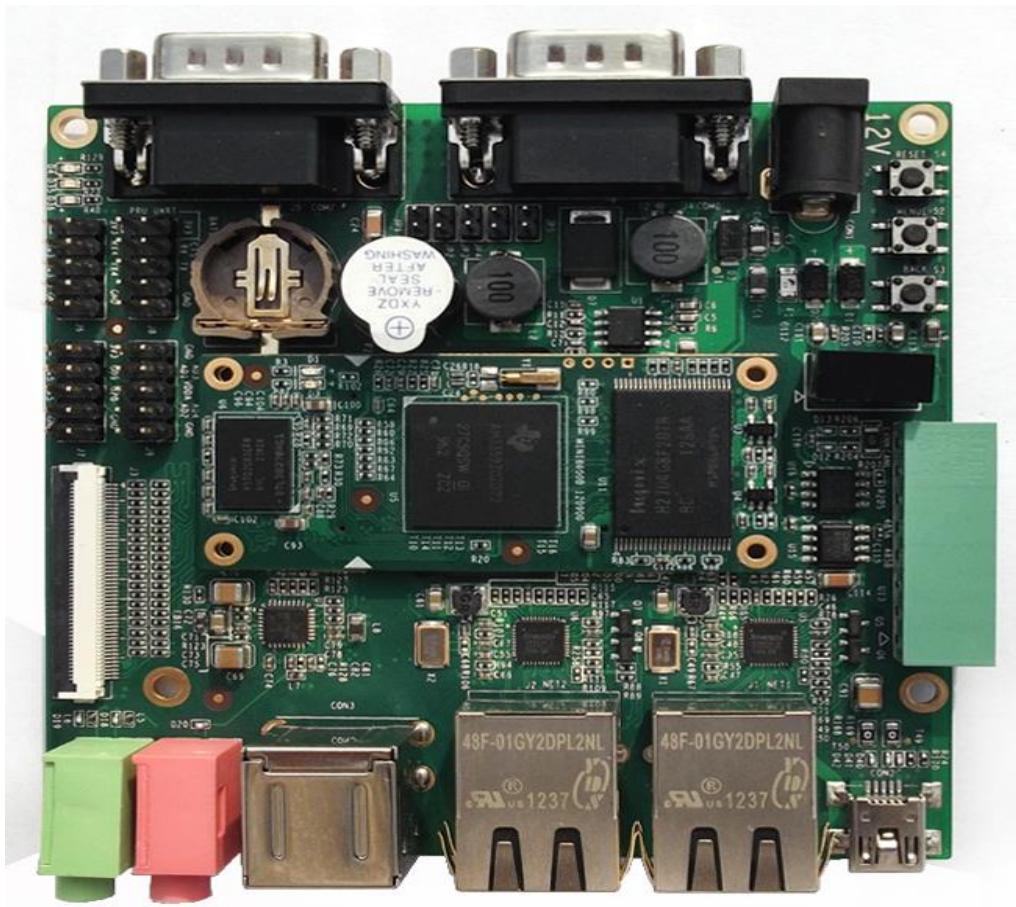


SBC8600B 单板机



用户手册

版本 SBC8600B-UM-V2.0

2017 年 6 月 26 日

版权声明：

- SBC8600B 开发套件及其相关知识产权由深圳市英蓓特科技有限公司所有。
- 本文档由深圳市英蓓特科技有限公司版权所有，并保留一切权利。在未经英蓓特公司书面许可的情况下，不得以任何方式或形式来修改、分发或复制本文档的任何部分。
- Microsoft, MS-DOS, Windows, Windows95, Windows98, Windows2000, Windows xp, 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	2012-12-21	初始版本
1.1	2014-5-1	文档修订
2.0	2017-6-26	软硬件版本升级

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	<i>Mini8600B/Mini8610B</i>	1
1.2.2	扩展板.....	3
1.3	硬件尺寸	6
1.4	配套模块支持情况.....	8
第 2 章	硬件系统	9
2.1	CPU	9
2.1.1	<i>CPU</i> 介绍.....	9
2.1.2	<i>CPU</i> 特性.....	9
2.2	外围芯片介绍	10
2.2.1	<i>NAND Flash H27U4G8F2DTR-BC</i>	10
2.2.2	<i>DDR H5TQ2G83CFR-H9C</i>	11
2.2.3	<i>Ethernet AR8035</i>	11
2.2.4	<i>MAX3232</i>	11
2.3	硬件接口	12
2.3.1	<i>Mini86x0B</i>	12
2.3.2	扩展板.....	18
第 3 章	LINUX 操作系统	26
3.1	软件资源	26
3.1.1	软件资源的位置	26
3.1.2	<i>BSP</i> 软件包.....	27
3.2	嵌入式 LINUX 的组成	28
3.3	开发环境搭建	29

3.3.1	交叉编译工具安装.....	29
3.3.2	添加环境变量.....	30
3.4	准备源代码.....	30
3.4.1	从产品光盘获取源代码	30
3.4.2	用 <i>git</i> 和 <i>repo</i> 工具获取源代码	32
3.5	编译.....	33
3.6	系统定制	34
3.6.1	<i>U-Boot LOGO</i> 制作.....	34
3.6.2	进入内核配置菜单.....	35
3.6.3	内核配置	35
3.6.4	编译内核	36
3.6.5	驱动测试	36
3.7	驱动介绍	37
3.7.1	<i>BSP</i> 的所有驱动源码路径.....	37
3.7.2	<i>NAND</i>	39
3.7.3	<i>SD/MMC</i>	40
3.7.4	<i>LCDC</i>	41
3.7.5	<i>Audio in/out</i>	42
3.8	驱动开发	43
3.8.1	<i>GPIO_keys</i> 驱动.....	43
3.8.2	<i>GPIO_leds</i> 驱动.....	49
3.9	系统更新	52
3.9.1	<i>TF</i> 卡系统映像更新.....	52
3.9.2	<i>NAND Flash</i> 更新/恢复	55
3.10	显示模式配置	59
3.11	测试和演示.....	61
3.11.1	<i>LED</i> 测试.....	61

3.11.2	KEYPAD 测试	61
3.11.3	触摸屏测试.....	62
3.11.4	背光测试	62
3.11.5	ADC 测试.....	63
3.11.6	RTC 测试.....	64
3.11.7	TF 卡测试.....	64
3.11.8	USB DEVICE 测试.....	65
3.11.9	USB HOST 测试.....	67
3.11.10	AUDIO 测试.....	68
3.11.11	网络测试	69
3.11.12	CAN 测试.....	71
3.11.13	RS485 测试.....	73
3.11.14	串口测试	73
3.11.15	蜂鸣器测试.....	74
3.11.16	休眠唤醒测试.....	74
3.11.17	GPIO 测试.....	75
3.11.18	Debian 配置	76
3.11.19	TISDK 系统演示.....	78
3.12	上层开发	81
3.12.1	LED 应用程序开发示例.....	81
3.12.2	CAN 应用程序开发示例	82
3.12.3	串行接口应用程序开发	90
第 4 章	ANDROID 系统.....	97
4.1	开发环境	97
4.1.1	获取 android 源代码.....	97
4.1.2	编译启动代码.....	97
4.2	演示.....	98

4.3	测试.....	100
第 5 章	WINDOWS EMBEDDED COMPACT 7 操作系统.....	101
5.1	软件资源	101
5.1.1	软件资源的位置	101
5.1.2	预编译映像和 <i>BSP</i>	101
5.2	系统开发	103
5.2.1	集成开发环境安装.....	103
5.2.2	提取 <i>BSP</i> 及样例工程文件到集成开发环境.....	103
5.2.3	Sysgen & <i>BSP</i> 编译.....	104
5.2.4	驱动介绍	104
5.3	系统映像更新	105
5.3.1	<i>TF</i> 卡映像更新	105
5.3.2	<i>NAND Flash</i> 映像更新	110
5.4	使用说明	111
5.5	应用开发	111
5.5.1	如何使用 <i>openGL ES demo</i>	111
5.5.2	应用程序接口与示例.....	112
5.5.3	<i>GPIO</i> 应用程序接口与示例.....	112
附录.....	115	
附录一	安装 <i>UBUNTU LINUX</i> 系统	115
附录二	安装 <i>LINUX USB ETHERNET/RNDIS GADGET</i> 驱动	127
附录三	制作 <i>LINUX</i> 启动盘	129
附录四	搭建 <i>TFTP</i> 服务器.....	135
附录五	FAQ 总结	137
技术支持和保修服务	138	

第1章 概述

1.1 产品介绍

SBC8600B 是英蓓特推出的一款基于 AM335x 的嵌入式单板机，它采用核心板 Mini86x0B 加底板的分离式结构进行设计。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 个千兆以太网口，2 路 USB Host 和 1 路 USB OTG，LCD 触摸屏，TF 卡等接口。支持 Linux-4.1，WinCE 7 及 Android 4.0 三种操作系统。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，为开发者提供了完善的软件开发环境，缩短产品开发周期，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

SBC8600B 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 Mini8600B/Mini8610B

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

- 512MByte NAND Flash
- 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
- 2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

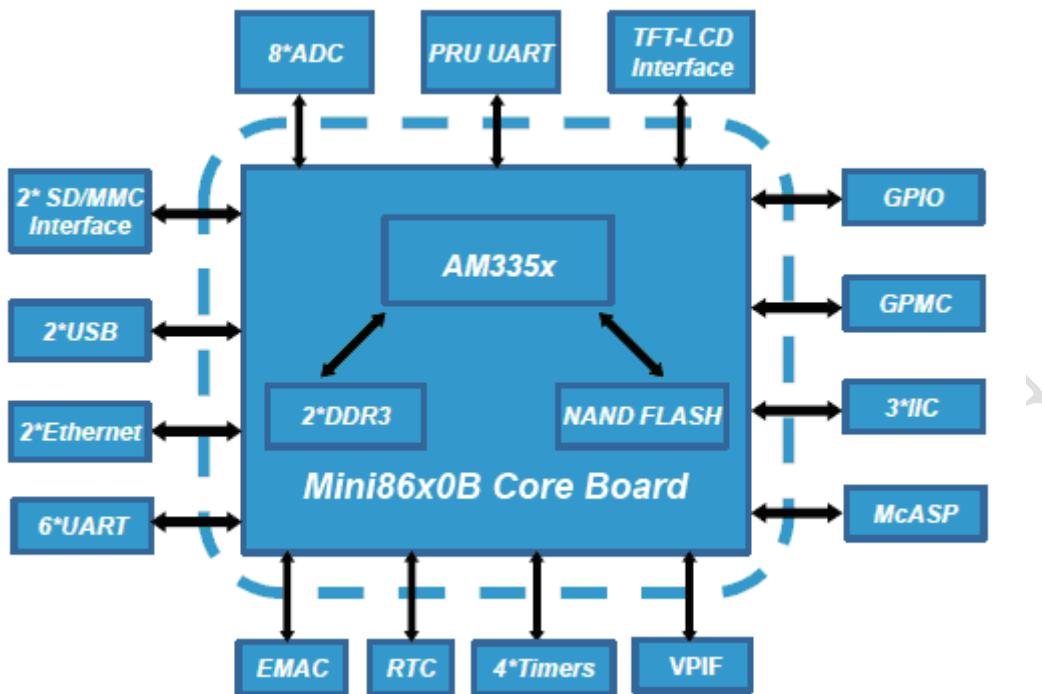


图 1-1 Mini86x0B 结构图

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

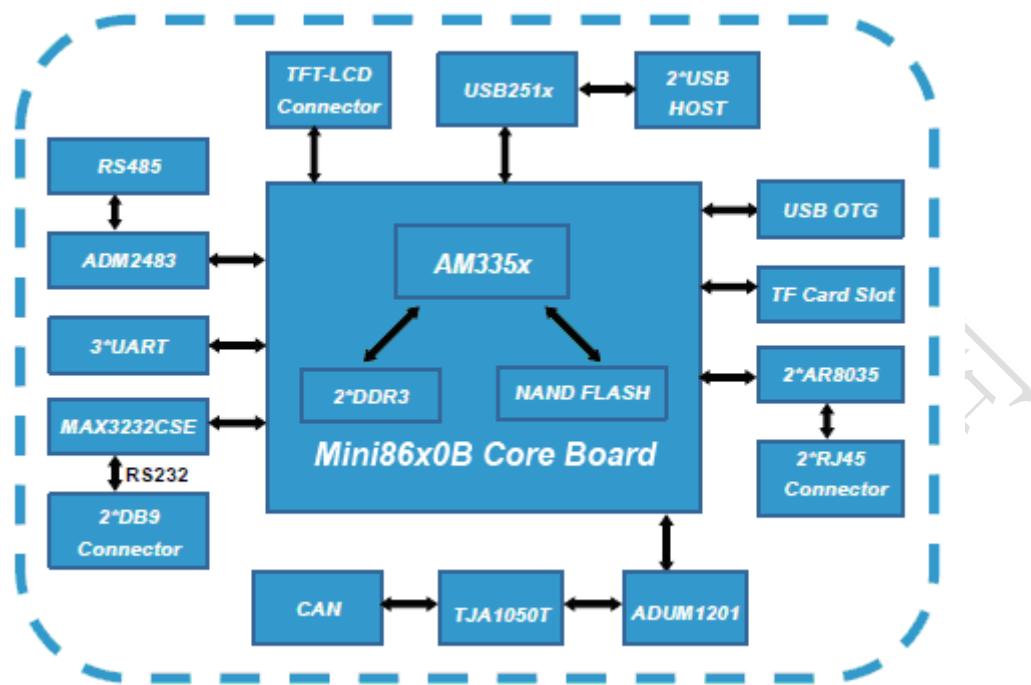


图 1-2 SBC8600B结构图

深圳市英蓓特科技有限公司

1.3 硬件尺寸

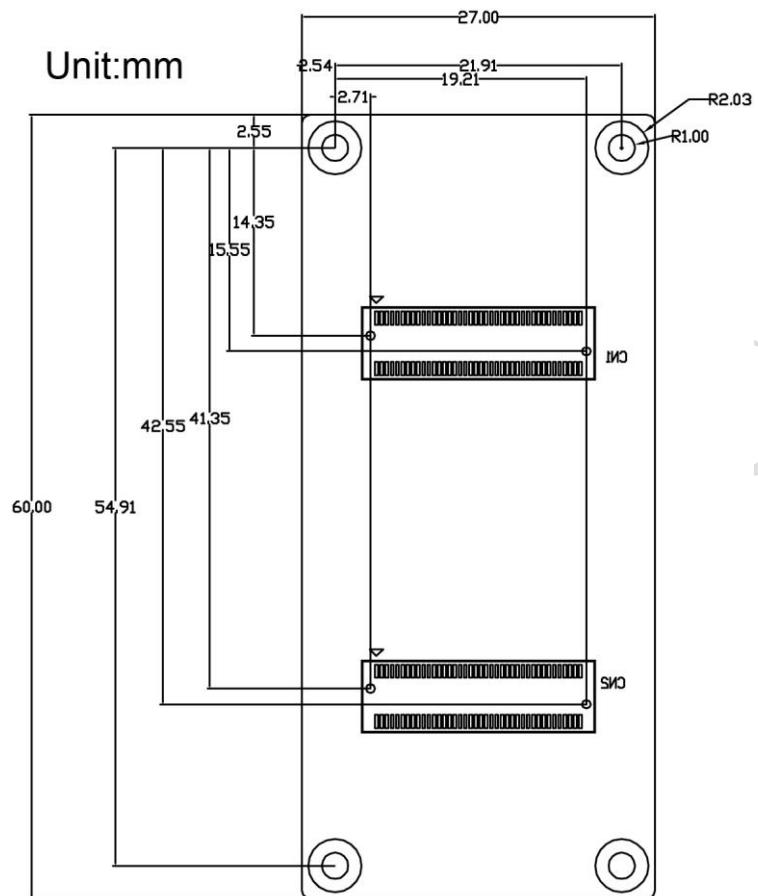


图 1-3 Mini8600B/Mini8610B硬件尺寸图

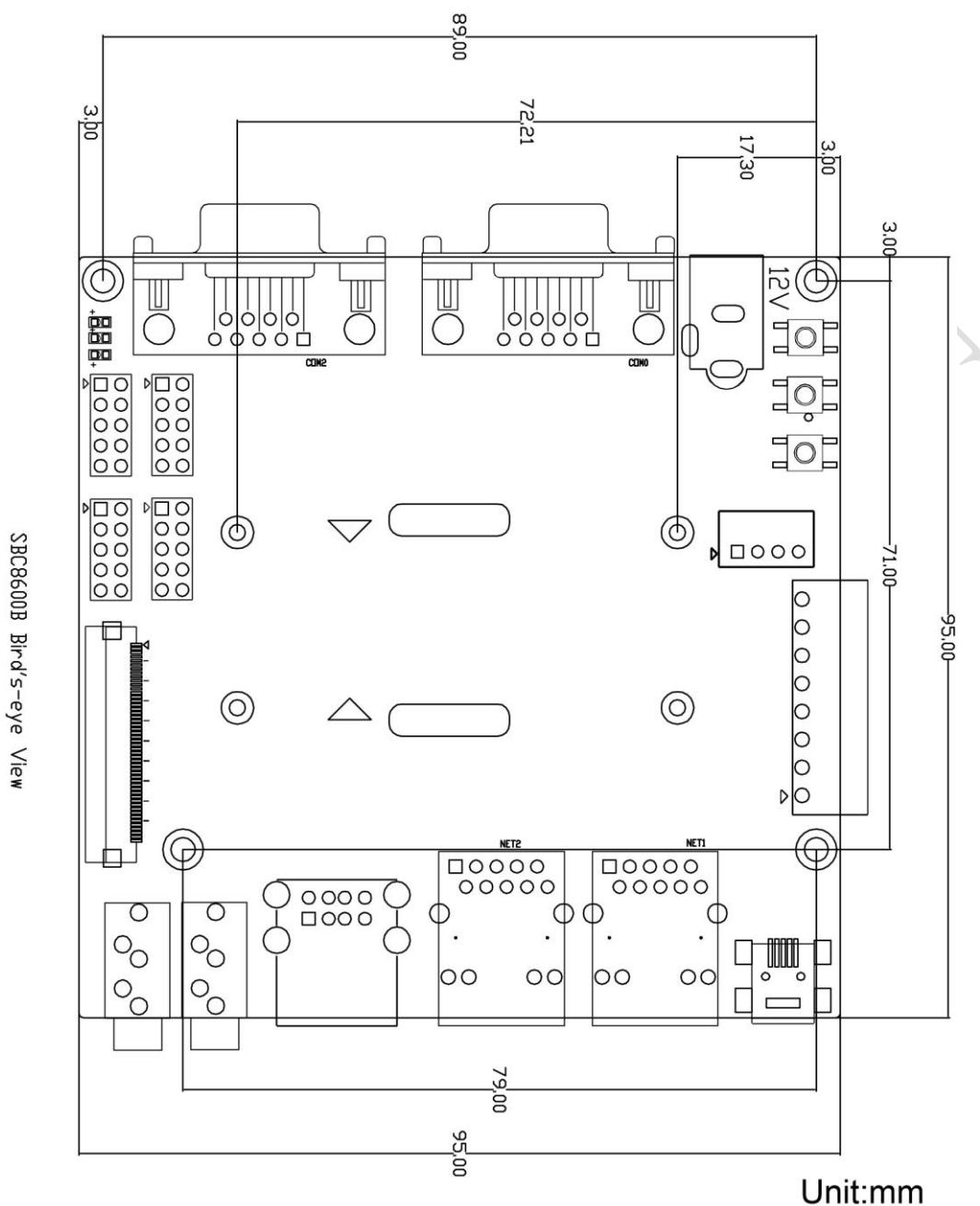


图 1-4 扩展板硬件尺寸

1.4 配套模块支持情况

表 1-1 SBC8600B外接模块支持

名称	Linux	Android	WinCE	相关资料

第2章 硬件系统

2.1 CPU

2.1.1 CPU 介绍

AM335x 是基于 ARM Cortex-A8 的微处理器，在图像、图形处理、外设和诸如 etherCAT 和 PROFIBUS 的工业接口选项方面进行了增强，并支持 Linux, WinCE, Android 等高级操作系统。

微处理器包含下列子系统：MPU 子系统基于 ARM Cortex-A8 微处理器；POWERVR SGX 图形加速子系统，主要用于 3D 图形加速以支持显示和游戏效果；可编程实时单元子系统(PRUSS)，使用户可以创建各种超越本地外设的数字资源。此外，PRUSS 独立于 ARM 核，这就允许设备有独立的操作和时钟，在复杂系统解决方案中有更大的灵活性。

2.1.2 CPU 特性

Clock

AM335x 微处理器有两个输入时钟：OSC1 和 OCC0，两个输出时钟信号：LCKOUT1 和 LCKOUT2.

OSC1 振荡器为 RTC 提供 32.768KHZ 参考时钟并用于连接 RTC_XTALIN 和 RTC_XTALOUT 终端。

OSC0 振荡器为所有无 RT 功能的时钟提供 19.2-MHz, 24-MHz, 25-MHz 或 26-MHz 参考时钟，并用于连接 XTALIN 和 XTALOUT 终端。

Reset

复位功能取决于 CPU 的 PWRONRSTn 信号，低电平有效。

通用接口（General-Purpose Interface）

通用接口包括 4 组通用输入输出接口（GPIO），每一组 GPIO 模组提供 32 个专用的通

用接口输入输出管脚，因此通用的 GPIO 可以高达 128 个 (4x32) 管脚。

可编程实时单元子系统 (Programmable Real-Time Unit Subsystem)

AM335x 下的可编程实时单元子系统(PRUSS)包括两个可编程实时单元、12KB 带 Single-Error 检测 (奇偶校验) 的共享 RAM、3 个可被每个 PRU 的 120 字节寄存器 BANK、用于处理系统输入事件的中断控制器模块和以下内部外设：

- 1 路 UART，具有流控制，最高 12Mbps
- 两路 MII 以太网端口，支持工业以太网，例如 EtherCAT™
- 1 路 MDIO 端口
- 一路增强型捕获模块 (eCAP)

3D 图形引擎

POWERVR® SGX 图形加速器子系统用于 3D 图形加速以支持显示和游戏效果，该子系统的主要特性如下：

- Tile-Based 架构，处理能力高达 20Mploy/秒
- 通用可扩展渲染引擎是一个具有像素和顶点渲染功能的多线程引擎
- 超过 Microsoft VS3.0、PS3.0 和 OGL2.0 的高级渲染功能指令集
- 工业标准 API，支持 Direct3D Mobile、OGL-ES 1.1 和 2.0、OpenVG 1.0 和 OpenMaxIL

2.2 外围芯片介绍

2.2.1 NAND Flash H27U4G8F2DTR-BC

H27U4G8F2DTR-BC 是 SBC8600B 的 NAND Flash 芯片，容量为 512MB。

若想了解更多关于此芯片的信息，请打开 Disk-SBC8600B\HW design\datasheet\NAND Flash\H27U4G8F2DTR-BC.pdf 文档。

2.2.2 DDR H5TQ2G83CFR-H9C

H5TQ2G83DFR-H9C 是 SBC8600B 的 DDR3 SDRAM 芯片，容量为 256MB，
Mini86x0B 配有 2 片 H5TQ2G83DFR-H9C 芯片。

若想了解更多关于此芯片的信息，请打开 Disk-SBC8600B\HW design\datasheet\DDR\
H5TQ2G83DFR.pdf 文档

2.2.3 Ethernet AR8035

AR8035 是 SBC8600B 低功耗、低 BOM 成本的以太网芯片，它集成了 10/100/1000
千兆位收发器。它是单端口 10/100/1000 Mbps 三速以太网 PHY，并支持 MAC.TM RGMII
接口。

AR8035 支持 IEEE 802.3az 高效节能以太网（EEE）标准和 Atheros 专有的的
SmartEEE，它允许无需 802.3az 功能支持的传统 MAC/SoC 设备作为完整的 802.3az 系统。

SBC8600B 可通过直通网线连接到网络 hub 上，也可用交叉网线与电脑直接相连。

若想了解更多关于此芯片的信息，请打开 Disk-SBC8600B\HW
design\datasheet\LAN\AR8035.pdf 文档。

2.2.4 MAX3232

MAX3232 的功能主要是将 TTL 电平转换为 RS232 电平，以适应与 PC 的 RS232 串口
相互通信。

SBC8600B 使用的是 UART0 作调试串口，因 CPU 的 UART0 默认电压是 1.8V，需要
将电压转换为 3.3V，方可满足外部使用。

若想了解更多关于此芯片的信息，请打开 Disk-SBC8600B\HW design\datasheet\
Serial\MAX3232CSE.pdf 文档。

2.3 硬件接口

2.3.1 Mini86x0B

2.3.1.1 CN1 接口



图 2-1 Mini86x0B CN1 接口示意图

表 2-1 CN1 接口

CN1		
管脚	信号	描述
1	GND	GND
2	VDDS_RTC	Supply voltage for RTC
3	CLK_OUT1	Clock out1
4	CLK_OUT2	Clock out2
5	MMC0_DAT0	MMC0 data bus
6	MMC0_DAT1	MMC0 data bus
7	MMC0_DAT2	MMC0 data bus
8	GLOBLE_RESETN	SYS_RESET IN/ OUTPUT
9	MMC0_DAT3	MMC0 data bus
10	AM335X_PWRON_RESETN	CPU PWRON Reset
11	GND	GND
12	GND	GND
13	AM355X_PRU_UART0_CTS	PRU UART0 Clear To Send
14	AM355X_PRU_UART0_RX	PRU UART0 receive data
15	AM355X_PRU_UART0_RTS	PRU UART0 request to send
16	AM355X_PRU_UART0_TX	PRU UART0 transmit data
17	AM355X_UART0_RX	UART0 receive data
18	AM355X_UART3_RX	UART3 receive data

CN1		
管脚	信号	描述
19	AM355X_UART0_TX	UART0 transmit data
20	AM355X_UART3_TX	UART3 transmit data
21	AM355X_CAN0_RX	CAN0 receive data
22	AM355X_I2C0_SDA	I2C0 master serial data
23	AM355X_CAN0_TX	CAN0 transmit data
24	AM355X_I2C0_SCL	I2C0 master serial clock
25	AM355X_UART4_RX	UART4 receive data
26	AM355X_UART1_RX	UART1 receive data
27	AM355X_UART4_TX	UART4 transmit data
28	AM355X_UART1_TX	UART1 transmit data
29	GND	GND
30	GND	GND
31	MII1_COL	MII1 collision detect
32	AM355X_USB0_DRVVBUS	USB0 controller VBUS control output
33	MII1_TX_CLK	MII1 transmit clock
34	AM355X_USB1_DRVVBUS	USB1 controller VBUS control output
35	MII1_TX_EN	MII1 transmit enable
36	MII1_REF_CLK	MII1 reference clock
37	MII1_TXD3	MII1 transmit data
38	MII1_CRS	MII1 carrier sense
39	MII1_TXD2	MII1 transmit data
40	MII1_RX_ER	MII1 receive data error
41	MII1_TXD1	MII1 transmit data
42	MII1_RX_DV	MII1 receive data valid
43	MII1_TXD0	MII1 transmit data
44	MII1_RX_CLK	MII1 receive clock
45	MII_MDIO	MII MDIO DATA
46	MII1_RXD3	MII1 receive data
47	MII_MDC	MII MDIO CLK
48	MII1_RXD2	MII1 receive data
49	GND	GND
50	MII1_RXD1	MII1 receive data
51	AM355X_USB0_DM	USB0 DM-
52	MII1_RXD0	MII1 receive data

CN1		
管脚	信号	描述
53	AM355X_USB0_DP	USB0 DP
54	MMC0_CMD	MMC0 Command Signal
55	GND	GND
56	USB0_VBUS	USB0 bus voltage
57	AM355X_USB1_DM	USB1 data-
58	AM355X_USB1_ID	USB1 ID
59	AM355X_USB1_DP	USB1 data+
60	AM355X_USB0_ID	USB0 ID
61	GND	GND
62	USB1_VBUS	USB1 bus voltage
63	GPMC_A0	GPMC address
64	GPMC_A7	GPMC address
65	GPMC_A5	GPMC address
66	GPMC_A11	GPMC address
67	GPMC_A4	GPMC address
68	GPMC_A10	GPMC address
69	GPMC_A3	GPMC address
70	GPMC_A9	GPMC address
71	GPMC_A2	GPMC address
72	GPMC_A8	GPMC address
73	GPMC_A6	GPMC address
74	GPMC_A1	GPMC address
75	GND	GND
76	GND	GND
77	VDD_3V3	Power
78	VDD_3V3	Power
79	VDD_3V3	Power
80	VDD_3V3	Power

2.3.1.2 CN2 接口



图 2-2 Mini86x0B CN2 接口示意图

表 2-2 CN2 接口

CN2		
管脚	信号	描述
1	GND	GND
2	GND	GND
3	MCASP0_AHCLKX	MCASP0 transmit master clock
4	MCASP0_ACLKX	MCASP0 transmit bit clock
5	MCASP0_FSX	MCASP0 transmit frame sync
6	MCASP0_AXR0	MCASP0 serial data(I/O)
7	MCASP0_AHCLKR	MCASP0 receiver master clock
8	MMC0_CLK	MMC0 clock
9	MCASP0_FSR	MCASP0 receive frame sync
10	MCASP0_AXR1	MCASP0 serial data(I/O)
11	GND	GND
12	GND	GND
13	VDDA_ADC	Supply voltage range for ADC
14	AM355X_ADC0	ADC0
15	AM355X_ADC1	ADC1
16	AM355X_ADC2	ADC2
17	AM355X_ADC3	ADC3
18	AM355X_ADC4	ADC4
19	AM355X_ADC5	ADC5
20	AM355X_ADC6	ADC6
21	AM355X_ADC7	ADC7
22	GND_ADC	GND ADC
23	GND	GND
24	GND	GND

CN2		
管脚	信号	描述
25	LCD_DATA1	LCD data bus
26	LCD_DATA12	LCD data bus
27	LCD_DATA0	LCD data bus
28	LCD_DATA10	LCD data bus
29	LCD_DATA5	LCD data bus
30	LCD_DATA13	LCD data bus
31	LCD_DATA4	LCD data bus
32	LCD_DATA11	LCD data bus
33	LCD_DATA6	LCD data bus
34	LCD_DATA14	LCD data bus
35	LCD_DATA8	LCD data bus
36	LCD_VSYNC	LCD vertical sync
37	GND	GND
38	GND	GND
39	LCD_DATA9	LCD data bus
40	LCD_PCLK	LCD pixel clock
41	LCD_DATA15	LCD data bus
42	GPMC_AD11	GPMC address & data
43	LCD_DATA3	LCD data bus
44	GPMC_AD15	GPMC address & data
45	LCD_DATA2	LCD data bus
46	GPMC_AD14	GPMC address & data
47	LCD_DATA7	LCD data bus
48	GPMC_WAIT0	GPMC wait0
49	LCD_HSYNC	LCD horizontal sync
50	GPMC_BEN1	GPMC byte enable 1
51	GND	GND
52	GND	GND
53	LCD_EN	LCD AC bias enable chip select
54	GPMC_WPN	GPMC write protect
55	GPMC_AD13	GPMC address & data
56	GPMC_CSN3	GPMC chip select
57	GPMC_AD9	GPMC address & data
58	GPMC_CSN2	GPMC chip select
59	GPMC_AD10	GPMC address & data
60	GPMC_CLK	GPMC clock

CN2		
管脚	信号	描述
61	GPMC_AD8	GPMC address & data
62	GPMC_AD6	GPMC address & data
63	GPMC_AD12	GPMC address & data
64	GND	GND
65	GND	GND
66	GPMC_CSN1	GPMC chip select1
67	GPMC_ADVN_ALE	GPMC address valid/address latch enable
68	GPMC_AD5	GPMC address & data
69	GPMC_BEN0_CLE	GPMC byte enable 0/Command latch enable
70	GPMC_AD4	GPMC address & data
71	GPMC_OEN_REN	GPMC output /read enable
72	GPMC_AD1	GPMC address & data
73	GPMC_AD2	GPMC address & data
74	GPMC_AD0	GPMC address & data
75	GPMC_AD3	GPMC address & data
76	GPMC_CSN0	GPMC chip select0
77	GPMC_AD7	GPMC address & data
78	GPMC_WEN	GPMC write enable
79	GND	GND
80	GND	GND

2.3.2 扩展板

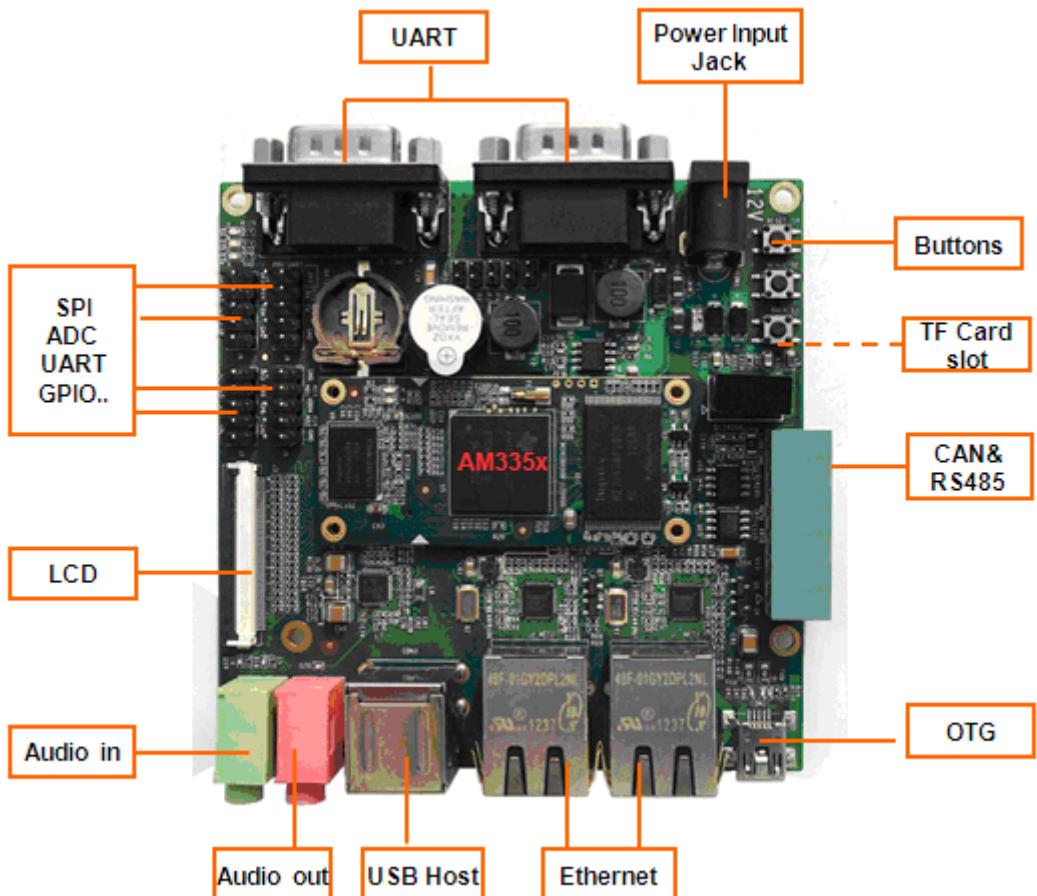


图 2-3 SBC8600B 扩展板接口示意图

— 代表接口在板子背面

— 代表接口在板子正面

2.3.2.1 电源接口

表 2-3 电源接口

CON1		
管脚	信号	描述
1	GND	GND
2	+12V	Power supply (+12V)
3	NC	NC

2.3.2.2 TFT_LCD 接口

表 2-4 TFT_LCD 接口

J3		
管脚	信号	描述
1	B0	LCD Pixel data bit 0
2	B1	LCD Pixel data bit 1
3	B2	LCD Pixel data bit 2
4	B3	LCD Pixel data bit 3
5	B4	LCD Pixel data bit 4
6	B5	LCD Pixel data bit 5
7	B6	LCD Pixel data bit 6
8	B7	LCD Pixel data bit 7
9	GND1	GND
10	G0	LCD Pixel data bit 8
11	G1	LCD Pixel data bit 9
12	G2	LCD Pixel data bit 10
13	G3	LCD Pixel data bit 11
14	G4	LCD Pixel data bit 12
15	G5	LCD Pixel data bit 13
16	G6	LCD Pixel data bit 14
17	G7	LCD Pixel data bit 15
18	GND2	GND
19	R0	LCD Pixel data bit 16
20	R1	LCD Pixel data bit 17
21	R2	LCD Pixel data bit 18
22	R3	LCD Pixel data bit 19
23	R4	LCD Pixel data bit 20
24	R5	LCD Pixel data bit 21
25	R6	LCD Pixel data bit 22
26	R7	LCD Pixel data bit 23
27	GND3	GND
28	DEN	AC bias control (STN) or pixel data enable (TFT)
29	H SYNC	LCD Horizontal Synchronization
30	V SYNC	LCD Vertical Synchronization
31	GND	GND
32	CLK	LCD Pixel Clock

J3		
管脚	信号	描述
33	GND4	GND
34	X+	X+ Position Input
35	X-	X- Position Input
36	Y+	Y+ Position Input
37	Y-	Y- Position Input
38	NC	NC
39	NC	NC
40	NC	NC
41	NC	NC
42	IIC_CLK	IIC master serial clock
43	IIC_DAT	IIC serial bidirectional data
44	GND5	GND
45	VDD1	3.3V
46	VDD2	3.3V
47	VDD3	5V
48	VDD4	5V
49	RESET	Reset
50	PWREN	Backlight enable

注意：

请不要带电拔插 LCD 排线

2.3.2.3 音频输出接口

表 2-5 音频输出接口

HEADPHONE1		
管脚	信号	描述
1	GND	GND
2	NC	NC
3	Right	Right output
4	NC	NC
5	Left	Left output

2.3.2.4 音频输入接口

表 2-6 音频输入接口

MIC1		
管脚	信号	描述
1	GND	GND
2	NC	NC
3	MIC In	input
4	NC	NC
5	MIC In	input

2.3.2.5 USB HOST 接口

表 2-7 USB HOST 接口

CON3		
管脚	信号	描述
1	VBUA	+5V
2	DA-	USB Data-
3	DA+	USB Data+
4	GNDA	GND

2.3.2.6 USB OTG 接口

表 2-8 USB OTG 接口

CON2		
管脚	信号	描述
1	VB	+5V
2	D-	USB Data-
3	D+	USB Data+
4	ID	USB ID
5	G1	GND

2.3.2.7 TF 卡接口

表 2-9 TF 卡接口

TF1		
管脚	信号	描述
1	DAT2	Card data 2
2	CD/DAT3	Card data 3
3	CMD	Command Signal
4	VDD	VDD

TF1		
管脚	信号	描述
5	CLOCK	Clock
6	VSS	VSS
7	DAT0	Card data 0
8	DAT1	Card data 1
9	CD	Card detect

2.3.2.8 LAN 接口

表 2-10 LAN 接口

J1,J2		
管脚	信号	描述
1	TD1+	Transmit Data1+
2	TD1-	Transmit Data1-
3	TD2+	Transmit Data2+
4	TD2-	Transmit Data2-
5	TCT	Transmit common terminal
6	RCT	Receive common terminal
7	RD1+	Receive Data1+
8	RD1-	Receive Data1-
9	RD2+	Receive Data2+
10	RD2-	Receive Data2-
11	GRLA	+2.5V
12	GRLC	LINK active LED
13	YELC	100M linked LED
14	YELA	+2.5V

2.3.2.9 串口

表 2-11 串口

J4(UART0), J5(UART2)		
管脚	信号	描述
1	NC	NC
2	RXD	Receive data
3	TXD	Transmit data
4	NC	NC
5	GND	GND
6	NC	NC
7	RTS	Request To Send

J4(UART0), J5(UART2)		
管脚	信号	描述
8	CTS	Clear To Send
9	NC	NC

2.3.2.10 CAN&RS485 接口

表 2-12 CAN&RS485 接口

U22		
管脚	信号	描述
1	+12V	+12V
2	GND	GND
3	GND2	Isolated GND
4	485B1	485B
5	485A1	485A
6	GND1	Isolated GND
7	CANL	CANL
8	CANH	CANH

2.3.2.11 ADC

表 2-13 ADC

J9		
管脚	信号	描述
1	GND	GND
2	GND	GND
3	ADC_CH1	ADC1
4	ADC_CH3	ADC3
5	VDDA_ADC	Power
6	VDDA_ADC	Power
7	ADC_CH2	ADC2
8	ADC_CH4	ADC4
9	GND	GND
10	GND	GND

2.3.2.12 SPI 接口

表 2-14 SPI 接口

J8		
管脚	信号	描述
1	+3.3V	3.3V
2	+3.3V	3.3V
3	SPI0_D1	SPI0 data1
4	SPI0_CLK	SPI0 clock
5	SPI0_CS0	SPI enable0
6	SPI0_D0	SPI data0
7	GND	GND
8	GND	GND
9	GND	GND
10	GND	GND

2.3.2.13 扩展接口

表 2-15 扩展接口 1

J6		
管脚	信号	描述
1	VIO_3V3	+3.3V
2	VIO_3V3	+3.3V
3	UART3_TX_3V3	UART3 Transit data 3.3V level
4	UART4_TX_3V3	UART4 Transit data 3.3V level
5	UART3_RX_3V3	UART3 receive data 3.3V level
6	UART4_RX_3V3	UART4 receive data 3.3V level
7	GND	GND
8	GND	GND
9	GND	GND
10	GND	GND

表 2-16 扩展接口 2

J7		
管脚	信号	描述
1	VIO_3V3	+3.3V
2	VIO_3V3	+3.3V
3	UART5_TX_3V3	UART5 Transit data 3.3V level
4	GPIO0_9	GPIO
5	UART5_RX_3V3	UART5 receive data 3.3V level

J7		
管脚	信号	描述
6	GPIO2_0	GPIO
7	GND	GND
8	GND	GND
9	GND	GND
10	GND	GND

2.3.2.14 按钮

表 2-17 按钮

S1-3		
管脚	信号	描述
S2	MENU	System menu key
S3	BACK	System back key
S4	Reset	System Reset key

2.3.2.15 LED

表 2-18 LED

LEDs		
LED 灯	灯定义	描述
1	D4	电源指示灯
2	D35	用户自定义灯
3	D36	用户自定义灯

第3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。

3.1 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，可以让您轻松快速地使用 SBC8600B 开发套件来进行 Linux 应用和系统的开发。

3.1.1 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 3-1 程序和代码

类别	位置
Demo 程序	CD\android
	CD\linux\demo\tisdk
应用程序	CD\linux\example\libsocketcan-0.0.9.tar.bz2
	CD\linux\example\uart_test.tar.bz2
	CD\linux\example\can_test.tar.bz2
源代码	CD\linux\source\linux-4.1.tar.xz
	CD\linux\source\rootfs.tar.xz
	CD\linux\source\u-boot-2015.07.tar.xz

工具	CD\linux\tools
预编译映像	CD\linux\image\

3.1.2 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 3-2 BSP 软件包内容

名称		备注	源码/二进制文件
BIOS	spl	NAND	提供源码
		MMC/SD	提供源码
		FAT	提供源码
	u-boot	NAND	提供源码
		MMC/SD	提供源码
		FAT	提供源码
		NET	提供源码
Kernel	linux-4.1	支持ROM/CRAM/EXT4/FILE/NFS/JFFS2/UBIFS等多种文件系统	提供源码
Device Driver	Serial	串口驱动	提供源码
	RTC	硬件时钟驱动	提供源码
	NET	10/100M/1000M以太网驱动	提供源码
	CAN	CAN总线驱动	提供源码
	Flash	NAND Flash 驱动 (支持 NAND boot)	提供源码
	SPI	SPI驱动	提供源码
	I2C	I2C驱动	提供源码
	LCD	TFT LCD 驱动	提供源码
	Touch Screen	4 线触摸屏控制器驱动	提供源码
	ADC	4 路普通ADC通道	提供源码
	MMC/SD	MMC/SD控制器驱动	提供源码
	USB OTG	USB OTG 2.0 驱动	提供源码
	Audio	声卡驱动(支持录/放音)	提供源码
	Keypad	GPIO键盘驱动	提供源码
	LED	用户LED灯驱动	提供源码
Rootfs	Debian 8	精简版, 不带桌面系统	提供映像
Demo	Android	Android 4.0.3 系统	提供源码
	TISDK	TISDK系统	提供源码

3.2 嵌入式 Linux 的组成

SBC8600B 出厂默认在 NAND Flash 中写入了的 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 和 rootfs 四个部分。以下为系统结构示意图：

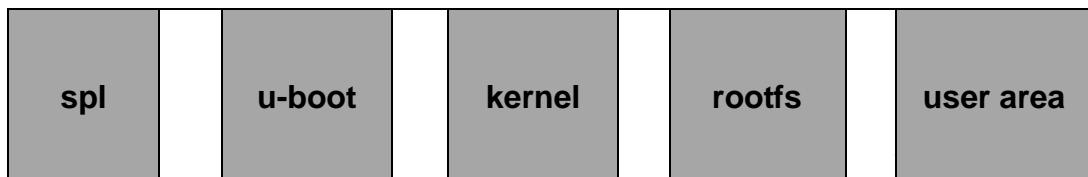


图 3-1 嵌入式Linux系统结构

- 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SBC8600B的硬件进行定制；
- 4) rootfs采用开源文件系统ubifs，特别适用于嵌入式系统。
- 5) User分区的使用

初次使用前需进行格式化等操作：

```
• root@arm:~# cat /proc/mtd
dev:    size  erasesize name
mtd0: 00020000 00020000 "NAND.SPL"
mtd1: 00020000 00020000 "NAND.SPL.backup1"
mtd2: 00020000 00020000 "NAND.SPL.backup2"
mtd3: 00020000 00020000 "NAND.SPL.backup3"
mtd4: 00040000 00020000 "NAND.u-boot-spl-os"
mtd5: 00100000 00020000 "NAND.u-boot"
mtd6: 00020000 00020000 "NAND.u-boot-env"
mtd7: 00020000 00020000 "NAND.u-boot-env.backup1"
mtd8: 00400000 00020000 "NAND.logo"
mtd9: 00800000 00020000 "NAND.kernel"
mtd10: 19000000 00020000 "NAND.file-system"
mtd11: 06200000 00020000 "NAND.user"
```

获知User分区对应的设备为/dev/mtd11。

- `root@arm:~# flash_erase /dev/mtd11 0 0`
- `root@arm:~# ubiattach /dev/ubi_ctrl -m 11`
- `root@arm:~# ubimkvol /dev/ubi1 -N User -m`

挂载:

ubiattach命令系统启动后仅需执行一次，不用重复执行。

- `root@arm:~# ubiattach /dev/ubi_ctrl -m 11`
- `root@arm:~# mount -t ubifs /dev/ubi1_0 /mnt`

卸载:

- `root@arm:~# umount /mnt`

3.3 开发环境搭建

用户使用 SBC8600B 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:

- ❑ 每一条指令前都加上了符号“•”，以便防止由于指令较长占用多行而造成误解。
- ❑ 请注意命令行中的空格；漏掉任何空格都会造成程序运行失败。

3.3.1 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/linux/tools目录下的交

叉编译工具解压到\$HOME目录下；

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

3.3.2 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=$HOME/tools/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux/bin:$HOME/tools:$PATH`

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。
- ❑ Android 开发环境的搭建除了以上步骤外，还需要进行额外的安装和配置，请访问<http://source.android.com/source/initializing.html>获取更多信息。

3.4 准备源代码

Linux 源代码可从产品附带的光盘获取，也可通过`git`和`repo`工具获取，具体方法如下：

3.4.1 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/linux/source/u-boot-2015.07.tar.xz`
- `tar xvf /media/cdrom/linux/source/linux-4.1.tar.xz`
- `mkdir rootfs`
- `sudo tar xvf /media/cdrom/linux/source/rootfs.tar.xz -C rootfs`
- `tar xvf /media/android/source/TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1.tar.bz2`

执行完以上操作后，当前目录下会生成 u-boot-2015.07、linux-4.1 和 rootfs 目录。

注意：

书 源码文件请不要解压到其他位置，以免编译时出错。

深圳市英蓓特科技有限公司

3.4.2 用 git 和 repo 工具获取源代码

用户还可以在 ubuntu 系统下通过 git 和 repo 工具下载源码：

1) 通过以下命令获取 bootloader 源码：

- `$ cd ~`
- `$ git clone -b u-boot-2015.07 https://github.com/embest-tech/u-boot-am33x.git`
- `$ cd u-boot-am33x/`

查看当前分支：

- `$ git branch`
* u-boot-2015.07

列出所有分支：

- `$ git branch -a`
* u-boot-2015.07
remotes/origin/AM335XPSP_04.06.00.03
remotes/origin/AM335XPSP_04.06.00.05
remotes/origin/AM335XPSP_04.06.00.06
remotes/origin/AM335XPSP_04.06.00.07
remotes/origin/AM335XPSP_04.06.00.08
remotes/origin/HEAD -> origin/master
remotes/origin/TIOP_AM335XPSP_04.06.00.08
remotes/origin/add-musb-to-AM335XPSP_04.06.00.03
remotes/origin/amsdk-05.04.01.00
remotes/origin/amsdk-05.06.00.00
remotes/origin/int_am335xpsp_04.06.00.01-v2011.09-for-sdk-05.03.00.00
remotes/origin/int_am335xpsp_04.06.00.01-v2011.09-for-sdk-05.03.01.00
remotes/origin/int_am335xpsp_04.06.00.01-v2011.09-for-sdk-05.04.00.00
remotes/origin/master
remotes/origin/u-boot-2015.07

2) 获取 linux 内核源码：

- `$ cd ~`
- `$ git clone -b linux-4.1 https://github.com/embest-tech/linux-am33x.git`
- `$ cd linux-am33x`

列出所有分支：

- `$ git branch`

```
* linux-4.1
```

注意：

随产品附带的光盘源码来自于 git 上的源码。以后源码有更新，我们会及时更新到 git 网站上去，用户可以随时用 git 工具更新到最新的源码。

3.5 编译

1) 编译启动代码

SBC8600B 支持 MMC/SD 启动与 NAND Flash 启动，如果不短接 SBC8600 底板上的 JP5 引脚，系统优先选择 NAND Flash 启动，如果短接 JP5 的引脚，系统优先选择 MMC/SD 启动。

下面介绍启动代码映像文件的生成方法：

- `cd u-boot-2015.07`
- `make distclean`
- `make am335x_sbc8600_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-4.1`
- `make distclean`
- `make embest_ti_sbc8600_defconfig`
- `make zImage`

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。

3) 文件系统生成

目前 SBC8600B 提供两种文件系统格式：Ramdisk 和 UBI 文件系统，其中 Ramdisk 文件系统在从 TF 卡启动开发板时使用，UBI 文件系统在从 NAND Flash 启动开发

板时使用。

3.6 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。

3.6.1 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

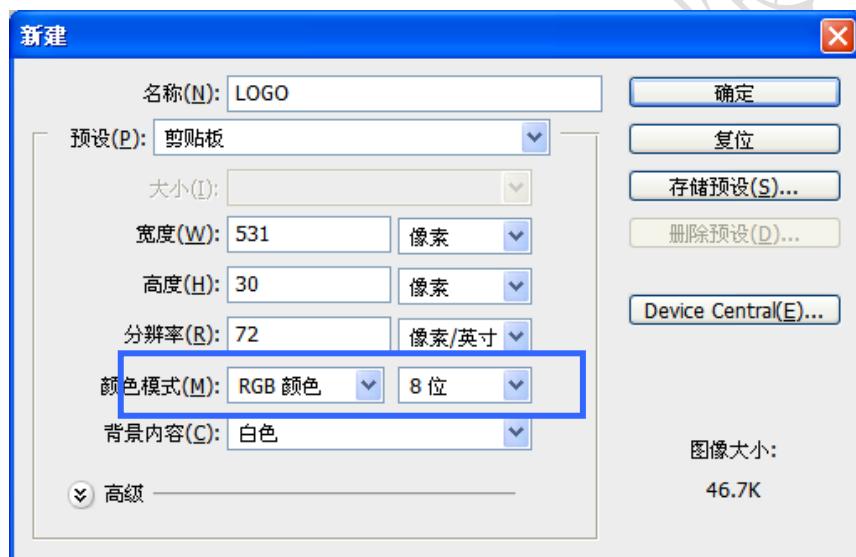


图 3-2 RGB颜色-8 位

- 保存图像，另存为 logo.bmp，弹出对话框：

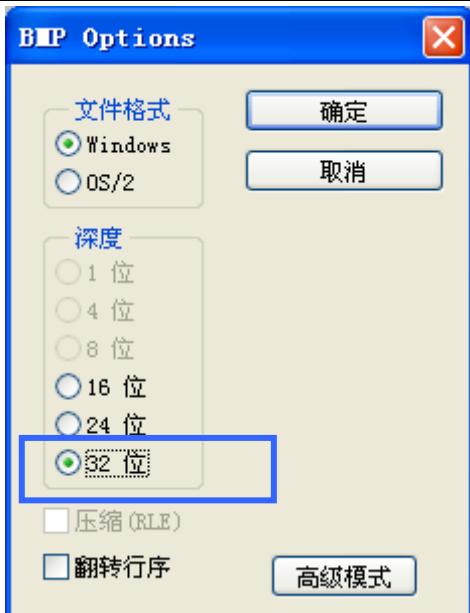


图 3-3 保存为 32 位色格式

3.6.2 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-4.1/arch/arm/configs/embest_ti_sbc8600_defconfig`

执行以下命令来进入系统配置菜单；

- `cd linux-4.1`
- `cp arch/arm/configs/embest_ti_sbc8600_defconfig .config`
- `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`。

3.6.3 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
```

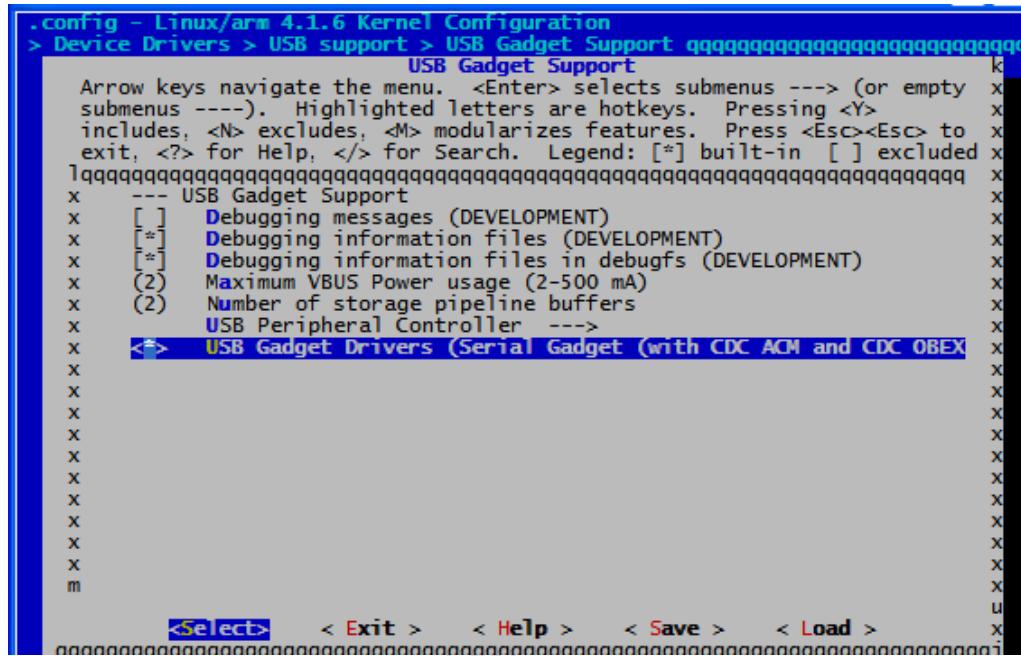


图 3-4 USB Gadget Driver Configuration

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

3.6.4 编译内核

请执行以下命令重新编译内核:

- **make dtbs zImage**

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

3.6.5 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 **CD\linux\tools\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]。

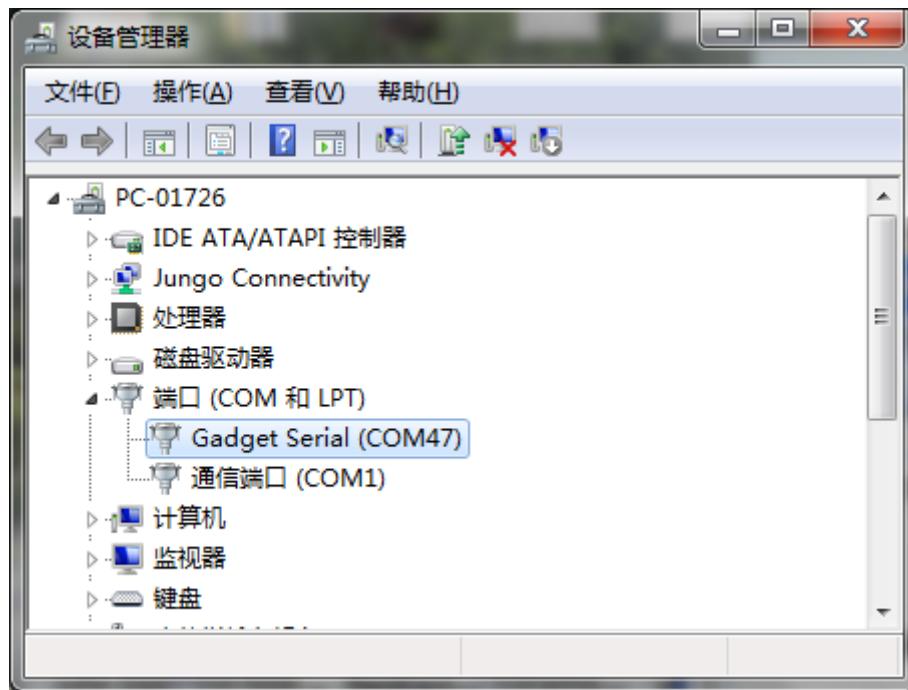


图 3-5 Gadget Serial 驱动安装完成

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。

3.7 驱动介绍

3.7.1 BSP 的所有驱动源码路径：

表 3-3 驱动路径

类别	名称	说明	驱动源码路径
BIOS	spl	NAND	drivers/mtd/nand/omap_gpmc.c
		MMC/SD	drivers/mmc/omap_hsmmc.c
		FAT	fs/
	u-boot	NAND	drivers/mtd/nand/omap_gpmc.c
		MMC/SD	drivers/mmc/omap_hsmmc.c
		FAT	fs/
		NET	drivers/net/cpsw.c
内核	Linux-4.1	支持 ROM/CRAM/EXT2/EXT3/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
	Flash	NAND flash 驱动(支持 nand boot)	drivers/mtd/nand/omap2.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

3.7.2 NAND

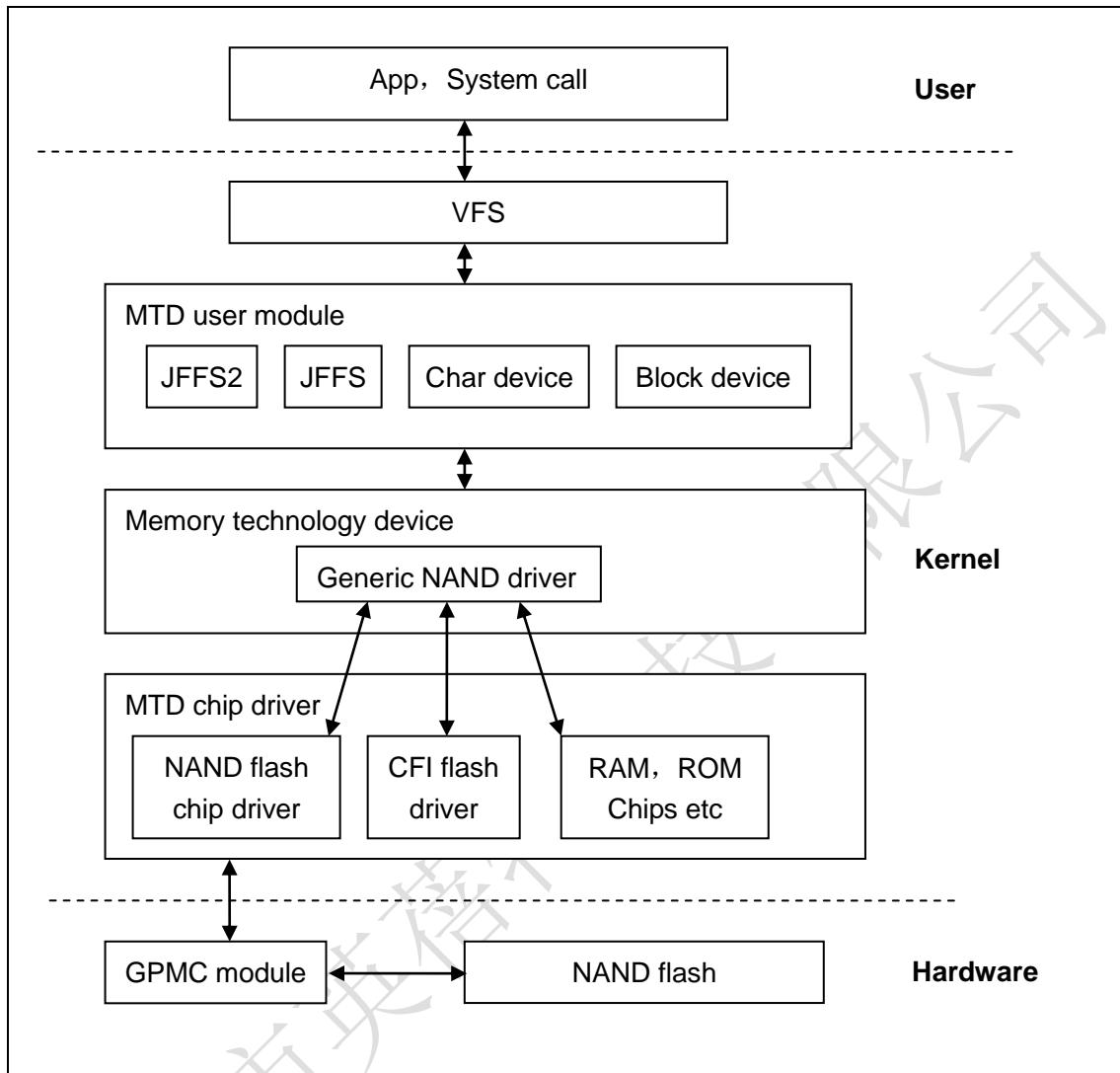


图 3-6 Modular structure for NAND

嵌入式系统中使用的固态存储器主要为 flash，在本系统中为 NAND Flash。

NAND Flash 作为块设备使用，其上建立有文件系统，用户与 NAND flash 的交互主要通过具体的文件系统来完成。为了屏蔽不同 flash 存储器之间的差异，内核在文件系统与具体的 flash 驱动之间插入了 MTD 子系统进行管理。

所以，用户访问 NAND Flash 经过以下流程：

User->System Call->VFS->Block Device Driver->MTD->NAND Flash Driver->NAND Flash。

驱动参考文件：

Linux-4.1/drivers/mtd/nand/

Linux-4.1/drivers/mtd/nand/omap2.c

3.7.3 SD/MMC

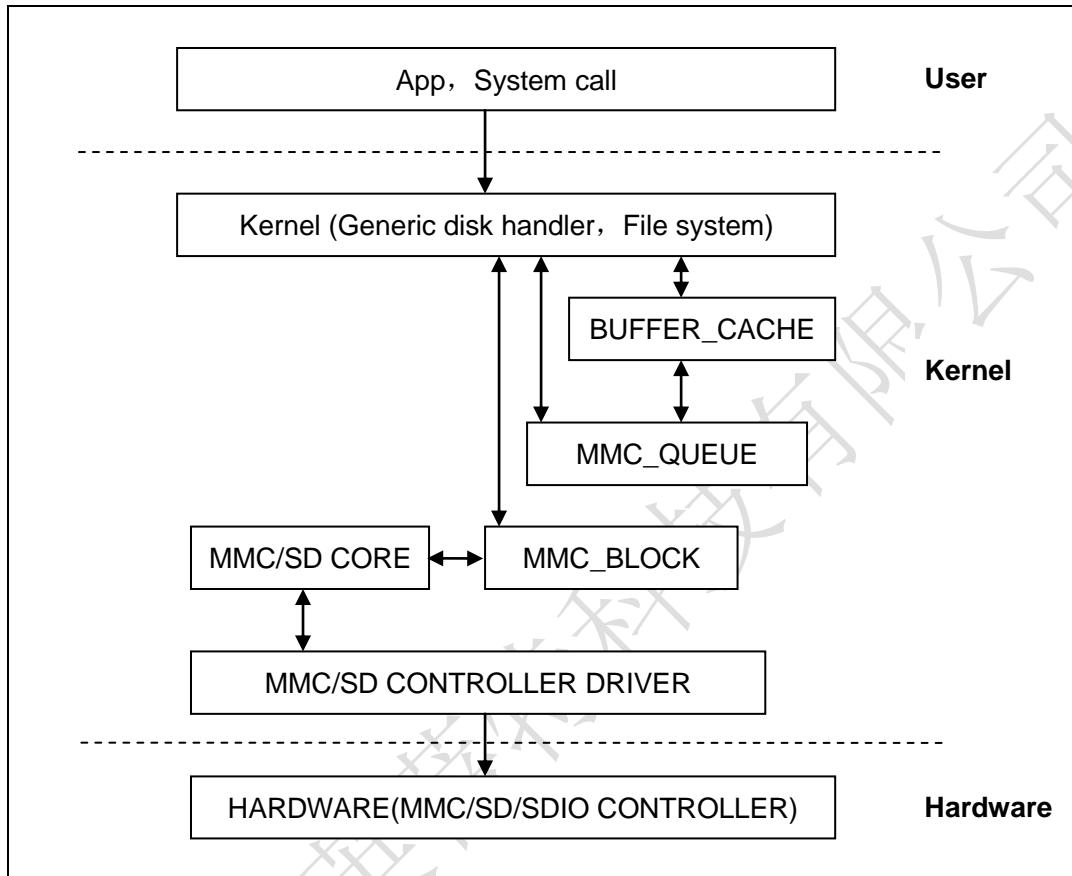


图 3-7 Modular structure for SD/MMC

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-4.1/drivers/mmc/

Linux-4.1/drivers/mmc/host/omap_hsmmc.c

3.7.4 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) 分辨率提升至 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-4.1/drivers/video/

Linux-4.1/drivers/gpu/drm/tilcdc/tilcdc_panel.c

Linux-4.1/drivers/video/of_display_timing.c

3.7.5 Audio in/out

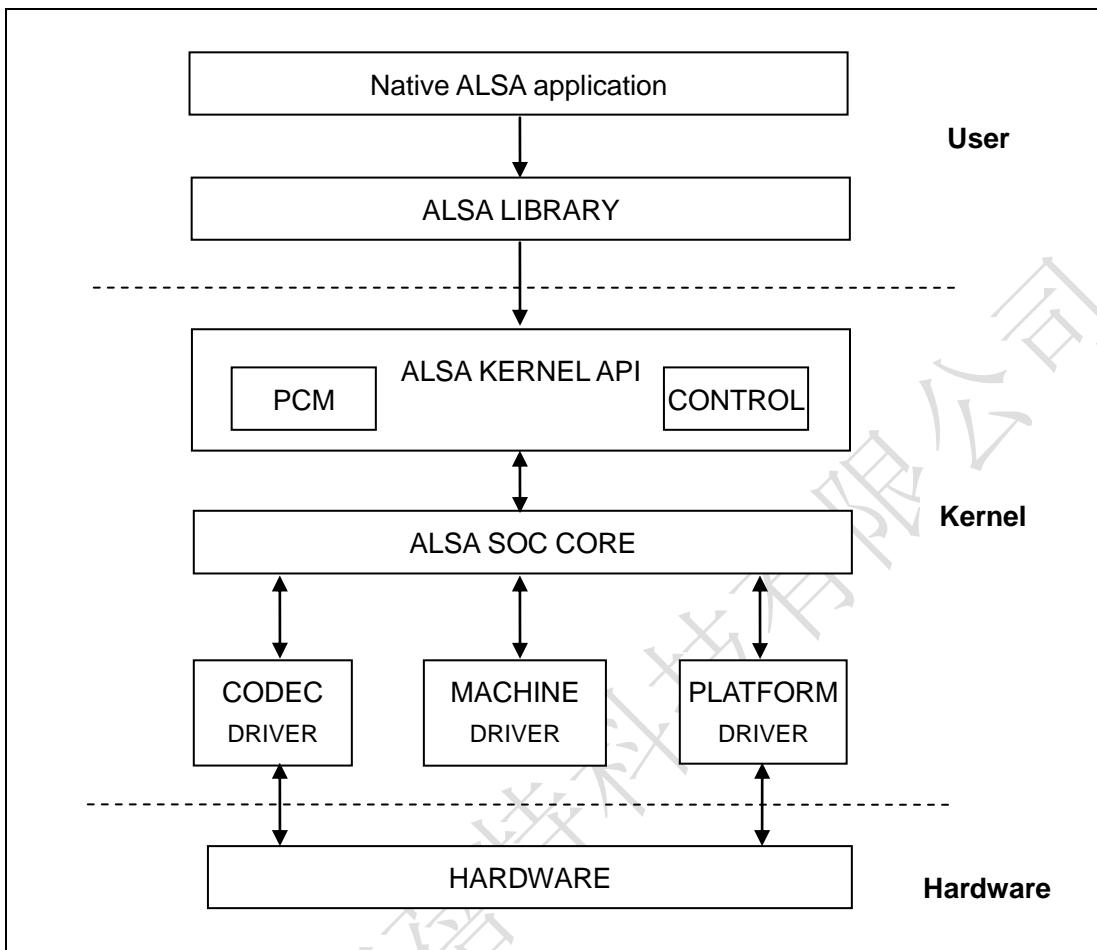


图 3-8 Modular structure for Audio

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-4.1/sound/soc/

Linux-4.1/sound/soc/davinci/davinci-evm.c

Linux-4.1/sound/soc/codecs/sgtl5000.c

3.8 驱动开发

3.8.1 GPIO_keys 驱动

1) 设备定义

Linux-4.1/arch/arm/boot/dts/embest-SBC-SBC8600.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-4.1/arch/arm/boot/dts/embest-SBC-SBC8600.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.gpi  
o0_20 */  
        0x08C (PIN_INPUT_PULLUP | MUX_MODE7) /* gpmc_clk.gpio2_1 */  
    >;
```

```
};
```

3) 驱动设计

Linux-3.2.0-psp04.06.00.08.sdk/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(pdev);
    }

...
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);
}
```

3.8.2 GPIO_leds 驱动

1) 设备定义

Linux-4.1/arch/arm/boot/dts/embest-SBC-SBC8600.dts

配置 GPIO1.30 为“sys”（系统心跳灯）、GPIO1.31 为“D36”，均为高电平有效。

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

    led@0 {
        label = "sys";
        gpios = <&gpio1 30 GPIO_ACTIVE_HIGH>;
        linux,default-trigger = "heartbeat";
        default-state = "off";
    };

    led@1 {
        label = "D36";
        gpios = <&gpio1 31 GPIO_ACTIVE_HIGH>;
        linux,default-trigger = "none";
        default-state = "off";
    };
}
```

2) GPIO pinmux 配置

Linux-4.1/arch/arm/boot/dts/embest-SBC-SBC8600.dts

配置 GPIO1.30 和 GPIO1.31 为 MODE7(gpio 模式)、AM33XX_PIN_OUTPUT(配置输出)

```
user_leds_default: pinmux_user_leds_default {
    pinctrl-single,pins = <
        /* leds */
```

```

        0x080 (PIN_OUTPUT_PULLUP | MUX_MODE7) /* gpmc_csn1.GPIO1_30
*/
        0x084 (PIN_OUTPUT_PULLUP | MUX_MODE7) /* gpmc_csn2.GPIO1_31
*/
    };
}

```

3) 驱动设计

Linux-4.1/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
<t piepho@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);
}
```

3.9 系统更新

SBC8600B 支持从 TF 卡和 NAND Flash 启动系统，下面将详细介绍两种不同的系统映像更新方式。

3.9.1 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\linux\tools\HP USB Disk Storage Format Tool.zip。

- a) 把 MMC/SD 卡插入 PC 下读卡器中
- b) 打开 HP USB Disk Storage Format Tool，出现类似提示如下：

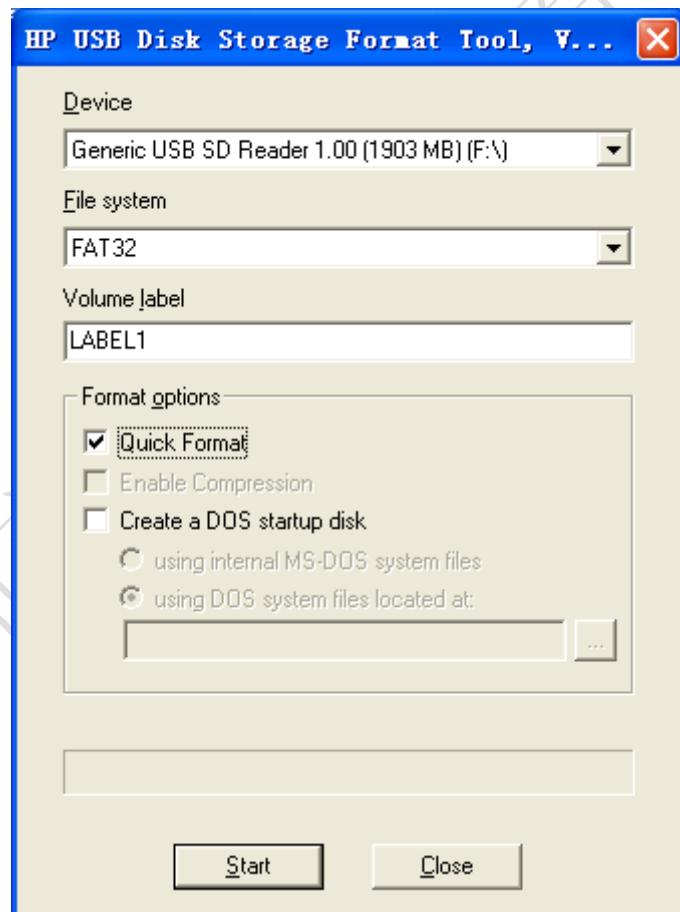


图 3-9 HP Format Tool

- 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 linux/image 目录下的所有文件拷贝到 TF 卡上，将 TF 卡接入板子，上电启动，串口信息显示如下：

注意：

- 默认 4.3 寸 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【3.10 显示模式配置】。
- SBC8600B 默认优先从 NAND flash 启动，如果 NAND flash 里面已经有映像，则需要用跳线帽短接板上的 JP5 引脚，使 SBC8600B 从 TF 卡启动。

```
U-Boot SPL 2015.07-ga66ad5e-dirty (Jun 19 2017 - 14:48:34)
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-ga66ad5e-dirty (Jun 19 2017 - 14:48:34 +0800)

Watchdog enabled
I2C: ready
DRAM: 512 MiB
NAND: 512 MiB
MMC: OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - bad CRC, using default environment

Net: <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot: 0
WARNING: Could not determine device tree to use
```

```
switch to partitions #0, OK
mmc0 is current device
SD/MMC found on device 0
reading boot.scr
** Unable to read file boot.scr **
reading uEnv.txt
199 bytes read in 5 ms (38.1 KiB/s)
Loaded env from uEnv.txt
Importing environment from mmc0 ...
Running uenvcmd ...
reading /embest_SBC_SBC8600.dtb
36786 bytes read in 11 ms (3.2 MiB/s)
reading /zImage
3537760 bytes read in 260 ms (13 MiB/s)
reading ramdisk.img
12034853 bytes read in 861 ms (13.3 MiB/s)
Booting from ramdisk ...
Kernel image @ 0x82000000 [ 0x0000000 - 0x35fb60 ]
## Loading init Ramdisk from Legacy Image at 88080000 ...
    Image Name:
        Created: 2017-04-24 9:02:19 UTC
        Image Type: ARM Linux RAMDisk Image (gzip compressed)
        Data Size: 12034789 Bytes = 11.5 MiB
        Load Address: 23000000
        Entry Point: 23000000
        Verifying Checksum ... OK
## Flattened Device Tree blob at 88000000
    Booting using the fdt blob at 0x88000000
    Loading Ramdisk to 8f485000, end 8ffff2e5 ... OK
    Loading Device Tree to 8f479000, end 8f484fb1 ... 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 cpufreq
[    0.000000] Linux version 4.1.6 (chengpg@embest-tech) (gcc version 4.9.2 20140904
(prerelease) (crosstool-NG linaro-1.13.1-4.9-2014.09 - Linaro GCC 4.9-2014.09) ) #56
PREEMPT Mon Jun 19 18:01:17 CST 2017
[    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 SBC8600
[ 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 )

.....
[ 2.649525] RAMDISK: gzip image found at block 0
[ 5.561390] EXT4-fs (ram0): mounting ext2 file system using the ext4 subsystem
[ 5.570741] EXT4-fs (ram0): mounted filesystem without journal. Opts: (null)
[ 5.578166] VFS: Mounted root (ext2 filesystem) on device 1:0.
[ 5.584888] devtmpfs: mounted
[ 5.588600] Freeing unused kernel memory: 268K (c090b000 - c094e000)
[ 5.644967] EXT4-fs (ram0): re-mounted. Opts: (null)

Starting logging: OK
Populating /dev using udev: [ 5.820982] udevd[84]: starting version 2.1.1
[ 5.839500] random: udevd urandom read with 17 bits of entropy available
[ 6.901556] FAT-fs (mmcblk0p1): 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...
[ 7.781337] net eth0: initializing cpsw version 1.12 (0)
[ 7.867393] 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。

3.9.2 NAND Flash 更新/恢复

NAND 启动映像的更新需要借助于 u-boot 来完成。不管 NAND Flash 是否有数据，都可以利用 u-boot 对 NAND Flash 更新映像。

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**, **zImage**, **ramdisk.img**, **rootfs.tar.xz**, **logo.bmp**, **uEnv.txt** 映像文件拷贝到 TF 卡中。

2) 更新

- a) 将带有系统映象的 TF 卡插入开发板，用跳线帽短接 **JP5** 引脚，上电启动，不用进入 u-boot 命令行。

```
U-Boot 2015.07-ga66ad5e-dirty (Jun 19 2017 - 14:48:34 +0800)

Watchdog enabled
I2C: ready
DRAM: 512 MiB
NAND: 512 MiB
MMC: OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - bad CRC, using default environment

Net: <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot: 0
WARNING: Could not determine device tree to use
switch to partitions #0, OK
mmc0 is current device
SD/MMC found on device 0
reading boot.scr
** Unable to read file boot.scr **
reading uEnv.txt
199 bytes read in 5 ms (38.1 KiB/s)
Loaded env from uEnv.txt
Importing environment from mmc0 ...
Running uenvcmd ...
reading /embest_SBC_SBC8600.dtb
36786 bytes read in 11 ms (3.2 MiB/s)
reading /zImage
3537760 bytes read in 260 ms (13 MiB/s)
reading ramdisk.img
12034853 bytes read in 861 ms (13.3 MiB/s)
Booting from ramdisk ...
Kernel image @ 0x82000000 [ 0x0000000 - 0x35fb60 ]
## Loading init Ramdisk from Legacy Image at 88080000 ...

Image Name:
Created: 2017-04-24 9:02:19 UTC
Image Type: ARM Linux RAMDisk Image (gzip compressed)
Data Size: 12034789 Bytes = 11.5 MiB
Load Address: 23000000
Entry Point: 23000000
```

```
Verifying Checksum ... OK
## Flattened Device Tree blob at 88000000
Booting using the fdt blob at 0x88000000
Loading Ramdisk to 8f485000, end 8ffff2e5 ... OK
Loading Device Tree to 8f479000, end 8f484fb1 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
```

```
.....
[    5.589121] VFS: Mounted root (ext2 filesystem) on device 1:0.
[    5.595856] devtmpfs: mounted
[    5.599582] Freeing unused kernel memory: 268K (c090b000 - c094e000)
[    5.655671] EXT4-fs (ram0): re-mounted. Opts: (null)

Starting logging: OK
Populating /dev using udev: [    5.829716] udevd[85]: starting version 2.1.1
[    5.835775] random: udevd urandom read with 17 bits of entropy available
[    6.892135] FAT-fs (mmcblk0p1): 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...
[    7.772444] net eth0: initializing cpsw version 1.12 (0)
[    7.857384] net eth0: phy found : id is : 0x4dd072
ssh-keygen: generating new host keys: ED25519
Starting sshd: OK
running system update...
UPDATE FLAG  : TRUE
=====NANDFLASH UPDATE=====
Welcome to ARM
arm login: Writing MLO ...
Erasing 128 Kibyte @ 0 -- 100 % complete
Writing at 0x00000000
Writing MLO ...
Erasing 128 Kibyte @ 0 -- 100 % complete
Writing at 0x00000000
Writing MLO ...
```

```
Erasing 128 Kibyte @ 0 -- 100 % complete
Writing at 0x00000000
Writing MLO ...
Erasing 128 Kibyte @ 0 -- 100 % complete
Writing at 0x00000000
Writing u-boot.img ...
Erasing 128 Kibyte @ e0000 -- 100 % complete
Writing at 0x00000000
Writing at 0x00020000
Writing at 0x00040000
Erasing 128 Kibyte @ 0 -- 100 % complete
Erasing 128 Kibyte @ 0 -- 100 % complete
Writing logo.bmp ...
Erasing 128 Kibyte @ 3e0000 -- 100 % complete
Writing at 0x00000000
Writing at 0x00020000
Writing at 0x00040000
Writing at 0x00060000
Writing embest_SBC_SBC8600.dtb ...
Erasing 128 Kibyte @ 20000 -- 100 % complete
Writing at 0x00000000
Writing zImage ...
Erasing 128 Kibyte @ 7e0000 -- 100 % complete
Writing at 0x00000000
.....
Writing at 0x00340000
Writing rootfs.tar.xz ...
ls: /dev/ubi[0-9]: No such file or directory
Erasing 12[    10.857820] cpsw 4a100000.ethernet eth0: Link is Up - 100Mbps/Full - flow
control rx/tx
Eras[    18.699786] ubi0: attaching mtd10lete
Erasing 128 Kibyte @ f1e0000 -- 100 % complete
[    19.436363] ubi0: scanning is finished
[    19.446587] ubi0: empty MTD device detected
[    19.480742] ubi0: attached mtd10 (name "NAND.file-system", size 242 MiB)
[    19.490520] ubi0: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
[    19.498281] ubi0: min./max. I/O unit sizes: 2048/2048, sub-page size 512
[    19.505034] ubi0: VID header offset: 2048 (aligned 2048), data offset: 4096
[    19.516521] ubi0: good PEBs: 1936, bad PEBs: 0, corrupted PEBs: 0
[    19.530000] ubi0: user volume: 0, internal volumes: 1, max. volumes count: 128
[    19.537565] ubi0: max/mean erase counter: 0/0, WL threshold: 4096, image sequence
number: 3648342632
```

```
[ 19.546764] ubi0: available PEBs: 1852, total reserved PEBs: 84, PEBs reserved for  
bad PEB handling: 80  
[ 19.558880] ubi0: background thread "ubi_bgt0d" started, PID 578  
UBI device number 0, total 1936 LEBs (245825536 bytes, 234.4 MiB), available 1852 LEBs  
(235159552 bytes, 224.3 MiB), LEB size 126976 bytes (124.0 KiB)  
Set volume size to 235159552  
Volume ID 0, size 1852 LEBs (235159552 bytes, 224.3 MiB), LEB size 126976 bytes (124.0  
KiB), dynamic, name "rootfs", alignment 1  
[ 19.750310] UBIFS (ubi0:0): default file-system created  
[ 19.776923] UBIFS (ubi0:0): background thread "ubifs_bgt0_0" started, PID 589  
[ 19.878742] UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "rootfs"  
[ 19.906996] UBIFS (ubi0:0): LEB size: 126976 bytes (124 KiB), min./max. I/O unit sizes:  
2048 bytes/2048 bytes  
[ 19.937002] UBIFS (ubi0:0): FS size: 233635840 bytes (222 MiB, 1840 LEBs), journal  
size 11681792 bytes (11 MiB, 92 LEBs)  
[ 19.973269] UBIFS (ubi0:0): reserved for root: 4952683 bytes (4836 KiB)  
[ 19.987334] UBIFS (ubi0:0): media format: w4/r0 (latest is w4/r0), UUID  
A1EB0AA1-88B6-43D1-8B86-0CD553781279, small LPT model  
/media/mmcblk0p1/rootfs.tar.xz (1/1)  
[ 28.197424] random: nonblocking pool is initialized 0:08 50 s  
100 % 18.0 MiB / 71.2 MiB = 0.252 1.5 MiB/s 0:48  
/  
[ 76.686683] UBIFS (ubi0:0): un-mount UBI device 0  
[ 76.691654] UBIFS (ubi0:0): background thread "ubifs_bgt0_0" stops  
UPDATE : OK  
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 NAND Flash 启动 Linux 系统。

3) Uboot 参数设置

映像默认为 4.3 寸屏显示，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【**3.10 显示模式配置**】。

3.10 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 4.3 寸 LCD、7 寸 LCD、VGA、LVDS 和 9.7 寸电容屏显示模式进行配置。

注意:

 VGA、LVDS 和 9.7 寸电容屏暂不支持！

在开始配置前，需要首先进入 u-boot 模式。请重新启动开发套件，然后在系统提示倒数读秒时按下 PC 键盘上的任意键进入 u-boot 模式，如下表所示：

```
U-Boot SPL 2015.07-ga66ad5e-dirty (Jun 19 2017 - 14:48:34)
The Expected Linux image was not found. Please check your NAND configuration.
Trying to start u-boot now...

U-Boot 2015.07-ga66ad5e-dirty (Jun 19 2017 - 14:48:34 +0800)

Watchdog enabled
I2C: ready
DRAM: 512 MiB
NAND: 512 MiB
MMC: OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - bad CRC, using default environment

Net: <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot: 0
U-Boot# 0 (在这里按空格键进入 u-boot 命令行)
```

1) 使用 4.3"LCD 显示

在 u-boot 模式下执行以下命令来配置为 4.3 寸 LCD 显示模式：

- U-Boot# **setenv dispmode 4.3inch_LCD; saveenv; reset**

2) 使用 7"LCD 显示

在 u-boot 模式下执行以下命令来配置为 7 寸 LCD 显示模式：

- U-Boot# **setenv dispmode 7.0inch_LCD; saveenv; reset**

注意:

 清除 u-boot 环境变量的命令为： U-Boot# **run erase_env; reset**

3.11 测试和演示

本小节将对 SBC8600B 上的各个设备进行测试，并且会针对 Android 系统和 DVSDK 系统进行演示。

3.11.1 LED 测试

主板上的 D35 为系统心跳灯、D36 为用户 LED 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) 控制用户 LED 灯：

- `root@arm:~# echo 1 > /sys/class/leds/D36/brightness`
- `root@arm:~# echo 0 > /sys/class/leds/D36/brightness`

LED 灯会随着用户的操作进行亮灭。

3.11.2 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。

3.11.3 触摸屏测试

此测试要求 Linux 从 NAND Flash 启动：

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。

3.11.4 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，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，屏幕是黑的。

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。

3.11.5 ADC 测试

SBC8600B 带有 4 路 ADC 通道，用于测量外部模拟信号电压。

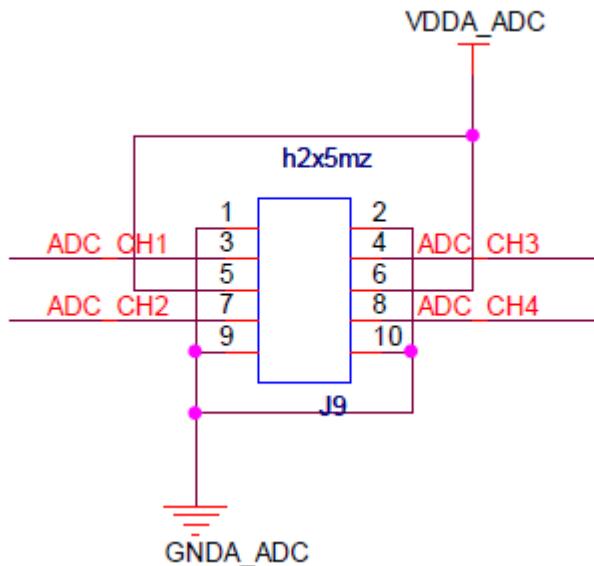


图 3-10 ADC 通道原理图

表 3-4 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：

Voltage Input = VDDA_ADC * (value read) / 4096

其中 VDDA_ADC 为 1.8V。

3.11.6 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 2017 年 6 月 30 日 12 点 10 分

```
root@arm:~# date -s "2017-06-30 12:10"
Fri Jun 30 12:10:00 UTC 2017
```

- 2) 把系统时钟写入 RTC

- root@arm:~# hwclock -w

- 3) 读取 RTC

```
root@arm:~# hwclock
Fri Jun 30 12:11:32 2017  0.000000 seconds
```

可以看到，硬件时钟 RTC 被设置成 2017 年 6 月 30 日，系统时钟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
Fri Jun 30 12:13:06 UTC 2017
```

可以看到，系统时间被恢复为硬件时间。

注意：

■ 开发板自身未带电池（型号 CR1220），用户需自行购买。

3.11.7 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  3.7G   34M  3.6G   1% /media/mmcblk0p1
```

- 2) 输入下述指令后，即可看到 TF 卡里面的内容：

```
root@arm:~# ls /media/mmcblk0p1
MLO           ramdisk.img      uEnv.txt
logo.bmp      u-boot.img
```

- 3) 手动卸载 TF 卡。

- root@arm:~# umount /media/mmcblk0p1

- 4) 手动挂载 TF 卡。

```
root@arm:~# mount -t vfat /dev/mmcblk0p1 /mnt
root@arm:~# df -h
Filesystem           Size   Used Available Use% Mounted on
...
/dev/mmcblk0p1        3.7G   34M     3.6G   1% /mnt
root@arm:~# ls /mnt
MLO           ramdisk.img      uEnv.txt
logo.bmp      u-boot.img
```

3.11.8 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，详细的安装方法请参考[附录二](#)。
- 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：

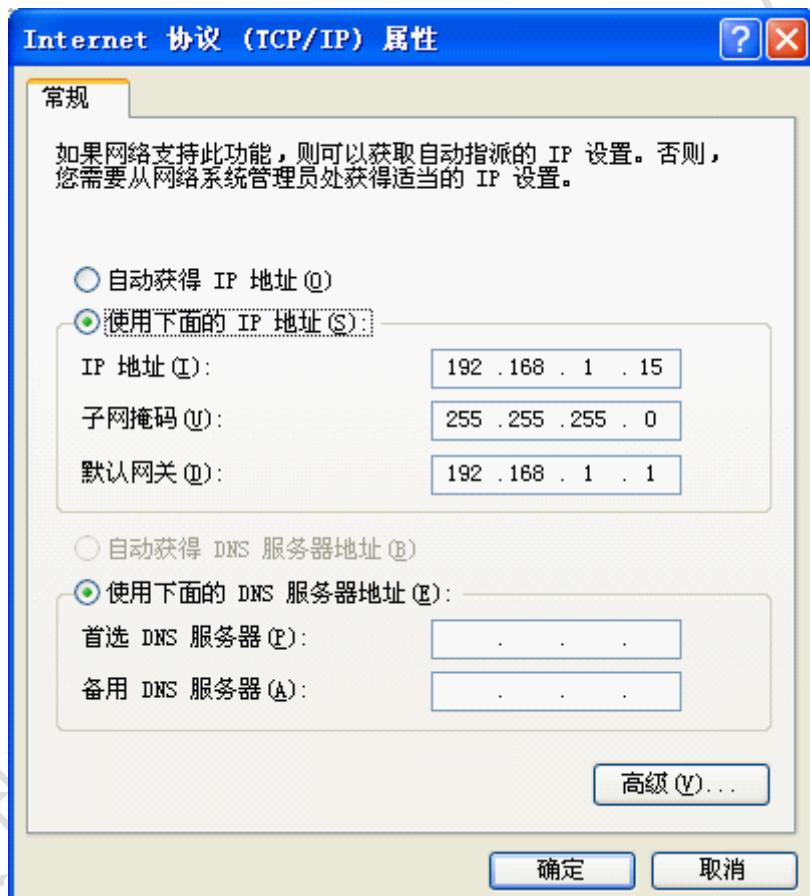


图 3-11 IP 配置

- 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。

3.11.9 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
```

3.11.10 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.....
```

3.11.11 网络测试

SBC8600B 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：

SBC8600B 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。

3.11.12 CAN 测试

SBC8600B 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 SBC8600B 的 CAN 接口和另一个 CAN 设备。

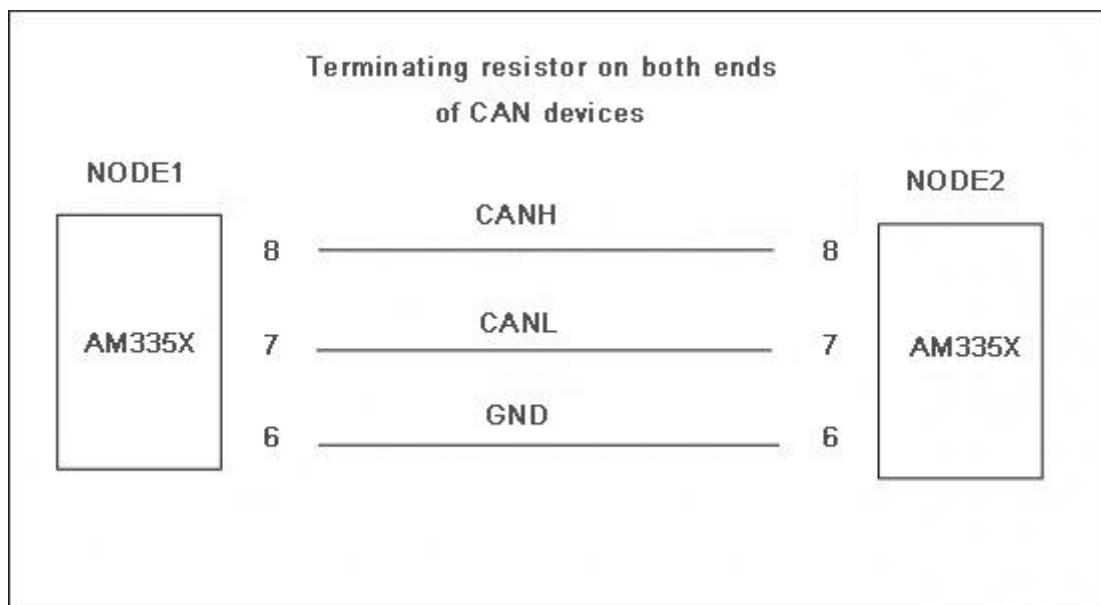


图 3-12 CAN 连接

测试方法如下：

- 1) 将 SBC8600B 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 SBC8600B 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：

- 该命令每次只发送一次数据，若再次发送数据，需重新输入该命令。
- 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)

1MKBPS (1000000)

以上的波特率均能正常通信，还有其它波特率可以设置，用户可以自己尝试，看能否通信。

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`。

3.11.13 RS485 测试

SBC8600B 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 SBC8600B 的 RS485 接口和另一个 RS485 设备。

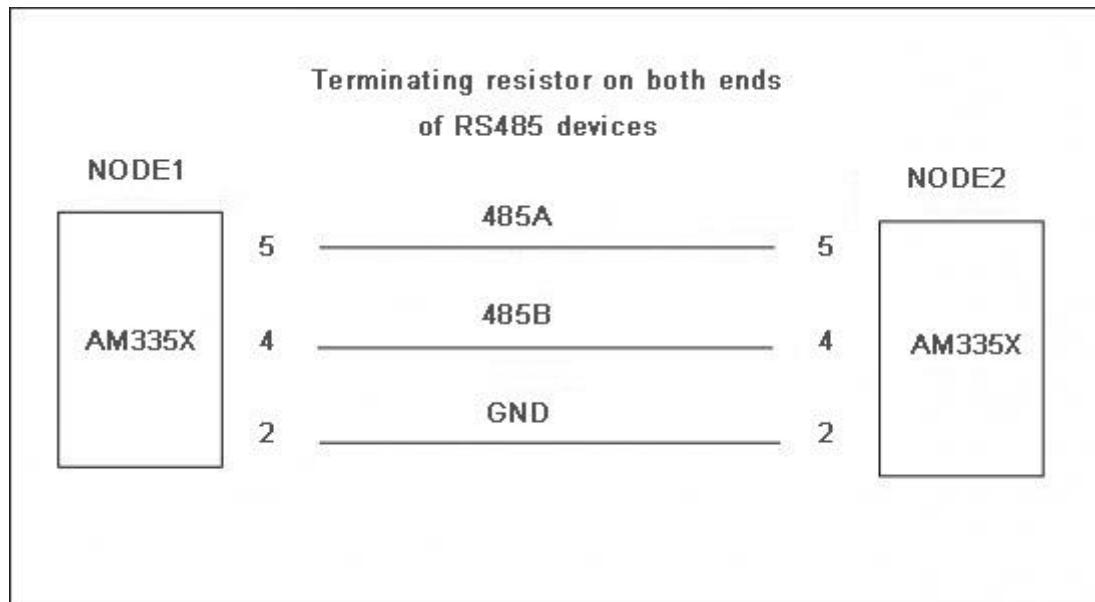


图 3-13 RS485 连接

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 linux\example\uart_test 下的 uart_test 到 TF 卡中，将 TF 卡插入 SBC8600B 的 TF 卡座子，在终端下运行如下命令：

```
root@arm:~# /embest/uart_test -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
```

3.11.14 串口测试

短接板上 J5 接口的 RX3V3 和 TX3V3 管脚，拷贝 linux\example\uart_test 下的 uart_test

文件到 TF 卡中，将 TF 卡插入 SBC8600B 的 TF 卡座子，在 linux 终端输入如下命令：

- `root@arm:~# /embest/uart_test -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
```

SBC8600B 上串口 3、串口 4 和串口 5 (J6 和 J7) 的测试方法同上。

3.11.15 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`

3.11.16 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
```

3.11.17 GPIO 测试

测试 J7 上的 GPIO0_19 和 GPIO2_0。

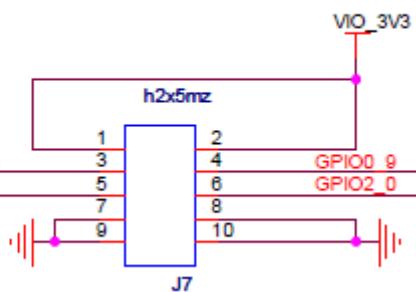


图 3-14 GPIO 接口

注意：

书 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。

3.11.18 Debian 配置

【版本】 ARM Debian 8

1) 网络配置

表 3-5 网络配置

配置文件	<code>/etc/network/interfaces</code>
DHCP	<code>iface eth0 inet dhcp</code>
Static	<code>iface eth0 inet static hwaddress ether d6:6e:0d:6e:2e:6d address 192.168.1.210 netmask 255.255.255.0</code>

手动 DHCP:

- `root@arm:~# dhclient -v eth0`

手动配置静态 IP:

- root@arm:~# ifconfig eth0 192.168.1.210

注意：

囧 因非联网状态下，开启 DHCP 会显著增加启动时间，故默认禁用 DHCP。若需启用，请编辑网络配置文件自行修改。

2) 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

3) 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 /etc/systemd/system/user.service

```
[Unit]
Description=user program
Documentation=user
After=network.target remote-fs.target

[Service]
Type=forking
PIDFile=/run/user.pid
ExecStart=/home/userPrgram
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s QUIT $MAINPID
PrivateTmp=true

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

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

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

无报错则可重启系统测试是否生效。

4) 禁止 LCD 显示闪烁光标

系统启动后， 默认情况下 LCD 中会显示当前光标位置并闪烁。若用户觉得光标影响到应用程序或用户体验， 可使用下列命令禁止光标：

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

恢复：

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

3.11.19 TISDK 系统演示

1) 准备 TF 启动卡

在 SBC8600B 评估板上启动 TISDK 系统需要借助 TF 卡， 目前有两种方法可以制作 TF 启动卡

A. 使用命令行制作 TF 启动卡

将 TF 卡格式化为两个分区（请参考制作 Linux 启动盘）， 并通过读卡器连接到 PC 上， 然后在 Ubuntu Linux 系统中执行以下命令；

- `cp /media/cdrom/linux/demo/tisdk/image/MLO /media/LABEL1`
- `cp /media/cdrom/linux/demo/tisdk/image/u-boot.img /media/LABEL1`
- `cp /media/cdrom/linux/demo/tisdk/image/zImage /media/LABEL1/`
- `rm -rf /media/LABEL2/*`
- `sudo tar xvf`
- `/media/cdrom/linux/demo/tisdk/image/tisdk-rootfs-am335x-evm.tar.gz -C /media/LABEL2`
- `sync`
- `umount /media/LABEL1`
- `umount /media/LABEL2`

B. 利用 Win32DiskImager.exe 制作 TF 启动卡

使用 MicroSD 卡卡套或者 USB 读卡器将 MicroSD 卡连接到 PC，在 windows 下解压 CD\linux\tools 目录下的 `win32diskimager-v0.7-binary.zip` 后， 运行

win32diskimager-v0.7-binary 文件夹下的软件 Win32DiskImager.exe，注意该软件运行约两分钟后才在桌面上显示出来，如下图：

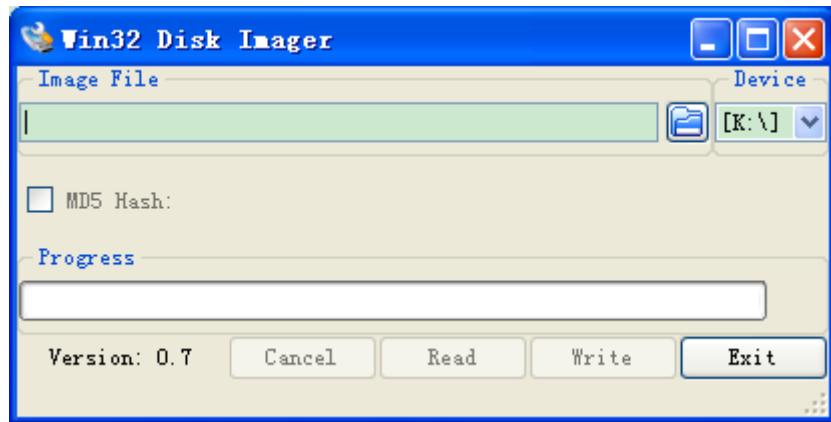


图 3-15 Win32 Disk Imager Tool

点击图标  打开并选中光盘资料下的映像文件

linux/demo/tisdk/image/am335x-tisdk.img。

并根据 TF 卡被识别的盘符在下接列表框  进行设置



图 3-16 Win32 Disk Imager Tool设置

最后点击 Write 开始烧写，烧写完成后会出现“Write Successful”对话框，点击“OK”按钮即可完成启动卡的制作。



图 3-17 启动卡制作完成

- 2) 在进行上述操作后, 用跳线帽短接开发板上的 **JP5** 引脚。将 TF 卡插入 SBC8600B 卡槽, 上电启动, 系统启动串口信息如下: (黑体字为需要输入的字符内容)

```
CCCCCCCC
U-Boot SPL 2011.09-svn55 (Dec 04 2012 - 09:33:23)
Texas Instruments Revision detection unimplemented
Booting from MMC...
OMAP SD/MMC: 0
reading u-boot.img
reading u-boot.img

U-Boot 2011.09-svn55 (Dec 04 2012 - 09:33:23)

I2C: ready
DRAM: 512 MiB
WARNING: Caches not enabled
Did not find a recognized configuration, assuming General purpose EVM in Profile 0 with
Daughter board
NAND: HW ECC Hamming Code selected
512 MiB
MMC: OMAP SD/MMC: 0
*** Warning - bad CRC, using default environment

Net: cpsw
Hit any key to stop autoboot: 0
Booting from dv-sdk ...
reading zImage

3175384 bytes read
## Booting kernel from Legacy Image at 80007fc0 ...
Image Name: Linux-3.2.0
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 3175320 Bytes = 3 MiB
Load Address: 80008000
```

```
Entry Point: 80008000
Verifying Checksum ... OK
XIP Kernel Image ... OK
OK
Starting kernel ...
..... //中间部分省略
Arago Project http://arago-project.org am335x-evm ttyO0

Arago 2011.09 am335x-evm ttyO0

am335x-evm login: root //输入用户名 root 进入系统
```

3) TISDK 文件系统中带有一些预装的应用程序，基于 QT 来实现，完全图形化操作，用户可以轻松的执行里面的演示程序。

注意：

■ 系统默认支持 4.3 寸屏幕，如果需要修改显示模式，请参考 3.10 显示模式配置章节的内容。

■ u-boot 源码下 include/configs/文件夹中，包含 sbc8600_tisdk.h 的配置文件。

用户要编译 tisdk 的 u-boot。输入命令如下：

```
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- sbc8600_tisdk_config
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

3.12 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。

3.12.1 LED 应用程序开发示例

1) 编写代码

led_acc.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_acc.c -o led_acc**

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_acc 文件所在的目录，输入下面命令回车 led_acc 即在后台运行。

- **./led_acc &**

3.12.2 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节序。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 describing 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_recv(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 linux\example\can_test.tar.bz2 解压到 work 目录并编译。

- **cd \$HOME/work**
- **tar xvf /media/cdrom/linux/example/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“3.11.11 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网址：
<http://blog.csdn.net/zhangxiaopeng0829/article/details/7646639>。

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
```

3.12.3 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 3-6 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 3-7 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; /* Disnable 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; /* Disnable parity checking */
        break;
    case 'S':
    case 's': /*as no parity*/
        break;
    }
```

```
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：

如果我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 linux\example\ 目录下的 uart_test.tar.bz2 文件复制到\work 目录下，并进行编译；

- **cd \$HOME/work**
- **tar xvf /media/cdrom/linux/example/uart_test.tar.bz2**
- **cd uart_test**
- **make**

注意：

关于串行接口的测试请参考 3.11.14 串口测试章节的内容。

第4章 Android 系统

4.1 开发环境

4.1.1 获取 android 源代码

1) 执行以下命令来获取 repo 源代码;

- `$ mkdir ~/bin`
- `$ curl https://raw.githubusercontent.com/android/tools_repo/master/repo > ~/bin/repo`
- `$ chmod a+x ~/bin/repo`
- `$ export PATH=~/bin:$PATH`

2) 执行以下命令来初始化 repo 源代码;

- `$ mkdir ~/TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1`
- `$ cd ~/TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1`
- `$ repo init --repo-url=https://github.com/android/tools_repo.git -u https://github.com/embest-tech/rowboat-manifest.git -m TIOP-TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1.xml`

3) 执行以下命令来同步 repo 源代码;

- `$ cd ~/TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1`
- `$ repo sync`

4.1.2 编译启动代码

1) 执行以下命令来编译 Android 的 bootloader 文件;

- `cd TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1/u-boot`
- `make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- sbc8600_android_config`
- `make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-`

执行完成后在当前目录下会生成启动映像文件 MLO 和 u-boot.img。

2) 执行下面指令生成 Android 内核和 ubi 文件系统:

- `cd TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1/`
- `export PATH=$HOME/TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin:$PATH`
- `export PATH=$HOME/tools:$PATH`
- `make TARGET_PRODUCT=sbc8600 sgx_clean kernel_clean clean`
- `make TARGET_PRODUCT=sbc8600 OMAPES=4.x`
- `source ./build_ubi.sh sbc8600`

执行完成后在 `TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1/kernel/arch/arm/boot/` 目录下会生成内核映像文件 **uImage**。

在 `TI-Android-ICS-4.0.3-DevKit-EVM-SK-3.0.1/temp/` 目录下会生成 Android 文件系统 **ubi.img**。

4.2 演示

SBC8600B 提供 Android 系统演示，使用方法如下。

- 3) 拷贝 `CD\android\image` 目录下所有文件到 TF 卡。
- 4) 将 TF 卡放入开发板，用跳线帽短接 SBC8600B 的 **JP5** 引脚，上电启动，超级终端将会显示下述信息：

注意:

如客户需要从 TF 卡启动，而不想短接 JP5，可以在 uboot 下输入命令擦除 nand flash 中的映像，重启后将会从 TF 卡启动。

```
CCCCCCCC  
U-Boot SPL 2011.09-svn55 (Dec 04 2012 - 09:36:25)  
Texas Instruments Revision detection unimplemented  
Booting from MMC...  
OMAP SD/MMC: 0  
reading u-boot.img  
reading u-boot.img
```

```
U-Boot 2011.09-svn55 (Nov 22 2012 - 11:35:28)

I2C: ready
DRAM: 512 MiB
WARNING: Caches not enabled
Did not find a recognized configuration, assuming General purpose EVM in Profile 0 with
Daughter board
NAND: HW ECC Hamming Code selected
512 MiB
MMC: OMAP SD/MMC: 0
*** Warning - bad CRC, using default environment

NAND erase.chip: device 0 whole chip
Skipping bad block at 0x03620000
Erasing at 0x1ffe0000 -- 100% complete.
OK
reading MLO

36079 bytes read
HW ECC BCH8 Selected

NAND write: device 0 offset 0x0, size 0x8cef
36079 bytes written: OK
reading flash-uboot.img

234620 bytes read
HW ECC BCH8 Selected

NAND write: device 0 offset 0x80000, size 0x3947c
234620 bytes written: OK
reading zImage

2719416 bytes read
HW ECC BCH8 Selected

NAND write: device 0 offset 0x280000, size 0x297eb8
2719416 bytes written: OK
reading ubi.img

72744960 bytes read
```

```
SW ECC selected  
  
NAND write: device 0 offset 0x780000, size 0x4560000  
72744960 bytes written: OK
```

- 5) 烧写完成后，板上 led 灯会闪烁提示，请拔掉 TF 卡和跳线帽。
- 6) 重启开发板，即可进入 android 操作系统。
- 7) 映像默认为 4.3 寸屏显示，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【3.10 显示模式配置】。

注意：

- 书 当 VGA 显示时，需在 uboot 下输入以下命令禁用 android 触摸屏校准程序。
- 书 SBC8600# **setenv calibration 0**

4.3 测试

1) USB OTG 测试

接上 OTG 设备(鼠标、键盘等)到 OTG 接口，启动系统，观察 OTG 设备是否能够正常使用。

2) 休眠唤醒测试

启动系统，依次点击 setting -> developer options -> stay awake，去掉此处的勾选，然后设置依次点击 setting -> developer options -> display 设置休眠时间(不设置则默认 min)，等待一段时间，观察屏幕是否会熄灭，如果是，稍后点击屏幕观察是否亮屏。

提示：此处休眠唤醒会出现随机死机问题，暂时无法解决，如果休眠后无法唤醒请重启后再次测试，另外测试次数越多，随机死机问题出现概率越大。

3) 有线网测试

进入网络设置，选择网卡 eth0，填充好 ip、route 等，其他的可保持默认值，但是不可为空，稍等一会，测试网络是否连通。

第5章 Windows Embedded Compact 7 操作系统

本章节将简要介绍产品光盘所提供的软件资源，并且会详细讲解 WinCE 系统开发、系统更新和应用程序接口等内容。

5.1 软件资源

DVD 光盘中包含了 BSP 软件包、工程范例、应用范例和预编译映像等资源，可以让您轻松快速地使用 SBC8600B 开发套件来进行 WinCE 应用和系统的开发。

5.1.1 软件资源的位置

您可以通过以下表格在产品附带的光盘中找到相应的软件资源：

表 5-1 WinCE 软件资源

类别	位置
BSP	CD\WINCE700\BSP\SBC8600.rar
	CD\WINCE700\BSP\COMMON_TI_V1.rar
	CD\WINCE700\BSP\3rdParty.rar
	CD\WINCE700\BSP\PowerVR.rar
工程范例	CD\WINCE700\project\SBC8600
应用范例	CD\WINCE700\app\
预编译映像	CD\WINCE700\Image\

5.1.2 预编译映像和 BSP

光盘提供的预编译映像总共包含五个映像文件，分别是 MLO、xldrnand.nb0、Ebootsd.nb0、Ebootnd.nb0 和 NK.bin，它们的功能请参考下表中的内容；

表 5-2 预编译映像

映像	注释
MLO	用于 TF 卡启动的一级引导程序
xldrnand.nb0	用于 NAND Flash 启动的一级引导程序

Ebootsd.nb0	用于 TF 卡启动的二级引导程序
Ebootnd.nb0	用于 NAND Flash 启动的二级引导程序
NK.bin	WinCE 内核和文件系统

下方表格列出了 BSP 软件包所包含的内容以及文件格式：

表 5-3 BSP 软件包

类别	名称	文件格式
X-Loader (First boot loader)	NAND	源代码
	SD	源代码
EBOOT (Second boot loader)	NAND	源代码
	SD	源代码
OAL	Boot parameter	源代码
	KILT(EMAC)	源代码
	Serial debug	源代码
	REBOOT	源代码
	Watchdog	源代码
	RTC	源代码
	Kernel profiler	源代码
	System timer	源代码
	Interrupt controller	源代码
	MMU	源代码
Driver	NLED driver	源代码
	GPIO/I2C/SPI/MCASP driver	源代码
	Serial port driver	源代码
	Audio driver	源代码
	NAND driver	源代码
	Display driver	源代码
	TOUCH driver	源代码
	SD/MMC/SDIO driver	源代码
	EMAC driver	Source
	USB OTG driver	源代码
	GPIO keyboard driver	源代码
	DMA driver	源代码
	Backlight driver	源代码
	Battery driver	源代码
	PRU driver	源代码
SDK	powerVR DDK & SDK	二进制文件 & 源代码

5.2 系统开发

5.2.1 集成开发环境安装

请按照下面步骤将集成开发环境安装到 windows XP:

- 1) Visual Studio 2008
- 2) Visual Studio 2008 SP1
- 3) Windows Embedded Compact 7
- 4) Windows Embedded Compact 7 Updates
- 5) ActiveSync 4.5

注意:

默认光盘没有提供 Windows Embedded Compact 7 集成开发环境, 请自行参考下载:
<http://www.microsoft.com/download/en/default.aspx>。

5.2.2 提取 BSP 及样例工程文件到集成开发环境

请按照下面步骤进行:

- 1) 解压[CD\WINCE700\BSP\SBC8600.rar] 到 [C:\WINCE700\PLATFORM] 目录下。
- 2) 解压[CD\WINCE700\BSP\COMMON_TI_V1.rar]到 [C:\WINCE700\PLATFORM\COMMON\SRC\SOC].
- 3) 解压[CD\WINCE700\BSP\3rdParty.rar] 到[C:\WINCE700].
- 4) 解压[CD\WINCE700\BSP\powerVR.rar] 到[C:\WINCE700\public].
- 5) 将[CD\WINCE700\project\SBC8600] 拷贝到 [C:\WINCE700\OSDesigns] 目录下。

注意:

本文的 Windows Embedded Compact 7 默认安装路径是[C:\WINCE700]..

5.2.3 Sysgen & BSP 编译

下面是 Sysgen 和 BSP 编译的详细步骤

- 1) 打开[C:\WINCE700\OSDesigns\SBC8600]下的工程文件 SBC8600.sln。
- 2) 在 VS2008 窗口选择[Build-> Build Solution]，开始 sysgen 和 build BSP。
- 3) 在 sysgen 阶段和编译阶段成功完成后，[C:\WINCE700\OSDesigns\SBC8600\SBC8600\RelDir\SBC8600_ARMV7_Release]目录下会生成映像文件 MLO、EBOOTSD.nb0 和 NK.bin，将 MLO、EBOOTSD.nb0 和 NK.bin 文件拷贝到 TF 卡中。
- 4) 接入 TF 卡，用跳线帽短接 **JP5**，上电启动 SBC8600B。

5.2.4 驱动介绍

BSP 的所有驱动源码路径：

表 5-4 驱动源码路径

驱动名称	驱动位置
NLED driver	BSP\SBC8600\SRC\DRIVERS\NLED
GPIO	BSP\SBC8600\SRC\DRIVERS\GPIO BSP\COMMON_TI_V1\COMMON_TI_AMXX\GPIO
I2C	BSP\COMMON_TI_V1\COMMON_TI_AMXX\OAL\OALI2C
SPI	BSP\COMMON_TI_V1\COMMON_TI_AMXX\SPI BSP\SBC8600\SRC\DRIVERS\MCSPI
MCASP driver	BSP\COMMON_TI_V1\COMMON_TI_AMXX\MCASP
Serial port driver	BSP\COMMON_TI_V1\COMMON_TI_AMXX\SERIAL BSP\SBC8600\SRC\DRIVERS\UART
Audio driver	BSP\SBC8600\SRC\DRIVERS\WAVEDEV2
NAND driver	BSP\SBC8600\SRC\DRIVERS\BLOCK BSP\COMMON_TI_V1\COMMON_TI_AMXX\BLOCK
Display driver	BSP\COMMON_TI_V1\COMMON_TI_AMXX\DSS_Netra BSP\SBC8600\SRC\DRIVERS\DISPLAY
TOUCH driver	BSP\SBC8600\SRC\DRIVERS\TOUCH
SD/MMC/SDIO driver	BSP\SBC8600\SRC\DRIVERS\SDHC BSP\COMMON_TI_V1\COMMON_TI_AMXX\SDHC

	BSP\COMMON_TI_V1\COMMON_TI\SDHC
EMAC driver	BSP\COMMON_TI_V1\AM33X\CPSW3Gminiport BSP\SBC8600\SRC\DRIVERS\EMAC
USB OTG driver	BSP\SBC8600\SRC\DRIVERS\USB BSP\COMMON_TI_V1\AM33X\USB
GPIO keyboard driver	BSP\SBC8600\SRC\DRIVERS\KEYPAD
Backlight driver	BSP\SBC8600\SRC\DRIVERS\BACKLIGHT
Battery driver	BSP\SBC8600\SRC\DRIVERS\BATTERY
PRU driver	BSP\COMMON_TI_V1\AM33X\PRU BSP\SBC8600\SRC\DRIVERS\PRU
DMA driver	BSP\SBC8600\SRC\DRIVERS\EDMA BSP\COMMON_TI_V1\COMMON_TI_AMXX\EDMA

假若用户想要参考更多的 Windows Embedded Compact 7 驱动开发，请参考 PB7.0 的参考文档，打开的方法如下：（在电脑端操作）

开始->

所有程序->

Microsoft Visual Studio 2008->

Microsoft Visual Studio 2008 Document->

Content(C) ->

Windows Embedded Compact 7->Device Driver.

5.3 系统映像更新

SBC8600B 支持 TF 卡与 NAND 启动，本章会针对两种不同的系统更新方式进行介绍。

5.3.1 TF 卡映像更新

1) TF 卡格式化

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

软件下载链接：<http://dl.vmall.com/c0tf7n4bz9>

- a) 把 MMC/SD 卡插入 PC 下读卡器中
- b) 打开 HP USB Disk Storage Format Tool，出现类似提示如下：

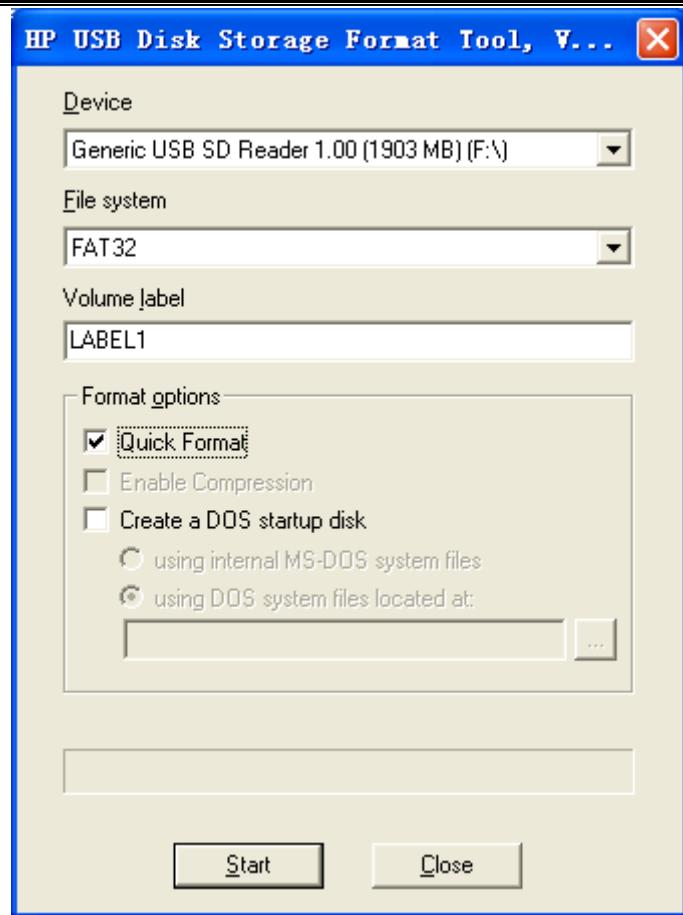


图 5-1 TF卡格式化

- 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
- ❑ HP USB Disk Storage Format Tool 格式化 TF 卡时将清除 TF 存储卡的分区。

2) 拷贝内核映像

- 将 CD\WINCE700\image 目录下的 **MLO**、**EBOOTSD.nb0** 和 **NK.bin** 映像文件拷贝到 TF 卡中。

3) 启动系统

插入 TF 卡，用跳线帽短接 JP5，重启系统，这时系统从 TF 卡启动，在数秒时按空格健进入 eboot 菜单选择启动设备和显示方式，具体步骤如下：

a) 进入 EBOOT 菜单

```
CCCCCCCC
Texas Instruments Windows CE SD X-Loader33X
Built Jul 27 2012 at 11:25:59
Version BSP_WINCE_ARM_A8 02.30.00.03
open ebootsd.nb0 file
Init HW: controller RST
SDCARD: requested speed 1000000, actual speed 1000000
SDCARD: requested speed 25000000, actual speed 19200000
read ebootsd.nb0 file

jumping to ebootsd image

Microsoft Windows CE Bootloader Common Library Version 1.4 Built Jul 27 2012 11:23:05
I2C EEPROM returned wrong magic value 0xffffffff
INFO:OALLogSetZones: dpCurSettings.ulZoneMask: 0x8409

Texas Instruments Windows CE EBOOT for AM33x, Built Jul 27 2012 at 11:25:53
EBOOT Version 0.0.1, BSP BSP_WINCE_ARM_A8 02.30.00.03
AHCLKX pinmux:0
AHCLKX CTRL:0x8001
pin function:0x0
pin dir:0x8000000

TI AM33X

ecc type:3
System ready!
Preparing for download...
INFO: Predownload....
Checking bootloader blocks are marked as reserved (Num = 18)

BOOT_CFG_SIGNATURE is different, read -1, expect 1111705159
WARN: Boot config wasn't found, using defaults
INFO: SW3 boot setting: 0x04
IsValidMBR: MBR sector = 0x480 (valid MBR)
OpenPartition: Partition Exists=0x1 for part 0x20.

>>> Forcing cold boot (non-persistent registry and other data will be wiped) <<<
```

```
e0311800 56e4 -> 0 18 31 e0 e4 56  
e0311800 57e4 -> 0 18 31 e0 e4 57  
Hit space to enter configuration menu [56] 5... (在此处按空格键进入 EBOOT 菜单)
```

b) 按[2]->[2]选择从 TF 卡启动

```
Main Menu  
-----  
[1] Show Current Settings  
[2] Select Boot Device  
[3] Select KITL (Debug) Device  
[4] Network Settings  
[5] SDCard Settings  
[6] Set Device ID  
[7] Save Settings  
[8] Flash Management  
[9] Enable/Disable OAL Retail Messages  
[a] Select Display Resolution  
[b] Select OPP Mode  
[0] Exit and Continue
```

Selection: 2

```
Select Boot Device  
-----
```

```
[1] Internal EMAC  
[2] NK from SDCard FILE  
[3] NK from NAND  
[0] Exit and Continue
```

Selection (actual Internal EMAC): 2

Boot device set to NK from SDCard FILE

c) 按[a]进入“Select Display Resolution”菜单并选择 LCD\LVDS 输出模式

```
Main Menu  
-----  
[1] Show Current Settings  
[2] Select Boot Device  
[3] Select KITL (Debug) Device  
[4] Network Settings  
[5] SDCard Settings
```

```
[6] Set Device ID
[7] Save Settings
[8] Flash Management
[9] Enable/Disable OAL Retail Messages
[a] Select Display Resolution
[b] Select OPP Mode
[0] Exit and Continue

Selection: a

-----
Select Display Resolution

[1] LCD 480x272 60Hz          //For 4.3-inch LCD
[2] DVI  640x480 60Hz(N/A)
[3] DVI  640x480 72Hz(N/A)
[4] LCD  800x480 60Hz          //For 7-inch LCD
[5] DVI  800x600 60Hz(N/A)      //For LVDS
[6] DVI  800x600 56Hz(N/A)
[7] VGA  1024x768 60Hz         //For VGA
[8] DVI  1280x720 60Hz(N/A)
[0] Exit and Continue Selection (actual LCD 480x272 60Hz): 4
```

d) 输入[0]继续启动

```
Main Menu

[1] Show Current Settings
[2] Select Boot Device
[3] Select KITL (Debug) Device
[4] Network Settings
[5] SDCard Settings
[6] Set Device ID
[7] Save Settings
[8] Flash Management
[9] Enable/Disable OAL Retail Messages
[a] Select Display Resolution
[b] Select OPP Mode
[0] Exit and Continue

Selection: 0
mode = 3
LcdPdd_LCD_GetMode:3
```

```
mode = 3
LcdPdd_LCD_Initialize:3
OEMPreDownload: Filename nk.bin
Init HW: controller RST
SDCARD: requested speed 1000000, actual speed 1000000
SDCARD: requested speed 25000000, actual speed 19200000

BL_IMAGE_TYPE_BIN

+OEMMultiBinNotify(0x8feb24d8 -> 1)
Download file information:
-----
[0]: Address=0x80002000 Length=0x03c9e9bc Save=0x80002000
-----
Download file type: 1
+OEMIsFlashAddr(0x80002000) g_eboot.type 1
-----
.....
.....rom_offset=0x0.
..ImageStart = 0x80002000, ImageLength = 0x3c9e9bc, LaunchAddr = 0x8000b6a0

Completed file(s):
-----
+OEMIsFlashAddr(0x80002000) g_eboot.type 1
[0]: Address=0x80002000 Length=0x3c9e9bc Name="" Target=RAM
ROMHDR at Address 80002044h
Launch Windows CE image by jumping to 0x8000b6a0...

Windows CE Kernel for ARM (Thumb Enabled)
CPU CP15 Control Register = 0xc5387f
CPU CP15 Auxiliary Control Register = 0x42
I2C EEPROM returned wrong magic value 0xffffffff
+OALTimerInit(1, 24000, 200)
--- High Performance Frequency is 24 MHz---
```

5.3.2 NAND Flash 映像更新

1) 格式化 TF 卡

请参考 4.3.1 TF 卡系统映像更新中的 TF 卡格式化部分。

2) 拷贝映像文件

- 将 CD\WINCE700\image 目录下的 **MLO**、**EBOOTND.nb0**、**NK.bin**、

XLDRNAND.nb0 和 EBOOTSD.nb0 映像文件拷贝至 TF 卡中。

3) 更新映像文件

插入 TF 卡，用跳线帽短接 JP5，重新启动系统，这时系统从 TF 卡启动。超级终端输出启动信息，按[SPACE]进入 EBOOT 菜单，按以下步骤更新 NAND Flash 映像：

- 按[8] 进入 Flash 管理菜单。
- 分别按[9]->[4]->[A]、[9]->[3]->[B] 和[9]->[2]->[C] 写 XLDR、EBOOT 和 NK 映像。
- 然后按[0]键回到主菜单，并分别按下[2]、[3]选择从 NAND Flash 启动，按[A] 选择 LCD、LVDS 或 VGA 输出模式，按[7]和[y]保存启动设置。

拔除 TF 卡和 JP5 上的跳线帽，重新启动系统，这时系统将从 NAND Flash 启动。

5.4 使用说明

5.5 应用开发

本章介绍如何在 SBC8600B 进行 Windows Embedded Compact 7 应用程序开发

5.5.1 如何使用 openGL ES demo

1) 把 vs2008 里的 catalog items view 里把 PowerVR 选上，如下图所示：

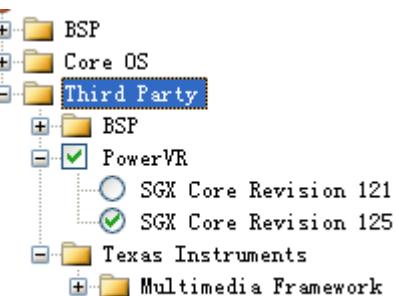


图 5-2 Catalog Item

- 2) 在 VS2008 菜单选择[Build-> Build Solution]，在 sysgen 和编译 BSP 完成后，用新生成的 nk.bin 代替 TF 卡中的 nk.bin。
- 3) 拷贝 C:\WINCE700\PUBLIC\PowerVR\oak\target\Rev125\ARMV4I\retail*.exe 到 SBC8600B windows embedded compact 7 系统，双击 demo 进行测试。

5.5.2 应用程序接口与示例

SBC8600B 应用程序开发所用到的 API 均采用微软 Windows Embedded Compact 7 标准应用程序接口定义，SBC8600B 仅在标准 API 基础上扩展了 GPIO 的接口定义，控制 GPIO PIN 的状态的应用程序请参照 CD\WINCE700\app\GPIOAppDemo。

Windows Embedded Compact 7 标准应用程序接口定义可以查看 MSDN Windows Embedded Compact 7 API 相关帮助文档。

5.5.3 GPIO 应用程序接口与示例

GPIO 应用程序接口与示例

表 5-5 IOCTL 码描述

IOCTL 码	描述
IOCTL_GPIO_SETBIT	Set GPIO pin as 1
IOCTL_GPIO_CLRBIT	Set GPIO pin as 0
IOCTL_GPIO_GETBIT	Read GPIO pin
IOCTL_GPIO_SETMODE	Set the working mode of GPIO pin
IOCTL_GPIO_GETMODE	Read the working mode of GPIO pin
IOCTL_GPIO_GETIRQ	Read the corresponding IRQ of GPIO pin

操作示例如下：

1) 打开 GPIO 设备

```
HANDLE hFile = CreateFile (_T ("GIO1:"), (GENERIC_READ|GENERIC_WRITE),
(FILE_SHARE_READ|FILE_SHARE_WRITE), 0, OPEN_EXISTING, 0, 0);
```

2) 设置 GPIO 工作模式

```
DWORD id = 48, mode = GPIO_DIR_OUTPUT;
```

设置 GPIO 工作模式：

```
DWORD pInBuffer [2];
pInBuffer [0] = id;
pInBuffer [1] = mode;
DeviceIoControl (hFile, IOCTL_GPIO_SETMODE, pInBuffer, sizeof (pInBuffer), NULL, 0,
NULL, NULL);
```

读 GPIO 工作模式：

```
DeviceIoControl (hFile, IOCTL_GPIO_GETMODE, &id, sizeof(DWORD), &mode,
sizeof(DWORD), NULL, NULL);
```

"id"为 GPIO 引脚号, "mode"为 GPIO 模式定义, 包括:

表 5-6 GPIO模式

模式定义	描述
GPIO_DIR_OUTPUT	Output mode
GPIO_DIR_INPUT	Input mode
GPIO_INT_LOW_HIGH	Rising edge trigger mode
GPIO_INT_HIGH_LOW	Falling edge trigger mode
GPIO_INT_LOW	low level trigger mode
GPIO_INT_HIGH	high level trigger mode
GPIO_DEBOUNCE_ENABLE	Jumping trigger enable

3) GPIO 引脚操作

```
DWORD id = 48, pinState = 0;
```

输出高电平:

```
DeviceIoControl (hFile, IOCTL_GPIO_SETBIT, &id, sizeof (DWORD), NULL, 0, NULL, NULL);
```

输出低电平:

```
DeviceIoControl (hFile, IOCTL_GPIO_CLRBIT, &id, sizeof (DWORD), NULL, 0, NULL, NULL);
```

读引脚状态

```
DeviceIoControl (hFile, IOCTL_GPIO_GETBIT, &id, sizeof (DWORD), &pinState, sizeof (DWORD), NULL, NULL);
```

"id"为 GPIO 引脚号, "pin"返回引脚状态。

4) 其它可选操作

读 GPIO 引脚对应的 IRQ 号:

```
DWORD id = 0, irq = 0;  
DeviceIoControl (hFile, IOCTL_GPIO_GETIRQ, &id, sizeof (DWORD), &irq, sizeof (DWORD), NULL, NULL);
```

"id"为 GPIO 引脚号, "irq"返回 IRQ 号。

5) 关闭 GPIO 设备

```
CloseHandle (hFile);
```

注意：

- GPIO 引脚定义：0~127 MPU Bank0~3 GPIO 引脚。
- GPIO 引脚 0~127 必须在 SBC8600/SRC/inc/bsp_padcfg.h 文件下被设置为 GPIO。

深圳市英蓓特科技有限公司

附录

附录一 安装 Ubuntu Linux 系统

我们都知道在开发软件之前需要安装嵌入式开发环境。而产品光盘中提供的开发环境(`linux/source`目录下)需要在Linux系统下才能运行。如果当前您使用的是Windows系统,那么首先需要安装Linux操作系统,然后才能在该系统下安装相应的开发环境。我们推荐使用VirtualBox虚拟机来在Windows中安装Ubuntu Linux操作系统。下面我们将依次介绍虚拟机VirtualBox和Ubuntu Linux系统的安装过程。

安装 VirtualBox 虚拟机

您可以访问<http://www.virtualbox.org/wiki/Downloads>来下载最新版本的VirtualBox虚拟机。在安装VirtualBox虚拟机之前,请确保您的PC拥有至少512MB的内存空间。建议提供1G以上的内存空间。

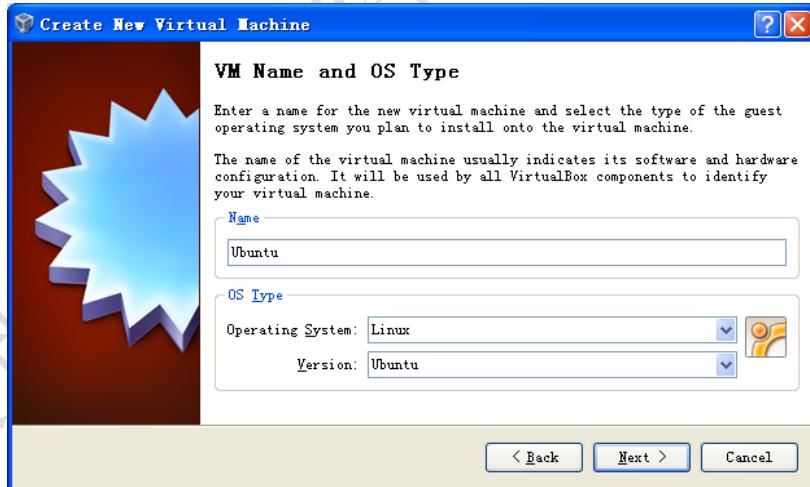
- 1) 安装过程很简单，此处略过。安装后从开始菜单启动 VirtualBox，然后单击程序窗口上方的 **New** 按钮，弹出新建虚拟机窗口；



附图1 新建虚拟机窗口

单击 **Next** 按钮开始新建虚拟机。

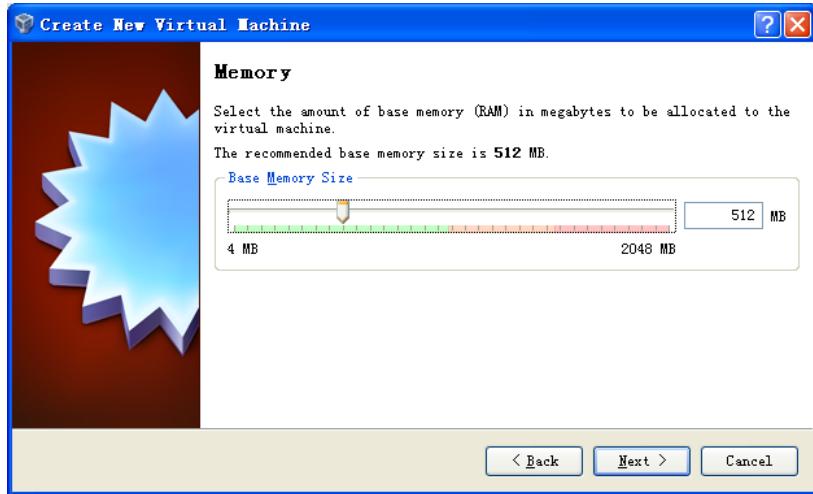
- 2) 在下方窗口中为新建的虚拟机指定名称和操作系统类型；



附图2 虚拟机名称和操作系统类型

您可以在 **Name** 一栏中输入新建虚拟机的名称，例如 **Ubuntu**。在 **Operating System** 一栏中选择 **Linux**，然后单击 **Next** 按钮。

- 3) 在以下窗口中为虚拟机分配适当大小的内存空间，分配完成后单击 **Next** 按钮；

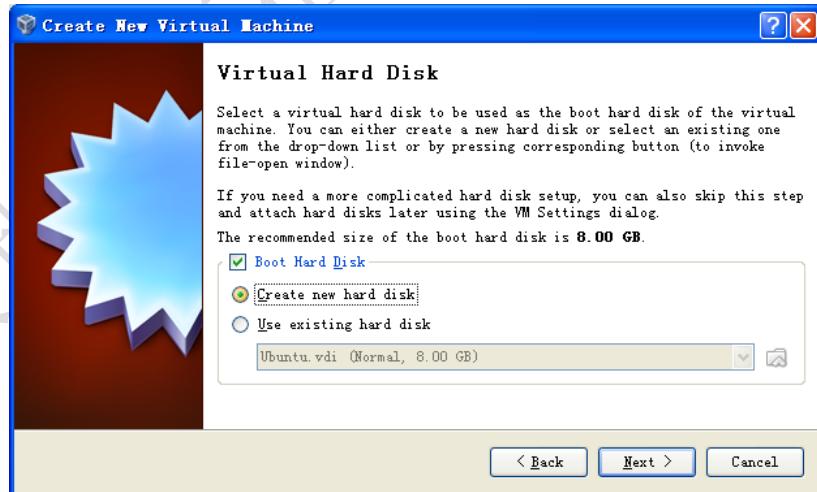


附图3 内存分配窗口

注意：

- 如果您的 PC 内存为 1G 或者更少，请保留默认设置；
- 如果您的 PC 内存超过 1G，则可以将 1/4 或者更多的内存分配给虚拟机，例如 PC 内存为 2G，则将 512MB 的内存分配给虚拟机。

- 4) 如果这是您第一次使用 VirtualBox，请在下方窗口中选择 **Create new hard disk** 选项，然后单击 **Next** 按钮；



附图4 创建新的虚拟硬盘窗口

- 5) 在下方虚拟硬盘创建向导窗口中单击 Next 按钮;



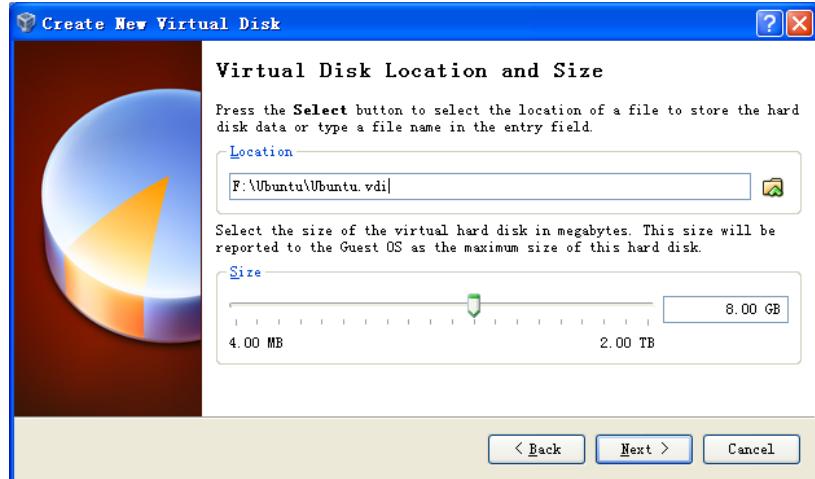
附图5 虚拟硬盘创建向导

- 6) 在下方窗口中选择 Fixed-size storage 选项，并单击 Next 按钮;



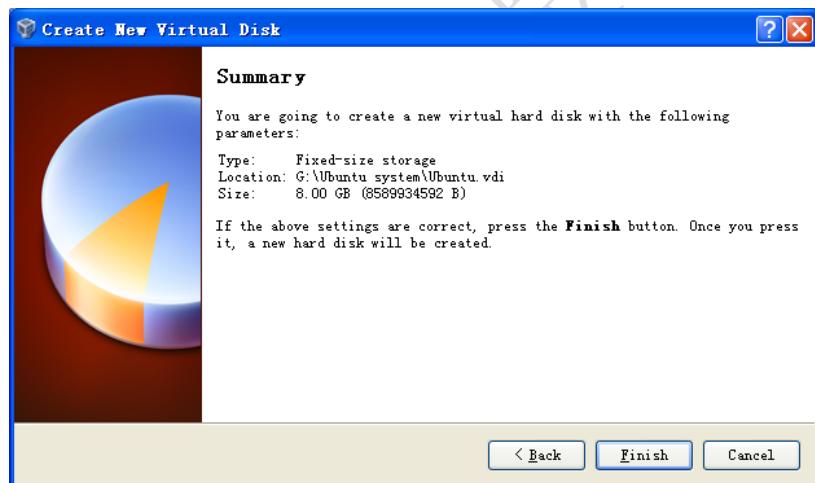
附图6 选择第二个选项

- 7) 在下方窗口中指定硬盘数据的存储位置以及默认虚拟硬盘的容量（至少 8G），
然后单击 **Next** 按钮；



附图7 虚拟硬盘设置窗口

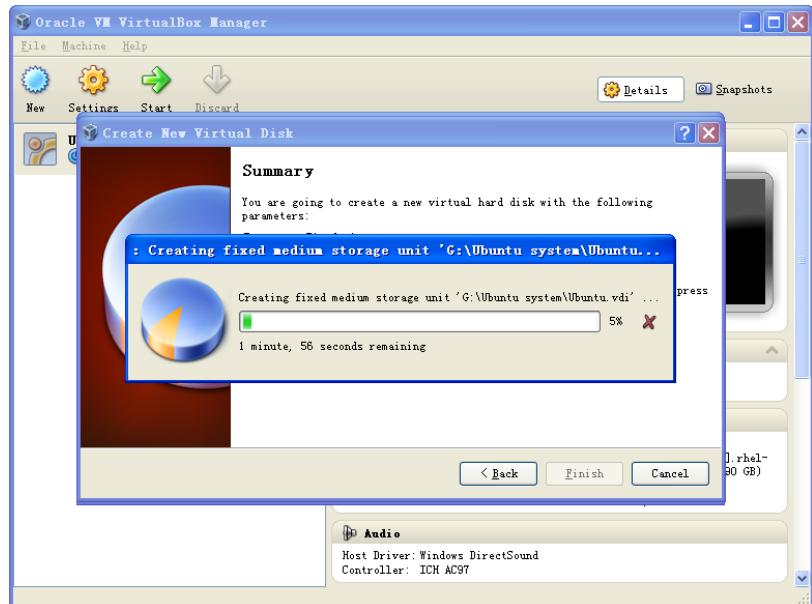
- 8) 在下方的虚拟硬盘信息窗口中单击 **Finish** 按钮；



附图8 虚拟硬盘信息

深圳

- 9) PC 开始创建虚拟硬盘驱动器;



附图9 创建虚拟硬盘驱动器

- 10) 完成驱动器创建后会显示摘要信息。请单击完成按钮即完成虚拟机安装过程;

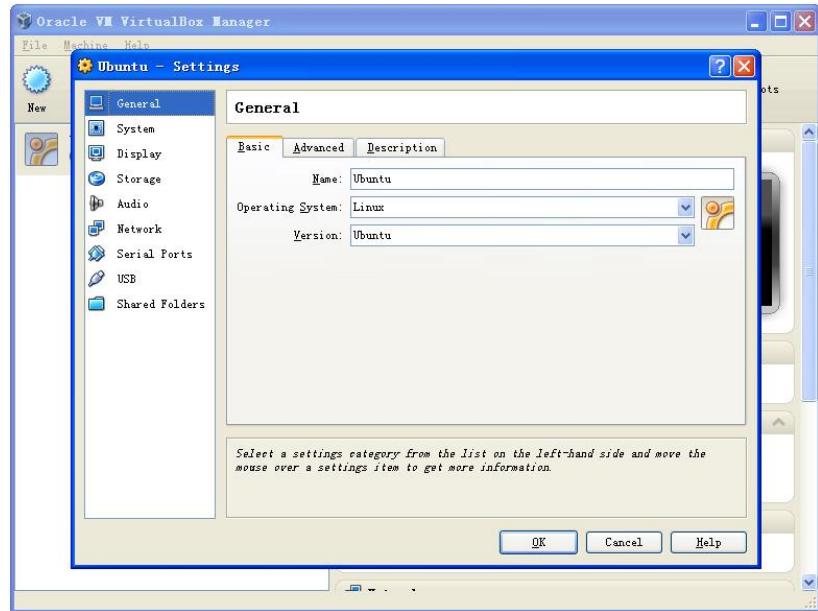


附图10 虚拟机配置完成

安装 Ubuntu Linux 系统

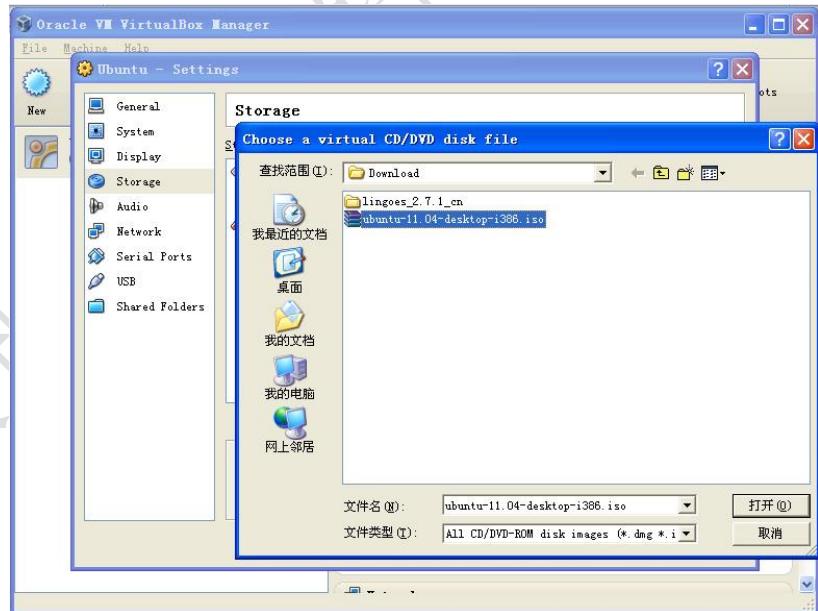
虚拟机安装完成后，我们就可以开始安装 Ubuntu Linux 系统了。首先请访问 <http://www.Ubuntu.com/download/Ubuntu/download>，下载 Ubuntu 的 ISO 映像文件，然后按照以下步骤进行安装。

- 1) 从开始菜单启动 VirtualBox, 然后在程序窗口上方单击 **Setting** 按钮, 弹出虚拟机设置窗口, 如下图所示;



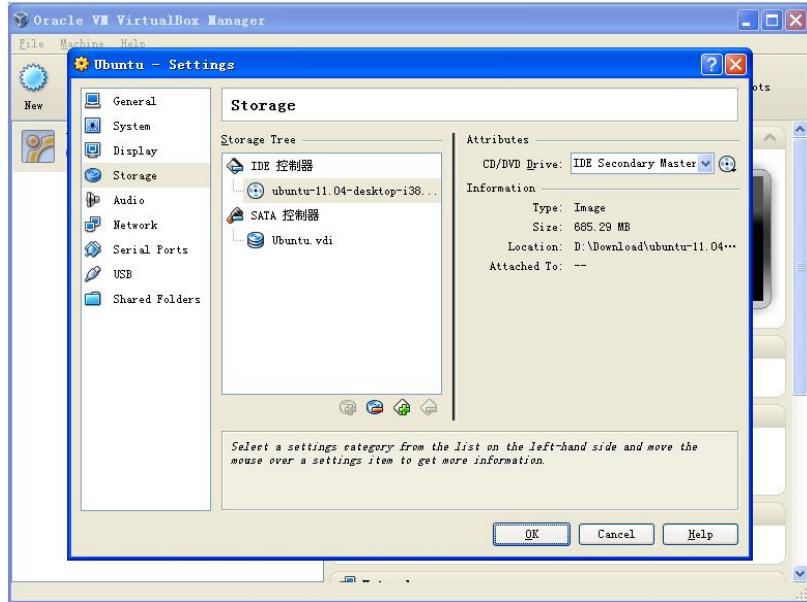
附图11 虚拟机设置

- 2) 在窗口左方选择 **Storage** 选项, 然后单击 IDC 控制器下 **Empty** 文字右方的光盘图标来指定 Ubuntu 映像文件的位置, 如下图所示;



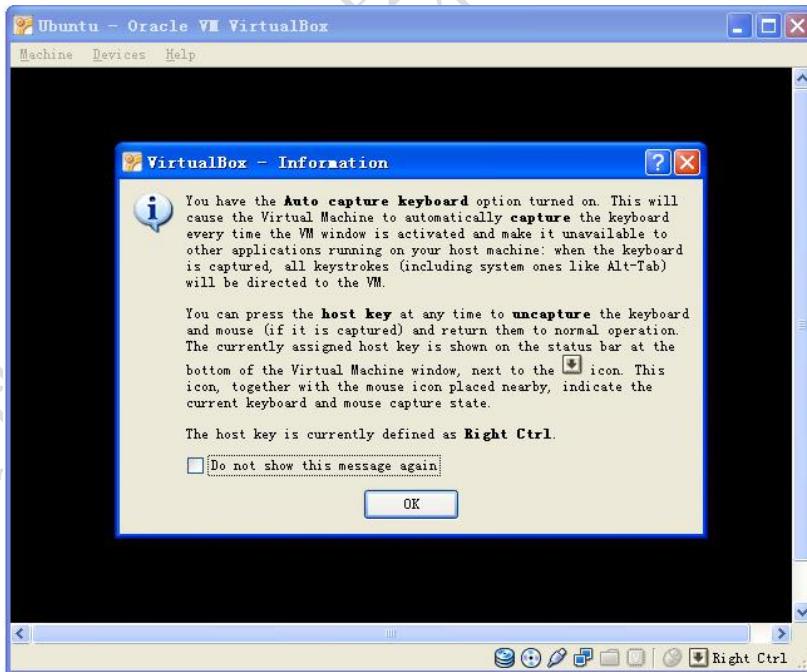
附图12 指定 Ubuntu 映像位置

- 3) 选中刚才添加的映像并单击 **OK** 按钮, 如下图所示;



附图13 选中 ISO 映像

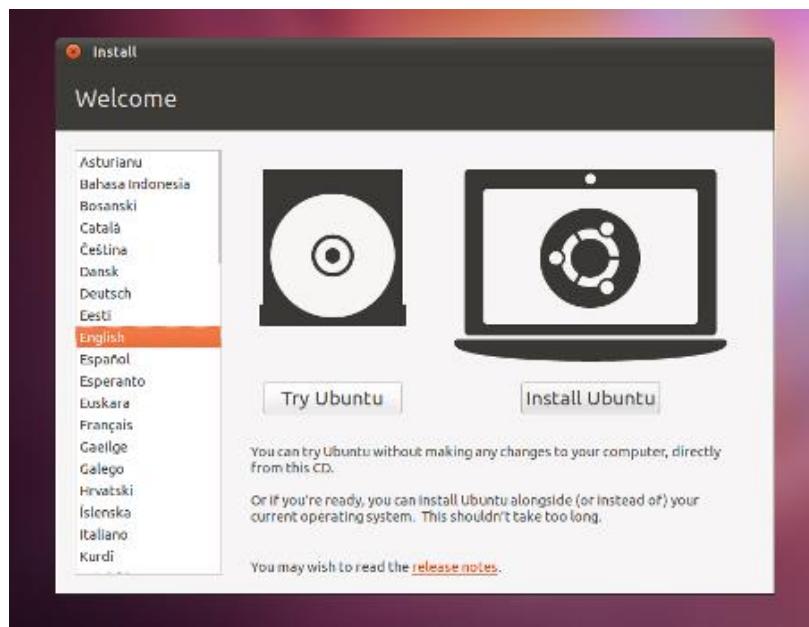
- 4) 单击 VirtualBox 窗口上方的 **Start** 按钮, 屏幕会弹出 Ubuntu 的安装初始化窗口, 如下图所示;



附图14 Ubuntu 初始化窗口

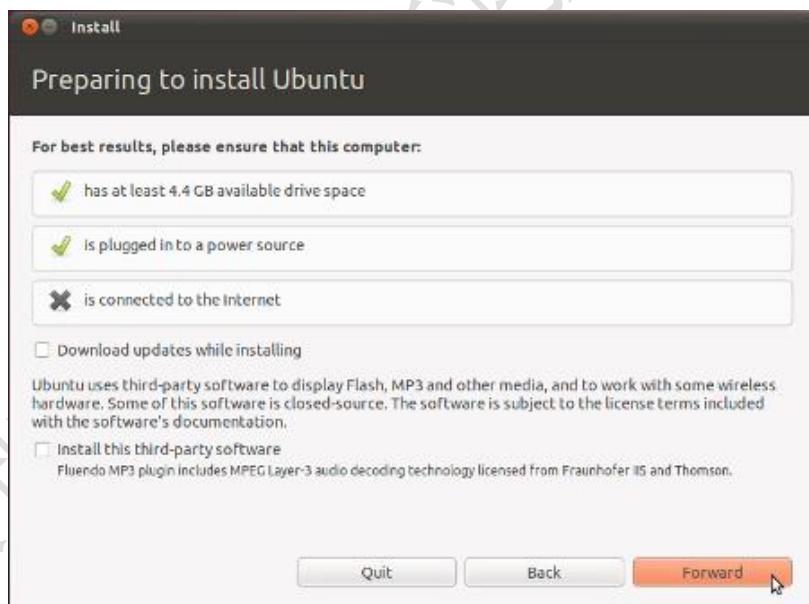
在 Ubuntu 安装窗口初始化过程中会出现一些提示信息, 只需要单击提示信息下方的 **OK** 按钮即可继续初始化进程。

- 5) 在出现以下窗口时单击 **Install Ubuntu** 进行安装，如下图所示：



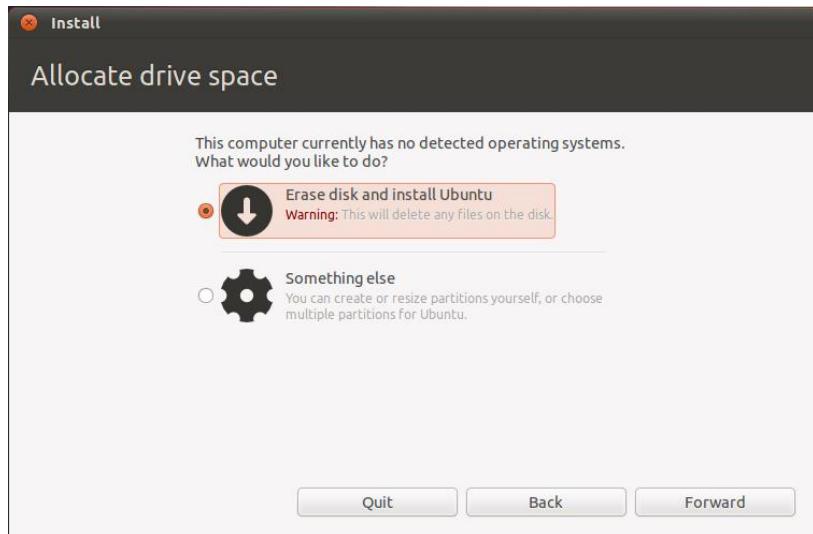
附图15 Ubuntu 安装窗口

- 6) 单击 Forward 按钮继续安装，如下图所示：



附图16 安装前的信息

- 7) 选择 Erase disk and install Ubuntu 选项，并单击 Forward 按钮。

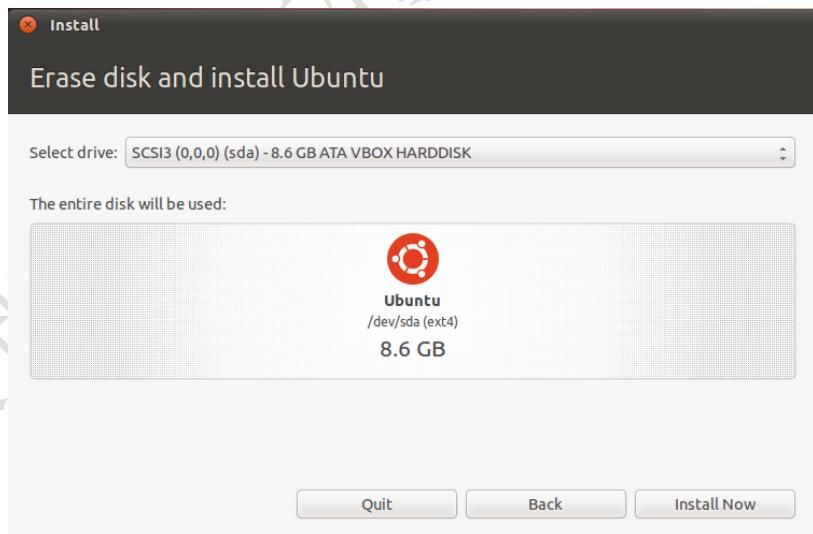


附图17 Ubuntu 安装选项

注意：

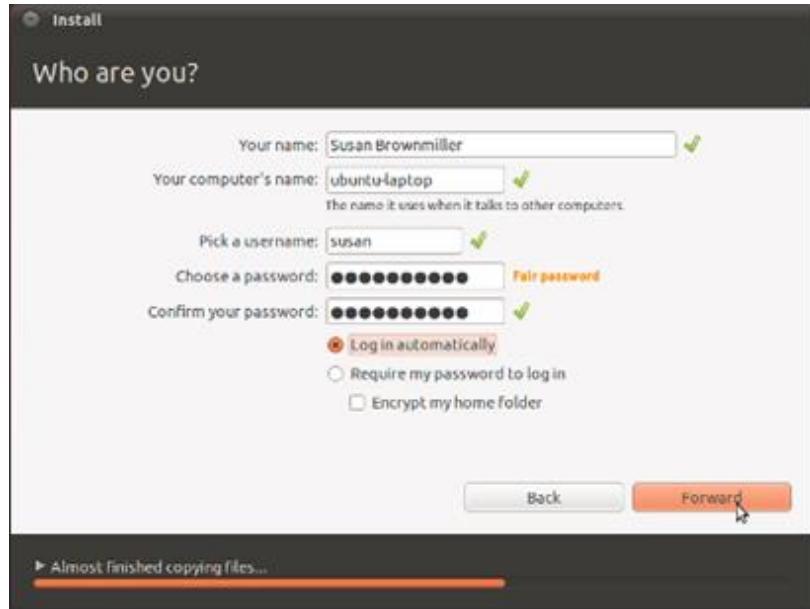
选择该选项并不会删除硬盘上物理分区中的任何内容。

- 8) 单击下方窗口中的 **Install Now** 按钮开始安装 Ubuntu;



附图18 安装确认窗口

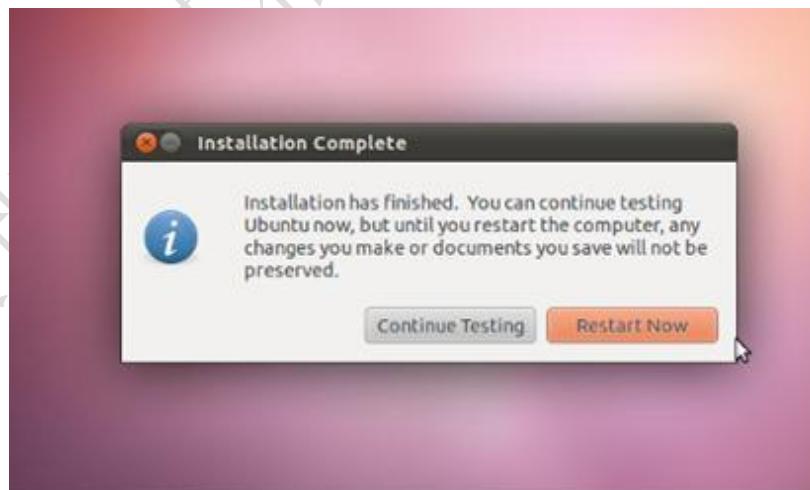
- 9) 在安装过程中安装程序会询问一些简单的问题，请输入相应的信息并单击 **Forward** 按钮即可。安装过程中的最后一个问题窗口如下图所示；



附图19 指定用户名和密码

在相应的文本框内输入用户名和密码后，选择 **Log in automatically** 选项并单击 **Forward** 按钮。

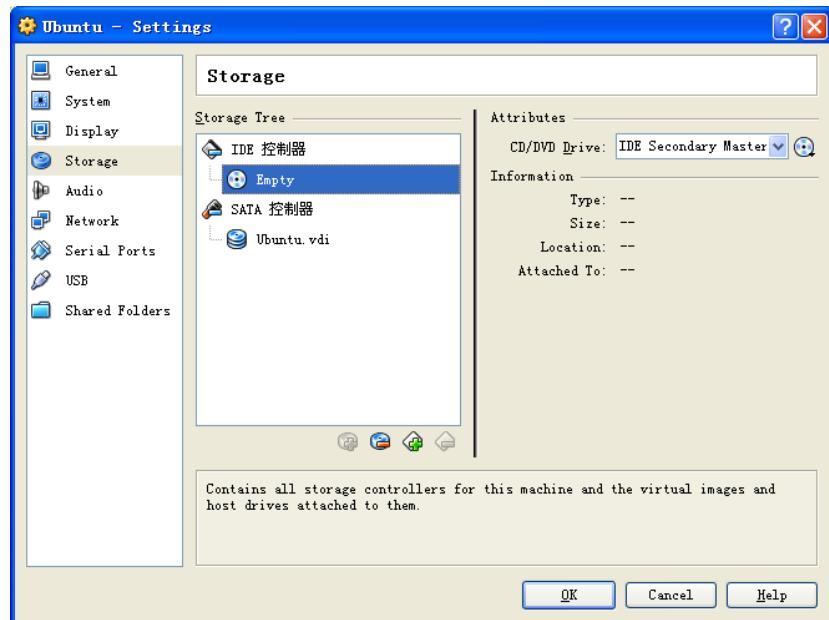
- 10) Ubuntu 的安装依据不同的 PC 性能可能需要 15 分钟至 1 小时左右。安装完成后会弹出如下图所示的提示窗口，请选择 **Restart Now** 重新启动 Ubuntu 系统；



附图20 重新启动系统

- 11) 重新启动后就可以使用 Ubuntu 系统了。通常在重新启动 Ubuntu 系统后 VirtualBox 会自动弹出附图 13 中载入的映像文件，如果没有自动弹出，您可以

在 VirtualBox 的 **Setting** 窗口中手动弹出该映像，使 IDE 控制器下显示为 Empty，如下图所示；

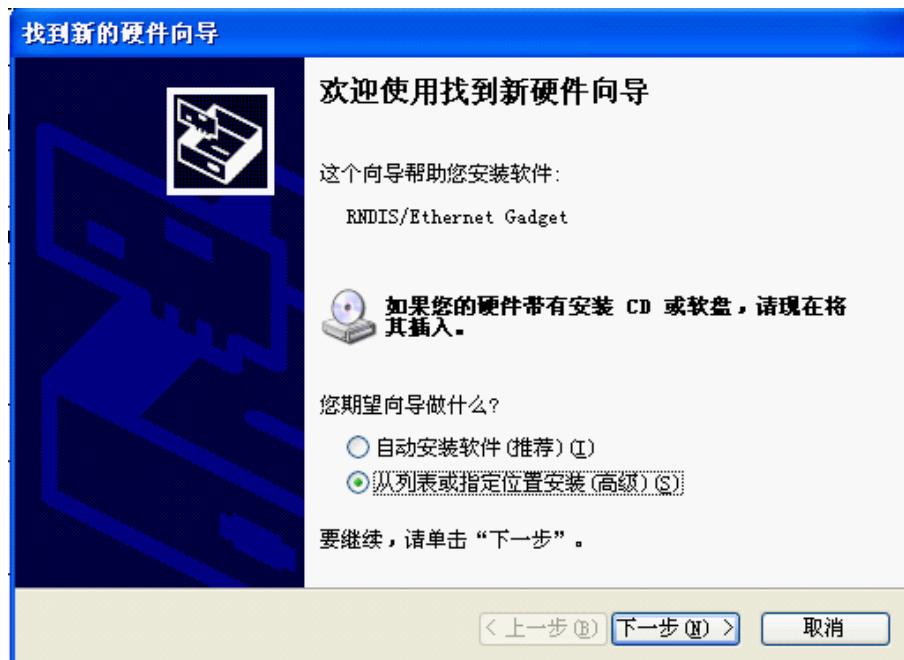


附图21 弹出 ISO 映像

深圳市英蓓特科技有限公司

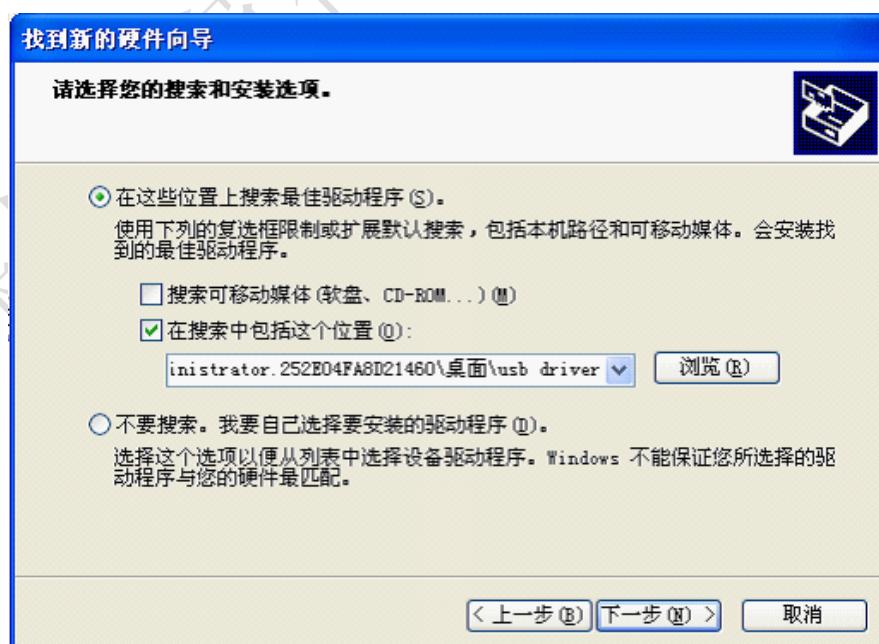
附录二 安装 Linux USB Ethernet/RNDIS Gadget 驱动

- 1) 如果你还没安装 Linux USB Ethernet/RNDIS Gadget 驱动, PC 会提示发现新硬件界面, 选中“从列表或指定位置安装”, 然后点击“下一步”。



附图22 找到新硬件

- 2) 指定 USB 驱动路径[光盘\linux\tools], 然后点击“下一步”。



附图23 选择驱动位置

- 3) 出现以下提示时，选择继续安装



附图24 警告信息

- 4) 等待驱动安装完毕



附图25 硬件向导完成窗口

附录三 制作 Linux 启动盘

以下内容将介绍如何制作一个双分区的 Linux 启动盘，以便让 SBC8600B 能够从启动盘的第一个分区启动系统，同时在第二个分区中保存根文件系统；

- 1) 将一张 TF 卡插入读卡器并连接到 PC 上；在 Ubuntu 系统中执行以下命令来确定 TF 卡的设备名；

- `$ dmesg | tail`

表 1 设备信息

```
[ 6854.215650] sd 7:0:0:0: [sdc] Mode Sense: 0b 00 00 08
[ 6854.215653] sd 7:0:0:0: [sdc] Assuming drive cache: write through
[ 6854.215659]   sdc: sdc1
[ 6854.218079] sd 7:0:0:0: [sdc] Attached SCSI removable disk
[ 6854.218135] sd 7:0:0:0: Attached scsi generic sg2 type 0
...
```

以上信息显示 TF 卡的设备名为**/dev/sdc**；

- 2) 执行以下命令来查看 Ubuntu 系统自动挂载的设备路径；

- `$ df -h`

表 2 设备路径

Filesystem	Size	Used	Avail	Use%	Mounted on
...					
/dev/sdc1	400M	94M	307M	24%	/media/disk

/dev/sdc1 所在行的尾端显示设备路径为**/media/disk**；

注意：

- 如果 TF 卡存在两个或两个以上分区，则以上设备路径信息会以 /dev/sdc1、/dev/sdc2、/dev/sdc3 等的形式显示每个分区。

3) 执行以下命令来卸载该设备；

- **\$ umount /media/disk**

4) 执行 fdisk 命令

- **\$ sudo fdisk /dev/sdc**

请确保命令行中输入的是整个设备的路径，而不是分区路径 /dev/sdc1 或 /dev/sdc2 等；

5) 上面的命令执行完成后，输入字符 p 来打印设备的分区记录，如下表所示：

表 3 分区记录

Command (m for help): [p]						
Disk /dev/sdc: 2021 MB, 2021654528 bytes						
255 heads, 63 sectors/track, 245 cylinders						
Units = cylinders of 16065 * 512 = 8225280 bytes						
Device	Boot	Start	End	Blocks	Id	System
/dev/sdc1	*	1	246	1974240+	c	W95 FAT32 (LBA)
Partition 1 has different physical/logical endings:						
phys=(244, 254, 63) logical=(245, 200, 19)						

根据以上信息记下设备的字节数，例如上表中的 2021654528 bytes；然后输入字符 d 来删除所有分区；

6) 如果表 3 中没有显示 255 heads 和 63 sectors/track，请按照以下操作来恢复 TF 卡；

A) 按照下表中的提示输入字符来设置 Heads 和 Sectors；

表 4 设置 Heads 和 Sectors

```
Command (m for help): [ x ] (输入 x 进入专家模式)
Expert Command (m for help): [ h ] (输入 h 来设置 heads)
Number of heads (1-256, default xxx): [ 255 ] (将 heads 设置为 255)
Expert Command (m for help): [ s ] (输入 s 来设置 sectors)
Number of sectors (1-63, default xxx): [ 63 ] (将 sector 设置为 63)
```

B) 使用以下公式计算 Cylinders 的数量;

$$\text{Cylinders} = \text{之前记下的设备字节数} \div 255 \div 63 \div 512$$

然后按照下表中的提示输入字符来设置 Cylinders;

表 5 设置 Cylinders

```
Expert Command (m for help): [ c ] (输入 c 来设置 cylinders)
Number of cylinders (1-256, default xxx): (输入刚才计算的 cylinder 数量)...
Expert Command (m for help): [ r ] (输入 r 回到一般模式)
```

7) 输入字符 p 来检查刚才设置的参数, 如下表所示;

表 6 检查参数

```
Command (m for help): [ p ]
63 sectors/track, 245 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot Start End Blocks Id System
```

8) 按照下表中的操作来建立 FAT32 启动分区和从 Windows 传输文件;

表 7 建立 FAT32 启动分区

```
Command (m for help): [ n ] (输入 n 开始建立分区)
Command action
e   extended
```

```
p primary partition (1-4)

[p] (输入 p 建立主分区)

Partition number (1-4): [1] (将分区号码设置为 1)

First cylinder (1-245, default 1): [】] (按下 Enter 键)

Using default value 1

Last cylinder or +size or +sizeM or +sizeK (1-61, default 61): [+5] (输入+5)

Command (m for help): [t] (输入 t)

Selected partition 1

Hex code (type L to list codes): [c] (输入 c 设置分区系统类型)

Changed system type of partition 1 to c (W95 FAT32 (LBA))
```

9) 输入 **a** 和 **1** 来将 TF 卡设置为 bootable 模式;

表 8 设置 bootable 模式

```
Command (m for help): [a]

Partition number (1-4): [1]
```

10) 按照下表的提示输入, 以便为根文件系统建立分区;

表 9 建立根文件系统分区

```
Command (m for help): [n] (输入 n 来建立分区)

Command action

e extended

p primary partition (1-4)

[p] (输入 p 选择主分区)

Partition number (1-4): [2] (将分区号码设置为 2)

First cylinder (7-61, default 7): [】] (按下 Enter 键)

Using default value 52
```

```
Last cylinder or +size or +sizeM or +sizeK (7-61, default 61): [] (按下 Enter 键)  
Using default value 245
```

- 11) 输入字符 **p** 来检查建立的分区，如下表所示：

表 10 检查分区

```
Command (m for help): [ p ]  
  
Disk /dev/sdc: 2021 MB, 2021654528 bytes  
255 heads, 63 sectors/track, 245 cylinders  
Units = cylinders of 16065 * 512 = 8225280 bytes  
  
Device Boot Start End Blocks Id System  
/dev/sdc1 * 1 6 409626 c W95 FAT32 (LBA)  
/dev/sdc2 7 61 1558305 83 Linux
```

- 12) 输入字符 **w** 来保存新分区记录，如下表所示：

表 11 保存分区

```
Command (m for help): [w]  
  
The partition table has been altered!  
  
Calling ioctl() to re-read partition table.  
  
WARNING: Re-reading the partition table failed with error 16: Device or resource busy.  
The kernel still uses the old table.  
The new table will be used at the next reboot.  
  
WARNING: If you have created or modified any DOS 6.x
```

partitions, please see the fdisk manual page for additional information.

13) 执行以下命令来格式化新建的两个分区；

- **\$ sudo mkfs.msdos -F 32 /dev/sdc1 -n LABEL1**
- **\$ sudo mkfs.ext4 -L LABEL2 /dev/sdc2**

注意：

- 以上命令中的 LABEL1 和 LABEL2 卷标可以由您自定义为其他名称。
- 在 FAT 和 EXT4 双分区的格式化完成后，需在 Ubuntu 中手动挂载 EXT4 分区，然后在 Windows 中重新格式化 FAT 分区，否则可能会出现无法从 TF 卡启动的情况。

附录四 搭建 TFTP 服务器

1) 安装客户端

```
$>sudo apt-get install tftp-hpa  
$>sudo apt-get install tftpd-hpa
```

2) 安装 inet

```
$>sudo apt-get install xinetd  
$>sudo apt-get install netkit-inetd
```

3) 服务器配置

首先，在根目录下建一个 tftpboot，并把属性改成任意用户可读写：

```
$>cd /  
$>sudo mkdir tftpboot  
$>sudo chmod 777 tftpboot
```

其次，在/etc/inetd.conf 里添加：

```
$>sudo vi /etc/inetd.conf //把下面的语句添加的此文件里  
tftp dgram udp wait root /usr/sbin/in.tftpd /usr/sbin/in.tftpd -s /tftpboot
```

然后，重新加载 inetd 进程：

```
$>sudo /etc/init.d/inetd reload
```

最后，进入目录 /etc/xinetd.d/，并在其中新建文件 tftp，把指定的内容加入到 tftp 文件中：

```
$>cd /etc/xinetd.d/ //进入目录 /etc/xinetd.d/  
$>sudo touch tftp //新建文件 tftp  
$>sudo vi tftp //编辑文件 tftp,把下面内容加入 tftp 文件中  
service tftp  
{  
    disable = no  
    socket_type  = dgram  
    protocol      = udp  
    wait          = yes  
    user          = root  
    server        = /usr/sbin/in.tftpd  
    server_args   = -s /tftpboot -c  
    per_source    = 11  
    cps           = 100 2  
}
```

4) 重新启动服务:

```
$>sudo /etc/init.d/xinetd restart  
$>sudo in.tftpd -l /tftpboot
```

5) 测试服务器

在/tftpboot 文件夹下新建立一个文件

```
$>touch abc
```

进入另外一个文件夹

```
$>tftp 192.168.1.15 (192.168.1.15 为本机 IP)  
$>tftp> get abc
```

如果可以下载说明服务器已经安装成功。

深圳市英蓓特科技有限公司

附录五 FAQ 总结

请访问: http://www.elinux.org/SBC8600_FAQ

深圳市英蓓特科技有限公司

技术支持和保修服务

技术支持



英蓓特科技对所销售的产品提供一年的免费技术支持服务，技术支持服务范围：
提供英蓓特科技嵌入式平台产品的软硬件资源；
帮助用户正确地编译和运行我们提供的源代码；
用户在按照本公司提供的产品文档操作的情况下，如本公司的嵌入式软硬件产品出现异常问题，我们将提供技术支持；
帮助用户判定是否存在产品故障。



以下情况不在我们的免费技术支持服务范围内，但我们将根据情况酌情处理：
用户自行开发中遇到的软硬件问题；
用户自行修改嵌入式操作系统遇到的问题；
用户自己的应用程序遇到的问题；
用户自行修改本公司提供的软件代码遇到的问题。

保修服务

- 1) 产品自出售之日起，在正常使用状况下为印刷电路板提供 12 个月的免费保修服务；
- 2) 以下情况不属于免费服务范围，英蓓特科技将酌情收取服务费用：
 - 无法提供产品有效购买凭证、产品识别标签撕毁或无法辨认，涂改标签或标签与实际产品不符；
 - 未按用户手册操作导致产品损坏的；
 - 因天灾（水灾、火灾、地震、雷击、台风等）或零件之自然耗损或遇不可抗拒力导致的产品外观及功能损坏；
 - 因供电、磕碰、房屋漏水、动物、潮湿、杂 / 异物进入板内等原因导致的产品外

观及功能损坏；

- 用户擅自拆焊零件或修改而导致不良或授权非英蓓特科技认可的人员及机构进行产品的拆装、维修，变更产品出厂规格及配置或扩充非英蓓特科技公司销售或认可的配件及由此引致的产品外观及功能损坏；
 - 用户自行安装软件、系统或软件设定不当或由电脑病毒等造成的故障；
 - 非经授权渠道购得此产品者。
 - 非英蓓特科技对用户做出的超出保修服务范围的承诺（包括口头及书面等）由承诺方负责兑现，英蓓特科技恕不承担任何责任；
- 3) 保修期内由用户发到我们公司的运费由用户承担，由我们公司发给用户的运费由我们承担；保修期外的全部运输费用由用户承担。
- 4) 若板卡需要维修，请联系技术支持服务部。

注意：

未经本公司许可私自将产品寄回的，英蓓特科技公司不承担任何责任。

联系方式

技术支持

电话： +86-755-33190868-872/875/897
Email: support@embest-tech.com

销售信息

电话： +86-755-33190868-860/861/862
传真： +86-755-25616057
Email: : chinasales@embest-tech.com

公司信息

网站： <http://www.embest-tech.cn>
地址： 深圳市南山区留仙大道 1183 号南山云谷创新产业园山水楼 4 楼 B