波特率为 125Kbps。
- `root@arm:~# /bin/ip link set can0 type can restart-ms 100`
- 设置延时恢复，必需步骤。

- `root@arm:~# ifconfig can0 up`

开启 CAN 设备。

- 2) 在 SOM860E 和另一个 CAN 设备两端分别执行发送和接收数据的命令，执行以下命令发送数据包；

- `root@arm:~# cansend can0 "5A1#1122334455667788"`

注意：

该命

要确保另一端处于接收状态。这样接收端才会打印发送的信息。

- 3) 接收数据包；

- `root@arm:~# candump can0`

执行命令后，当收到数据时会在终端下打印接收的数据。

- 4) 关闭设备；

- `root@arm:~# ifconfig can0 down`

用户可以根据以上命令进行相互收发测试，还可以设置不同的波特率进行通信，在设置不同波特率之前必须先关闭设备，可设置的波特率有：

25Kbps (250000)

50Kbps (50000)

125Kbps (125000)

500Kbps (500000)

650Kbps (650000)

1Mbps (1000000)

以上的波特率均能正常通信。

注意：

两块开发板通过 CAN 通信，必须设置成相同的波特率。

源码可参考开源软件 can-utils。

2.11.14 RS485 测试

SOM860E 可以作为一个 RS485 使用。按照下图所示连接原理，并参考原理图找到对应的引脚，用连接线连接 SOM860E 的 RS485 接口和另一个 RS485 设备。

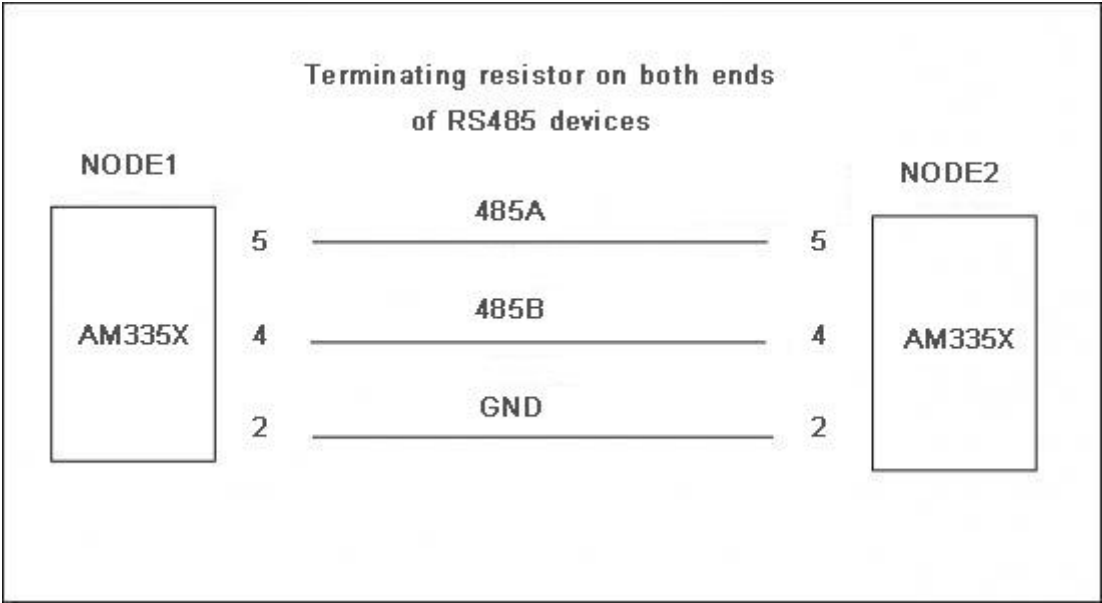


图 2-13 RS485 连接

485 通信只支持半双工通信，即通信一端同一时间只能发送或者只能接收信息。拷贝 CD\Source\App\uart 下的 uart 可执行程序到 TF 卡中，将 TF 卡插入 SOM860E 的 TF 卡座子，在终端下运行如下命令：

```
root@arm:~# /test/app/uart -d /dev/ttyS1 -b 115200
/dev/ttyS1 SEND: 1234567890
/dev/ttyS1 RECV 10 total
/dev/ttyS1 RECV: 1234567890
/dev/ttyS1 SEND: 1234567890
/dev/ttyS1 RECV 10 total
/dev/ttyS1 RECV: 1234567890
/dev/ttyS1 SEND: 1234567890
/dev/ttyS1 RECV 10 total
/dev/ttyS1 SEND: 1234567890
/dev/ttyS1 RECV 10 total
```

2.11.15 串口测试

短接板上 J5 接口的 RX3V3 和 TX3V3 管脚，拷贝 CD\Source\App\uart 下的 uart 可执行文件到 TF 卡中，将 TF 卡插入 SOM860E 的 TF 卡座子，在 linux 终端输入如下命令：

- `root@arm:~# /embest/uart -d /dev/ttyS2 -b 115200`

打印如下信息表示测试成功。

```
/dev/ttyS2 SEND: 1234567890
/dev/ttyS2 RECV 10 total
/dev/ttyS2 RECV: 1234567890
/dev/ttyS2 SEND: 1234567890
/dev/ttyS2 RECV 10 total
/dev/ttyS2 RECV: 1234567890
/dev/ttyS2 SEND: 1234567890
/dev/ttyS2 RECV 10 total
/dev/ttyS2 RECV: 1234567890
/dev/ttyS2 SEND: 1234567890
/dev/ttyS2 RECV 10 total
/dev/ttyS2 RECV: 1234567890
/dev/ttyS2 SEND: 1234567890
/dev/ttyS2 RECV 10 total
/dev/ttyS2 RECV: 1234567890
```

SOM860E 上串口 3、串口 4 和串口 5（J6 和 J7）的测试方法同上。

2.11.16 蜂鸣器测试

1) 打开蜂鸣器。

- `root@arm:~# echo 1 > /sys/class/leds/buzzer/brightness`

2) 关闭蜂鸣器。

- `root@arm:~# echo 0 > /sys/class/leds/buzzer/brightness`

2.11.17 休眠唤醒测试

- `root@arm:~# echo mem > /sys/power/state`

```
[ 4351.715262] PM: Syncing filesystems ... done.
[ 4351.729920] Freezing user space processes ... (elapsed 0.001 seconds)
done.
```

```
[ 4351.738777] Freezing remaining freezable tasks ... (elapsed 0.001
seconds) done.
[ 4351.747471] Suspending console(s) (use no_console_suspend to debug)
[Debug Uart Input, Touch Screen or Button S2]
[ 4352.047646] PM: suspend of devices complete after 293.057 msecs
[ 4352.049277] PM: late suspend of devices complete after 1.602 msecs
[ 4352.051137] PM: noirq suspend of devices complete after 1.833 msecs
[ 4352.051147] PM: Successfully put all powerdomains to target state
[ 4352.051147] PM: Wakeup source UART
[ 4352.068706] PM: noirq resume of devices complete after 17.431 msecs
[ 4352.070122] PM: early resume of devices complete after 1.071 msecs
[ 4352.070950] net eth0: initializing cpsw version 1.12 (0)
[ 4352.075081] sgtl5000 0-000a: Failed to get mclock: -2
[ 4352.145051] net eth0: phy found : id is : 0x4dd072
[ 4352.147181] net eth1: initializing cpsw version 1.12 (0)
[ 4352.150627] sgtl5000 0-000a: Failed to get mclock: -2
[ 4352.225065] net eth1: phy found : id is : 0x4dd072
[ 4352.344861] PM: resume of devices complete after 274.700 msecs
[ 4352.422824] Restarting tasks ...
[ 4352.426558] usb 2-1: USB disconnect, device number 2
[ 4352.454743] done
```

2.11.18 Unique ID

- root@arm:~# cat /sys/devices/soc0/soc_id

```
A90BC968F798A90BC968F998
```

或

- root@arm:~# cat /proc/cpuinfo

```
processor       : 0
model name     : ARMv7 Processor rev 2 (v7l)
BogoMIPS      : 498.89
Features      : half thumb fastmult vfp edsp thumbee neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant   : 0x3
CPU part      : 0xc08
CPU revision  : 2

Hardware      : Generic AM33XX (Flattened Device Tree)
```


Revision	: 0000
Serial	: A90BC968F798A90BC968F998

2.11.19 GPIO 测试

测试 J7 上的 GPIO0_19 和 GPIO2_0。

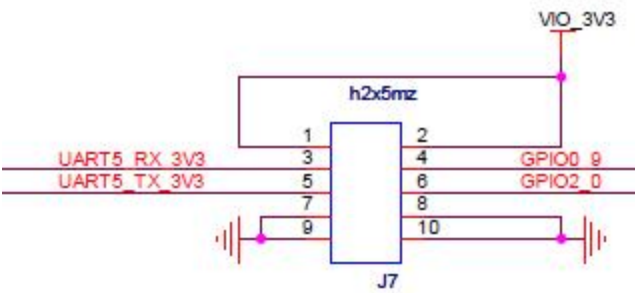


图 2-14 GPIO接口

注意：

原理图中 **GPIO0_9** 标注不正确，实际上连接的是 **GPIO0_19**，请用户特别注意。

1) 初始化 GPIO，生成 gpioX 目录，只需执行一次。配置 GPIO0_19 命令如下：

```
root@arm:~# echo $((32 * 0 + 19)) > /sys/class/gpio/export
```

生成目录：/sys/class/gpio/gpio19。同理，配置 GPIO2_0 命令如下：

```
root@arm:~# echo $((32 * 2 + 0)) > /sys/class/gpio/export
```

生成目录：/sys/class/gpio/gpio64。不同 GPIO 对应不同 ID 目录。

2) 设置为输出模式：

```
root@arm:~# echo out > /sys/class/gpio/gpio19/direction
```

输出高电平：

```
root@arm:~# echo 1 > /sys/class/gpio/gpio19/value
```

输出低电平：

```
root@arm:~# echo 0 > /sys/class/gpio/gpio19/value
```

3) 设置为输入模式：

```
root@arm:~# echo in > /sys/class/gpio/gpio19/direction
```

读取输入电平：

```
• root@arm:~# cat /sys/class/gpio/gpio19/value
```

注意：

📖 测试 GPIO2_0 时，将以上命令中的 **gpio19** 替换为 **gpio64** 即可。

2.11.20 SPI 测试

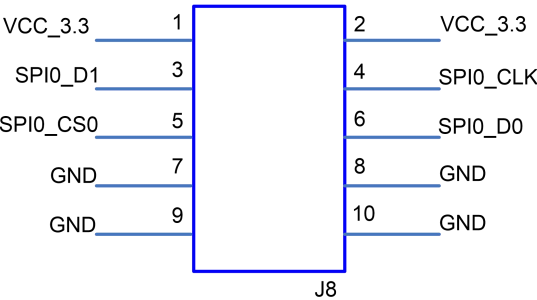


图 2-15 SPI0 接口

短接 J8 的 3 和 6 管脚，执行命令：

```
• root@arm:~# /test/app/spidev_test -D /dev/spidev1.0 -s 1000000 -p "Hello SPI" -v
```

```
spi mode: 0x0
bits per word: 8
max speed: 1000000 Hz (1000 KHz)
TX | 48 65 6C 6C 6F 20 53 50 49 _____ | Hello.SPI
RX | 48 65 6C 6C 6F 20 53 50 49 _____ | Hello.SPI
```

SPI 发送字符串 “Hello SPI”，且接收到了同样的字符串，表示收发成功。

注意：

📖 SPI0_D0 和 SPI0_D1 功能可互换，由内核 dts 决定，默认配置为 **ti,pindir-d0-out-d1-in**；故 SPI0_D0 为 SPI_MOSI，SPI0_D1 为 SPI_MISO。

2.11.21 CAMERA 测试



图 2-16 CAM8200-U

1) 连接摄像头模块

```
[ 507.281230] usb 2-1.2: New USB device found, idVendor=090c, idProduct=f37d
[ 507.288207] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 507.295565] usb 2-1.2: Product: SMI
[ 507.301263] usb 2-1.2: Manufacturer: SMI
[ 507.315001] uvcvideo: Found UVC 1.00 device SMI (090c:f37d)
[ 507.367637] input: SMI as /devices/platform/ocp/47400000.usb/47401c00.usb/musb-hdrc.1.auto/usb2/2-1/2-1.2/2-1.2:1.0/input/input2
```

2) 实时监控

- `root@arm:~# /test/app/luvc_test -c -S -f mjpg /dev/video0`

注意:

- 推荐配合 7 英寸 LCD 测试。
- 推荐摄像头分辨率 640*480。

2.11.22 WIFI 测试



图 2-17 Wi-Pi

Wi-Pi: 核心 RT5370

拥有相同核心型号的 WIFI 模块均可支持，操作方法一致。

1) 连接 WIFI 模块

```
[ 139.165325] usb 2-1.2: New USB device found, idVendor=148f, idProduct=5370
[ 139.172303] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=
3
[ 139.181356] usb 2-1.2: Product: 802.11 n WLAN
[ 139.185776] usb 2-1.2: Manufacturer: Ralink
[ 139.191570] usb 2-1.2: SerialNumber: 1.0
[ 139.297591] usb 2-1.2: reset high-speed USB device number 4 using musb-hdrc
[ 139.428426] ieee80211 phy0: rt2x00_set_rt: Info - RT chipset 5390, rev 0502 dete
cted
[ 139.452619] ieee80211 phy0: rt2x00_set_rf: Info - RF chipset 5370 detected
[ 141.405892] ieee80211 phy0: rt2x00lib_request_firmware: Info - Loading firmware fi
le 'rt2870.bin'
[ 141.424760] ieee80211 phy0: rt2x00lib_request_firmware: Info - Firmware detected
- version: 0.29
```

2) 扫描 WIFI 热点

- **root@arm:~# iwlist wlan0 scan**

```
Cell 35 - Address: F0:3E:90:61:E1:78
Channel:7
Frequency:2.442 GHz (Channel 7)
Quality=45/70 Signal level=-65 dBm
Encryption key:on
ESSID:"EMBEST_WIFI"
Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s
Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s
36 Mb/s; 48 Mb/s; 54 Mb/s
Mode:Master
Extra:tsf=000000063352af2f
Extra: Last beacon: 720ms ago
IE: Unknown: 000B454D424553545F57494649
IE: Unknown: 010482848B96
IE: Unknown: 030107
IE: Unknown: 0706434E20010D14
IE: Unknown: 2A0100
IE: Unknown: 32080C1218243048606C
IE: Unknown: DD1E00904C33AD0103FFFF00000000000000000000
```

[illegible]

3) 加入热点网络 ESSID 和密钥

- `root@arm:~# wpa_passphrase MY_WIFI MYPASSWD >>`
`/etc/wpa_supplicant/wpa_supplicant.conf`

4) 重启 WIFI 服务

- `root@arm:~# systemctl restart wpa_supplicant.service`

等待一段时间，即可连接成功。

5) 查看 WIFI 获取的 IP

- `root@arm:~# ifconfig wlan0`

```
wlan0      Link encap:Ethernet  HWaddr 40:A5:EF:05:5A:B9
           inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:20 errors:0 dropped:0 overruns:0 frame:0
           TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:3503 (3.4 KiB)  TX bytes:1736 (1.6 KiB)
```

2.11.23 BLUETOOTH 测试



图 2-18 CSR4.0 Bluetooth

1) 连接蓝牙模块

```
[ 2456.167582] usb 2-1.2: new full-speed USB device number 5 using musb-hdrc
[ 2456.306046] usb 2-1.2: New USB device found, idVendor=0a12, idProduct=0001
[ 2456.313035] usb 2-1.2: New USB device strings: Mfr=0, Product=2, SerialNumber=0
[ 2456.322207] usb 2-1.2: Product: CSR8510 A10
[ 2456.529269] Bluetooth: hci0 hardware error 0x35
```

2) 扫描蓝牙终端

```
root@arm:~# hciconfig hci0 up; hcitool scan
```

28:B2:BD:79:93:F1	GBRLT-XXXXX
0C:30:21:D3:5A:9B	BLUE-XXXXX

3) Ping 测试

```
root@arm:~# l2ping 0C:30:21:D3:5A:9B
```

```
Ping: 0C:30:21:D3:5A:9B from 00:1A:7D:DA:71:13 (data size 44) ...
44 bytes from 0C:30:21:D3:5A:9B id 0 time 9.98ms
44 bytes from 0C:30:21:D3:5A:9B id 1 time 93.48ms
44 bytes from 0C:30:21:D3:5A:9B id 2 time 225.96ms
44 bytes from 0C:30:21:D3:5A:9B id 4 time 90.96ms
4 sent, 4 received, 0% loss
```

2.11.24 Debian 配置

【版本】 ARM Debian 8

1) 静态网络配置

表 2-5 网络配置

配置文件	/etc/network/interfaces
------	-------------------------

Static	iface eth0 inet static address 192.168.1.210 netmask 255.255.255.0
--------	--

手动配置静态 IP:

- `root@arm:~# ifconfig eth0 192.168.1.210`

注意:

 需禁止 DHCP 服务: `systemctl stop system-network.service; systemctl disable system-network.service;`

2) 动态网络配置


注释掉`/etc/network/interfaces`中关于静态网络配置的信息:

开启 DHCP 服务:

- `root@arm:~# systemctl restart system-network.service;`
- `root@arm:~# systemctl enable system-network.service;`

eth0 和 eth1 插拔网线可自动获取 IP。

注意:

 DHCP 服务默认监控 eth0, eth1 和 wlan0, 若仅需监控某些网口可编辑文件`/etc/NetworkManager/CP.sh`配置 `devlist` 列表变量。

3) 时区配置

系统默认为 UTC 时间, 下面以设置为 “Shanghai” 时间为例介绍设置方法:

- `root@arm:~# echo "Asia/Shanghai" > /etc/timezone`
- `root@arm:~# ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime`

4) 开机启动

准备用户程序 -> 创建配置文件 -> 使能配置 -> 重启生效;

- `root@arm:~# vi /lib/systemd/system/user.service`

```
[Unit]
Description=User Program
After=network.target

[Service]
ExecStart=/etc/User.sh

[Install]
WantedBy=multi-user.target
```

- `root@arm:~# systemctl enable user.service; sync`

```
Created      symlink      /etc/systemd/system/multi-user.target.wants/user.service →
/lib/systemd/system/user.service.
```

无报错则可重启系统测试是否生效。

5) 禁止 LCD 显示闪烁光标

系统启动后，默认情况下 LCD 中已禁止光标闪烁。若用户需要恢复光标闪烁，可使用下列命令：

- `root@arm:~# echo 1 > /sys/class/graphics/fbcon/cursor_blink`

禁止

- `root@arm:~# echo 0 > /sys/class/graphics/fbcon/cursor_blink`

2.12 上层开发

本节主要介绍应用程序的开发，并通过实例来说明应用程序开发的一般流程。

2.12.1 LED 应用程序开发示例

1) 编写代码

led_demo.c 源码，控制开发板上的 led 灯按累加器的方式闪烁。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/ioctl.h>
```



```
#include <fcntl.h>

#define LED1 "/sys/class/leds/sys_led/brightness"
#define LED2 "/sys/class/leds/user_led/brightness"

int main(int argc, char *argv[])
{
    int f_led1, f_led2;
    unsigned char i = 0;
    unsigned char dat1, dat2;
    if((f_led1 = open(LED1, O_RDWR)) < 0){
        printf("error in open %s",LED1);
        return -1;
    }
    if((f_led2 = open(LED2, O_RDWR)) < 0){
        printf("error in open %s",LED2);
        return -1;
    }
    for(;;){
        i++;
        dat1 = i&0x1 ? '1':'0';
        dat2 = (i&0x2)>>1 ? '1':'0';
        write(f_led1, &dat1, sizeof(dat1));
        write(f_led2, &dat2, sizeof(dat2));
        usleep(300000);
    }
}
```

2) 交叉编译

- **arm-linux-gcc led_demo.c -o led_demo**

3) 下载运行

通过 TF 卡或 U 盘或网络下载到开发板系统，进入 led_demo 文件所在的目录，输入下面命令回车 led_demo 即在后台运行。

- **./led_demo &**

2.12.2 CAN 应用程序开发示例

1) 定义发送的数据；

CAN 帧数据使用语法: <can_id>#{R|data}; CAN_ID 可以为 3 个 (标准帧格式) 或 8 个 (扩展帧格式) 十六进制字符; Data 数据为 0 到 8 个十六进制值 (可选用分隔符 “.” 分隔)。

常见用法如下:

```
char *cmd_str = "123#1122334455667788";

char *cmd_str = "123#11.22.33.44.55.66.77.88";

char *cmd_str = "12345678#112233";
```

2) 建立套接字:

在使用 CAN 网络之前你首先需要打开一个套接字。CAN 的套接字使用到了一个新的协议族, 所以在调用 `socket()` 这个系统函数的时候需要将 `PF_CAN` 作为第一个参数。当前有两个 CAN 的协议可以选择, 一个是原始套接字协议 (`raw socket protocol`), 另一个是广播管理协议 `BCM` (`broadcast manager`), 如:

```
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
```

在成功创建一个套接字之后, 你通常需要使用 `bind()` 函数将套接字绑定在某个 CAN 接口上。在绑定(`CAN_RAW`)或连接(`CAN_BCM`)套接字之后, 你可以在套接字上使用 `read()/write()`。

基本的 CAN 帧结构体和套接字地址结构体定义在 `include/linux/can.h` 文件中,

如:

```
/**
 * struct can_frame - basic CAN frame structure
 *
 * @can_id: the CAN ID of the frame and CAN_*_FLAG flags, see above.
 * @can_dlc: the data length field of the CAN frame
 * @data: the CAN frame payload.
 */
struct can_frame {
    canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
    __u8 can_dlc; /* data length code: 0 .. 8 */
    __u8 data[8] __attribute__((aligned(8)));
};
```

结构体的有效数据在 `data[]` 数组中, 它的字节对齐是 64bit 的, 所以用户可以比较方便的在 `data[]` 中传输自己定义的结构体和共用体。CAN 总线中没有默认的字序。

在 CAN_RAW 套接字上调用 `read()`，返回给用户空间的数据是一个 `struct can_frame` 结构体。

`sockaddr_can` 结构体有接口的索引，这个索引绑定了特定接口：

```
/**
 * struct sockaddr_can - the sockaddr structure for CAN sockets
 * @can_family: address family number AF_CAN.
 * @can_ifindex: CAN network interface index.
 * @can_addr: protocol specific address information
 */
struct sockaddr_can {
    sa_family_t can_family;
    int can_ifindex;
    union {
        /* transport protocol class address information (e.g. ISOTP) */
        struct { canid_t rx_id, tx_id; } tp;
        /* reserved for future CAN protocols address information */
    } can_addr;
};
```

3) 指定接口索引并绑定

为了将套接字和所有的 CAN 接口绑定，指定接口索引需要调用 `ioctl()`，接口索引必须是 0，这样套接字便可以从所有使能的 CAN 接口接收 CAN 帧。

```
int s;
struct sockaddr_can addr;
struct ifreq ifr;

s = socket(PF_CAN, SOCK_RAW, CAN_RAW);

strcpy(ifr.ifr_name, "can0");
ioctl(s, SIOCGIFINDEX, &ifr);

addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
bind(s, (struct sockaddr *)&addr, sizeof(addr));
```

4) 从套接字上读取 CAN 帧

```
struct can_frame frame;

nbytes = read(s, &frame, sizeof(struct can_frame));

if (nbytes < 0) {
    perror("can raw socket read");
    return 1;
}

/* paranoid check ... */
if (nbytes < sizeof(struct can_frame)) {
    fprintf(stderr, "read: incomplete CAN frame\n");
    return 1;
}
```

5) 从套接字上写 CAN 帧

```
nbytes = write(s, &frame, sizeof(struct can_frame));
```

6) 打开 can 接口

```
int can_do_start(const char *name)
{
    return set_link(name, IF_UP, NULL);
}
```

7) 停止 can 接口:

```
int can_do_stop(const char *name)
{
    return set_link(name, IF_DOWN, NULL);
}
```

8) CAN 示例代码如下:

can_test 目录下主要包含 can_test.c、lib.c、libsocketcan.c 源文件。

lib.c 主要定义了字符转换函数.例如:

```
int parse_canframe(char *cs, struct can_frame *cf);

/*Transfers a valid ASCII string deccribing a CAN frame into struct
can_frame*/
```

libsocketcan.c 主要定义了 can 接口函数。例如：

```
int can_do_start(const char *name) /*start the can interface*/  
int can_do_stop(const char *name) /*stop the can interface*/
```

can_test.c 部分源码如下：

```
#define MAX_CANFRAME      "12345678#01.23.45.67.89.AB.CD.EF"  
#define MAX_LONG_CANFRAME "12345678 [8] 10101010 10101010 10101010  
10101010 10101010 10101010 10101010 10101010 '.....'"  
static int s = -1;  
char buf[sizeof(MAX_LONG_CANFRAME)+1]="";  
char *cmd_str = "111#1122334455667788";//定义要发送的内容  
  
int main(void)  
{  
    struct sockaddr_can addr;  
    static struct ifreq ifr;  
    const char* name = argv[1];  
  
    if ((argc < 2) || !strcmp(argv[1], "--help"))  
        help();  
  
    if (argc < 3)  
        cmd_show_interface(name);  
  
    cmd_stop(argc, argv, name); // can stop  
  
    while (argc-- > 0) {  
        if (!strcmp(argv[0], "bitrate"))  
            cmd_bitrate(argc, argv, name);  
        if (!strcmp(argv[0], "ctrlmode"))  
            cmd_ctrlmode(argc, argv, name);  
        if (!strcmp(argv[0], "start"))  
            cmd_start(argc, argv, name);  
        if (!strcmp(argv[0], "stop"))  
            cmd_stop(argc, argv, name);  
        argv++;  
    }  
    cmd_start(argc, argv, name); // can start  
    if (s != -1) {  
        return 0;  
    }  
}
```

```

s = socket(PF_CAN, SOCK_RAW, CAN_RAW);

if(s < 0) {
    perror("socket");
    return 1;
}

memset(&ifr.ifr_name, 0, sizeof(ifr.ifr_name));
strcpy(ifr.ifr_name, "can0");
if(ioctl(s, SIOCGIFINDEX, &ifr) < 0) {
    perror("SIOCGIFINDEX");
    exit(1);
}

addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;

if(bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("bind");
    return 1;
}

funct_select();
}

发送信息:
int can_send(char *buf)
{
    int nbytes;
    struct can_frame frame;

    if(parse_canframe(buf, &frame)) {
        fprintf(stderr, "\nWrong CAN-frame format!\n\n");
        fprintf(stderr, "Try: <can_id>#{R|data}\n");
        fprintf(stderr, "can_id can have 3 (SFF) or 8 (EFF) hex chars\n");
        fprintf(stderr, "data has 0 to 8 hex-values that can (optionally)");
        fprintf(stderr, " be seperated by '.'\n\n");
        fprintf(stderr, "e.g. 5A1#11.2233.44556677.88 / 123#DEADBEEF / ");
        fprintf(stderr, "5AA# /\n    1F334455#1122334455667788 / 123#R ");
        fprintf(stderr, "for remote transmission request.\n\n");
        return 1;
    }

    if((nbytes = write(s, &frame, sizeof(frame)))
        != sizeof(frame)) {

```

```
        perror("write");
        return 1;
    }

    return 0;
}

接收信息:
int can_rcv(char *buf)
{
    int ret, nbytes;
    fd_set rdfs;
    struct can_frame frame;

    FD_ZERO(&rdfs);
    FD_SET(s, &rdfs);

    if((ret = select(s+1, &rdfs, NULL, NULL, NULL)) < 0) {
        perror("select");
        exit(1);
    }
    if(FD_ISSET(s, &rdfs)) {
        nbytes = read(s, &frame, sizeof(struct can_frame));
        if(nbytes < 0) {
            perror("read");
            return 1;
        }
        if(nbytes < sizeof(struct can_frame)) {
            fprintf(stderr, "read: incomplete CAN frame\n");
            return 1;
        }
        sprint_long_canframe(buf, &frame, 0);
        printf("%s\n", buf);
    }
}
```

9) 交叉编译

执行以下命令将 **Source\App\can_test.tar.bz2** 解压到 work 目录并编译。

- **cd \$HOME/work**
- **tar xvf /media/cdrom/Source/App/can_test.tar.bz2**
- **cd can_test**

- **make**




10) 下载运行

CAN 测试需要两个 CAN 接口设备，连接方法请参考“CAN 测试”，通过 TF 卡或 U 盘或网络下载到开发板系统，进入 **can_test** 文件所在的目录，在终端下运行如下命令：

```
root@arm:~# ./can_test can0 bitrate 125000 ctrlmode triple-sampling on
can0 state: STOPPED
can0 bitrate: 125000, sample-point: 0.875
can0 ctrlmode: loopback[OFF], listen-only[OFF], tripple-sampling[ON],one-shot[OFF],
bd_can d_can: can0: setting CAN BT = 0x518
err-reporting[OFF]
can0 state: ERROR-ACTIVE

Select 1 : Send a message
Select 2 : Receive messages
>
```

注意：

-  "<TRIPLE-SAMPLING>" 表示选中的 CAN 控制器的模式：LOOPBACK, LISTEN-ONLY, or TRIPLE-SAMPLING。
-  "state ERROR-ACTIVE" 表示 CAN 控制器的当前状态："ERROR-ACTIVE", "ERROR-WARNING", "ERROR-PASSIVE", "BUS-OFF" or "STOPPED"
-  关于 socket can 的详细介绍请参考如下文档：[linux-am335x/Documentation/networking/can.txt](#)。

发送信息：按“1”再回车。

```
Select 1 : Send a message
Select 2 : Receive messages
> 1
Information is sent.....
Select 3 : Stop Send
>
```

接收信息：按“2”再回车。

```
Select 1 : Send a message
Select 2 : Receive messages
> 2
111 [8] 11 22 33 44 55 66 77 88
```



```
111 [8] 11 22 33 44 55 66 77 88
111 [8] 11 22 33 44 55 66 77 88
```

2.12.3 串行接口应用程序开发

以下表格列出了串行接口操作所需要的头文件：

表 2-6 串口相关头文件

头文件	注释
#include <stdio.h>	标准输入输出定义
#include <stdlib.h>	标准函数库定义
#include <unistd.h>	UNIX 标准函数定义
#include <sys/types.h>	
#include <sys/stat.h>	
#include <fcntl.h>	文件控制定义
#include <termios.h>	PPSIX 终端控制定义

1) 打开串行接口；

Linux 下的串行接口文件保存在/dev 目录下；通过使用标准的“打开文件”函数可以打开串行接口；

例如：

打开串行接口

```
int fd;
fd = open( "/dev/ttyS0", O_RDWR); /*以读写方式打开串口*/
if (-1 == fd){
    perror(" 提示错误！");/* 不能打开串口一*/
}
```

2) 设置串行接口；

串口设置包括波特率、效验位、停止位和结构体 struct termios 各个成员的值；

例如：

设置波特率的代码

```
struct termios opt;
tcgetattr(fd, &opt);
cfsetispeed(&opt,B115200);    /*设置为 115200bps*/
cfsetospeed(&opt,B115200);
```

```
tcsetattr(fd,TCANOW,&opt);
```

表 2-7 校验位和停止位

无校验	奇校验 (Odd)
8 位	7 位
Option.c_cflag &= ~PARENB; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= ~CSIZE; Option.c_cflag = ~CS8;	Option.c_cflag = ~PARENB; Option.c_cflag &= ~PARODD; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= ~CSIZE; Option.c_cflag = ~CS7;
偶校验	Space 校验
7 位	7 位
Option.c_cflag &= ~PARENB; Option.c_cflag = ~PARODD; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= ~CSIZE; Option.c_cflag = ~CS7;	Option.c_cflag &= ~PARENB; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= &~CSIZE; Option.c_cflag = CS8;

结构体成员值

```
struct termio
{
    unsigned short  c_iflag;      /* 输入模式标志 */
    unsigned short  c_oflag;      /* 输出模式标志 */
    unsigned short  c_cflag;      /* 控制模式标志*/
    unsigned short  c_lflag;      /* local mode flags */
    unsigned char   c_line;       /* line discipline */
    unsigned char   c_cc[NCC];    /* control characters */
};
```

3) 设置校验的函数:


校验函数设置

```
/**
 * @brief 设置串口数据位，停止位和效验位
 * @param fd 类型 int 打开的串口文件句柄
 * @param databits 类型 int 数据位 取值为 7 或者 8
 * @param stopbits 类型 int 停止位 取值为 1 或者 2
 * @param parity 类型 int 效验类型 取值为 N,E,O,,S
 */
int set_Parity(int fd,int databits,int stopbits,int parity)
{
```

```
struct termios options;
if (tcgetattr( fd,&options) != 0) {
    perror("SetupSerial 1");
    return(FALSE);
}
options.c_cflag &= ~CSIZE;
switch (databits) /*设置数据位数*/
{
case 7:
    options.c_cflag |= CS7;
    break;
case 8:
    options.c_cflag |= CS8;
    break;
default:
    fprintf(stderr,"Unsupported data size\n"); return (FALSE);
}
switch (parity)
{
case 'n':
case 'N':
    options.c_cflag &= ~PARENB; /* Clear parity enable */
    options.c_iflag &= ~INPCK; /* Enable parity checking */
    break;
case 'o':
case 'O':
    options.c_cflag |= (PARODD | PARENB); /* 设置为奇效验*/
    options.c_iflag |= INPCK; /* Disable parity
checking */
    break;
case 'e':
case 'E':
    options.c_cflag |= PARENB; /* Enable parity */
    options.c_cflag &= ~PARODD; /* 转换为偶效验*/
    options.c_iflag |= INPCK; /* Disable parity checking */
    break;
case 'S':
case 's': /*as no parity*/
    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;break;
default:
    fprintf(stderr,"Unsupported parity\n");
```

```
        return (FALSE);
    }
    /* 设置停止位*/
    switch (stopbits)
    {
        case 1:
            options.c_cflag &= ~CSTOPB;
            break;
        case 2:
            options.c_cflag |= CSTOPB;
            break;
        default:
            fprintf(stderr,"Unsupported stop bits\n");
            return (FALSE);
    }
    /* Set input parity option */
    if (parity != 'n')
        options.c_iflag |= INPCK;
    tcflush(fd,TCIFLUSH);
    options.c_cc[VTIME] = 150; /* 设置超时 15 seconds*/
    options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
    if (tcsetattr(fd,TCSANOW,&options) != 0)
    {
        perror("SetupSerial 3");
        return (FALSE);
    }
    return (TRUE);
}
```

注意:

 如果不是开发终端之类的，只是串口传输数据，而不需要串口来处理，那么可以使用原始模式（Raw Mode）方式来通讯，如以下代码：

```
options.c_iflag  &= ~(ICANON | ECHO | ECHOE | ISIG); /*Input*/
options.c_oflag  &= ~OPOST; /*Output*/
```

4) 读写串行接口;

设置完成后，就可以将串行接口作为文件来进行读写；读取串行接口时可以使用 `read` 函数；如果串行接口设置为原始模式（Raw Mode），则该函数返回的字符数

即为串行接口接收到的字符数；也可以使用 `fcntl` 或 `select` 函数来实现异步读取；

下列表格包含了读写操作的代码范例；

串行接口写操作

```
char  buffer[1024];int    Length;int    nByte;  nByte = write(fd, buffer ,Length)
```

串行接口读操作

```
char  buff[1024];int    Len;int    readByte = read(fd,buff,Len);
```

5) 关闭串行接口与关闭文件操作相同；

关闭串行接口

```
close(fd);
```

6) 以下表格包含了串行接口测试程序主函数；

程序范例

```
int main(int argc, char *argv[])
{
    int  fd, next_option, havearg = 0;
    char *device;
    int i=0,j=0;
    int nread;          /* Read the counts of data */
    char buff[512];      /* Recvice data buffer */
    pid_t pid;
    char *xmit = "1234567890"; /* Default send data */
    int speed ;
    const char *const short_options = "hd:s:b:";

    const struct option long_options[] = {
        { "help",    0, NULL, 'h'},
        { "device",  1, NULL, 'd'},
        { "string",  1, NULL, 's'},
        { "baudrate", 1, NULL, 'b'},
        { NULL,      0, NULL, 0 }
    };

    program_name = argv[0];
    do {
        next_option = getopt_long (argc, argv, short_options,
```

```
long_options, NULL);
    switch (next_option) {
        case 'h':
            print_usage (stdout, 0);
        case 'd':
            device = optarg;
            havearg = 1;
            break;
        case 'b':
            speed = atoi(optarg);
            break;
        case 's':
            xmit = optarg;
            havearg = 1;
            break;
        case -1:
            if (havearg) break;
        case '?':
            print_usage (stderr, 1);
        default:
            abort ();
    }
}while(next_option != -1);


sleep(1);
fd = OpenDev(device);
if (fd > 0) {
    set_speed(fd, speed);
} else {
    fprintf(stderr, "Error opening %s: %s\n", device, strerror(errno));
    exit(1);
}
if (set_Parity(fd,8,1,'N')== FALSE) {
    fprintf(stderr, "Set Parity Error\n");
    close(fd);
    exit(1);
}
pid = fork();
if (pid < 0) {
    fprintf(stderr, "Error in fork!\n");
} else if (pid == 0){
    while(1) {
```

```
        printf("%s SEND: %s\n",device, xmit);
        write(fd, xmit, strlen(xmit));    //循环写
        sleep(1);
        i++;
    }
    exit(0);
} else {
    while(1) {
        nread = read(fd, buff, sizeof(buff)); //循环读
        if (nread > 0) {
            buff[nread] = '\0';
            printf("%s RECV %d total\n", device, nread);
            printf("%s RECV: %s\n", device, buff);
        }
    }
}
close(fd);
exit(0);
}
```

7) 在 Ubuntu 系统中执行以下命令来将 **CD\Source\App** 目录下的 **uart.tar.xz** 文件复制到 **work** 目录下，并进行编译：

- **cd \$HOME/work**
- **tar -xvf /media/cdrom/Source/App/uart.tar.xz**
- **cd uart**
- **make**

注意：

 关于串行接口的测试请参考《串口测试》章节的内容。