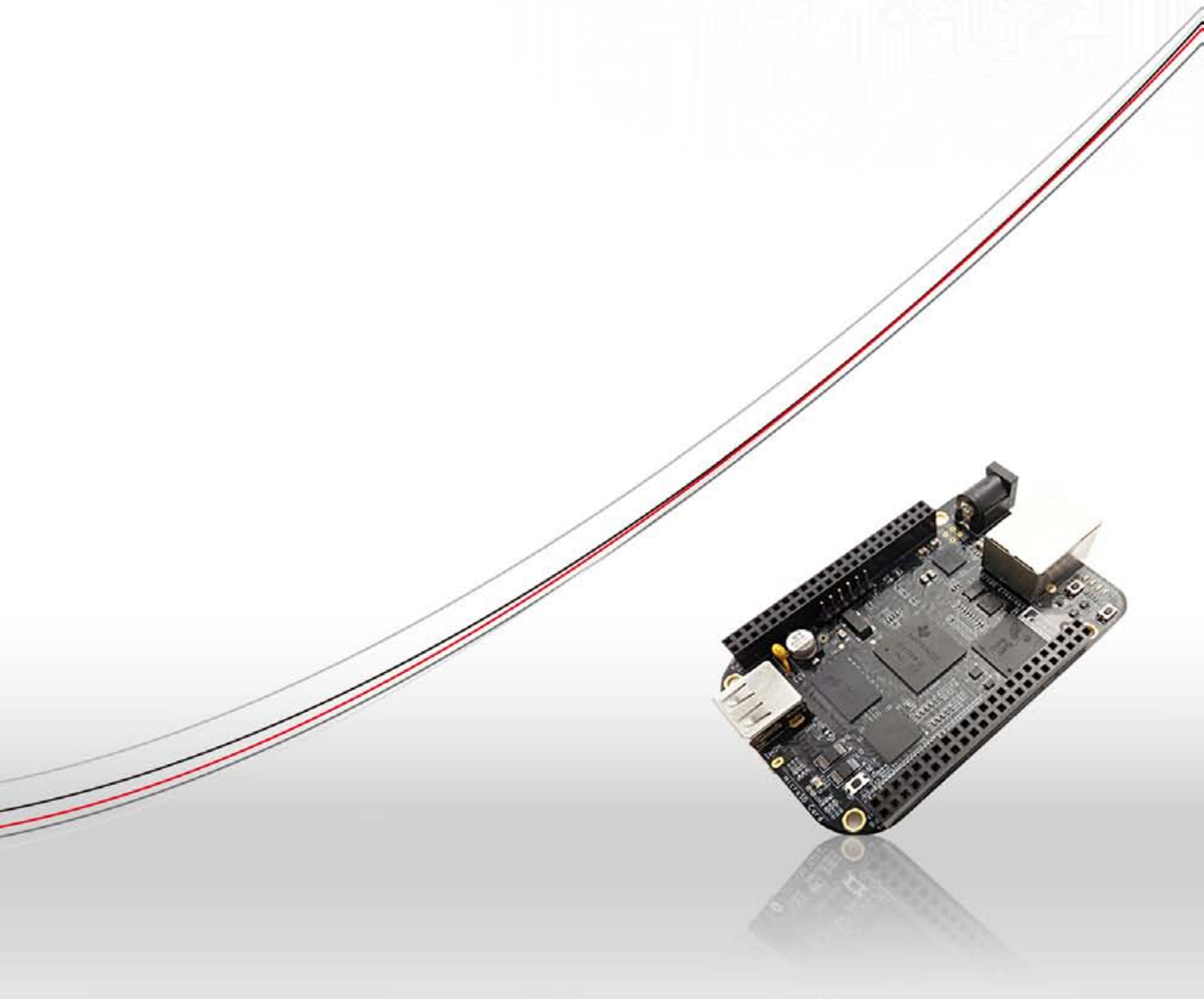
Texas Instruments

电子工程世界 eeworld.com.cn

深入浅出玩 TI Sitara

TI ARM MPU资源介绍及BeagleBone Black实战攻略



序言

这本电子书是 EEworld 众多坛友智慧的结晶，他们为此付出了很多的时间和精力，是他们学习历程的点滴积累，我想它也将会成为你学习 TI Sitara 系列芯片的良师益友，成为你学习路上的好伙伴，希望你能够喜欢上《深入浅出 TI Sitara》这本电子书，喜欢上 EEworld。

时间过的很快，时光已经进入了 2014 年，依稀记得 2012 年 EEworld 开展 Beaglebone 系列活动时的情境，论坛里关于 TI Sitara 的资料很少很少，即使整个网络上相关的资料也是屈指可数，学习起来难度很大。经过论坛两年时间的不断努力和推广，从 Beaglebone 到 Beaglebone Black，以及目前正在开展的 EEworld 芯币竞价活动——飞凌嵌入式工业级 Cortex-A8 开发板 OK335xS-Ⅱ活动，都是紧紧的围绕着 TI Sitara 系列芯片开展。论坛持续进行的一系列活动，同时也得到了坛友的积极响应。

经过了两年多时间坛友们的不懈努力和点滴积累，终于成就了这样的一本电子书，从这本电子书里你不仅可以学习到官方原汁原味的教程，带你开启学习 TI Sitara 的大门；同时你可以跟随 EEworld 先辈们的步伐，了解他们的学习经历，享受他们的学习成果，提升自身的学习效果。

这本电子书涉猎的范围很广：

1. 开发环境搭建：linux 系统，Android，IAR，tftp，nfs，Samba 等；
2. 驱动相关：GPIO，Nand，DS18B20，I2C，SPI 等；
3. 性能测试相关：iozone，iperf 等；
4. 项目实战：家庭多媒体服务器，数据采集等。

希望通过这本电子书的学习，你不仅能够熟悉和了解 TI Sitara 系列芯片，同时也能够激发你的兴趣和爱好，指导你做一些实际的事情。同样期望你能够记录下你的学习历程，经验或教训，希望在下一版的电子书里能够见到你的身影，为电子书增加新鲜的血液。

第一次写序言，难免有些紧张，非常感谢 soso 对我的信任和支持，让我在挤牙膏似的节奏中完成了这篇序言。

chenzhufly
2014-7-18

目录

序言 2
目录 3
第一章 基础篇 6
1.0 中国版BB-Black模块全解 6
1.1 看看BB-Black能干什么 16
1.2 BB-Black 开发环境的搭建 24
1.2.1 Linux的安装 24
1.2.2 Linux编译环境的搭建 29
1.2.3 开启tftp、NFS、Samba服务 32
1.2.4 uboot 、内核和文件系统的编译 34
1.2.5 用串口登录BeagleBone Black、用usb共享电脑网络、内核模块的本地编译 42
1.2.6 用SSH远程登录BeagleBone 46
1.2.7 基于BeagleBone板的Linux驱动的调试 50
1.2.8 BeagleBone 的NFS启动配置和流程 58
1.2.9 聊聊BeagleBone Black的cape和device tree overlay和dtc命令 62
1.2.10 从NAND FLASH中启动BeagleBone 68
1.2.11 启动BeagleBone，运行“Hello World”程序 72
1.2.12 使用EclipseCDT搭建BeagleBone Linux开发环境 76
1.2.13 学习Sitara AM335x 83
1.2.14 AM335x的PRUSSv2简介与应用 84
1.3 EEworld论坛网友玩BeagleBone Black之基础实验篇 92
1.3.1 BeagleBone Black GPIO的相关操作 92
1.3.2 BeagleBone Black的ADC 的使用 108
1.3.3 BeagleBone Black的串口UART的使用 111
1.3.4 BeagleBone Black的I2C的使用 113
1.3.5 BeagleBone Black的SPI的使用 122
1.3.6 EE_BeagleBone_Cape LCD 128
1.3.7 EE_BeagleBone_Cape之EEPROM 133
1.3.8 EE_BeagleBone_Cape 4x4矩阵键盘 138
1.3.9 EE_BeagleBone_Cape之NAND 144
1.3.10 BeagleBone Black 利用Ubuntu上网 153

1.3.11 基于BeagleBone和DS18B20的温度测量 155
1.3.12 BeagbleBone之网页控制LED灯 157
1.3.13 基于BeagleBone的数据采集界面的实现 161
1.3.14 基于BeagleBone的WIFI通信 184
1.3.15 BeagleBone上实现AM3359与FPGA的GPMC通信 187
1.3.16 BeagleBone上面试跑飞鸽传书 200
1.3.17 BeagleBone试用无刷电机和2.4G无线经验分享 201
1.3.18 BeagleBone外围电路设计心得Opencv 无刷电机 203
1.3.19 BeagleBone Black不一样的玩法：创意感应式乐器 207
1.3.20 用VNC在电脑上直接显示BBB的framebuffer图像 208
1.3.21 基于BB-Black和hmc5843三轴电子罗盘 209
1.3.22 BeagleBone之有生有色之实现无损音乐播放器 213
1.3.23 BB-Black的FFTW移植-使用cycle_counter 217
1.3.24 BB-Black 入门基础之OpenCV的交叉编译 218
1.4 EEworld论坛网友对BB-Black相关性能测评 226
1.4.1 BeagleBone IO速率测试 226
1.4.2 基于BeagleBone的内存性能测试 232
1.4.3 使用iozone测试BeagleBone文件系统性能 235
1.4.4 基于Iperf的BeagleBone网络性能测试 238

第二章 官方课程 242

2.0 本章内容综述 242
2.1 TI Sitara AM335x系列概述 242
2.1.1 AM335x的资源 244
2.1.2 AM335x的软件架构 245
2.1.3 AM335x Linux Kernel架构： 247
2.1.4 AM335x的文件系统： 248
2.1.5 AM335x的硬件资源 251
2.1.6 TAM335x平台工具 253
2.2 基于AM335x平台Uboot的一些移植方法 256
2.3 优化Linux的启动时间 263
2.4 如何使用CCS去Debug 269
2.5 如何使用CCS调试Linux 271
2.5.1 实验前准备 271
2.5.2 创建工程 272

2.5.3 配置仿真器文件 274
2.5.4 编译Linux 276
2.5.5 调试Linux 278
2.6 使用IAR连接J-Link 调试AM335x 283
2.6.1 IAR软件打开项目工程—uartEcho 283
2.6.2 对uartEcho工程中所需要使用的lib库文件进行编译 285
2.6.3 完成库文件的编译后，进行uartEcho工程文件的编译 291
2.6.4 配置并调试uartEcho工程 292
2.7 基于AM335x的工业自动化应用 296
2.8 基于AM335x LED WALL应用 308
2.9 AM335x在HMI系统中的应用 319
2.10 基于AM335x的 CAN 和 EtherCAT 功能演示 327

第三章EEworld论坛网友实战篇 330

3.1 家居生活管理中心 330
3.2 BeagleBone建立自己的家庭多媒体服务器 342
3.3 基于BB及FPGA的高速数据采集及DDS信号发生器设计与实现 348
3.3.1 方案总体设计 348
3.3.2 计算机端软件实现 350
3.3.3 BB端设备驱动及应用程序实现 354
3.3.4 FPGA端软件实现 371
3.4 基于BeagleBone的Android4.0.3 ICS的移植 376
3.5 基于Qt和MySQL的ZigBee数据采集系统 380
3.6 基于BBB 的简单频谱仪 386

附录1 BeagleBone创意列表 387

附录2 编委信息与后记 390

附录3 版权说明 391

第一章 基础篇

本章主要介绍中国版 BB-Black 开发板的硬件资源、功能参数、接口，并通过 EEworld 论坛网友评测实例（开发环境的搭建、运行、基础开发项目等），使得初次使用中国版 BB-Black 的工程师有一个初步认识。

BeagleBone Black 是开源社区组织 BeagleBoard.org 推出的新一代产品，从电子发烧友、工程师到学生，每个人都能够以仅 45 美元的价格获得一款即用型单板计算机。该平台仅有信用卡大小，是一款开放式软硬件开发平台，可快速地将开发者的构想转化为产品。为了让中国客户能同步享有 BeagleBone Black 的技术优势，TI 联合英蓓特科技发布了中国版 BB-Black。该产品软硬件与全球发售版本完全兼容，支持全部的“standard BeagleBone Cape Plug-in boards and software”，拥有中国版 BB-Black 的工程师不仅可以获得 BeagleBone 开源社区的丰富资源支持，还可以获得 TI 的官方支持以及英蓓特的完善技术服务。

1.0 中国版 BB-Black 模块全解

中国版 BB-Black 是英蓓特科技推出的一款基于 AM3359 处理器的开发套件。处理器集成了高达 1GHz 的 ARM Cortex-A8 内核，并提供了丰富的外设接口。中国版 BB-Black 的扩展接口包括网口、USB Host、USB OTG、TF 卡接口、串口、JTAG 接口（默认不焊）、HDMI D Type 接口、eMMC、ADC、I2C、SPI、PWM 和 LCD 接口。

中国版 BB-Black 的应用场景非常广泛，能够满足包括游戏外设、家庭和工业自动化、消费类医疗器械、打印机、智能收费系统、智能售货机称重系统、教育终端和高级玩具等在内的各个领域的不同需求。

图 1-1、图 1-2、图 1-3 所示为中国版 BB-Black 的硬件资源及相关接口。图 1-4 所示为整个板子的系统资源。

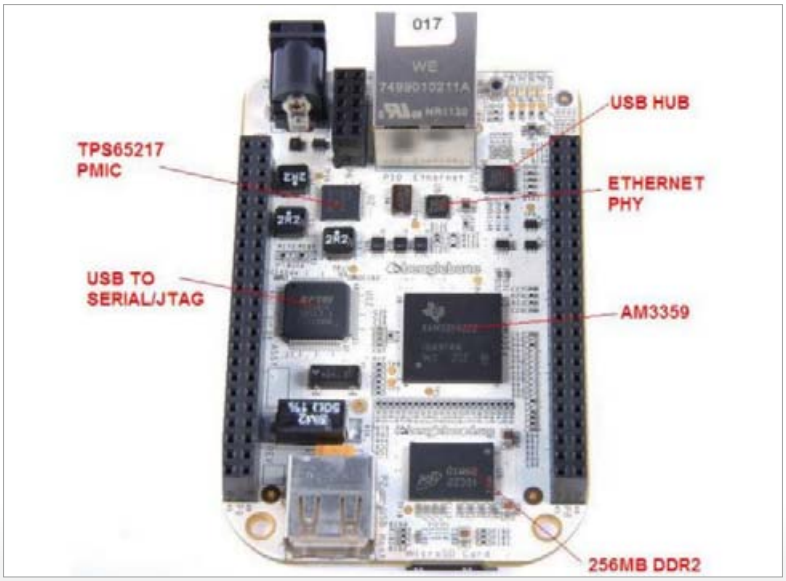


图 1-1：中国版 BB-Black 的正面资源

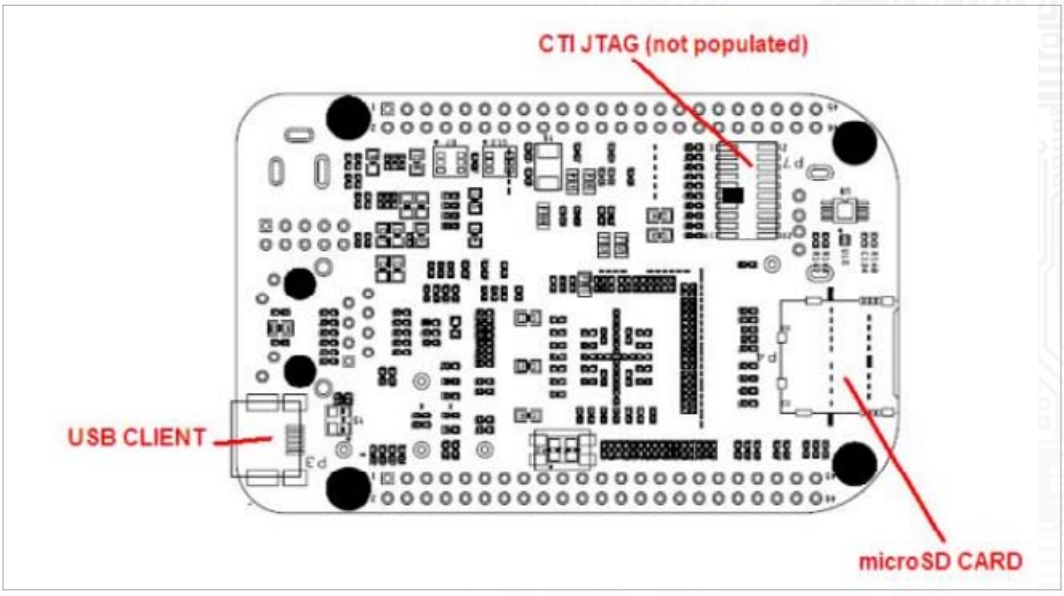


图 1-2：中国版 BB-Black 的背面资源

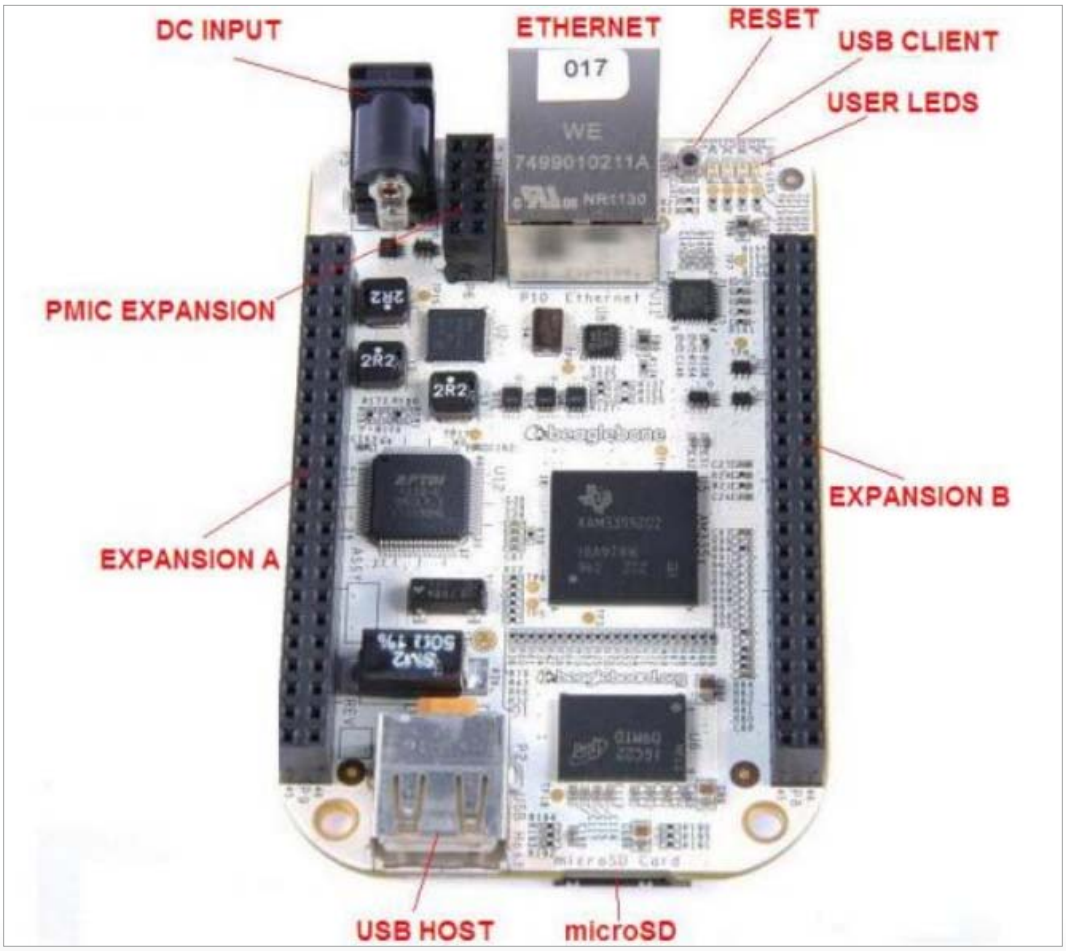


图 1-3 中国版 BB-Black 的相关接口

中国版 BB-Black 开发板采用了 AM3359 处理器。板上的其他资源如下所述。

板载存储器：

- ◆ 2GB eMMC Flash 存储器
- ◆ 512MB DDR3 SDRAM 存储器

板载接口：

- ◆一个 HDMI D type 接口（16 位色输出，支持音频输出）；
- ◆一个 10/100M 以太网接口（RJ45 连接器）；
- ◆一个集成了 PHY 的高速 USB 2.0 OTG 接口（Mini USB B 型连接器）；
- ◆一个集成了 PHY 的高速 USB 2.0 HOST 接口（USB A 型连接器）；
- ◆一个 TF 卡接口（兼容 SD/MMC）、一个 3 线调试串口（6-pin 2.54 间距连接器）；
- ◆一个 HDMI D type 接口、两个扩展接口，可扩展 LCD、UART、eMMC、ADC、I2C、SPI 和 PWM 等接口；
- ◆一个 JTAG 接口（20-pin 标准接口，默认未焊接连接器）；

按钮和按键：

- ◆一个启动选择按键
- ◆一个电源按键
- ◆一个复位按键
- ◆一个 LED 电源指示灯
- ◆四个用户自定 LED 灯

下面介绍板上 CPU&CPU 周边芯片、以及各种板载接口的引脚定义。

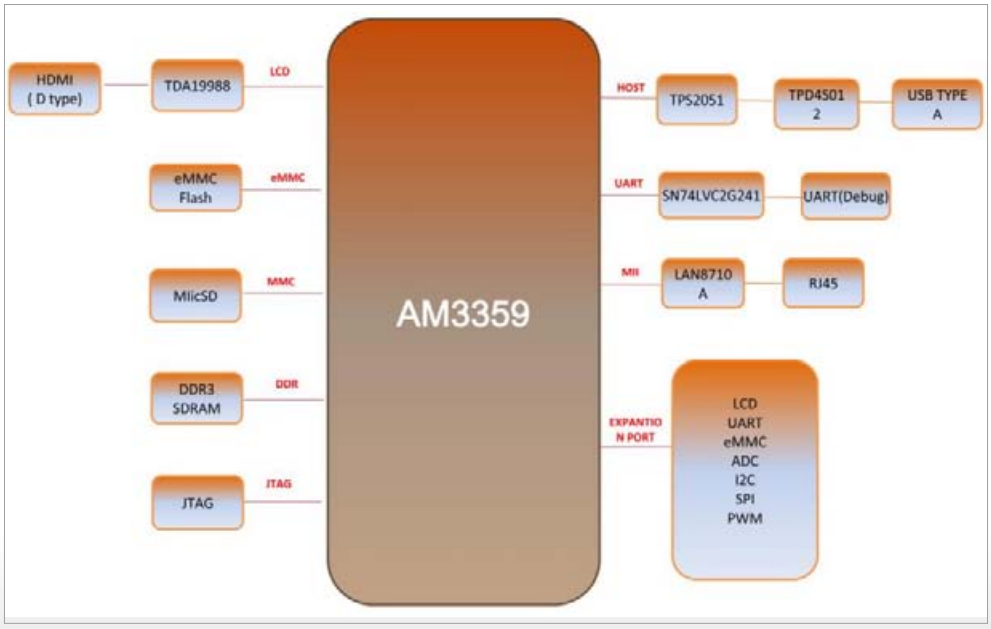


图 1-4 中国版 BB-Black 的系统图

CPU：AM3359 是基于 ARM Cortex-A8 内核的微处理器，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 和 Android 等高级操作系统。处理器包含了多个子系统。微处理器单元（MPU）子系统基于 ARM Cortex-A8 微 处理器；POWERVR SGX 图形加速子系统用于 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 内核，这就允许设备有独立的操作和时钟，从而在复杂系统解决方案中有更大的灵活性。

AM3359 的时钟信号包括两个输入时钟 -OSC1 和 OSC0，和两个输出时钟 - LCKOUT1 和 LCKOUT2，其中：

◆ OSC1 ：为 RTC 提供 2.768KHz 参考时钟并用于连接 RTC_XTALIN 和 RTC_ XTALOUT 终端。

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

复位功能由 CPU 的 PWRONRSTn 信号实现；低电平有效。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。可编程实时单元和工业通讯子系统（PRU-ICSS）包含了两个 32 位 RISC 内核（可编程实时单元，即 PRUs）、存储器、终端控制器以及能够支持更多周边接口和协议的内部外设。

与 AM1x 和 OMAP-L13x 系列处理器相比，AM3359 拥有的这种子系统是下一代的 PRU 子系统。

3D 图形引擎方面，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：

1、Tile-Based 架构，处理能力高达 20Mploy/ 秒

2、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

3、超过 Microsoft VS3.0、PS3.0 和 OGL2.0 的高级渲染功能指令集

4、工业标准 API，支持 Direct3D Mobile、OGL-ES 1.1 和 2.0、OpenVG 1.0 和 OpenMax

CPU 周边芯片

TPS65217 电源管理芯片：TPS65217 是一个综合的电源管理芯片，它包括 3 个降压转换器、一个 LED 驱动器、四格 LDO 和电池充电单元，并通过 IIC 与 CPU 相连。TPS65217 的主要作用是为 CPU、eMMC 和 DDR 等板载芯片提供 1.1V、1.2V、1.5V、1.8V 或 3.3V 等电压，以保证芯片正常工作。

MTFC2GMVEAeMMCFlash 存储器 :MTFC2GMVEA-0M WT 是中国版 BB-Black 的 Flash 存储器，大小为 2GB。

MT41K256M16HA-125 DDR 存储器：MT41K256M16HA-125 是中国版 BB-Black 的 DDR3 SDRAM 存储器，大小为 512MB，由 1 片 16bit 的 MT41K256M16HA-125 芯片构成。

LAN8710A-EZC-TR Ethernet 芯片 :LAN8710A-EZC-TR 是一个低功耗的 10BASE-T/100BASE-TX 物理层以太网收发器。兼容 IEEE 802.3-2005 标准，支持 1.6V~3.6V 之间的可变 I/O 电压。该芯片支持物理层自动协商机制，因此中国版 BB-Black 可通过直通网线或者交叉网线连接到网络 hub 或者电脑。

TDA19988 发送器：TDA19988 是 NXP 推出的小尺寸、低功耗的 HDMI 发送器，支持 HDMI1.4a 标准。TDA19988 能够将 RGB 信号转换成 HDMI 信号，并且后向兼容 DVI1.0 标准。

接口 / 按钮 /LED

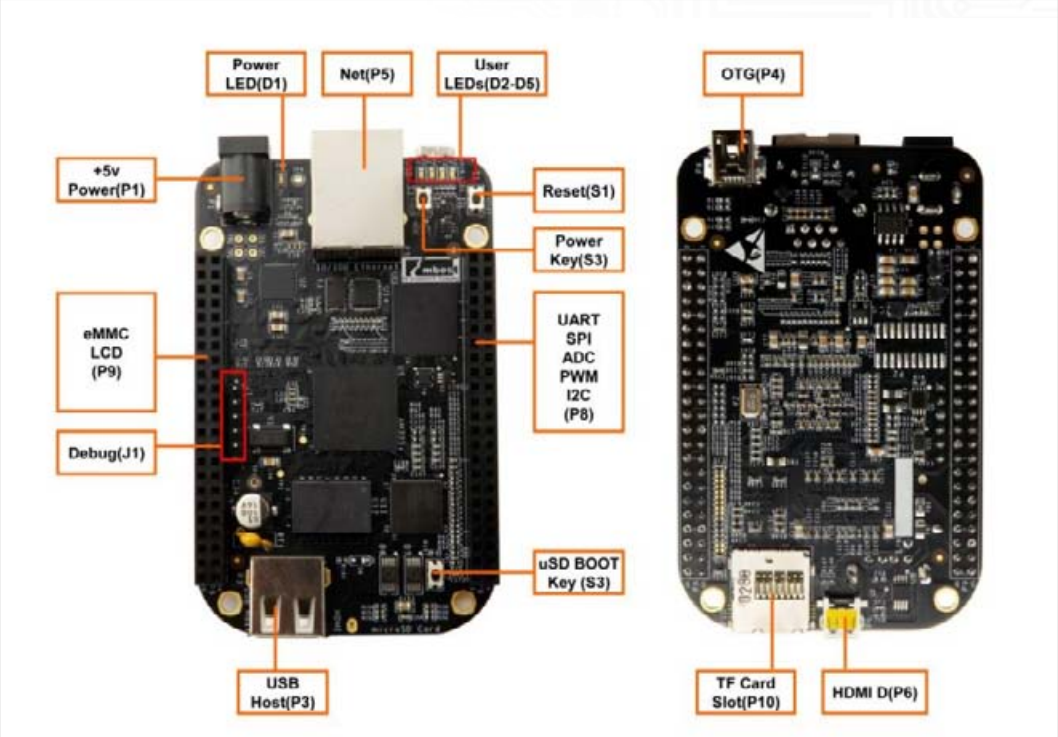


图 1-5 接口 / 按钮 /LED

表 1-1 电源接口（P1）

引脚	信号定义	描述
1	GND	GND
2	+5V	Power supply（+5V）
3	NC	NC

表 1-2 JTAG 接口（P2）

引脚	信号定义	描述
1	TMS	Test mode select
2	NTRST	Test system reset
3	TDI	Test data input
4	GND	GND
5	VIO	3.3V
6	NC	NC
7	TDO	Test data output
8	GND	GND
9	RTCK	Receive test clock
10	GND	GND
11	TCK	Test clock
12	GND	GND
13	EMU0	Test emulation 0
14	EMU1	Test emulation 1
15	SRST	System reset
16	GND	GND
17	EMU2	Test emulation 2
18	EMU3	Test emulation 3
19	EMU4	Test emulation 4
20	GND	GND

表 1-3 USB Host 接口（P3）

引脚	信号定义	描述
1	VB	+5V
2	D-	USB data-
3	D+	USB data+
4	GND	GND

表 1-4 USB Device 接口（P4）

引脚	信号定义	描述
1	VB	+5V
2	D-	USB data-
3	D+	USB data+
4	ID	USB ID
5	G1	GND

表 1-5 LAN 接口

引脚	信号定义	描述
1	RD+	Receive data+
2	RD-	Receive data-
3	TD+	Transmit data+
4	RCT	Receive common terminal
5	TCT	Transmit common terminal
6	TD-	Transmit data-
7	NC	No connect
8	GND	GND
9	GRNA	Green LED backlight anode
10	GRNC	Green LED backlight catgode
11	YELC	Yellow LED backlight catgode
12	YELA	Yellow LED backlight anode
13	SHD1	Shield
14	SHD2	Shield

表 1-6 HDMI D Type 接口

引脚	信号定义	描述
1	HPLG	Hot plug and play detect
2	NC	No connect
3	HDMI_TX2+	TMDS data 2+
4	DAT2_S	GND
5	HDMI_TX2-	TMDS data 2-
6	HDMI_TX1+	TMDS data 1+
7	DAT1_S	GND
8	HDMI_TX1-	TMDS data 1-
9	HDMI_TX0+	TMDS data 0+
10	DAT0_S	GND
11	HDMI_TX0-	TMDS data 0-
12	HDMI_CLK+	TMDS data clock+
13	CLK_S	GND
14	HDMI_CLK-	TMDS data clock-
15	HDMICONN_CEC	Consumer electronics control
16	DDC/CEC GND	GND
17	HDMI_I2CSCL	IIC master serial clock
18	HDMI_I2CSDA	IIC serial bidirectional data
19	5V	5V

表 1-7 右边扩展接口(P8)

引脚	信号定义	描述
1	GND	GND
2	GND	GND
3	GPIO1_6	mmc1_dat6
4	GPIO1_7	mmc1_dat7
5	GPIO1_2	mmc1_dat2
6	GPIO1_3	mmc1_dat3
7	TIMER4	Time interrupt input
8	TIMER7	Time interrupt input
9	TIMER5	Time interrupt input
10	TIMER6	Time interrupt input
11	GPIO1_13	GPIO
12	GPIO1_12	GPIO
13	EHRPWM2B	eHRPWM output
14	GPIO0_26	GPIO
15	GPIO1_15	GPIO
16	GPIO1_14	GPIO
17	GPIO0_27	GPIO
18	GPIO2_1	GPIO
19	EHRPWM2A	eHRPWM output
20	GPIO1_31	mmc1_dat5
21	GPIO1_0	mmc1_dat4
22	GPIO1_5	mmc1_dat5
23	GPIO1_4	mmc1_dat4
24	GPIO1_1	mmc1_dat1
25	GPIO1_0	mmc1_dat0
26	GPIO1_29	GPIO
27	GPIO2_22	LCD vertical sync
28	GPIO2_24	LCD pixel clock
29	GPIO2_23	LCD horizontal sync
30	GPIO2_25	LCD AC bias enable chip select
31	UART5_CTSN	LCD data14
32	UART5_RTSN	LCD data15
33	UART4_RTSN	LCD data13
34	UART3_RTSN	LCD data17
35	UART4_CTSN	LCD data12
36	UART3_CTSN	LCD data16
37	UART5_TXD	LCD data08
38	UART5_RXD	LCD data09
39	GPIO2_12	LCD data06
40	GPIO2_13	LCD data07
41	GPIO2_10	LCD data04
42	GPIO2_11	LCD data05
43	GPIO2_8	LCD data02
44	GPIO2_9	LCD data03
45	GPIO2_6	LCD data00
46	GPIO2_7	LCD data01

表 1-8 左边扩展接口（P9）

引脚	信号定义	描述
1	DGND	GND
2	DGND	GND
3	VDD_3V3B	3.3V power
4	VDD_3V3B	3.3V power
5	VDD_5V	5V power
6	VDD_5V	5V power
7	SYS_5V	5V power
8	SYS_5V	5V power
9	PWR_BUT	Push-button monitor input
10	SYS_RESETn	System reset input
11	UART4_RXD	UART receive data
12	GPIO1_28	GPIO
13	UART4_TXD	UART transit data
14	EHRPWM1A	eHRPWMoutput
15	GPIO1_16	GPIO
16	EHRPWM1B	eHRPWM output
17	I2C1_SCL	I2C clock
18	I2C1_SDA	I2C data
19	I2C2_SCL	I2C clock
20	I2C2_SDA	I2C data
21	UART2_TXD	UART transit data
22	UART2_RXD	UART receive data
23	GPIO1_17	GPIO
24	UART1_TXD	UART transit data
25	GPIO3_21	GPIO
26	UART1_RXD	UART receive data
27	GPIO3_19	GPIO
28	SPI1_CS0	SPI chip select
29	SPI1_D0	SPI data0
30	SPI1_D1	SPI data1
31	SPI1_SCLK	SPI clock
32	VADC	ADC power
33	AIN4	ADC input
34	AGND	ADC GND
35	AIN6	ADC input
36	AIN5	ADC input
37	AIN2	ADC input
38	AIN3	ADC input
39	AIN0	ADC input
40	AIN1	ADC input
41	CLKOUT2	Audio input master clock
42	GPIO0_7	Interrupt output
43	DGND	GND
44	DGND	GND
45	DGND	GND
46	DGND	GND

表 1-9 按钮

按钮	信号定义	描述
S1	Reset	重启按键
S2	uSD BOOT	启动方式选择
S3	Power	在系统运行时长按该按钮来关闭系统

表 1-10 LED

LED	信号定义	描述
S1	Power LED	电源指示灯
S2	User LED	用户自定义 LED 灯
S3	User LED	用户自定义 LED 灯
S4	User LED	用户自定义 LED 灯
S5	User LED	用户自定义 LED 灯

详情请点击：>> 中国版 BB -Black 英蓓特参考资料

1.1 看看 BB-Black 能干什么

本节内容是关于 EE_BeagleBone_Cape 硬件系统设计的记录，主要谈谈为什么要设计这样的一块载板，这块载板有哪些功能，能够干哪些事情。

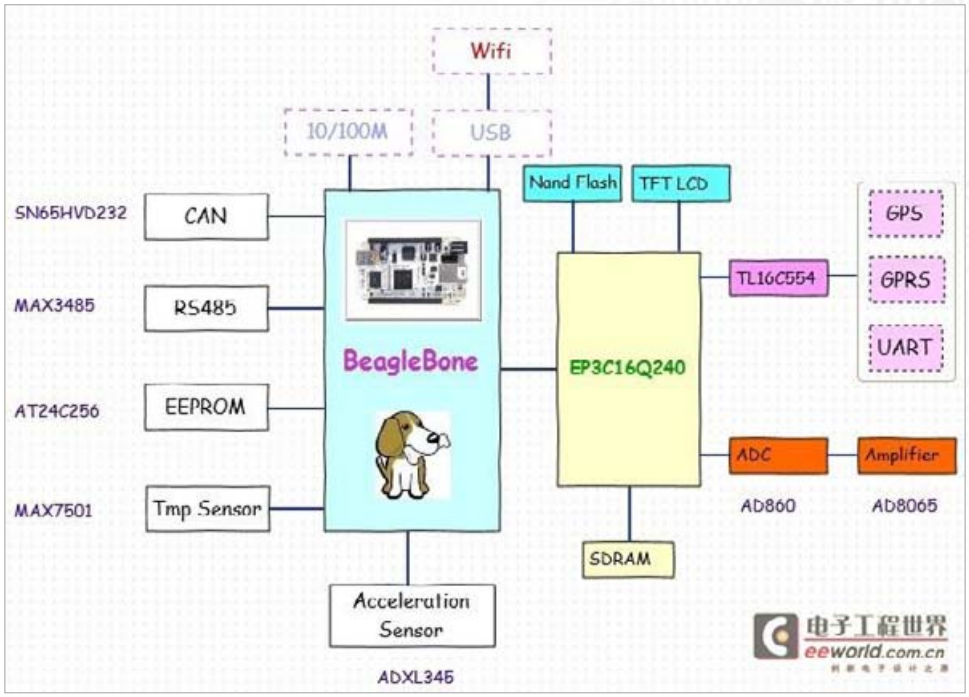


图 1-6 EE_BeagleBone_Cape 系统设计框图

外扩资源介绍：

1. EP3C16Q240q240+SDRAM 1 片
2. NAND Flash 1 片
3. LCD 1 路
4. CAN 1 路
5. RS485 1 路
6. EEPROM 1 片
7. 温度传感器 1 路
8. 加速度传感器 1 路
9. 串口扩展 TL16C554 1 片
10. 高速 ADC(ADS826) 1 路
11. 100M 网口、USB（BeagleBone 自带）。

EE_BeagleBone_Cape 设计来源：看到上面列的这些功能，非常的丰富，当然也会引起大伙的质疑。很多人都认为是简单的功能堆砌，又是一个大而全，华而不实的东西，仅仅是为了吸引大伙的眼球罢了。在这我将和大家分享我的设计初衷，我设计的这些功能都是从实际的项目中来的，只不过不是一个，而是很多个，有些是已经成功的实施的，我想换个平台；有的是在设计中失败了，我还想再尝试尝试，诸如此类。

项目来源 1：高速数据采集和传输

要求：信号频率 20M，采集时间 50ms，网络传输。首先想到的就是 ARM+FPGA 架构，FPGA 数据采集，ARM 数据传输，而且 ARM+FPGA 也是一种非常典型的架构，在我的工作范围内使用很频繁。从 AM3359 的手册上可以看到，GPMC 支持同步模式，最大主频 100M，这个很诱人，按 16bit 的数据宽度来算，也可以达到 1.6G 的带宽，即使打点折扣也很棒了!

当然会有人问，为什么不能用 ARM 直接驱动 ADC 采样，在我所接触的芯片中比如 S3C6410、AT91SAM9260、AT91SAM9263 等等，外设频率都很难达到 20M，所以使用 FPGA 是比较明智的选择。

我现在的这个载板里面有一个采样频率 60M 的 ADC(ADS826)，还有一片 FPGA，主要就是完成这个功能。

项目来源 2：数字示波器

有了高速 ADC，如果在加入个 LCD，就能很方便的实现示波器的功能。 那么就把 LCD 加进来吧，但是有个问题是为什么把 LCD 挂在 FPGA 上面呢？ 这个主要就是为了这个项目量身设计的。直接挂在 ARM 上当然毋庸置疑是可以驱动的，但是在数据采集和实施显示的时候，将消耗掉非常多的系统资源，所以在这个项目里面，必须要用 FPGA 来刷新 LCD，一方面把 ADC 采集的数据刷到 LCD 上，另一面处理 ARM 的控制信息，然后刷新到 LCD 上。其中的 SDRAM 也可以取到数据缓存的作用。 其次如果功能简单点，也可以去掉 ARM，单独有 FPGA + ADC +LCD 来实现简易示波器的功能。

项目来源 3：现场设备管理 –CAN

这是个实在的项目，已经基本完成。主设备节点 AT91SAM9260，从设备节点 LM3S8962；主从设备节点通过 CAN 总线连接。在这里我只想更换个 CPU 而已，因为项目都已经基本完成了，也无所谓好不好。不过也有一些理由吧：

1、AM3359 自带 CAN 控制器这是个不错的选择，AT91SAM9260 还需要外扩一个 MCP2515，不管从调试来说，还是从设计来说，AM3359 都是要简单一点。

2、其次是 AM3359 带两个网口，因为这个项目后续可能还会加个网络摄像头，这样我一块片子就搞定了，不需要在外扩网口，或者增加个路由器了。

项目来源 4: 串口扩展

苦逼啊! 这是来自一个失败的经历，我想验证一下而已。从手册上可以看到 AM3359 已经带了 6 个串口，据说还可以用 PRU 外扩 4 个串口，总过就能到到 10 个串口了，好丰富，所以串口扩展对于 AM3359 来说可能没有市场。我们本来打算用 S3C6410 扩展两个串口，第一次选用了一个 SPI 接口的扩展芯片，调试的时候总是不稳定；第二次换了个并口 TL16C554，据说这玩意很成熟，更加苦逼，只能发不能收，调试了很久无果；又折回头折腾 SPI 扩展的方案，居然峰回路转发现个 BUG，应用程序处理的时候有些问题! 第一个方案解决了问题，第二个方案当然就没人理睬了，呵呵 . 不过我还是打算深究一下其中的原因，顺便在 AM3359 上做个验证，说不准哪天就要用到了。

项目来源 5：电能数据采集

主要使用 ARM + LCD + FPGA + ADC，目前我正在使用，顺便验证一下新平台，我当前采用的架构是 AT91SAM9263 挂载 LCD，通过外部总线和 FPGA 通信，FPGA 控制 ADC 进行电能数据的采样。

至于 EEPROM 可以存放 ID，BeagleBone 的很多其他载板 ID 就是存放在 EEPROM 中 温度传感器只不过是通用设备，可以检测环境的温度 加速度传感器，很多设计中作为防拆设计，其他的在手机中还是很常用的 GPS 就更不用说了，大家都很清楚用途。其他的就不再多说了，呵呵，有心的人自己发掘吧。

EE_BeagleBone_Cape 设计点滴

1) 为了节约空间，FPGA 的 AS 接口没有画，如果要配置 EPCS，只能用 jtag 口下载 jic 文件了

2) 部分接口不是直接连在 BeagleBone 上的，采用独立的模块，可以使用杜邦线连接。比如 RS232,CAN,RS485,GPS 等等

3) 电源部分貌似不能同时上电，同时上电的话 FPGA 的管脚状态将影响 AM3359 的 SYSBOOT 引脚，从而导致 AM3359 无法正常启动

4) LCD 部分可以兼容群创的 4.3 寸、7 寸、10 寸屏，已经实测过。

EE_BeagleBone_Cape 各功能原理图设计

1、BeagleBone 外扩接口

主要通过 P8 和 FPGA 相连，P9 有一部分接口也接在 FPGA 上。

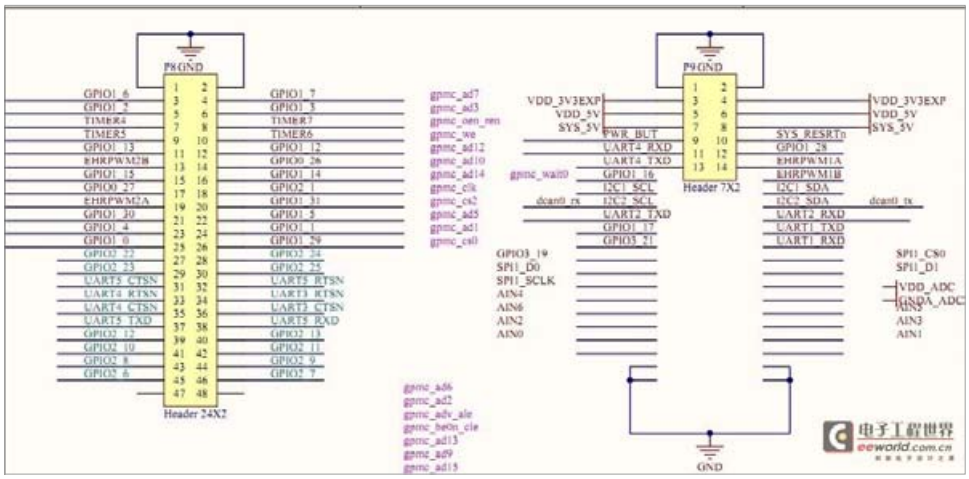


图 1-7 BeagleBone 外扩接口

2、EP3C16 + SDRAM 常规设计，没啥好说的，只不过我把 AS 的接口去掉了，这样在配置 EPCS 芯片的时候，需要转换成 JIC 的文件格式，通过 JTAG 下载。

3、NAND 这里做了一些兼容设计，可以支持不同类型的 NAND Flash。

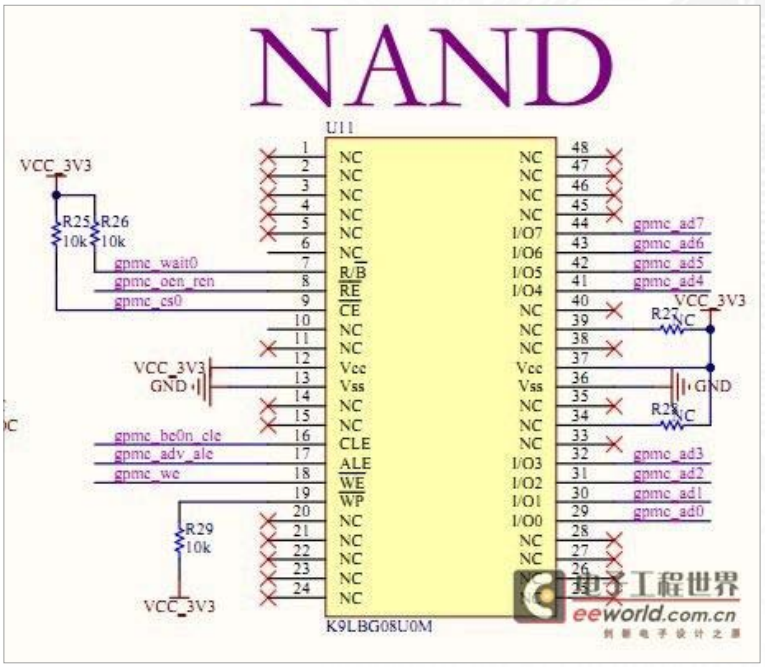


图 1-8 NAND

4、TFT LCD 我使用的群创的屏，4.3 寸、7 寸、10 寸的我都试过，没有问题。

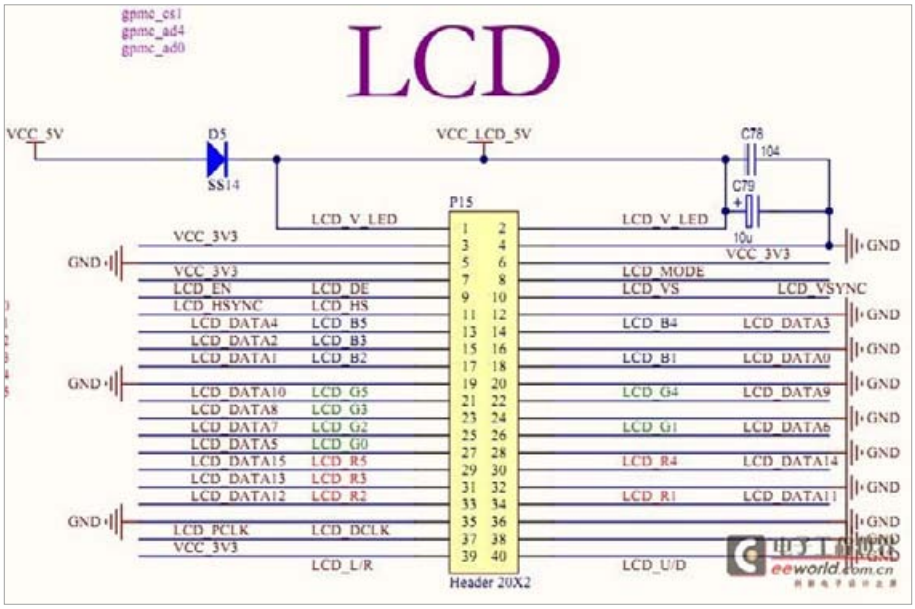


图 1-9 TFT LCD

5、CAN 这里也是可以和 TJA1050 之类兼容，我手头正好有 SN65HVD232，所以就采用这个。

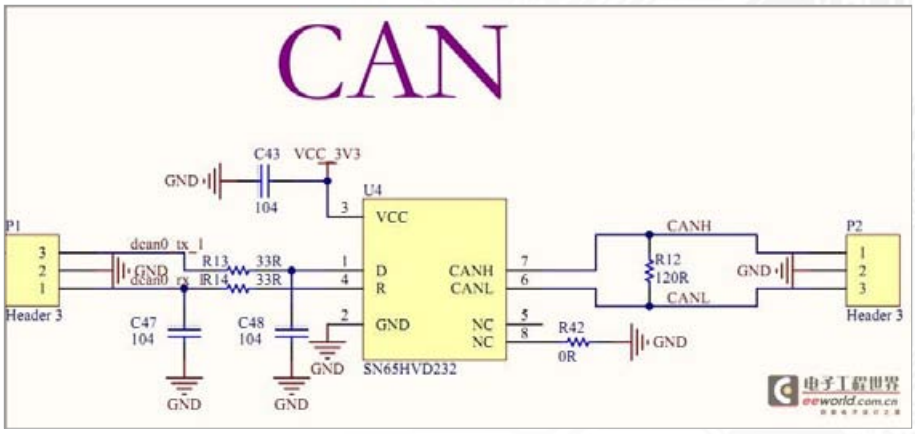


图 1-10 CAN

6、RS485 常用芯片，我实际焊接的是 MAX485。

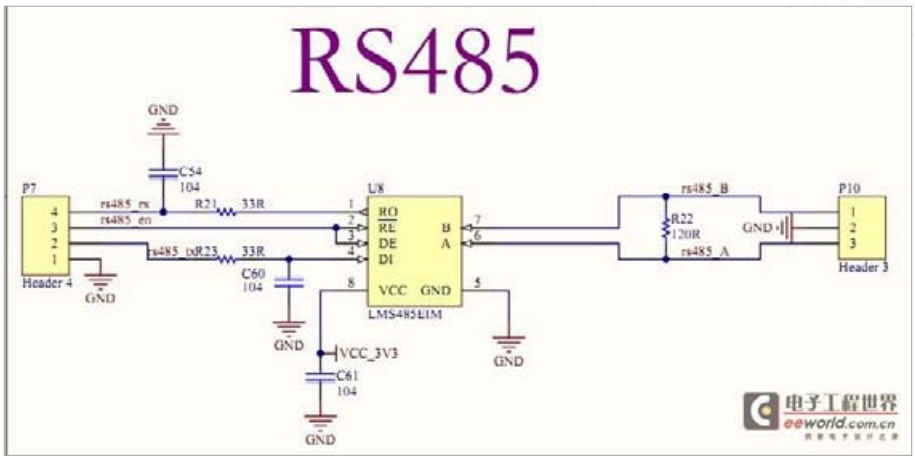


图 1-11 MAX485

7、I2C 接口

Max7501 是因为我手头有，就选用了这个，也做了兼容设计，可以采用 LM75 等；貌似目前 linux 还不支持 Max7501，苦逼啊。

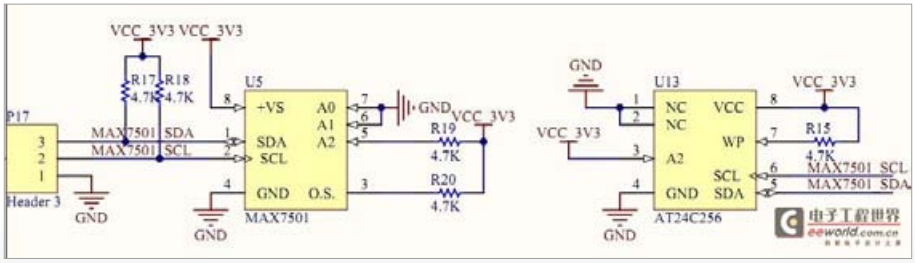


图 1-12 I2C 接口

8、加速度传感器 焊接很麻烦哦，我没打算调试了。

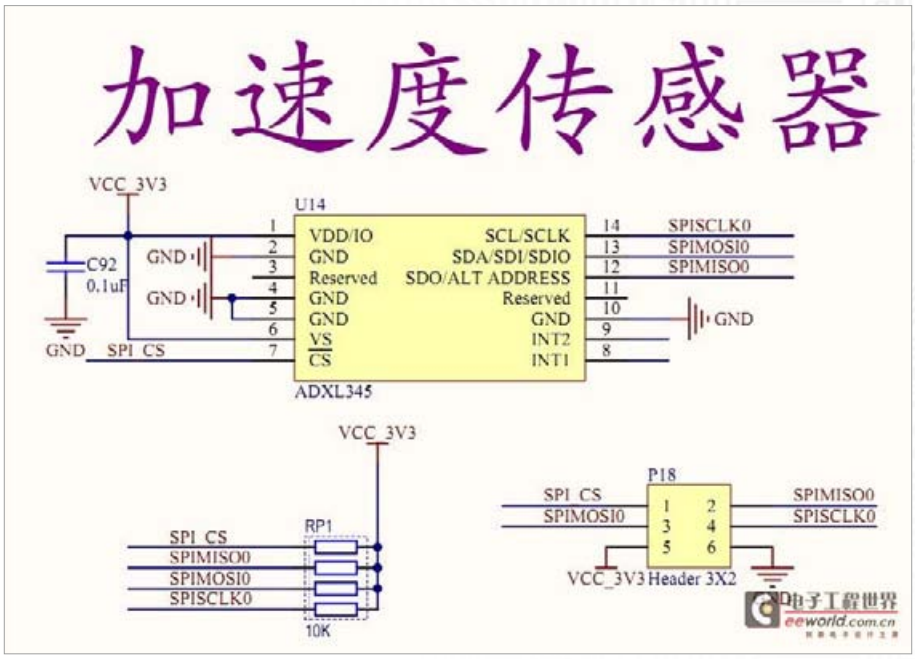


图 1-13 加速度传感器

9、GPS 天宝的典型设计，直接把管脚引出就行了，使用的是 3.3V TTL 电平。

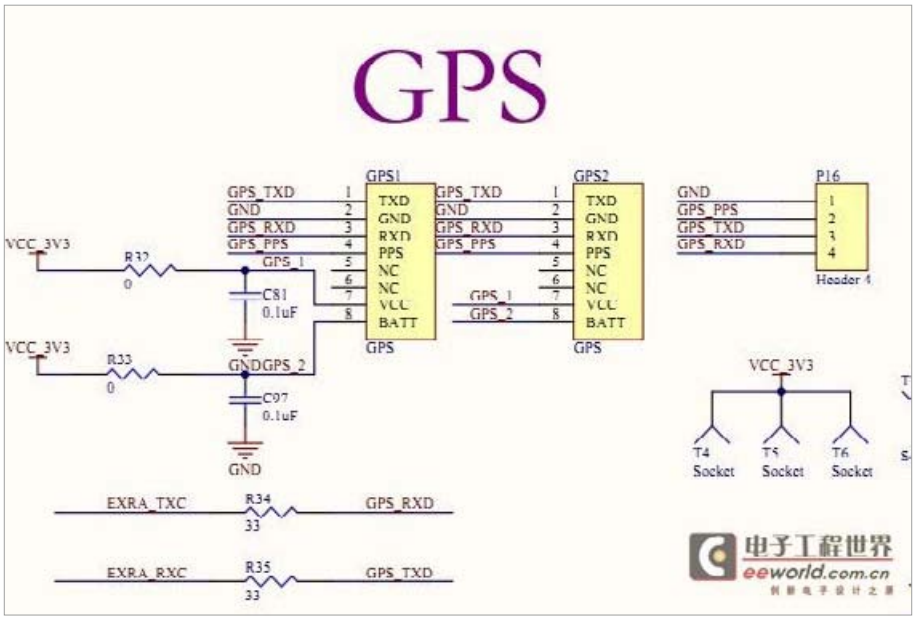


图 1-14 GPS

10、ADC 可以参考数据手册中的典型设计。

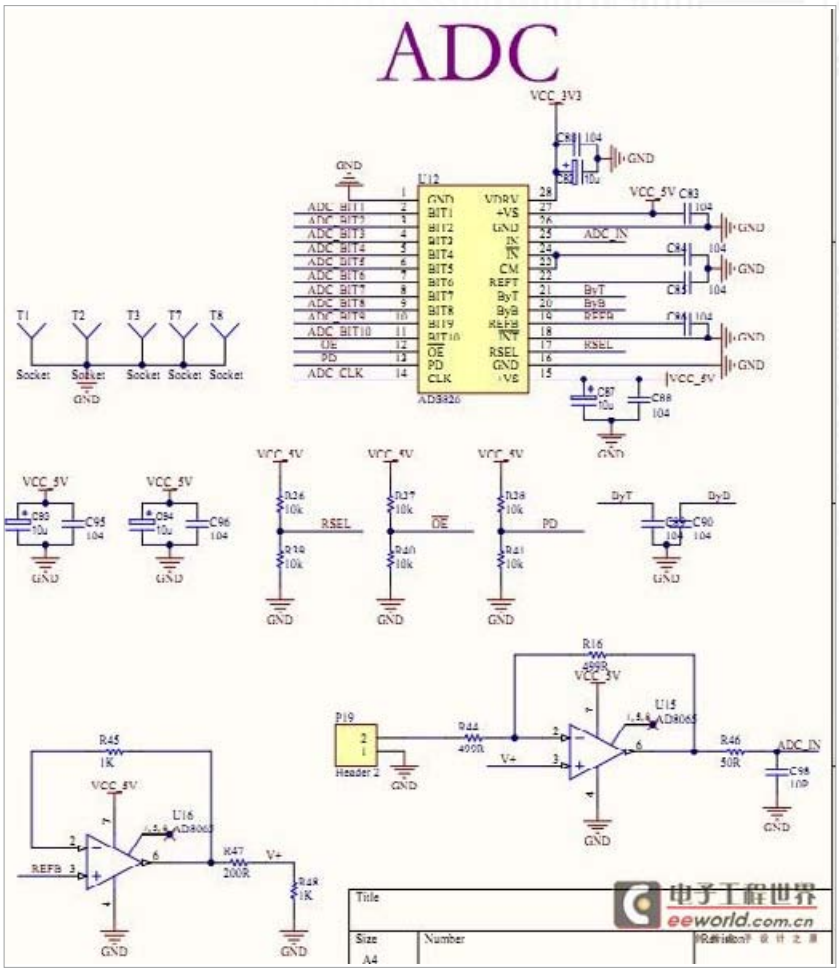


图 1-15 ADC

11、串口扩展

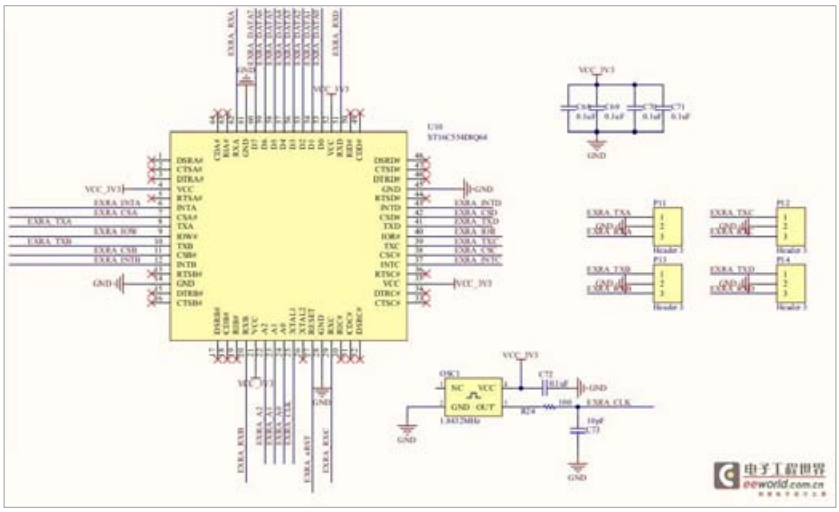


图 1-16 串口扩展

实际效果图

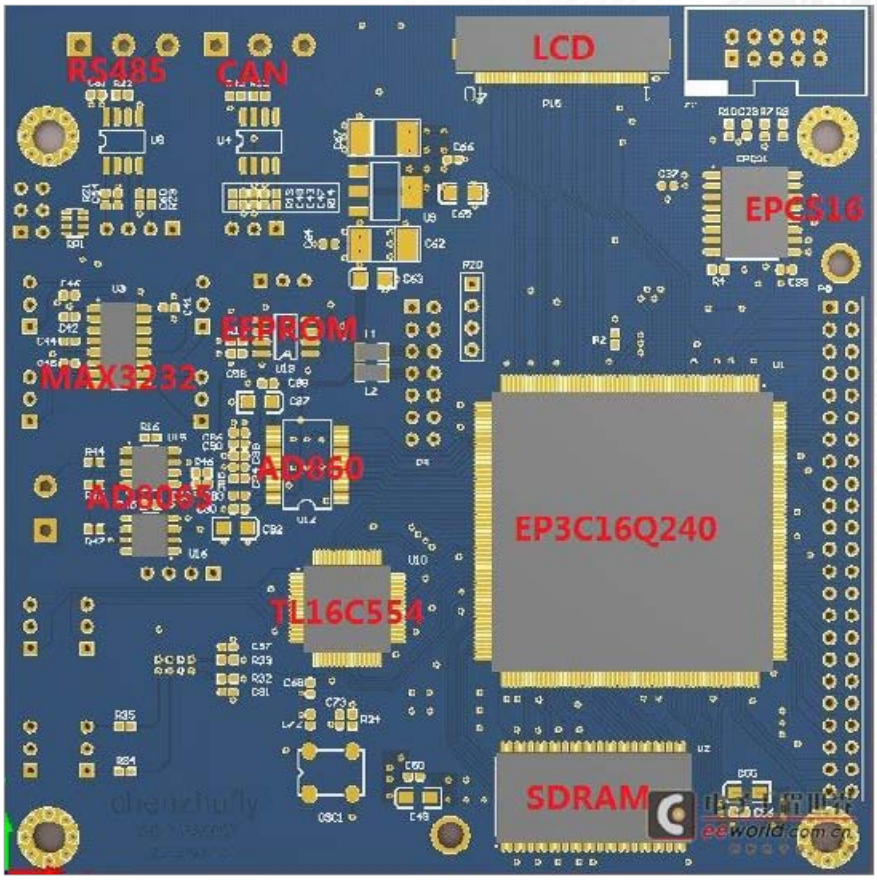


图 1-17 PCB 效果图

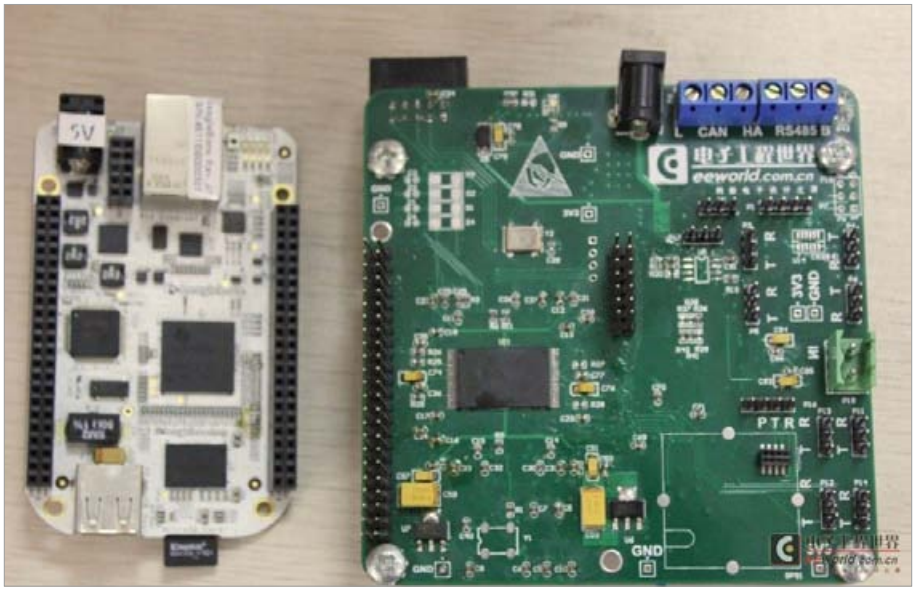


图 1-18 焊接效果图

设计资料：>> 点击下载原理图链接

点击查看详情：>> EE_BeagleBone_Cape 硬件系统设计

1.2 BB-Black 开发环境的搭建

1.2.1 Linux 的安装

1、安装 VMware 7.13

2、下载 Ubuntu 10.4：看官方笔记都提到 LTS 版本，如是下载了个，花了几十个小时，>> 下载地址

3、安装 Ubuntu 10.4 ，打开 VMware7.13

开始从 VMware 选择安装虚拟机，选择光盘映像目录，然后开始。

安装完成后没图形界面，用 startx 也启动不了。提示了一个安装图形界面的命令，于是继续。

执行命令：

sudo apt-get install xinit

安装完成，重启，运行 startx，终端由黑色界面变成白底黑字。出现 X 型的鼠标指针。这不是我需要的，还是需要安装 gnome 的桌面套件。

安装命令为：

sudo apt-get install ubuntu-desktop

执行之后会提示你需要安装的软件包数量及大小，Ubuntu Server 的图形界面需要下载 850 个软件包，安装后需要占用 2G 多的硬盘空间，另外还有 KDE、XFCE 桌面环境这两种图形桌面环境可选，安装命令分别是：

```
sudo apt-get install kubuntu-desktop
sudo apt-get install xubuntu-desktop
sudo apt-get install ubuntu-desktop/kubuntu-desktop/xubuntu-desktop
```

安装好之后 startx 进入图形界面，和 Ubuntu Desktop 的界面一样，连 OpenOffice 都一起安装上了。

上面是安装服务器版本的 ubuntu（就是那个 LTS），有些麻烦。所以我又改回去，把上面安装的一部分一起连映像一起删了（映像就是好，想删就删）。重新安装桌面版本，如是：

下载：ubuntu-10.04-desktop-i386.iso

打开 VMware7.13

开始从 VMware 选择安装虚拟机，选择光盘映像目录里的 ubuntu-10.04-desktop-i386.iso，然后开始安装。

注意这里还要填写一些信息。有系统名、用户名、密码（这个是你建立的登录用户信息）等

然后出现下面的画面。

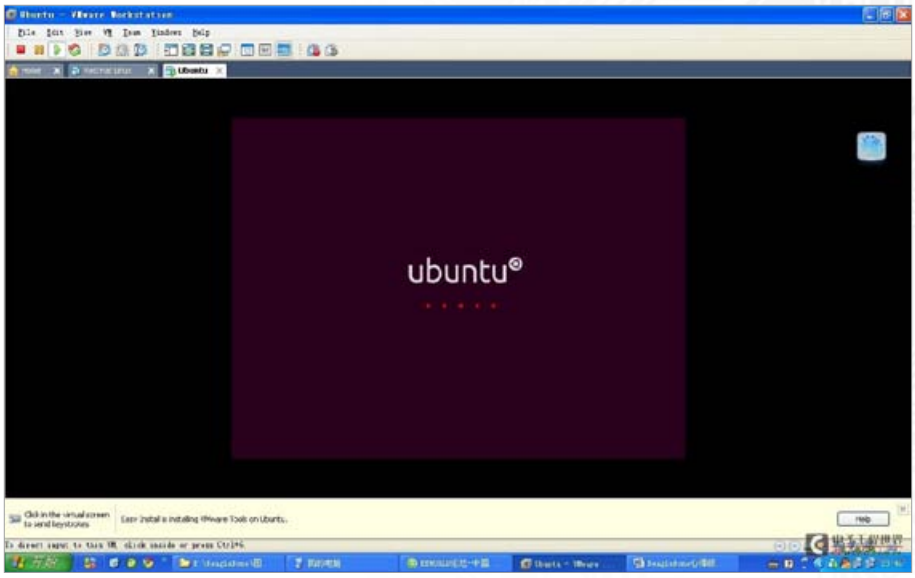


图 1-19

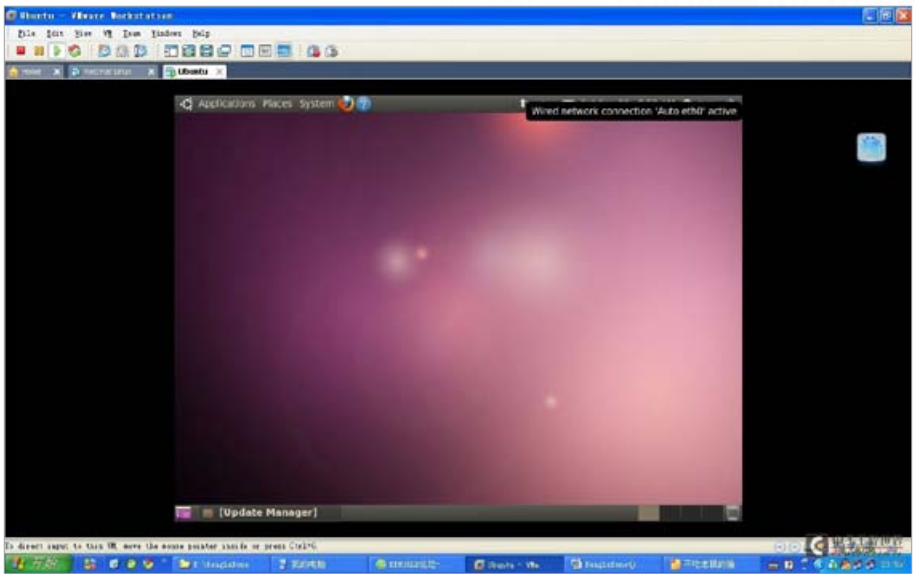


图 1-20

安装完成后输入密码就进入系统了。

VMware tools

选择菜单中 reinstall vmvare tools(vmvare 7.XX 在这里应该是安装好了 VMware tools 继续是为了试下。

如果是第一次安装第一个虚拟机就是 install VMware tools) , 如是 VMware tools 盘符出现在桌面 (这里就是 VMware tools 的文件夹)。

进入终端 (应用 applications 里的….)

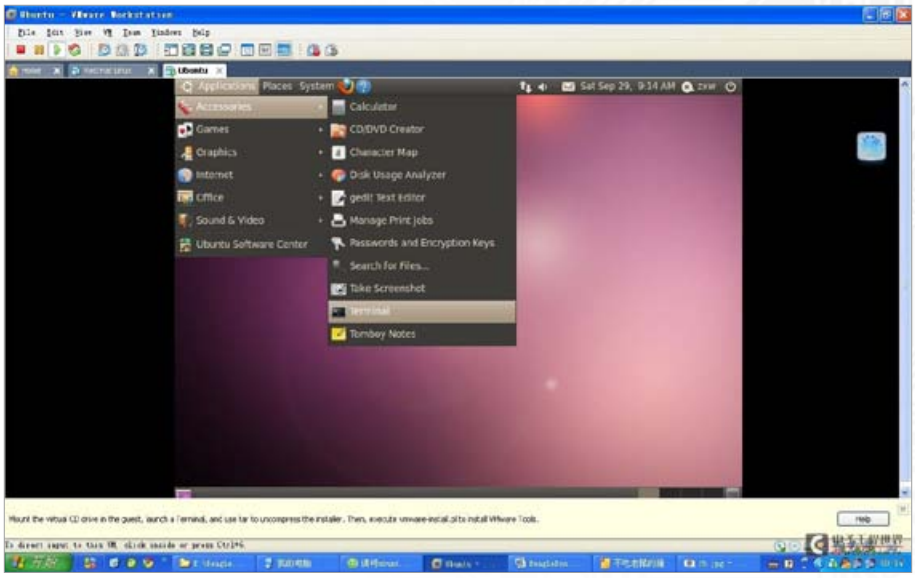


图 1-21

挂载完之后我们就可以开始安装来，但在安装前：

1, 要先获得 root 的权限，否则你无法复制，解压及安装，获取过 root 的可以忽略这一操作。首先我们设置 root 的密码，打开终端，输入“sudo passwd root”，给 root 设置一个密码。设置完密码之后，接着输入“sudo su”按下回车键，这个时候我建立的用户就获取到 root 的权限了。

2, 要把 media 目录里看到 VMware tools 文件夹里的 Vm…文件拷贝到其他地方，如 tmp 目录，media 这个目录是只读的。没法子建立目录。

然后解压：./tar zxvf VMwareTools-8.4.5-324285.tar.gz

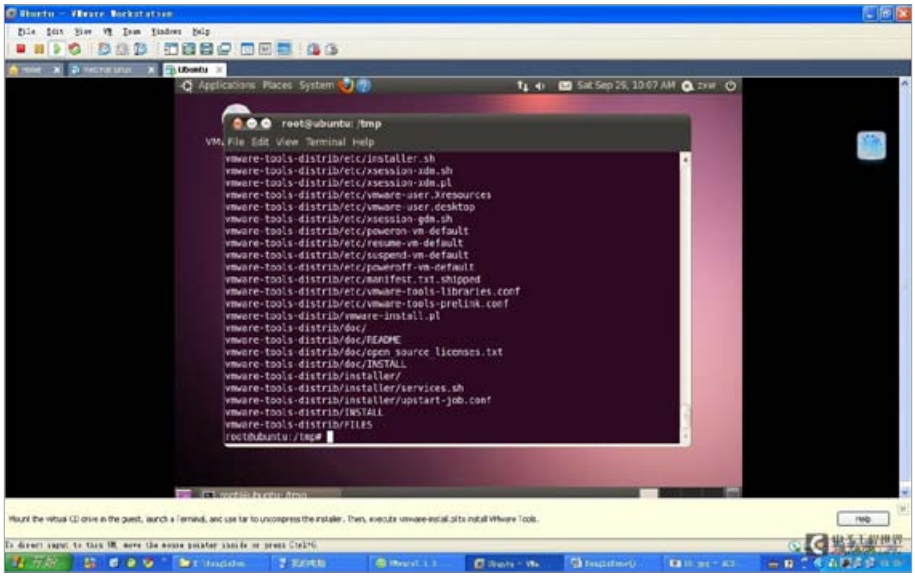


图 1-22

这个时候在该目录下就出现了 VMware-tools-distrib 目录，我们就可以开始安装

VMware tool 了。(注意使用 table 键，少敲按键哦）

进入 VMware-tools-distrib 的目录，执行 ./vMware-install.pl，桌面的安装已经安装了 VMware-tools，因为安装时提示卸载老的 VMware-tools

我这里就选择反安装后重装。

一路回车，耐心等待屏幕刷新，直到最后，vm 安装完毕。

注意：

1. 必须切换到 root 用户 切换的方法 su 设置密码的上面说了。

2. 安装必须进行 VMware-tools 的解包 (拷贝到另外目录，那个目录是只读的）要不然不会出现这个 VMware-tools-distrib 的目录。

3，好像桌面的安装已经安装了 VMware-tools，因为安装时提示卸载老的 VMware-tools，建立一个文件夹用于 ubuntu 和 WINXP 交换文件

设置目录共享文件夹目录：

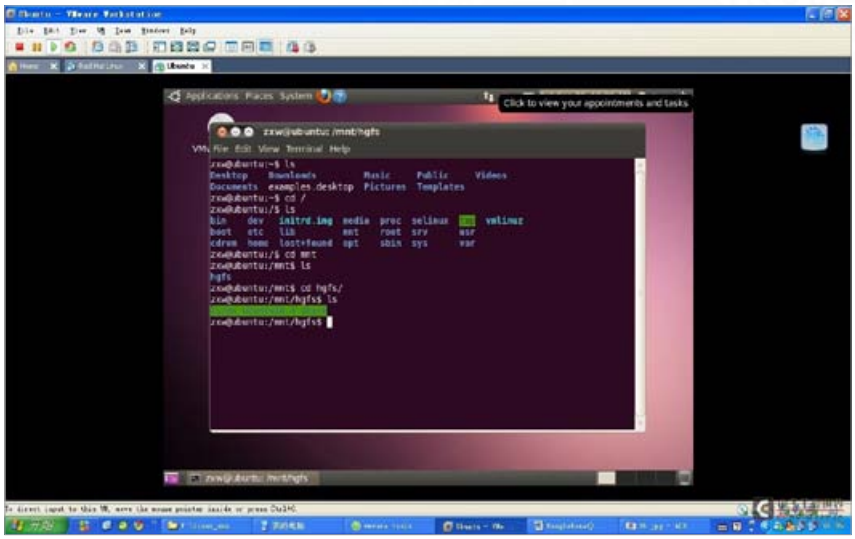


图 1-23

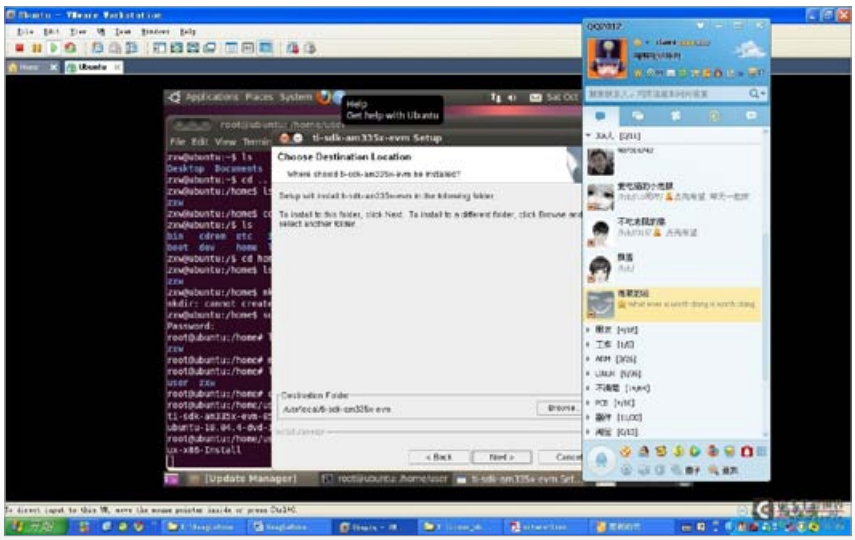


图 1-24

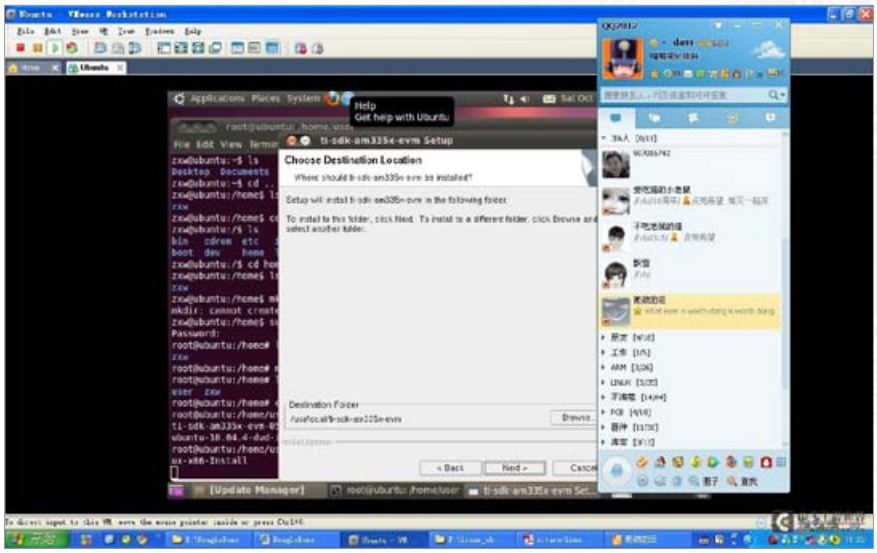

Choose Destination Location
Destination Folder
< Back
Next >
[Update Manager]

图 1-25

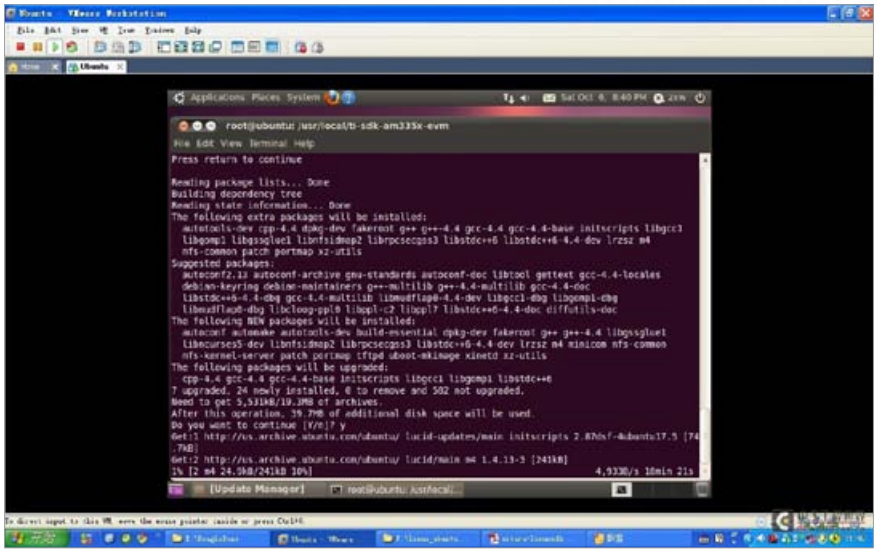

root@ubuntu: /usr/local/ti-sdk-am335x-evm
Press return to continue
[Update Manager]

图 1-26

1.2.2 Linux 编译环境的搭建

TI SDK 安装包和 CCS5 的安装

1、下载安装包

名字为：CCS5.2.1.00018_linux.tar.gz
ti-sdk-am335x-evm-05.05.00.00-linux-x86-Instal
在 TI 网站有可以找到

2、敲入 su 和密码，转为 root 用户
把它们两个文件通过共享目录拷贝到 LINUX 的 usr/local

3、用 tar xzf CCS5.2.1.00018_linux.tar.gz 解压 CCS5

4、运行 TI SDK 包：
./ ti-sdk-am335x-evm-05.05.00.00-linux-x86-Install

5、运行过程提示目录，默认

6、运行过程提示 CC5 是否安装，选择安装（默认安装）

7、经过一阵，会提示完成安装，中间提示找不到 CCS，CCS 没装好，不知道为啥找不到。

8、进入 ti-dfk-sm335x-evm
随后运行：./setup.sh 进行相关配置

9、只好使用命令行安装 CCS5.2

进入 CCS5.2.1.00018_linux 目录
运行：
使用 ccs_install.sh 安装
./ccs_install.sh $PWD
也可以用以下命令：
cd CCS
./ccs_setup*.bin --setupfile ccs_installini.xml

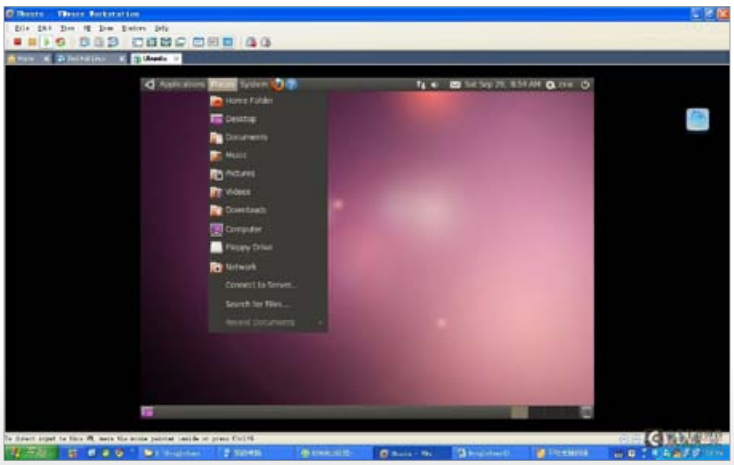


图 1-27

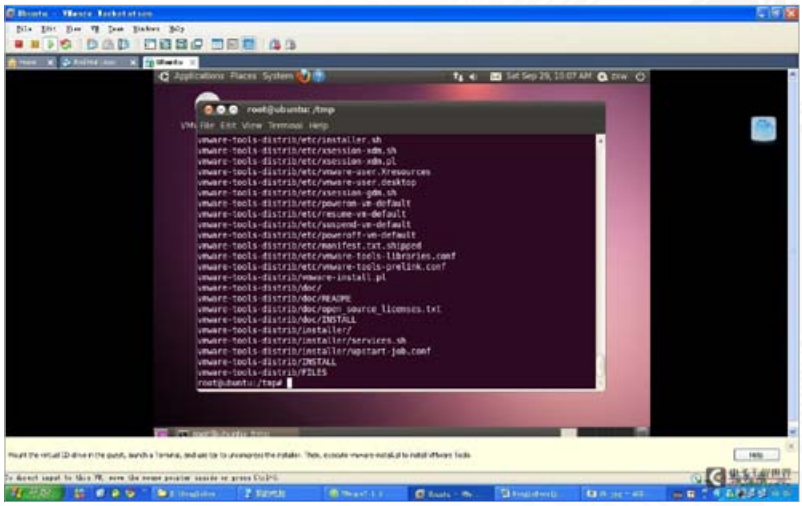


图 1-28

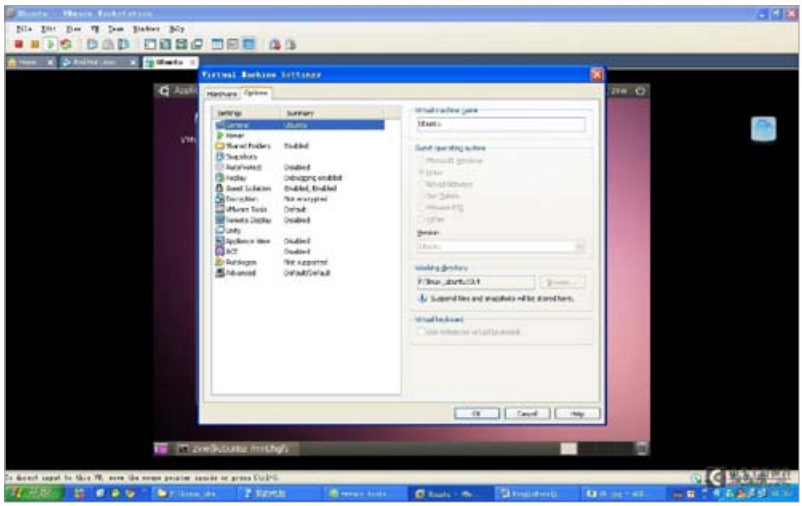


图 1-29

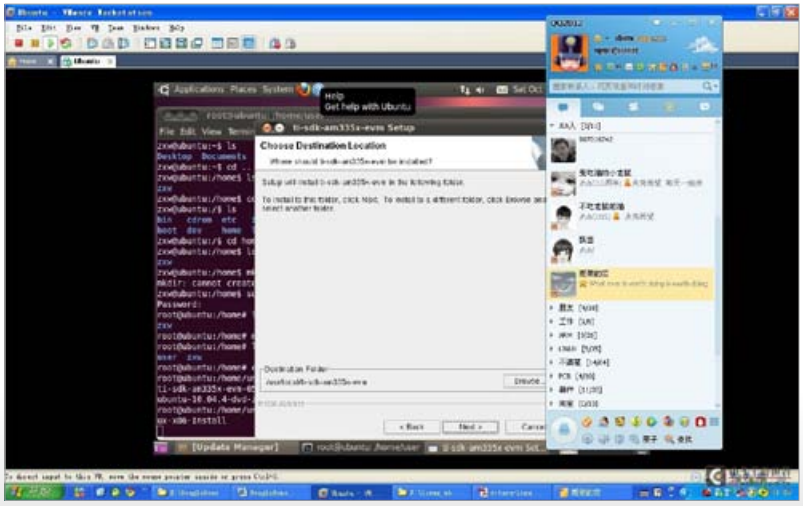


图 1-30

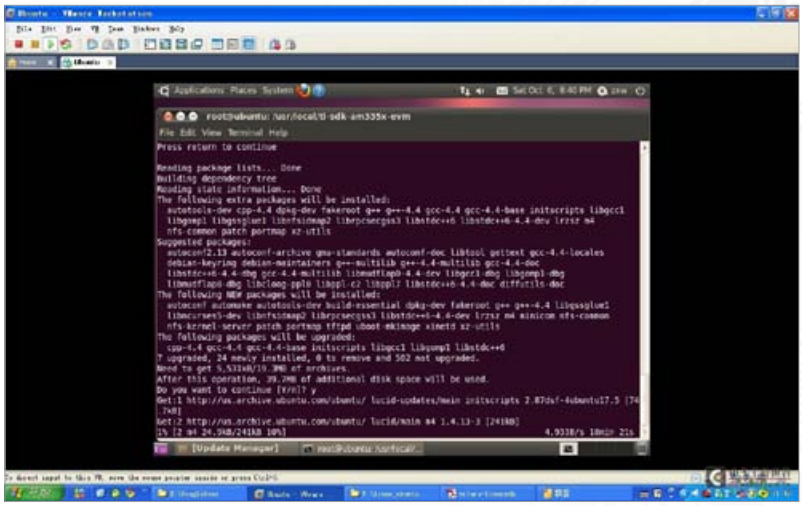


图 1-31

点击查看详情：>>BeagleBone 开发环境安装之路

1.2.3 开启 tftp、NFS、Samba 服务

1、装好 Linux 后，开启相应的服务及软件，包括：tftp，NFS，Samba，使用 apt-get install 安装即可。

安装 ti-sdk-am335x-evm-05.03.02.00-Linux-x86-Install 后，可运行其中的 setup.sh。

装好后即设置交叉编译链的环境变量：vi ~/.bashrc 后面添加 export PATH=$PATH:/home/bblpp/ti-sdk-am335x-evm-05.03.02.00/linux-devkit/bin 运行 . ~/.bashrc ，然后输入 arm- 就可以得到 arm-linux 交叉编译链啦。

2、/etc/xinetd.d/tftp 为 tftp 的配置文件，默认配置为 /tftpboot 目录。

BeagleBone 开发板中已经支持 tftp 命令了，所以网络配置好后就可以使用 tftp 来 down 一应用程序

tftp 192.168.1.16 -g -r main
Busybox 中 tftp 命令的用法
tftp [option] ... host [port]
如果要下载或上传文件的话是一定要用这些 option 的。
-g 表示下载文件 (get)
-p 表示上传文件 (put)
-l 表示本地文件名 (local file)
-r 表示远程主机的文件名 (remote file)
例如，要从远程主机 192.168.1.2 上下载 embedexpert，则应输入以下命令
tftp 192.168.1.2 -g -r embedexpert

3、NFS： PC 端装好 NFS 服务，/etc/init.d/nfs-kernel-server

编辑 /etc/exports 添加：/work/nfsroot *(rw,sync,no_root_squash) 目录。

BeagleBone 板子 mount -t nfs 的时候貌似提示不知道的文件系统 无法挂载，想起之前在 DM3730 的时候也遇到过这样的问题，内核通过 NFS 启动，但是启动后无法挂载 NFS，参考英码的说明文档，用 opkg 安装了两个文件，就 ok 了。

>> 下载相关文件

opkg install 先 portmap，再 nfs。
挂载的时候使用命令：
mount -t nfs 192.168.1.85:/work /mnt/nfs -o nolock,proto=tcp,nfsvers=3
貌似后面不输入那些参数依然无法挂载，也可能是我在局域网中的原因。
4、配置 samba：

apt-get install samba 后，增加一个 samba 的用户名，也可以是登录的用户名。
sudo smbpasswd -a bblpp 将用户加到 samba 用户组中，设置密码后，编辑 /etc/samba/sab.conf 文件，找到 security 项，将其设置为 user，security = user，确定下面这

项没有被注释：

passdb backend = tdbsam

然后在文件最后面增加目录相关设置。可参考如下。

>> 下载相关文件

重启 samba 服务：service smbd restart（有的是 service samba restart）

5、至此可 hello world 啦：

```
#include "stdio.h"
int main()
{
    printf("hello world!\n");
    return 1;
}
```

arm-arago-linux-gnueabi-gcc -o main main.c 生成 main

然后使用 tftp 命令：tftp 192.168.1.16 -g -r main 即可 down 进开发板中，增加可执行权限：+x 即可

图 1-32

也或者拷入 NFS 目录下，确保挂载成功，直接执行

图 1-33

点击查看详情：>> 开启 tftp，NFS，Samba 服务

1.2.4 uboot 、内核和文件系统的编译

官网如下：http://beagleboard.org/static/BeagleBone/latest/README.htm

目前使用 TI 提供的 linux 开发环境作为开发平台。需要有一台运行能运行 linux 环境的计算机。

首先需要到 TI 官网下载最新的 sdk：http://software-dl.ti.com/dsps/dsps_public_sw/am_bu/sdk/AM335xSDK/latest/index_FDS.html 下 载 ti-sdk-am335x-evm-05.05.00.00-Linux-x86-Install 然后运行这个文件：

```
1 sudo chmod a+x ti-sdk-am335x-evm-05.04.01.00-Linux-x86-Install
2 ./ti-sdk-am335x-evm-05.04.01.00-Linux-x86-Install
```

按提示一路执行下来即可。

安装后找到 ti-sdk-am335x-evm-05.04.01.00 文件夹，运行 setup.sh, 如果你用的系统不是 ubuntun 10.04lts，它会提示环境不支持安装，我用的系统是 ubuntu11.04，结果不通过，唯有分析一下 setup.sh 干了些啥事。

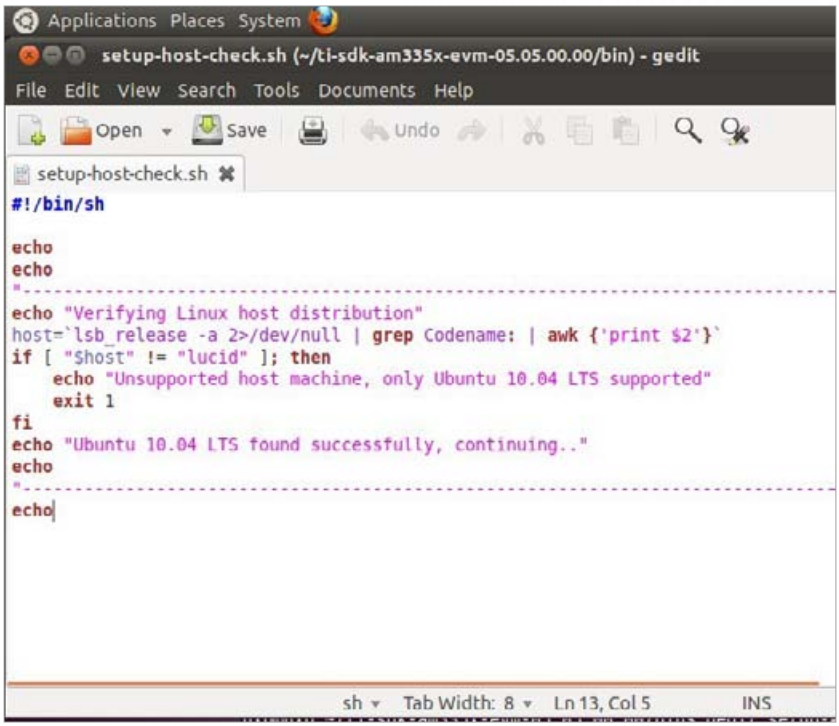


图 1-34

到 bin 目录下 setup-host-check.sh，如果不想换系统（ubuntun 10.04lts），可以尝试修改一下脚本，我把它改成了下面那样，哈哈。

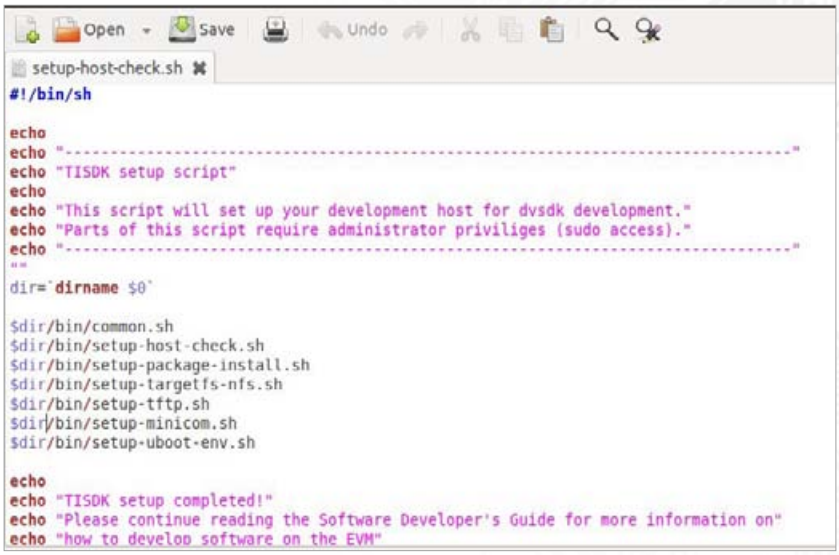


图 1-35

之后，再用 运行命令 sudo ./setup.sh 一路回车即可。

这个 setup.sh 的脚本做了些什么东西呢?

看源码可以知道，它为我们安装了 nfs，tftp，minicom，uboot 环境。完成了上述工作后，让我们看看 TI 为我们提供了什么东西吧。

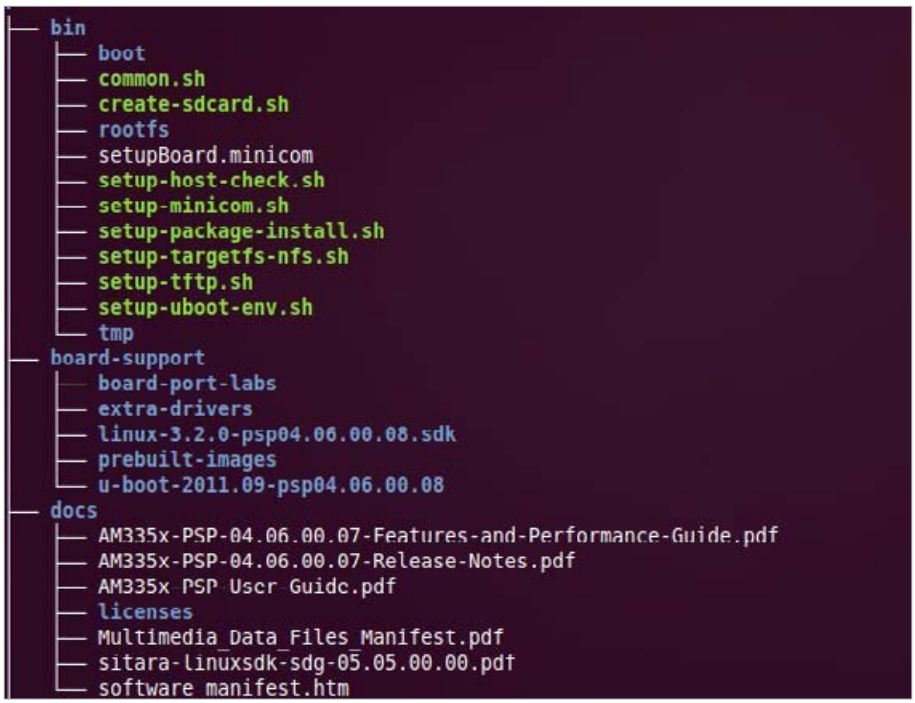


图 1-36

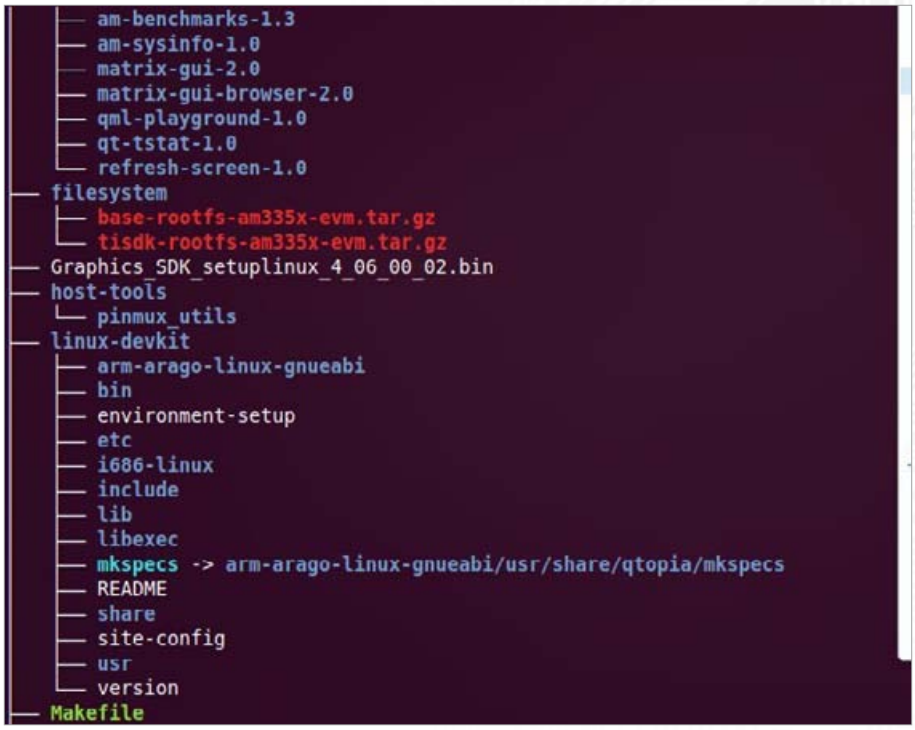


图 1-37

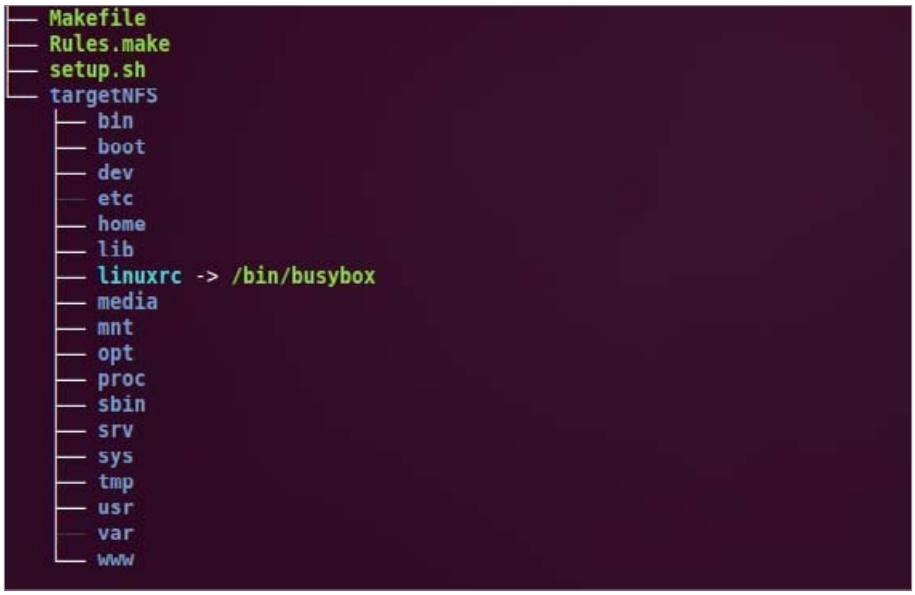


图 1-38

board-support 目录放了 linux 源码，u-boot 源码，额外驱动，和已经编译好的 u-boot，uImage，MLO（MLO 是用来从 micro SD 里面引导 u-boot 用的）。

inux-devkit 是开发用的一些工具，交叉工具链在其 bin 目录，最好在 path 里面加入其路径。

在 /etc/environment 中修改

如图：

图 1-39

之后重启一下虚拟机新的 PATH 即可生效。

有了交叉工具链，就可以开始编译 uboot 和 kernel 了。

编译 uboot 时，输入如下指令：

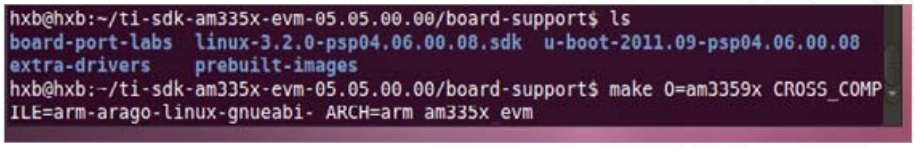


图 1-40

等编译完成时，在 am3359x下可以得到 uboot.img 和 MLO。

接下来就是编译内核：

生成 .config 文件，可以使用如下命令：

图 1-41

配置内核：

图 1-42

开始编译内核，生成 uImage：

图 1-43

编译内核时，需要安装有库 ncurses library

Ubuntu下可以如下安装：

Sudo apt-get install libncurses5-dev

制作 uImage 也需要 mkimage 这个工具，ubuntu下安装方法：

Sudo apt-get install uboot-mkimage

编译好 uImage 之后，到了生成模块 ko：

图 1-44

生成的 ko 分散在不同的地方，可以用如下命令复制到文件系统相应的地方：

图 1-45

至此，需要的 uboot 和 kerne 已经编译完成，接下来的就是制作我们的 TF 卡啦。

开始

我准备的环境：

1、虚拟机 Vmware8
2、ubuntu11.04
3、安装好的 ti-sdk-am335x-evm-05.04.01.00
4、一张 4G 的 TF 卡

格式化 TF 卡

插上 TF 卡时，通常三 FAT32 格式的，但在 linux 下，我们需要一个 ext3 文件系统，又因为 BeagleBone 没有 nandflash，只能把文件系统放在 tf 卡上，所以我们需要格式化 tf 卡，让其拥有一个 ext3 的文件系统。

操作方法：
转到 ti-sdk-am335x-evm-05.04.01.00/bin：
然后如图：

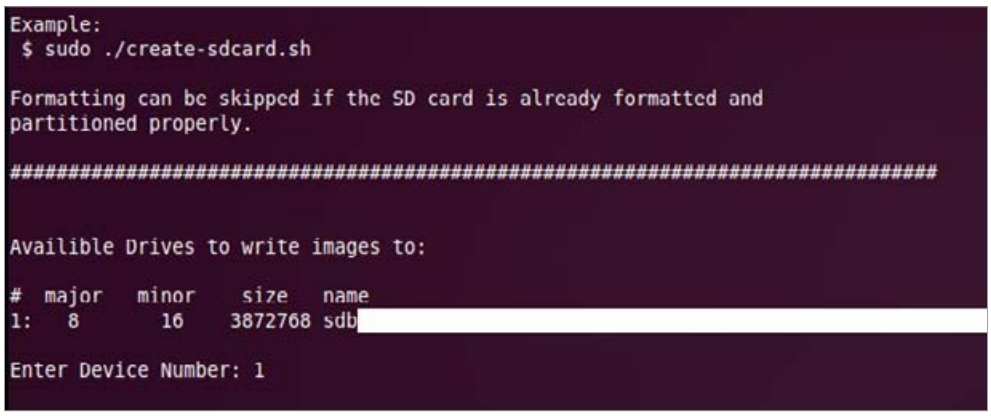


图 1-46

接下来选择分区数目，我选择了两个分区，一个 boot 的分区，用于存放 uboot 和 kernel 的映像文件，另一个 rootfs 分区，用于存放根文件系统。

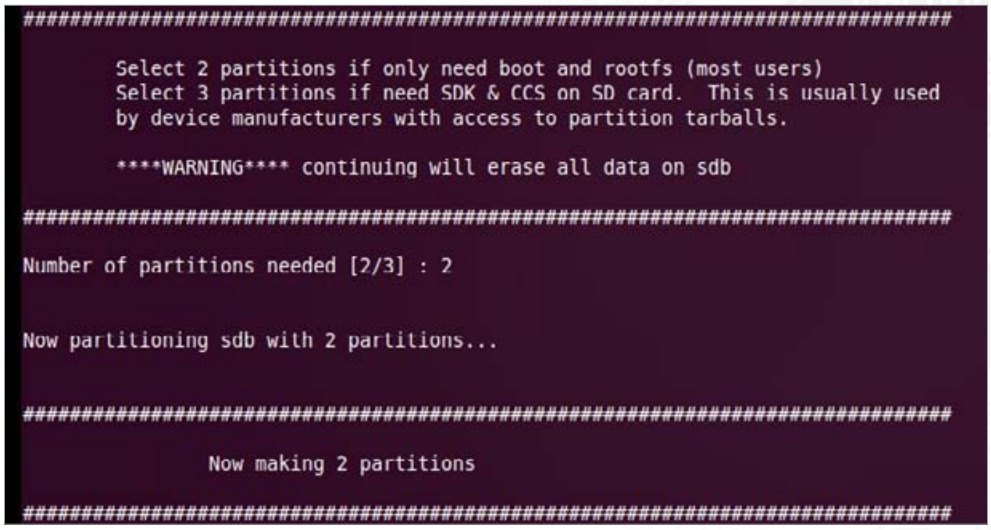


图 1-47

然后继续执行：

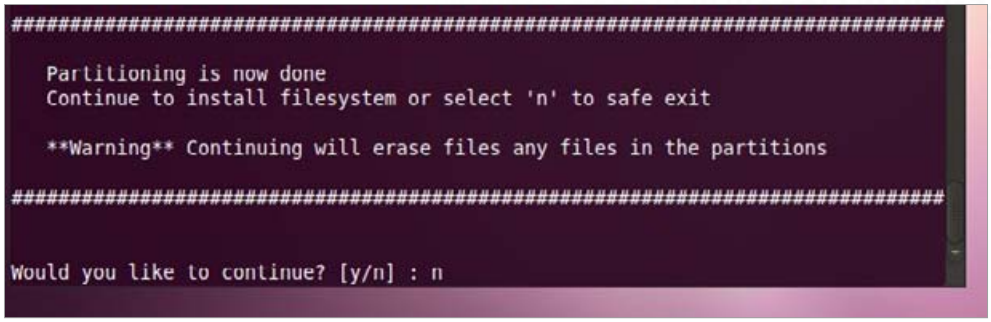


图 1-48

选择 n 退出脚本，选择自己构建文件系统。

1、先到 uboot 目录下复制文件，如图：

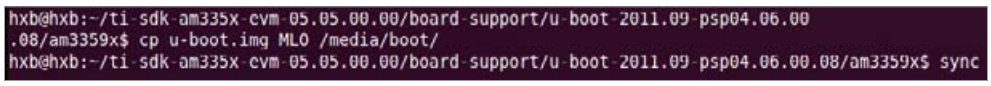


图 1-49

2、再到 linux 目录复制 kernel。

图 1-50

3、解压 TI 为我们制作好的文件系统到 tf 卡的 rootfs 分区：

图 1-51

4、把之前编译好的 ko 放到文件系统中：

```
hxb@hxb:~/ti-sdk-am335x-evm-05.05.00.00/board-support/linux-3.2.0-psp04.06.00.08
.sdk$ sudo make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- INSTALL_MOD_PATH
=/media/rootfs modules_install
```

图 1-52

至此，我们的 tf 算是制作完成了，卸载后，插入 BeagleBone 中，看运行效果吧：）

上点后，倒计时结束后，uboot 开始引导 linux，过了大概 10 秒钟，就可以看到登陆界面：

图 1-53

利用做好的系统，利用手头的 GPS 模块做了个串口的实验，

GPS －> UART4

UART4 对应 /dev/ttyO4. 硬件连接如图：

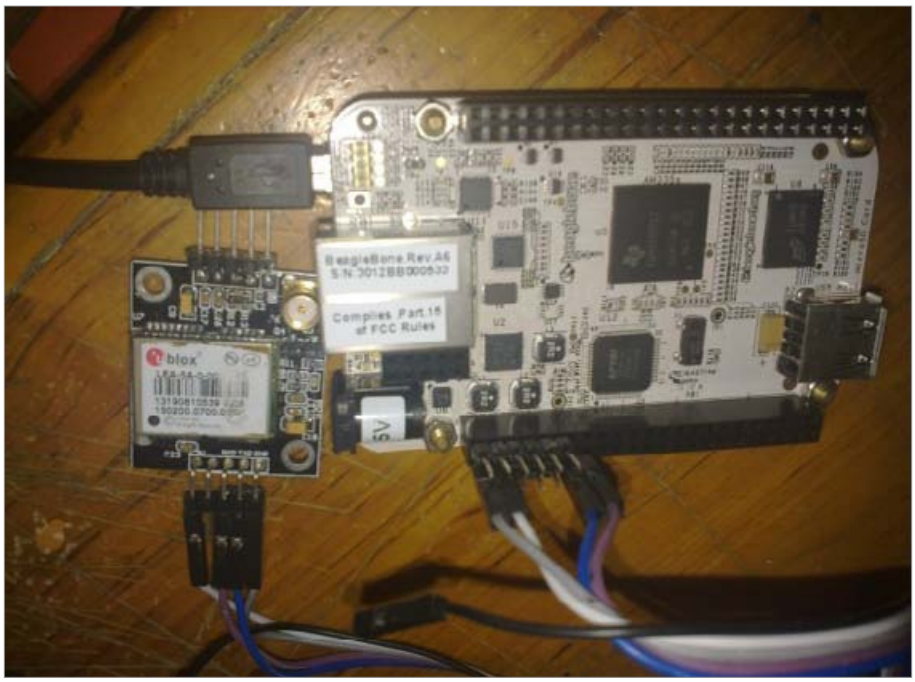


图 1-54

把编译好的测试程序下载到 rootfs 中，运行 bone，看到：

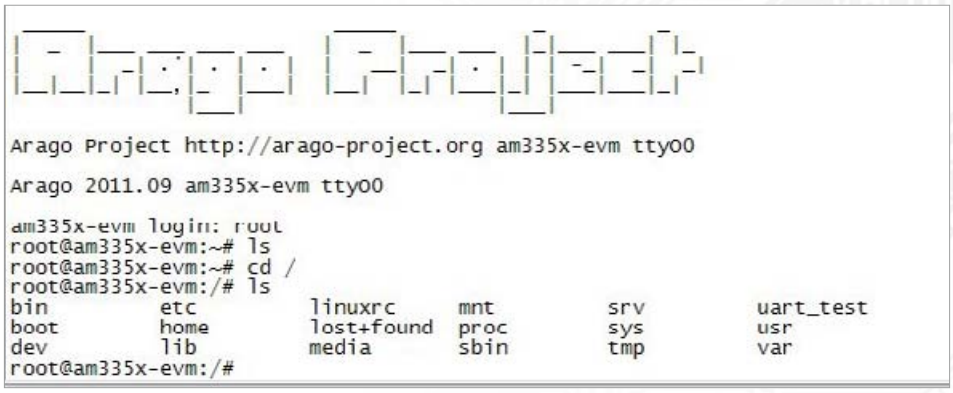


图 1-55

然后 输入命令 ./uart_test 即开始接受 GPS 数据，并保存在指定的文件中。根据提示输入“quit”即可退出程序。 在后面附上源程序，有兴趣的可以试一下 :) 。

点击查看详情：>> 教你一步步为 beaglebone 制作 linux 系统

1.2.5 用串口登录 BeagleBone Black、用 usb 共享电脑网络、内核模块的本地编译

串口连接 BBB

使用 usb 线可以连接 BBB 和电脑，用 ssh 就可以登录 BBB 来进行操作。但有时候万一系统配置出现故障，或 ssh 用不了了，那就只能用串口连接了。首先要有一个串口转 USB 模块，用 GND, TXD, RXD 这三根线连接到板子上。具体连接方法见下图。

Table 13. J1 Serial Header Pins

PIN NUMBER	SIGNAL
1	Ground
4	Receive
5	Transmit

Figure 51 shows the pin location on the board.

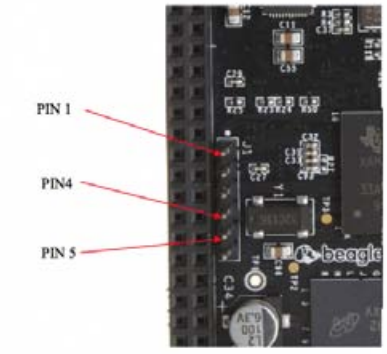


Figure 53. Serial Header

图 1-56

理论上串口是随时都可以连接的，但我为了看到更全的输出信息，我在给 BBB 上电之前就先把它和电脑连好，在电脑端打开串口调试助手（用 linux 或 mac 的话可以打开终端的 screen）然后用 5V 电源供电或者用 usb 连到电脑上供电。

mac 打开 screen 的命令:screen /dev/cu.usbserial 115200linux 打开 screen 的命令:sudo screen /dev/ttyUSB0 115200 供电以后，串口马上就会开始输出启动信息。过了一会就会出现一个 Angstrom 的字符画 logo，并要求登录，输入用户名和密码（用户名是 root，如果你没改过的话）回车就登录了。

```
.---0---.
| | .-. o o
| | |-----.-----.-----.| | .----..-----.-----.
| | | __ | ---'| '--.| .-'| | |
| | | | | |--- || --'| | | ' | | | |
'---'---'--'--'--. |-----''----''--' '-----'-'-'-'
-' |
'---'

The Angstrom Distribution BeagleBone ttyO0
Angstrom v2012.12 - Kernel 3.8.13
BeagleBone login:
```

启动完成以后，在这里可以像之前用 ssh 连接一样直接输入 shell 命令。可见串口是比 usb+ssh 更可靠的连接方式，但是因为串口的传输速度比 usb 线还是要慢很多的（你可以输入 dmesg 感受一下），所以如果能用 ssh 的话，一般我们还是用 ssh。

这里有一篇文章可以参考《用串口连接 BBB 的 2.5 种方法》

http://dave.cheney.net/2013/09/22/two-point-five-ways-to-access-the-serial-console-on-your-beaglebone-black

用 usb 联网

BeagleBone 如果用 usb 连接到了可以上网的电脑，可以利用分享电脑网络的办法让 BBB 也联网。

Windows7 下的配置方法可以参考 http://lanceme.blogspot.com/2013/06/windows-7-internet-sharing-for.html

电脑系统是 Ubuntu 的话：

BBB 上的操作：

```
route add default gw 192.168.7.1
vim /etc/resolv.conf
```

在最后添加一行 nameserver 8.8.8.8

在 Ubuntu 上：

通过 ifconfig 命令得知 eth0 是本机连接网络的接口，eth1 是 usb 连接 BBB 的接口

```
sudo su
iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
iptables --append FORWARD --in-interface eth1 -j ACCEPT
echo 1 > /proc/sys/net/ipv4/ip_forward
```

然后在BBB上ping www.baidu.com检测一下。

但重启电脑或 BBB 就无效了，须重新配置。

若是 mac OS X 系统：插上 usb 以后，网络连接里会出现一个到 BBB 的连接，记住这个连接的名字。

然后打开系统设置里的“共享”，先取消勾选左边的“互联网共享”，然后在右边勾选刚才的连接名字，再重新勾选左边的互联网共享，电脑端就好了。然后保持 usb 线连接，再插上串口线，用串口连接到 BBB（此时 ssh 是连接不通的），输入 udhcpc -i usb0，就完成配置了。然后就可以拔掉串口线了，如果本次开机过程中断开了 usb 线（比如同时使用 5v 电源在供电）也没关系，重新插上还可以恢复网络连接。

图 1-57

联网以后第一件事就是 opkg update。

本地编译内核模块

因为买来的 BBB 里没有内核源文件，所以没法编译。现在联网了，可以不必重新交叉编译内核，直接在 BBB 上下载配置一下内核源文件。

```
opkg update
opkg install kernel-headers
opkg install kernel-dev
cd /usr/src/kernel
make scripts
```

就配置完成了。这要耗费约 200M 空间，可以先输入 df -h 看看 eMMC 剩余空间够不够。

测试一下。

源文件 mykernelmod.c

```
#include
#include
static int __init enable_usermode(void)
{
printk(KERN_INFO "Usermode enabled.\n");
return 0;
}
static void __exit disable_usermode(void)
{
printk(KERN_INFO "Usermode disabled.\n");
}
module_init(enable_usermode);
module_exit(disable_usermode);
```

Makefile（M 必须大写，注意 make 语句前面必须是 tab 键，直接从网页上复制过去可能会变成空格）

```
obj-m += mykernelmod.o
KDIR = /usr/src/kernel
PWD := $(shell pwd)
all:
make -C $(KDIR) M=$(PWD) ARCH=arm modules
clean:
make -C $(KDIR) M=$(PWD) ARCH=arm clean
```

然后执行 make，成功生成 mykernelmod.ko

insmod mykernelmod.ko 之后 dmesg | tail 可以看到输出了“Usermode enabled.”然后 rmmod mykernelmod，再 dmesg | tail，看到输出了”Usermode disabled.”

点击查看详情：>> 用串口登录 Beaglebone Black、用 usb 共享电脑网络、内核模块的本地编译

1.2.6 用 SSH 远程登录 BeagleBone

先看看 wiki 的 SSH 介绍

Secure Shell（缩写为 SSH），由 IETF 的网络工作小组（Network Working Group）所制定；SSH 为一项创建在应用层和传输层基础上的安全协议，为计算机上的 Shell（壳层）提供安全的传输和使用环境。

传统的网络服务程序，如 rsh、FTP、POP 和 Telnet 其本质上都是不安全的；因为它们在网络上用明文传送数据、用户帐号和用户口令，很容易受到中间人（man-in-the-middle）攻击方式的攻击。就是存在另一个人或者一台机器冒充真正的服务器接收用户传给服务器的数据，然后再冒充用户把数据传给真正的服务器。

而 SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。通过 SSH 可以对所有传输的数据进行加密，也能够防止 DNS 欺骗和 IP 欺骗。

SSH 之另一项优点为其传输的数据可以是经过压缩的，所以可以加快传输的速度。SSH 有很多功能，它既可以代替 Telnet，又可以为 FTP、POP、甚至为 PPP 提供一个安全的“通道”。

大概就是一种相对安全的通讯协议，而且开源。我们不管了，好用就好。

1、怎么用呢，其实我们上次安装的 12.04 版本 ubuntu 已经带了，我们可以执行这个命令 check 一下

sudo apt-get install ssh

会提示你已经安装了哦，没有的话也没关系，联网情况下就会安装。

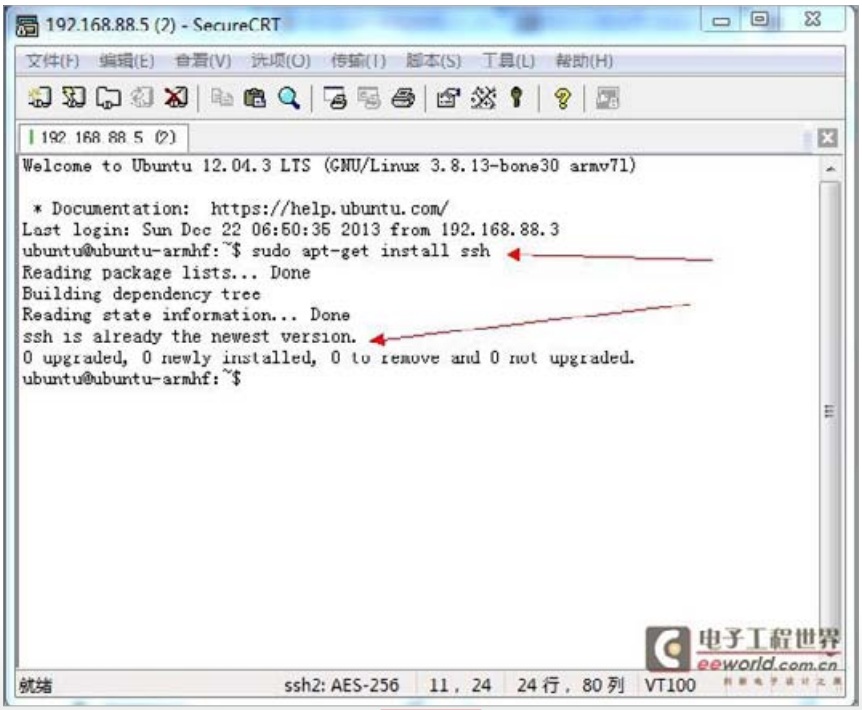


图 1-58

2、然后确定板子在网络中的 IP，我是局域网，路由器查看的 TM 截图

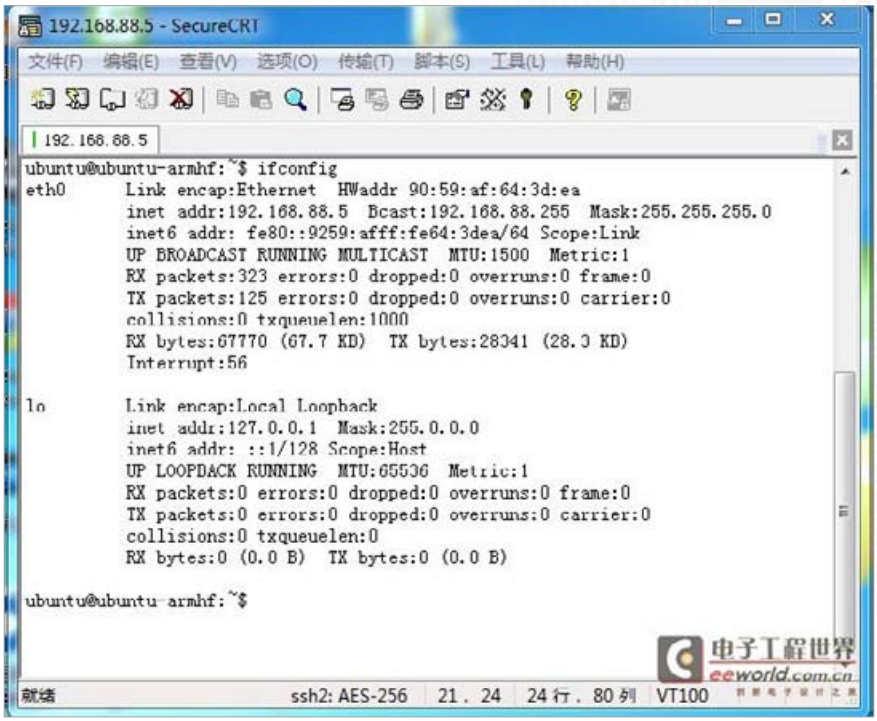


图 1-59

3、还用 SecureCRT 工具，选 SSH2, 写 IP, 写登陆用户名：ubuntu TM 截图

图 1-60

下一步，填密码，默认 ubuntu

图 1-61

好了成功了

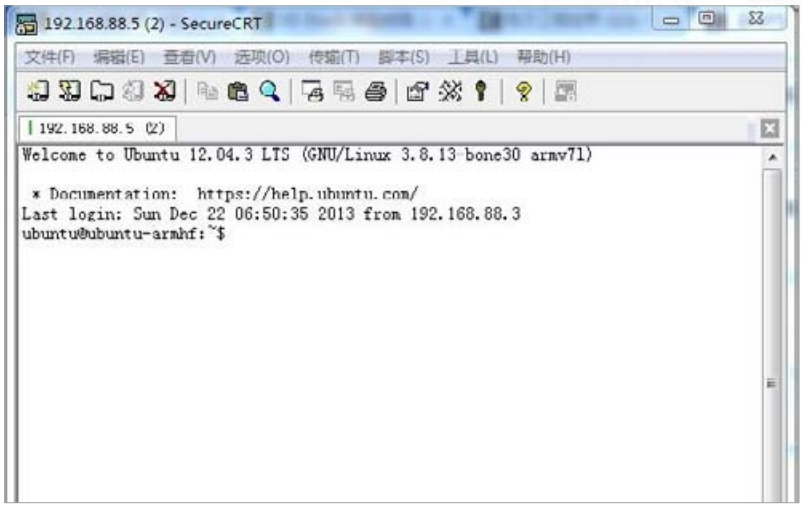


图 1-62

4、我们做个测试，把上一篇说的 u 盘枚举命令执行，操作，看到服务器和文件了。

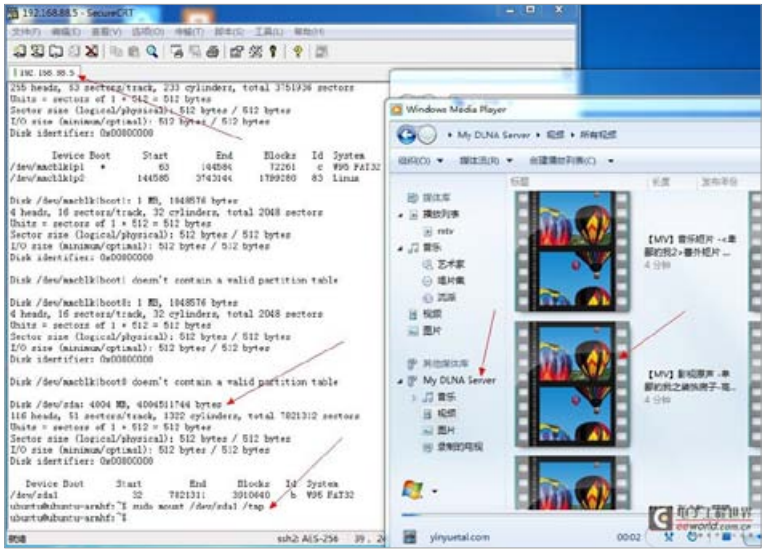


图 1-63

图 1-64

如果你的路由器支持 IP 映射并且有固定外网 IP 或用花生壳之类的，那就可以在远程操作狗板了，是不是很酷。

点击查看详情：>> 用 SSH 远程登录 BeagleBone

1.2.7 基于 BeagleBone 板的 Linux 驱动的调试

今天来分享一下针对 BeagleBone 扩展板，进行 Linux 驱动的调试吧！第二次修正 BeagleBone 板子的外围电路还是比较顺利的！

具体编译内核的过程就不在介绍了！论坛里有很多人写了这方面的教程，Ti 官方的文档也提供了较为全面的编译过程，只不过是全英文的！我主要针对 LCD，触摸屏，Codec 音频等基本的设备的驱动进行了添加，是我的扩展板能够正常工作！

首先是 LCD 屏：相信玩过 BeagleBone 的朋友都知道，AM335x 这款芯片对于 LCD 控制部分是有过勘误的。AM335x 这款芯片勘误手册 就是 16 位和 24 位的更换需要交换 B 和 R，单独使用是没有问题的，但是 16 位 NANDFLASH 的扩展会和 LCD24 位显示有冲突，本人把 16 位 NANDFLASH 的接口做出来了，焊接的时候就可以根据实际情况选择是 16 位还是 8 位，因此我设计了一组可以选择 16 位和 24 位的插针：

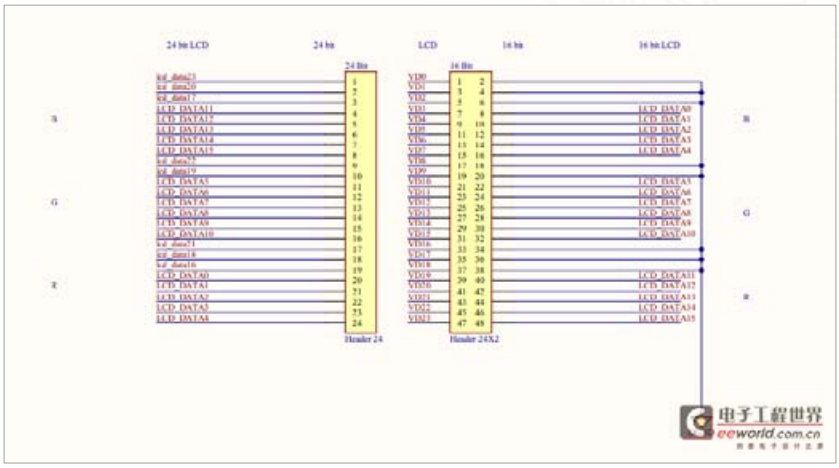


图 1-65

实际焊接的时候，我选择的方案是 8 位的 NANDFLASH，因此跳线选择了 24 位 LCD 显示！

对应着要修改 Linux 内核的板级文件 board-am335xevm.c

在 BeagleBone 的配置函数中加入 LCD 的初始化：

```
/* BeagleBone Rev A3 and after */
static struct evm_dev_cfg BeagleBone_dev_cfg[] = {
        {tps65217_init,      DEV_ON_BASEBOARD, PROFILE_NONE},
        {i2c2_init,          DEV_ON_BASEBOARD, PROFILE_NONE},
        {mii1_init,          DEV_ON_BASEBOARD, PROFILE_NONE},
        {usb0_init,          DEV_ON_BASEBOARD, PROFILE_NONE},
        {usb1_init,          DEV_ON_BASEBOARD, PROFILE_NONE},
        {bbtoys7lcd_init, DEV_ON_BASEBOARD, PROFILE_NONE},//24bit LCD 2013年7月30日
        {bone_tsc_init,        DEV_ON_BASEBOARD, PROFILE_ALL},//触摸 2013年8月1日
        //{boneleds_init, DEV_ON_BASEBOARD, PROFILE_ALL},//2013年7月30日
        {mcasp0_init,         DEV_ON_BASEBOARD, PROFILE_NONE},//Codec 音频
        //{evm_nand_init, DEV_ON_BASEBOARD, PROFILE_NONE},
        {NULL, 0, 0},
};
```

具体的 LCD 初始化程序，可仿照文件里原来 LCD 的定义来编写，只不过定义成自己的名字，值得注意的是还要在内核中 da8xx-fb.c 文件，添加关于显示屏的偏移等因素设置。

修改完毕后，就是配置内核，编译内核，然后将内核放入内存卡中，启动查看效果!

编译内核过程简介：

生成 .config 文件：

make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- am335x_evm_defconfig

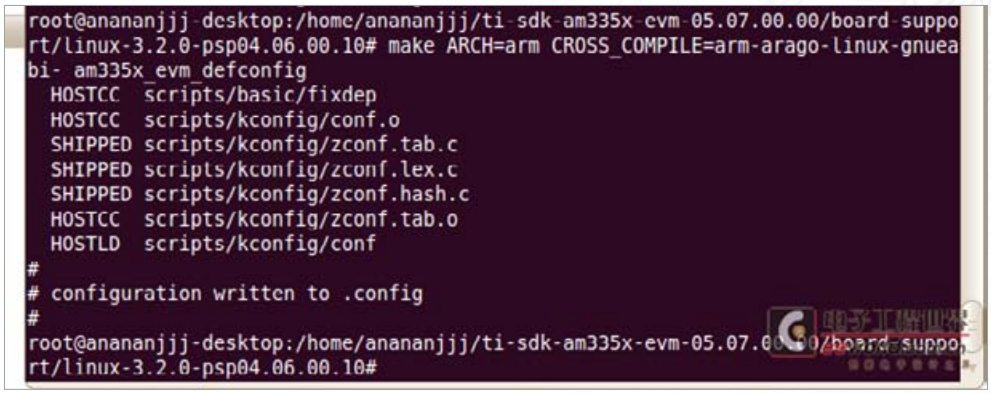


图 1-66

配置内核：

make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- menuconfig

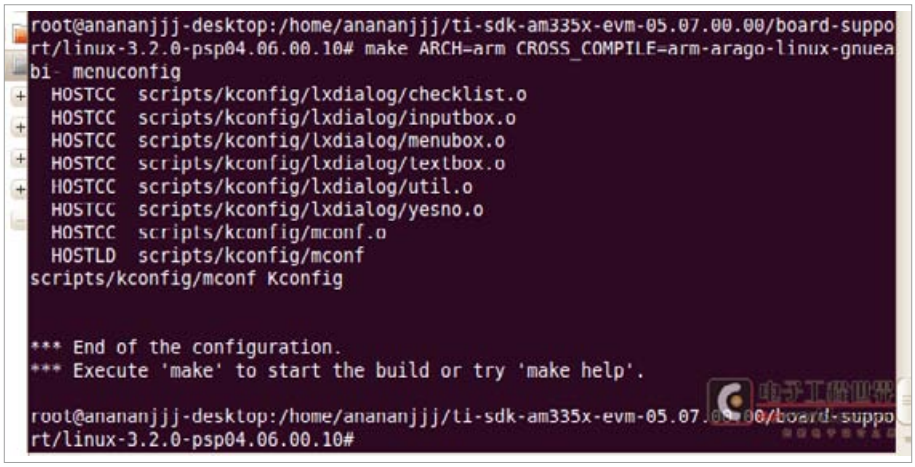


图 1-67

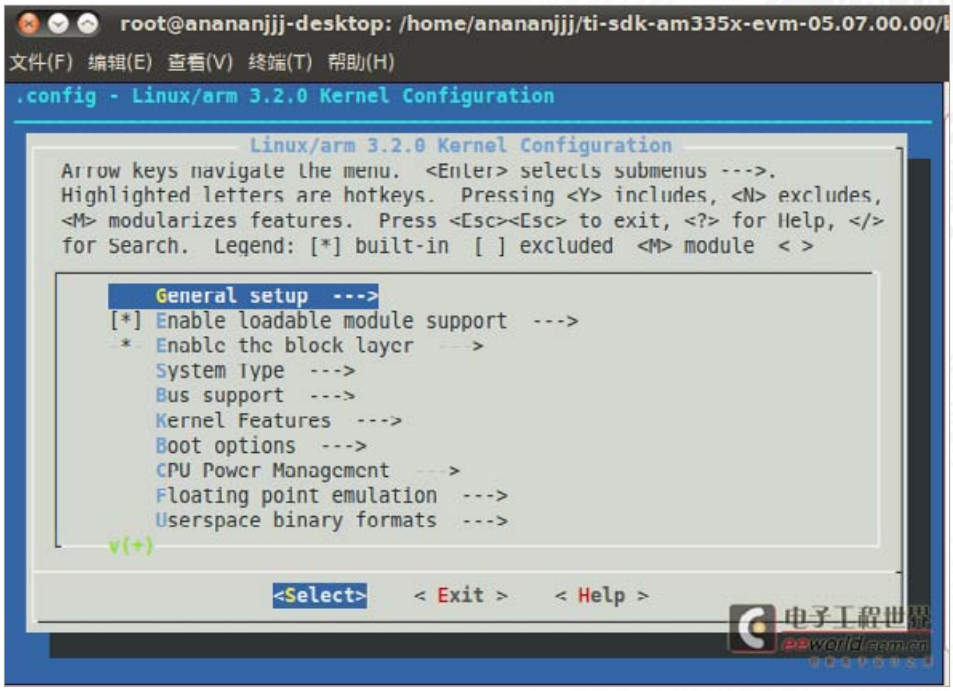


图 1-68

开始编译内核，生成 uImage：

make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- uImage

（时间稍长）

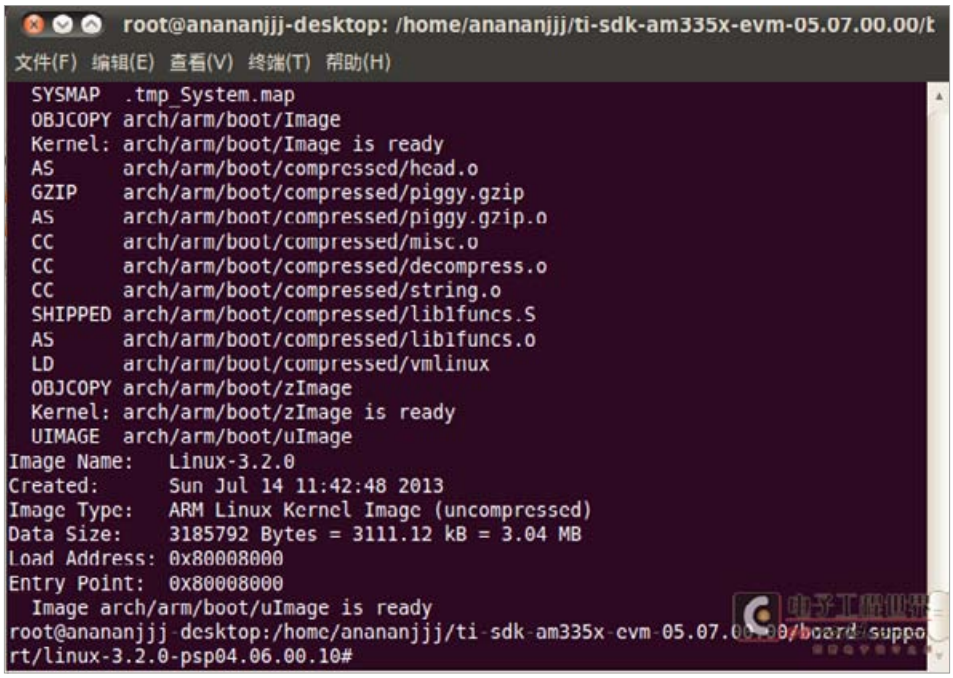


图 1-69

编译好 uImage 之后，到了生成模块 ko：

make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- modules

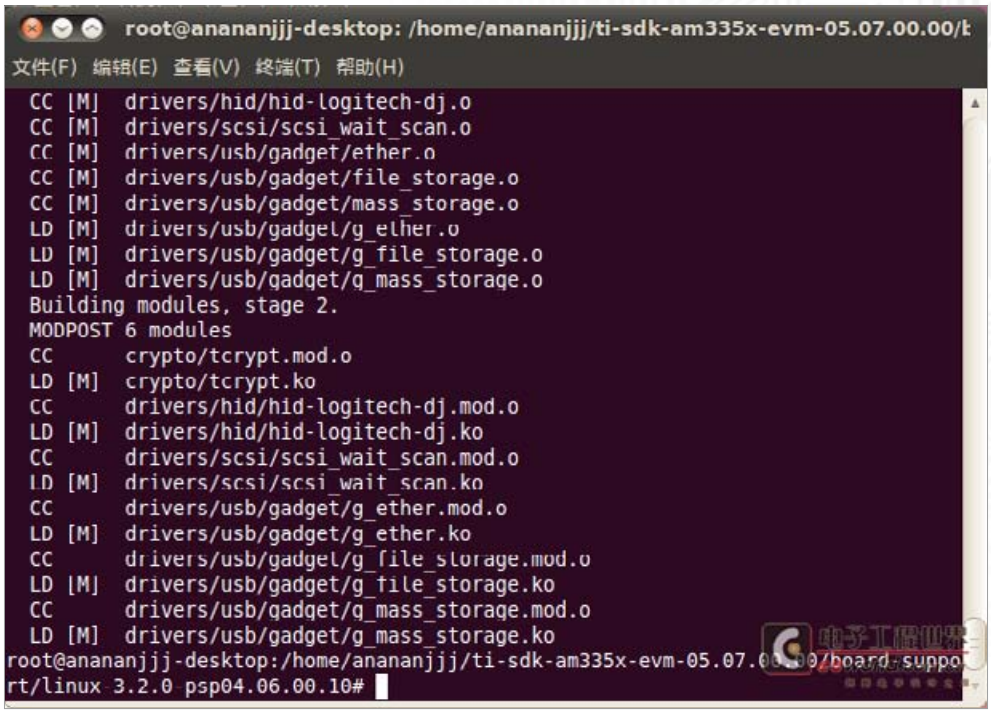


图 1-70

或者缺省编译

make linux

生成的 ko 分散在不同的地方，可以用如下命令复制到文件系统相应的地方：

例如：make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- INSTALL_MOD_PATH=<path to root of file system> modules_install

要解压 TI 为我们制作好的文件系统到 tf 卡的 rootfs 分区：(此处要先格式化 tf 卡! 因此先暂时在主文件夹下操作一次!)

找到文件系统目录：

root@anananjjj-desktop:/home/anananjjj/ti-sdk-am335x-evm-05.07.00.00/filesystem#

执行：

sudo tar -xzvf arago-base-tisdk-image-am335x-evm.tar.gz -C /home/anananjjj/rootfs/

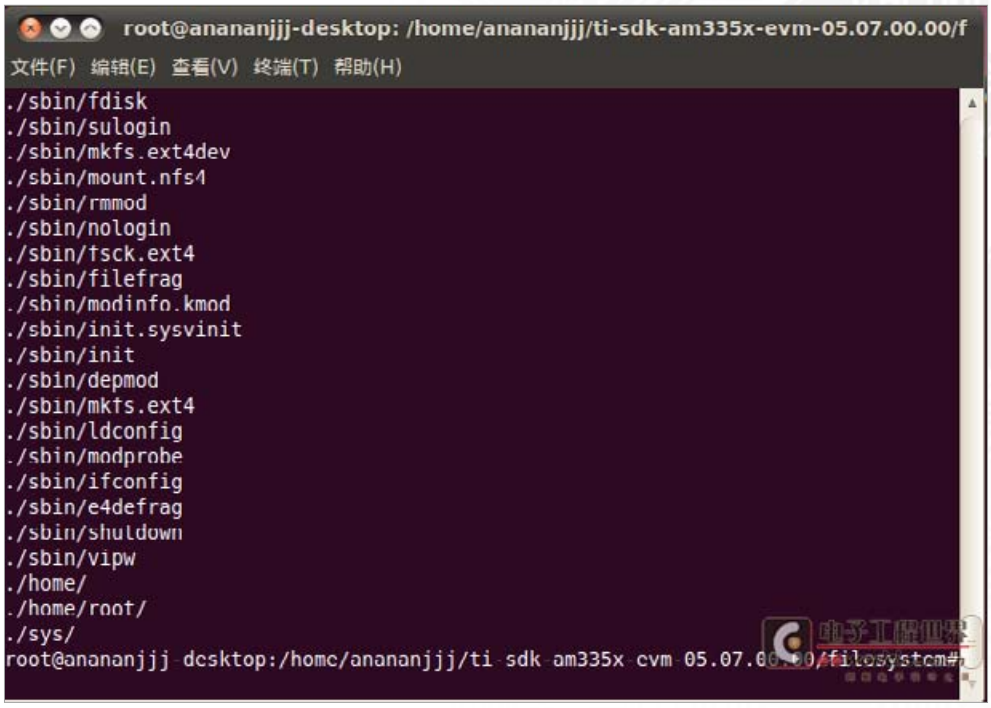


图 1-71

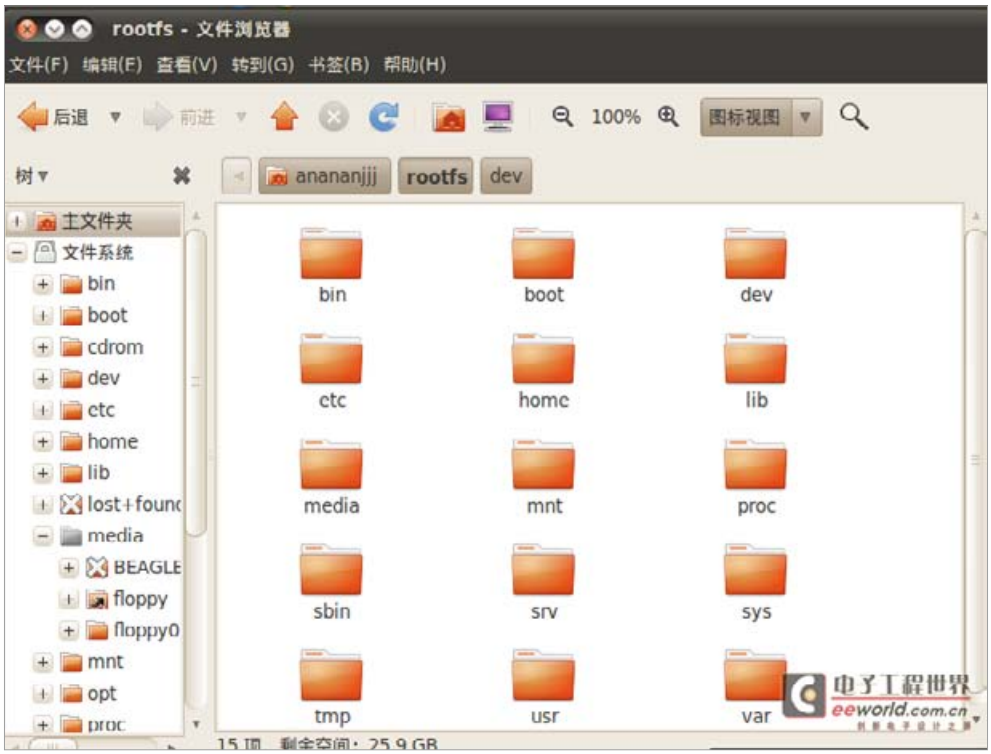


图 1-72

make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- INSTALL_MOD_PATH=/home/anananjjj/rootfsmodules_install

```
root@anananjjj-desktop:/home/anananjjj/ti-sdk-am335x-evm-05.07.00.00/board-suppo
rt/linux-3.2.0-psp04.06.00.10# make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnuea
bi- INSTALL_MOD_PATH=/home/anananjjj/rootfs modules_install
  INSTALL crypto/tcrypt.ko
  INSTALL drivers/hid/hid-logitech-dj.ko
  INSTALL drivers/scsi/scsi_wait_scan.ko
  INSTALL drivers/usb/gadget/g_ether.ko
  INSTALL drivers/usb/gadget/g_file_storage.ko
  INSTALL drivers/usb/gadget/g_mass_storage.ko
  DEPMOD  3.2.0
root@anananjjj-desktop:/home/anananjjj/ti-sdk-am335x-evm-05.07.00.00/board-suppo
rt/linux-3.2.0-psp04.06.00.10#
```

图 1-73

也可以将模块安装在 NFS 目录下：

Make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- INSTALL_MOD_PATH=/home/anananjjj/ti-sdk-am335x-evm-05.07.00.00/targetNFSmodules_install（这样可以为以后 NFS 调试开发板做准备！）

至此，需要的 kernel 已经编译完成，

我在启动阶段设置了加载 Matrix v2 GUI 界面！因此，系统启动之后会进入 Matrix v2 界面！

图 1-74

触摸屏：

之后就是触摸屏的移植，这款芯片支持 4 线，5 线，8 线触摸方式，我使用的是普通的 4 线电阻屏，选择好对应的四路 AD 引脚，设置好触摸参数就可以使用触摸屏了：

```
/* TSc controller */
#include <linux/lis3lv02d.h>

/* TSc controller */
static struct tsc_data am335x_touchscreen_data  = {
        .wires  = 4,
```

```
        .x_plate_resistance = 200,
        .mode = TI_TSCADC_TSCMODE,
};
static struct tsc_data bone_touchscreen_data  = {
        .mode = TI_TSCADC_GENMODE,
};
```

图 1-75

具体效果可以看下面的视频演示。

3、音频 Codec

这个添加的比较彻底，因 Ti 其他 Demo 板都是使用的 McASP1，而扩展的是 McASP0，因此需要重新添加! 好在有 McASP1 作为例子，添加起来倒是不难!

```
static struct snd_platform_data am335x_evm_snd_data0 = {
        .tx_dma_offset        = 0x46000000,        /* McASP0*/
        .rx_dma_offset        = 0x46000000,
        .op_mode        = DAVINCI_MCASP_IIS_MODE,
        .num_serializer        = ARRAY_SIZE(am335x_iis_serializer_direction0),
        .tdm_slots        = 2,
        .serial_dir        = am335x_iis_serializer_direction0,
        .asp_chan_q        = EVENTQ_2,
        .version        = MCASP_VERSION_3,
        .txnumevt        = 1,
        .rxnumevt        = 1,
};

static u8 am335x_iis_serializer_direction0[] = {
        INACTIVE_MODE,        INACTIVE_MODE,        RX_MODE,        TX_MODE,
```

```
        INACTIVE_MODE,          INACTIVE_MODE,          INACTIVE_MODE,
INACTIVE_MODE,
        INACTIVE_MODE,          INACTIVE_MODE,          INACTIVE_MODE,
INACTIVE_MODE,
        INACTIVE_MODE,          INACTIVE_MODE,          INACTIVE_MODE,
INACTIVE_MODE,
};
```

还要使能 McASP0 口的引脚!

```
/* Module pin mux for mcasp0 */
static struct pinmux_config mcasp0_pin_mux[] = {
{"mcasp0_aclkx.mcasp0_aclkx",OMAP_MUX_MODE0 |AM33XX_PIN_INPUT_PULLDOWN},
{"mcasp0_fsx.mcasp0_fsx", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
{"mcasp0_ahclkr.mcasp0_axr2",OMAP_MUX_MODE2| AM33XX_PIN_INPUT_PULLDOWN},
{"mcasp0_ahclkx.mcasp0_axr3", OMAP_MUX_MODE2 | AM33XX_PIN_INPUT_PULLDOWN},
        {NULL, 0},
};
```

接着就是在配置函数中加入初始化函数!

重新编译内核之后，更新内核，上电就可以查看效果了!

点击查看详情：>> 针对 beaglebone 扩展板，进行 Linux 驱动的调试

1.2.8 BeagleBone 的 NFS 启动配置和流程

关于 BeagleBone 开发环境的搭建，前面已经有两位坛友比较详细的记录了，我就不再多费口舌了。

1. BeagleBone 开发环境的搭建

http://bbs.eeworld.com.cn/thread-323132-1-2.html

2. BeagleBone 开发软件安装过程

http://bbs.eeworld.com.cn/thread-326034-1-2.html

还有一个关于 U-Boot 的操作手册，虽然讲的还算详细，但是个人感觉实际用处不是很大，有兴趣的可以看看。

1.AM335x U-Boot User's Guide

http://processors.wiki.ti.com/index.php/AM335x_U-Boot_User's_Guide

2. AM335x PSP User Guide 有关于启动模式的一些配置介绍

http://processors.wiki.ti.com/index.php/AM335x_PSP_User%27s_Guide

坑爹了！！居然在 uboot 下不能运行 saveenv！！

U-Boot# saveenv

Saving Environment to NAND...

Erasing Nand...

Attempt to erase non page aligned data

居然要保存到 Nand，BeagleBone 这个小板没有 Nand 啊，悲剧。。。。

查看了一下 u-boot 的源码，确实把环境变量放在 Nand 中，怎么办？？？？

u-boot-2011.09-psp04.06.00.03\include\configs\am335x_evm.h 的第 19 行

#define CONFIG_NAND_ENV

终于挂上 NFS 了，记录如下。首先感谢 sblpp 和 zhdphao 两位坛友的帮助!

1. 在 BeagleBone 的文件系统中挂载 NFS 文件系统

开发环境搭建的时候在 PC 上要运行 setup.sh，主要就是安装一些文件和配置。

主要需要这些文件：

#sudo apt-get install xinetd tftpd nfs-kernel-server minicom build-essential libncurses5-dev uboot-mkimage autoconf automake

1) 安装完后，需要修改 /etc/exports 文件

#gedit /etc/exports

增加 nfs 文件目录 /mnt/nfs *(rw,sync,no_root_squash)

2) #/etc/init.d/nfs-kernel-server restart // 重启 NFS 服务

到这里只能说 PC 端的 NFS server 搭建好了,在 BeagleBone 上还无法挂载 NFS 系统,会报错的 :

mount: wrong fs type, bad option, bad superblock on 192.168.1.104:/home
missing codepage or helper program, or other error
(for several filesystems (e.g. nfs, cifs) you might
need a /sbin/mount.helper program)
In some cases useful info is found in syslog - try
dmesg | tail or so

怎么办呢? 这个在 sblpp 的大作中已经说的很明白了，还要在 BeagleBone 上安装两个文件 :

nfs-utils-client_1.1.2-2.1_armv7a.ipk 和 portmap_6.0-r3.1_armv7a.ipk

一定要注意是在 BeagleBone 上安装! ! !

1) root@BeagleBone:/# opkg install ./portmap_6.0-r3.1_armv7a.ipk
Installing portmap (6.0-r3.1) to root...
Configuring portmap.
Adding system startup for /etc/init.d/portmap.
2) root@BeagleBone:/# opkg install ./nfs-utils-client_1.1.2-2.1_armv7a.ipk
Installing nfs-utils-client (1.1.2-2.1) to root...
Configuring nfs-utils-client.

如何把这两个文件拷贝到 BeagleBone 里面去的呢? 我是这么做的 :

BeagleBone 启动的时候，已经在 PC 上虚拟出了一个 U 盘，我直接把这两个文件拷贝到这个 U 盘里，然后在 BeagleBone 上建立了一个脚本文件 sd.sh，内容如下 :

mkdir /home/sd
mount -t vfat /dev/mmcblk0p1 /home/sd
这个时候在 /home/sd 下就有所需要的东西了
root@BeagleBone:/home#./sd.sh
root@BeagleBone:/home# cd sd
root@BeagleBone:/home/sd# ls
Docs info.txt Drivers
LICENSE.txt nfs-utils-client_1.1.2-2.1_armv7a.ipk uImage
MLO portmap_6.0-r3.1_armv7a.ipk
README.htm u-boot.img

2. 通过 NFS 文件系统启动 BeagleBone

在没有实现上面功能的时候，其实我已经尝试这从 NFS 根文件系统启动 BeagleBone，并已经成功实现，条条大路通罗马，呵呵

主要步骤：

在 BeagleBone 启动的时候，敲回车，使得启动停在 uboot 部分

U-Boot 2011.09-00000-gf63b270-dirty (Nov 14 2011 - 10:37:14)

I2C: ready

DRAM: 256 MiB

No daughter card present

NAND: HW ECC Hamming Code selected

nand_get_flash_type: unknown NAND device: Manufacturer ID: 0x10, Chip ID: 0x10

No NAND device found!!!

0 MiB

MMC: OMAP SD/MMC: 0

*** Warning - readenv() failed, using default environment

Net: cpsw

Hit any key to stop autoboot: 0

U-Boot#

然后依次输入：

U-Boot# mmc rescan

U-Boot# setenv ipaddr 192.168.1.103

U-Boot# setenv serverip 192.168.1.104

U-Boot# setenv bootargs console=ttyO0,115200n8 root=/dev/nfs nfsroot=192.168.1.104:/mnt/nfs ip=192.168.1.103:192.168.1.104:192.168.1.1:255.255.255.0::eth0:off

U-Boot# run mmc_load_uimage

U-Boot#bootm 0x80007fc0

系统启动：

[4.293236] IP-Config: Complete:

[4.296636] device=eth0, addr=192.168.1.103, mask=255.255.255.0, gw=192.168.1.1,

[4.304648] host=192.168.1.103, domain=, nis-domain=(none),

[4.310965] bootserver=192.168.1.104, rootserver=192.168.1.104, rootpath=

[4.417566] VFS: Mounted root (nfs filesystem) on device 0:15.

[4.424250] Freeing init memory: 240K

grep: /proc/mounts: No such file or directory
grep: /proc/mounts: No such file or directory
Populating /dev using udev: /etc/init.d/S10udev: line 79: can't create /proc/sys/kernel/hotplug: nonexistent directory

到这就挂载成功了!

点击查看详情：>>BeagleBone 的 NFS 启动配置和流程

1.2.9 聊聊 BeagleBone Black 的 cape 和 device tree overlay 和 dtc 命令

我们知道 beagleboard 官网上有一些官方的硬件外设，比如 lcd 显示屏之类的，他们管这些外设叫做 cape。其实这里是我理解狭隘了，应该说只要是修改了芯片引脚功能，或占用了空闲的引脚的东西,都可以叫做 cape。比如之前我们提到的开启某些引脚的 AD 转换功能，其实也是给设备添加了一个 virtual cape。BeagleBone Black 中用一个叫做 capemgr 的软件管理所有的 cape，不论它是实实在在的扩展板，还是虚拟的 cape。这个软件的目录是

/sys/devices/bone_capemgr.*/(这里的 * 是一个每次系统启动可能会不一样的数字(与启动顺序有关))

如果你看过我的博客，也许还会记得我们加载 device tree overlay 时打开了一个文件，正是这个目录下的 slots 文件。slots 文件就是 capemgr 这个软件的对外接口。slot 这个单词是“插槽”的意思，看，很形象吧! 我要插上一个 cape，就向这个“插槽”里“插入”(echo) 相应的设备。echo 这个命令的含义是“向标准设备输出”嘛。另外，.dtbo 文件只有放到 /lib/firmware 目录下才能被使用。

还记得我们第一次打开 slots 文件看到了什么吗?

```
# cat /sys/devices/bone_capemgr.*/slots
# cat /sys/devices/bone_capemgr.*/slots

0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
```

这里前 4 项为什么是空的呢? 它们是给那些有 EEPROM 的实体 cape 预留的位置。通过实体 cape 上的开关，可以达到我们用 echo something > $SLOTS 语句一样的效果。不难看出，这样的实体 cape 最多只能插 4 个。

4 和 5 这两项则是系统已经加载的两个 virtual cape，因为 eMMC 和 HDMI 确实需要占用一些引脚，所以根据前面的定义，它们也是 cape。

其实 device tree overlay 的作用之一是将设备和驱动进行绑定 (前提是驱动或驱动模块已经存在于内核中)，只不过相比从前那种重新编译内核的方式来说，这种方式实在太方便了。

关于 cat $SLOTS

cat 后显示的内容中，左侧的 L 字母代表是否加载。有时 4、5 这两项依然存在，但没有 L 字母，则也没有加载这两个 cape。详见我的日志《为 BBB 制作专属自己的 cape (二)》中的“系统存在 Bug，运行时卸载 cape 会导致 kernel panic”一节。

关于 dtc 命令

参考我的日志《使用 BeagleBone Black 的 PRU》前面写到的“修改系统 dtb 文件”这一部分操作可知：dtc 命令虽然叫做“编译”，但其作用主要只是转变一下数据存储的格式，由人类可阅读的 dts 转成机器识别的 dtb 或 dtbo 或者反过来转换。

实际上由 dtb 转换成 dts 文件后，跟初始 dts 文件还是有不同的，从这个不同也能看出 dts 文件中什么内容是不重要的。以 BBB 自带的 BB-ADC-00A0.dtbo 为例，我们看一下它转换回 dts 文件是什么样的。

```
root@BeagleBone:~/tmp# dtc -I dtb -O dts BB-ADC-00A0.dtbo
/dts-v1/;
/ {
compatible = “ti,BeagleBone”, “ti,BeagleBone-black”;
part-number = “BB-ADC”;
exclusive-use = “P9.39”, “P9.40”, “P9.37”, “P9.38”, “P9.33”, “P9.36”,
 “P9.35”, “tscadc”;
fragment@0 {
target = <0xdeadbeef>;
__overlay__ {
 #address-cells = <0x1>;
 #size-cells = <0x1>;

tscadc {
compatible = “ti,ti-tscadc”;
reg = <0x44e0d000 0x1000>;
interrupt-parent = <0xdeadbeef>;
interrupts = <0x10>;
ti,hwmods = “adc_tsc”;
status = “okay”;
adc {
ti,adc-channels = <0x8>;
};
};
helper {
compatible = “bone-iio-helper”;
vsense-name = “AIN0”, “AIN1”, “AIN2”, “AIN3”, “AIN4”, “AIN5”, “AIN6”,
 “AIN7”;
vsense-scale = <0x1000 0x1000 0x1000 0x1000 0x1000 0x1000 0x1000 0x1000>;
status = “okay”;
linux,phandle = <0x1>;
phandle = <0x1>;
};
};
};
__symbols__ {
test_helper = “/fragment@0/__overlay__/helper”;
};
__fixups__ {
ocp = “/fragment@0:target:0”;
```

```
intc = “/fragment@0/__overlay__/tscadc:interrupt-parent:0”;
};
};
root@BeagleBone:~/tmp# dtc -I dtb -O dts BB-ADC-00A0.dtbo
/dts-v1/;
/ {
        compatible = “ti,BeagleBone”, “ti,BeagleBone-black”;
        part-number = “BB-ADC”;
        exclusive-use = “P9.39”, “P9.40”, “P9.37”, “P9.38”, “P9.33”, “P9.36”,
“P9.35”, “tscadc”;
        fragment@0 {
                target = <0xdeadbeef>;
                __overlay__ {
                        #address-cells = <0x1>;
                        #size-cells = <0x1>;

                        tscadc {
                                compatible = “ti,ti-tscadc”;
                                reg = <0x44e0d000 0x1000>;
                                interrupt-parent = <0xdeadbeef>;
                                interrupts = <0x10>;
                                ti,hwmods = “adc_tsc”;
                                status = “okay”;
                                adc {
                                        ti,adc-channels = <0x8>;
                                };
                        };

                        helper {
                                compatible = “bone-iio-helper”;
                                vsense-name = “AIN0”, “AIN1”, “AIN2”,
“AIN3”, “AIN4”, “AIN5”, “AIN6”, “AIN7”;
                                vsense-scale = <0x1000 0x1000 0x1000 0x1000
0x1000 0x1000 0x1000 0x1000>;
                                status = “okay”;
                                linux,phandle = <0x1>;
                                phandle = <0x1>;
                        };
                };
        };
        __symbols__ {
                test_helper = “/fragment@0/__overlay__/helper”;
        };
        __fixups__ {
                ocp = “/fragment@0:target:0”;
                intc = “/fragment@0/__overlay__/tscadc:interrupt-parent:0”;
        };
};
```

BB-ADC-00A0.dts 原文是这样的

```
/*
* Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
*/
/dts-v1/;
/plugin/;
/ {
compatible = “ti,BeagleBone”, “ti,BeagleBone-black”;
/* identification */
part-number = “BB-ADC”;
/* state the resources this cape uses */
exclusive-use =
/* the pin header uses */
 “P9.39”, /* AIN0 */
 “P9.40”, /* AIN1 */
 “P9.37”, /* AIN2 */
 “P9.38”, /* AIN3 */
 “P9.33”, /* AIN4 */
 “P9.36”, /* AIN5 */
 “P9.35”, /* AIN6 */
/* the hardware IP uses */
 “tscadc”;
fragment@0 {
target = <&ocp>;
__overlay__ {
/* avoid stupid warning */
 #address-cells = <1>;
 #size-cells = <1>;
tscadc {
compatible = “ti,ti-tscadc”;
reg = <0x44e0d000 0x1000>;
interrupt-parent = <&intc>;
interrupts = <16>;
ti,hwmods = “adc_tsc”;
status = “okay”;
adc {
ti,adc-channels = <0 1 2 3 4 5 6 7>;
};
};
test_helper: helper {
compatible = “bone-iio-helper”;
vsense-name = “AIN0”, “AIN1”, “AIN2”, “AIN3”, “AIN4”, “AIN5”, “AIN6”,
 “AIN7”;
vsense-scale = <100 100 100 100 100 100 100 100>;
status = “okay”;
};
};
};
```

```
};
/*
 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;
/ {
    compatible = "ti,BeagleBone", "ti,BeagleBone-black";
    /* identification */
    part-number = "BB-ADC";

    /* state the resources this cape uses */
    exclusive-use =
        /* the pin header uses */
        "P9.39",    /* AIN0 */
        "P9.40",    /* AIN1 */
        "P9.37",    /* AIN2 */
        "P9.38",    /* AIN3 */
        "P9.33",    /* AIN4 */
        "P9.36",    /* AIN5 */
        "P9.35",    /* AIN6 */
        /* the hardware IP uses */
        "tscadc";
    fragment@0 {
        target = <&ocp>;
        __overlay__ {
            /* avoid stupid warning */
            #address-cells = <1>;
            #size-cells = <1>;
            tscadc {
                compatible = "ti,ti-tscadc";
                reg = <0x44e0d000 0x1000>;
                interrupt-parent = <&intc>;
                interrupts = <16>;
                ti,hwmods = "adc_tsc";
                status = "okay";
                adc {
                    ti,adc-channels = <0 1 2 3 4 5 6 7>;
                };
            };
            test_helper: helper {
                compatible = "bone-iio-helper";
                vsense-name  = "AIN0", "AIN1", "AIN2", "AIN3", "AIN4",
"AIN5", "AIN6", "AIN7";
                vsense-scale = <100     100     100     100     100     100
100     100>;
                status = "okay";
```

```
            };
        };
    };
};
```

二者区别请自行比对。

当我想把 dts 文件转换成 dtbo 文件时，出现了错误。

```
root@BeagleBone:~/tmp# dtc -I dts -O dtb BB-ADC-00A0.dts
ERROR (phandle_references): Reference to non-existent node or label “ocp”
ERROR (phandle_references): Reference to non-existent node or label “intc”
ERROR: Input tree has errors, aborting (use -f to force output)
root@BeagleBone:~/tmp# dtc -I dts -O dtb BB-ADC-00A0.dts
ERROR (phandle_references): Reference to non-existent node or label “ocp”
ERROR (phandle_references): Reference to non-existent node or label “intc”
ERROR: Input tree has errors, aborting (use -f to force output)
```

（加上 -f 也是没有用的）但如果加上 -@ ，就可以顺利输出了。这就是 device tree overlay 和原生 device tree 的不同吧——device tree（比如 /boot 目录里那些 dtb 文件反编译出来的 dts 文件）都是包含从根节点到每个子节点的全部信息的。而 device tree overlay（比如 /lib/firmware 目录中的那些）只需要包含要修改的内容即可。至于后缀名 dtbo 和 dtb，似乎单纯是为了区分二者，内容格式其实都是一样的，都可以用 dtc 命令反编译成 dts 文件。

点击查看详情：>> 聊聊 Beaglebone Black 的 cape 和 device tree overlay 和 dtc 命令

1.2.10 从 NAND FLASH 中启动 BeagleBone

本贴总结如何从 NAND FLASH 启动 BeagleBone。

一、BeagleBone Linux 启动过程了解

1、上电，处理器 PoR（Power-on Reset）后就会跳到复位向量所指的程序段去运行，这段程序位于处理器的 ROM 中，由处理器生产厂商提供，我们称其为 ROM BootLoader（RBL）。RBL 在初始化固定存储设备（NANDFlash，MMC，Etc.）后，根据 CPU 的跳线选择，从持久性存储设备（BeagleBone 默认是从 MMC）中将第二部要用到的 bootloader 读入内存（至于是内部还是外部，要根据 ROM 中的程序来定，BeagleBone 是内部 ram），而后将控制权转交。

2、Bootloader，分两步：

第一步的 bootloader 我们称为 Secondary Program Loader（SPL），或 MLO。SPL 完成有限的初始化工作，最重要的就是初始化 DDR RAM（或是其他大一点的外部 RAM），因为下一步的 U-Boot 将要进驻其中。在这之后，SPL 从持久性存储设备中将 U-Boot（其他 bootloader 也行，我们这里以 U-Boot 为例）读入 DDR RAM 中，随后转交 CPU 控制权。

第二步终于到了万众瞩目的 U-Boot 阶段，使能大部分的处理器功能，将操作系统内核读入 DDR RAM 中，而后配置启动参数启动 Linux 内核。当然 U-Boot 的功能可不止这些，U-Boot 中提供了很多应用程序，可以使用 tftp 下载文件，或是格式化存储设备等等。

3、Linux 内核启动……内核初始化，MMU 使能，外部设备初始化，PID(1)Init 进程启动……

至于从什么地方加载内核到 DDRRAM 中，完全由 U-Boot 的启动参数或命令决定的。

为什么要分这么几步呢？源自网络上的一段解释：

一步 Boot 起来不行吗？>_< 其实任何一门技术都是让我们的生活变的简单，Bootloader 也是，所以其不可能是麻烦……首先，第一步的 RBL 不可能太复杂，ROM 大小是一个限制，最重要的还是因为其所能获取的系统信息太少，所以处理器只能使用简单的方法去寻找能完成复杂功能的代码，而后将处理器初始化工作交给它来完成。其次，SPL 也不能很复杂，因为其位于处理器的内部 RAM 中，一般处理器的内部 RAM 都要小于 128KB，其大小注定其不能完成很复杂的功能，是故，再寻找能力更强大一些的代码，将处理器初始化工作交给它来完成。到了在 DDR RAM 中执行的 Bootloader(U-Boot 算一个)，其功能就比较完善了，此时就可以启动操作系统了。

第一部分是对 BeagleBone 启动过程简单的介绍，了解到 BeagleBone 默认是从 MMC SD Card 读取 SPL 的，看下板子上焊接了 R80-R95，R100-R115 电阻哪些焊接了，哪些没焊接，然后再对照 AM335x Technical Reference Manual 看下，就知道啦！如何改成从 NAND FLASH 读取 SPL 呢？改下部分引脚的电平就行了，直接接地或接 +3.3V 就行了，0402 的电阻手工不好焊接。

SD 卡启动配置 SYSBOOT[7:0]00010111/SYSBOOT[15:8]01000000。

NAND 启动配置 SYSBOOT[7:0]00010010/SYSBOOT[15:8]01000010

BeagleBone 板子上没有 NAND FLASH，连 saveenv 都搞不了，怎么板办？

扩展吧…

借鉴下这篇帖子 http://bbs.eeworld.com.cn/thread-355960-1-1.html

二、硬件准备完成，下面就是软件了：

从 SD 卡启动，U-BOOT 界面下通过 TFTP 将 MLO 下载到内存，再从内存写入 NAND 的 SPL 分区：

```
U-Boot# dcache off（tftp一直没反应，加了这个就可以了）
Data (writethrough) Cache is OFF
U-Boot# tftp 0x80007fc0 MLO
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'MLO'.
Load address: 0x80007fc0
Loading: ########
done
Bytes transferred = 37143 (9117 hex)
U-Boot# nand erase 0x00000000 0x9117
NAND erase: device 0 offset 0x0, size 0x9117
Erasing at 0x0 -- 100% complete.
OK
U-Boot# nand write 0x80007fc0 0x00000000 0x9117
NAND write: device 0 offset 0x0, size 0x9117
37143 bytes written: OK
```

类似的将 u-boot.img、uImage、jffs2 文件镜像写入相应的分区：

```
U-Boot# dcache off
Data (writethrough) Cache is OFF
U-Boot# tftp 0x80007fc0 u-boot.img
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'u-boot.img'.
Load address: 0x80007fc0
Loading: ##############################################
done
Bytes transferred = 230316 (383ac hex)
U-Boot# nand erase 0x00080000 0x383ac
NAND erase: offset is not a number
U-Boot# nand erase 0x00080000 0x383ac
NAND erase: device 0 offset 0x80000, size 0x383ac
Erasing at 0xa0000 -- 100% complete.
OK
U-Boot# nand write 0x80007fc0 0x00080000 0x383ac
NAND write: device 0 offset 0x80000, size 0x383ac
230316 bytes written: OK
```

```
U-Boot# dcache off
Data (writethrough) Cache is OFF
U-Boot# tftp 0x82000000 jffs2.img
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'jffs2.img'.
Load address: 0x82000000
Loading: #################################################################
done
Bytes transferred = 14138752 (d7bd80 hex)
U-Boot# nandecc hw 0
HW ECC Hamming Code selected
U-Boot# nand erase 0x00780000 0xd7c000
NAND erase: device 0 offset 0x780000, size 0xd7c000
Erasing at 0x14e0000 -- 100% complete.
OK
U-Boot# nand write 0x82000000 0x00780000 0xd7c000
NAND write: device 0 offset 0x780000, size 0xd7c000
14139392 bytes written: OK
U-Boot# dcache off
Data (writethrough) Cache is OFF
U-Boot# tftp 0x82000000 jffs2.img
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'jffs2.img'.
Load address: 0x82000000
Loading: #################################################################
done
Bytes transferred = 14138752 (d7bd80 hex)
U-Boot# nandecc hw 0
HW ECC Hamming Code selected
U-Boot# nand erase clean 0x00780000 0x0f880000
NAND erase: device 0 offset 0x780000, size 0xf880000
Skipping bad block at  0x081a0000
Skipping bad block at  0x081e0000
Skipping bad block at  0x09980000
Skipping bad block at  0x0a100000
Skipping bad block at  0x0b700000
Skipping bad block at  0x0b940000
Skipping bad block at  0x0dea0000
Skipping bad block at  0x0e620000
Skipping bad block at  0x0f460000
Erasing at 0xffe0000 -- 100% complete. Cleanmarker written at 0xffe0000.
OK
U-Boot# nand write 0x82000000 0x00780000 0xd80000
NAND write: device 0 offset 0x780000, size 0xd80000
14155776 bytes written: OK
```

关于jffs2文件系统制作，下载，启动，请参考如下文档:

http://blog.chinaunix.net/uid-20780360-id-3306649.html

http://processors.wiki.ti.com/index.php/AM335x_NAND_Driver%27s_Guide

http://processors.wiki.ti.com/index.php/AM335x_JFFS2_Support_Guide

http://processors.wiki.ti.com/index.php/MTD_Utilities

点击查看详情：>> 从 NAND FLASH 启动 BeagleBone

1.2.11 启动 BeagleBone，运行“Hello World”程序

安装 Ubuntu10.4 的过程不必多说! 安装完成之后：先共享了 win7 的 E 盘!

具体步骤：

1. 安装 open-vm-dkms，在终端中输入：sudo apt-get install open-vm-dkms

2. 更新完成之后继续输入：sudo vim /etc/fstab

3. 在 fstab 中加入以下语句：

.host:/ /mnt/hgfs vmhgfs defaults,ttl=5 0 0

4. 在终端中输入：sudo reboot

重启后 press s 键 to skip mountfailure

5. 在终端中输入 mount -a

6. 建立 E 的起动器：

图 1-76

结果：

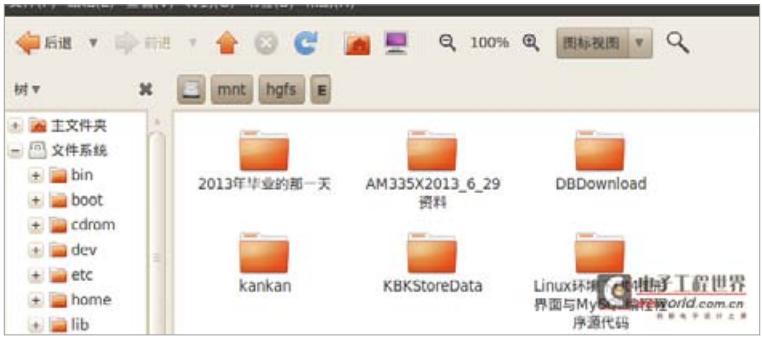


图 1-77

之后复制官方最新的 SDK，ti-sdk-am335x-evm-05.07.00.00-Linux-x86-Install 选择安装解压在我的文件夹中! 点击 setup.h 安装!

根据官方说明文档的介绍，将 SDK 安装完成之后，包括 tfpt、交叉编译工具等一系列工具都配置完成了!

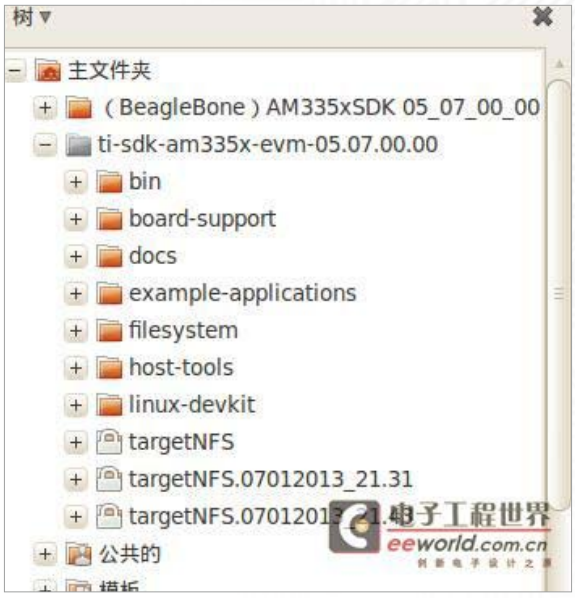


图 1-78

接下来就要设置 交叉编译环境 arm-arago-linux-gnueabi-gcc
我采用的方法是在 修改 /etc/environment 文件
具体是：
运行 $sudo gedit /etc/environment
添加：home/anananjjj/ti-sdk-am335x-evm-05.07.00.00/linux-devkit/bin"
具体如下图：

图 1-79

重启之后，输入：$ echo$PATH
会看到：

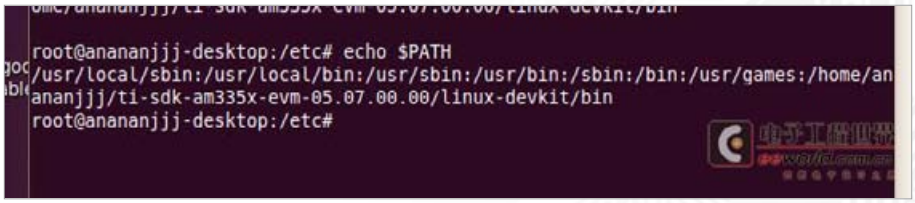


图 1-80

接下来在 Unbutu下安装 minicom，

使用以下命令：

sudo apt-get install minicom

具体使用方法不再详细叙述!

连接上狗板!

会在 Ubuntu10.04 下出现 BEAGLEBONE 图标，点击右键将其弹出 ! 这样就可以使用 USB 转 ETHERNET 的功能了!

在终端输入命令：

#dmesg| grep ttyUSB*

你会发现有 USB 转串口设备有两个!

图 1-81

USB0 是 USB 转 Jtag，USB1 是 USB 转串口!

输入命令：

#minicom -s

设置串口为 ttyUSB1

启动信息终于显示出来了!

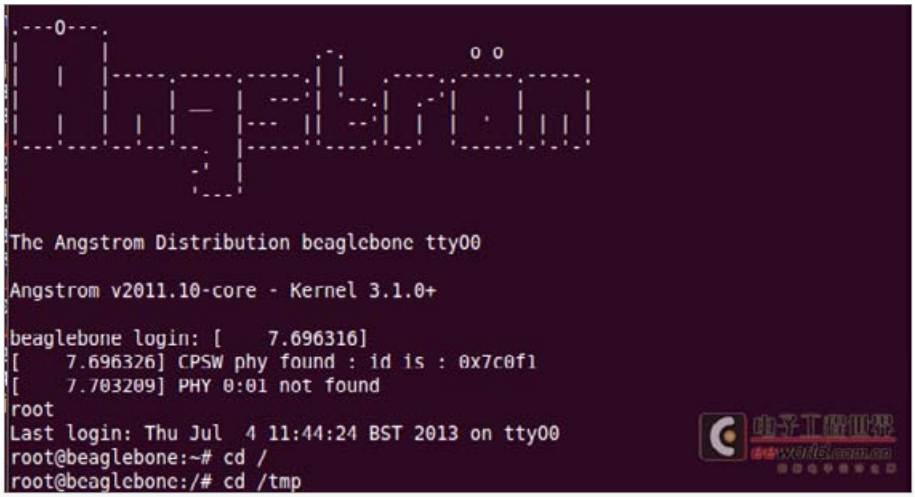


图 1-82

接下来，利用交叉编译工具编译“Hello World！”历程!

输入：arm-arago-linux-gnueabi-gccmain.c -o main

利用 tftp 将程序的 down 如开发版!

```
root@beaglebone:/tmp# tftp 192.168.7.1 -g -r main
root@bcaglcbonc:/tmp# ls
main
root@beaglebone:/tmp# ./main
hello world!
```

图 1-83

我根据 BeagleBoard.org 上的镜像资源制作的 SD，本身自带 USB Host 和 LCD 的驱动! 所以 可以使用键盘和 LCD 功能! 因此我将程序 down 如开发版之后，可以在开发板上直接用键盘敲击命令显示!

图 1-84

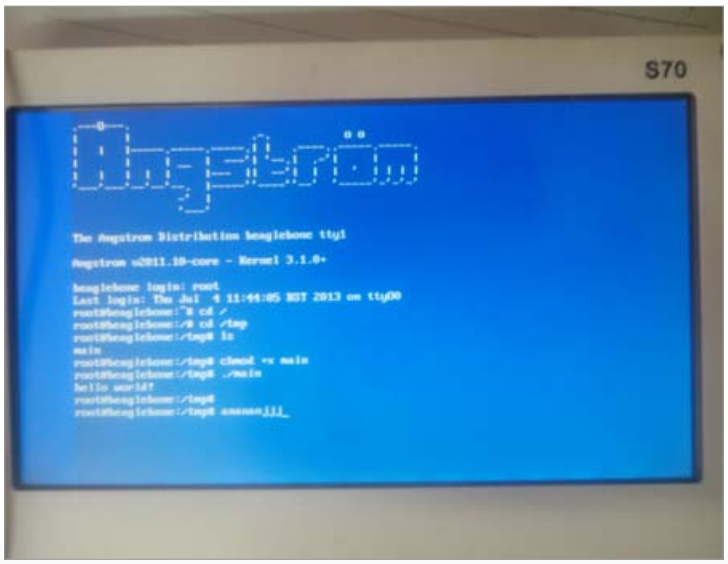


图 1-85

点击查看详情：>>BeagleBone 开发环境搭建

1.2.12 使用 EclipseCDT 搭建 BeagleBone Linux 开发环境

今天看了很久 Derek Molly 的视频 (>> 点击查看详情)(youtube 网址)，最后终于成功搭建好 BeagleBone Linux 的开发环境，在此记录一下。

我的主机环境：Ubuntu 12.04+Eclipse (Kepler Service Release 1)，开发板是 BeagleBone Black，开发板的操作系统是 Angstrom v2012.12 。

先说一句，通过 ssh 远程登陆到 BeagleBone 中，通过安装 g++ (opkg install g++) 也是可以开发的，但是调试时未免麻烦。

下面进入正题：

首先下载 Eclipse + CDT plugin，现在可以直接下载集成好的，下载地址为 http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/keplersr1 ， 打开链接后在页面右侧有 Download Link，可以根据自己的操作系统进行选择，我的是 Linux 64bit。

下载好之后直接解压就可以使用了 (起码 Linux 是这样)，无需安装。

为了能够更方便地通过 PC 来查看、操作 BeagleBone，以及更加方便地调试，需要安装另外一个插件，RSE-runtime。 打开 Eclipse 之后，依次选择 Help -> Install new software, 在弹出来的对话框里的第一个输入框里输入 Kepler - http://download.eclipse.org/releases/kepler，之后 Eclipse 会联网下载可更新软件目录，下载目录完成后，对话框的中间会有很多插件可供安装，点击 Mobile and Device Development 前面的小三角，在展开的目录项中勾选 Remote System Explorer End-User Runtime，然后按照提示即可安装完成。如下图所示：

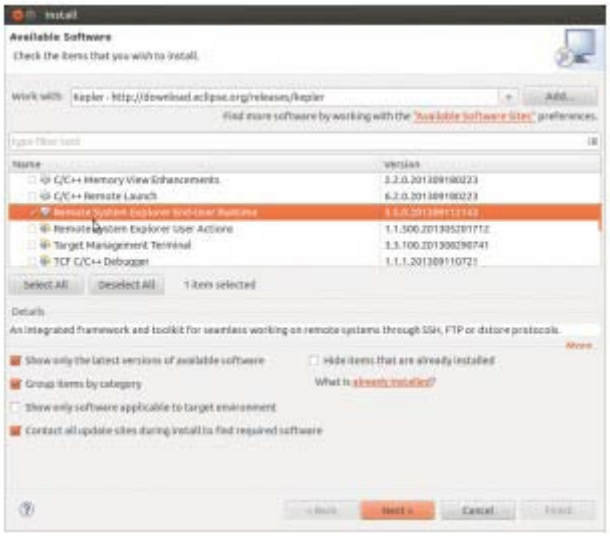


图 1-86

接着来验证一下是否顺利安装成功了。要先确保安装好 Linux 操作系统的 BeagleBone Black 和计算机已经可靠连接。回到 Eclipse 主界面，选择 Window -> Show View -> Other, 弹出的对话框中选择 Remote System，点击 OK。Remote System 的窗口就会显示在界面的下方，可以根据需要把这个窗口拖放的自己喜欢的位置，我把它拖到了界面的左侧，

和 Project Explorer 处于同一位置。

a. 在 Remote System 窗口中右击，然后选择 New -> Connection，弹出如下图所示的对话框

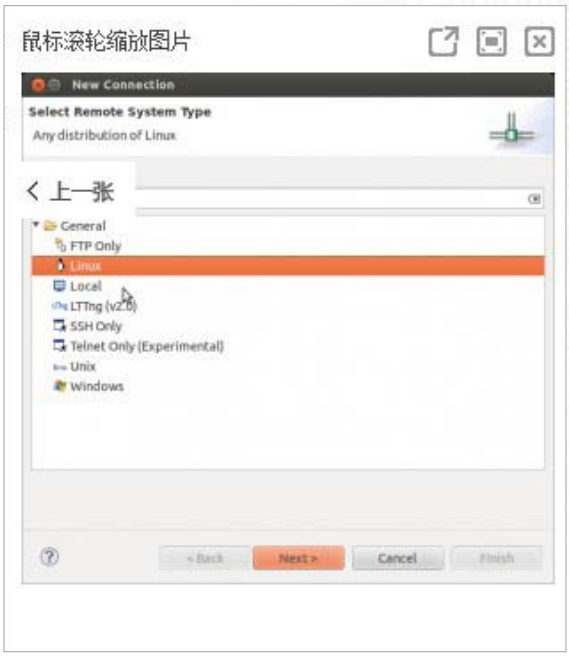


图 1-87

b. 选择 Linux，单击 Next。

c. 然后，按照下图进行设置：

图 1-88

其中 Host Name 填写 BeagleBoneBlack 的 IP 地址，第二条会自动填写。

Description 可以任意填写一个自己喜欢用的名字。

点击 Next。

d. 勾选 Configuration 中的 ssh.files, 点击 Next:

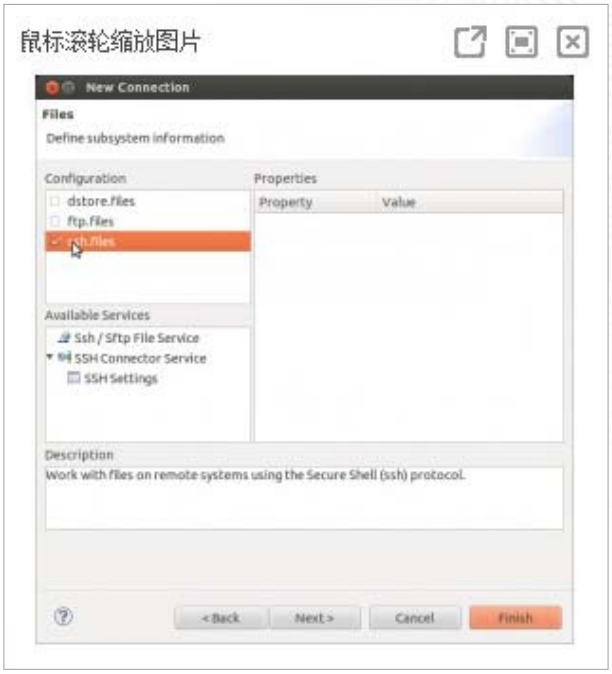


图 1-89

e. 勾选 Configuration 中的 processes.linux.shell, 点击 Next:

图 1-90

f. 勾选 Configuration 中的 ssh.linux, 点击 Finish:

图 1-91

g. 右击 Ssh Terminals, 选择 Connect，在弹出的对话框中填写 BeagleBone Black 上 Linux 中的用户名和密码，就可以登陆了。右击 Ssh Terminal，选择 Launch Terminal，可以在下方的窗口中看到 Terminal，这样，就已经远程登陆到 Linux 中了。试一试，可以交互：

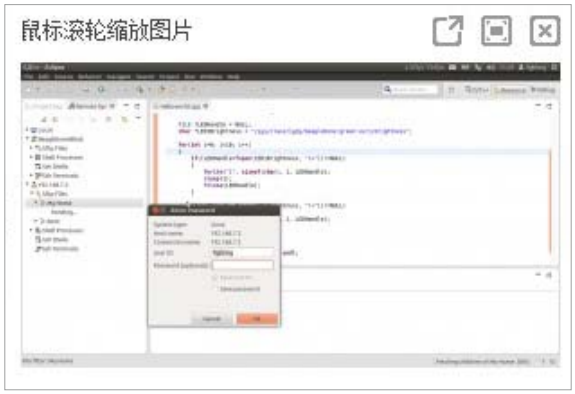


图 1-92

下面编写一个 C++ 程序控制 BeagleBone Black 上 4 个 User LED 的亮灭（这里先以控制一个为例）：

新建一个 C++ Project

源程序为

```
        #include
        #include
using namespace std;
int main() {
        cout << "LED 1 starts flashing" << endl;
        FILE *LEDHandle = NULL;
```

```
        char *LED1Brightness = "/sys/class/leds/BeagleBone:green:usr1/brightness";
        for(int i=0; i<10; i++)
        {
                if((LEDHandle=fopen(LED1Brightness, "r+"))!=NULL)
                {
                   fwrite("1", sizeof(char), 1, LEDHandle);
                        sleep(1);
                        fclose(LEDHandle);
                }
                if((LEDHandle=fopen(LED1Brightness, "r+"))!=NULL)

                {
                        fwrite("0", sizeof(char), 1, LEDHandle);
                        sleep(1);
                        fclose(LEDHandle);
                }
        }
        cout << "LED 1 ends flashing" << endl;
        return 0;
}
```

下面配置交叉编译的工具，首先下载 arm-linux-gnueabi, 直接在 Ubuntu 中输入

sudo install apt-get install arm-linux-gnueabi

然后单击工程，选择 Project->Properties-> C/C++ Build -> Settings, 将 GCC C++ Compiler 和 GCC C++ Linker 中的 Command 改为 arm-linux-gnueabi-g++，将 GCC C Compiler 的 Command 改为 arm-linux-gnueabi-gcc，将 GCC Assembler 的 Command 改为 arm-linux-gnueabi-as。同一窗口中，进入 C/C++ General -> Paths and Symbols, Includes 选项卡中选择 GNU C，添加 /usr/arm-linux-gnueabi/include，添加方式就是点击 ADD，然后选择目录，并单击 Apply to all configurations：

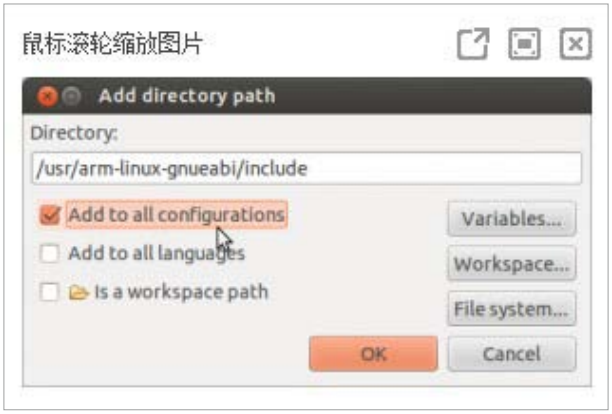


图 1-93

成功添加之后如下：

图 1-94

同样的方法，为 GNU C++ 添加 /usr/arm-linux-gnueabi/include/c++/4.6.3，为 Library Paths 选项卡添加 /usr/arm-linux-gnueabi/lib。这样，就顺利配置好交叉编译器了。可以编译了。

Build All。编译万整个工程之后，在 Project Explorer 中可以看到成功编译出的可执行文件。

复制可执行文件，切换到 RemoteSystem 窗口，粘贴到 BeagleBone Black 根目录下。在下方的 Terminal 中输入：

chmod a+x Helloworld

./Helloworld

可以看见 User Led 已经在闪了，同时 Terminal 中出现了一句话：

图 1-95

20 秒以后，User Led 熄灭，同时 Terminal 中输出另一句：

图 1-96

成功了!

点击查看详情：>> 使用 EclipseCDT 搭建 Beaglebone Linux 开发环境

1.2.13 学习 Sitara AM335x

BeagleBone 环境的搭建方法
参考：http://bbs.eeworld.com.cn/thread-359196-1-1.html

把 BeagleBone 变成一个高清晰度 IP 摄像头
参考：http://bbs.eeworld.com.cn/thread-416406-1-1.html

PhyCORE-AM335X Linux 快速入门
参考：http://bbs.eeworld.com.cn/thread-359221-1-1.html

AM335x 电源选择
参考：http://bbs.eeworld.com.cn/thread-359223-1-1.html

破解 pdf-file 密码
参考：http://bbs.eeworld.com.cn/thread-359224-1-1.html

Passing Time with a SPI Framebuffer Driver
参考：http://bbs.eeworld.com.cn/thread-359225-1-1.html

引导 linux
参考：http://bbs.eeworld.com.cn/thread-359226-1-1.html

linux下的驱动
参考：http://bbs.eeworld.com.cn/thread-359227-1-1.html

am335x 接入 LVDS 显示器总结
参考：http://bbs.eeworld.com.cn/thread-329624-1-1.html

1.2.14 AM335x 的 PRUSSv2 简介与应用

PRUSSv2= Programmable Real-time Unit Sub-System = PRU-ICSS = PRU Industrial Communication Sub-System = 上一代 PRUSS 的 v2 进化版

这是一个 AM335x 等芯片上自带的，独立于 ARM CPU 运行的子系统。其时钟频率为 200MHz，可以直接控制特定的 IO 口，可以达到非常高的实时性要求。一般两种情况需要用到它：一是 linux 系统的实时性不满足要求的时候；二是芯片的功能模块不够用的时候（比如你想要 10 个 UART，但芯片上只有 6 个，那你可以用它再创造 4 个）。

几个可能的应用场合：

* 高速 ADC
* 摄像机接口
* 显示屏接口
* 音频处理
* 电机反馈控制

从某种意义上说，有了它，我们就有了一个片上 CPLD 或 FPGA，只不过 PRU 编程不是用 VDHL 语言，而是用专门的汇编语言。它的汇编语句都是在一个时钟周期内执行完的，没有流水线之类可能扰乱程序时序的因素，这使得程序的可预测性大大提高。

AM335x 芯片上带有两个 PRU。（不知道能不能同时工作？）

配置 BBB，安装 PRU assembler 和示例程序

```
root@BeagleBone:~/tmp# git clone git://github.com/beagleboard/am335x_pru_package.
git
Cloning into  ‘am335x_pru_package’...
...
root@BeagleBone:~/tmp# cd am335x_pru_package/pru_sw/app_loader/interface
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/app_loader/interface# make CROSS_
COMPILE=””  #无需交叉编译
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/app_loader/interface# cd ../../
utils/pasm_source
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/utils/pasm_source# ./linuxbuild
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/utils/pasm_source# cd ../../
example_apps
root@BeagleBone:~/tmp# git clone git://github.com/beagleboard/am335x_pru_package.
git
Cloning into 'am335x_pru_package'...
...
root@BeagleBone:~/tmp# cd am335x_pru_package/pru_sw/app_loader/interface
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/app_loader/interface# make CROSS_
COMPILE="" #无需交叉编译
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/app_loader/interface# cd ../../
utils/pasm_source
```

```
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/utils/pasm_source# ./linuxbuild
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/utils/pasm_source# cd ../../
example_apps
```

编辑一下 example_apps 目录中的 Makefile，把 PASM?=../utils/pasm_2 改成 PASM?=../utils/pasm_2.arm，然后继续

```
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/example_apps# make CROSS_
COMPILE=” ”
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/example_apps# make CROSS_
COMPILE=” ”
```

编译完成以后 PRU assembler 和示例程序就安装好了。但 BBB 默认是没有使能 pruss 的（BB white 好像不用进行下面这一步），所以还需要修改一下系统 dtb 文件

```
cd /boot
cp am335x-boneblack.dtb am335x-boneblack.dtb_orig
dtc -I dtb -O dts am335x-boneblack.dtb > am335x-boneblack.dts
cd /boot
cp am335x-boneblack.dtb am335x-boneblack.dtb_orig
dtc -I dtb -O dts am335x-boneblack.dtb > am335x-boneblack.dts
```

（插一句，我之前没有仔细看过 dtc 命令，原来还可以反编译 dtb 文件啊。。。）

找到

```
pruss@4a30000 {
...
status = “disabled”;
...
};
pruss@4a30000 {
...
status = “disabled”;
...
};
```

把 "disabled" 改成 "okay"，保存。

```
dtc -I dts -O dtb am335x-boneblack.dts > am335x-boneblack.dtb_pru
cp am335x-boneblack.dtb_pru am335x-boneblack.dtb
dtc -I dts -O dtb am335x-boneblack.dts > am335x-boneblack.dtb_pru
cp am335x-boneblack.dtb_pru am335x-boneblack.dtb
```

然后重启。至此，BBB 也已经配置好了。

重启后我们用自带的 example_apps 测试一下

```
root@BeagleBone:~#cd tmp/am335x_pru_package/pru_sw/example_apps/bin
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/example_apps/bin# modprobe uio_
pruss #每次重启都要运行这句话
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/example_apps/bin# ./PRU_
memAccessPRUDataRam
INFO: Starting PRU_memAccessPRUDataRam example.
```

```
AM33XX
INFO: Initializing example.
INFO: Executing example.
File ./PRU_memAccessPRUDataRam.bin open passed
INFO: Waiting for HALT command.
INFO: PRU completed transfer.
INFO: Example executed succesfully.
root@BeagleBone:~#cd tmp/am335x_pru_package/pru_sw/example_apps/bin
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/example_apps/bin# modprobe uio_
pruss #每次重启都要运行这句话
root@BeagleBone:~/tmp/am335x_pru_package/pru_sw/example_apps/bin# ./PRU_
memAccessPRUDataRam
INFO: Starting PRU_memAccessPRUDataRam example.
AM33XX
        INFO: Initializing example.
        INFO: Executing example.
File ./PRU_memAccessPRUDataRam.bin open passed
        INFO: Waiting for HALT command.
        INFO: PRU completed transfer.
INFO: Example executed succesfully.
```

测试成功。

本文最开始的 git 文件建议在电脑上也 clone 一份，里面有几个关于 PRU 的 pdf 文档可能需要经常查看。

现在，已经安装好了am335x_pru_package，本文将用它来编写、编译一个最简单的 PRU 程序——闪烁 BBB 上的 led 灯。

大体来说每个 PRU 程序都包括两部分：
在 ARM 核的 Linux 系统中运行的 C 语言程序

在 PRU 中运行的汇编程序（编译成 .bin 文件）
其中c语言程序有两个作用:将汇编程序传到PRU中;与PRU程序互相传递数据和互动。

汇编程序源文件包括 .p 和 .hp 两种后缀的文件，最后编译成 .bin 文件被传到 PRU 执行。.hp 文件类似于 c 语言的 .h 头文件，可以在 .p 文件中被 include，它不是必须的，本例为了简单起见也不用 .hp 文件。

OK，我们开始做吧!
Step by step 步骤
首先新建两个文件，内容分别如下：

```
//mytest.c
#include <stdio.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>
```

```
#define PRU_NUM 0

int main (void)
{
unsigned int ret;
tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;

prussdrv_init ();//Initialize the PRU
if (prussdrv_open(PRU_EVTOUT_0))//Open PRU Interrupt
{
printf("prussdrv_open open failed\n");
return (-1);
}
prussdrv_pruintc_init(&pruss_intc_initdata);
prussdrv_exec_program (PRU_NUM, "./prucode.bin");//Execute example on PRU
prussdrv_pru_wait_event (PRU_EVTOUT_0);//Waiting for this instruction: MOV r31.b0,
PRU0_ARM_INTERRUPT+16
prussdrv_pru_clear_event (PRU0_ARM_INTERRUPT);
prussdrv_pru_disable (PRU_NUM);//Disable PRU and close memory mapping
prussdrv_exit ();

return(0);
}
//mytest.c
#include <stdio.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>

#define PRU_NUM 0

int main (void)
{
    unsigned int ret;
    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;

    prussdrv_init ();//Initialize the PRU
    if (prussdrv_open(PRU_EVTOUT_0))//Open PRU Interrupt
    {
        printf("prussdrv_open open failed\n");
        return (-1);
    }
    prussdrv_pruintc_init(&pruss_intc_initdata);
    prussdrv_exec_program (PRU_NUM, "./prucode.bin");//Execute example on PRU
    prussdrv_pru_wait_event (PRU_EVTOUT_0);//Waiting for this instruction: MOV r31.
b0, PRU0_ARM_INTERRUPT+16
    prussdrv_pru_clear_event (PRU0_ARM_INTERRUPT);
    prussdrv_pru_disable (PRU_NUM);//Disable PRU and close memory mapping
    prussdrv_exit ();

    return(0);
}
```

```
//prucode.p
.origin 0
.entrypoint START

//Refer to this mapping in the file - \prussdrv\include\pruss_intc_mapping.h
#define PRU0_ARM_INTERRUPT 19
#define CONST_PRUCFG C4

//Refer to AM335X Technical Reference Manual & BBB SRM
#define GPIO1 0x4804c000
#define GPIO_CLEARDATAOUT 0x190
#define GPIO_SETDATAOUT 0x194

START:
// Enable OCP master port
LBCO r0, CONST_PRUCFG, 4, 4
CLR r0, r0, 4 // Clear SYSCFG[STANDBY_INIT] to enable OCP master port
SBCO r0, CONST_PRUCFG, 4, 4

MOV r1, 3 // loop 3 times
LOOP0:
MOV r2, 1<<22
MOV r3, GPIO1 | GPIO_SETDATAOUT
SBBO r2, r3, 0, 4

MOV r0, 100000000
DEL1:
SUB r0, r0, 1
QBNE DEL1, r0, 0

MOV r2, 1<<22
MOV r3, GPIO1 | GPIO_CLEARDATAOUT
SBBO r2, r3, 0, 4

MOV r0, 100000000
DEL2:
SUB r0, r0, 1
QBNE DEL2, r0, 0

SUB r1, r1, 1
QBNE LOOP0, r1, 0

// Send notification to Host for program completion
MOV r31.b0, PRU0_ARM_INTERRUPT+16

// Halt the processor
HALT
//prucode.p
.origin 0
.entrypoint START
```

```
//Refer to this mapping in the file - \prussdrv\include\pruss_intc_mapping.h
#define PRU0_ARM_INTERRUPT      19
#define CONST_PRUCFG            C4

//Refer to AM335X Technical Reference Manual & BBB SRM
#define GPIO1 0x4804c000
#define GPIO_CLEARDATAOUT 0x190
#define GPIO_SETDATAOUT 0x194

START:
    // Enable OCP master port
    LBCO      r0, CONST_PRUCFG, 4, 4
    CLR       r0, r0, 4         // Clear SYSCFG[STANDBY_INIT] to enable OCP
master port
    SBCO      r0, CONST_PRUCFG, 4, 4

    MOV r1, 3 // loop 3 times
LOOP0:
    MOV r2, 1<<22
    MOV r3, GPIO1 | GPIO_SETDATAOUT
    SBBO r2, r3, 0, 4

    MOV r0, 100000000
DEL1:
    SUB r0, r0, 1
    QBNE DEL1, r0, 0

    MOV r2, 1<<22
    MOV r3, GPIO1 | GPIO_CLEARDATAOUT
    SBBO r2, r3, 0, 4

    MOV r0, 100000000
DEL2:
    SUB r0, r0, 1
    QBNE DEL2, r0, 0

    SUB r1, r1, 1
    QBNE LOOP0, r1, 0

    // Send notification to Host for program completion
    MOV       r31.b0, PRU0_ARM_INTERRUPT+16

    // Halt the processor
    HALT
```

然后编译它们。编译之前先从 am335x_pru_package 中复制一些必要文件到标准目录中，省得我们添加引用目录。

把 pru_sw/app_loader/lib 目录中的 libprussdrv.a 拷贝到 /usr/lib 中；

把 pru_sw/app_loader/include 目录中的两个文件都拷贝到 /usr/include 中；

把 pru_sw/utils 目录中的 pasm 拷贝到 /usr/bin 中。
下面开始编译，执行下面两条命令。

```
gcc mytest.c -lpthread -lprussdrv -o mytest
pasm -b prucode.p
gcc mytest.c -lpthread -lprussdrv -o mytest
pasm -b prucode.p
```

（其中 -lpthread -lprussdrv 参数是指要使用 libpthread.so 和 libprussdrv.a 这两个库文件。）

编译完以后就生成了要在 linux 中运行的 mytest 程序和要传到 PRU 中的 prucode.bin 文件。

注意，在运行程序前，还需要做一件事。用 lsmod 命令查看一下 uio_pruss 模块有没有加载。如果没有的话，手动加载一下。这个应该加载一次就可以了。

```
modprobe uio_pruss
modprobe uio_pruss
```

一切就绪！下面输入 ./mytest 来执行一下程序吧！
顺利的话你会看到第二个 led 灯亮灭了三次，每次亮和灭的时间是 1 秒钟。

几点分析

在执行过程中，终端处于忙碌状态，如果在这期间你按下 Crtl+C 终止了程序，会发现 led 还在按照原来的规律继续闪烁，可见灯的亮灭，或者说 PRU 的程序运行并不受 linux 程序的影响。

PRU 简介中提到 AM33xx 中 PRUSSv2 的主频是 200MHz，每条汇编指令执行时间是一个时钟周期。通过这个程序也可以得到验证：

```
MOV r0, 100000000
:
SUB r0, r0, 1
QBNE DEL1, r0, 0
    MOV r0, 100000000
DEL1:
    SUB r0, r0, 1
    QBNE DEL1, r0, 0
```

在这个循环等待的部分，不断对寄存器 r0 执行自减 1 和判断是否为零这两条指令，执行了 100M 次，所以延时为 1 秒。

汇编程序 START: 紧接着的第一部分

```
// Enable OCP master port
LBCO r0, CONST_PRUCFG, 4, 4
CLR r0, r0, 4 // Clear SYSCFG[STANDBY_INIT] to enable OCP master port
SBCO r0, CONST_PRUCFG, 4, 4
    // Enable OCP master port
    LBCO      r0, CONST_PRUCFG, 4, 4
    CLR       r0, r0, 4         // Clear SYSCFG[STANDBY_INIT] to enable OCP
master port
    SBCO      r0, CONST_PRUCFG, 4, 4
```

它把 SYSCFG 寄存器的 STANDBY_INIT 位置零，否则的话将无法访问任何外部 memory，包括 ARM 的内存和 IO 口等。所以记得在每个程序一开始都加上这三条指令。

Table 207. SYSCFG Register Field Descriptions

Bit	Field	Type	Reset	Description
31-6	Reserved	R	0h	
5	SUB_MWAIT	R	0h	Status bit for wait state. 0 = Ready for Transaction 1 = Wait until 0
4	STANDBY_INIT	R/W	1h	1 = Initiate standby sequence. 0 = Enable OCP master ports.

表 1-11

Second choices

最后，如果你不想用官方工具，这里有 second choice：http://www.element14.com/community/community/knode/single-board_computers/next-gen_BeagleBone/blog/2013/06/16/bbb--getting-ace-working

如果你想用 java 脚本与 PRU 里的程序互动，可以参考这里：https://npmjs.org/package/pru

上面这两个东西我都没仔细看过，仅列出来供参考。

点击查看详情：>>AM335x 的 PRUSSv2 简介与使用

1.3 EEworld 论坛网友玩 BeagleBone Black 之基础实验篇

1.3.1 BeagleBone Black GPIO 的相关操作

今天的任务不是很多，主要也就介绍下如何简单操作 GPIO 口工作，并且点亮 LED 灯。

建议操作前先把 BeagleBone Black 系统参考手册看一遍。不多，一会儿就看完了。

直接上正题。我们要做的 LED 灯原理图如下

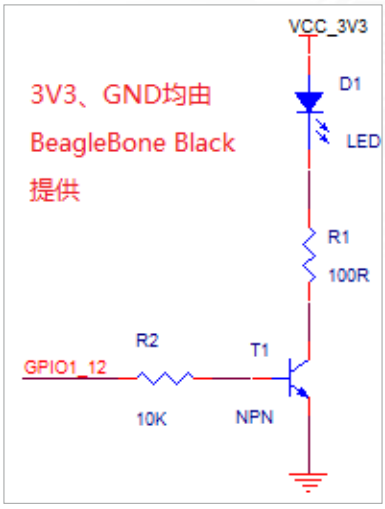


图 1-97 LED 灯原理图

没错，今天就要对 GPIO_12 这个 IO 口进行控制。请做好准备，不管是面包板还是万用板，反正赶紧搭起来就对了。

下面把 P8 P9 的 IO 口贴出来吧。

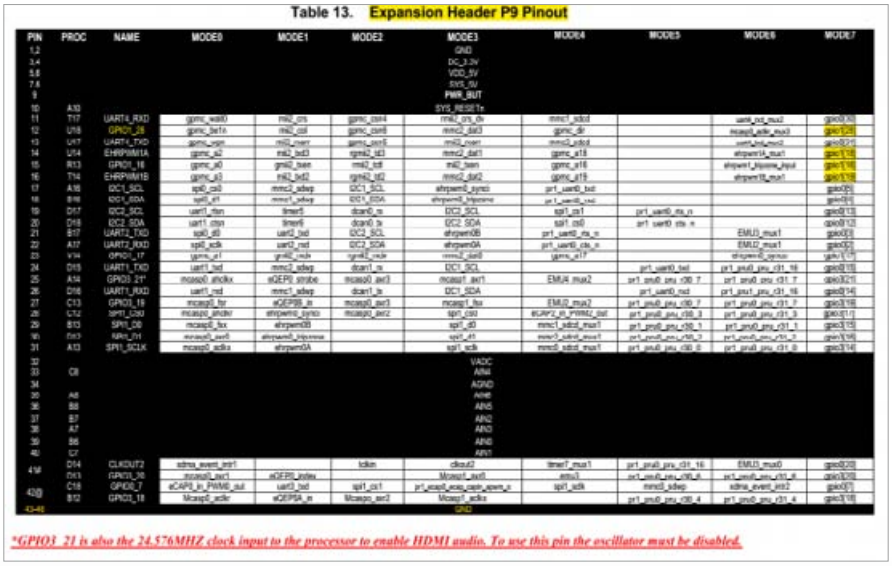


Table 13. Expansion Header P9 Pinout

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1,2						GND				
3,4						DC_3.3V				
5,6						VDD_5V				
7,8						SYS_5V				
9						PWR_BUT				
10	A10					SYS_RESETn				
11	T17	UART4_RXD	gpmc_wait0	mii2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd		uart4_rxd_mux2	gpio0[30]
12	U18	GPIO1_28	gpmc_ben1	mii2_col	gpmc_csn6	mmc2_dat3	gpmc_dir		mcasp0_axr_mux3	gpio1[28]
13	U17	UART4_TXD	gpmc_wpn	mii2_rxer	gpmc_csn5	mii2_rxer	mmc2_sdcd		uart4_txd_mux2	gpio0[31]
14	U14	EHRPWM1A	gpmc_a2	mii2_txd3	rgmii2_td3	mmc2_dat1	gpmc_a18		ehrpwm1A_mux1	gpio1[18]
15	R13	GPIO1_16	gpmc_a0	gmii2_txen	mii2_tctl	mii2_txen	gpmc_a16		ehrpwm1_tripzone_input	gpio1[16]
16	T14	EHRPWM1B	gpmc_a3	mii2_txd2	rgmii2_td2	mmc2_dat2	gpmc_a19		ehrpwm1B_mux1	gpio1[19]
17	A16	I2C1_SCL	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0_synci	pr1_uart0_txd			gpio0[5]
18	B16	I2C1_SDA	spi0_sl1	mmc1_sdwp	I2C1_SDA	ehrpwm0_tripzone	pr1_uart0_rxd			gpio0[4]
19	D17	I2C2_SCL	uart1_rtsn	timer5	dcan0_tx	I2C2_SCL	spi1_cs1	pr1_uart0_rts_n		gpio0[13]
20	D18	I2C2_SDA	uart1_ctsn	timer6	dcan0_rx	I2C2_SDA	spi1_cs0	pr1_uart0_cts_n		gpio0[12]
21	B17	UART2_TXD	spi0_d0	uart2_txd	I2C2_SCL	ehrpwm0B	pr1_uart0_rts_n		EMU3_mux1	gpio0[3]
22	A17	UART2_RXD	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A	pr1_uart0_cts_n		EMU2_mux1	gpio0[2]
23	V14	GPIO1_17	gpmc_a1	gmii2_rxdv	rgmii2_rxdv	mmc2_dat0	gpmc_a17		ehrpwm0_synco	gpio1[17]
24	D13	UART1_TXD	uart1_txd	mmc2_sdwp	dcan1_rx	I2C1_SCL		pr1_uart0_txd	pr1_pru0_pru_r31_16	gpio0[15]
25	A14	GPIO0_27	mcasp0_ahclkx	eQEP0_strobe	mcasp0_axr3	mcasp1_axr1	EMU4_mux2	pr1_pru0_pru_r30_7	pr1_pru0_pru_r31_7	gpio0[27]
26	D16	UART1_RXD	uart1_rxd	mmc1_sdwp	dcan1_tx	I2C1_SDA		pr1_uart0_rxd	pr1_pru1_pru_r31_16	gpio0[14]
27	C13	GPIO3_19	mcasp0_fsr	eQEP0B_in	mcasp0_axr3	mcasp1_fsx	EMU3_mux2	pr1_pru0_pru_r30_7	pr1_pru0_pru_r31_7	gpio3[19]
28	C12	SPI1_CS0	mcasp0_ahclkr	ehrpwm0_synci	mcasp0_axr2	spi1_cs0	eCAP2_in_PWM2_out	pr1_pru0_pru_r30_3	pr1_pru0_pru_r31_3	gpio3[17]
29	B13	SPI1_D0	mcasp0_fsx	ehrpwm0B		spi1_d0	mmc1_sdcd_mux1	pr1_pru0_pru_r30_1	pr1_pru0_pru_r31_1	gpio3[15]
30	D13	SPI1_D1	mmc0_sdwp	ehrpwm0_tripzone		spi1_d1	mmc0_sdcd_mux1	pr1_pru0_pru_r30_5	pr1_pru0_pru_r31_5	gpio3[16]
31	A13	SPI1_SCLK	mcasp0_aclkx	ehrpwm0A		spi1_sclk	mmc0_sdcd_mux1	pr1_pru0_pru_r30_0	pr1_pru0_pru_r31_0	gpio3[14]
32						VADC				
33	C8					AIN4				
34						AGND				
35	A8					AIN6				
36	B8					AIN5				
37	B7					AIN2				
38	A7					AIN3				
39	B6					AIN0				
40	C7					AIN1				
41,42	D14	CLKOUT2	xdma_event_intr1		tclkin	clkout2	timer7_mux1	pr1_pru0_pru_r31_16	EMU3_mux0	gpio0[20]
	D13	GPIO3_20	mcasp0_axr1	eQEP0_index		Mcasp1_axr0	emu3	pr1_pru0_pru_r30_6	pr1_pru0_pru_r31_6	gpio3[20]
	C18	GPIO0_7	eCAP0_in_PWM0_out	uart3_txd	spi1_cs1	pr1_ecap0_ecap_capin_apwm_o	spi1_sclk	mmc0_sdwp	xdma_event_intr2	gpio0[7]
43,44	B12	GPIO3_18	Mcasp0_aclkr	eQEP0A_in	Mcasp0_axr2	Mcasp1_aclkx		pr1_pru0_pru_r30_4	pr1_pru0_pru_r31_4	gpio3[18]
45-46						GND				

*GPIO3_21 is also the 24.576MHZ clock input to the processor to enable HDMI audio. To use this pin the oscillator must be disabled.

图 1-98

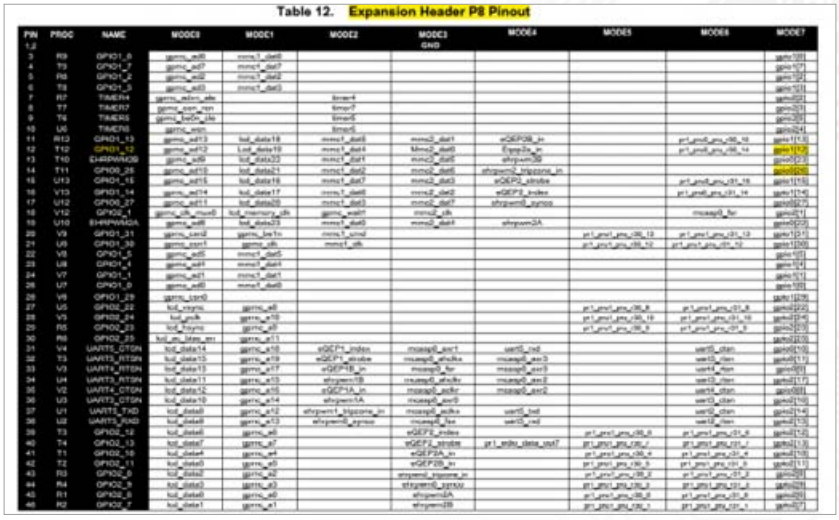


Table 12. Expansion Header P8 Pinout

PIN 1,2	PROC	NAME	MODE0	MODE1	MODE2	MODE3 GND	MODE4	MODE5	MODE6	MODE7
3	R9	GPIO1_6	gpmc_ad6	mmc1_dat6						gpio1[38]
4	T9	GPIO1_7	gpmc_ad7	mmc1_dat7						gpio1[39]
5	R8	GPIO1_2	gpmc_ad2	mmc1_dat2						gpio1[34]
6	T8	GPIO1_3	gpmc_ad3	mmc1_dat3						gpio1[35]
7	R7	TIMER4	gpmc_adv_ale		timer4					gpio2[2]
8	T7	TIMER7	gpmc_oen_ren		timer7					gpio2[3]
9	T6	TIMER5	gpmc_be0n_cle		timer5					gpio2[0]
10	U6	TIMER6	gpmc_wen		timer6					gpio2[1]
11	R13	GPIO1_13	gpmc_ad13	lcd_data18	mmc1_dat6	mmc2_dat1	eQEP2B_in		pr1_pru0_pru_r30_10	gpio1[13]
12	T12	GPIO1_12	gpmc_ad12	lcd_data19	mmc1_dat4	Mmc2_dat0	Eqep2a_in		pr1_pru0_pru_r30_14	gpio1[12]
13	T10	EHRPWM2B	gpmc_a9	lcd_data22	mmc1_dat1	mmc2_dat5	ehrpwm2B			gpio0[23]
14	T11	GPIO0_26	gpmc_a10	lcd_data21	mmc1_dat2	mmc2_dat6	ehrpwm2_tripzone_in			gpio0[26]
15	U13	GPIO1_15	gpmc_a11	lcd_data16	mmc1_dat7	mmc2_dat3	eQEP2_strobe		pr1_pru0_pru_r31_15	gpio1[15]
16	V13	GPIO1_14	gpmc_a14	lcd_data17	mmc1_dat5	mmc2_dat2	eQEP2_index		pr1_pru0_pru_r31_14	gpio1[14]
17	U12	GPIO0_27	gpmc_a11	lcd_data20	mmc1_dat3	mmc2_dat7	ehrpwm0_synco			gpio0[27]
18	V12	GPIO2_1	gpmc_clk_mux0	lcd_memory_clk	gpmc_wait1	mmc2_clk			mcasp0_fsr	gpio2[1]
19	U10	EHRPWM0A	gpmc_a8	lcd_data23	mmc1_dat0	mmc2_dat4	ehrpwm0A			gpio0[22]
20	V9	GPIO1_31	gpmc_csn3	gpmc_be1n	mmc1_cmd			pr1_pru1_pru_r30_13	pr1_pru1_pru_r31_13	gpio1[31]
21	U9	GPIO1_30	gpmc_csn1	gpmc_clk	mmc1_clk			pr1_pru1_pru_r30_12	pr1_pru1_pru_r31_12	gpio1[30]
22	V8	GPIO1_5	gpmc_ad5	mmc1_dat5						gpio1[37]
23	U8	GPIO1_4	gpmc_ad4	mmc1_dat4						gpio1[36]
24	V7	GPIO1_1	gpmc_ad1	mmc1_dat1						gpio1[33]
25	U7	GPIO1_0	gpmc_ad0	mmc1_dat0						gpio1[32]
26	V6	GPIO1_29	gpmc_csn0							gpio1[29]
27	U5	GPIO2_22	lcd_vsync	gpmc_a8				pr1_pru1_pru_r30_8	pr1_pru1_pru_r31_8	gpio2[22]
28	V5	GPIO2_24	lcd_pclk	gpmc_a10				pr1_pru1_pru_r30_10	pr1_pru1_pru_r31_10	gpio2[24]
29	R5	GPIO2_23	lcd_hsync	gpmc_a9				pr1_pru1_pru_r30_9	pr1_pru1_pru_r31_9	gpio2[23]
30	R6	GPIO2_25	lcd_ac_bias_en	gpmc_a11						gpio2[25]
31	V4	UART5_CTSN	lcd_data14	gpmc_a18	eQEP1_index	mcasp0_axr1	uart5_rxd		uart5_ctsn	gpio0[10]
32	T5	UART5_RTSN	lcd_data15	gpmc_a19	eQEP1_strobe	mcasp0_ahclkx	mcasp0_axr3		uart5_rtsn	gpio0[11]
33	V3	UART4_RTSN	lcd_data13	gpmc_a17	eQEP1B_in	mcasp0_fsr	mcasp0_axr3		uart4_rtsn	gpio0[9]
34	U4	UART3_RTSN	lcd_data11	gpmc_a13	ehrpwm1B	mcasp0_ahclkr	mcasp0_axr2		uart3_rtsn	gpio0[17]
35	V2	UART4_CTSN	lcd_data12	gpmc_a16	eQEP1A_in	mcasp0_axr2	mcasp0_axr2		uart4_ctsn	gpio0[8]
36	U3	UART3_CTSN	lcd_data10	gpmc_a14	ehrpwm1A	mcasp0_axr0			uart3_ctsn	gpio0[16]
37	U1	UART5_TXD	lcd_data9	gpmc_a12	ehrpwm1_tripzone_in	mcasp0_aclkx	uart5_txd		uart5_ctsn	gpio0[14]
38	U2	UART5_RXD	lcd_data8	gpmc_a13	ehrpwm0_synci	mcasp0_fsx	uart5_rxd		uart5_rtsn	gpio0[15]
39	T3	GPIO2_12	lcd_data6	gpmc_a6		eQEP2_index		pr1_pru1_pru_r30_6	pr1_pru1_pru_r31_6	gpio2[12]
40	T4	GPIO2_13	lcd_data7	gpmc_a7		eQEP2_strobe	pr1_edio_data_out7	pr1_pru1_pru_r30_7	pr1_pru1_pru_r31_7	gpio2[13]
41	T1	GPIO2_10	lcd_data4	gpmc_a4		eQEP2A_in		pr1_pru1_pru_r30_4	pr1_pru1_pru_r31_4	gpio2[10]
42	T2	GPIO2_11	lcd_data5	gpmc_a5		eQEP2B_in		pr1_pru1_pru_r30_5	pr1_pru1_pru_r31_5	gpio2[11]
43	R3	GPIO2_8	lcd_data2	gpmc_a2		ehrpwm2_tripzone_in		pr1_pru1_pru_r30_2	pr1_pru1_pru_r31_2	gpio2[8]
44	R4	GPIO2_9	lcd_data3	gpmc_a3		ehrpwm0_synco		pr1_pru1_pru_r30_3	pr1_pru1_pru_r31_3	gpio2[9]
45	R1	GPIO2_6	lcd_data0	gpmc_a0		ehrpwm2A		pr1_pru1_pru_r30_0	pr1_pru1_pru_r31_0	gpio2[6]
46	R2	GPIO2_7	lcd_data1	gpmc_a1		ehrpwm2B		pr1_pru1_pru_r30_1	pr1_pru1_pru_r31_1	gpio2[7]

图 1-99

今天我们要对 P8 口的 GPIO1_12 进行操作。3V3 和 GND 也是板子提供的。

注意：

为了在 linux下操作某一引脚，引脚值通常为 通道号 x32+ 该通道上的引脚编号。

这里 GPIO1_12 为例，就是 1x32+12 =44。

关于 GPIO，大家可以参考 Linux 的 gpio 接口手册。也就是说 Linux下基本上也是差不多这么操作 GPIO 口的，就像下面这些代码（摘自 Linux gpio.txt）。

```
/sys/class/gpio/
        "export" ... Userspace may ask the kernel to export control of
                a GPIO to userspace by writing its number to this file.
                Example:  "echo 19 > export" will create a "gpio19" node
                for GPIO #19, if that's not requested by kernel code.
        "unexport" ... Reverses the effect of exporting to userspace.
                Example:  "echo 19 > unexport" will remove a "gpio19"
                node exported using the "export" file.
```

相关操作函数

```
        /* export the GPIO to userspace */
        int gpio_export(unsigned gpio, bool direction_may_change);

        /* reverse gpio_export() */
        void gpio_unexport();

        /* create a sysfs link to an exported GPIO node */
        int gpio_export_link(struct device *dev, const char *name,
                unsigned gpio);

        /* change the polarity of a GPIO node in sysfs */
        int gpio_sysfs_set_active_low(unsigned gpio, int value);
```

我们这里操作系统已经封装好啦，各位就不要纠结再往下的东西了. 以后用到了再深入也不急嘛。

gpio 接口手册链接在这里（论坛放链接需要审核，而我写帖子总会修改几次）。

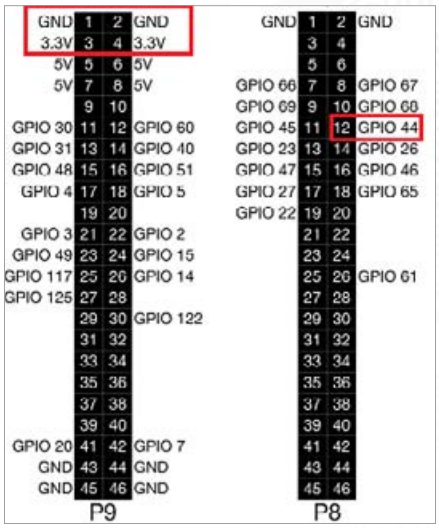


图 1-100

然后看看它们都在哪里。好了。硬件部分就介绍到这里。

下面看我们如何对 BB Black 操作的：

cd /sys/class/gpio/　　进入 GPIO 文件夹

可以看到，该文件夹下有 文件或文件夹 export　gpiochip0　gpiochip32　gpiochip64　gpiochip96　unexport

只有四个 GPIOCHIP，其他的没有啊，那没有我们怎么操作呢?

不急，我们这么操作。echo 44 > export

复制代码这样就新建了一个引脚值为 44 的 GPIOCHIP 文件夹。cd gpio44/

复制代码然后发现有这几个文件 / 文件夹

active_low　direction　edge　power　subsystem　uevent　value

先确定方向 direction，这里参数应该是 out 或 in。或者你要是不守规矩的，high 或者 low 也勉强可以用。

echo out > direction

复制代码这里其他几个文件就先不介绍了，老实说，我们不是很清楚，用到了的时候再说吧。有同学知道的补充下啊。

然后就是 IO 口的开关了。

echo 1 > value　　//IO 口置高
echo 0 > value　　//IO 口置低

操作截图如下：

```
root@ubuntu-armhf: /sys/class/gpio/gpio44
root@ubuntu-armhf:/# cd /sys/class/gpio/
root@ubuntu-armhf:/sys/class/gpio# ls
export  gpiochip0  gpiochip32  gpiochip64  gpiochip96  unexport
root@ubuntu-armhf:/sys/class/gpio# echo 44 > export
root@ubuntu-armhf:/sys/class/gpio# ls
export  gpio44  gpiochip0  gpiochip32  gpiochip64  gpiochip96  unexport
root@ubuntu-armhf:/sys/class/gpio# cd gpio44
root@ubuntu-armhf:/sys/class/gpio/gpio44# ls
active_low  direction  edge  power  subsystem  uevent  value
root@ubuntu-armhf:/sys/class/gpio/gpio44# echo out > direction
root@ubuntu-armhf:/sys/class/gpio/gpio44# cat direction
out
root@ubuntu-armhf:/sys/class/gpio/gpio44# echo 1 > value
root@ubuntu-armhf:/sys/class/gpio/gpio44# echo 0 > value
root@ubuntu-armhf:/sys/class/gpio/gpio44# echo 1 > value
root@ubuntu-armhf:/sys/class/gpio/gpio44# echo 0 > value
root@ubuntu-armhf:/sys/class/gpio/gpio44# cat value
0
root@ubuntu-armhf:/sys/class/gpio/gpio44# echo 1 > value
root@ubuntu-armhf:/sys/class/gpio/gpio44# cat value
1
root@ubuntu-armhf:/sys/class/gpio/gpio44# echo 0 > value
root@ubuntu-armhf:/sys/class/gpio/gpio44#
root@ubuntu-armhf:/sys/class/gpio/gpio44#
```

图 1-101

上实物图：

我这条件差是差了点，但好在这些基本的东西还是有的。

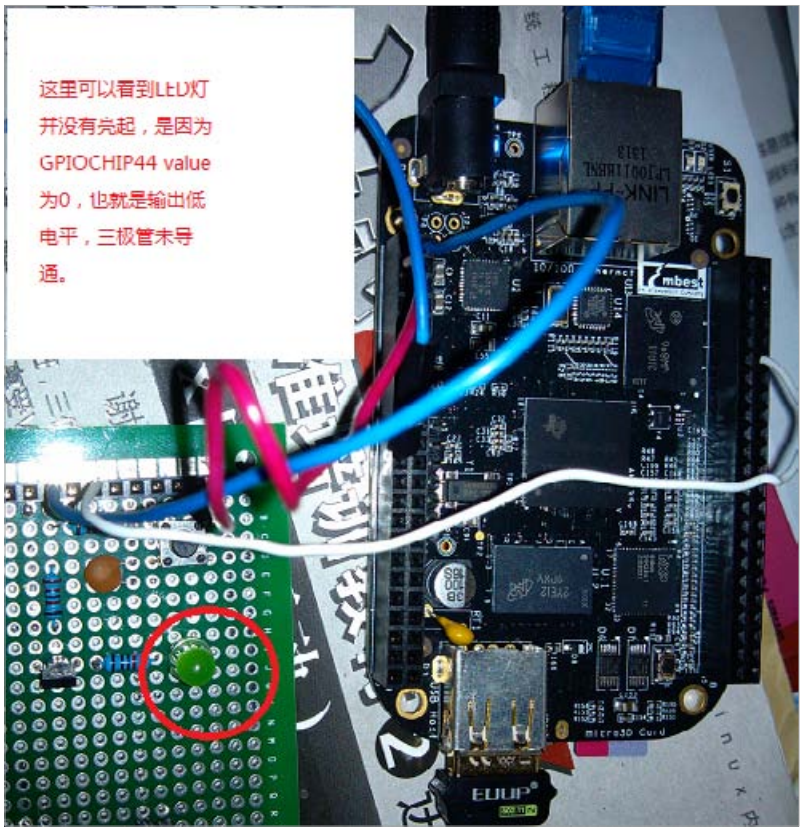


图 1-102

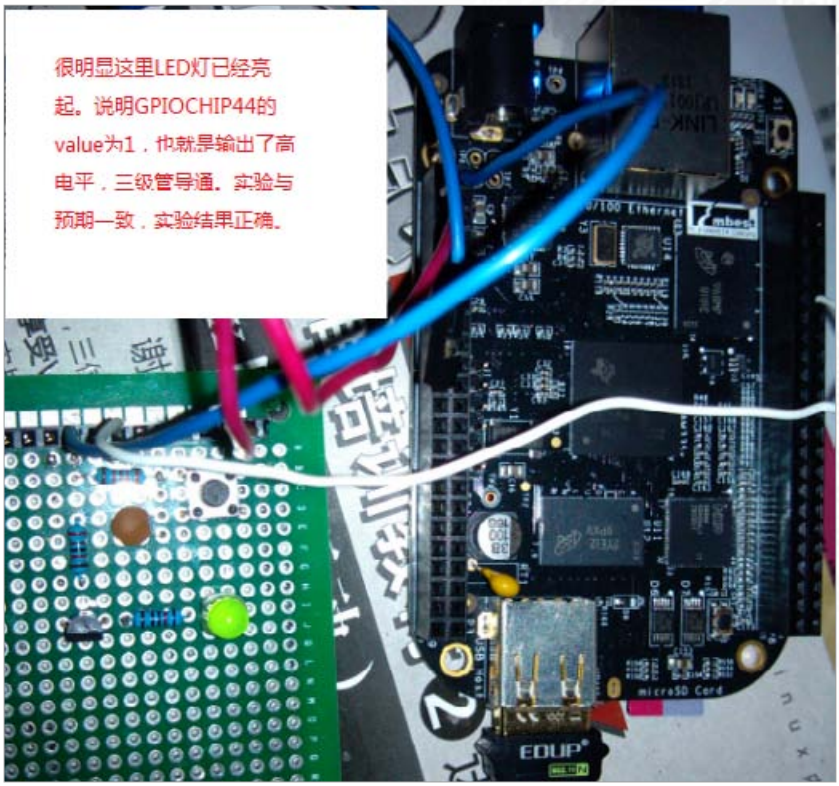


图 1-103

那个代码添加功能太折腾人，我这就不用了。各位辛苦一下，自个粘贴复制。eclipse 里边写一个 IO 口闪烁的函数。

```
#define MAX 64
int gpioLEDFlash(int gpioPin, int times)
{
FILE* LEDHandle = NULL;
char setValue[4], GPIONum[4], GPIOValue[MAX], GPIODirection[MAX];
cout<< "GPIO LED Flash Pin:" << gpioPin << " Start" << endl;

sprintf(GPIONum, "%d", gpioPin); //IO口的值 咱们之前用的是44对吧?
sprintf(GPIOValue, "/sys/class/gpio/gpio%d/value", gpioPin);//要追加的文本
sprintf(GPIODirection, "/sys/class/gpio/gpio%d/direction", gpioPin);//IO口操作方向

//export the gpio pin
if( ( LEDHandle = fopen("/sys/class/gpio/export", "ab") ) == NULL) //使用该IO口
{
  printf("cannot export gpio pin.\n");
  return -1;
}
strcpy(setValue, GPIONum);
fwrite(&setValue, sizeof(char), sizeof(setValue) - 1 , LEDHandle); //这里其实就是导出要使用的IO口，同 echo 44 > export .简单吧,嘿嘿。
fclose(LEDHandle);

//set the direction of the gpio pin
if( ( LEDHandle = fopen(GPIODirection, “rb+”) ) == NULL)
```

```
{
  printf("cannot open direction handle.\n");
  return -2;
}
strcpy(setValue, "out");
fwrite(&setValue, sizeof(char), sizeof(setValue) - 1, LEDHandle); //不用多说了，配置IO方向。
fclose(LEDHandle);

//Flash the LED times
times *= 2; //为什么要*2，一亮一灭是一个循环。
for(int i = 0; i<= times; i ++)
{
  if( ( LEDHandle = fopen(GPIOValue, "rb+") ) == NULL ) //更不必多说，value 0/1
  {
   printf("cannot open value handle.\n");
   return -3;
  }
  if(i%2 == 0)
  {
   strcpy(setValue, "0");
  }
  else
  {
   strcpy(setValue, "1");
  }
  fwrite(&setValue, sizeof(char), sizeof(setValue) - 1, LEDHandle);
  fclose(LEDHandle);
  sleep(1);
}

//Unexport the pin
if( (LEDHandle = fopen("/sys/class/gpio/unexport", "ab")) == NULL) //回收该GPIO
{
  printf("cannot unexport the GPIO pin.\n");
  return -4;
}
strcpy(setValue, GPIONum);
fwrite(&setValue, sizeof(char), sizeof(setValue) - 1, LEDHandle);
fclose(LEDHandle);
cout<< "GPIO LED Flash Pin:" << gpioPin << " End." << endl;
return 0;
}
```

然后主函数调用

```
int main()
{
gpioLEDFlash(44, 10);//循环10次
return 0;
}
```

运行结果

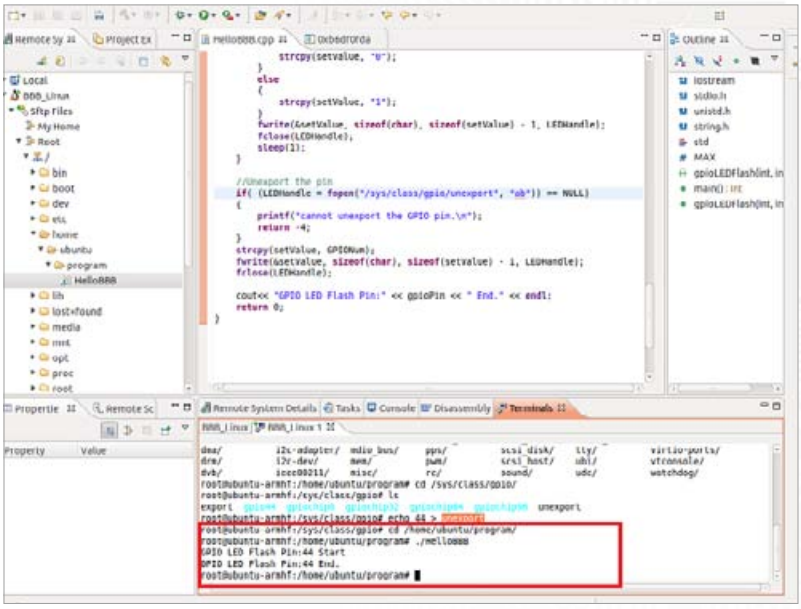


图 1-104

这个视频就不放了吧。

注意，需要使用管理员权限，如果想要调试，请用 root 用户登录。

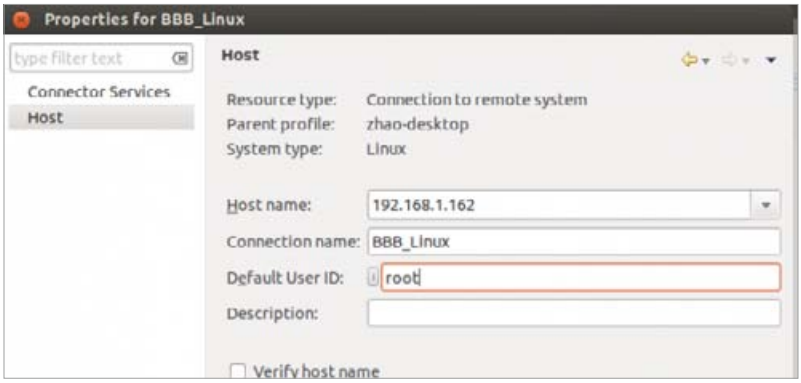


图 1-105

点击查看详情：>> BB Black 入门基础之初识 GPIO

有了上面的基础，下面就容易了。咱们使用 poll() 来轮询设备，配置 GPIO 中断，接着控制 LED 灯。

前期准备

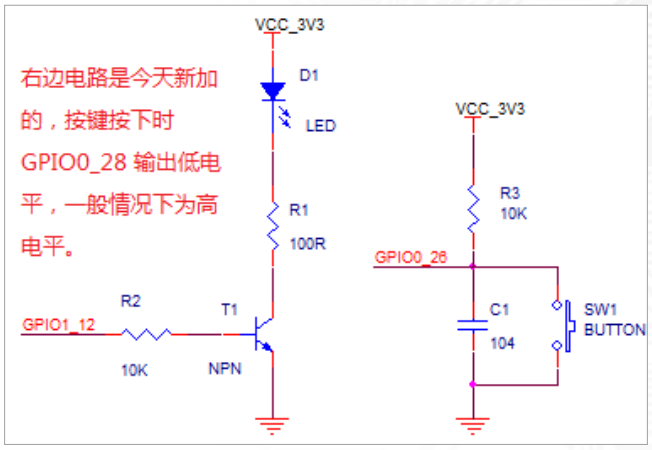


图 1-106

不管是面包板还是万用板还是印制电路板还是别的，只要有类似功能的都可以。

今天要做的就是读入按键部分的高低电平信号，并控制 LED 灯亮灭。

GPIO 操作步骤大致总结如下：

1.在文件系统中配置内核GPIO支持。这点我们不需要做了。直接在 /sys/class/gpio 可用。

如果你非知道怎么在 sysfs 中配置，我们再进一步学习交流。elinux.org/GPIO 上写着是这么配置的。

```
Symbol: GPIO_SYSFS [=y]
Prompt: /sys/class/gpio/... (sysfs interface)
Defined at drivers/gpio/Kconfig:51
Depends on: GPIOLIB && SYSFS && EXPERIMENTAL
Location:
-> Kernel configuration
-> Device Drivers
-> GPIO Support (GPIOLIB [=y])
```

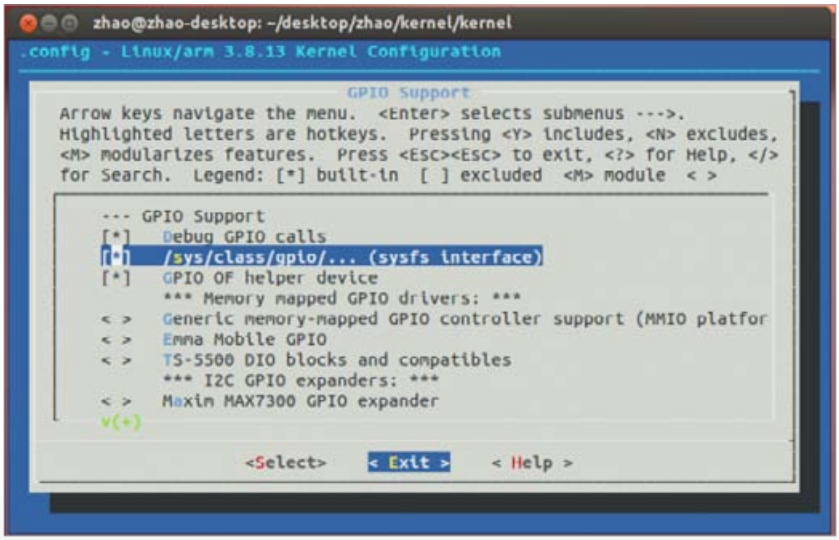


图 1-107

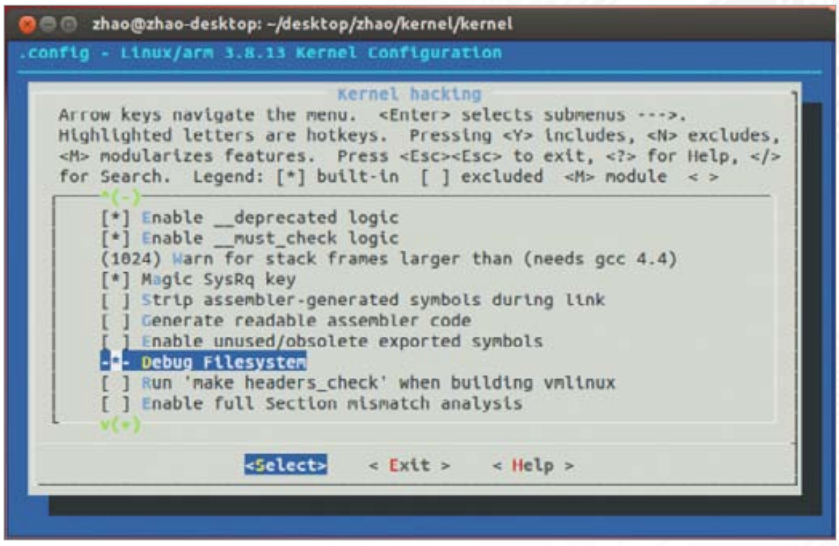


图 1-108

当然，这部分现就不做展开了。别忘了咱们的目的，先入门再说吧。

2. 导出我们需要操作并且可用的 GPIO 口。

3. 配置 GPIO 的 direction 为 in 或 out。

4. 配置 GPIO 作为中断源 (如果有需要) 。需要配置上下沿触发 rising, falling，或者 both，即两者都触发中断。

注意 : 如果配置了 GPIO 作为中断源，那么程序对于该 GPIO 的 value 的读取会一直被阻塞，直到有相应的中断发生。

好了，下面介绍程序部分。

咱们的核心程序是 poll()。这里就简单介绍下吧，深入的我也不会了。

// 下面的都是个人总结，难免有误。如有错误请指正。

poll 函数原型如下 :

int poll(struct pollfd fds[], nfds_tnfds, int timeout);

返回值 :

-1 表示有错误 . 一种可能是调用被中断。

0 表示在文件描述符就绪之前调用超时。

>0 表示一个或多个文件描述符就绪，在 revents 域中可以读取返回值 .

参数 :

参数 fds 文件描述符数组。

它是一个如下结构体

```
struct pollfd
{
intfd; /* File descriptor */
shortevents; /* Requested events bit mask */
shortrevents; /* Returned events bit mask */
};
```

参数nfds_t unsigned int类型数据，表示poll()中共需要轮询处理的文件描述符的个数。

参数 timeout 超时时间，单位毫秒。timeout = -1，则持续等待；timeout=0，直接运行而不等待；timeout>0，则等待 timeout 时间。

关于 pollfd 结构体，也简单介绍下。

events 和 revents 这两个域都是 bit masks 操作，下表列举了每个域中的各种值。

Bit	Input in *events?*	Returned in *revents?*	Description 除了高优先级的数据都可以读
POLLIN	•	•	Data other than high-priority data can be read
POLLRDNORM	•	•	Equivalent to POLLIN
POLLRDBAND	•	•	Priority data can be read (unused on Linux)
POLLPRI	•	•	High-priority data can be read 高优先级的数据可以读
POLLRDHUP	•	•	Shutdown on peer socket
POLLOUT	•	•	Normal data can be written 普通数据可写
POLLWRNORM	•	•	Equivalent to POLLOUT
POLLWRBAND	•	•	Priority data can be written 优先级数据可写
POLLERR		•	An error has occurred 有错误发生
POLLHUP		•	A hangup has occurred
POLLNVAL		•	File descriptor is not open 文件描述符未打开
POLLMSG			Unused on Linux (and unspecified in SUSv3)

表 1-12

第一组的几位和输入事件相关第二组的几位和输出事件相关。

第三组的几位返回的是文件描述符的附加信息。

最后一个位保留

这里可能对POLLIN和POLLPRI疑惑比较大，我当时也是，找了好多资料都没怎么讲到。

就近似这么理解吧：

POLLPRI用于高优先级数据输入，也就是紧急数据的读物，POLLIN处理的是普通数据。

补充：

http://blog.csdn.net/jnu_simba/article/details/8806654

这篇博客讲得不错，关于文件描述符的。下面摘录其中一部分：

用户程序不能直接访问内核中的文件描述符表，而只能使用文件描述符表的索引（即 0 1 2 3 ...），这些索引就称为文件描述符 (File Descriptor)，用 int 型变量保存。当调用 open 打开一个文件或创建一个新文件时，内核分配一个文件描述符并返回给用户程序，该文件描述符表项中的指针指向新打开的文件。当读写文件时，用户程序把文件描述符传给 read 或 write，内核根据文件描述符找到相应的表项，再通过表项中的指针找到相应的文件。

默认情况下（没有重定向），每个进程的标准输入 (stdin)、标准输出 (stdout) 和标准错误输出 (stderr) 都指向控制终端，因为在程序启动时（在 main 函数还没开始执行之前）会自动把控制终端打开三次，分别赋给三个 FILE* 指针 stdin、stdout 和 stderr，这三个文件指针是 libc 中定义的全局变量，这三个文件的描述符分别是 0、1、2，保存在相应的 FILE 结构体中。进程从标准输入读也就是读用户的键盘输入，进程往标准输出或标准错误输出写也就是输出到显示器上。头文件 unistd.h 中有如下的宏定义来表示这三个文件描述符：

```
#define STDIN_FILENO 0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2
```

好了，有了前面的铺垫，上代码。

改编自 BridgeRun 的源程序。

>> 下载源程序

http://bbs.eeworld.com.cn/forum.php?mod=attachment&aid=MTQwODAxfDg2Mjg5ZTg3fDEzOTMzMTkxMzV8NTI4MzcwfDQyODYwMQ%3D%3D

以下是改过的程序，多少有点差别。但是人家的思想很好，很有借鉴意义，是吧？嘿嘿。

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>
using namespace std;
#define POLL_TIMEOUT (3 * 1000) //超时时间定为3s
#define MAX_BUF 64
typedef unsigned char uint8_t;
```

// 注意以下的几个函数，都做成了独立可复用的函数模块。这个思想很值得我们学习借鉴的。

```
//gpio export function
```

```
int gpioExport(uint8_t gpioPin)
{
int fd, len;
char buf[MAX_BUF];
fd = open( "/sys/class/gpio/export", O_WRONLY);
if (fd < 0)
{
perror("gpio/export");
return fd;
}
len = snprintf(buf, sizeof(buf), "%d", gpioPin);
write(fd, buf, len);
close(fd);
return 0;
}

// gpio unexport function
int gpioUnexport(uint8_t gpioPin)
{
int fd, len;
char buf[MAX_BUF];
fd = open( "/sys/class/gpio/unexport", O_WRONLY);
if (fd < 0)
{
perror("gpio/export");
return fd;
}
len = snprintf(buf, sizeof(buf), "%d", gpioPin);
write(fd, buf, len);
close(fd);
return 0;
}

//gpio set direction function
int gpioSetDirection(uint8_t gpioPin, uint8_t outdir)
{
int fd, len;
char buf[MAX_BUF];
len = snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/direction", gpioPin);
fd = open(buf, O_WRONLY);
if (fd < 0)
{
perror("gpio/direction");
return fd;
}
if (outdir)
{
strcpy(buf, "out");
}
else
{
```

```
strcpy(buf, "in");
}
len = strlen(buf);
write(fd, buf, len);
close(fd);
return 0;
}

// gpio set value function
int gpioSetValue(uint8_t gpioPin, uint8_t value)
{
int fd, len;
char buf[MAX_BUF];
len = snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/value", gpioPin);
fd = open(buf, O_WRONLY);
if (fd < 0)
{
perror("gpio/set-value");
return fd;
}
if (value)
{
strcpy(buf, "1");
}
else
{
strcpy(buf, "0");
}
len = strlen(buf);
write(fd, buf, len);
close(fd);
return 0;
}

//gpio get value function
int gpioGetValue(uint8_t gpioPin, uint8_t *value)
{
int fd;
char buf[MAX_BUF];
char ch;
snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/value", gpioPin);
fd = open(buf, O_RDONLY);
if (fd < 0)
{
perror("gpio/get-value");
return fd;
}
read(fd, &ch, 1);
if (ch != '0')
{
```

```
*value = 1;
}
else
{
*value = 0;
}
close(fd);
return 0;
}

//gpio set edge function
int gpioSetEdge(uint8_t gpioPin, const char *edge)
{
int fd;
char buf[MAX_BUF];
snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/edge", gpioPin);
fd = open(buf, O_WRONLY);
if (fd < 0)
{
perror("gpio/set-edge");
return fd;
}
write(fd, edge, strlen(edge) + 1);
close(fd);
return 0;
}

// gpio fd open function
int gpioFdOpen(uint8_t gpioPin)
{
int fd;
char buf[MAX_BUF];
snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/value", gpioPin);
fd = open(buf, O_RDONLY | O_NONBLOCK );
if (fd < 0)
{
perror("gpio/fd_open");
}
return fd;
}

// close a gpio with the file descriptor we have known.
int gpioFdClose(int fd)
{
return close(fd);
}
```

// 到此为止就是几个函数模块的声明与实现。

// 下面就是主函数。最重要的当然要看 poll() 了。

```
int main(int argc, char **argv, char **envp)
{
struct pollfd fdset[2];
int nfds = 2;
int gpio_fd, timeout, rc;
char *buf[MAX_BUF];
uint8_t gpioPin, ledPin;
bool isLEDFlashing = false;

if (argc < 3)
{
printf("Usage: HelloBBB <input-gpio-num> <led-gpio-num>\n");
printf("Waits for a change in the GPIO pin voltage level or input on stdin\n");
exit(-1);
}
gpioPin = atoi(argv[1]); // string to int
ledPin = atoi(argv[2]);
gpioExport(gpioPin);
gpioSetDirection(gpioPin, 0); //input
gpioSetEdge(gpioPin, "rising");
gpio_fd = gpioFdOpen(gpioPin); //the open file descriptor for input gpio.
gpioExport(ledPin);
gpioSetDirection(ledPin, 1); //led output
timeout = POLL_TIMEOUT;
while(1)
{
memset((void *)fdset, 0, sizeof(fdset) ); //fill with 0, clear the memory.
fdset[0].fd = STDIN_FILENO;
fdset[0].events = POLLIN; //Data other than high-priority data can be read
fdset[1].fd = gpio_fd;
fdset[1].events = POLLPRI; //High-priority data can be read
rc = poll(fdset, nfds, timeout);
if (rc < 0) // if poll is unsuccessful, returns a negative value.
{
printf("\npoll() failed!\n");
return -1;
}
if (rc == 0) //timeout, loop again.
{
printf("#");
}
if (fdset[1].revents & POLLPRI) // if the button was pressed, high-priority value
{
read(fdset[1].fd, buf, MAX_BUF);
printf("\n poll() GPIO %d interrupt occurred\n", gpioPin);
if(isLEDFlashing)
{
gpioSetValue(ledPin, 1);
}
else
{
```

```
gpioSetValue(ledPin, 0);
}
isLEDFlashing = !isLEDFlashing;
}
if (fdset[0].revents & POLLIN)
{
(void)read(fdset[0].fd, buf, 1);
printf("\n poll() stdin read 0x%2.2X\n", (unsigned int) buf[0]);
}
fflush(stdout);
}
gpioFdClose(gpio_fd);
return 0;
}
```

看看运行是什么下效果吧。这里的参数参数 GPIO 口为什么是 26 和 44 就不用再解释了吧。

结果还算满意。怎么样，很有成就感吧？

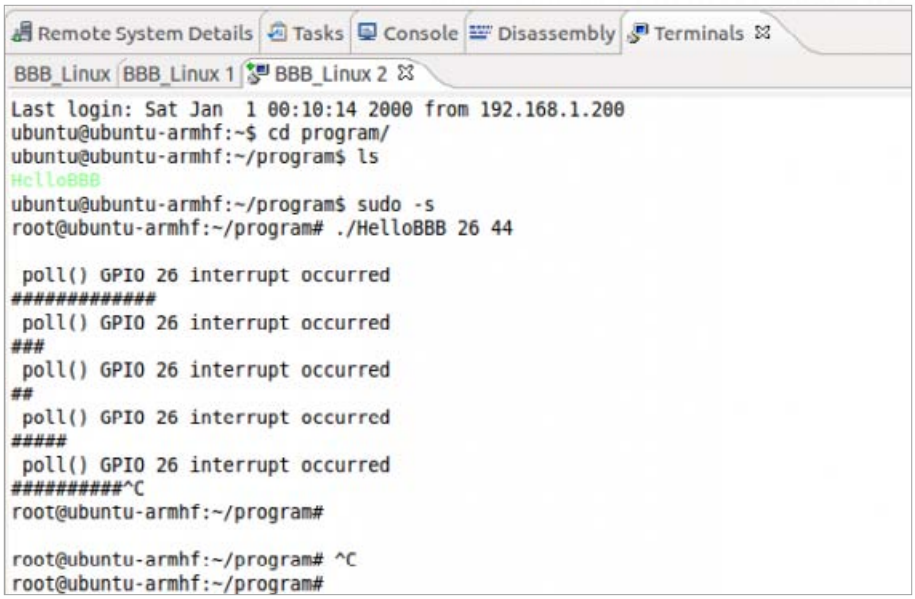


图 1-109

好啦。今天就到这里吧，程序部分个人感觉还算好，就是开始接触 Linux 系统接口会有那么一个过程。

其他的暂时没什么要注意的了吧。今天的知识点不多，但足够消化一会儿啦。

点击查看详情：>> BB Black 入门基础之初识 GPIO

拓展阅读：>>GPIO 扩展阅读

1.3.2 BeagleBone Black 的 ADC 的使用

BeagleBone Black 的处理器芯片 AM3359 上的 8 个 ADC 脚同时也是触摸屏控制器 (TSC)，我们知道一般的电阻屏分为 4 线、5 线或 8 线的，除去触摸屏以外的 ADC 接口可以作为普通的 ADC 使用。(我们在查询 TI 公司的 4000 页 AM33xx 手册时需要到 Touchscreen Controller 这一章去找有关 ADC 的内容。)

特别注意：BeagleBone Black 的 ADC 管脚最大只能输入 1.8V!!!

基本使用方法：

因为这个路径太常用，我们先把它存成环境变量

```
# export SLOTS=/sys/devices/bone_capemgr.8/slots
```

加载 BeagleBone 自带的 device tree 文件

```
# echo BB-ADC > $SLOTS
```

这时 /sys/bus/iio/devices 目录下会出现一个 iio:device0 目录 (原本是没有的)。里面有如下内容

```
-r--r--r-- 1 root root 4096 Jan 1 01:25 dev
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage0_raw
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage1_raw
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage2_raw
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage3_raw
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage4_raw
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage5_raw
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage6_raw
-rw-r--r-- 1 root root 4096 Jan 1 01:25 in_voltage7_raw
-r--r--r-- 1 root root 4096 Jan 1 01:25 name
drwxr-xr-x 2 root root 0 Jan 1 01:25 power
lrwxrwxrwx 1 root root 0 Jan 1 01:25 subsystem -> ../../../../../bus/iio
-rw-r--r-- 1 root root 4096 Jan 1 01:19 uevent
```

里面的 in_voltage*_raw 文件便是 8 个 ADC 引脚的值。可以用 cat 命令查看其数值

```
# cat in_voltage1_raw

2965
```

BeagleBone 的 ADC 是 12 位的，所以这个数值是 0 到 4095 之间的某个数，对应着 0 到 1.8V 电压。另外 BeagleBone black 只有 0 到 6 这 7 个 ADC 是可被外部使用的，第 7 个可能是连到板子上的某处了。这 7 个管脚的位置如图所示：

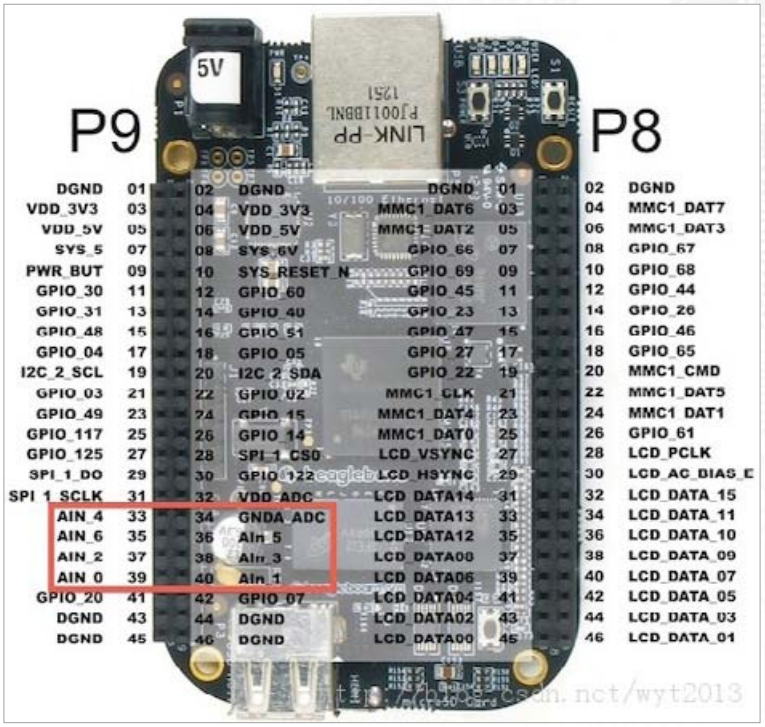


图 1-110

如果想在 c 语言程序中使用 ADC 值，目前的办法是用 read() 函数读取文件中的数值。经过不严谨的测试，发现系统刷新 in_voltage*_raw 文件的速度还是挺快的。我如果每毫秒采一次样，输出数值是在一个小范围内波动的，而且相邻两次的数值很少有相同的。但如果我把采样速度增大 10 倍，每 0.1 毫秒采样一次，就会出现连续重复的数值。可见它的刷新速度是毫秒级的，这应当够一般使用了。

源代码

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h> //define O_WRONLY and O_RDONLY
#define SYSFS_ADC_DIR "/sys/bus/iio/devices/iio:device0/in_voltage3_raw"
#define MAX_BUF 64
void main()
{
int fd, len;
char buf[MAX_BUF];
int ch[5];
int i;
for(i=0;i<50;i++)
{
snprintf(buf,sizeof(buf),SYSFS_ADC_DIR);
fd = open(buf, O_RDONLY);
read(fd,ch,4);
printf("%s\n",ch);
```

```
close(fd);
usleep(1000);//pause for 1 ms
}
}
```

这里只是读取了 in_voltage*_raw 文件中的字符串并输出，没有转变成整型数字，实际使用时肯定还得做转化。

详见：>> 使用 BBB 的 ADC

1.3.3 BeagleBone Black 的串口 UART 的使用

BeagleBone Black 上有 UART1-UART5 共 5 个可用的 uart 串口，UART0 连到了 BeagleBone Black 新增的串口调试引脚上，我们最后再来说它。我们进入 /lib/firmware 目录中可以看到系统自带了一些与 uart 有关的 device tree

```
/lib/firmware/BB-UART1-00A0.dts
/lib/firmware/BB-UART2-00A0.dts
/lib/firmware/BB-UART4-00A0.dts
/lib/firmware/BB-UART5-00A0.dts
```

(不知道为何没有 UART3)

以 UART1 为例，打开 BB-UART1-00A0.dts 文件我们会看到这部分内容

```
    exclusive-use =
        /* the pin header uses */
        "P9.24",    /* uart1_txd */
        "P9.26",    /* uart1_rxd */
        /* the hardware ip uses */
        "uart1";
```

注释写得很清楚，P9.24 和 P9.26 分别用作了 txd 和 rxd 引脚。因此我们就可以将这两个引脚连同地线这 3 根线连上串口转 USB 模块，稍后把模块插到 usb 接口上。

下面我们挂载 device tree 来启动 UART1

echo BB-UART1 > $SLOTS

进入 /dev 目录会发现比原来多了一个设备 ttyO1(注意是大写字母 O，不是数字 0)。

这时 BeagleBone 已经准备好了，下面准备一下电脑端。

首先要确认已经装了电脑端的 PL2303 驱动，mac 版的可以在 http://pbxbook.com/other/mac-tty.html#minicom 找到。win 版的一搜便是，网上有很多。

然后需要一个串口助手。mac 系统自带了一个在终端里运行的串口助手，名字叫 screen，我们就先用它做实验 (如果不习惯命令行，你也可以在网上找到各种有图形界面的串口助手)。

此时电脑端也准备好了。下面开始连接。

把 usb 转串口模块插到电脑上，然后在终端中进入电脑的 /dev 目录，会发现多了一个 cu.usbserial 文件。输入

screen /dev/cu.usbserial 9600

不出错的话会成功打开串口，波特率为 9600，终端界面也会清空，等待接收数据。

在 BeagleBone Black 的终端里，我们输入

echo "What a wonderful day" > /dev/ttyO1

如果在 screen 程序的终端上显示出同样的内容，串口测试就成功了。

如果你不小心把 screen 程序终端关闭了，那么再打开一个新的终端的话，会发现连接不上 cu.usbserial 了，是因为刚刚的串口进程还没关闭，串口还是被占用状态。在电脑端的终端中输入 top 命令查看进程，找到 screen 进程，记住 PID 进程号，然后按 q 键退出查看进程，然后输入 kill PID 号 杀掉之前的 screen 进程就可以重新连接了。

如何调波特率?

在 BeagleBone Black 的终端上，输入

stty -F /dev/ttyO1 38400

就可以把 ttyO1 的波特率改成 38400 了。波特率设置不能超过终端的最大速度，可以输入 stty speed 查看终端的最大速度。输入 stty --help 可以看到 stty 命令更详细的参数。

最后我们说说 UART0，BeagleBone Black 比上一代 BeagleBone 新增了串口调试引脚，我们直接把串口转 usb 模块的 TXD，RXD 和地线连到对应的引脚上（如图所示），无需进行任何配置，立刻就能开始使用了。

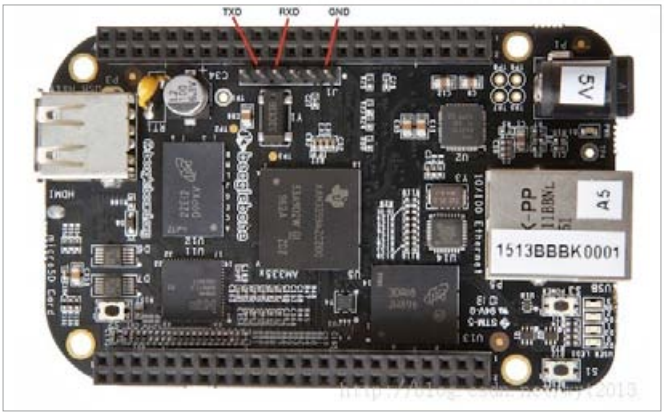


图 1-111

使用方法类似，在电脑端输入（注意这个波特率超过了之前说的最大值）

screen /dev/cu.usbserial 115200

就能打开串口调试助手。在 BBB 上输入

echo "What a wonderful day" > /dev/ttyO0

测试发现没有问题。

详见：>>BeagleBone Black 的串口 UART 的使用

1.3.4 BeagleBone Black 的 I2C 的使用

本文我将使用 BBB 的 I2C1 读取气压传感器芯片 BMP085 和三轴陀螺仪 L3G4200D 的值。

首先说明，我使用的硬件外设是一个九轴气压传感器模块，包括一个三轴陀螺仪芯片，一个三轴磁场传感器芯片，一个三轴加速度计芯片和一个气压传感器芯片。这 4 个芯片使用同一个 I2C 与 BBB 通信，下面记录一下它的使用方法。

I2C 是一种串行通讯方法，它只需要两根线就能实现通讯，一根时钟线 SCL，一根数据线 SDA。一般情况下这两根线都使用上拉电阻，同时把芯片的管脚设置成开漏输出（简单理解开漏输出的含义就是:让它输出低电平时，它能输出低电平;而让它输出高电平时，它就断路，什么也不输出，由外接电平决定这个引脚的电平）。如果芯片内部带有上拉电阻（比如 BBB 的芯片就自带上拉电阻），那不外接上拉也可以。

BBB 系统自带了一个 Linux下的 I2C 工具 i2c-tools，非常好用，下文以 i2c 开头的命令都是这个工具包里的，如果你的系统里没有的话，可以搜索并下载 i2c-tools 工具包。

BBB 上有两个可用的 I2C，i2c-0 和 i2c-1，分别对应 header 上的 I2C1 和 I2C2（总是这么混乱 = =）。我们这里使用 i2c-1，对应的 header 是 P9_19 和 P9_20。我怎么知道它有两个可用的 I2C 呢? 使用命令 i2cdetect -l 就可以看到

```
i2c-0 i2c OMAP I2C adapter I2C adapter
i2c-1 i2c OMAP I2C adapter I2C adapter
i2c-0  i2c          OMAP I2C adapter                    I2C adapter
i2c-1   i2c          OMAP I2C adapter                    I2C adapter
```

OK,下面开始操作。

检查引脚功能配置

首先确认一下 i2c-1 对应的 IO 口复用功能是否正确（BBB 默认就是正确的，所以无需进行配置）。查表得，P9_19 和 P9_20 分别对应 95 和 94 号引脚。（在新页面打开图片可看到完整图）

表 1-13

输入命令

```
# cat /sys/kernel/debug/pinctrl/44e10800.pinmux/pins | grep 97c
# cat /sys/kernel/debug/pinctrl/44e10800.pinmux/pins | grep 97c
```

上述命令把 pins 这个文件中包含 97c 的内容输出（97c 是表中看到的引脚地址），得到 95 号引脚的功能和复用寄存器值

```
pin 95 (44e1097c) 00000073 pinctrl-single
pin 95 (44e1097c) 00000073 pinctrl-single
```

95 号引脚的功能寄存器值是 0x00000073，化成 2 进制是 1110011，其含义是：启用功能 3（即 I2C2_SCL），使能上下拉，开启上拉（所以我们可以不必外接上拉电阻了），使能输入，高速模式。同样可以检查 94 号引脚，也是 0x00000073。

查找 i2c 设备的地址

（此时我们还没有插入设备）使用命令 # i2cdetect -y -r 1 ，可以查看 i2c 设备地址。其中 -y 选项用来屏蔽讨厌的确认环节，-r 是因为 AM3359 不支持一种叫做 Quick Write 的东东，1 代表我们要查看 i2c-1 总线上的设备（也就是 P9_19 和 P9_20 上插着的设备）。输出如下

```
0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

这里的地址都是 16 进制表示的。-- 代表该地址没有设备，UU 代表这个地址正忙（也许被内部资源占用了，见图中的 0x54 到 0x57 这 4 个地址）。下面我把 i2c 设备插上以后再执行命令，输出变成了

```
0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- 53 UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- 53 UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

会发现多出了4个地址：0x1e，0x53，0x69 和 0x77。因为我接入的模块上包含 4 个芯片，所以这个总线上显示了4个地址。通过读芯片手册得知气压计对应的是 0x77 这个地址。这一步就完成了。

需要补充说明的是，这里显示的是 i2c 设备的地址（1110111b=0x77），i2c 的设备地址只有 7 位，最高位当做 0。而读 / 写地址则在最低位增加一个 1/0（11101111b / 11101110b），这使得读写地址与设备地址看起来很不相同。

查看和修改设备的寄存器值

输入命令 # i2cdump -y 1 0x77，我们可以查看设备的寄存器值，其中 -y 还是屏蔽确认环节，1 还是代表查看 i2c-1 总线，0x77 是要查询的设备地址。

```
0 1 2 3 4 5 6 7 8 9 a b c d e f 0123456789abcdef
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
80: a5 94 4d 19 b3 27 38 43 8a 2a 1e 05 fb af c7 6e ??M??'8C?*?????n
90: 84 df 5f b0 56 5a 15 7a 00 3a 80 00 d4 bd 09 80 ??_?VZ?z.:?.????
a0: a5 94 4d 19 b3 27 38 43 8a 2a 1e 05 fb af c7 6e ??M??'8C?*?????n
b0: 84 df 5f b0 56 5a 15 7a 00 3a 80 00 d4 bd 09 80 ??_?VZ?z.:?.????
c0: 00 00 bc 33 00 00 00 00 00 00 00 10 00 00 00 03 ..?3.......?...?
d0: 55 02 06 00 00 00 00 00 00 00 00 00 00 00 00 00 U??.............
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
f0: 00 00 00 00 00 00 80 00 00 00 00 00 00 00 00 00 ......?.........
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
80: a5 94 4d 19 b3 27 38 43 8a 2a 1e 05 fb af c7 6e    ??M??'8C?*?????n
90: 84 df 5f b0 56 5a 15 7a 00 3a 80 00 d4 bd 09 80    ??_?VZ?z.:?.????
a0: a5 94 4d 19 b3 27 38 43 8a 2a 1e 05 fb af c7 6e    ??M??'8C?*?????n
b0: 84 df 5f b0 56 5a 15 7a 00 3a 80 00 d4 bd 09 80    ??_?VZ?z.:?.????
```

```
c0: 00 00 bc 33 00 00 00 00 00 00 00 10 00 00 00 03     ..?3.......?...?
d0: 55 02 06 00 00 00 00 00 00 00 00 00 00 00 00 00     U??.............
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
f0: 00 00 00 00 00 00 80 00 00 00 00 00 00 00 00 00     ......?.........
```

这里用 16 进制显示了每个寄存器的值，具体哪个寄存器是干嘛的，就得查阅芯片的数据手册了。

如果想取出某个特定寄存器的值，比如 0x80 寄存器，可以使用命令 i2cget -y 1 0x77 0x80 实现。如果想向某个寄存器写入值,可使用命令 i2cset -y 1 0x69 0x20 0x0f 实现。(向 i2c-1 总线上，设备地址为 0x69，寄存器地址为 0x20 处写入值 0x0f)

使用 BMP085 驱动读取气压值

很巧的是，BBB 自带了 BMP085 气压芯片的驱动，所以我们可以更方便地读取气压值。输入命令

```
# echo bmp085 0x77 > /sys/class/i2c-adapter/i2c-1/new_device
# echo bmp085 0x77 > /sys/class/i2c-adapter/i2c-1/new_device
```

加载成功后使用如下命令就可以读取气压值了

```
# cat sys/bus/i2c/drivers/bmp085/1-0077/pressure0_input
# cat sys/bus/i2c/drivers/bmp085/1-0077/pressure0_input
```

使用完毕以后，我们可以用如下命令从驱动中卸载这个 i2c 设备

```
# echo 0x77 > /sys/class/i2c-adapter/i2c-1/delete_device
# echo 0x77 > /sys/class/i2c-adapter/i2c-1/delete_device
```

想查看加载或卸载是否成功的话，可以输入如下命令

```
# dmesg | grep bmp
# dmesg | grep bmp
```

你应该会看到类似下面的输出

```
[ 6428.602566] i2c i2c-1: new_device: Instantiated device bmp085 at 0x77
[ 6428.633419] bmp085 1-0077: Successfully initialized bmp085!
[ 6436.479407] i2c i2c-1: delete_device: Deleting device bmp085 at 0x77
[ 6428.602566] i2c i2c-1: new_device: Instantiated device bmp085 at 0x77
[ 6428.633419] bmp085 1-0077: Successfully initialized bmp085!
[ 6436.479407] i2c i2c-1: delete_device: Deleting device bmp085 at 0x77
```

问题

但是 BBB 没有带其他芯片的驱动，我暂时还不知道如何自己写驱动。(见最后更新) 更奇怪的是我查看其他设备地址的寄存器，得到很诡异的输出。

比如 # i2cdump -y 1 0x69 c 时 :

```
0 1 2 3 4 5 6 7 8 9 a b c d e f 0123456789abcdef
```

```
00: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
10: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
20: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
30: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
40: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
50: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
60: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
70: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
80: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
90: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
a0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
b0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
c0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
d0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
e0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
f0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 ????????????????
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
10: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
20: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
30: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
40: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
50: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
60: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
70: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
80: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
90: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
a0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
b0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
c0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
d0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
e0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
f0: c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4 c4    ????????????????
```

更新解答上面的问题

是输出模式的原因。# i2cdump -y 1 0x69 c 命令中的 c 代表连续字节输出。把它去掉，用默认的字节输出模式就可以了。

```
0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  0123456789abcdef
00: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3 ?f??L?????$?;?.?
10: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83 D?”???@`????????
20: 07 00 00 00 00 00 00 00 00 f0 fb 06 ce fc 00 20 ?........?????.
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
80: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3 ?f??L?????$?;?.?
90: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83 D?" ???@`????????
a0: 07 00 00 00 00 00 00 00 00 f0 fb 06 ce fc 00 20 ?........?????.
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3    ?f??L?????$?;?.?
10: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83    D?" ???@`????????
20: 07 00 00 00 00 00 00 00 00 f0 fb 06 ce fc 00 20    ?........?????.
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
80: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3    ?f??L?????$?;?.?
90: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83    D?" ???@`????????
a0: 07 00 00 00 00 00 00 00 00 f0 fb 06 ce fc 00 20    ?........?????.
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

再多说两句，这个设备是 L3G4200D 三轴陀螺仪，通过查芯片手册得知，0x20 寄存器的第 4 位是芯片使能，现在 0x20 寄存器的值是 0x07，即 00000111b，我要把它变成 00001111b，即 0x0f，输入命令 # i2cset -y 1 0x69 0x20 0x0f

```
0 1 2 3 4 5 6 7 8 9 a b c d e f 0123456789abcdef
00: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3 ?f??L?????$?;?.?
10: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83 D?" ???@`????????
20: 0f 00 00 00 00 00 1d ff 33 00 d5 ff fc ff 00 20 ?.....?.3.?.?..
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
80: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3 ?f??L?????$?;?.?
90: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83 D?" ???@`????????
a0: 0f 00 00 00 00 00 1d ff 52 00 f7 ff 24 00 00 20 ?.....?.R.?.$..
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ............⋯.
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
```

```
00: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3    ?f??L?????$?;?.?
10: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83    D?”???@`????????
20: 0f 00 00 00 00 00 1d ff 33 00 d5 ff fc ff 00 20    ?.....?.3.?.?..
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
80: c4 66 a5 cc 4c d0 89 c1 c2 e0 24 18 3b 87 00 d3    ?f??L?????$?;?.?
90: 44 b5 22 04 0c 80 40 60 1a 11 ee 90 83 01 85 83    D?”???@`????????
a0: 0f 00 00 00 00 00 1d ff 52 00 f7 ff 24 00 00 20    ?.....?.R.?.$..
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ................
```

就成功使能陀螺仪了。根据芯片手册，0x28 到 0x2d 这 6 个寄存器存放的是三轴角速度的值，所以我们比较上面两次 i2cdump 的结果会发现，这几个寄存器是有变化的，说明芯片正常工作了。

再次更新：其实 BBB 带了其他芯片驱动的……

使用 L3G4200D 三轴陀螺仪，需要先加载 st_sensors.ko 和 st_sensors_i2c.ko，然后再加载 st_gyro.ko 和 st_gyro_i2c.ko。方法是

```
# insmod /lib/modules/3.8.13/kernel/drivers/iio/gyro/st_gyro.ko
# insmod /lib/modules/3.8.13/kernel/drivers/iio/gyro/st_gyro.ko
```

其他 3 个类似。加载完 4 个模块以后(可以用 lsmod 命令确认一下),跟上述类似的方法，加载 l3g4200d 这个 i2c 设备。

```
# echo l3g4200d 0x69 > /sys/bus/i2c/devices/i2c-1/new_device
# echo l3g4200d 0x69 > /sys/bus/i2c/devices/i2c-1/new_device
```

注　意，/sys/class/i2c-adapter/i2c-1/new_device　和 /sys/bus/i2c/devices/i2c-1/new_device 这两个目录其实是一样的。都是指向同一个地方的链接。

完成以后就会发现在 /sys/bus/i2c/devices 目录下多了一个 1-0069 目录，里面有一个 iio:device0 目录,这个目录下就是上面加载的 4 个驱动提供的用户接口,即 3 个轴的角速度等。我们可以用 cat 命令读取文件内容来观察，是在不断变化的。

使用 adxl345 加速度计时出了点问题，加载驱动方法类似，没什么问题：

```
# insmod /lib/modules/3.8.13/kernel/drivers/input/misc/adxl34x.ko
```

```
# insmod /lib/modules/3.8.13/kernel/drivers/input/misc/adxl34x-i2c.ko
# insmod /lib/modules/3.8.13/kernel/drivers/input/misc/adxl34x.ko
# insmod /lib/modules/3.8.13/kernel/drivers/input/misc/adxl34x-i2c.ko
```

但是在执行下面命令之后，

```
# echo adxl34x 0x1e > /sys/bus/i2c/devices/i2c-1/new_device
# echo adxl34x 0x1e > /sys/bus/i2c/devices/i2c-1/new_device
```

虽然在 /sys/bus/i2c/devices 目录下多了一个 1-001e 目录，但是该目录下没有出现想要的有关加速度计的内容。经过

```
# dmesg | grep adxl34x | tail -5
# dmesg | grep adxl34x | tail -5
```

显示记录，发现给出了一个错误提示：

```
[ 6960.709459] adxl34x 1-001e: no IRQ?
[ 6960.709459] adxl34x 1-001e: no IRQ?
```

这是驱动源码给出的错误信息。但如何解决就不清楚了，因为还不太了解驱动的工作原理。

这个过程表明，即便不存在的设备或有问题的设备，使用类似命令 echo “设备名”“I2C 地址” > /sys/bus/i2c/devices/i2c-1/new_device 也能创建一个设备节点，只不过里面什么有用的东西都没有就是了。

另外关于 /sys 目录下的内容简单记录一下：

在 /sys 目录下有 block, bus, class, dev, devices, firmware, fs, kernel, module, power 这些子目录。

devices 目录是这里面最重要的目录，这是内核对系统中所有设备的分层次表达模型。

block、bus、class、dev 这 4 个目录分别是所有的块设备、按总线分类、按类别分类和按主次设备号分类的设备列表。实际上这 4 个目录的内容最后都是指向 devices 目录里相应设备的链接。相当于给我们提供了几种不同的方式找到想要找的设备。

其他目录的内容如下：

module 目录里显示着几乎所有已经加载的驱动模块的信息，包括以内联方式编译到内核映像文件中的模块和外部模块 (ko 文件)。

kernel 目录是内核所有可调整参数的位置，目前只有几项较新的设计在使用它，其它内核可调整参数仍然位于 sysctl (/proc/sys/kernel) 接口中。

fs 目录按照设计是用于描述系统中所有文件系统，但目前只有少数文件系统支持 sysfs 接口，一些传统的虚拟文件系统 (VFS) 层次控制参数仍然在 sysctl (/proc/sys/fs) 接口中。

firmware 目录是系统加载固件机制的对用户空间的接口。

power 目录是系统中电源选项，这个目录下有几个属性文件可以用于控制整个机器的电源状态，如可以向其中写入控制命令让机器关机、重启等。

点击查看详情：>> 使用 Beaglebone Black 的 I2C

1.3.5 BeagleBone Black 的 SPI 的使用

SPI 是可以全双工通信的一种串行总线，两个设备之间双向通信的话一般使用 3 根线：SCLK，MISO，MOSI，多个设备之间双向通信的话，每个设备还需要再加上一根地址线 CSn。相比之下 I2C 只能半双工，而且一般需要上拉电阻，但无论几个设备，都只需要 2 根线。更多基础知识请谷歌百度。

BeagleBone Black 使用的 AM3359 芯片上有两个 SPI，但 SPI1 连接到了板子的 HDMI 芯片上，所以除非禁用 HDMI，否则我们只能使用 SPI0。本文将利用自带的 spidev 驱动使能 SPI0，并进行一下简单的验证。

配置 device tree

首先我们用我在《使用 BBB 的 I2C》这篇文章中使用的方法检验一下 SPI 相关的引脚功能是否配置正确。检查结果是，不正确，也就是说 SPI 默认是没有启用的，新版 arm linux 配置硬件的方式是利用 device tree，所以我们必须要配置一个 device tree 来启用它。我们先到 /lib/firmware 目录中看看有没有现成的 device tree source (.dts) 文件可供使用。我们发现有一个 BB-SPI0-00A0.dts。内容如下

```
/dts-v1/;
/plugin/;

/ {
compatible = “ti,BeagleBone”, “ti,BeagleBone-black”;

/* identification */
part-number = “BB-SPI0”;
version = “00A0”;

/* state the resources this cape uses */
exclusive-use =
/* the pin header uses */
 “P9.17”, /* spi0_cs0 */
 “P9.18”, /* spi0_d1 */
 “P9.21”, /* spi0_d0 */
 “P9.22”, /* spi0_sclk */
/* the hardware ip uses */
 “spi0”;

fragment@0 {
target = <&am33xx_pinmux>;
__overlay__ {
/* default state has all gpios released and mode set to uart1 */
bb_spi0_pins: pinmux_bb_spi0_pins {
pinctrl-single,pins = <
0x150 0x30 /* spi0_sclk.spi0_sclk, INPUT_PULLUP | MODE0 */
0x154 0x30 /* spi0_d0.spi0_d0, INPUT_PULLUP | MODE0 */
0x158 0x10 /* spi0_d1.spi0_d1, OUTPUT_PULLUP | MODE0 */
```

```
0x15c 0x10 /* spi0_cs0.spi0_cs0, OUTPUT_PULLUP | MODE0 */
>;
};
};
};

fragment@1 {
target = <&spi0>; /* spi0 is numbered correctly */
__overlay__ {
status = “okay”;
pinctrl-names = “default”;
pinctrl-0 = <&bb_spi0_pins>;

 #address-cells = <1>;
 #size-cells = <0>;

/* add any spi devices connected here */
/* note that you can do direct SPI via spidev now */

// commented out example of an adafruit 1.8” TFT display
// from firmare/capes/cape-bone-adafruit-lcd-00A0.dts
// lcd@0 {
// #address-cells = <1>;
// #size-cells = <0>;
//
// compatible = “adafruit,tft-lcd-1.8-red”, “sitronix,st7735”;
// reg = <0>;
//
// spi-max-frequency = <8000000>;
// spi-cpol;
// spi-cpha;
//
// pinctrl-names = “default”;
// pinctrl-0 = <&bone_adafruit_lcd_pins>;
//
// st7735-rst = <&gpio4 19 0>;
// st7735-dc = <&gpio4 21 0>;
// };
};
};
};
/dts-v1/;
/plugin/;

/ {
    compatible = “ti,BeagleBone”, “ti,BeagleBone-black”;

    /* identification */
    part-number = “BB-SPI0”;
    version = “00A0”;
```

```
/* state the resources this cape uses */
exclusive-use =
    /* the pin header uses */
     “P9.17”,     /* spi0_cs0 */
     “P9.18”,     /* spi0_d1 */
     “P9.21”,     /* spi0_d0 */
     “P9.22”,     /* spi0_sclk */
    /* the hardware ip uses */
     “spi0”;

fragment@0 {
    target = <&am33xx_pinmux>;
    __overlay__ {
        /* default state has all gpios released and mode set to uart1 */
        bb_spi0_pins: pinmux_bb_spi0_pins {
            pinctrl-single,pins = <
                0x150 0x30  /* spi0_sclk.spi0_sclk, INPUT_PULLUP | MODE0 */
                0x154 0x30  /* spi0_d0.spi0_d0, INPUT_PULLUP | MODE0 */
                0x158 0x10  /* spi0_d1.spi0_d1, OUTPUT_PULLUP | MODE0 */
                0x15c 0x10  /* spi0_cs0.spi0_cs0, OUTPUT_PULLUP | MODE0 */
            >;
        };
    };
};

fragment@1 {
    target = <&spi0>; /* spi0 is numbered correctly */
    __overlay__ {
        status =  “okay”;
        pinctrl-names =  “default”;
        pinctrl-0 = <&bb_spi0_pins>;

        #address-cells = <1>;
        #size-cells = <0>;

        /* add any spi devices connected here */
        /* note that you can do direct SPI via spidev now */

        // commented out example of an adafruit 1.8”  TFT display
        // from firmare/capes/cape-bone-adafruit-lcd-00A0.dts
        // lcd@0 {
        //  #address-cells = <1>;
        //  #size-cells = <0>;
        //
        //  compatible =  “adafruit,tft-lcd-1.8-red”,  “sitronix,st7735”;
        //  reg = <0>;
        //
        //  spi-max-frequency = <8000000>;
        //  spi-cpol;
        //  spi-cpha;
        //
```

```
            //  pinctrl-names = “default”;
            //  pinctrl-0 = <&bone_adafruit_lcd_pins>;
            //
            //  st7735-rst = <&gpio4 19 0>;
            //  st7735-dc = <&gpio4 21 0>;
            // };
        };
    };
};
```

从这个文件里我们能得到很多信息(我在此唠叨两句,也算跟大家分享一下我学习的过程),首先我们从 exclusive-use 这一部分能看出来 AM3359 芯片对 SPI 引脚的命名是跟一般不太一样的，它没用 MISO 和 MOSI，而是 D0 和 D1。通过查询 4000 页手册我们得知，原来是因为这两个引脚的功能是可以通过配置寄存器来互换的。默认的对应方式如下

Table 26-30. Pins Used for SPI Boot

Signal name	Pin Used in Device
cs	spi0_cs0
miso	spi0_d0
mosi	spi0_d1
clk	spi0_sclk

表 1-14

再接着看，发现有一句注释

```
/* note that you can do direct SPI via spidev now */
/* note that you can do direct SPI via spidev now */
```

这个spidev就是我们要用的spi驱动，然后谷歌一下它的用法就可以了。再下面有一些被注释掉的东西，是要根据不同设备来替换的。

（以下操作都在 BeagleBone 上进行）

我们把自带的文件复制一份，保存为 BB-SPI0-01-00A0.dts ，然后增加一个节点，内容如下（就是原文件中注释部分要替换的内容）

```
spidev@0 {
spi-max-frequency = <24000000>;
reg = <0>;
compatible = “linux,spidev”;
};
spidev@0 {
        spi-max-frequency = <24000000>;
        reg = <0>;
        compatible = “linux,spidev”;
};
```

保存以后编译这个 dts 文件

```
dtc -O dtb -o BB-SPI0-01-00A0.dtbo -b 0 -@ BB-SPI0-01-00A0.dts
dtc -O dtb -o BB-SPI0-01-00A0.dtbo -b 0 -@ BB-SPI0-01-00A0.dts
```

然后把生成的 .dtbo 文件放到 /lib/firmware 目录中

```
cp BB-SPI0-01-00A0.dtbo /lib/firmware/
cp BB-SPI0-01-00A0.dtbo /lib/firmware/
```

然后把它“插”到“插槽”中（请看我的博文《聊聊 BeagleBone Black 的 cape 和 device tree overlay》）

```
echo BB-SPI0-01 > /sys/devices/bone_capemgr.*/slots
echo BB-SPI0-01 > /sys/devices/bone_capemgr.*/slots
```

OK，这时我们进入 /dev 目录中就会发现比原来多了一个设备 spidev1.0 ，说明 device tree 配置没有问题，该设备已成功加载。

使用 SPI

因为我手边没有 SPI 设备，所以我把 D0 和 D1 也就是 P9.18 和 P9.21 这两个引脚连接起来进行自发自收，如果收到了发送的数据即成功。时钟线就不必管了，因为自己跟自己的时钟肯定是同步的。

测试程序使用的是 linux 自带的一个 spidev_test.c 程序（下载地址是 https://www.kernel.org/doc/Documentation/spi/，不过还是建议直接把整个 kernel 下载下来比较方便搜索查询）。这个程序的内容就是发送一串 16 进制数，然后 printf 接收到的内容（不知道这串数有没有什么别的含义）。

下面把这个文件传到 BeagleBone 上，用 gcc 编译一下，生成可执行文件 spidev_test。假设现在就在这个文件的目录下，那么我们输入

```
./spidev_test -D /dev/spidev1.0
./spidev_test -D /dev/spidev1.0
```

来进行测试。得到输出

```
spi mode: 0
bits per word: 8
max speed: 500000 Hz (500 KHz)
FF FF FF FF FF FF
40 00 00 00 00 95
FF FF FF FF FF FF
FF FF FF FF FF FF
FF FF FF FF FF FF
DE AD BE EF BA AD
F0 0D
spi mode: 0
bits per word: 8
max speed: 500000 Hz (500 KHz)
FF FF FF FF FF FF
40 00 00 00 00 95
FF FF FF FF FF FF
FF FF FF FF FF FF
FF FF FF FF FF FF
DE AD BE EF BA AD
F0 0D
```

说明测试成功了。否则会输出一串 FF。

为什么 dts 文件要那样改?

刚刚我在自带的 BB-SPI0-00A0.dts 文件中加了一个节点，然后向其中加了几个属性，SPI0 就能用了。增加一个节点还能够理解，但为什么要加这几个属性? 这个问题我想了几天也没想得很清楚。不过我知道的是，这 3 个属性缺一不可。

其中 compatible 属性是每个节点必须有的,它的作用是将这个设备和某个驱动进行绑定。比如这里就是将这个 spi 设备与 linux -> spidev 这个驱动绑定。我把逗号换成了箭头，是因为我觉得其实这个逗号表达的是从属关系，用箭头更合适。但是，我在 kernel 文件中翻遍了也没找到哪里有“linux,spidev”这样的字眼。spidev 驱动倒是找到了，而且在这个驱动文件中发现了如下内容

```
static const struct of_device_id spidev_dt_ids[] = {
{ .compatible = “rohm,dh2228fv” },
{},
};
```

我试着把 BB-SPI0-01-00A0.dts 里的 compatible 值换成“rohm,dh2228fv”，结果居然也成功了! 这似乎说明以后如果我们知道要用哪个驱动的话，到驱动文件里搜索 compatible 找到相应内容就可以了。不过，我遗憾地发现大部分驱动文件里都没有这个属性。可能只有一些硬件外设的驱动，或者是别的公司做的驱动里才会有。所以，我又迷惘了……

点击查看详情：>> 用 Beaglebone Black 的 SPI

1.3.6 EE_BeagleBone_Cape LCD

1. 概述

这篇文档是关于 EE_BeagleBone_Cape LCD，内容不是很多，图形界面的编程不再这里加以讨论和学习，主要说说硬件的配置和测试。

2. LCD 硬件设计点滴

我使用的是群创的 7 寸屏，这里主要注意一下 AM3359 的 LCD 控制器数据线和群创 RGB 的连接关系就可以了。如下所示

```
lcd_data0 B1
lcd_data1 B2
lcd_data2 B3
lcd_data3 B4
lcd_data4 B5
lcd_data5 G0
lcd_data6 G1
lcd_data7 G2
lcd_data8 G3
lcd_data9 G4
lcd_data10 G5
lcd_data11 R1
lcd_data12 R2
lcd_data13 R3
lcd_data14 R4
lcd_data15 R5
```

以下两篇文章还是比较有用的：

1) am335x 接入 LVDS 显示器总结

http://bbs.eeworld.com.cn/thread-329624-1-1.html

2) 给 BeagleBone 加装 1280*800 的 LCD 模块

http://bbs.eeworld.com.cn/thread-332175-1-1.html

3. LCD 相关驱动

3.1 初始化

不管加什么模块，第一件事情都是初始化，需要在 BeagleBone_dev_cfg 中增加初始化函数。

```
/* BeagleBone Rev A3 and after */
static struct evm_dev_cfg BeagleBone_dev_cfg[] = {
{mii1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
```

```
{usb0_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{usb1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{mmc0_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{lcdc_init,  DEV_ON_BASEBOARD,, PROFILE_NONE},
{NULL, 0, 0},
};
```

接个我们看看lcdc_init都干了些什么事情，

1. 管脚分配
2. 配置时钟
3. 注册设备

```
static void lcdc_init(int evm_id, int profile)
{

setup_pin_mux(lcdc_pin_mux);


if (conf_disp_pll(300000000)) {
pr_info("Failed configure display PLL, not attempting to"
"register LCDC\n");
return;
}


if (am33xx_register_lcdc(&TFC_S9700RTWV35TR_01B_pdata))
pr_info("Failed to register LCDC device\n");
return;
}
```

3.2 Pin_mux

看看和 LCD 相关的管脚分配吧，也就如下的一些信号了。

```
/* Module pin mux for LCDC */
static struct pinmux_config lcdc_pin_mux[] = {
{"lcd_data0.lcd_data0", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data1.lcd_data1", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data2.lcd_data2", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data3.lcd_data3", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data4.lcd_data4", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data5.lcd_data5", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
```

```
| AM33XX_PULL_DISA},
{"lcd_data6.lcd_data6", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data7.lcd_data7", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data8.lcd_data8", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data9.lcd_data9", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data10.lcd_data10", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data11.lcd_data11", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data12.lcd_data12", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data13.lcd_data13", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data14.lcd_data14", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data15.lcd_data15", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"gpmc_ad8.lcd_data16", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"gpmc_ad9.lcd_data17", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"gpmc_ad10.lcd_data18", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"gpmc_ad11.lcd_data19", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"gpmc_ad12.lcd_data20", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"gpmc_ad13.lcd_data21", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"gpmc_ad14.lcd_data22", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"gpmc_ad15.lcd_data23", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT},
{"lcd_vsync.lcd_vsync", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
{"lcd_hsync.lcd_hsync", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
{"lcd_pclk.lcd_pclk", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
{"lcd_ac_bias_en.lcd_ac_bias_en", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
{NULL, 0},
};
```

这是关于 LCD 的一些配置信息

```
static struct lcd_ctrl_config lcd_cfg = {
&disp_panel,
.ac_bias = 255,
.ac_bias_intrpt = 0,
.dma_burst_sz = 16,
.bpp = 32,
.fdd = 0x80,
.tft_alt_mode = 0,
.stn_565_mode = 0,
.mono_8bit_mode = 0,
.invert_line_clock = 1,
.invert_frm_clock = 1,
```

```
.sync_edge = 0,
.sync_ctrl = 1,
.raster_order = 0,
};

struct da8xx_lcdc_platform_data TFC_S9700RTWV35TR_01B_pdata = {
.manu_name = "ThreeFive",
.controller_data = &lcd_cfg,
.type = "TFC_S9700RTWV35TR_01B",
};
```

我们还是需要深入的了解一下 TFC_S9700RTWV35TR_01B 在什么地方定义的，以后我们改其他的屏就在这个地方了 drivers/video/Da8xx-fb.c ，正好的我的 7 寸屏和这个配置比较匹配，就不需要修改了，省事!

```
static struct da8xx_panel known_lcd_panels[] = {
/* Sharp LCD035Q3DG01 */
[0] = {
.name = "Sharp_LCD035Q3DG01",
.width = 320,
.height = 240,
.hfp = 8,
.hbp = 6,
.hsw = 0,
.vfp = 2,
.vbp = 2,
.vsw = 0,
.pxl_clk = 4608000,
.invert_pxl_clk = 1,
},
/* Sharp LK043T1DG01 */
[1] = {
.name = "Sharp_LK043T1DG01",
.width = 480,
.height = 272,
.hfp = 2,
.hbp = 2,
.hsw = 41,
.vfp = 3,
.vbp = 3,
.vsw = 10,
.pxl_clk = 7833600,
.invert_pxl_clk = 0,
},
/* ThreeFive S9700RTWV35TR */
[2] = {
.name = "TFC_S9700RTWV35TR_01B",
.width = 800,
.height = 480,
.hfp = 39,
```

```
.hbp = 39,
.hsw = 47,
.vfp = 13,
.vbp = 29,
.vsw = 2,
.pxl_clk = 30000000,
.invert_pxl_clk = 0,
},
};

struct da8xx_panel {
const char name[25]; /* Full name _*/
unsigned short width;
unsigned short height;
int hfp; /* Horizontal front porch */
int hbp; /* Horizontal back porch */
int hsw; /* Horizontal Sync Pulse Width */
int vfp; /* Vertical front porch */
int vbp; /* Vertical back porch */
int vsw; /* Vertical Sync Pulse Width */
unsigned int pxl_clk; /* Pixel clock */
unsigned char invert_pxl_clk; /* Invert Pixel clock */
};
```

4. 相关测试结果

图 1-112

点击查看详情：>>EE_BeagleBone_Cape 之 LCD

1.3.7 EE_BeagleBone_Cape 之 EEPROM

1. 概述

这篇文档是关于 EE_BeagleBone_Cape AT24C02 的配置和测试，以及 i2c-tools 的简单的使用。

2. 硬件连接图

我目前使用的是 AT24C02, 不是图上画的 AT24C256; 根据硬件原理图 , 可以看出 AT24C02 的设备 ID 为 0x54

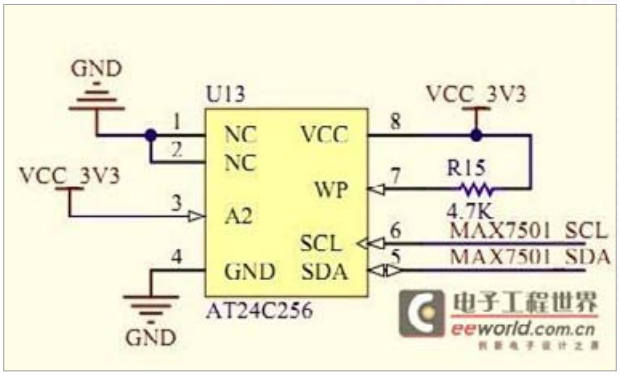


图 1-113

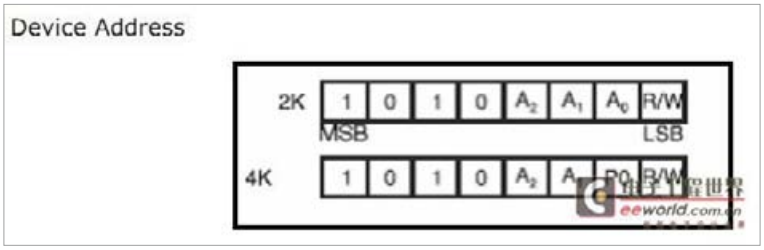


图 1-114

把 BeagleBone P9 的 pin_17(SCL) ， pin_18(SDA) 分别接 AT24C02 的 SCL，SDA 管脚。

图 1-115

3. Linux 配置

3.1 初始化

我这里初始化了 2 个 i2c 接口 , 分别是 i2c1,i2c2, 对应的 linux 设备文件为 i2c-2 和 i2c-3

```
static struct i2c_board_info am335x_i2c1_boardinfo[] = {
{
I2C_BOARD_INFO("tlv320aic3x", 0x1b),
},
{
I2C_BOARD_INFO("tsl2550", 0x39),
},
{
I2C_BOARD_INFO("tmp275", 0x48),
},
{
I2C_BOARD_INFO("24c02", 0x54),
},
};

static void i2c1_init(int evm_id, int profile)
{
setup_pin_mux(i2c1_pin_mux);
omap_register_i2c_bus(2, 100, am335x_i2c1_boardinfo,
ARRAY_SIZE(am335x_i2c1_boardinfo)); //注册为i2c-2，速率100k
return;
}

static struct i2c_board_info am335x_i2c2_boardinfo[] = {
{
 I2C_BOARD_INFO("24c02", 0x54),
},
};

static void i2c2_init(int evm_id, int profile)
{
setup_pin_mux(i2c2_pin_mux);
omap_register_i2c_bus(3, 100, am335x_i2c2_boardinfo,
ARRAY_SIZE(am335x_i2c2_boardinfo)); //注册为i2c-3，速率100k
return;
}
```

3.2 Pin_mux

```
static struct pinmux_config i2c1_pin_mux[] = {
{"spi0_d1.i2c1_sda", OMAP_MUX_MODE2 | AM33XX_SLEWCTRL_SLOW |
AM33XX_PIN_INPUT_PULLUP},
{"spi0_cs0.i2c1_scl", OMAP_MUX_MODE2 | AM33XX_SLEWCTRL_SLOW |
AM33XX_PIN_INPUT_PULLUP},
```

```
{NULL, 0},
};

static struct pinmux_config i2c2_pin_mux[] = {
{“uart1_ctsn.i2c2_sda”, OMAP_MUX_MODE3 | AM33XX_SLEWCTRL_SLOW |
AM33XX_PIN_INPUT_PULLUP},
{“uart1_rtsn.i2c2_scl”, OMAP_MUX_MODE3 | AM33XX_SLEWCTRL_SLOW |
AM33XX_PIN_INPUT_PULLUP},
{NULL, 0},
};
```

4. 启动信息

```
[ 1.096405] registered am33xx_sr device
[ 1.100524] at24 2-0054: 256 byte 24c02 EEPROM, writable, 1 bytes/write
[ 1.107574] at24 3-0054: 256 byte 24c02 EEPROM, writable, 1 bytes/write
[ 1.116729] mtdoops: mtd device (mtddev=name/number) must be supplied
```

5. 测试相关

I2c-tools 相关的命令

```
root@am335x-evm:~# ls /usr/sbin/i2c*
/usr/sbin/i2c-stub-from-dump /usr/sbin/i2cget
/usr/sbin/i2cdetect /usr/sbin/i2cset
/usr/sbin/i2cdump
```

使用 i2cdetect 命令产看一下在 i2c-2 下面挂载的设备

```
root@am335x-evm:~# i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n]
0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- UU -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 51 52 53UU 55 56 57 -- -- -- -- -- -- -- -- 说明0x54上面 有 设备
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

使用 i2cdump 命令查看一下目前 AT24C02 存储的内容

```
root@am335x-evm:~# i2cdump -f -y 2 0x54
No size specified (using byte-data access)
```

```
0 1 2 3 4 5 6 7 8 9 a b c d e f 0123456789abcdef
00: 0c 01 02 ff ff ff ff ff ff ff ff ff ff ff ff ff ???.............
10: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
20: ff ff 22 ff ff ff ff ff ff ff ff ff ff ff ff ff ..".............
30: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
40: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
50: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
60: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
70: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
80: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
90: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
a0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
e0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
f0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
```

使用i2cset命令分别对地址0x40,0x51,0x62写入1,2,3

```
root@am335x-evm:~# i2cset -f -y 2 0x54 0x40 0x1
root@am335x-evm:~# i2cset -f -y 2 0x54 0x51 0x2
root@am335x-evm:~# i2cset -f -y 2 0x54 0x62 0x3
```

使用 i2cget 命令读取对应地址的数据，返回值和写入的是一样的

```
root@am335x-evm:~# i2cget -f -y 2 0x54 0x62
0x03
root@am335x-evm:~# i2cget -f -y 2 0x54 0x63
0xff
root@am335x-evm:~# i2cget -f -y 2 0x54 0x52
0xff
root@am335x-evm:~# i2cget -f -y 2 0x54 0x51
0x02
```

再次查看一下 AT24C02 存储的内容

```
root@am335x-evm:~# i2cdump -f -y 2 0x54
No size specified (using byte-data access)
0 1 2 3 4 5 6 7 8 9 a b c d e f 0123456789abcdef
00: 0c 01 02 ff ff ff ff ff ff ff ff ff ff ff ff ff ???.............
10: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
20: ff ff 22 ff ff ff ff ff ff ff ff ff ff ff ff ff ..".............
30: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
40: 01 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ?...............
50: ff 02 ff ff ff ff ff ff ff ff ff ff ff ff ff ff .?..............
60: ff ff 03 ff ff ff ff ff ff ff ff ff ff ff ff ff ..?.............
70: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
80: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
90: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
```

```
a0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
e0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
f0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
root@am335x-evm:~#
```

点击查看详情：>>EE_BeagleBone_Cape 之 EEPROM

1.3.8 EE_BeagleBone_Cape 4x4 矩阵键盘

1. 概述

这篇文档是关于 EE_BeagleBone_Cape 4x4 矩阵键盘的设计相关，欢迎大伙拍砖

2. 矩阵键盘硬件连接

图 1-116

具体使用了以下几个管脚：

```
gpio0[15] Row1
gpio3[21] Row2
gpio0[14] Row3
gpio3[19] Row4
gpio3[17] Col1
gpio3[15] Col2
gpio3[16] Col3
gpio3[14] Col4
```

3. 矩阵键盘设计点滴

（1）注意我这个 4x4 的矩阵键盘没有外接电阻哦，所以作为输入的引脚做好拉地，配置成 AM33XX_PIN_INPUT_PULLDOWN

（2）注意 Row 和 Col 的引脚输入输出关系不要定义反了，Row 为 input，Col 为 output。我开始的时候定义反了搞死都出不来结果。

4. 代码相关

4.1 Pinmux

看看 4x4 键盘涉及到的几个管脚的定义吧，如下所示：

```
/* Matrix GPIO Keypad Support for profile-0 only: TODO */

/* pinmux for keypad device */
static struct pinmux_config matrix_keypad_pin_mux[] = {
{"uart1_txd.gpio0_15", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLDOWN},
{"mcasp0_ahclkx.gpio3_21", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLDOWN},
{"uart1_rxd.gpio0_14", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLDOWN},
{"mcasp0_fsr.gpio3_19", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLDOWN},

{"mcasp0_ahclkr.gpio3_17", OMAP_MUX_MODE7 | AM33XX_PIN_OUTPUT},
{"mcasp0_fsx.gpio3_15", OMAP_MUX_MODE7 | AM33XX_PIN_OUTPUT},
{"mcasp0_axr0.gpio3_16", OMAP_MUX_MODE7 | AM33XX_PIN_OUTPUT},
{"mcasp0_aclkx.gpio3_14", OMAP_MUX_MODE7 | AM33XX_PIN_OUTPUT},

{NULL, 0},
};

/* Keys mapping */
static const uint32_t am335x_evm_matrix_keys[] = {
KEY(0, 0, KEY_A),
KEY(1, 0, KEY_3),
KEY(2, 0, KEY_2),
KEY(3, 0, KEY_1),

KEY(0, 1, KEY_B),
KEY(1, 1, KEY_6),
KEY(2, 1, KEY_5),
KEY(3, 1, KEY_4),

KEY(0, 2, KEY_C),
KEY(1, 2, KEY_9),
KEY(2, 2, KEY_8),
KEY(3, 2, KEY_7),

KEY(0, 3, KEY_D),
KEY(1, 3, KEY_OK),
KEY(2, 3, KEY_0),
KEY(3, 3, KEY_DELETE),
};
const struct matrix_keymap_data am335x_evm_keymap_data = {
.keymap = am335x_evm_matrix_keys,
.keymap_size = ARRAY_SIZE(am335x_evm_matrix_keys),
};

static const unsigned int am335x_evm_keypad_row_gpios[] = {
GPIO_TO_PIN(0, 15), GPIO_TO_PIN(3, 21), GPIO_TO_PIN(0, 14), GPIO_TO_PIN(3, 19)
};
```

```
static const unsigned int am335x_evm_keypad_col_gpios[] = {
GPIO_TO_PIN(3, 17), GPIO_TO_PIN(3, 15), GPIO_TO_PIN(3, 16), GPIO_TO_PIN(3, 14)
};

static struct matrix_keypad_platform_data am335x_evm_keypad_platform_data = {
.keymap_data = &am335x_evm_keymap_data,
.row_gpios = am335x_evm_keypad_row_gpios,
.num_row_gpios = ARRAY_SIZE(am335x_evm_keypad_row_gpios),
.col_gpios = am335x_evm_keypad_col_gpios,
.num_col_gpios = ARRAY_SIZE(am335x_evm_keypad_col_gpios),
.active_low = false,
.debounce_ms = 5,
.col_scan_delay_us = 2,
};

static struct platform_device am335x_evm_keyboard = {
.name = "matrix-keypad",
.id = -1,
.dev = {
.platform_data = &am335x_evm_keypad_platform_data,
},
};
static void matrix_keypad_init(int evm_id, int profile)
{
int err;

setup_pin_mux(matrix_keypad_pin_mux);
err = platform_device_register(&am335x_evm_keyboard);
if (err) {
pr_err("failed to register matrix keypad (4x4) device\n");
}
}
```

4.2 初始化

```
/* BeagleBone Rev A3 and after */
static struct evm_dev_cfg BeagleBone_dev_cfg[] = {
{tps65217_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{mii1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{usb0_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{usb1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{mmc0_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{i2c1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{i2c2_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{lcdc_init, DEV_ON_BASEBOARD, (PROFILE_0 | PROFILE_1 |
PROFILE_2 | PROFILE_7) },
{evm_nand_init, DEV_ON_BASEBOARD,
(PROFILE_ALL & ~PROFILE_2 & ~PROFILE_3)},
{matrix_keypad_init, DEV_ON_BASEBOARD, PROFILE_0},
{NULL, 0, 0},
};
```

看看 include\linux\Input.h 关于键值的定义吧

```
#define KEY_1 2
#define KEY_2 3
#define KEY_3 4
#define KEY_4 5
#define KEY_5 6
#define KEY_6 7
#define KEY_7 8
#define KEY_8 9
#define KEY_9 10
#define KEY_0 11
```

4.3 测试程序

从网络上找个一个测试程序，能用!

```
#include
#include
#include
#include
#include
#include
int fd;
int loop_flag = 1;
void exit_loop()
{
loop_flag--;
}
void deal_int(int signum)
{
if(fd > 0)
close(fd);
exit_loop();
}
int main()
{
struct input_event evt;
fd = open("/dev/input/event0",O_RDWR);
if(fd < 0)
printf("Can not open\n");
while(loop_flag > 0)
{
read(fd,&evt,sizeof(struct input_event));
switch(evt.type)
{
case EV_KEY:
printf("Key -->\nCode: %d\nValue: %d\n",evt.code,evt.value);
break;
case EV_MSC:
printf("MSC -->\nCode: %d\nValue: %d\n",evt.code,evt.value);
```

```
break;
case EV_SYN:
printf("<-- \n");
default:
printf("...\n");
}
}
return 0;
}
```

4.4 测试结果

```
MSC -->
Code: 4
Value: 12
Key -->
Code: 2 // 1
Value: 1
<--
...
MSC -->
Code: 4
Value: 12
Key -->
Code: 2
Value: 0
<--
...
MSC -->
Code: 4
Value: 9
Key -->
Code: 6 // 5
Value: 1
<--
...
MSC -->
Code: 4
Value: 9
Key -->
Code: 6
Value: 0
<--
...
MSC -->
Code: 4
Value: 6
Key -->
Code: 10 // 9
Value: 1
```

```
<--
...
MSC -->
Code: 4
Value: 6
Key -->
Code: 10
Value: 0
<--
...
```

点击查看详情：>> EE_BeagleBone_Cape 4x4 矩阵键盘

1.3.9 EE_BeagleBone_Cape 之 NAND

1. 概述

这篇文档是关于 EE_BeagleBone_Cape nand 部分

设计的一些心得和体会，和大家分型一下。欢迎大伙拍砖。

2. NAND 设计点滴

还是把原理图图贴上来吧，呵呵

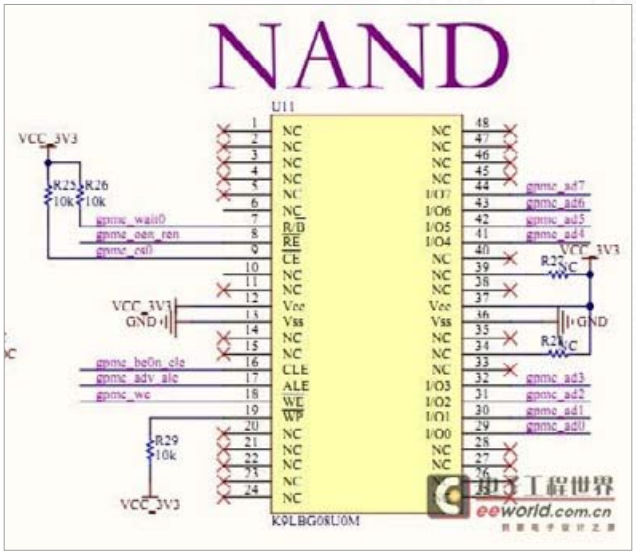


图 1-117

1) 关于管脚电压，主要是 34，39 脚。这里做了一点兼容性设计，MT29F2G08 这两个引脚需要接 3.3V，而我使用的 K9LBG08U0M 和 K9F2G08 是不需要接的。

2) 由于需要通过 FPGA 连接，FPGA 的代码是必不可少的。

3) 关于 NAND 和 NOR 的区别网络上很多，可以看看以下链接：

关于 NAND 和 NOR 的区别很容易上网查阅到，通过下面的这张图也可以清楚的看到二者的区别。

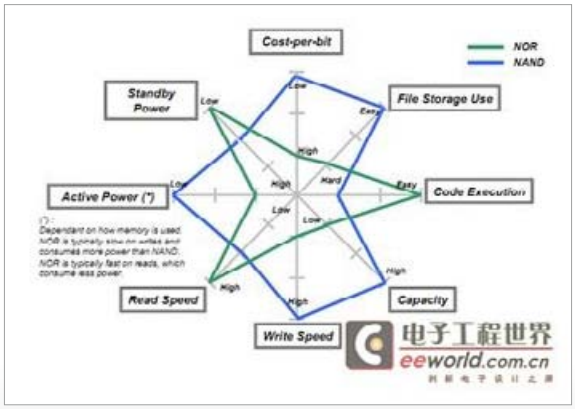


图 1-118

4) 在选择 NAND 的容量的时候，还需要先了解下 uboot 和 linux 的支持情况，要不让是比较苦逼的。比如 U-boot 2011.09 是不支持 K9LBG08U0M 这样 4GB 的 Nand 的 flash 的，让我走了一些弯路，后面有介绍。

5) 能够识别了也不是就完事了，分区的格式化也是个问题，我在使用 3.1 内核的时候，搞死都不行；换成 3.2 内核就一切 ok 啦。非常的诡异，具体啥情况我还没细细研究，有兴趣的可以研究下哦：）

3. FPGA 代码

1) 首先呢，是做好管脚分配，主要是把 BeagleBone 的控制引脚和 NAND 的引脚一一对应上 . 关于管脚分配前面已经写过了，就不重复啦

```
// for NAND
assign gpmc_we = TIMER6 ;
assign gpmc_adv_ale = TIMER4 ;
assign gpmc_be0n_cle = TIMER5;
assign gpmc_cs0 = GPIO1_29 ;
assign gpmc_oen_ren = TIMER7 ;
assign UART4_RXD = gpmc_wait0 ;
assign gpmc_ad = ((GPIO1_29== 1'b0)
&& (TIMER6 == 1'b0)) ?
{GPIO1_7,GPIO1_6,GPIO1_5,GPIO1_4,GPIO1_3,GPIO1_2,GPIO1_1,GPIO1_0} : 16'bz;

assign
{GPIO1_7,GPIO1_6,GPIO1_5,GPIO1_4,GPIO1_3,GPIO1_2,GPIO1_1,GPIO1_0} = ((GPIO1_29==
1'b0) && (TIMER7 == 1'b0)) ? gpmc_ad : 16'bz;
```

4. U-Boot 中的 NAND 支持

1) 首先看看 drivers/mtd/nand/ 目录下的 Nand_ids.c 文件吧，这个文件列出了 U-Boot 支持的 NAND 列表，在 U-boot 2011.09 中，没有找到支持 4GB 的 NAND 类型，悲剧! 怪不得 U-Boot 启动的时候没打印信息，我还以为是焊接问题，焊了又焊，深刻的教训!

```
/* 8 Gigabit */
{"NAND 1GiB 1,8V 8-bit", 0xA3, 0, 1024, 0, LP_OPTIONS},
{"NAND 1GiB 3,3V 8-bit", 0xD3, 0, 1024, 0, LP_OPTIONS},
{"NAND 1GiB 1,8V 16-bit", 0xB3, 0, 1024, 0, LP_OPTIONS16},
{"NAND 1GiB 3,3V 16-bit", 0xC3, 0, 1024, 0, LP_OPTIONS16},

/* 16 Gigabit */
{"NAND 2GiB 1,8V 8-bit", 0xA5, 0, 2048, 0, LP_OPTIONS},
{"NAND 2GiB 3,3V 8-bit", 0xD5, 0, 2048, 0, LP_OPTIONS},
{"NAND 2GiB 1,8V 16-bit", 0xB5, 0, 2048, 0, LP_OPTIONS16},
{"NAND 2GiB 3,3V 16-bit", 0xC5, 0, 2048, 0, LP_OPTIONS16},
```

下面没有了，呵呵

2) 在来看看 include/configs 下的 Am335x_evm.h 文件吧第 46 行，说明已经支持

NAND 了，要不然也不会有打印呵呵

```
#define CONFIG_MMC
#define CONFIG_NAND
#define CONFIG_SPI
```

再看看第 210 行，这是和 NAND 配置相关的一些代码，还有 SPL 相关的配置

```
/* NAND boot config */
#define CONFIG_SPL_NAND_SIMPLE
#define CONFIG_SPL_NAND_SUPPORT
#define CONFIG_SYS_NAND_5_ADDR_CYCLE
#define CONFIG_SYS_NAND_PAGE_COUNT (CONFIG_SYS_NAND_BLOCK_SIZE / \
CONFIG_SYS_NAND_PAGE_SIZE)
#define CONFIG_SYS_NAND_PAGE_SIZE 2048
#define CONFIG_SYS_NAND_OOBSIZE 64
#define CONFIG_SYS_NAND_BLOCK_SIZE (128*1024)
#define CONFIG_SYS_NAND_BAD_BLOCK_POS NAND_LARGE_BADBLOCK_POS
#define CONFIG_SYS_NAND_ECCPOS { 2, 3, 4, 5, 6, 7, 8, 9, \
10, 11, 12, 13, 14, 15, 16, 17, \
18, 19, 20, 21, 22, 23, 24, 25, \
26, 27, 28, 29, 30, 31, 32, 33, \
34, 35, 36, 37, 38, 39, 40, 41, \
42, 43, 44, 45, 46, 47, 48, 49, \
50, 51, 52, 53, 54, 55, 56, 57, }

#define CONFIG_SYS_NAND_ECCSIZE 512
#define CONFIG_SYS_NAND_ECCBYTES 14

#define CONFIG_SYS_NAND_ECCSTEPS 4
#define CONFIG_SYS_NAND_ECCTOTAL (CONFIG_SYS_NAND_ECCBYTES * \
CONFIG_SYS_NAND_ECCSTEPS)
#define CONFIG_SYS_NAND_U_BOOT_START CONFIG_SYS_TEXT_BASE
#define CONFIG_SYS_NAND_U_BOOT_OFFS 0x80000
```

第 19 行，说明可以在 nand 中存放环境变量

```
#define CONFIG_SYS_NO_FLASH
#define CONFIG_NAND_ENV
```

3) 看看我换成 K9F2G08 的打印信息吧

```
U-Boot 2011.09 (Nov 04 2011 - 21:39:19)

I2C: ready
DRAM: 256 MiB
WARNING: Caches not enabled
No daughter card present
```

```
NAND: HW ECC Hamming Code selected
256 MiB
MMC: OMAP SD/MMC: 0
*** Warning - bad CRC, using default environment
```

5. Linux 中的 NAND 支持

1) 同样的，我们先看看 drivers/mtd/nand/ 目录下的 Nand_ids.c 文件，有心的人可以看出 U-Boot 的目录结构和文件和 linux 的一样的。但是支持的 NAND 的类型就大很多了哦，最大可以到 64GB。

```
/* 16 Gigabit */
{"NAND 2GiB 1,8V 8-bit", 0xA5, 0, 2048, 0, LP_OPTIONS},
{"NAND 2GiB 3,3V 8-bit", 0xD5, 0, 2048, 0, LP_OPTIONS},
{"NAND 2GiB 1,8V 16-bit", 0xB5, 0, 2048, 0, LP_OPTIONS16},
{"NAND 2GiB 3,3V 16-bit", 0xC5, 0, 2048, 0, LP_OPTIONS16},
........
........
/* 512 Gigabit */
{"NAND 64GiB 1,8V 8-bit", 0x1E, 0, 65536, 0, LP_OPTIONS},
{"NAND 64GiB 3,3V 8-bit", 0x3E, 0, 65536, 0, LP_OPTIONS},
{"NAND 64GiB 1,8V 16-bit", 0x2E, 0, 65536, 0, LP_OPTIONS16},
{"NAND 64GiB 3,3V 16-bit", 0x4E, 0, 65536, 0, LP_OPTIONS16},
```

看看 \arch\arm\mach-omap2\Board-am335xevm.c 中关于 NAND 支持吧

如果要让 BeagleBone 支持 nand，需要在 BeagleBone_dev_cfg 中增加 nand 的初始化函数，我是这样做的

```
/* BeagleBone Rev A3 and after */
static struct evm_dev_cfg BeagleBone_dev_cfg[] = {
{mii1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{usb0_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{usb1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{mmc0_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{evm_nand_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{NULL, 0, 0},
};
```

看看 NAND 初始化部分都干了些啥子，主要设计 nand_pin_mux 和 am335x_nand_partition 这两个

```
static void evm_nand_init(int evm_id, int profile)
{
setup_pin_mux(nand_pin_mux);
board_nand_init(am335x_nand_partitions,
ARRAY_SIZE(am335x_nand_partitions), 0, 0);
}
```

下面这个就是管脚的定义了

```
/* Pin mux for nand flash module */
static struct pinmux_config nand_pin_mux[] = {
{"gpmc_ad0.gpmc_ad0",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad1.gpmc_ad1",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad2.gpmc_ad2",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad3.gpmc_ad3",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad4.gpmc_ad4",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad5.gpmc_ad5",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad6.gpmc_ad6",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad7.gpmc_ad7",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_wait0.gpmc_wait0", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_wpn.gpmc_wpn",   OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_csn0.gpmc_csn0",   OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
{"gpmc_advn_ale.gpmc_advn_ale", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
{"gpmc_oen_ren.gpmc_oen_ren",   OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
{"gpmc_wen.gpmc_wen", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
{"gpmc_ben0_cle.gpmc_ben0_cle",   OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
{NULL, 0},
};
```

这是 NAND 的分区，可以看到和系统启动的时候打印是一致的

```
/* NAND partition information */
static struct mtd_partition am335x_nand_partitions[] = {
/* All the partition sizes are listed in terms of NAND block size */
{
.name = "SPL",
.offset = 0, /* Offset = 0x0 */
.size = SZ_128K,
},
{
.name = "SPL.backup1",
.offset = MTDPART_OFS_APPEND, /* Offset = 0x20000 */
.size = SZ_128K,
},
{
.name = "SPL.backup2",
.offset = MTDPART_OFS_APPEND, /* Offset = 0x40000 */
.size = SZ_128K,
},
{
.name = "SPL.backup3",
.offset = MTDPART_OFS_APPEND, /* Offset = 0x60000 */
.size = SZ_128K,
},
{
.name = "U-Boot",
.offset = MTDPART_OFS_APPEND, /* Offset = 0x80000 */
```

```
.size = 15 * SZ_128K,
},
{
.name = "U-Boot Env",
.offset = MTDPART_OFS_APPEND, /* Offset = 0x260000 */
.size = 1 * SZ_128K,
},
{
.name = "Kernel",
.offset = MTDPART_OFS_APPEND, /* Offset = 0x280000 */
.size = 40 * SZ_128K,
},
{
.name = "File System",
.offset = MTDPART_OFS_APPEND, /* Offset = 0x780000 */
.size = MTDPART_SIZ_FULL,
},
};
```

2) 这也为什么我的4GB NAND也是可以被linux检测到的原因

看看 4GB 的系统的打印信息吧

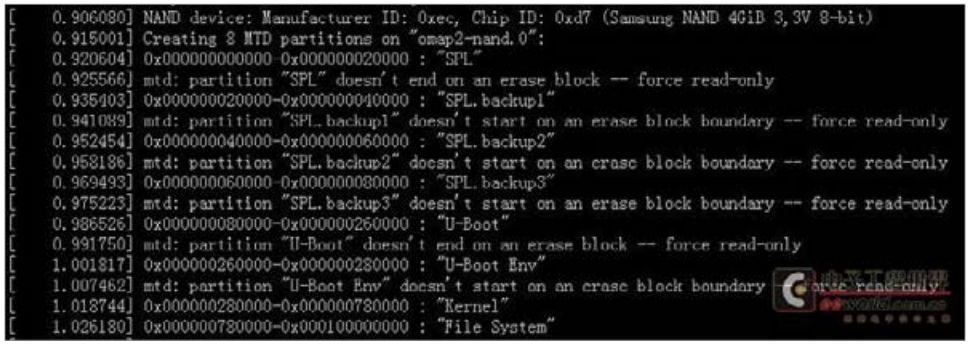


图 1-119

再看看 256MB 的系统的打印信息吧

```
[ 0.885371] omap_hsmmc.0: alias fck already exists
[ 0.894289] mtdoops: mtd device (mtddev=name/number) must be supplied
[ 0.901336] omap2-nand driver initializing
[ 0.906150] NAND device: Manufacturer ID: 0xec, Chip ID: 0xda (Samsung NAND 256MiB
3,3V 8-bit)
[ 0.915251] Creating 8 MTD partitions on "omap2-nand.0":
[ 0.920854] 0x000000000000-0x000000020000 : "SPL"
[ 0.927785] 0x000000020000-0x000000040000 : "SPL.backup1"
[ 0.935221] 0x000000040000-0x000000060000 : "SPL.backup2"
[ 0.942519] 0x000000060000-0x000000080000 : "SPL.backup3"
[ 0.950013] 0x000000080000-0x000000260000 : "U-Boot"
[ 0.957714] 0x000000260000-0x000000280000 : "U-Boot Env"
[ 0.965087] 0x000000280000-0x000000780000 : "Kernel"
[ 0.974097] 0x000000780000-0x000010000000 : "File System"
```

6. NAND Flash 相关操作

U-Boot 下的相关命令：具体的就不解释了，有兴趣的可以从网上查看

```
U-Boot# nand
nand - NAND sub-system
Usage:
nand info - show available NAND devices
nand device [dev] - show or set current device
nand read - addr off|partition size
nand write - addr off|partition size
read/write 'size' bytes starting at offset 'off'
to/from memory address 'addr', skipping bad blocks.
nand erase[.spread] [clean] off size - erase 'size' bytes from offset 'off'
With '.spread', erase enough for given file size, otherwise,
'size' includes skipped bad blocks.
nand erase.part [clean] partition - erase entire mtd partition'
nand erase.chip [clean] - erase entire chip'
nand bad - show bad blocks
nand dump[.oob] off - dump page
nand scrub off size | scrub.part partition | scrub.chip
really clean NAND erasing bad blocks (UNSAFE)
nand markbad off [...] - mark bad block(s) at offset (UNSAFE)
nand biterr off - make a bit error at offset (UNSAFE)
```

查看一下信息吧

```
U-Boot# nand info
Device 0: nand0, sector size 128 KiB
U-Boot# nand device
Device 0: nand0, sector size 128 KiB
```

Linux下的相关操作

看看有几个分区

```
root@am335x-evm:~# cat /proc/mtd
dev: size erasesize name
mtd0: 00020000 00020000 "SPL"
mtd1: 00020000 00020000 "SPL.backup1"
mtd2: 00020000 00020000 "SPL.backup2"
mtd3: 00020000 00020000 "SPL.backup3"
mtd4: 001e0000 00020000 "U-Boot"
mtd5: 00020000 00020000 "U-Boot Env"
mtd6: 00500000 00020000 "Kernel"
mtd7: 0f880000 00020000 "File System"

root@am335x-evm:~# cat /proc/partitions
major minor #blocks name
31  0  128 mtdblock0
```

```
31 1 128 mtdblock1
31 2 128 mtdblock2
31 3 128 mtdblock3
31 4 1920 mtdblock4
31 5 128 mtdblock5
31 6 5120 mtdblock6
31 7 254464 mtdblock7
179 0 3813376 mmcblk0
179 1 72261 mmcblk0p1
179 2 923737 mmcblk0p2
179 3 2795310 mmcblk0p3
```

使用 flash_eraseall 是有些问题，暂时问题不明，如果哪位大侠知道，麻烦告诉我一下，谢谢!

```
root@am335x-evm:~# flash_eraseall -j /dev/mtd7
flash_eraseall has been replaced by `flash_erase 0 0`; please use it
flash_erase: error!: /dev/mtd7: unable to get NAND oobinfo
 error 22 (Invalid argument)
```

使用 mkfs 就没有问题

```
root@am335x-evm:~# mkfs -t ext2 /dev/mtd7
mke2fs 1.41.14 (22-Dec-2010)
/dev/mtd7 is not a block special device.
Proceed anyway? (y,n)
root@am335x-evm:~# mkfs -t ext2 /dev/mtdblock7
mke2fs 1.41.14 (22-Dec-2010)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
63744 inodes, 254464 blocks
12723 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
32 block groups
8192 blocks per group, 8192 fragments per group
1992 inodes per group
Superblock backups stored on blocks:
8193, 24577, 40961, 57345, 73729, 204801, 221185

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 32 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

挂载到 /mnt/nand，创建一个 hello_eeworld 目录，然后重启

```
root@am335x-evm:~# cd /mnt
root@am335x-evm:/mnt# ls
card cf nand net ram
root@am335x-evm:/mnt# ls nand
root@am335x-evm:/mnt# mount /dev/mtdblock7 nand/
root@am335x-evm:/mnt# ls nand/
lost+found
root@am335x-evm:/mnt# mkdir nand/hello_eeworld
root@am335x-evm:/mnt# ls nand/
hello_eeworld lost+found
root@am335x-evm:/mnt# reboot
```

重启后，hello_eeworld 还在，说明是能够操作成功的。

```
am335x-evm login: root
root@am335x-evm:~# mount /dev/mtdblock7 /mnt/nand
root@am335x-evm:~# ls /mnt/nand
hello_eeworld lost+found
root@am335x-evm:~#
```

点击查看详情：>> EE_BeagleBone_Cape 之 NAND

1.3.10 BeagleBone Black 利用 Ubuntu 上网

参考了很多人的方案，不过最后还是靠自己完成了!

曲折，有时觉得自己特慢热，学东西特慢！ 不过最后的最后，我成功地完成了!

好了，言归正传。

BeagleBone Black 本身就是一个 DHCP 服务器，网络地址为 192.168.7.0/30，整个网络只有 192.168.7.0、192.168.7.1、192.168.7.2、192.168.7.3。其中，第一个和第四个不能用，分别是网络号和广播的地址，剩下的 192.168.7.2 是 BeagleBone Black 自己的地址（熟悉 BBB 的朋友都应该很熟悉这个地址，这个地址就是你第一次在浏览器里键入，用以打开 BBB 的 IDE 地址）。

所以，只有一个 192.168.7.1 是分配给我的电脑的。（当然了，这些都是可以更改的）我的主机是 Ubuntu 12.04 LTS，有两张有线网卡 eth0、eth1。平时我上网是都是用路由器登陆学校里的 VPN，然后在通过路由器上网。但是只要我插上 BBB，我就不能上网了，也无法操作 BBB。很头疼。也正是为了解决这个问题，我才开始折腾的。打开 Network Manager，发现两个网卡的地址竟然是同一个!!!不知道为什么。于是我就打算不用这个软件，直接设置文件系统里的相关文件来上网，顺便让我的 BBB 也可以享一下福。首先，我们要做的就是为两张网卡分别设置静态地址：

在终端中键入：

```
sudo vim /etc/network/interfaces
```

在打开的文件中输入：

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.8.2
gateway 192.168.8.1
netmask 255.255.255.0
auto eth1
iface eth1 inet static
address 192.168.7.1
netmask 255.255.255.252
broadcast 192.168.7.3
```

其中，192.168.8.0/24 是我的路由器的 LAN 口网络，192.168.8.1 是直接和我的电脑相连的那个 interface 的网络地址，192.168.8.2 是我的 eth0 的网络地址。192.168.7.0/30 是我的 BBB 所在的网络，其他的几个地址之前都已经提到过了。当然了，既然已经放弃了 Network Manager，就需要配置 DNS 服务：

在终端中键入：

```
sudo vim /etc/resolv.conf
```

然后添加我最近的一个 DNS 服务器：

```
nameserver 10.10.0.21
```

不过，为了防止重启之后，DNS 的信息被重置，还需要新建一个文件：

```
vim /etc/resolvconf/resolv.conf.d/tail
```

文件里添加

```
nameserver 10.10.0.21
```

这样，Ubuntu 应该就可以同时连接 BBB 和 Internet 了。

为了能够让 BBB 通过 Ubuntu 上网，还需要做以下设置：设置

Ubuntu 的 ipv4 包转发规则设置 BBB 的网关为 Ubuntu，并设置 DNS 首先设置 Ubuntu 的 ipv4 包转发规则：

打开 /etc/sysctl.conf，找到 net.ipv4.ip_forward=1，去掉前面的注释；

键入一下命令是改动生效：

```
sudo sysctl -p
```

通过 iptables 实现包转发：

```
sudo iptables -A POSTROUTING -t nat -j MASQUERADE
```

可能还需要以下命令：

```
echo 1>/proc/sys/net/ipv4/ip_forward
```

这样，Ubuntu 的部分就设置好了。接着我们设置 BBB，比较简单，键入：

vim .profile

在打开的文件末尾添加：

```
/usr/sbin/route add default gw 192.168.7.1
echo "nameserver 10.10.0.21" >> /etc/resolv.conf
```

1.3.11 基于 BeagleBone 和 DS18B20 的温度测量

1. 硬件连接

BeagleBone 的 P8 第 6 脚 GPIO1_3 接 DS18B20 的数据脚，连接关系如下图所示：

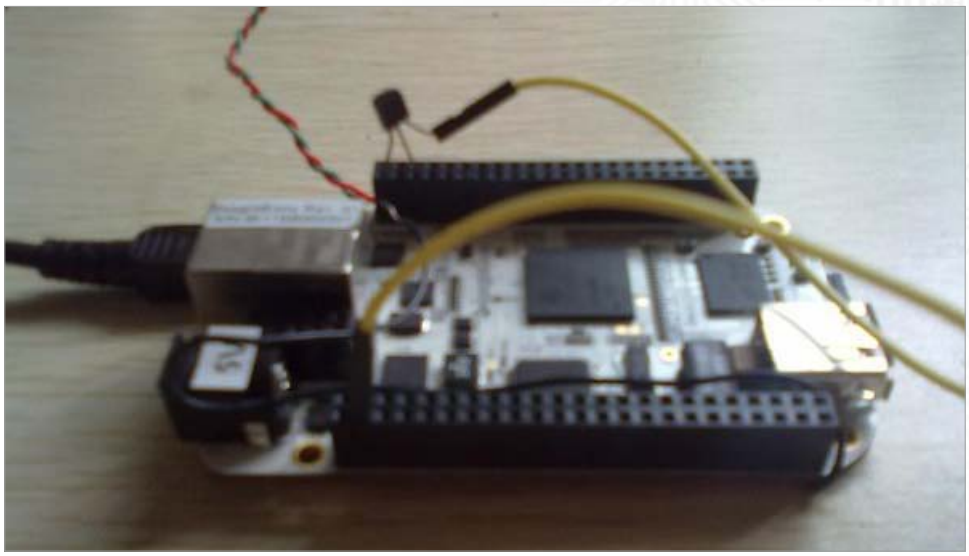


图 1-120

2. 代码修改

主要修改 board-am335xevm.c

1) 增加头文件

#include <linux/w1-gpio.h>

2) 增加管脚定义和 w1 设备

```
#define BEAGLEBONE_W1_GPIO GPIO_TO_PIN(1, 3)  //P8_6

static struct w1_gpio_platform_data bone_w1_gpio_pdata = {
 .pin = BEAGLEBONE_W1_GPIO,
 .is_open_drain = 0,
};

static struct platform_device bone_w1_device = {
 .name = "w1-gpio",
 .id = -1,
 .dev.platform_data = &bone_w1_gpio_pdata,
};
```

3) 增加初始化函数 bonew1_gpio_init

```
static void bonew1_gpio_init(int evm_id, int profile )
{
 int err;
```

```
 err = platform_device_register(&bone_w1_device);
 if (err)
  pr_err("failed to register w1-gpio\n");
 else
  pr_info("w1-gpio connected to P8_6\n");
}
```

4) 添加到 BeagleBone 的配置函数

```
/* BeagleBone Rev A3 and after */
static struct evm_dev_cfg BeagleBone_dev_cfg[] = {
{mii1_init, DEV_ON_BASEBOARD, PROFILE_NONE},
{bonew1_gpio_init, DEV_ON_BASEBOARD, PROFILE_ALL},
```

3. 测试结果：

查看启动信息：

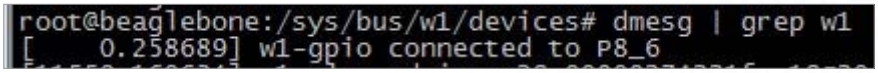


图 1-121

查看 DS18B20 的温度值

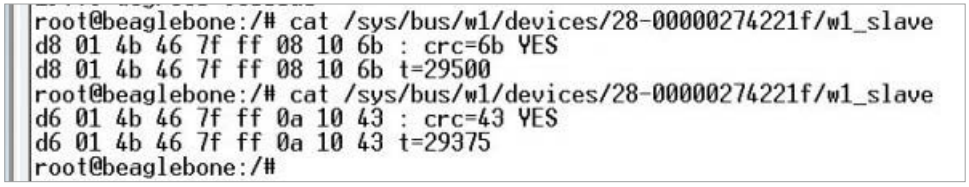


图 1-122

转换成摄氏度以后的值

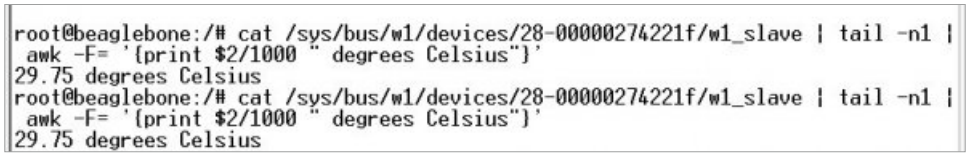


图 1-123

点击查看详情：[>> 基于 BeagleBone 和 DS18B20 的温度测量]

1.3.12 BeagbleBone 之网页控制 LED 灯

A、基于 node.js 的 LED 流水灯设计

1、前提：

已建立 PC 与 beagblebone 之间的网络连接

2、效果描述：

采用 BeagleBone 的 node.js 开发环境，即在浏览器输入 http://IP_ADDR_OF_BB:3000，参照自带的 blinkled 程序，做一个将 beagblebone 的 4 个 USR-LED 流水灯效果的程序

3、代码

如下，注释已经很明显

4、效果

实现了4个 LED 逐个依次点亮的效果。具体效果做过单片机的 LED 流水灯的都懂得。

// 本程序从采用 node.js 实现 beagblebone 的四个 LED 流水灯效果

```
// get the BeagleBone variables and functions
var bb = require('./bonescript');
// the delay duration between two LEDs, in microseconds
var delayMs = 200;
// the whole number of LEDS
var NUM_LEDS = 4;
// 4 LEDs
var ledPin0 = bone.USR0;
var ledPin1 = bone.USR1;
var ledPin2 = bone.USR2;
var ledPin3 = bone.USR3;
// 4 LED status
var led0 = LOW;
var led1 = LOW;
var led2 = LOW;
var led3 = LOW;
// the counter used for couting for the current lighting LED
var counter = 0;
// setup function set the pin mode of the 4 LEDS
setup = function() {

    pinMode(ledPin0, OUTPUT);
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2, OUTPUT);
    pinMode(ledPin3, OUTPUT);
};
/* the loop function increase the counter, and divided by 4
```

```
* using the remain to judge the number of the LED to be lighted

*/

loop = function() {
    // Firstly unlighting all LEDS
    led0 = LOW;
    led1 = LOW;
    led2 = LOW;
    led3 = LOW;
    // increase the counter and divided by 4, to
    counter ++;
    var remain = counter % NUM_LEDS;
    // using the remain to judge the LED to be lighted, set the respond led
status to be HIGH

    if(0 == remain) {
        led0 = HIGH;
    }

    else if(1 == remain) {
        led1 = HIGH;
    }

    else if(2 == remain) {
        led2 = HIGH;
    }

    else if(3 == remain) {
        led3 = HIGH;    }

    // output the status to the LED
    digitalWrite(ledPin0, led0);
    digitalWrite(ledPin1, led1);
    digitalWrite(ledPin2, led2);
    digitalWrite(ledPin3, led3);
    // delay for the required duration
    delay(delayMs);

};
bb.run();
```

B、基于 node.js 的网页控制 LED 设计

1、前提：

已建立 PC 与 beagblebone 之间的网络连接

2、效果描述：

采用 BeagleBone 的 node.js 开发环境，即在浏览器输入 http://IP_ADDR_OF_

BB:3000，参照自带的 blinkled 程序，做一个将 beagblebone 的 USR0-LED 通过网页刷新来控制亮和灭，

3、代码

如下，注释已经很明显

4、效果

即在另一个浏览器窗口输入 http://IP_ADDR_OF_BB:8000，刷新一次，则 LED-USR0 则改变一次亮灭。

5、亮点和进一步创新点

在网页上实现 LED 的控制。

进一步可采取 node.js 的一些功能比如（nowjs 的 server 与 client 的通讯功能，在网页上动态控制 LED、显示模拟输入或数字输入量。

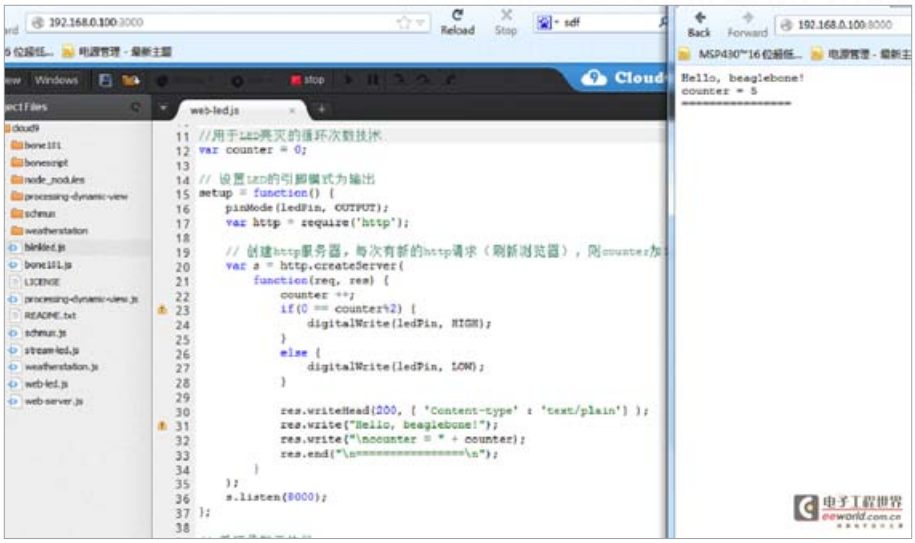


图 1-124

```
// 打开浏览器；输入http://IP_Addr_if_BeagleBone:8000，刷新，则LED-USR0相应亮灭
// 采用下述代码取代引用到bonescript的一些常量和函数
var bb = require('./bonescript');
// 选取LED0作为显示
var ledPin = bone.USR0;
//用于LED亮灭的循环次数技术
var counter = 0;
// 设置LED的引脚模式为输出
setup = function() {
    pinMode(ledPin, OUTPUT);
    var http = require('http');

    // 创建http服务器，每次有新的http请求（刷新浏览器），则counter加1，除以2的余数作为
LED的状态
```

```
    var s = http.createServer(
        function(req, res) {
            counter ++;
            if(0 == counter%2) {
                digitalWrite(ledPin, HIGH);
            }
            else {
                digitalWrite(ledPin, LOW);
            }
  res.writeHead(200, { 'Content-type' : 'text/plain'} );
  res.write("Hello, BeagleBone!");
  res.write("\ncounter = " + counter);
  res.end("\n================\n");
                }
    );
    s.listen(8000);
};
// 循环函数无执行
loop = function() {
     // Your function to run in a loop
};
// 启动bb
bb.run();
```

点击查看详情：>>beagblebone 之 node.js 的 LED 流水灯设计

1.3.13 基于 BeagleBone 的数据采集界面的实现

一、根据我需要的功能，我利用 Stellaris Graphics Library 实现了数据采集界面!（发在深夜，所以图片和视频的光线较为虚幻!）

图片展示：

图 1-125

主界面：这里的按钮的动作都是带声音的! 可以在下面的视频中听到! 通过 Codec 的功能实现!

图 1-126

说明界面：这里使用的是我自己做的包含中文和英文的字库! 我加了大概 2000 多个常用汉字和符号! 具体可参考我的另一篇 M4 的帖子：http://bbs.eeworld.com.cn/thread-333915-1-1.html

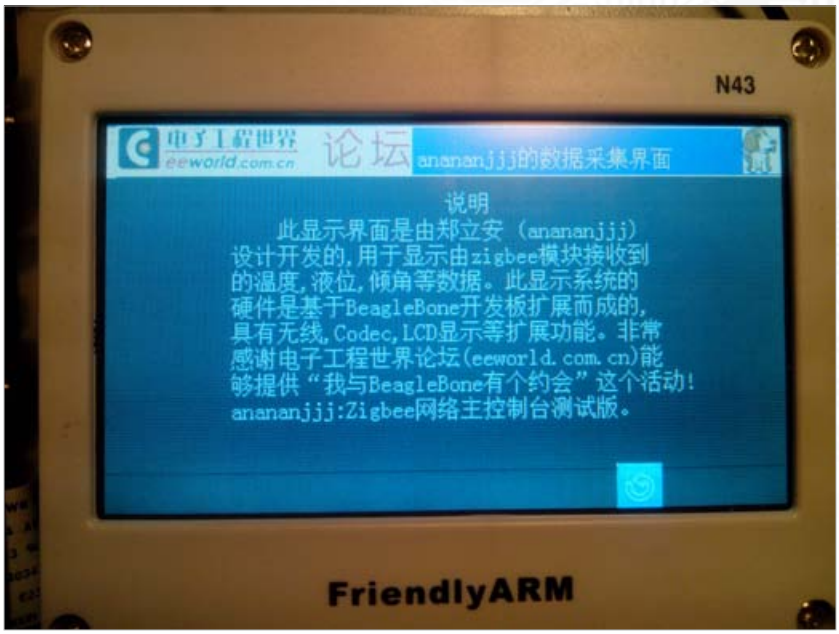


图 1-127

加速度传感器测角度界面：表盘的绘制使用了 sin，cos 函数，所以要使用浮点功能，使能浮点功能的方法可以看我的帖子：http://bbs.eeworld.com.cn/thread-356085-1-1.html

图 1-128

单轴的界面：

图 1-129

图 1-130

图 1-131

液位界面：

图 1-132

二、视频演示：背景的嘟嘟声是 Codec 按键音的效果!

三、程序说明：

1、首先我是基于官方的 grlib_demo 程序实现的此界面!

我在主界面上添加了电子工程世界和板子的 Logo !
图片是利用 GIMP 2 软件编辑的，并将其转换成 .pnm 格式

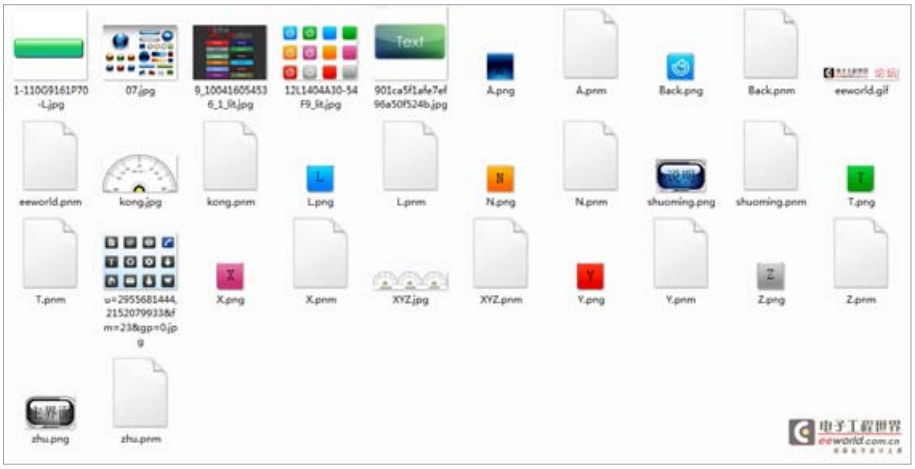


图 1-133

这些便是我的图像素材! 然后利用 \tools\pnmtoc 下的 pnmtoc.exe 逐一转换成图像数组：

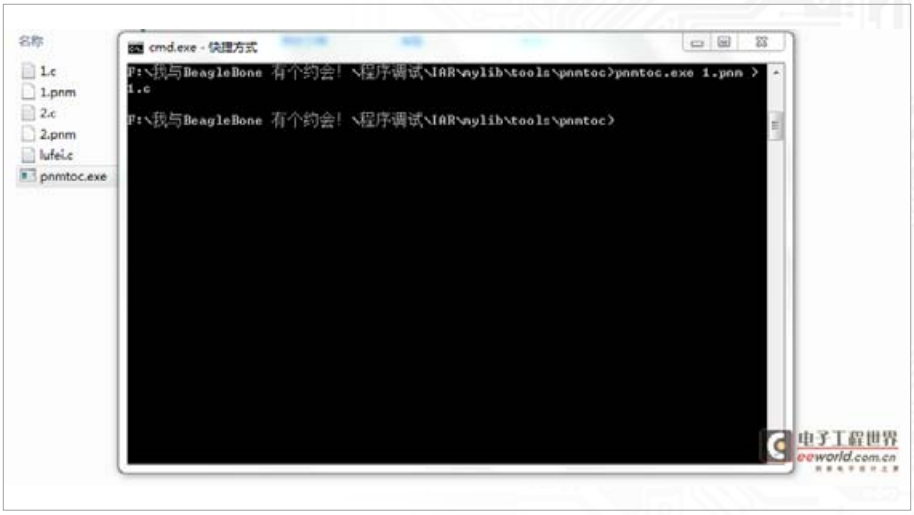


图 1-134

然后就是调用函数进行显示了：

主函数中自然是对 LCD，Codec 等外设进行了初始化配置：

```
int main(void)
{
unsigned long i = 0;
unsigned char *dest;
unsigned char *src;
int xdata;
int ydata;

MMUConfigAndEnable();

CacheEnable(CACHE_ALL);
SetupIntc();

SetUpLCD();

/* configuring the base ceiling */
RasterDMAFBConfig(SOC_LCDC_0_REGS,
(unsigned int)(g_pucBuffer+PALETTE_OFFSET),
(unsigned int)((g_pucBuffer+PALETTE_OFFSET) +
sizeof(g_pucBuffer) - 2 - PALETTE_OFFSET),
FRAME_BUFFER_0);

RasterDMAFBConfig(SOC_LCDC_0_REGS,
(unsigned int)(g_pucBuffer+PALETTE_OFFSET),
(unsigned int)((g_pucBuffer+PALETTE_OFFSET) +
sizeof(g_pucBuffer) - 2 - PALETTE_OFFSET),
FRAME_BUFFER_1);

src = (unsigned char *) palette_32b;
dest = (unsigned char *) (g_pucBuffer+PALETTE_OFFSET);
```

```
/* Copy palette info into buffer */
for( i = PALETTE_OFFSET; i < (PALETTE_SIZE+PALETTE_OFFSET); i++)
{
*dest++ = *src++;
}

GrOffScreen16BPPInit(&g_s35_480x272x24Display, g_pucBuffer, LCD_WIDTH,
LCD_HEIGHT);

/* Initialize a drawing context.*/
GrContextInit(&sContext, &g_s35_480x272x24Display);

/* enable End of frame interrupt */
RasterEndOfFrameIntEnable(SOC_LCDC_0_REGS);

/* enable raster */
RasterEnable(SOC_LCDC_0_REGS);
```

然后就是主界面的绘制：

```
tRectangle sRect;

// Fill the top 24 rows of the screen with blue to create the banner.
sRect.sXMin = 0;
sRect.sYMin = 0;
sRect.sXMax = GrContextDpyWidthGet(&sContext) - 1;
sRect.sYMax = 35;
GrContextForegroundSet(&sContext, ClrDarkBlue);
GrRectFill(&sContext, &sRect);
// Put a white box around the banner.
GrContextForegroundSet(&sContext, ClrWhite);
GrRectDraw(&sContext, &sRect);
GrImageDraw(&sContext, g_logo30x34, 445, 0);//画BeagleBone logo
GrImageDraw(&sContext, g_eeworldlogo, 0, 0); //画电子工程世界的logo
// Put the application name in the middle of the banner.
GrContextFontSet(&sContext, &g_sFontCH16);
GrStringDraw(&sContext, “anananjjj的数据采集界面”, -1,
210 , 15, 0);
//此处使用的就是我自己做的16X16的汉字加英文字库：g_sFontCH16
// Initialize the sound driver.
// Add the first panel to the widget tree.
g_ulPanel = 0;
WidgetAdd(WIDGET_ROOT, (tWidget *)g_psPanels);
CanvasTextSet(&g_sTitle, g_pcPanelNames[0]);//添加第一个界面
// Issue the initial paint request to the widgets.
WidgetMessageQueueAdd(WIDGET_ROOT, WIDGET_MSG_PAINT, 0, 0, 0, 0);
}
//Codec的嘟嘟声播放函数
void ClickPlay(void)
{
I2CCodecIfInit(SOC_I2C_2_REGS, I2C_SLAVE_CODEC_AIC31);
```

```
SoundClickPlay((unsigned char*)toneRaw, sizeof(toneRaw));
delay(0x9FFFF);
}
```

对应的按钮控件：

```
//*****************************************************************************
//
//主屏幕包含5个功能按钮和6个数据显示框
//
//*****************************************************************************
//按键1: 温度查看
RectangularButton(g_sTemperature, 0, 0, 0, &g_s35_480x272x24Display, 0,
70, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Temperature32x32,
g_Temperature32x32, 0, 0, OnTemperature);
//按键2: 液位查看
RectangularButton(g_sLiquid, 0, 0, 0, &g_s35_480x272x24Display, 0,
130, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Liquid32x32,
g_Liquid32x32, 0, 0, OnLiquid);

//按键3: 备用查看
RectangularButton(g_sNone, 0, 0, 0, &g_s35_480x272x24Display, 0,
190, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_None32x32,
g_None32x32, 0, 0, OnNone);

//按键4: 加速度查看
RectangularButton(g_sAcceleration, 0, 0, 0, &g_s35_480x272x24Display, 240,
130, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Acceleration32x32,
g_Acceleration32x32, 0, 0, OnAcceleration);

//按键5: 说明查看
RectangularButton(g_sIllustration, 0, 0, 0, &g_s35_480x272x24Display, 0,
232, 60,40, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Illustration60x40,
g_Illustration60x40, 0, 0, OnIllustration);

//按键6: X轴查看
RectangularButton(g_sXaxis, 0, 0, 0, &g_s35_480x272x24Display, 80,
240, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Xaxis32x32,
g_Xaxis32x32, 0, 0, OnXaxis);

//按键7: Y轴查看
RectangularButton(g_sYaxis, 0, 0, 0, &g_s35_480x272x24Display, 180,
240, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Yaxis32x32,
g_Yaxis32x32, 0, 0, OnYaxis);
```

```
//按键8: Y轴查看
RectangularButton(g_sZaxis, 0, 0, 0, &g_s35_480x272x24Display, 280,
240, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Zaxis32x32,
g_Zaxis32x32, 0, 0, OnZaxis);

//按键9:主界面查看
RectangularButton(g_sMain, 0, 0, 0, &g_s35_480x272x24Display, 410,
240, 60,40, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Main60x40,
g_Main60x40, 0, 0, OnMain);


//按键10:返回查看
RectangularButton(g_sBack, 0, 0, 0, &g_s35_480x272x24Display, 350,
240, 32,32, PB_STYLE_IMG | PB_STYLE_TEXT, ClrBlack,
ClrBlack, 0, ClrSilver, &g_sFontCm20, " ", g_Back32x32,
g_Back32x32, 0, 0, OnBack);

Canvas(g_sTitle, 0, 0, 0, &g_s35_480x272x24Display, 50+X_OFFSET, 30,
200, 50, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_OPAQUE , 0, 0, ClrSilver,
&g_sFontCH16, 0, 0, 0);//标题画布让其显示在画布上端
```

其相应的调用函数：

```
void OnTemperature(tWidget *pWidget)
{
if(g_ulPanel == 0)//只有在主界面的时候案件响应
{
//
// Remove the current panel.
//
WidgetRemove((tWidget *)(g_psPanels + g_ulPanel));
//
// Increment the panel index.
//
g_ulPanel=6;

//
// Add and draw the new panel.
//
WidgetAdd(WIDGET_ROOT, (tWidget *)(g_psPanels + g_ulPanel));
WidgetPaint((tWidget *)(g_psPanels + g_ulPanel));

//
// Set the title of this panel.
//
CanvasTextSet(&g_sTitle, g_pcPanelNames[g_ulPanel]);
WidgetPaint((tWidget *)&g_sTitle);
```

```
//
// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sTemperature);
PushButtonTextOff(&g_sTemperature);
// PushButtonFillOn(&g_sTemperature);
WidgetPaint((tWidget *)&g_sTemperature);

//
// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sLiquid);
PushButtonTextOff(&g_sLiquid);
// PushButtonFillOn(&g_sLiquid);
WidgetPaint((tWidget *)&g_sLiquid);

//
// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sNone);
PushButtonTextOff(&g_sNone);
//PushButtonFillOn(&g_sNone);
WidgetPaint((tWidget *)&g_sNone);

//
// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sAcceleration);
PushButtonTextOff(&g_sAcceleration);
// PushButtonFillOn(&g_sAcceleration);
WidgetPaint((tWidget *)&g_sAcceleration);

// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sXaxis);
PushButtonTextOff(&g_sXaxis);
// PushButtonFillOn(&g_sXaxis);
WidgetPaint((tWidget *)&g_sXaxis);

// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sYaxis);
PushButtonTextOff(&g_sYaxis);
//PushButtonFillOn(&g_sYaxis);
```

```
WidgetPaint((tWidget *)&g_sYaxis);

// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sZaxis);
PushButtonTextOff(&g_sZaxis);
//PushButtonFillOn(&g_sZaxis);
WidgetPaint((tWidget *)&g_sZaxis);

// Clear the button from the display since the last panel is being
// displayed.
//
PushButtonImageOff(&g_sIllustration);
PushButtonTextOff(&g_sIllustration);
//PushButtonFillOn(&g_sIllustration);
WidgetPaint((tWidget *)&g_sIllustration);

//
// Display the main button.
//
PushButtonImageOff(&g_sMain);
PushButtonTextOff(&g_sMain);
//PushButtonFillON(&g_sMain);
WidgetPaint((tWidget *)&g_sMain);

//
// Display the back button.
//
PushButtonImageOn(&g_sBack);
PushButtonTextOn(&g_sBack);
PushButtonFillOff(&g_sBack);
WidgetPaint((tWidget *)&g_sBack);

//
// Play the key click sound.
//
ClickPlay();
}
else//其他情况案件不响应
{
return;
}
}
```

其他的处理方法大同小异，这里不再赘述!

2、接下来是每个界面画布的函数处理：

首先是添加界面的标题数组：

```
const char *g_pcPanelNames[] =
{
 “主界面”,
 “说明”,
 “加速度”,
 “X轴”,
 “Y轴”,
 “Z轴”,
 “温度”,
 “液位”,
 “保留”,
};
```

这里实在相应的按钮调用函数里来切换标题的。

然后是每个界面的主画布的添加：

```
//*****************************************************************************
//
// 第一个，主界面画布
//
//
//*****************************************************************************
Canvas(g_sMaininterface, g_psPanels, 0, 0, &g_s35_480x272x24Display,
0, 40, 480, 232, CANVAS_STYLE_APP_DRAWN, 0, 0, 0, 0, 0,
0, OnMaininterfacePaint);

//*****************************************************************************
//
//第二个，说明界面画布
//
//*****************************************************************************
Canvas(g_sIntroduction, g_psPanels+1, 0, 0, &g_s35_480x272x24Display, 0,
40, 480, 232, CANVAS_STYLE_APP_DRAWN, 0, 0, 0, 0, 0, 0,
OnIntroPaint);

//*****************************************************************************
//
// 第三个。整体加速度测角的界面画布，这里定义了CANVAS_STYLE_IMG,需要用于显示表盘图
片!
//
//*****************************************************************************
Canvas(g_sAcc, g_psPanels + 2, 0, 0, &g_s35_480x272x24Display, 0,
40, 480, 232, CANVAS_STYLE_APP_DRAWN | CANVAS_STYLE_IMG, 0,
ClrGray, 0, 0, 0, g_Acceleration, OnAccPaint);
//*****************************************************************************
//
// 第四个，X轴表盘
//
//*****************************************************************************
Canvas(g_sXaxis1, g_psPanels + 3, 0, 0, &g_s35_480x272x24Display, 0,
40, 480, 232, CANVAS_STYLE_APP_DRAWN | CANVAS_STYLE_IMG, 0,
ClrGray, 0, 0, 0, g_Accsingle, OnXaxisPaint);
```

```
//*****************************************************************************
//
// 第五个。Y轴表盘
//
//*****************************************************************************
Canvas(g_sYaxis1, g_psPanels + 4, 0, 0, &g_s35_480x272x24Display, 0,
40, 480, 232, CANVAS_STYLE_APP_DRAWN | CANVAS_STYLE_IMG, 0,
ClrGray, 0, 0, 0, g_Accsingle, OnYaxisPaint);

//*****************************************************************************
//
// 的六个，Z轴表盘
//
//*****************************************************************************
Canvas(g_sZaxis1, g_psPanels + 5, 0, 0, &g_s35_480x272x24Display, 0,
40, 480, 232, CANVAS_STYLE_APP_DRAWN | CANVAS_STYLE_IMG, 0,
ClrGray, 0, 0, 0, g_Accsingle, OnZaxisPaint);
//*****************************************************************************
//
//第七个，温度界面，加进度条!
//
//*****************************************************************************
Canvas(g_sTemValueCanvas, g_psPanels + 6, 0, 0,
&g_s35_480x272x24Display, 210+X_OFFSET, 30+Y_OFFSET, 60, 40,
CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_OPAQUE, ClrBlack, 0, ClrSilver,
&g_sFontCm24, "50%",
0, 0);

tSliderWidget g_sTem[] =
{
SliderStruct(g_psPanels + 6, g_sTem + 1, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 115+Y_OFFSET, 220, 30,
0, 100, 25, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL |
SL_STYLE_OUTLINE | SL_STYLE_TEXT | SL_STYLE_BACKG_TEXT),
ClrGray, ClrBlack, ClrSilver, ClrWhite, ClrWhite,
&g_sFontCm20, "25%", 0, 0, OnTemChange),
SliderStruct(g_psPanels + 6, g_sTem + 2, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 155+Y_OFFSET, 220, 25,
0, 100, 25, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL |
SL_STYLE_OUTLINE | SL_STYLE_TEXT),
ClrWhite, ClrBlueViolet, ClrSilver, ClrBlack, 0,
&g_sFontCm18, "Foreground Text Only", 0, 0, OnTemChange),
SliderStruct(g_psPanels + 6, g_sTem + 3, 0,
&g_s35_480x272x24Display, 240+X_OFFSET, 70+Y_OFFSET, 26, 110,
0, 100, 50, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL |
SL_STYLE_VERTICAL | SL_STYLE_OUTLINE | SL_STYLE_LOCKED),
ClrDarkGreen, ClrDarkRed, ClrSilver, 0, 0, 0, 0, 0, 0, 0),
SliderStruct(g_psPanels + 6, g_sTem + 4, 0,
&g_s35_480x272x24Display, 280+X_OFFSET, 30+Y_OFFSET, 30, 150,
0, 100, 75, (SL_STYLE_IMG | SL_STYLE_BACKG_IMG |
```

```
SL_STYLE_VERTICAL | SL_STYLE_OUTLINE), 0, ClrBlack, ClrSilver,
0, 0, 0, 0, g_pucGettingHotter28x148,
g_pucGettingHotter28x148Mono, OnTemChange),
SliderStruct(g_psPanels + 6, g_sTem + 5, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 30+Y_OFFSET, 195, 37, 0,
100, 50, SL_STYLE_IMG | SL_STYLE_BACKG_IMG, 0, 0, 0, 0, 0, 0,
0, g_pucGreenSlider195x37, g_pucRedSlider195x37,
OnTemChange),
SliderStruct(g_psPanels + 6, &g_sTemValueCanvas, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 80+Y_OFFSET, 220, 25, 0,
100, 50, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL | SL_STYLE_TEXT |
SL_STYLE_BACKG_TEXT | SL_STYLE_TEXT_OPAQUE |
SL_STYLE_BACKG_TEXT_OPAQUE), ClrBlue, ClrYellow, ClrSilver,
ClrYellow, ClrBlue, &g_sFontCm18, "Text in both areas", 0, 0,
OnTemChange),
};

//*****************************************************************************
//
// 第八个，液位界面，加进度条
//
//*****************************************************************************
Canvas(g_sLiqValueCanvas, g_psPanels + 7, 0, 0,
&g_s35_480x272x24Display, 210+X_OFFSET, 30+Y_OFFSET, 60, 40,
CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_OPAQUE, ClrBlack, 0, ClrSilver,
&g_sFontCm24, "50%",
0, 0);

tSliderWidget g_sLiq[] =
{
SliderStruct(g_psPanels + 7, g_sLiq + 1, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 115+Y_OFFSET, 220, 30,
0, 100, 25, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL |
SL_STYLE_OUTLINE | SL_STYLE_TEXT | SL_STYLE_BACKG_TEXT),
ClrGray, ClrBlack, ClrSilver, ClrWhite, ClrWhite,
&g_sFontCm20, "25%", 0, 0, OnLiqChange),
SliderStruct(g_psPanels + 7, g_sLiq + 2, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 155+Y_OFFSET, 220, 25,
0, 100, 25, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL |
SL_STYLE_OUTLINE | SL_STYLE_TEXT),
ClrWhite, ClrBlueViolet, ClrSilver, ClrBlack, 0,
&g_sFontCm18, "Foreground Text Only", 0, 0, OnLiqChange),
SliderStruct(g_psPanels + 7, g_sLiq + 3, 0,
&g_s35_480x272x24Display, 240+X_OFFSET, 70+Y_OFFSET, 26, 110,
0, 100, 50, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL |
SL_STYLE_VERTICAL | SL_STYLE_OUTLINE | SL_STYLE_LOCKED),
ClrDarkGreen, ClrDarkRed, ClrSilver, 0, 0, 0, 0, 0, 0, 0),
SliderStruct(g_psPanels + 7, g_sLiq + 4, 0,
&g_s35_480x272x24Display, 280+X_OFFSET, 30+Y_OFFSET, 30, 150,
0, 100, 75, (SL_STYLE_IMG | SL_STYLE_BACKG_IMG |
SL_STYLE_VERTICAL | SL_STYLE_OUTLINE), 0, ClrBlack, ClrSilver,
```

```
0, 0, 0, 0, g_pucGettingHotter28x148,
g_pucGettingHotter28x148Mono, OnLiqChange),
SliderStruct(g_psPanels + 7, g_sLiq + 5, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 30+Y_OFFSET, 195, 37, 0,
100, 50, SL_STYLE_IMG | SL_STYLE_BACKG_IMG, 0, 0, 0, 0, 0, 0,
0, g_pucGreenSlider195x37, g_pucRedSlider195x37,
OnLiqChange),
SliderStruct(g_psPanels + 7, &g_sLiqValueCanvas, 0,
&g_s35_480x272x24Display, 5+X_OFFSET, 80+Y_OFFSET, 220, 25, 0,
100, 50, (SL_STYLE_FILL | SL_STYLE_BACKG_FILL | SL_STYLE_TEXT |
SL_STYLE_BACKG_TEXT | SL_STYLE_TEXT_OPAQUE |
SL_STYLE_BACKG_TEXT_OPAQUE), ClrBlue, ClrYellow, ClrSilver,
ClrYellow, ClrBlue, &g_sFontCm18, "Text in both areas", 0, 0,
OnLiqChange),
};

//进度条相关定义

#define SLIDER_TEXT_VAL_INDEX (0)
#define SLIDER_LOCKED_INDEX (2)
#define SLIDER_CANVAS_VAL_INDEX (4)

#define NUM_SLIDERS (sizeof(g_psSliders) / \
sizeof(g_psSliders[0]))
//*****************************************************************************
//
//第九个，保留界面，随便加的东西
//
//*****************************************************************************
tCanvasWidget g_psCheckBoxIndicators[] =
{
CanvasStruct(g_psPanels + 8, g_psCheckBoxIndicators + 1, 0,
&g_s35_480x272x24Display, 230+X_OFFSET, 30+Y_OFFSET, 50, 42,
CANVAS_STYLE_IMG, 0, 0, 0, 0, 0, g_ledOFF, 0),
CanvasStruct(g_psPanels + 8, g_psCheckBoxIndicators + 2, 0,
&g_s35_480x272x24Display, 230+X_OFFSET, 82+Y_OFFSET, 50, 48,
CANVAS_STYLE_IMG, 0, 0, 0, 0, 0, g_ledOFF, 0),
CanvasStruct(g_psPanels + 8, 0, 0,
&g_s35_480x272x24Display, 230+X_OFFSET, 134+Y_OFFSET, 50, 42,
CANVAS_STYLE_IMG, 0, 0, 0, 0, 0, g_ledOFF, 0)
};
tCheckBoxWidget g_psCheckBoxes[] =
{
CheckBoxStruct(g_psPanels + 8, g_psCheckBoxes + 1, 0,
&g_s35_480x272x24Display, 40+X_OFFSET, 30+Y_OFFSET, 185, 42,
CB_STYLE_OUTLINE | CB_STYLE_FILL | CB_STYLE_TEXT, 16,
ClrMidnightBlue, ClrGray, ClrSilver, &g_sFontCm22, "Select",
0, OnCheckChange),
CheckBoxStruct(g_psPanels + 8, g_psCheckBoxes + 2, 0,
&g_s35_480x272x24Display, 40+X_OFFSET, 82+Y_OFFSET, 185, 48,
CB_STYLE_IMG, 16, 0, ClrGray, 0, 0, 0,
```

```
g_TILogo, OnCheckChange),
CheckBoxStruct(g_psPanels + 8, g_psCheckBoxIndicators, 0,
&g_s35_480x272x24Display, 40+X_OFFSET, 134+Y_OFFSET, 189, 42,
CB_STYLE_OUTLINE | CB_STYLE_TEXT, 16,
0, ClrGray, ClrGreen, &g_sFontCm20, "Select",
0, OnCheckChange),
};
```

3、以上画布函数都有相应的调用函数：

```
//*****************************************************************************
//
// //说明界面，使用自己的文字库显示信息
//
//*****************************************************************************
void OnIntroPaint(tWidget *pWidget, tContext *pContext)
{
//
// Display the introduction text in the canvas.
//
GrContextFontSet(pContext, &g_sFontCH16);
GrContextForegroundSet(pContext, ClrSilver);

GrStringDraw(pContext, " 此显示界面是由郑立安 (anananjjj)", -1, 0+X_OFFSET,
50+Y_OFFSET, 0);
GrStringDraw(pContext, "设计开发的,用于显示由zigbee模块接收到", -1,
0+X_OFFSET,68+Y_OFFSET, 0);
GrStringDraw(pContext, "的温度,液位,倾角等数据。此显示系统的", -1,
0+X_OFFSET, 86+Y_OFFSET, 0);
GrStringDraw(pContext, "硬件是基于BeagleBone开发板扩展而成的,", -1,
0+X_OFFSET, 104+Y_OFFSET, 0);
GrStringDraw(pContext, "具有无线,Codec,LCD显示等扩展功能。非常", -1,
0+X_OFFSET, 122+Y_OFFSET, 0);
GrStringDraw(pContext, "感谢电子工程世界论坛(eeworld.com.cn)能", -1,
0+X_OFFSET, 140+Y_OFFSET, 0);
GrStringDraw(pContext, "够提供“我与BeagleBone有个约会”这个活动!", -1,
0+X_OFFSET, 158+Y_OFFSET, 0);
GrStringDraw(pContext, "anananjjj:Zigbee网络主控制台测试版。", -1,
0+X_OFFSET, 176+Y_OFFSET, 0);
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sMain); //删除主界面按钮（不执行的话会在其他界面显示）
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetRemove((tWidget *)&g_sXaxis); //增加X轴按钮
WidgetRemove((tWidget *)&g_sYaxis); //增加Y轴按钮
WidgetRemove((tWidget *)&g_sZaxis); //增加Z轴按钮
}
//*****************************************************************************
```

```
//
// 主界面的显示
//
//*****************************************************************************
void OnMaininterfacePaint(tWidget *pWidget, tContext *pContext)
{

tRectangle sRectT;
tRectangle sRectL;
tRectangle sRectN;
tRectangle sRectX;
tRectangle sRectY;
tRectangle sRectZ;

// Fill the top 24 rows of the screen with blue to create the banner.
sRectT.sXMin = 40;
sRectT.sYMin = 70;
sRectT.sXMax = 200;
sRectT.sYMax = 110;

sRectL.sXMin = 40;
sRectL.sYMin = 130;
sRectL.sXMax = 200;
sRectL.sYMax = 170;

sRectN.sXMin = 40;
sRectN.sYMin = 190;
sRectN.sXMax = 200;
sRectN.sYMax = 230;

sRectX.sXMin = 280;
sRectX.sYMin = 80;
sRectX.sXMax = 440;
sRectX.sYMax = 120;

sRectY.sXMin = 280;
sRectY.sYMin = 130;
sRectY.sXMax = 440;
sRectY.sYMax = 170;

sRectZ.sXMin = 280;
sRectZ.sYMin = 180;
sRectZ.sXMax = 440;
sRectZ.sYMax = 220;
GrContextForegroundSet(&sContext, ClrWhite);
GrRectFill(&sContext, &sRectT);
GrRectFill(&sContext, &sRectL);
GrRectFill(&sContext, &sRectN);
GrRectFill(&sContext, &sRectX);
GrRectFill(&sContext, &sRectY);
GrRectFill(&sContext, &sRectZ);
```

```
// Put a white box around the banner.
GrContextForegroundSet(&sContext, ClrBlue);
GrRectDraw(&sContext, &sRectT);
GrRectDraw(&sContext, &sRectL);
GrRectDraw(&sContext, &sRectN);
GrRectDraw(&sContext, &sRectX);
GrRectDraw(&sContext, &sRectY);
GrRectDraw(&sContext, &sRectZ);
GrContextForegroundSet(&sContext, ClrBlack);
// Put the application name in the middle of the banner.
GrContextFontSet(&sContext, &g_sFontCH16);
GrStringDraw(&sContext, “温度: ℃”, -1,
45, 80, 0);
GrStringDraw(&sContext, “液位: cm”, -1,
45, 140, 0);
GrStringDraw(&sContext, “保留: “, -1,
45, 200, 0);
GrStringDraw(&sContext, “X轴: ℃;X: g”, -1,
285, 90, 0);
GrStringDraw(&sContext, “Y轴: ℃;Y: g”, -1,
285, 140, 0);
GrStringDraw(&sContext, “Z轴: ℃;X: g”, -1,
285, 190, 0);

// Add the title block and the previous and next buttons to the widget tree.
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sTemperature); //温度按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sLiquid); //液位按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sNone); //备用按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sAcceleration);//加速度按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sIllustration); //说明按钮

WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //增加X轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //增加Y轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //增加Z轴按钮
WidgetRemove((tWidget *)&g_sMain); //删除主界面按钮（不执行的话会在其他界面显示）
WidgetRemove((tWidget *)&g_sBack); //删除返回按钮（不执行的话会在其他界面显示）
}

//*****************************************************************************
//
// 加速度界面显示(画线）//处理三个表盘的指
//
//*****************************************************************************
void OnAccPaint(tWidget *pWidget, tContext *pContext)
{
int Xx,Xy,Yx,Yy,Zx,Zy;
Xx=(int)(80*sin(2*(3.1415926)*XAngle/360));
Xy=(int)(80*cos(2*(3.1415926)*XAngle/360));
Yx=(int)(80*sin(2*(3.1415926)*YAngle/360));
Yy=(int)(80*cos(2*(3.1415926)*YAngle/360));
Zx=(int)(80*sin(2*(3.1415926)*ZAngle/360));
```

```
Zy=(int)(80*cos(2*(3.1415926)*ZAngle/360));

GrContextForegroundSet(pContext, ClrBlack);
//画表盘指针

GrLineDraw(pContext,80,182,80+Xx,182-Xy);
GrLineDraw(pContext,240,182,240+Yx,182-Yy);
GrLineDraw(pContext,400,182,400+Zx,182-Zy);

WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sMain); //删除主界面按钮(不执行的话会在其他界面显示)
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //X轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //Y轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //Z轴按钮

}

//*****************************************************************************
//
// X轴加速度界面显示(画线)
//
//*****************************************************************************
void OnXaxisPaint(tWidget *pWidget, tContext *pContext)
{
int Xx,Xy;
Xx=(int)(160*sin(2*(3.1415926)*XAngle/360));
Xy=(int)(160*cos(2*(3.1415926)*XAngle/360));

GrContextForegroundSet(pContext, ClrBlack);
//画表盘指针

GrLineDraw(pContext,239,236,239+Xx,236-Xy);

WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sMain); //增加主界面按钮

WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //增加Y轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //增加Z轴按钮

WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //X轴按钮
```

```
}

//*****************************************************************************
//
// Y轴加速度界面显示(画线)
//
//*****************************************************************************
void OnYaxisPaint(tWidget *pWidget, tContext *pContext)
{

int Yx,Yy;
Yx=(int)(160*sin(2*(3.1415926)*YAngle/360));
Yy=(int)(160*cos(2*(3.1415926)*YAngle/360));

GrContextForegroundSet(pContext, ClrBlack);
//画表盘指针

GrLineDraw(pContext,239,236,239+Yx,236-Yy);

WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sMain); //增加主界面按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //增加X轴按钮

WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //增加Z轴按钮

WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //Y轴按钮

}

//*****************************************************************************
//
// Z轴加速度界面显示(画线)
//
//*****************************************************************************
void OnZaxisPaint(tWidget *pWidget, tContext *pContext)
{

int Zx,Zy;
Zx=(int)(160*sin(2*(3.1415926)*ZAngle/360));
Zy=(int)(160*cos(2*(3.1415926)*ZAngle/360));

GrContextForegroundSet(pContext, ClrBlack);
//画表盘指针

GrLineDraw(pContext,239,236,239+Zx,236-Zy);
```

```
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sMain); //增加主界面按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //增加X轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //增加Y轴按钮

WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //Z轴按钮
}

//*****************************************************************************
//
// 保留界面.
//
//*****************************************************************************
void OnCheckChange(tWidget *pWidget, unsigned int bSelected)
{
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sMain); //删除主界面按钮(不执行的话会在其他界面显示)
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //X轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //Y轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //Z轴按钮
unsigned long ulIdx;

//
// Find the index of this check box.
//
for(ulIdx = 0; ulIdx < NUM_CHECK_BOXES; ulIdx++)
{
if(pWidget == (tWidget *)(g_psCheckBoxes + ulIdx))
{
break;
}
}

//
// Return if the check box could not be found.
//
if(ulIdx == NUM_CHECK_BOXES)
{
return;
}
```

```
//
// Set the matching indicator based on the selected state of the check box.
//
CanvasImageSet(g_psCheckBoxIndicators + ulIdx,
bSelected ? g_ledON : g_ledOFF);
WidgetPaint((tWidget *)(g_psCheckBoxIndicators + ulIdx));

//
// Play the key click sound.
//
ClickPlay();
}

//*****************************************************************************
//
//温度界面
//
//*****************************************************************************
void OnTemChange(tWidget *pWidget, int lValue)
{
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sMain); //删除主界面按钮(不执行的话会在其他界面显示)
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //X轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //Y轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //Z轴按钮
static char pcCanvasText[5];
static char pcSliderText[5];

//
// Is this the widget whose value we mirror in the canvas widget and the
// locked slider?
//
if(pWidget == (tWidget *)&g_sTem[SLIDER_CANVAS_VAL_INDEX])
{
//
// Yes - update the canvas to show the slider value.
//
usprintf(pcCanvasText, "%3d%%", lValue);
CanvasTextSet(&g_sTemValueCanvas, pcCanvasText);
WidgetPaint((tWidget *)&g_sTemValueCanvas);

//
// Also update the value of the locked slider to reflect this one.
//
SliderValueSet(&g_sTem[SLIDER_LOCKED_INDEX], lValue);
WidgetPaint((tWidget *)&g_sTem[SLIDER_LOCKED_INDEX]);
```

```
}

if(pWidget == (tWidget *)&g_sTem[SLIDER_TEXT_VAL_INDEX])
{
//
// Yes - update the canvas to show the slider value.
//
usprintf(pcSliderText, "%3d%%", lValue);
SliderTextSet(&g_sTem[SLIDER_TEXT_VAL_INDEX], pcSliderText);
WidgetPaint((tWidget *)&g_sTem[SLIDER_TEXT_VAL_INDEX]);
}
}
//*****************************************************************************
//
//液位界面
//
//*****************************************************************************
void OnLiqChange(tWidget *pWidget, int lValue)
{
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBack); //增加返回按钮
WidgetRemove((tWidget *)&g_sTemperature); //温度按钮
WidgetRemove((tWidget *)&g_sLiquid); //液位按钮
WidgetRemove((tWidget *)&g_sNone); //备用按钮
WidgetRemove((tWidget *)&g_sAcceleration);//加速度按钮
WidgetRemove((tWidget *)&g_sMain); //删除主界面按钮(不执行的话会在其他界面显示)
WidgetRemove((tWidget *)&g_sIllustration); //说明按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sXaxis); //X轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sYaxis); //Y轴按钮
WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sZaxis); //Z轴按钮
static char pcCanvasText[5];
static char pcSliderText[5];

//
// Is this the widget whose value we mirror in the canvas widget and the
// locked slider?
//
if(pWidget == (tWidget *)&g_sTem[SLIDER_CANVAS_VAL_INDEX])
{
//
// Yes - update the canvas to show the slider value.
//
usprintf(pcCanvasText, "%3d%%", lValue);
CanvasTextSet(&g_sTemValueCanvas, pcCanvasText);
WidgetPaint((tWidget *)&g_sTemValueCanvas);

//
// Also update the value of the locked slider to reflect this one.
//
SliderValueSet(&g_sTem[SLIDER_LOCKED_INDEX], lValue);
WidgetPaint((tWidget *)&g_sTem[SLIDER_LOCKED_INDEX]);
}
```

```
if(pWidget == (tWidget *)&g_sTem[SLIDER_TEXT_VAL_INDEX])
{
//
// Yes - update the canvas to show the slider value.
//
usprintf(pcSliderText, "%3d%%", lValue);
SliderTextSet(&g_sTem[SLIDER_TEXT_VAL_INDEX], pcSliderText);
WidgetPaint((tWidget *)&g_sTem[SLIDER_TEXT_VAL_INDEX]);
}
}
```

好了函数大概就是这个过程!

点击查看详情：>>Beaglebone 外围电路设计：数据采集界面的实现!

点击查看详情：>> Beaglebone 外围电路设计：数据的无线采集和显示效果

1.3.14 基于 BeagleBone 的 WIFI 通信

硬件环境：BeagleBone

无线网卡：TL-WN321G+

操作系统：linux-3.1.0-psp04.06.00.03.sdk

图 1-133

1. Linux-3.1.0 内核已经支持 RT73 的驱动，所以配置一下内核就可以了，主要由以下两个步骤：

1）无线网络协议栈的配置

```
Networking support    —>
Wireless    —>  Generic IEEE 802.11 Networking Stack (mac80211)
```

如果没有配置 MAC80211，是看不到 RT73 的驱动的。

2）选择 RT73 USB 无线网卡驱动

Device Drivers —> Network device support —> Wireless LAN —> Ralink driver support —> <*> Ralink rt2501/rt73(USB) support

2. 接着 Make uImage，把内核映像文件拷贝到 SD 卡中

3. 安装 wireless_tools

opkg install wireless-tools_29-r4_armv7a.ipk 可从

http://bbs.eeworld.com.cn/thread-330851-1-1.htm

下载附件 wireless-tools_29-r4_armv7a.zip

4. 插入 USB 无线网卡，查看 USB 设备

```
root@BeagleBone:/etc# lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 148f:2573 Ralink Technology, Corp. RT2501USB Wireless
Adapter
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

5. 激活网卡

```
root@BeagleBone:~# ifconfig wlan0 up
root@BeagleBone:~# 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:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:280 (280.0 B)  TX bytes:280 (280.0 B)
wlan0     Link encap:Ethernet  HWaddr 00:25:86:AD:C2:D5
          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)
```

6. 扫描可用的无线网络

iwlist wlan0 scan

7. 配置 wpa.conf

新建文件 /etc/wpa.conf

```
 ctrl_interface=/var/run/wpa_supplicant
network={
                                        ssid="无线网路名称"
                                        psk="密码"
}
```

8. 连接 wlan0 到网络

```
wpa_supplicant -B -i wlan0 -c /etc/wpa.conf
-B Background 在后台以daemon 运行
-i interface
-c 配置文件
```

不要用 iwconfig wlan0 来连接网络哦，这个是对 WEP 加密方式的无线网络

9. 设置 IP

```
ifconfig wlan0 192.168.10.125
```

10. 执行 ping

```
root@BeagleBone:/etc# ifconfig wlan0 192.168.10.125
root@BeagleBone:/etc# ping 192.168.10.105
PING 192.168.10.105 (192.168.10.105) 56(84) bytes of data.
64 bytes from 192.168.10.105: icmp_req=1 ttl=64 time=1133 ms
64 bytes from 192.168.10.105: icmp_req=2 ttl=64 time=126 ms
64 bytes from 192.168.10.105: icmp_req=3 ttl=64 time=11.1 ms
```

到此无线网卡安装成功，可以无线通信了!

点击查看详情：>>BeagleBone 的 WIFI 通信

1.3.15 BeagleBone 上实现 AM3359 与 FPGA 的 GPMC 通信

1. 参考资料

《AM335x ARM® Cortex™-A8 Microprocessors Technical Reference Manual》
《BeagleBone_revA3_SCH.pdf》
《BeagleBone_revA3_SRM.pdf》
《BeagleBone_revA3_BOM.xls》

2. 测试的硬件环境：BeagleBone + EE_FPGA

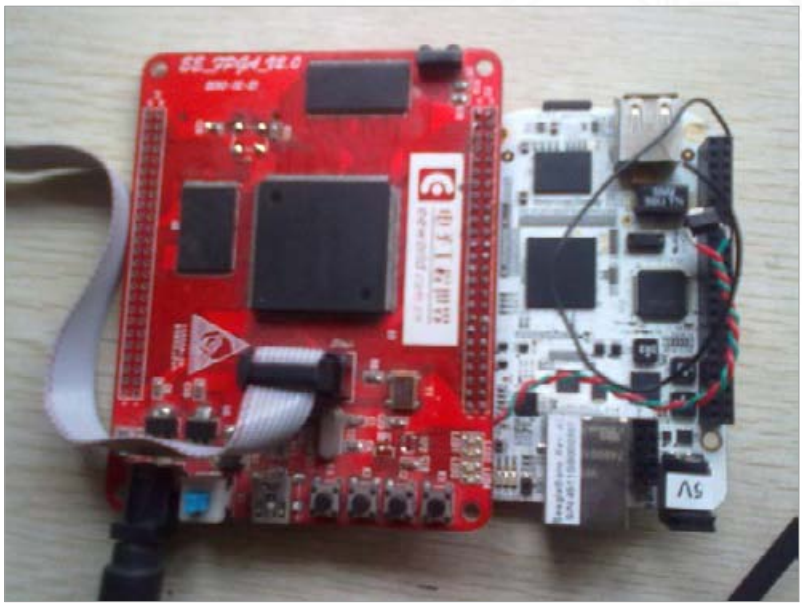


图 1-134

3. 硬件连接图：用的是 GPMC to 16-Bit Nonmultiplexed Memory

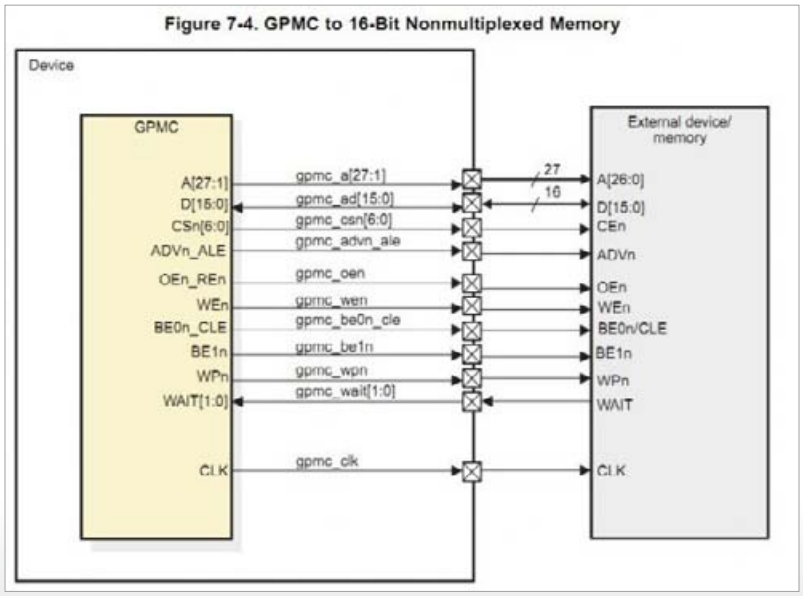


图 1-135

4. 设计思路

初始化 GPMC 相关引脚

完成 GPMC 驱动设计

完成 GPMC 应用设计

5. 初始化相关代码，主要修改并编译

/linux-3.1.0-psp04.06.00.03.sdk/arch/arm/mach-omap2/board-am335xevm.c

配置相关引脚

```
/* Pin mux for fpga module */
static struct pinmux_config fpga_pin_mux[] = {
{"gpmc_ad0.gpmc_ad0", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad1.gpmc_ad1", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad2.gpmc_ad2", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad3.gpmc_ad3", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad4.gpmc_ad4", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad5.gpmc_ad5", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad6.gpmc_ad6", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad7.gpmc_ad7", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad8.gpmc_ad8", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad9.gpmc_ad9", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad10.gpmc_ad10", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad11.gpmc_ad11", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad12.gpmc_ad12", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad13.gpmc_ad13", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad14.gpmc_ad14", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"gpmc_ad15.gpmc_ad15", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
{"lcd_data0.gpmc_a0", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data1.gpmc_a1", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data2.gpmc_a2", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data3.gpmc_a3", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data4.gpmc_a4", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data5.gpmc_a5", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data6.gpmc_a6", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data7.gpmc_a7", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_vsync.gpmc_a8", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_hsync.gpmc_a9", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
```

```
{"lcd_pclk.gpmc_a10", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_ac_bias_en.gpmc_a11", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data8.gpmc_a12", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data9.gpmc_a13", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data10.gpmc_a14", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},
{"lcd_data11.gpmc_a15", OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT
| AM33XX_PULL_DISA},

{"gpmc_advn_ale.gpmc_advn_ale", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},

{"gpmc_oen_ren.gpmc_oen_ren", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},

{"gpmc_wen.gpmc_wen", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},

{"gpmc_ben0_cle.gpmc_ben0_cle", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},

{"gpmc_csn1.gpmc_csn1", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},

{"gpmc_clk.gpmc_clk", OMAP_MUX_MODE0 | AM33XX_PULL_DISA},

{NULL, 0},

};

static void evm_fpga_init(int evm_id, int profile)

{

setup_pin_mux(fpga_pin_mux);

}

/* BeagleBone Rev A3 and after */

static struct evm_dev_cfg BeagleBone_dev_cfg[] = {

{mii1_init, DEV_ON_BASEBOARD, PROFILE_NONE},

{usb0_init, DEV_ON_BASEBOARD, PROFILE_NONE},

{usb1_init, DEV_ON_BASEBOARD, PROFILE_NONE},

{mmc0_init, DEV_ON_BASEBOARD, PROFILE_NONE},

{evm_fpga_init, DEV_ON_BASEBOARD, PROFILE_NONE},

{NULL, 0, 0},
```

6. GPMC 驱动设计

```
/*
Char device driver fpga.
Do a global replace of 'fpga' with your driver name.
*/

#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <asm/uaccess.h>
#include <linux/string.h>
#include <linux/kernel.h>
#include <linux/device.h>
#include <linux/slab.h>

#include <plat/gpmc.h>
#include "plat/gpio.h"
#include "plat/dma.h"

#include "asm/uaccess.h"
#include "asm/io.h"
#include "asm/atomic.h"


#define USER_BUFF_SIZE 128


struct fpga_dev {
dev_t devt;
struct cdev cdev;
struct semaphore sem;
struct class *class;
char *user_buff;
};

static struct fpga_dev fpga_dev;

unsigned long mem_base;
static void __iomem *fpga_base;
static void __iomem *gpmc_base;

/* GPMC register offsets */
#define GPMC_REVISION 0x00
#define GPMC_SYSCONFIG 0x10
#define GPMC_SYSSTATUS 0x14
#define GPMC_IRQSTATUS 0x18
#define GPMC_IRQENABLE 0x1c
#define GPMC_TIMEOUT_CONTROL 0x40
#define GPMC_ERR_ADDRESS 0x44
```

```
#define GPMC_ERR_TYPE 0x48
#define GPMC_CONFIG 0x50
#define GPMC_STATUS 0x54
#define GPMC_PREFETCH_CONFIG1 0x1e0
#define GPMC_PREFETCH_CONFIG2 0x1e4
#define GPMC_PREFETCH_CONTROL 0x1ec
#define GPMC_PREFETCH_STATUS 0x1f0
#define GPMC_ECC_CONFIG 0x1f4
#define GPMC_ECC_CONTROL 0x1f8
#define GPMC_ECC_SIZE_CONFIG 0x1fc
#define GPMC_ECC1_RESULT 0x200
#define GPMC_ECC_BCH_RESULT_0 0x240

#define GPMC_BASE_ADDR 0x50000000
#define GPMC_CS 1
#define GPMC_CS0 0x60
#define GPMC_CS_SIZE 0x30
#define STNOR_GPMC_CONFIG1 0x28601000
#define STNOR_GPMC_CONFIG2 0x00011001
#define STNOR_GPMC_CONFIG3 0x00020201
#define STNOR_GPMC_CONFIG4 0x08031003
#define STNOR_GPMC_CONFIG5 0x000f1111
#define STNOR_GPMC_CONFIG6 0x0f030080
static const u32 gpmc_nor[7] = {
STNOR_GPMC_CONFIG1,
STNOR_GPMC_CONFIG2,
STNOR_GPMC_CONFIG3,
STNOR_GPMC_CONFIG4,
STNOR_GPMC_CONFIG5,
STNOR_GPMC_CONFIG6, 0
};

static ssize_t fpga_write(struct file *filp, const char __user *buff,
size_t count, loff_t *f_pos)
{
ssize_t status;
size_t len = USER_BUFF_SIZE - 1;
int i,tmp;

if (count == 0)
return 0;

if (down_interruptible(&fpga_dev.sem))
return -ERESTARTSYS;

if (len > count)
len = count;

memset(fpga_dev.user_buff, 0, USER_BUFF_SIZE);

if (copy_from_user(fpga_dev.user_buff, buff, len)) {
```

```
status = -EFAULT;
goto fpga_write_done;
}

/* do something with the user data */

printk("fpga_write \n");
for (i = 0; i < len; i=i+2) {
tmp = fpga_dev.user_buff | fpga_dev.user_buff[i+1] << 8;
writew(tmp,fpga_base+i);
}

for (i = 0; i < len; i++) {
printk("0x%x ",fpga_dev.user_buff);
}

printk("\n");

fpga_write_done:

up(&fpga_dev.sem);

return status;
}

static ssize_t fpga_read(struct file *filp, char __user *buff,
size_t count, loff_t *offp)
{
ssize_t status;
size_t len;
// int i,tmp;

/*
Generic user progs like cat will continue calling until we
return zero. So if *offp != 0, we know this is at least the
second call.
*/
if (*offp > 0)
return 0;

if (down_interruptible(&fpga_dev.sem))
return -ERESTARTSYS;

strcpy(fpga_dev.user_buff, "fpga driver data goes here\n");

len = strlen(fpga_dev.user_buff);

if (len > count)
len = count;

if (copy_to_user(buff, fpga_dev.user_buff, len)) {
```

```
status = -EFAULT;
goto fpga_read_done;
}

fpga_read_done:

up(&fpga_dev.sem);

return status;
}

static int fpga_open(struct inode *inode, struct file *filp)
{
int status = 0;

if (down_interruptible(&fpga_dev.sem))
return -ERESTARTSYS;

if (!fpga_dev.user_buff) {
fpga_dev.user_buff = kmalloc(USER_BUFF_SIZE, GFP_KERNEL);

if (!fpga_dev.user_buff) {
printk(KERN_ALERT "fpga_open: user_buff alloc failed\n");
status = -ENOMEM;
}
}

up(&fpga_dev.sem);

return status;
}

static const struct file_operations fpga_fops = {
.owner = THIS_MODULE,
.open = fpga_open,
.read = fpga_read,
.write = fpga_write,
};

static int __init fpga_init_cdev(void)
{
int error;
u32 val;

fpga_dev.devt = MKDEV(0, 0);

error = alloc_chrdev_region(&fpga_dev.devt, 0, 1, "fpga");
if (error) {
printk(KERN_ALERT "alloc_chrdev_region() failed: %d\n", error);
return error;
}
```

```
cdev_init(&fpga_dev.cdev, &fpga_fops);
fpga_dev.cdev.owner = THIS_MODULE;

error = cdev_add(&fpga_dev.cdev, fpga_dev.devt, 1);
if (error) {
printk(KERN_ALERT "cdev_add() failed: %d\n", error);
unregister_chrdev_region(fpga_dev.devt, 1);
return error;
}

printk("Getting Chip Select\n");

// val = 0xf64;
// gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG7, val);
//gpmc_base = ioremap(GPMC_BASE_ADDR, SZ_4K);
val = gpmc_read_reg(GPMC_REVISION);
printk("GPMC revision %d.%d\n", (val >> 4) & 0x0f, val & 0x0f);


gpmc_write_reg(GPMC_IRQENABLE, 0);
gpmc_write_reg(GPMC_TIMEOUT_CONTROL, 0);


gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG1, gpmc_nor[0]);
gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG2, gpmc_nor[1]);
gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG3, gpmc_nor[2]);
gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG4, gpmc_nor[3]);
gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG5, gpmc_nor[4]);
gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG6, gpmc_nor[5]);
val = gpmc_cs_read_reg(GPMC_CS, GPMC_CS_CONFIG7);
printk("GPMC_CS_CONFIG7 value 0x%x\n", val);

if (gpmc_cs_request(GPMC_CS, SZ_2K, (unsigned long *)&mem_base) < 0){
printk(KERN_ERR "Failed request for GPMC mem for usrp_e\n");
return -1;
}
printk("Got CS0, address = %lx\n", mem_base);

if (!request_mem_region(mem_base, SZ_2K, "mem_fpga")){
printk(KERN_ERR "Request_mem_region failed.\n");
gpmc_cs_free(GPMC_CS);
return -1;
}

fpga_base = ioremap(mem_base, SZ_2K);

return 0;
}

static int __init fpga_init_class(void)
```

```
{
struct device *device;

fpga_dev.class = class_create(THIS_MODULE, "fpga");

if (IS_ERR(fpga_dev.class)) {
printk(KERN_ALERT "class_create(fpga) failed\n");
return PTR_ERR(fpga_dev.class);
}

device = device_create(fpga_dev.class, NULL, fpga_dev.devt, NULL,
"fpga");

if (IS_ERR(device)) {
class_destroy(fpga_dev.class);
return PTR_ERR(device);
}

return 0;
}

static int __init fpga_init(void)
{
printk(KERN_INFO "fpga_init()\n");

memset(&fpga_dev, 0, sizeof(struct fpga_dev));

sema_init(&fpga_dev.sem, 1);

if (fpga_init_cdev())
goto init_fail_1;

if (fpga_init_class())
goto init_fail_2;

return 0;

init_fail_2:
cdev_del(&fpga_dev.cdev);
unregister_chrdev_region(fpga_dev.devt, 1);

init_fail_1:

return -1;
}
module_init(fpga_init);

static void __exit fpga_exit(void)
{
printk(KERN_INFO "fpga_exit()\n");
```

```
device_destroy(fpga_dev.class, fpga_dev.devt);
class_destroy(fpga_dev.class);

cdev_del(&fpga_dev.cdev);
unregister_chrdev_region(fpga_dev.devt, 1);

release_mem_region(mem_base, SZ_2K);
gpmc_cs_free(GPMC_CS);
iounmap(fpga_base);

if (fpga_dev.user_buff)
kfree(fpga_dev.user_buff);
}
module_exit(fpga_exit);


MODULE_AUTHOR("chenzhufly");
MODULE_DESCRIPTION("fpga driver");
MODULE_LICENSE("Dual BSD/GPL");
MODULE_VERSION("0.1");
```

7. GPMC 应用程序设计，目前只实现了 BeagleBone 向 FPGA 写的流程

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>

#define FPGA_DEV "/dev/fpga"
#define PAGE_SIZE 10


int main(void)
{
int fd,i,res;
unsigned char buf[PAGE_SIZE];

printf("GPMC Test version 1.0-BeagleBone Build on %s %s\n\r",__DATE__,__TIME__);
fd=open(FPGA_DEV,O_RDWR);
if(fd<0)
{
printf("Can't Open %s !!!\n\r",FPGA_DEV);
return -1;
}


for(i=0;i<sizeof(buf);i++)
{
buf = i;
}
```

```
write(fd,&buf,PAGE_SIZE);

close(fd);
return 0;
}
```

8. FPGA 相关设计

ARM 和 FPGA 的管脚分配如下图所示

BeagleBone A3 Port 8 Multiplexer table - A3 Printe

PORT	PIN	PROC	NAME	MODE0	MODE1	fpga	MODE2	MODE3
P8_	1		GND					
P8_	2		GND					
P8_	3	R9	GPIO1_6	gpmc_ad6	mmc1_dat6	IO_24		
P8_	4	T9	GPIO1_7	gpmc_ad7	mmc1_dat7	IO_27		
P8_	5	R8	GPIO1_2	gpmc_ad2	mmc1_dat2	IO_29		
P8_	6	T8	GPIO1_3	gpmc_ad3	mmc1_dat6	IO_30		
P8_	7	R7	TIMER4	gpmc_advn_ale		IO_31	timer4	
P8_	8	T7	TIMER7	gpmc_oen_ren		IO_33	timer7	
P8_	9	T6	TIMER5	gpmc_be0n_cle		IO_34	timer5	
P8_	10	U6	TIMER6	gpmc_wen		IO_35	timer6	
P8_	11	R12	GPIO1_13	gpmc_ad13	lcd_data18	IO_37	mmc1_dat5	mmc2_da
P8_	12	T12	GPIO1_12	GPMC_AD12	LCD_DATA19	IO_39	MMC1_DAT4	MMC2_Da
P8_	13	T10	EHRPWM2B	gpmc_ad9	lcd_data22	IO_40	mmc1_dat1	mmc2_da
P8_	14	T11	GPIO0_26	gpmc_ad10	lcd_data21	IO_41	mmc1_dat2	mmc2_da
P8_	15	U13	GPIO1_15	gpmc_ad15	lcd_data16	IO_43	mmc1_dat7	mmc2_da
P8_	16	V13	GPIO1_14	gpmc_ad14	lcd_data17	IO_44	mmc1_dat6	mmc2_da
P8_	17	U12	GPIO0_27	gpmc_ad11	lcd_data20	IO_45	mmc1_dat3	mmc2_da
P8_	18	V12	GPIO2_1	gpmc_clk_mux0	lcd_memory_clk	IO_46	gpmc_wait1	mmc2_c
P8_	19	U10	EHRPWM2A	gpmc_ad8	lcd_data23	IO_47	mmc1_dat0	mmc2_da
P8_	20	V9	GPIO1_31	gpmc_csn2	gpmc_be1n	IO_48	mmc1_cmd	
P8_	21	U9	GPIO1_30	gpmc_csn1	gpmc_clk	IO_56	mmc1_clk	
P8_	22	V8	GPIO1_5	gpmc_ad5		IO_57	mmc1_dat5	
P8_	23	U8	GPIO1_4	gpmc_ad4	mmc1_dat3	IO_58		
P8_	24	V7	GPIO1_1	gpmc_ad1	mmc1_dat1	IO_59		
P8_	25	U7	GPIO1_0	gpmc_ad0	mmc1_dat0	IO_60		
P8_	26	V6	GPIO1_29	gpmc_csn0		IO_61		
P8_	27	U5	GPIO2_22	lcd_vsync	gpmc_a8	IO_63		
P8_	28	V5	GPIO2_24	lcd_pclk	gpmc_a10	IO_64		
P8_	29	R5	GPIO2_23	lcd_hsync	gpmc_a9	IO_67		
P8_	30	R6	GPIO2_25	lcd_ac_bias_en	gpmc_a11	IO_68		
P8_	31	V4	UART5_CTSN	lcd_data14	gpmc_a18	IO_69	eQEP1_index	mcasp0
P8_	32	T5	UART5_RTSN	lcd_data15	gpmc_a19	IO_70	eQEP1_strobe	mcasp0
P8_	33	V3	UART4_RTSN	lcd_data13	gpmc_a17	IO_72	eQEP1B_in	mcasp0

表 1-15

FPGA 简要代码如下所示：

```
module interface_test
(
input clk,
 input ebgcs,
 input eboe,
 input ebwe,
 input [11:0]ebaddr,
input [15:0]ebdata,
output led,
output led1
);
wire [15:0] ram_q;
wire ram_wren;
wire ram_rden;
assign ram_wren = ((ebgcs == 1'b0) && (ebwe == 1'b0)) ? 1'b1 : 1'b0;
assign ram_rden = ((ebgcs == 1'b0) && (eboe == 1'b0)) ? 1'b1 : 1'b0;
TestRam testram_inst (
```

```
.clock ( clk),
.data ( ebdata[15:0] ),
.rdaddress ( ebaddr[11:1] ),
.rden ( ram_rden ),
.wraddress ( ebaddr[11:1] ),
.wren ( ram_wren ),
.q ( ram_q[15:0] )
);

wire ARM_READ_REG_EN;
assign ARM_READ_REG_EN = ((ebgcs == 1'b0) && (eboe == 1'b0) && (ebaddr[11:1] ==
11'b111_1111_1111)) ? 1'b1 : 1'b0;
assign led1 = ARM_READ_REG_EN ? 1'b1 :
ram_rden ? ram_q[0] : 1'bz;

endmodule
```

FPGA 管脚分配如下所示：

```
set_location_assignment PIN_89 -to ebaddr[0]
set_location_assignment PIN_90 -to ebaddr[1]
set_location_assignment PIN_87 -to ebaddr[2]
set_location_assignment PIN_88 -to ebaddr[3]
set_location_assignment PIN_84 -to ebaddr[4]
set_location_assignment PIN_86 -to ebaddr[5]
set_location_assignment PIN_81 -to ebaddr[6]
set_location_assignment PIN_82 -to ebaddr[7]
set_location_assignment PIN_63 -to ebaddr[8]
set_location_assignment PIN_67 -to ebaddr[9]
set_location_assignment PIN_64 -to ebaddr[10]
set_location_assignment PIN_68 -to ebaddr[11]

set_location_assignment PIN_60 -to ebdata[0]
set_location_assignment PIN_59 -to ebdata[1]
set_location_assignment PIN_28 -to ebdata[2]
set_location_assignment PIN_30 -to ebdata[3]
set_location_assignment PIN_58 -to ebdata[4]
set_location_assignment PIN_57 -to ebdata[5]
set_location_assignment PIN_24 -to ebdata[6]
set_location_assignment PIN_27 -to ebdata[7]
set_location_assignment PIN_47 -to ebdata[8]
set_location_assignment PIN_40 -to ebdata[9]
set_location_assignment PIN_41 -to ebdata[10]
set_location_assignment PIN_45 -to ebdata[11]
set_location_assignment PIN_39 -to ebdata[12]
set_location_assignment PIN_37 -to ebdata[13]
set_location_assignment PIN_44 -to ebdata[14]
set_location_assignment PIN_43 -to ebdata[15]

set_location_assignment PIN_61 -to ebgcs
set_location_assignment PIN_33 -to eboe
```

```
set_location_assignment PIN_35 -to ebwe

set_location_assignment PIN_23 -to clk
set_location_assignment PIN_31 -to led
set_location_assignment PIN_34 -to led1
```

9. 测试结果

BeagleBone 向 EE_FPGA 写 0，1，2，3，4，5，6，7，8，9;通过 signaltap 抓的结果。

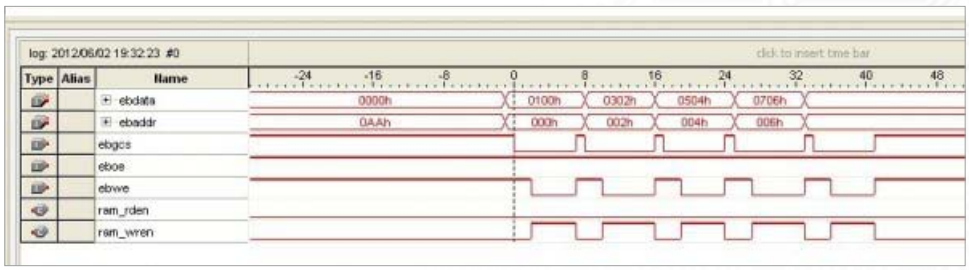


图 1-136

点击查看详情：>> 在 beaglebone 上实现 AM3359 与 FPGA 的 GPMC 通信

1.3.16 BeagleBone 上面试跑飞鸽传书

刚才在坛子里面看到一位兄弟在跑一个 TCP/UDP 的一个测试，想到在这个上面来跑一下飞鸽传书了试一下：

步骤一：程序通过 TF 卡放到 BeagleBone_board 上面，编译生成可执行文件 BeagleBone_ipmsg。

图 1-137

步骤二：运行测试。

在 Windows 上面运行一个飞鸽传书软件，网上下的版本是都支持的，因为自己写的这个协议写得比较简单，但是程序虽然升级了，但是同样是可以向下兼容的，所以版本是莫有问题的。

图 1-138

大家也可以下载附件里面的程序去玩一下，人多的时候比较好玩一点!

点击查看详情：>>BeagleBone 上面试跑飞鸽传书

1.3.17 BeagleBone 试用无刷电机和 2.4G 无线经验分享

连同第二周到现在都很忙啊。现在手上的调试都完成了接下来就是开发板和 BeagleBone 开发板连的简单控制了测试和个模块弄能调试了这个星期的测试主要是 M3 开发板和各模块无线测试。在这星期测试中也出了不少问题要解决的。单桨测试时电调那边芯片出现过热导致出现部分芯片移位和烧坏了部分电阻但进过跟换问题已解决了部分。

1、接下来是部分调试运用到的程序，是运用 TI 例子项目文件测试的和修改运用。
2、电调部分出现过热和更换后电调。
3、2.4G 无线运放的添加以增加遥控范围的功能运用。

相应开发板 PCB 原理图和相应技术文档和参考文档下载

1、在调试过程中芯片会出现被锁的现象解锁软件以前没遇过
LMFlashProgrammer.msi
2、TI 项目文件安装后修改数据可用部分调试和参考
SW-RDK-BLDC-9107.exe
3 、2.4G 无线参考文件和资料
zigbee.rar
4、四轴飞行器开发板 PCB dxp.exe 09 文件需要的可以下载参考
四轴飞行器 PCB.rar
5 、无刷电机工作原理参考文件
ourdev_570001.pdf

>>下载地址，链接为 http://bbs.eeworld.com.cn/thread-353478-1-1.html

项目更大整过项目联系一起测试出现了不少问题。

正在努力解决!!!!!!!

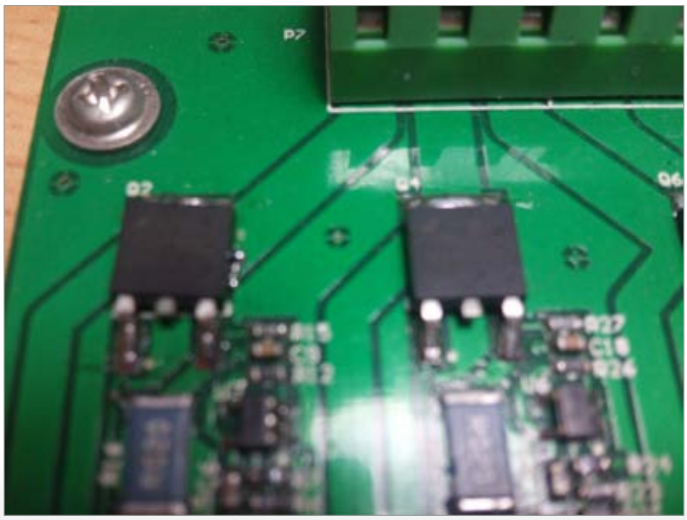


图 1-139

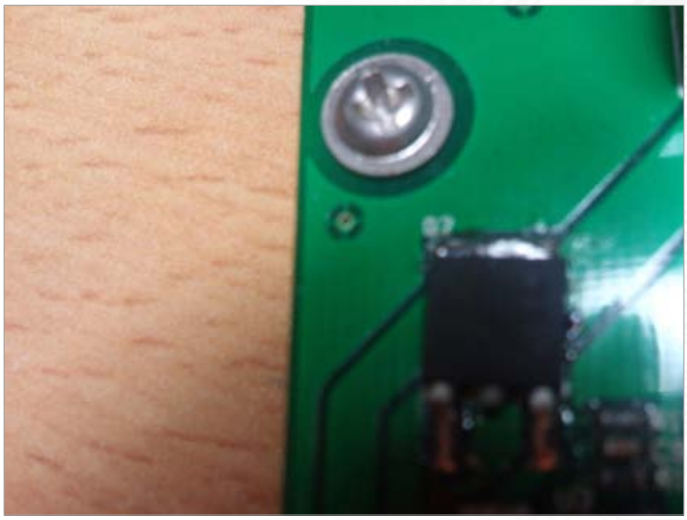


图 1-140

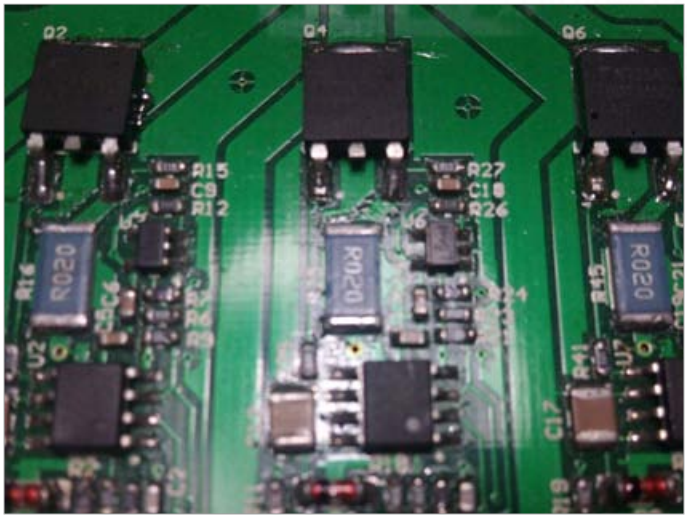


图 1-141

相关下载：>> BeagleBone 试用无刷电机和 2.4G 无线经验分享

1.3.18 BeagleBone 外围电路设计心得 Opencv 无刷电机

开源人脸识别基本功能代码

OpenCV 这方面各项目开发国外已经很开源了，我们虚拟公司也参考和学习修改运用在修改调试过来的知反反复复了多少遍。

一、如何使用 OpenCV 的人脸检测器检测到人脸：

```
/ /执行的输入图像上的人脸检测，使用给定的（Haar）级联。
/ /返回一个矩形，在给定的图像中检测到的区域。
CvRect detectFaceInImage(IplImage *inputImg, CvHaarClassifierCascade* cascade)
{
/ /最小脸的大小.
CvSize minFeatureSize = cvSize(20, 20);
/ /只搜索一面.
int flags = CV_HAAR_FIND_BIGGEST_OBJECT | CV_HAAR_DO_ROUGH_SEARCH;
/ /详细的搜索.
float search_scale_factor = 1.1f;
IplImage *detectImg;
IplImage *greyImg = 0;
CvMemStorage* storage;
CvRect rc;
double t;
CvSeq* rects;
CvSize size;
int i, ms, nFaces;
storage = cvCreateMemStorage(0);
cvClearMemStorage( storage );

/ /如果图像是彩色的，使用的灰度图像的副本.
detectImg = (IplImage*)inputImg;
if (inputImg->nChannels > 1) {
size = cvSize(inputImg->width, inputImg->height);
greyImg = cvCreateImage(size, IPL_DEPTH_8U, 1 );
cvCvtColor( inputImg, greyImg, CV_BGR2GRAY );
detectImg = greyImg; // Use the greyscale image.
}
//检测的灰度图像中的所有面.
t = (double)cvGetTickCount();
rects = cvHaarDetectObjects( detectImg, cascade, storage,
search_scale_factor, 3, flags, minFeatureSize);
t = (double)cvGetTickCount() - t;
ms = cvRound( t / ((double)cvGetTickFrequency() * 1000.0) );
nFaces = rects->total;
printf("Face Detection took %d ms and found %d objects\n", ms, nFaces);
//获取第一个检测到的人脸（最大）.
if (nFaces > 0)
rc = *(CvRect*)cvGetSeqElem( rects, 0 );
else
rc = cvRect(-1,-1,-1,-1); / /找不到的脸.
```

```
if (greyImg)
cvReleaseImage( &greyImg );
cvReleaseMemStorage( &storage );
// cvReleaseHaarClassifierCascade的（级联）;
return rc; / /返回发现最大的面，或（-1，-1，-1，-1）.
}
// Haar的级联文件，使用的人脸检测.
char *faceCascadeFilename = "haarcascade_frontalface_alt.xml";
/ /加载的HaarCascade分类器进行人脸检测.
CvHaarClassifierCascade* faceCascade;
faceCascade = (CvHaarClassifierCascade*)cvLoad(faceCascadeFilename, 0, 0, 0);
if( !faceCascade ) {
printf("Couldnt load Face detector '%s'\n", faceCascadeFilename);
exit(1);
}
/ /获取下一帧的摄像头.
IplImage *inputImg = cvQueryFrame(camera);
/ /对输入图像执行面部检测，使用给定的（Haar）分类.
CvRect faceRect = detectFaceInImage(inputImg, faceCascade);
/ /确保有效侦测到脸部.
if (faceRect.width > 0) {
printf("Detected a face at (%d,%d)!\n", faceRect.x, faceRect.y);
}
.... Use 'faceRect' and 'inputImg' ....
//人脸检测程序完成时的资源cvReleaseHaarClassifierCascade（级联）;
```

二、如何进行预处理，人脸图像的人脸识别：

```
/ /无论是将图像转换为灰度，或使用现有的灰度图像.
IplImage *imageGrey;
if (imageSrc->nChannels == 3) {
imageGrey = cvCreateImage( cvGetSize(imageSrc), IPL_DEPTH_8U, 1 );
/ /从RGB（实际上它是BGR）转换为灰度。
cvCvtColor( imageSrc, imageGrey, CV_BGR2GRAY );
}
else {
/ /只使用输入图像，因为它已经是灰度.
imageGrey = imageSrc;
}
/ /调整的图像的大小是一致的大小，即使宽高比变化.
IplImage *imageProcessed;
imageProcessed = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 1);
/ /使图像具有固定大小.
/ / CV_INTER_CUBIC或CV_INTER_LINEAR的是良好的扩大，
/ //抽取伸缩CV_INTER_AREA是良好的，但糟糕的扩大.
cvResize(imageGrey, imageProcessed, CV_INTER_LINEAR);
/ /给一个标准的亮度和对比度的图像.
cvEqualizeHist(imageProcessed, imageProcessed);
..... Use 'imageProcessed' for Face Recognition ....
```

```
if (imageGrey)
cvReleaseImage(&imageGrey);
if (imageProcessed)
cvReleaseImage(&imageProcessed);
```

三、实现离线瞄准：

```
/ /告诉PCA退出的时候它有足够的特征脸.
CvTermCriteria calcLimit = cvTermCriteria( CV_TERMCRIT_ITER, nEigens, 1);
/ /计算平均图像，特征向量（特征脸）和特征（比率）.
cvCalcEigenObjects(nTrainFaces, (void*)faceImgArr, (void*)eigenVectArr,
CV_EIGOBJ_NO_CALLBACK, 0, 0, &calcLimit,
pAvgTrainImg, eigenValMat->data.fl);
//标准化矩阵的特征值.
cvNormalize(eigenValMat, eigenValMat, 1, 0, CV_L1, 0);
/ /每个training项目的PCA子空间上的图像.
CvMat projectedTrainFaceMat = cvCreateMat( nTrainFaces, nEigens, CV_32FC1 );
int offset = projectedTrainFaceMat->step / sizeof(float);
for(int i=0; i<nTrainFaces; i++) {
cvEigenDecomposite(faceImgArr, nEigens, eigenVectArr, 0, 0,
 pAvgTrainImg, projectedTrainFaceMat->data.fl + i*offset);
}
```

四、实现从相机的实时识别：

```
/ /获取下一个摄像机框架。等待，直到下一帧是准备好了，
/ /提供直接访问它，所以不要修改或释放返回的图像!
/ /将自动初始化摄像头，在第一帧上.
IplImage* getCameraFrame(CvCapture* &camera)
{
IplImage *frame;
int w, h;
/ /如果相机没有被初始化，然后打开它.
if (!camera) {
 printf("Acessing the camera ...\n");
 camera = cvCreateCameraCapture( 0 );
 if (!camera) {
 printf("Couldn't access the camera.\n");
 exit(1);
 }
 / /尝试设置相机的分辨率为320×240.
 cvSetCaptureProperty(camera, CV_CAP_PROP_FRAME_WIDTH, 320);
 cvSetCaptureProperty(camera, CV_CAP_PROP_FRAME_HEIGHT, 240);
 / /获取第一帧，以确保相机初始化.
 frame = cvQueryFrame( camera );
 if (frame) {
 w = frame->width;
 h = frame->height;
```

```
 printf("Got the camera at %dx%d resolution.\n", w, h);
 }
 / /等待一点点，使相机可以自动调节其亮度.
 Sleep(1000); / /（以毫秒为单位）
}
/ /等待下一个摄像机的帧准备就绪，然后抓住它.
frame = cvQueryFrame( camera );
if (!frame) {
 printf("Couldn't grab a camera frame.\n");
 exit(1);
}
return frame;
}
CvCapture* camera = 0; / /摄像头装置.
while ( cvWaitKey(10) != 27 ) { // 退出“ "Escape"的关键.
IplImage *frame = getCameraFrame(camera);
...
}
// /相机.
cvReleaseCapture( &camera );
```

应用程序 有 LINUX 环境 和 WIN7 环境的 WIN7 环境的建立一个 Projects 里面的文件要通过 Visual Studio + OpenCV 才能生成一个项目。

LINUX 环境安装要先搭好环境在进行移植。

同时还要根据提示调用不能的库和添加头文件。

IAR Embedded Workbench 环境下调试的项目文件 rdk-bldc.rar

相关下载：>> BeagleBone 外围电路设计心得 Opencv 无刷电机

1.3.19 BeagleBone Black 不一样的玩法：创意感应式乐器

Hackable Instruments 项目的目标是开发一个容易让玩家找到乐趣和新的方向的乐器，而不需要懂一些既专业又高深的知识，同时还能有自己独特的演奏风格。项目成员 Andrew McPherson 和 Victor Zappi 设计了一款简单的木制立方体，当用户触摸，滑动或轻击其含有感应器的某一面时，它就会随之而产生音乐。

去年，McPherson 在 Kickstarter 上展示了一个系统，通过在一个类似钢琴键的键盘上安上传感器，让演奏者得到连续的体验效果，而不需要其他的东西。触摸键针对的是那些知道弹钢琴的人，但是这款电子产品不需要人们懂钢琴或其他的弹奏技巧。

McPherson 说：“我们对如何利用技术来创造音乐十分感兴趣，这个立方体是个非常简单的数字音乐播放器，它只能发出几个音符，依靠它们来完成音色的变化。约束力对创造者而言是一种强大的动力——想想有人把勺子或搓衣板都可以做成乐器。通过制造一个非常受限制的工具，我们可以研究表演者作出的反应。”

这个带电池的 8 英寸木制立方体包括一个二维电容式触觉传感器，一个位于立方体下底部的压力传感器以及另一边的输出扬声器。里面则有一个带音频接口的 BeagleBone Black 开发板，用来运行 Linux 和一些 Zappi 自创的软件。

他们开发了这个立方体的两个版本。其中一个的软件通过压力制造音乐，另一个依靠传感器的触摸位置来产生音乐频率。两种版本都可以让演奏者在这个几乎空白的平台上探索新的，属于自己的演奏风格。

图 1-142

这个立方体已经有 10 名具有音乐背景的志愿者进行测试，包括古典音乐和现代电子音乐等各种风格。

“我们不建议任何‘正确’的演奏方式，盒子的设计和我们的其他介绍也是这样。”McPherson 说，“我们发现了许多令人吃惊的演奏方式，包括我们没有想到的技术，例如通过舔传感器产生残留水分的持续音。这些发现非常有趣，因为它证实了我们要进行测试的原则之一，即演奏者会产生与设计者意图不同的演奏方式。”

目前该产品还在进一步探索中，其团队计划公布开源数据，希望激发其他开发者的音乐灵感。

点击查看详情：>> 不一样的玩法：创意感应式乐器

1.3.20 用 VNC 在电脑上直接显示 BBB 的 framebuffer 图像

大家应该都会在 BBB 上用 VNC 了吧?（还不会的同学，方法是在 BBB 上运行

x11vnc -bg -o %HOME/.x11vnc.log.%VNCDISPLAY -auth /var/run/gdm/auth-for-gdm*/database -display :0 -forever

然后在电脑上打开 VNC 客户端，输入 192.168.7.2:0 就能连接并显示 LCD 上的图像了。）但是这个方法只能远程控制 X 桌面。如果我写了一个 QtE 程序，它直接跑在 framebuffer 上了，我也想在电脑上显示出来怎么办（比如给客户进行产品演示时会用到）？

我在网上找了半天解决办法，最后发现 x11vnc 虽然名字里带有 X11，但它其实是支持直接搬运 fb 的。虽然这个功能一直是 beta 版，但是我用着没什么问题。首先说明，我在 BBB 上安装了 480x272 分辨率的 4.3 寸 LCD 屏幕。在 BBB 上运行下面的命令就可以了

x11vnc -rawfb map:/dev/fb0@480x272x16

上面给出的是必备的参数，后面还可以根据喜好添加其他参数如 -bg 等。

还有一种方法

x11vnc -rawfb console

用这个命令的话还可以直接用电脑键盘在 BBB 的 console 里进行输入。

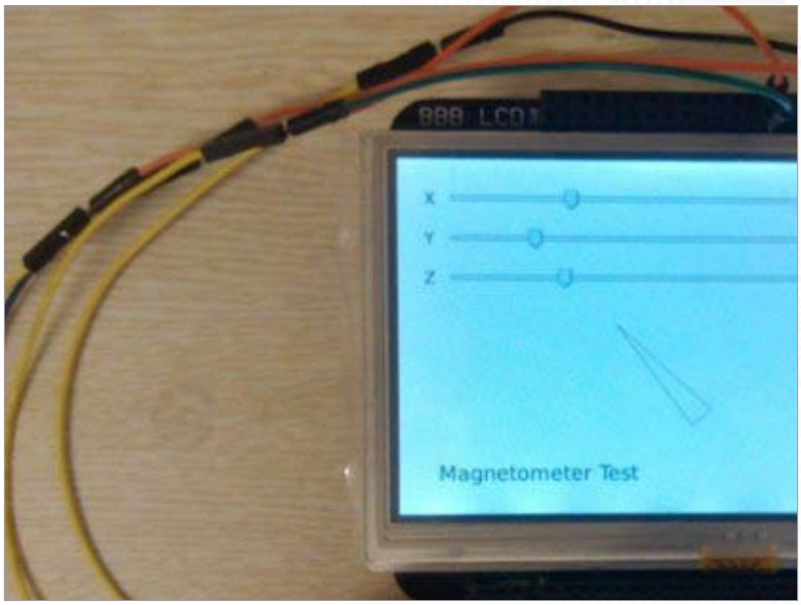
电脑端跟之前的一样，运行 VNC 客户端，输入 192.168.7.2:0 即可。

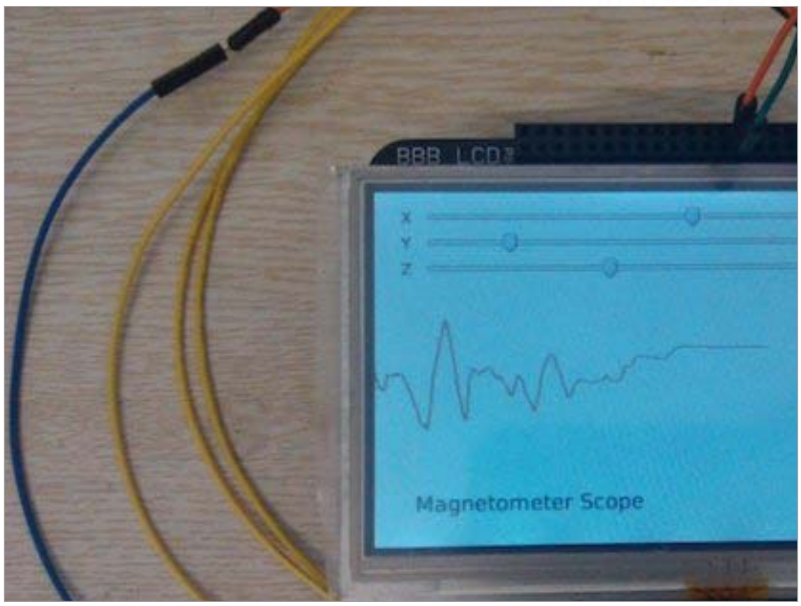
效果:刚连通的时候 client 画面没有反应，要等待四五秒钟才开始动态显示。实时性很好，而且基本不占用 CPU（6% 到 7%）。

注：关于 -rawfb 参数详见 http://www.karlrunge.com/x11vnc/faq.html 里的 Q-113

点击查看详情：>> 用 VNC 在电脑上直接显示 BBB 的 framebuffer 图像

1.3.21 基于 BB-Black 和 hmc5843 三轴电子罗盘





几次想照着课本系统地学习 Qt，但我发现还是有具体问题驱动时学习比较快。于是我给自己设定了这个任务：读取 HMC5843 的三轴磁场强度值，计算出角度，并把角度用直观形式显示在图形界面上。这里面涉及到一些问题，接下来就用问答的形式记录一下。

Q1: 搭建 Ubuntu-BBB 的 Qt 交叉编译环境、配置触摸屏

A1: http://blog.csdn.net/wyt2013/article/details/18549415

Q2: 去掉 Qt 界面的标题栏

A2: 在 mainwindow.cpp 中 MainWindow(QWidget *parent) 函数里的 ui->setupUI(this);下添加一行即可

```
this->setWindowFlags(Qt::FramelessWindowHint);
```

Q3: 去掉鼠标指针，保留触摸操作

A3: 首先 include <QWSServer>

然后在 mainwindow.cpp 中 MainWindow(QWidget *parent) 函数里的 ui->setupUI(this);下添加一行即可

```
QWSServer::setCursorVisible(false);
```

Q4: 在 Qt 中实现定时中断

A4: 使用 QTimer。

在 mainwindow.cpp 中构造函数 MainWindow(QWidget

*parent) 里的 ui->setupUI(this);下添加

```
QTimer *timer = new QTimer(this);
connect(timer,SIGNAL(timeout()),this,SLOT(timerUpDate()));
timer->start(T);
```

这样就设定好了一个周期为 T 毫秒的类似定时中断的东西。每过 T 毫秒就会触发一次 timerUpDate() 函数。

别忘了在 MainWindow 类中添加 timerUpDate() 函数声明：

```
private slots:
    void timerUpDate();
```

然后就是编写 timerUpDate() 函数体了。

注意如果函数体内做的事情耗时超过了定时周期 T，Qt 不会像单片机的定时中断那样挂掉，而是会尽可能完成每次任务，导致整体定时周期延长。

我这里就在这个函数中实现了定时读取 I2C 设备的数值，并更新各个 UI 元素。

Q5: 如何测试函数执行时间

A5: 使用 gettimeofday() 函数获取微秒为单位的时间。

```
#include<stdio.h>
#include<sys/time.h>
#include<unistd.h>

int main()
{
    struct  timeval start;
    struct  timeval end;

    unsigned  long diff;
```

```
    gettimeofday(&start,NULL);
    //Put things you want to test here.
    gettimeofday(&end,NULL);
    diff = 1000000 * (end.tv_sec-start.tv_sec)+ end.tv_usec-start.tv_usec;
    printf("It took %ld us.\n",diff);
    return 0;
}
```

Q6: 如何画直线图形

A6: 用 QPainter

首先引用头文件

```
#include <QPainter>
#include <QPointF>
```

在 .h 文件中添加声明

```
private:
    void paintEvent(QPaintEvent *);
```

然后在函数里实现绘图

```
void MainWindow::paintEvent(QPaintEvent *)
{
     QPainter painter(this);
     QPen pen;

     pen.setColor(Qt::darkRed);
     painter.setPen(pen);
     static const QPointF points[4] = {QPointF(-50, 0), QPointF(50, 10), QPointF(50,
-10),QPointF(-50, 0)};
     painter.drawPolyline(points, 4);
}
```

屏幕左上角是坐标 (0,0) 点。这里就绘制出了我第一个图片中所示的三角形。

Q7: 如何定时刷新图形

A7: 上面给出的是绘制静止图片的方法，想绘制动态图片的话只需要在前面提到的定时中断 timerUpDate() 函数中增加一句

```
update();
```

Q8: 如何旋转图形

A8: 可以直接旋转坐标系来实现。首先将坐标系原点平移到旋转中心点，然后旋转坐标系即可。在 paintEvent(QPaintEvent *) 中增加

```
painter.translate(200,160);
painter.rotate(dir);
```

Q9: 如何绘制波形图

A9: 使用 QPainterPath。

在 .h 文件中添加定义 QPainterPath path;

在 paintEvent 中使用如下三行：

```
this->path.lineTo(num,-dir/4);
painter.translate(300-num,180);
painter.drawPath(this->path);
```

这里 num 是一个每次定时中断都会自增的变量。所以 lineTo(num,-dir/4); 会绘制历史上所有的线，以及绘制最近一个端点和 (num,-dir/4) 这个坐标的连线。translate(300-num,180); 将坐标每次都向左平移一个单位。

Q10: 如何添加退出程序按钮

A10: 在图形界面拖放一个 PushButton，右键点击之，选择 "Go to slot"，然后选择第一个 clicked()，这时 Qt creator 会自动跳到 void MainWindow::on_pushButton_clicked() 函数中。在里面添加：

```
QApplication* app;
app->quit();
```

调试时建议用这个按钮退出程序，而非使用 Qt creator 上的停止按钮。因为有时停止按钮会失效，然后就只能重启 BBB 了。

完整的源代码放到了我的 github 里，欢迎参考

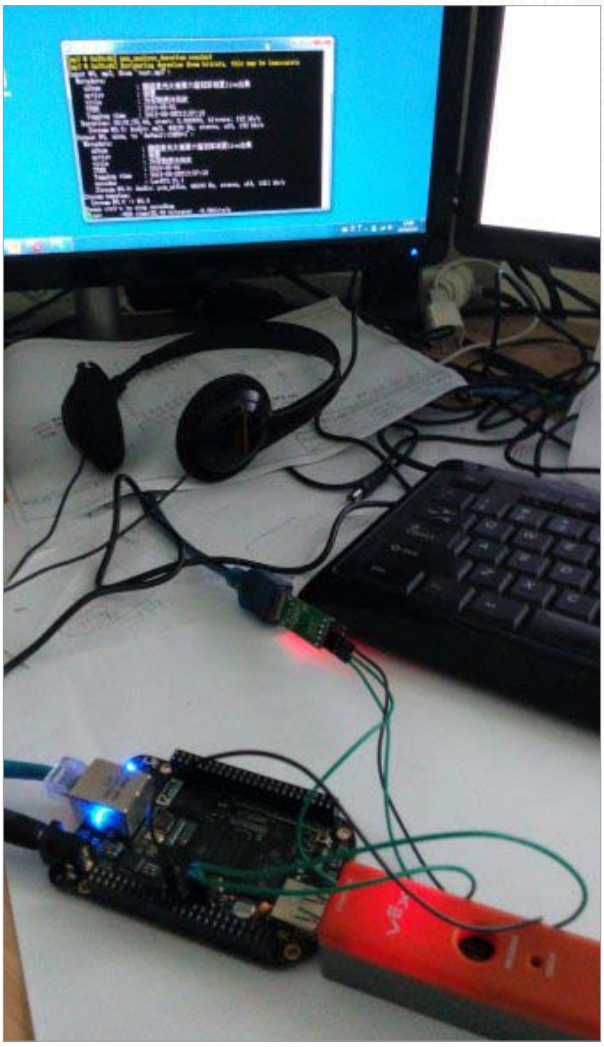
https://github.com/wytalfred/Beaglebone-magnetometer-Qt-display

https://github.com/wytalfred/beaglebone-mag-scope

点击查看详情：>> 基于 BBB 和 hmc5843 三轴电子罗盘

1.3.22 BeagleBone 之有生有色之实现无损音乐播放器

让我们实现在 BeagleBone Black 下播放音乐，包括 wav 无损音乐格式，闲暇之余也可以作为一个无损音乐播放器耍耍了。



一、原理：

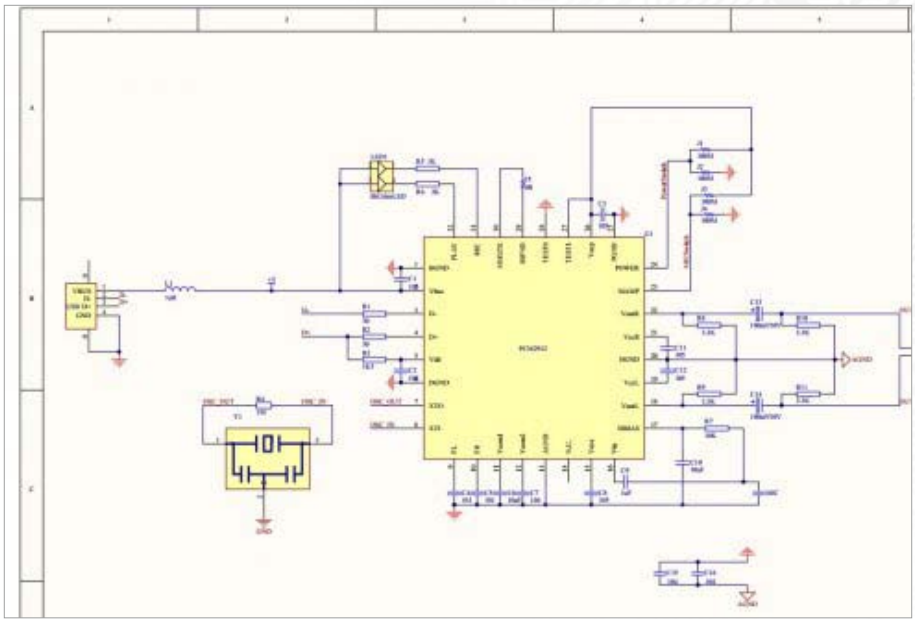
我们都知道，BeagleBone Black 板子上不带独立的音频输出，声音可以经由 hdmi 接口输出到外接设备上，但这样外接设备要求高，很麻烦。另一个办法是 AM3359 内部的 IIS 引脚，外接一个支持 IIS 接口的 DAC 芯片，这样可以通过软件实现各种范围的码率，据说能支持到 24bit/192kHz ，绝对的高端大气上档次的音质，如果后面的滤波和耳放能做好，可以匹敌当前几千的高端播放器，但是这个需要软硬件和系统支持，特别是扩展 CAPE 的绘制和耳放部分以及电源的设计，算是高难度的活了，这里也搁下不提。

两条路都被堵死了，那怎么办呢? 我把眼光瞄到了 usb 口上。

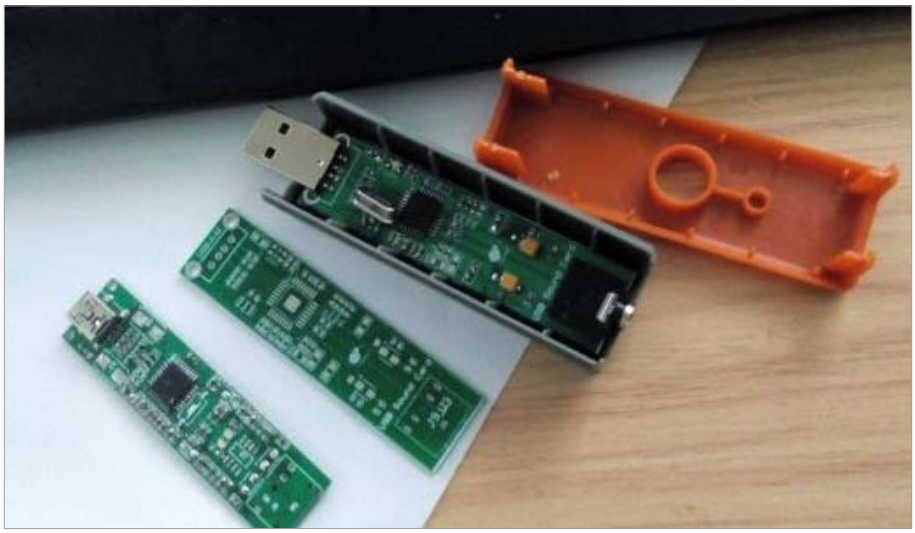
前一段时间在网上查资料 DIY 了个 usb 声卡，支持 16bit--44100K，本来准备开源发到坛子 DIY 区里的，上周有事一直没把资料整理好，这里先试试能用不，可以的话岂不一举两得。

二、usb 声卡

USB 声卡使用的 TI 的 PCM2912A 单芯片方案，外围简单，免驱动，带静音降噪，总之是各种 NB. 早早的在 TI 申请了几片，来回做了 2 次板子，终于成功，效果不错呢。
原理图见这里



板子见这里



第一块是第一次打板的，几个封装画错了。

音质效果不错，感觉比台式和笔记本的稍好些。

三、实现步骤

只有硬件不行，我们还需要在 BBB 里面安装一个解码器，我们选用的是大名鼎鼎的 ffmpeg，开源，强大。

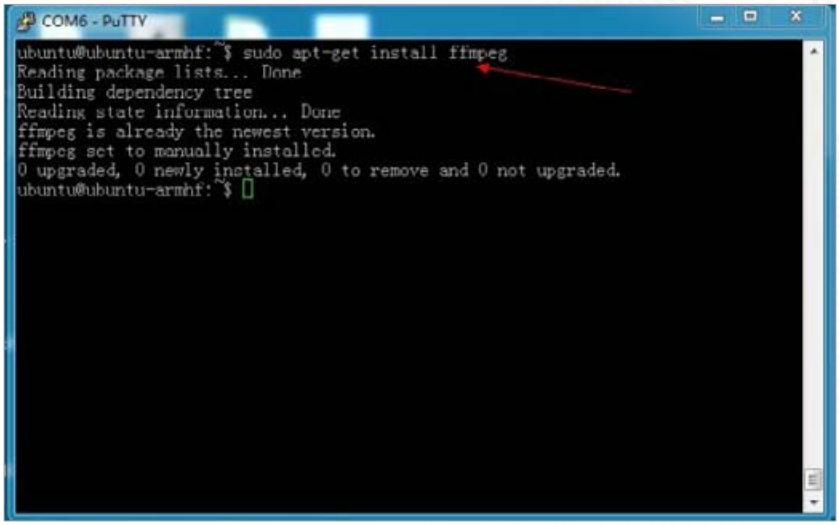
官方地址：http://ffmpeg.org/

下面是安装步骤

1、BeagleBone Black 里面解码器安装联网的前提下执行命令

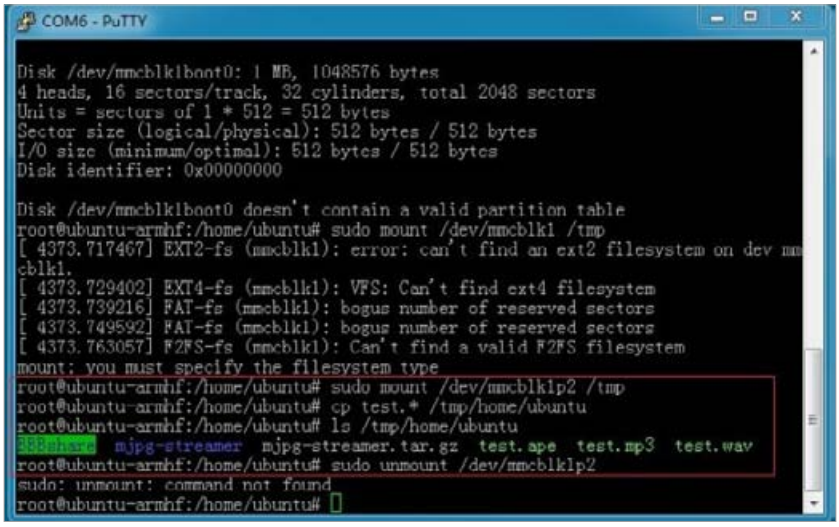
sudo apt-get install ffmpeg

就这么简单，这就是我喜欢 ubuntu 的原因。如果有问题注意一下权限。



安装提示完成后，执行下 ffmpeg 看看是否安装正确。

2、准备歌曲文件我是用 sd 卡拷贝的歌曲文件，然后在系统里面枚举，用 cp 命令拷贝到内部 emmc 里面去的。



3、确保 usb 声卡插入 usb 端口，执行以下命令查看

ls -ald /dev/dsp*

显示结果里面会比没有插入之前多出一个 /dev/dsp1 的设备，就是我们的声卡

这就说明声卡被正常识别了。

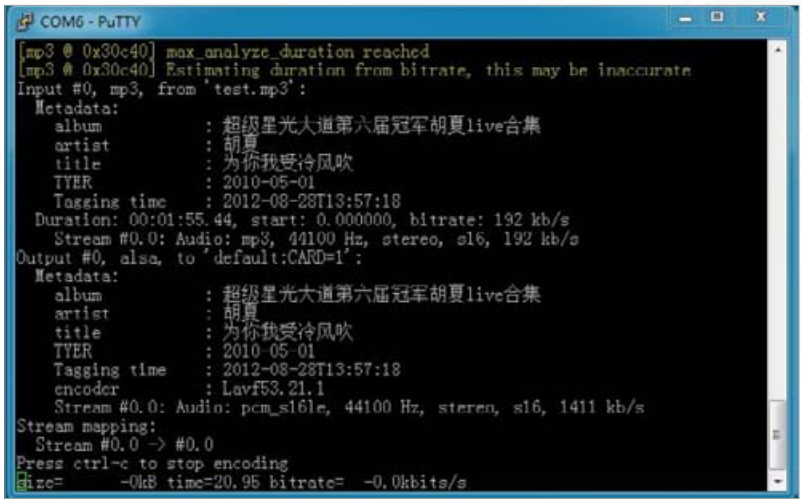
4、下面就是激动人心的时刻了

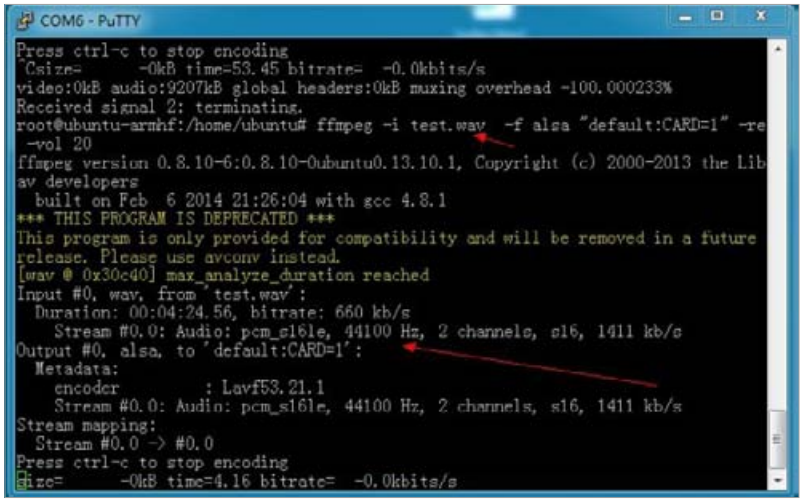
执行指令

```
sudo su
ffmpeg -i test.wav -f alsa "default:CARD=1" -re -vol 50
```

CARD=1就是我们的声卡。-vol 50 是音量，范围是0~255

test.mp3 文件名要更换为你的实际文件名，我尝试了 MP3 和 wav 可以播放，其它扩展名可能需要加载解码库插件，大家可以去尝试。





想终止当前播放可以用 CTRL+C 操作。

四、后记

本来这一篇应该是摄像头监控的实现的，由于摄像头还没借到，所以先上这个“有声”，下一篇我们就“有色”了。usb 声卡的资料最近会整理出来，预计周末前发到 diy 板块，大家分享一下。

视频 -- 声音是从耳机录出的，所以很小。

点击查看详情：>> BeagleBone 之有生有色之实现无损音乐播放器

1.3.23 BB-Black 的 FFTW 移植 – 使用 cycle_counter

要用 BBB 做 FFT，本来打算自己写，但是发现有 fftw 这么个好东西。fftw 可以自动优化代码，但是要使用 cycle_counter 寄存器来计算不同模式下的运算时间，cycle counter 属于特权寄存器但是在 linux 中用户模式不能直接访问，幸运的是可以在用户模式下可以对这一功能进行使能。。否则会使用 MEASURE 时执行 fft 时会提示非法指令，甚至在编译 fftw 后 make check 也会提示无法完成。

增加了cycle counter pitch 的网址连接

http://www.vesperix.com/arm/fftw-arm/source/index.html

配置源码 ./configure --enable-single --enable-neon --enable-armv7a-cycle-counter ARM_CPU_TYPE=cortex-a8 ARM_FLOAT_ABI=softfp

原文并没有给出用户模式下访问 cycle counter 的 pitch, 不知道是不是觉得太简单了便自己写了一个，实际上就是一条嵌入式汇编指令。将附件解压后修改 Makefile 中的 KER_DIR 修改成自己的内核源码路径，make 后 insmod cycle_counter.ko 即可。

指令的详细意思请参考 Arm cortex-A8 Technical Reference 中的 3.2.51 建议在 BBB 上编译驱动，交叉编译的话 insmod 时可能会因为内核版本不同提示 invaild format 之类的，不过好像也可以强制安装。

点击查看详情：BBB 的 FFTW 移植 – 使用 cycle_counter

1.3.24 BB-Black 入门基础之 OpenCV 的交叉编译

经过几天的不断尝试，终于顺利完成了OpenCV的交叉编译。哎，本来这就是一编译过程，硬是用了一周的时间。

现在把我的实现步骤贴出来，希望大家能少走弯路。

参考文献：

docs.opencv.org/doc/tutorials/introduction/

crosscompilation/arm_crosscompile_with_cmake.html

blog.csdn.net/msq19895070/article/details/24477575m.blog.csdn.net/blog/guo8113/17794395

blog.csdn.net/shuxiao9058/article/details/7525376

虚拟机环境：Ubuntu 12.04 LTS

编译器：arm-linux-gnueabihf-

BBB 系统：Ubuntu 12.04 armhf

OpenCV: 2.4.3

Cmake：2.8.7

OpenCV 交叉编译时用到的几个依赖库：

http://download.eeworld.com.cn/detail/lonerzf/538159

已编译的 BB Black 的 OpenCV 库：

http://download.eeworld.com.cn/detail/lonerzf/538163

一、安装依赖项

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
```

这几项与视频解码相关，不放心的话，可以先不装，之后再装也一样。只不过不装这几项呢，make 配置的结果会出现：

```
FFMPEG:                      NO
codec:                       NO
format:                      NO
util:                        NO
swscale:                     NO
```

结果就这么直观。

二、Cmake 的安装

sudo apt-get install cmake cmake-qt-gui

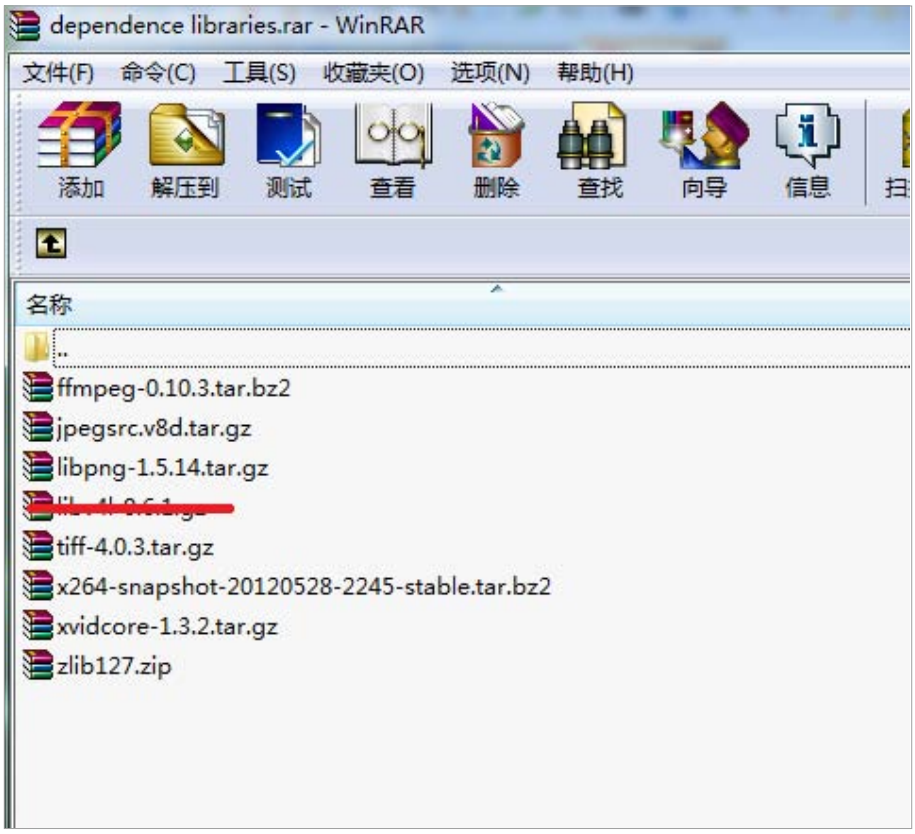
装好之后目前版本是 2.8.7 的。当然，也可以从官网下载源码 make。只是，别装 2.8.12 这个最新版，个别地方会报错。

三、相关库编译安装

OpenCV交叉编译时各个库之间依赖关系如下：

```
OpenCV
        |--------zlib
        |--------jpeg
        |--------libpng
                    |--------zlib
        |--------tiff
                    |--------zlib
        |--------ffmpeg
                    |--------x264
                    |--------xvidcore
```

考虑到个别库不太容易找到，这几个库的源码已经打包上传，可以在论坛的下载中心下载。内容如下：



1.png (60.87

为了简化操作，咱们这里用脚本的方式操作。

首先解压 OpenCV 到指定目录。我这里是：

```
/opt/lon/OpenCV-2.4.3/
```

添加环境变量

```
export COMPILE_PREFIX=arm-linux-gnueabihf-
export INSTALL_PREFIX=`pwd`/crosscompile_root
export DL_FOLDER=`pwd`/dl
export BUILD_FOLDER=`pwd`/buildpool
export CC=${COMPILE_PREFIX}gcc
```

然后新建crosscompile_root dl build buildpool 四个文件夹。如下图所示:

```
root@zhao-desktop:~/Desktop# cd /opt/lon/OpenCV-2.4.3/
root@zhao-desktop:/opt/lon/OpenCV-2.4.3# ls
3rdparty  build      CMakeLists.txt     dl       index.rst  README
android   buildpool  crosscompile_root  doc      ios        samples
apps      cmake      data               include  modules
root@zhao-desktop:/opt/lon/OpenCV-2.4.3#
```

接着将下载好的各压缩文件都复制到 dl 文件夹下。

同时新建一个 mycfg.sh 的脚本文件，并为其添加可执行权限

sudo chmod a+x mycfg.sh

```
root@zhao-desktop:/opt/lon/OpenCV-2.4.3# cd dl/
root@zhao-desktop:/opt/lon/OpenCV-2.4.3/dl# ls
ffmpeg-0.10.3.tar.bz2  tiff-4.0.3.tar.gz
jpegsrc.v8d.tar.gz     x264-snapshot-20120528-2245-stable.tar.bz2
libpng-1.5.14.tar.gz   xvidcore-1.3.2.tar.gz
mycfg.sh               zlib127.zip
root@zhao-desktop:/opt/lon/OpenCV-2.4.3/dl#
```

接着就是打开 mycfg.sh 并写入以下内容

```
#!/bin/bash
echo "Hello, zlib!"
pushd $BUILD_FOLDER
unzip $DL_FOLDER/zlib127.zip
pushd zlib-1.2.7
./configure --prefix=$INSTALL_PREFIX
make
make install
popd
echo "zlib end"

echo "Hello, jpeg!"
tar xf $DL_FOLDER/jpegsrc.v8d.tar.gz
pushd jpeg-8d
./configure --host=arm-linux-gnueabihf --prefix=$INSTALL_PREFIX
mkdir -p $INSTALL_PREFIX/bin
mkdir -p $INSTALL_PREFIX/man/man1
make
make install
popd
echo "jpeg end"
```

```
echo "Hello, png!"
tar xf $DL_FOLDER/libpng-1.5.14.tar.gz
pushd libpng-1.5.14
./configure --host=arm-linux-gnueabihf --prefix=$INSTALL_PREFIX CPPFLAGS=-
I$INSTALL_PREFIX/include LDFLAGS=-L$INSTALL_PREFIX/lib
make
make install
popd
echo "png end"
echo "Hello, x264!"
tar xjf $DL_FOLDER/x264-snapshot-20120528-2245-stable.tar.bz2
pushd x264-snapshot-20120528-2245-stable
./configure --host=arm-linux --cross-prefix=arm-linux-gnueabihf- --enable-shared
--prefix=$INSTALL_PREFIX
make
make install
popd
echo "x264 end"
echo "Hello, xvidcore!"
tar xf $DL_FOLDER/xvidcore-1.3.2.tar.gz
pushd xvidcore/build/generic
./configure --prefix=$INSTALL_PREFIX --host=arm-linux-gnueabihf  --disable-
assembly
make
make install
popd
echo "xvidcore end"

echo "Hello, tiff!"
tar xf $DL_FOLDER/tiff-4.0.3.tar.gz
pushd tiff-4.0.3
./configure --prefix=$INSTALL_PREFIX --host=arm-linux-gnueabihf --enable-shared
CPPFLAGS=-I$INSTALL_PREFIX/include LDFLAGS=-L$INSTALL_PREFIX/lib
make
make install
popd
echo "tiff end"

echo "Hello, ffmpeg!"
tar -xjvf $DL_FOLDER/ffmpeg-0.10.3.tar.bz2
pushd ffmpeg-0.10.3
./configure --prefix=$INSTALL_PREFIX --enable-shared --disable-static --enable-
gpl --enable-cross-compile --arch=arm --cpu=armv7-a --disable-stripping --target-
os=linux --enable-libx264 --enable-libxvid --cc=arm-linux-gnueabihf-gcc --enable-
swscale --extra-cflags=-I$INSTALL_PREFIX/include --extra-ldflags=-L$INSTALL_PREFIX/
lib --disable-asm
make
make install
popd
echo "ffmpeg end"
```

完了以后呢，不用多想，执行脚本吧。

```
./mycfg.sh
```

用了脚本之后这个过程也算半自动了吧，可以喝茶去了~

上一步完成了之后呢，进入上文新建的 build 文件夹，创建一个 toolchain.cmake 的文件，并填入如下内容：

```
set( CMAKE_SYSTEM_NAME Linux )
set( CMAKE_SYSTEM_PROCESSOR arm )
set( CMAKE_C_COMPILER arm-linux-gnueabihf-gcc )
set( CMAKE_CXX_COMPILER arm-linux-gnueabihf-g++ )
###########user defined#############
set( CMAKE_FIND_ROOT_PATH "/opt/lon/OpenCV-2.4.3/crosscompile_root" )
set( CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER )
set( CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY )
set( CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY )
#####################################
```

好，cmake 吧。

```
cmake -DCMAKE_TOOLCHAIN_FILE=toolchain.cmake ../
```

然后再运行如下指令

```
cmake-gui
```

减少一些没有用的项，如 gtk、cude、1394 等。其中 CUDE，当时就困扰了我好一会儿呢。看人家怎么解释的：

```
"CUDA_TOOLKIT_ROOT_DIR not found or specified", CUDA is a parallel computing
platform and programming model that enables dramatic increases in computing
performance by harnessing the power of the graphics processing unit (GPU). It is
only developed for NVdia GPU. However, it seems that not all PC will be equipped
with that kind of GPU, theoretically, it is not necessary to install it and we
make the option WITH_CUDA OFF. Also the opencv only support CUDA 4.0 now from the
post bellow.
```

确认之后 Config 并 General 生成配置信息。

下面附上我的 cmake 配置信息：

```
Detected version of GNU GCC: 113 (113)
checking for module 'gstreamer-base-0.10'
  package 'gstreamer-base-0.10' not found
checking for module 'libv4l1'
  package 'libv4l1' not found
Looking for linux/videodev.h
Looking for linux/videodev.h - not found
Looking for linux/videodev2.h
Looking for linux/videodev2.h - found
```

```
Looking for libavformat/avformat.h
Looking for libavformat/avformat.h - not found
Looking for ffmpeg/avformat.h
Looking for ffmpeg/avformat.h - not found
Could NOT find PythonLibs (missing:  PYTHON_LIBRARIES PYTHON_INCLUDE_DIRS)
(Required is at least version "2.7.3")
General configuration for OpenCV 2.4.3 =====================================
  Platform:
    Host:                        Linux 3.2.0-61-generic-pae i686
    Target:                      Linux arm
    CMake:                       2.8.7
    CMake generator:             Unix Makefiles
    CMake build tool:            /usr/bin/make
    Configuration:               Release
  C/C++:
    Built as dynamic libs?:      YES
    C++ Compiler:                /opt/lon/gcc-linaro-arm-linux-
gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-g++  (ver 1.13.1)
    C++ flags (Release):         -W -Wall -Werror=return-type -Werror=address
-Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations
-Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -Wno-narrowing -Wno-
delete-non-virtual-dtor -fdiagnostics-show-option -pthread -fomit-frame-pointer
-ffunction-sections -O3 -DNDEBUG  -DNDEBUG
    C++ flags (Debug):           -W -Wall -Werror=return-type -Werror=address
-Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations
-Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -Wno-narrowing -Wno-
delete-non-virtual-dtor -fdiagnostics-show-option -pthread -fomit-frame-pointer
-ffunction-sections -g  -O0 -DDEBUG -D_DEBUG -ggdb3
    C Compiler:                  /opt/lon/gcc-linaro-arm-linux-
gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-gcc
    C flags (Release):           -W -Wall -Werror=return-type -Werror=address
-Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations
-Wmissing-prototypes -Wstrict-prototypes -Wundef -Winit-self -Wpointer-arith
-Wshadow -Wno-narrowing -fdiagnostics-show-option -pthread -fomit-frame-pointer
-ffunction-sections -O3 -DNDEBUG  -DNDEBUG
    C flags (Debug):             -W -Wall -Werror=return-type -Werror=address
-Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations
-Wmissing-prototypes -Wstrict-prototypes -Wundef -Winit-self -Wpointer-arith
-Wshadow -Wno-narrowing -fdiagnostics-show-option -pthread -fomit-frame-pointer
-ffunction-sections -g  -O0 -DDEBUG -D_DEBUG -ggdb3
    Linker flags (Release):
    Linker flags (Debug):
    Precompiled headers:         YES
  OpenCV modules:
    To be built:                 core imgproc flann highgui features2d calib3d ml
video objdetect contrib nonfree photo legacy gpu stitching ts videostab
    Disabled:                    world
    Disabled by dependency:      -
```

```
    Unavailable:                 androidcamera java ocl python

  GUI:
    QT 4.x:                      NO
    GTK+ 2.x:                    NO
    GThread :                    NO
    GtkGlExt:                    NO
    OpenGL support:              NO

  Media I/O:
    ZLib:                        /opt/lon/OpenCV-2.4.3/crosscompile_root/lib/libz.
so (ver 1.2.7)
    JPEG:                        /opt/lon/OpenCV-2.4.3/crosscompile_root/lib/
libjpeg.so (ver 80)
    PNG:                         /opt/lon/OpenCV-2.4.3/crosscompile_root/lib/
libpng.so (ver 1.5.14)
    TIFF:                        /opt/lon/OpenCV-2.4.3/crosscompile_root/lib/
libtiff.so (ver 42 - 4.0.3)
    JPEG 2000:                   NO
    OpenEXR:                     build (ver 1.7.1)

  Video I/O:
    DC1394 1.x:                  NO
    DC1394 2.x:                  NO
    FFMPEG:                      YES
      codec:                     YES (ver 53.35.0)
      format:                    YES (ver 53.21.1)
      util:                      YES (ver 51.22.2)
      swscale:                   YES (ver 2.1.0)
      gentoo-style:              YES
    GStreamer:                   NO
    OpenNI:                      NO
    OpenNI PrimeSensor Modules:  NO
    PvAPI:                       NO
    GigEVisionSDK:               NO
    UniCap:                      NO
    UniCap ucil:                 NO
    V4L/V4L2:                    NO/YES
    XIMEA:                       NO
    Xine:                        NO
  Other third-party libraries:
    Use TBB:                     NO
    Use Cuda:                    NO
    Use OpenCL:                  NO
    Use Eigen:                   NO

  Python:
    Interpreter:                 /usr/bin/python (ver 2.7.3)
  Tests and samples:
    Tests:                       YES
    Performance tests:           YES
```

```
    Examples:                       NO

  Install path:                     /opt/lon/OpenCV-2.4.3/build/install

  cvconfig.h is in:                /opt/lon/OpenCV-2.4.3/build
-----------------------------------------------------------------
Configuring done
```

当然，更细节的内容你也可以在 CMakeCache.txt 文件中看到。

哦，差点忘了，还要在 CMakeCache.txt 文件中添加点东西：

（1）

```
CMAKE_CXX_FLAGS:STRING=
```

这一行后面添加

```
-mfpu=neon -I/opt/lon/OpenCV-2.4.3/crosscompile_root/include -L/opt/
lon/OpenCV-2.4.3/crosscompile_root/lib -L/opt/lon/gcc-linaro-arm-linux-
gnueabihf-4.8-2013.10_linux,-Wl,-rpath,/lib
```

（2）

```
CMAKE_EXE_LINKER_FLAGS_DEBUG:STRING=
```

这一行后面添加

```
-lpthread -lrt
```

好。下面就是 make & make install 完成操作了。

至此，应该就能完成 OpenCV 的交叉编译了。然后将

/opt/lon/OpenCV-2.4.3/build/install/opt/lon/OpenCV-2.4.3/crosscompile_root 两个文件夹下的内容合并，放到咱们的 BB Black 中，应该就能工作了。

如果编译失败，可考虑重装虚拟机。并且避免安装其他东西。这是阻碍了我很长时间的一个重要原因。希望能乡亲们带来一点点的提醒。

点击查看详情：>>BB Black 入门基础之 OpenCV 的交叉编译

1.4 EEworld 论坛网友对 BB-Black 相关性能测评

1.4.1 BeagleBone IO 速率测试

1. 一路走来，熟悉硬件系统，搭建软件开发环境，编译 Linux 系统等等，现在也该到对硬件做一些事情了，这是我这几天的研究心得，与君共享。1. GPIO 的 char 型驱动，这里主要就是点个灯，感受一下驱动的设计和硬件的控制驱动程序：

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/leds.h>
#include <linux/io.h>
#include <linux/semaphore.h>
#include <linux/kernel.h>
#include <linux/cdev.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <mach/gpio.h>
#include <plat/mux.h>
#include <linux/gpio.h>
/*********************************************/
#define NAME "leds"
#define GPIO_TO_PIN(bank, gpio) (32 * (bank) + (gpio))

static int major =251;//定义主设备号
/*********************************************/
void led_on(void)
{
gpio_set_value(GPIO_TO_PIN(1,22), 1);
}
void led_off(void)
{
gpio_set_value(GPIO_TO_PIN(1,22), 0);
}

void led_init(void)
{
int result;
/* Allocating GPIOs and setting direction */
result = gpio_request(GPIO_TO_PIN(1,22), "Leds");//usr1
if (result != 0)
printk("gpio_request(1_22) failed!\n");
result = gpio_direction_output(GPIO_TO_PIN(1,22), 1);
if (result != 0)
printk("gpio_direction(1_22) failed!\n");

}

struct light_dev
{
```

```
struct cdev cdev;
unsigned char value;
};

struct light_dev *light_devp;

MODULE_AUTHOR("chenzhufly");
MODULE_LICENSE("Dual BSD/GPL");

// 打开和关闭函数
int light_open(struct inode *inode,struct file *filp)
{
struct light_dev *dev;

// 获得设备结构体指针
dev = container_of(inode->i_cdev,struct light_dev,cdev);
// 让设备结构体作为设备的私有信息
filp->private_data = dev;

return 0;
}

int light_release(struct inode *inode,struct file *filp)
{
return 0;
}


// ioctl
int light_ioctl(struct file *filp,unsigned int cmd,
unsigned long arg)
{
struct light_dev *dev = filp->private_data;

switch(cmd)
{
case 0:
dev->value = 0;
led_off();
break;

case 1:
dev->value = 1;
led_on();
break;

default:

return -ENOTTY;
// break;
}
```

```
return 0;
}

struct file_operations light_fops =
{
.owner = THIS_MODULE,
.unlocked_ioctl = light_ioctl,
.open = light_open,
.release = light_release,
};
// 模块加载函数
int light_init(void)
{
int ret;
led_init();
printk(KERN_ALERT "led modules is install\n");
ret=register_chrdev(major,NAME,&light_fops);
if(ret<0)
{
printk("unable to register myled driver!\n");
return ret;
}
return 0;

}

// 模块卸载函数
void light_cleanup(void)
{
unregister_chrdev(major,NAME);
printk("Goodbye,cruel world!\n");
}

module_init(light_init);
module_exit(light_cleanup);
```

应用程序：

```
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/input.h>
#include <unistd.h>
#include <sys/ioctl.h>

int main(int argc, char * argv)
{
int i, n, fd;

fd = open("/dev/leds", O_RDWR);
```

```
if (fd < 0)
{
printf("can't open /dev/leds!\n");
exit(1);
}

while (1) {
ioctl(fd, 1, 1);
sleep(1);

ioctl(fd, 0, 1);
sleep(1);
}

close(fd);

return 0;
}
```

Makefile 文件：

```
ARCH=arm
CROSS_COMPILE=/home/chenzhufly/BeagleBone/linux-devkit/bin/arm-arago-linux-
gnueabi-
obj-m := leds.o
KDIR := /home/chenzhufly/BeagleBone/board-support/linux-3.1.0-psp04.06.00.03.sdk
PWD := $(shell pwd)
default:
make -C $(KDIR) M=$(PWD) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) modules
app: leds_test.c
$(CROSS_COMPILE)gcc -o leds_test leds_test.c
clean:
$(MAKE) -C $(KDIR) M=$(PWD) clean
```

leds.sh 脚本

```
insmod leds.ko
mknod /dev/leds c 251 0
./leds_test
```

2. 使用 echo 命令，这个我在前面也说过

点亮 usr1

```
root@BeagleBone:~# echo 1 >
/sys/class/leds/BeagleBone::usr1/brightness
```

关闭 usr1

```
root@BeagleBone:~# echo 0 > /sys/class/leds/BeagleBone::usr1/brightness
```

3. 做个应用程序实现流水灯功能吧

```
#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/BeagleBone::usr1/brightness" // usr1 led
#define LED2 "/sys/class/leds/BeagleBone::usr2/brightness" // usr2 led
#define LED3 "/sys/class/leds/BeagleBone::usr3/brightness" // usr3 led

int main(void)
{
int f_led1 = open(LED1, O_RDWR);
int f_led2 = open(LED2, O_RDWR);
int f_led3 = open(LED3, O_RDWR);

unsigned char dat1, dat2, dat3;
unsigned char i = 0;

if (f_led1 < 0)
{
printf("error in open %s",LED1);
return -1;
}
if (f_led2 < 0)
{
printf("error in open %s",LED2);
return -1;
}
if (f_led3 < 0)
{
printf("error in open %s",LED3);
return -1;
}
//add 10 times
for(i=1; i<30; i++)
{
dat1 = ((i%3) == 1) ? '1' : '0';
dat2 = ((i%3) == 2) ? '1' : '0';
dat3 = ((i%3) == 0) ? '1' : '0';
write(f_led1, &dat1, sizeof(dat1));
write(f_led2, &dat2, sizeof(dat2));
write(f_led3, &dat3, sizeof(dat3));
usleep(300000);
}
// all the bright
```

```
{
dat1 = '1';
dat2 = '1';
dat3 = '1';
write(f_led1, &dat1, sizeof(dat1));
write(f_led2, &dat2, sizeof(dat2));
write(f_led3, &dat3, sizeof(dat3));
}
}
```

有兴趣的可以试试，等我找个示波器测试一下 IO 的速率，再贴个图上来。

点击查看详情：>>Linux下 AM335X 的 GPIO 控制

1.4.2 基于 BeagleBone 的内存性能测试

曾经有人说你如何保证测试的正确性，这个确实很难以回答。所以我会采用一些开放的工具，得到一些大家在同等条件下得到的一些测试结果，供大家参考。

先来看看我采用的工具：memxfer: a simple benchmark for memory transfer

网站：http://www.jjj.de/memxfer/memxferpage.html

memxfer 简介，英文的就不翻译了，做这行不懂英文就可以不干了，哈哈

With memxfer you can compare the memory transfer performance between

1. copying techniques like the strcpy vs. copying integers or doubles.

2. machines (in case you can access them)

3. BIOS settings

4. copying with or without unrolling and prefetching

采用 TI 自带的编辑器，修改 makefile

```
#OPTS= -DATHLON -W -Wall -O2 -ffast-math -fomit-frame-pointer -fno-exceptions
OPTS= -W -Wall -O2 -ffast-math -fomit-frame-pointer -fno-exceptions
CROSS_COMPILE=/home/chenzhufly/BeagleBone/linux-devkit/bin/arm-arago-linux-gnueabi-
gcc
.PHONY: memxfer
memxfer:
$(CROSS_COMPILE) $(OPTS) memxferjj.cc -o memxferjj

#NICE=nice -n -20

.PHONY: run
run:
for m in $$(seq 0 12); do $(NICE) ./memxferjj -s 32M 2 $$m ; done
for m in $$(seq 0 12); do $(NICE) ./memxferjj -s 32k 2000 $$m ; done

SZS= 1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192
.PHONY: srun
srun:
for s in $(SZS); do $(NICE) ./memxferjj -s $${s}k 100 6 ; done
for s in $(SZS); do $(NICE) ./memxferjj -s $${s}k 100 10 ; done
```

执行后的结果：还是挺强大的

```
root@BeagleBone:/home/sd# ./memxferjj -s 1m 10 0 1 2 3 4 5 6 7 8
avg: 1048576 [ 0]"memcpy" 733.465 MB/s
avg: 1048576 [ 1]"char *" 112.502 MB/s
avg: 1048576 [ 2]"short *" 215.318 MB/s
avg: 1048576 [ 3]"int *" 388.395 MB/s
avg: 1048576 [ 4]"long *" 386.832 MB/s
avg: 1048576 [ 5]"long * (4x unrolled)" 549.209 MB/s
avg: 1048576 [ 6]"int64 *" 512.613 MB/s
avg: 1048576 [ 7]"double *" 512.663 MB/s
avg: 1048576 [ 8]"double * (4x unrolled)" 512.660 MB/s
```

再看看我笔记本电脑执行的结果，ubuntu 系统，cpu 是 I5

```
root@chenzhufly-desktop:/mnt/nfs/test/memxfer_cpu# ./memxferjj -s 1m 10 0 1 2 3 4
5 6 7 8
avg: 1048576 [ 0]"memcpy" 4151.100 MB/s
avg: 1048576 [ 1]"char *" 755.059 MB/s
avg: 1048576 [ 2]"short *" 810.110 MB/s
avg: 1048576 [ 3]"int *" 548.155 MB/s
avg: 1048576 [ 4]"long *" 3070.310 MB/s
avg: 1048576 [ 5]"long * (4x unrolled)" 4370.629 MB/s
avg: 1048576 [ 6]"int64 *" 699.056 MB/s
avg: 1048576 [ 7]"double *" 4130.524 MB/s
avg: 1048576 [ 8]"double * (4x unrolled)" 4100.041 MB/s
```

可能这些结果还不够直观，我们来看看网上其他平台的一些测试结果把，这样就可以看出来 BeagleBone 还是很强大的。

```
/MOXA-IA240
//CPU ARM920 192MHZ

root@Moxa:/home# ./memxferjj-art -s 1m 10 0 1 2 3 4 5 6 7 8
avg: 1048576 [ 0]"memcpy" 10.125 MB/s
avg: 1048576 [ 1]"char *" 7.556 MB/s
avg: 1048576 [ 2]"short *" 8.909 MB/s
avg: 1048576 [ 3]"int *" 10.138 MB/s
avg: 1048576 [ 4]"long *" 10.117 MB/s
avg: 1048576 [ 5]"long * (4x unrolled)" 10.130 MB/s
avg: 1048576 [ 6]"int64 *" 10.081 MB/s
avg: 1048576 [ 7]"double *" 10.150 MB/s
avg: 1048576 [ 8]"double * (4x unrolled)" 10.075 MB/s

-----------------------------------------------------------------------
moxa_DA662
CPU:arm11(intel XScale IXP425 533MHZ)

root@Moxa:/home# ./memxferjj-da662 -s 1m 10 0 1 2 3 4 5 6 7 8
avg: 1048576 [ 0]"memcpy" 49.951 MB/s
avg: 1048576 [ 1]"char *" 16.218 MB/s
avg: 1048576 [ 2]"short *" 29.268 MB/s
avg: 1048576 [ 3]"int *" 48.763 MB/s
```

```
avg: 1048576 [ 4]"long *" 48.798 MB/s
avg: 1048576 [ 5]"long * (4x unrolled)" 49.549 MB/s
avg: 1048576 [ 6]"int64 *" 49.140 MB/s
avg: 1048576 [ 7]"double *" 49.128 MB/s
avg: 1048576 [ 8]"double * (4x unrolled)" 49.121 MB/s

------------------------------------------------------------------
ATOP
CPU:IDT MIPS 266MHZ

# ./memxferjj-mips -s 1m 10 0 1 2 3 4 5 6 7 8
avg: 1048576 [ 0]"memcpy" 166.667 MB/s
avg: 1048576 [ 1]"char *" 35.714 MB/s
avg: 1048576 [ 2]"short *" 66.667 MB/s
avg: 1048576 [ 3]"int *" 142.857 MB/s
avg: 1048576 [ 4]"long *" 142.857 MB/s
avg: 1048576 [ 5]"long * (4x unrolled)" 142.857 MB/s
avg: 1048576 [ 6]"int64 *" 166.667 MB/s
avg: 1048576 [ 7]"double *" 0.539 MB/s
avg: 1048576 [ 8]"double * (4x unrolled)" 0.540 MB/s

------------------------------------------------------------------
ATOP
CPU:Freescale MPC5200B 400M主频

# ./memxferjj-ppc -s 10m 10 0 1 2 3 4 5 6 7 8
avg: 10485760 [ 0]"memcpy" 132.615 MB/s
avg: 10485760 [ 1]"char *" 58.949 MB/s
avg: 10485760 [ 2]"short *" 88.168 MB/s
avg: 10485760 [ 3]"int *" 116.789 MB/s
avg: 10485760 [ 4]"long *" 116.770 MB/s
avg: 10485760 [ 5]"long * (4x unrolled)" 132.567 MB/s
avg: 10485760 [ 6]"int64 *" 132.448 MB/s
avg: 10485760 [ 7]"double *" 132.578 MB/s
avg: 10485760 [ 8]"double * (4x unrolled)" 132.106 MB/s
------------------------------------------------------------------

成都中嵌科技
CPU:ATMEL9200 180M主频
[root@(none) test]$./memxferjj-AT9200 -s 1m 10 0 1 2 3 4 5 6 7 8
avg: 1048576 [ 0]"memcpy" 52.174 MB/s
avg: 1048576 [ 1]"char *" 18.427 MB/s
avg: 1048576 [ 2]"short *" 28.416 MB/s
avg: 1048576 [ 3]"int *" 42.938 MB/s
avg: 1048576 [ 4]"long *" 42.937 MB/s
avg: 1048576 [ 5]"long * (4x unrolled)" 44.260 MB/s
avg: 1048576 [ 6]"int64 *" 47.847 MB/s
avg: 1048576 [ 7]"double *" 47.851 MB/s
avg: 1048576 [ 8]"double * (4x unrolled)" 45.380 MB/s
```

点击查看详情：>> 基于 BeagleBone 的内存性能测试

1.4.3 使用 iozone 测试 BeagleBone 文件系统性能

1. iozone 简介：

iozone（www.iozone.org）是一个文件系统的 benchmark 工具，可以测试不同的操作系统中文件系统的读写性能。可以测试

```
Read
re-read
Write
re-write
read backwards
read strided
fread
fwritte
random read
pread
mmap
aio_read
aio_write
```

等等不同的模式下的硬盘的性能 .

2. iozone 移植：

下载地址：http://www.iozone.org/src/current/ iozone3_263.tar（可以下载最新版本）

解压文件，在文件目录下执行

```
make CC=$(TOOL_CHAIN_PREFIX)-gcc GCC=$(TOOL_CHAIN_PREFIX)-gcc linux-arm
```

其中 TOOL_CHAIN_PREFIX 为我们安装的交叉编译环境的地址

生成可执行文件 iozone。

3. iozone 常用的几个参数：

-a 全面测试，比如块大小它会自动加

-i N 用来选择测试项，比如 Read/Write/Random 比较常用的是 0 1 2, 可以指定成 -i 0 -i 1 -i2. 这些别的详细内容请查 man

```
0=write/rewrite
 1=read/re-read
 2=random-read/write
 3=Read-backwards
 4=Re-write-record
 5=stride-read
 6=fwrite/re-fwrite
 7=fread/Re-fread
```

```
8=random mix
9=pwrite/Re-pwrite
10=pread/Re-pread
11=pwritev/Re-pwritev
12=preadv/Re-preadv
```

-r block size 指定一次写入 / 读出的块大小

-s file size 指定测试文件的大小

-f filename 指定测试文件的名字 , 完成后会自动删除 (这个文件必须指定你要测试的那个硬盘中)

-F file1 file2... 指定多线程下测试的文件名

批量测试项 :

-g -n 指定测试文件大小范围 , 最大测试文件为 4G, 可以这样写 -g 4G

-y -q 指定测试块的大小范围

输出 :

下面是几个日志记录的参数 . 好象要输出成图象进行分析，需要指定 -a 的测试才能输出

-R 产生 Excel 到标准输出

-b 指定输出到指定文件上 . 比如 -Rb ttt.xls

4. 我的测试结果

```
root@BeagleBone:/home/sd# iozone -M -a -s 4m
Iozone: Performance Test of File I/O
Version $Revision: 3.263 $
Compiled for 32 bit mode.
Build: linux-arm
Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
Al Slater, Scott Rhine, Mike Wisner, Ken Goss
Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
Randy Dunlap, Mark Montague, Dan Million,
Jean-Marc Zucconi, Jeff Blomberg,
Erik Habbinga, Kris Strecker, Walter Wong.
Run began: Mon Apr 30 10:24:28 2012
Machine = Linux BeagleBone 3.2.14 #1 Mon Apr 9 12:21:19 CEST 2012 armv7l GNU/L Auto
Mode
Auto Mode
File size set to 4096 KB
Command line used: iozone -M -a -s 4m
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
random random bkwd record stride
KB reclen write rewrite read reread read write read rewrite read fwrite frewrite
fread freread
4096 4 28772 195281 211231 213643 188330 193043 198846 257011 191151 55104 164875
201455 208236
```

```
4096 8 67083 209966 213377 216137 204216 222209 207496 298019 204413 67543 179674
210169 212659
4096 16 70444 216502 219613 222232 218312 237656 217985 294742 215036 69163 180256
218116 221071
4096 32 71279 211744 225413 228815 225935 239028 224093 309035 224809 66924 172476
226271 228596
4096 64 69808 204903 231096 233737 230813 235334 228316 292235 230047 65357 161749
231282 233975
4096 128 67448 192091 230708 232834 232821 218535 228739 272718 230423 60393 139609
231713 233429
4096 256 63523 165227 231725 230010 227061 181310 230773 191464 311529 57387 121341
232399 240982
4096 512 62549 149162 238818 238361 231832 164836 238599 138271 231214 58060 114528
222292 226310
4096 1024 62335 152664 239084 241169 242322 162921 238971 119246 237186 63148 122771
232635 232635
4096 2048 61352 146521 235877 229659 235997 161513 228277 88869 234058 81337 151474
235783 237104
4096 4096 60231 145346 226659 220915 225647 151996 227103 59813 220795 62483 154251
245140 247029
iozone test complete.
```

附图：

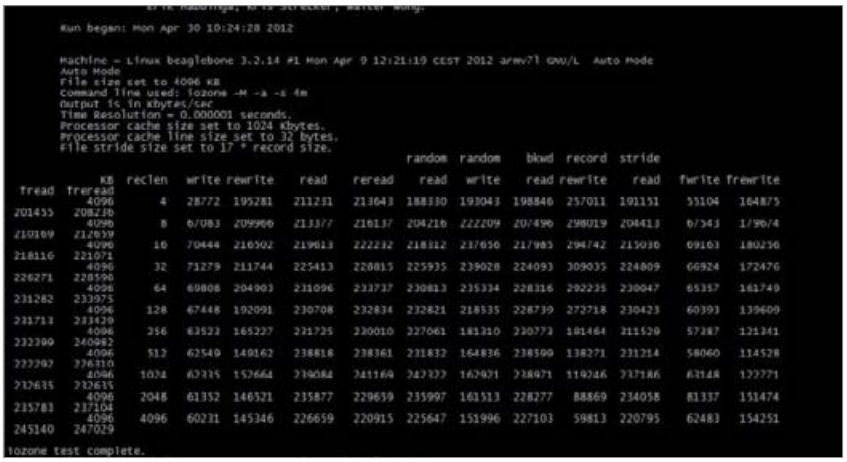


图 1-142

5. 编译后的 iozone 可执行文件：

大家可以尝试下，呵呵

附件下载：http://bbs.eeworld.com.cn/forum.php?mod=attachment&aid=OTAwMjN8Y2E4ZWRiMDV8MTM5MzMxNjcyOXw1MjgzNzB8MzMzNzYy

更深入细致的分析，还希望大家的参与，这里只能抛砖引玉啦!

点击查看详情：>> 使用 iozone 测试 BeagleBone 文件系统性能

1.4.4 基于 Iperf 的 BeagleBone 网络性能测试

一 . Iperf 简介 :

Iperf 是一个网络性能测试工具。可以测试 TCP 和 UDP 带宽质量，可以测量最大 TCP 带宽，具有多种参数和 UDP 特性，可以报告带宽，延迟抖动和数据包丢失。Iperf 在 linux 和 windows 平台均有二进制版本供自由使用。

Linux 版本下载地址 :
http://sourceforge.net/projects/iperf/ 目前版本时 iperf-2.0.5.tar.gz

Windows 版本下载地址 :
http://www.noc.ucf.edu/Tools/Iperf/

二 . Iperf 使用方法与参数说明 :

参数说明 :

-s 以 server 模式启动。#iperf -s

-c host 以 client 模式启动。host 是 server 端地址。#iperf -c serverip 通用参数 :

-f [kmKM] 分别表示以 Kbits, Mbits, KBytes, MBytes 显示报告，默认以 Mbits 为单位 ,#iperf -c 222.35.11.23 -f K

-i sec 以秒为单位显示报告间隔，#iperf -c 222.35.11.23 -i 2

-l 缓冲区大小，默认是 8KB,#iperf -c 222.35.11.23 -l 16

-m 显示 tcp 最大 mtu 值

-o 将报告和错误信息输出到文件 #iperf -c 222.35.11.23 -o ciperflog.txt

-p 指定服务器端使用的端口或客户端所连接的端口 #iperf -s -p 9999;iperf -c 222.35.11.23 -p 9999

-u 使用 udp 协议

-w 指定 TCP 窗口大小，默认是 8KB

-B 绑定一个主机地址或接口 (当主机有多个地址或接口时使用该参数)

-C 兼容旧版本 (当 server 端和 client 端版本不一样时使用)

-M 设定 TCP 数据包的最大 mtu 值

-N 设定 TCP 不延时

-V 传输 ipv6 数据包

server 专用参数 :

-D 以服务方式运行。#iperf -s -D

-R 停止 iperf 服务。针对 -D，#iperf -s -R

client 端专用参数：

-d 同时进行双向传输测试

-n 指定传输的字节数，#iperf -c 222.35.11.23 -n 100000

-r 单独进行双向传输测试

-t 测试时间，默认 20 秒,#iperf -c 222.35.11.23 -t 5

-F 指定需要传输的文件

-T 指定 ttl 值

三 . Iperf 移植

移植的事情已经过去很久了，很遗憾我自己没有做记录，现在也不打算在花时间重新做一遍了，大家看看下面的两个参考资料吧，写的还是比较详细的，如果还有什么移植上面的问题，我们可以再讨论。

1. Iperf 的使用，编译和移植
http://blog.sina.com.cn/s/blog_5d9051c00100isnn.html

2. 交叉编译 iperf 测试 arm 开发板上 wifi 模块的网络流量速度
http://apps.hi.baidu.com/share/detail/48191324

四 . BeagleBone 的实测结果

1. UDP 测试
BeagleBone 为服务器

```
^Croot@beaglebone:/home/sd# iperf -u -s
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:   106 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.100 port 5001 connected with 192.168.1.105 port 49300
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec    111 MBytes  93.4 Mbits/sec  0.226 ms    0/79325 (0%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
```

图 1-143

```
C:\Users\Leo>iperf -u -c 192.168.1.100 -b 100M -i2 -t10
------------------------------------------------------------
Client connecting to 192.168.1.100, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.105 port 49300 connected with 192.168.1.100 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 2.0 sec  22.0 MBytes  92.3 Mbits/sec
[  3]  2.0- 4.0 sec  22.2 MBytes  93.2 Mbits/sec
[  3]  4.0- 6.0 sec  22.1 MBytes  92.5 Mbits/sec
[  3]  6.0- 8.0 sec  22.4 MBytes  94.0 Mbits/sec
[  3]  8.0-10.0 sec  22.5 MBytes  94.5 Mbits/sec
[  3]  0.0-10.0 sec   111 MBytes  93.3 Mbits/sec
[  3] Sent 79326 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec   111 MBytes  93.4 Mbits/sec   0.226 ms    0/79325 (0%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
```

图 1-144

BeagleBone 为客户端

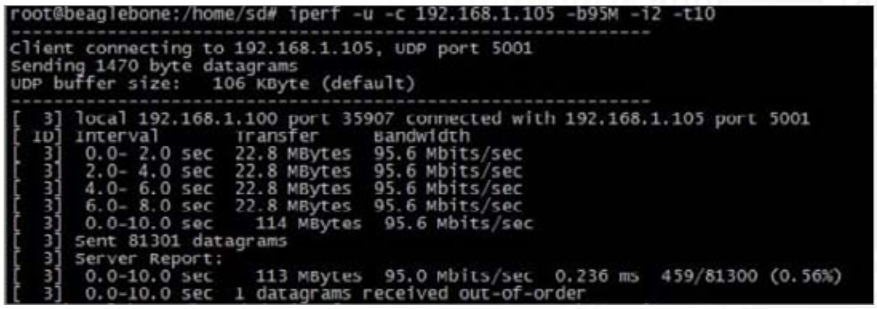


图 1-145

图 1-146

2. TCP 测试

BeagleBone 为服务器

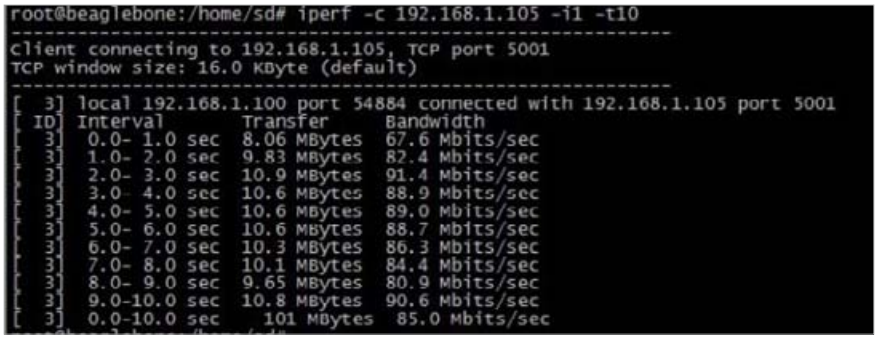


图 1-147

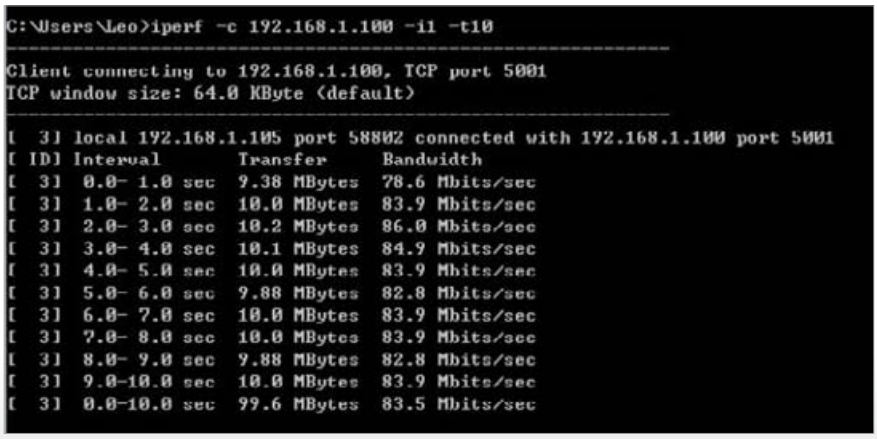


图 1-148

BeagleBone 为客户端

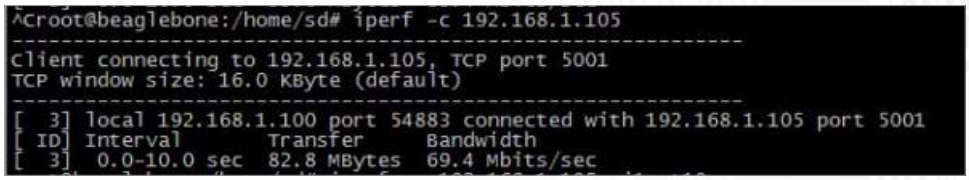


图 1-149

图 1-150

五 . 其他的一些参考参考资料 :

1. Iperf 工具使用

http://sharkyan.blog.51cto.com/536264/125016

2. Iperf 带宽性能测试使用方法与参数说明

http://www.net130.com/cms/Pub/special/special_fzjh/special_fzjh_yy/2010_07_27_40244.htm

六 . 总结

从测试结果来看，Beagbone 的百兆网口性能还是不错的，确实能够达到百兆的速度，不知道在千兆的接法下，表现如何，期待达人们的测试结果。如果对测试结果有异议，我们可以再切磋，争取把测试结果做到准确。

点击查看详情 : >> 基于 Iperf 的 BeagleBone 网络性能测试

第二章 官方课程

2.0 本章内容综述

本章内容来源 TI 官方 AM335X 系列培训课程（>> 观看课程视频），是该系列课程的书面整理。内容包括 AM335X 系列芯片的特点、TI 官方提供的软、硬件设计资源、相关软件操作、开发过程中相关调试与优化系统启动时间方法，当然还有相关的应用方向分析。

2.1 TI Sitara AM335x 系列概述

本节内容包括 TI Sitara AM335x 系列产品介绍、相关软、硬件资源及在线资源，旨在由浅入深的讲解 TI AM335x 相关的芯片知识和开发手段，尽快地让工程师熟悉 TI AM335x 开发环境以缩短开发周期。

本节内容由德州仪器半导体有限公司技术应用工程师鄂舒扬讲解。

今天我给大家介绍的内容是 TI Sitara 概述，主要内容包括：以 TI Sitara AM335x 产品为主，进行 TI Sitara 相关的介绍、TI 提供的硬件资源、TI 提供的软件资源以及相关的开发工具。TI Sitara 系列产品 am335x 是一颗集成了 Cortex-A8 在内的 ARM 芯片，是 TI 今年重点推出的产品之一。

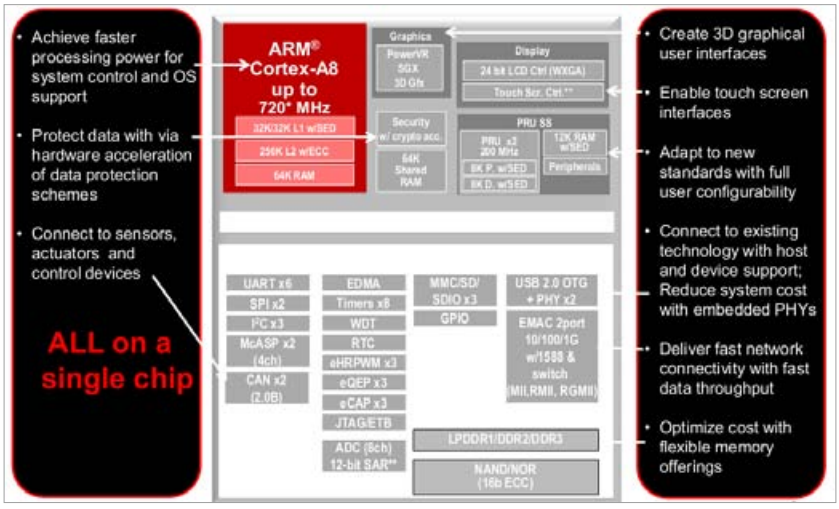


图 2-1

从上图可以看出，AM335x 系列产品除了集成 Cortex-A8 内核外，还具有非常丰富的外设，主要包括串口、LCD 控制器、ADC、以太网等。其中，以太网可支持 10 兆、百兆、千兆自适应的 PHY；该芯片同时支持 DDR1、DDR2 和 DDR3，DDR3 最高可以支持到 303 兆；AM335x 还提供了 GPMC 接口，该接口主要用于连接外部存储器，比如 Nand

Flash、Nor Flash 或者一些简单的 SRAM 等器件。此外，AM335x 还提供了一颗 3D 加速器，在 Android 系统和一些对 3D 要求较高的应用中，该芯片可以提供比较好的 3D 加速性能。

除上述内容外，AM335x 内部还包含了一个叫做 PRU 的系统，PRU 可以理解为一个内置的 MCU 或者 FPGA，用户可以在这个 PRU 里面去做一些自己的定制性开发（TI 为了方便应用开发，现已在 PRU 里提供了串口驱动）。比如，该芯片目前只支持 6 个串口，通过 PRU 可以再扩出 4 个串口，那么，AM335x 整体来说，就相当于已经支持了 10 个串口。同时 AM335x 还支持一些外部的工业总线，如 EtherCAT 和 PROFIBUS 等，这些总线的协议站都是在 PRU 系统里实现的。

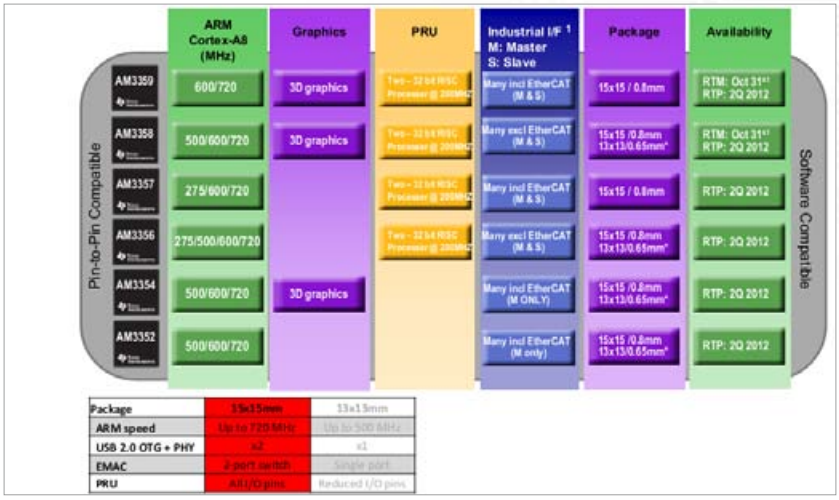


图 2-2

如上图所示，AM335x 是系列芯片的总称，产品包括从最低端的 AM3352 到最高端的 AM3359 系列。

其中，最低端的 AM3352 只包含一个 Cortex-A8 内核。高端一点如 AM3354 多了一个 3D 的加速，AM3356 和 AM3357 不包含 3D 模块，但支持 PRU 模块。AM3358 和 AM3359 是这个系列里面功能最全的芯片，功能包括 3D 模块、PRU 等。那么，我们不禁要问：AM3356 和 AM3357 以及 AM3358 和 AM3359 这两对芯片之间有什么样的差异呢？AM3357 内部集成了外部工业总线协议，包括 PROFIBUS 和 EtherCAT。例如，EtherCAT 和 PROFIBUS 在应用当中是需要交 License，但在 AM3357 和 AM3359 里已经实现了 License，也实现协议站。所以在使用过程当中，可以直接调用标准的接口，直接使用内部的库文件。

AM335x 系列里面，外部接口都是引脚兼容的。也就是说，如果开始设计是基于 AM3359 进行开发，后期如果要更改用 AM3352，硬件方面不用有任何的变动。

AM335x 采用两种封装方式：15mm×15mm 的 ZCZ 封装与 13mm×13mm 的 ZCE 封装。TI 官方推荐使用 ZCZ 封装，一方面原因是 ZCZ 封装在接口的支持方面要远远优于 13mm×13mm 的 ZCE 封装，比如，ZCZ 封装包含 2 路以太网，2 路 USB。而 ZCE 封装只支持 1 路以太网和 1 路 USB，另一方面原因是在引脚间距方面，ZCZ 的封装边距是 0.8mm，而 ZCE 的封装是 0.6mm，这对后续在生产时贴片方面也有一定的优势。

2.1.1 AM335x 的资源

TI 针对 AM335x 系列芯片提供了非常丰富的资源，包括软件资源、硬件资源以及一些相关的开发工具。

AM335x 软件资源

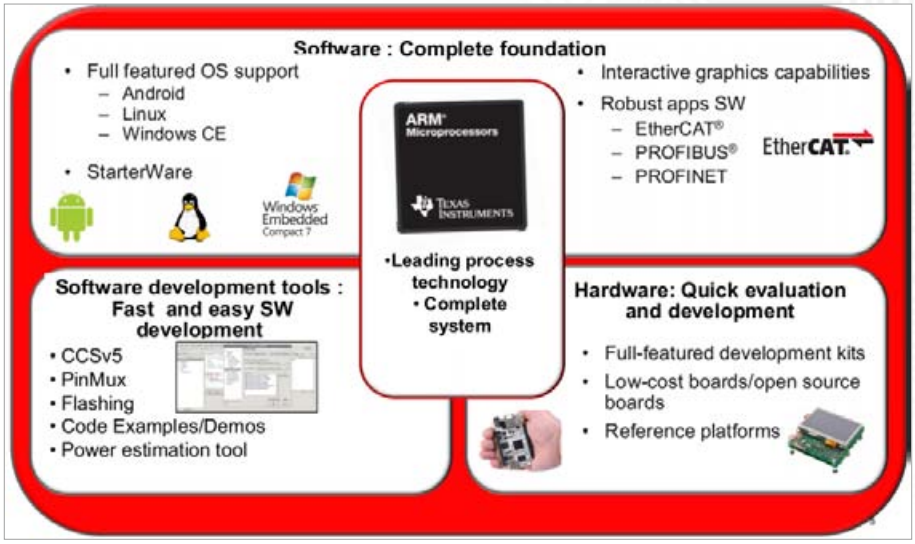


图 2-3

从上图中可以看出，在软件资源方面，AM335x 支持 Android 系统、Linux 系统和 WinCE 系统，以及 TI 开发的 StarterWare 系统。

图 2-4

以上的操作系统，TI 都提供免费源代码，用户可以直接从 TI 官网上下载。同时 TI 也提供一个叫 StarterWare 的软件，StarterWare 由 TI 维护，是一个不基于任何 OS 的裸跑程序。此外，AM335x 也可以支持 RTOS（实时的操作系统），这个实时操作系统是 TI 第三方维护。

2.1.2 AM335x 的软件架构

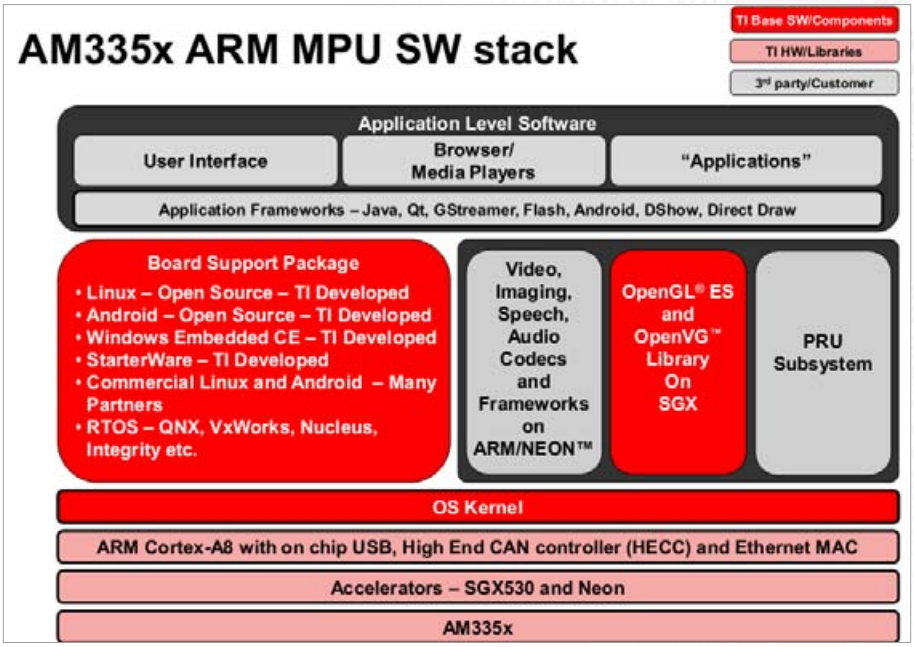


图 2-5

上图中用红色标识出的部分是由 TI 维护，也是 TI 编写的一些代码，包括各种板级支持包，以及 OpenGL 和 OpenGL VG 库的支持（这个库主要是跑到 SGX 上）。其他灰色部分，比如上层应用、上层 Demo，以及 PRU 内部的一些上层协议软件，都是由 TI 第三方维护。

下方列出了 AM335x Linux SDK 的一些相关资源的网址。

About the AM335x Linux SDK

- It is based on a Arago Open project
  http://www.arago-project.org/wiki/index.php/Main_Page
- Formal release
  http://www.ti.com/tool/linuxezsdk-sitara
- The SDK User guide
  In the installed folder.
  http://processors.wiki.ti.com/index.php/Sitara_Linux_Software_Developer%E2%80%99s_Guide

我们可以通过两种方式下载 Linux SDK。一种方式是通过 Arago 的网站，用 Git 的方式下载到最新的 SDK；还有一种方式，大家可以访问 TI 官方网站（http://www.ti.com/tool/linuxezsdk-sitara）去下载最新的 SDK。另外，针对 SDK 的使用，TI 在上图中所示网址中提供了详尽的说明。

目前 TI 发布的 SDK 版本是基于 Linux v3.2 这个版本，使用的交叉编译器来自于 Arago，这个交叉编译器在 TI 发布的 SDK 里面已经包含。另外 TI Sitara 所有的 Linux

SDK，或者说 Linux Kernel 的组成方式都是相同的，比如之前 Sitara 产品线里面的 AM35x、AM37x 以及 AM18x 等这些平台。如果客户使用过其中一个平台，再到下一个平台或者不同的平台去进行开发的话，就会发现其实大部分东西都是类似的。

以下是 TI 针对 Uboot 提供的资源：

- The Source Code

  The formal release is inside the Ezsdk.
  The OpenSource Project:
  http://arago-project.org/git/projects/?p=uboot-am33x.git;a=summary

- The Guide for the AM335x u-boot

  http://processors.wiki.ti.com/index.php/AM335x_UBoot_User%27s_Guide
  http://processors.wiki.ti.com/index.php/AM335x_board_bringup_tips

与 SDK 是一样的，Uboot 也可以通过两种方式下载。一种是通过 TI 官网；另外一种是到 Arago 网站去下载最新的 Uboot 资源。上述与 uboot 使用相关的网址提到了如何使用 Uboot，如何基于 Uboot 进行开发，如何基于 Uboot 进行 Pointing 的工作。

2.1.3 AM335x Linux Kernel 架构：

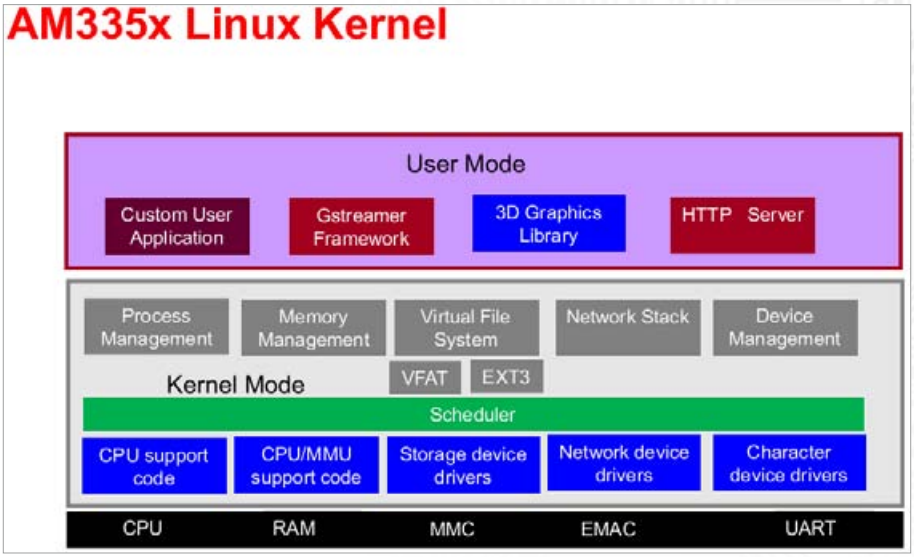


图 2-6

上层以及硬件抽象层所有的代码都是基于 Open Source3.2 版本，在板级支持包方面，TI 提供了丰富的接口 , 比如串口、网口、USB 等。所有 AM335x 上能提供的接口，在目前发布的 Linux Kernel 里面都有支持。

为方便用户开发，TI 针对 AM335x Linux KernelTI 提供了大量的资源。相关网址如下所示。

- The User Guide. This contains the guide for all the drivers
  http://processors.wiki.ti.com/index.php/AM335x_PSP_User%27s_Guide
- The performance evaluation
  http://processors.wiki.ti.com/index.php/AM335xPSP_04.06.00.07_Features_and_Performance_Guide
- The Source Code.
  The formal release is inside the Ezsdk.
  The OpenSource Project:
  http://aragoproject.org/git/projects/?p=linux-am33x.git;a=summary

里面提供包括如何使用 TI Linux Kernel，如何基于 TI Linux Kernel 进行软件移植；目前 TI Linux Kernel 上所能达到的性能和特性测试，以及如何去下载更新的 Linux Kernel 的途径。

2.1.4 AM335x 的文件系统：

在 TISDK 里面包含两种文件系统。一种文件系统里包含 TI 目前提供的所有 Demo、测试软件、应用程序等等。另外一种文件系统里是一个空的文件系统。TI 不推荐工程师自己去编译一套新的文件系统，因为在编译过程中，会浪费大量的时间。开发人员完全可以基于 TI 提供的这两种文件系统去进行开发、移植等等，这样会大大缩短开发时间。

目前 AM335x 采用的 Android 系统是 Android 4.0。从 Android 4.0 开始，谷歌要求每一个主芯片里都必须支持硬件的 3D 加速。如果产品中需要选择 Android 作为主操作系统，那么在选择 AM335x 平台时要从 AM3354 开始。AM335x Android 的下载可以从（http://www.ti.com/tool/androidsdk-sitara）进行下载，这里的代码是免费的。AM335x 提供的 Android 系统，除基本特性，比如 3D 和 Android 本身的一些特性之外，TI 还提供了 Android 本身的评估工具，以便于开发过程当中测试 AM335x 上的 Android 性能。同时 TI 的 Android 系统支持从 MMC 启动，支持从 Nand Flash 启动，以及从 USB Mass Storage 的设备上启动。

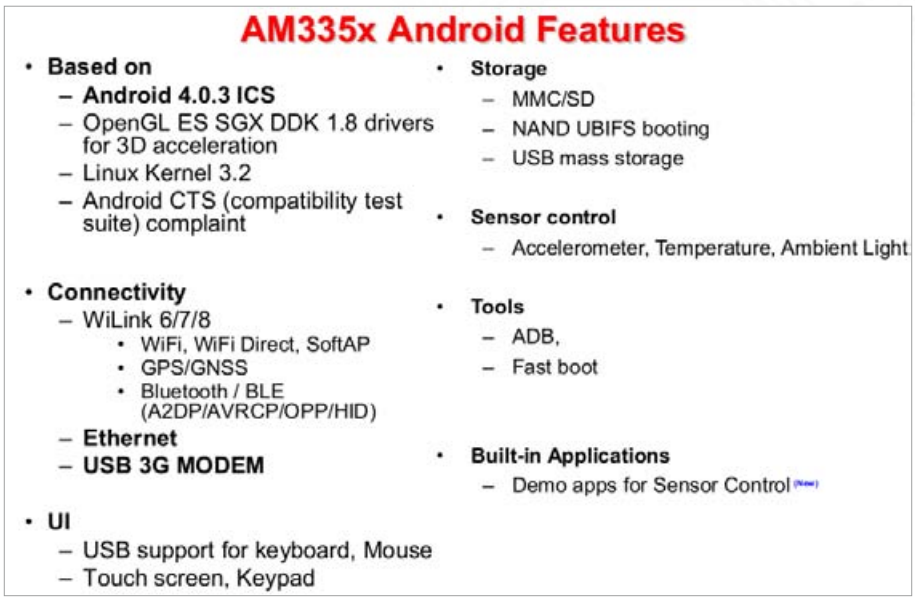


图 2-7

下面是针对 Android 所提供的所有文档的相关网址：

- How to use the AM335x Android sdk
  http://processors.wiki.ti.com/index.php/TI-Android-ICS-4.0.3-DevKit3.0.1_UserGuide

- The Evaluation on the AM335x Android sdk
  http://processors.wiki.ti.com/index.php/TI-Android-ICS-4.0.3-DevKit3.0.1_UserGuide

- How to develop the AM335x Android sdk

http://processors.wiki.ti.com/index.php/TI-Android-ICS-4.0.3-DevKit3.0.1_DevelopersGuide

http://processors.wiki.ti.com/index.php/TI-Android-ICS-PortingGuide
You can find more GREAT DOC in

http://processors.wiki.ti.com/index.php/Category:Sitara_Android

这些文档里包含如何使用 TI 的 Android 系统，如何基于 TI 的 Android 系统进行开发，以及 TI 的 Android 系统里提供的特性和性能。

TI StarterWare 系统：

StarterWare 是由 TI 自己开发的一套系统，它不基于任何的操作系统。它更多的作用是：当客户的板子第一次出来时，可以通过这套系统去测试各个外设是否通讯正常。TI 提供了 StarterWare 所有的源代码，工程师可以在源代码里去了解到整个 AM335x 的初始化流程（包括每个模块的初始化流程）。同时对于一些用不到操作系统的场景，比如说有些客户是从 M3 移植到 M4 平台上，客户也可以选择使用 StarterWare 作为自己的一个操作系统，这个源码是完全开放的，而且是完全免费的，可以 TI 网站上去下载。

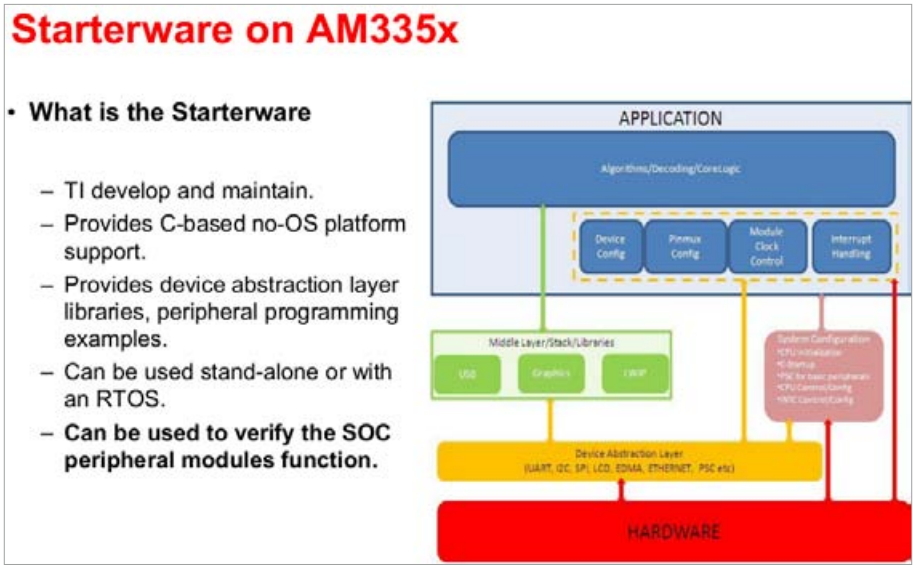


图 2-8

下面是 StarterWare 相关的所有文档的下载地址。

- Where to download the AM335x starterware
  http://processors.wiki.ti.com/index.php/StarterWare

- How to use the AM335x Starterware
  http://processors.wiki.ti.com/index.php/StarterWare_02.00.00.06_User_Guide

- How to develop the AM335x Starterware
  http://processors.wiki.ti.com/index.php/AM335X_StarterWare_Booting_And_Flashing

  http://processors.wiki.ti.com/index.php/AM335X_StarterWare_Environment_Setup
  You can find more GREAT DOC in
  http://processors.wiki.ti.com/index.php/Category:StarterWare

文档内容涉及如何使用 TI 的 StarterWare，如何通过 CCS 或者是 Linux 去编译，以及在 TI StarterWare 里所提供的所有模块的支持。

AM335x Windows Embeded:

AM335x 也支持 Windows 嵌入式系统，目前 AM335x 支持 CE7，不支持 CE6。从下图的网址中可以获得 TI CE7 的源码，以及相关的一些使用文档说明和移植说明。

- The WinCE package for AM335x
  http://www.ti.com/tool/wincesdk-a8

- The WinCE documents

  Internal wiki
  http://processors.wiki.ti.com/index.php/Category:Sitara_WinCE

  e2e support
  http://e2e.ti.com/support/embedded/wince/default.aspx

AM335x RTOS 系统：

AM335x 上可以跑 RTOS（实时操作系统），这部分的工作由 TI 的第三方合作伙伴去做 RTOS 的支持和开发，比如 QNX、Wind Riverr 和 RTLinux，更多详细的信息可以访问（http://www.ti.com/lsds/ti/dsp/support/software/os_overview.page）。因为 TI 不开发实时操作系统，所以如果在后续的使用过程当中有相关的需求，TI 会为您联系相关的合作伙伴。

2.1.5 AM335x 的硬件资源

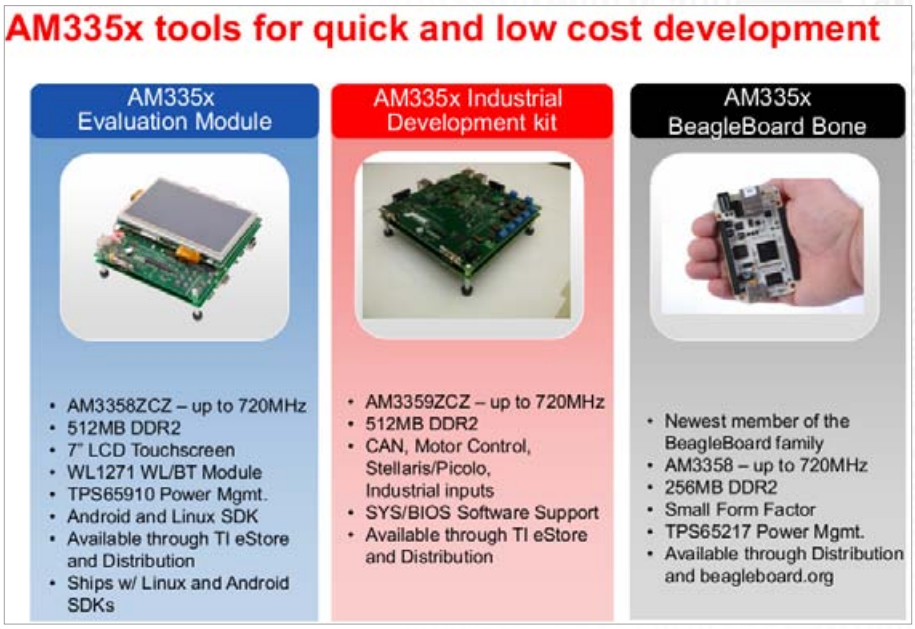


图 2-9

上图展示了目前市场上在销售的 3 种 AM335x EVM 板。左侧板子是通用的 EVM 板，它基本上支持所有的 AM335x 接口，包括以太网、触摸屏等等这样的支持。中间这块板子是针对工业客户提供的开发板，在这个板子里面 TI 实现了一些工业接口，比如 CAN、以太网、EtherCAT 和 PROFIBUS 等接口的支持。右边的板子是 BeagleBone，属于低成本板子，它只是实现了一些最小的系统，包含一个 CPU、一个外部的电源芯片、一个 DDR2、一个以太网的接口。BeagleBone 更适合于客户在进行 SOM 开发的时候相关的设计工作。同时针对 AM335x，TI 还提供了 Symbols 的支持。为了方便进行仿真，TI 也提供了 BSDL 模型。以上提到内容的相关资料可以从下面提供的网站中得到更详细的介绍：

- For schematic and PCB Layout.
  - BeagleBone http://beagleboard.org/hardware/design/
  - EVM http://www.ti.com/tool/tmdxevm3358

- Symbol for AM335x
  - http://www.ti.com/product/am3359#symbols

- BSDL model for both ZCZ and ZCE package
  - http://processors.wiki.ti.com/index.php/Device:AM335x:Device_Evaluation#Important_Documentation

此外，TI 为客户提供了原理图的 Checklist 和 PCB Layout 的 Checklist：

- Schematic & PCB layout checklist
  - http://processors.wiki.ti.com/index.php/AM335x_Schematic_Checklist
  - http://processors.wiki.ti.com/index.php/AM335x_Layout_Checklist

- DDR2/3 design & layout guide
  - More detail in AM335x Datasheet.

- We have provided power-consumption summary
  - http://processors.wiki.ti.com/index.php/AM335x_Power_Consumption_Summary

上面网址提供了所有在 AM335x 里设计原理图和设计 PCB 时需要注意的事项。

TI 官方建议设计人员在后续的设计过程当中可以经常的访问（http://processors.wiki.ti.com/index.php/AM335x_Power_Consumption_Summary），TI 会不断更新这个页面的内容，增加其他客户应用过程当中所遇到的一些问题的讲解。比如：AM335x 支持 DDR2 和 DDR3，很多客户没有太多这方面的设计经验，TI 会在 AM335x 的 Datasheet 里提供详尽的 DDR3 和 DDR2 原理图设计和 PCB 设计指南。

AM335x 是一个低功耗的器件，针对不同的客户应用，TI 提供了电源评估的工具。客户在使用工具时，可以根据自己的设计和应用选择相关的模块。该评估工具能计算出你的设计场景里所需要的功耗值。因为 AM335x 有上电时序和下电时序的需求，也有几路电源的要求。TI 公司为 AM335x 专门推荐了两颗芯片，一个是 TPS65910A 和 TPS65217。这两颗芯片也可以支持 DDR2 和 DDR3，更多详细的内容大家可以访问 http://processors.wiki.ti.com/index.php/Device:AM335x:Device_Evaluation#Power。

2.1.6 TAM335x 平台工具

下面主要介绍 TI 针对 AM335x 平台提供了哪些工具。

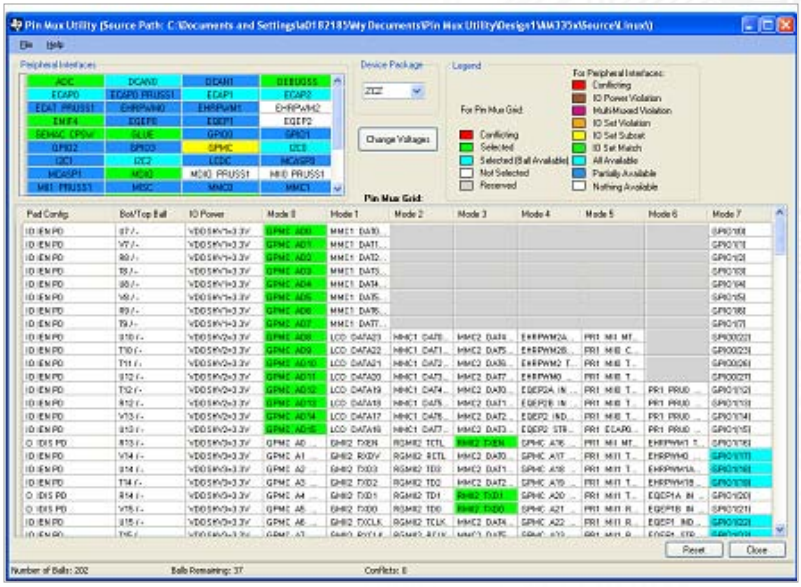


图 2-10

上图所示是 Pin Mux 工具。AM335x 支持丰富的外设，但本身管脚只有 300 多个，所以每个管脚上可能会有多个复用功能。Pin Mux 工具就是方便直观看到目前有多少个管脚可以配置出来，各个配置方式之间是否存在冲突，是否存在错误。如果没有这个工具，可能会给设计人员在硬件设计和软件设计中会造成很大的困扰。针对 Pin Mux 工具的使用的更多详细内信息可以访问如下网址：

http://processors.wiki.ti.com/index.php/Pin_Mux_Utility_for_ARM_MPU_Processors.

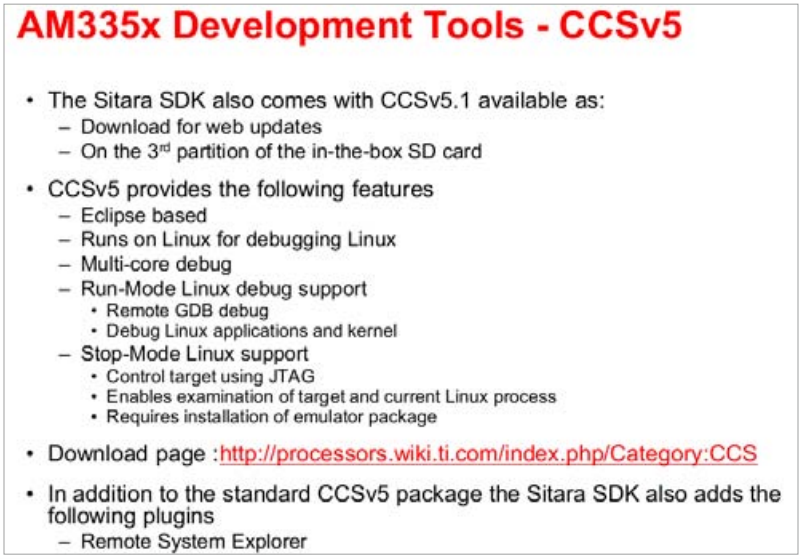


图 2-11

上图是 AM335x 另外一个开发工具 CCS 的相关内容。CCS 是一个通用的 TI 开发工具，

它支持绝大多数 TI 的平台。目前 CCS 已经是第五个版本。目前可以用 CCS 去进行 Linux 和 Uboot 的调试。更多的信息和下载信息可以访问：

http://processors.wiki.ti.com/index.php/Category:CCS。

针对 CCS 工具的使用，TI 提供了多种仿真器支持：

- TI has different level emulators for AM335x
  For the high end emulator
  https://estore.ti.com/XDS560v2-System-Trace-P2124.aspx

  For the low end emulator
  - http://www.ti.com/devnet/docs/catalog/embeddedsoftwarefulldetails.tsp?actionPerformed=productFolder&productId=10013

- The JTAG interface for Sitara, is not compatible with the ARM JTAG.
- The Document s for the emulators & JTAG
  http://processors.wiki.ti.com/index.php/XDS560
  http://processors.wiki.ti.com/index.php/XDS100
  http://processors.wiki.ti.com/index.php/JTAG_Adapters

上述网址展示了一个最高端的仿真器和一个最低端的仿真器。两者在调试和下载速度方面有些差异，其他的调试没有任何差异。这里需要注意的是，TI 接下来的接口标准和 ARM 的标准是有些差异的，在下面这个网站里面提供了两个差异之间的对比

(https://estore.ti.com/XDS560v2-system-Trace-P2124.aspx,

http://www.ti.com/devnet/docs/catalog/embedded Softwarefulldetails.tsp?actionPerformed=productFolder&productId=10013)。

当然在购买的仿真器里，TI 也提供了相应的转换接口，这样，您就可以在 TI 的平台上去使用 ARM 的仿真器进行仿真和调试。

除了以上提到的一些网址和资源之外，TI 针对 Sitara 和 AM335x，TI 提供了很多的在线支持资源：

- The wiki page

  The hardware, software design guide are all on TI wiki as below
  - http://processors.wiki.ti.com/index.php/Sitara

- The online forum
  - E2E forum -- Global export will support you here
- http://e2e.ti.com/support/dsp/sitara_arm174_microprocessors/default.aspx
- http://e2e.ti.com/support/embedded/linux/default.aspx

- http://e2e.ti.com/support/embedded/android/default.aspx
- http://e2e.ti.com/support/embedded/starterware/default.aspx
  - DeyiSupport -- Local FAE will support you here
- http://www.deyisupport.com/question_answer/f/25.aspx

在 TI 维基百科里有非常多的资源和内容供大家参阅，甚至有一些跟 TI 芯片本身没有直接关系的一些内容，比如 USB 的 Layout Guide，DDR Layout Guide，您在维基这个网站里都可以找到相应的描述。除了维基百科，TI 还提供了很多在线论坛，TI 的 E2E（Engineer to Engineer）论坛（http://e2e.ti.com/support/default.aspx），，在论坛上提出的问题，24 小时都会有人给回复，但是需要使用英文。针对中国客户 TI 提供了 deyisupport 论坛（http://e2e.ti.com/support/default.aspx），在这个论坛里有 TI 本地工程师的技术支持，您可以用中文去提问，也可以用英文来提问。TI 推荐开发过程当中去使用这两个论坛，有可能会提高开发人员的开发速度。

2.2 基于 AM335x 平台 Uboot 的一些移植方法

◆本节内容概览

◆什么是板级的移植

◆如何基于 TI 提供的 Uboot 移植到自己的板子上

什么是板子的移植呢?

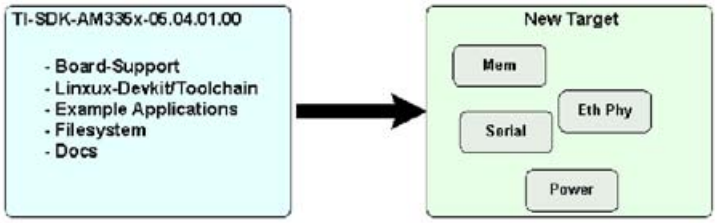


图 2-12

从上图可以看出，板子的移植就是将 SDK 里面提供的代码放到你自己的板子上。因为 TI 的 SDK 所提供的 Driver，所提供的软件完全是基于 TI 的 EVM 板，而你设计的产品会增加一些自己的东西。比如换个 PHY，增加一个 USB 驱动，甚至增加一个全新的外置模块。在这种情况下，需要基于 SDK 更改我们提供的软件去做相应的调整，这个过程就是一个板子的移植工作。

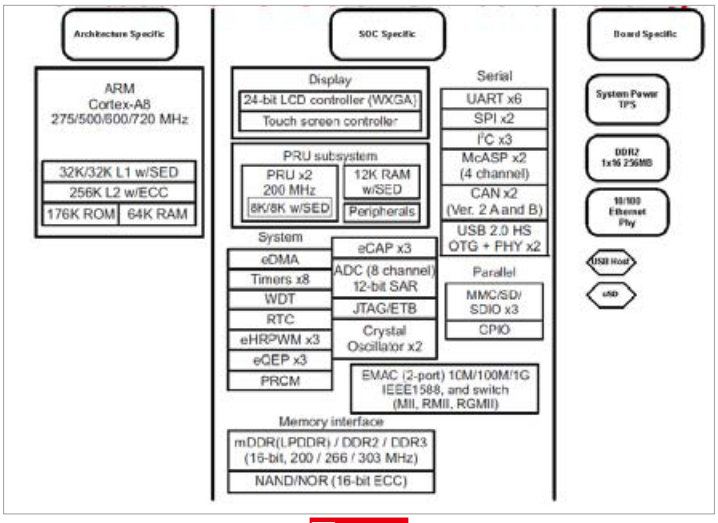


图 2-13

上图展示了三种不同的移植区别。最左边这张图是架构移植，很多芯片（尤其是嵌入式的芯片）架构不太一样，比如 TI 的 AM335x 是基于 Cortex-A8，也许你之前使用的是基于 MIPS 或者 Power PC 内核等芯片，那就要从 MIPS、Power PC 移植到 Cortex-A8，这个过程就是架构的移植。中间图展示的是板级或者 SoC 的移植，例如，TI AM335x 和 AM35x 是两个完全不同的芯片，虽然它们都是基于 Cortex-A8，但它们使用的外设和接口有很大的差别，那么从 AM35x 平台移植到 AM335x，需要改动的就是接口，也就是说 AM35x 可能有 2 个或 3 个 I2C，但是 AM335x 可能有 4 个或者更多的 I2C，从 2 个或 3 个

I2C 的移植，到 4 个或 5 个 I2C，这个过程就是一个 SoC 的移植过程。最右边的图是板子的移植过程，如上面提到的，因为 TI 提供的 SDK 是基于 EVM 板，而用户设计的产品肯定不会跟 TI 的 EVM 板是一模一样的。在这种情况下，要适当的去调整或者是删减一些模块或者驱动，这样的工作就叫做板子移植。

所有的 Linux 和 Uboot 的移植都遵循同样一套流程。第一步先进行架构的移植，第二步进行 SoC 的移植，第三步也就是最后一步是进行板级的移植。架构的移植和 SoC 的移植，在 TI 提供的 SDK 里有所支持，所以开发人员可以将所有的精力放在板间的移植。

下面看一下 SDK 跟移植相关的内容，或者说源代码是放在 SDK 的哪个目录里面。

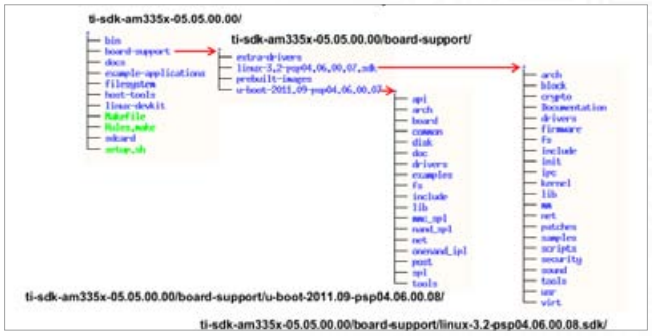


图 2-14

从图中可以看出，在 SDK 的 Board Support 目录有 Linux 和 Uboot 这两个文件夹，这两个文件夹下存放的就是与 Linux 和 Uboot 相关的一些源代码。

以前使用 TI 平台的客户在使用 AM335x 平台之后，第一个问题就是 X-Loader 在哪里。事实上，从 AM335x 开始，为了简化设计，TI 将 X-Loader 和 Uboot 合在一起。也就是说，在 Uboot 这个文件夹里可以找到既是 SPL，又是 Uboot 的一个代码。同时 TI 会使用宏的方式去区分这两个代码，编译器根据这两个宏的定义去编译出 X-Loader 和 Uboot.bin 文件。

图 2-15

上图展示的是 Uboot 跟移植相关的一些目录。在整个移植过程当中，其他的目录完全可以不用去看，大部分的驱动和代码 TI 都已经在里面有实现，所有相关的内容请只需访问这两

个路径。所有需要更改的文件和头文件，也都列在这两个目录下面。

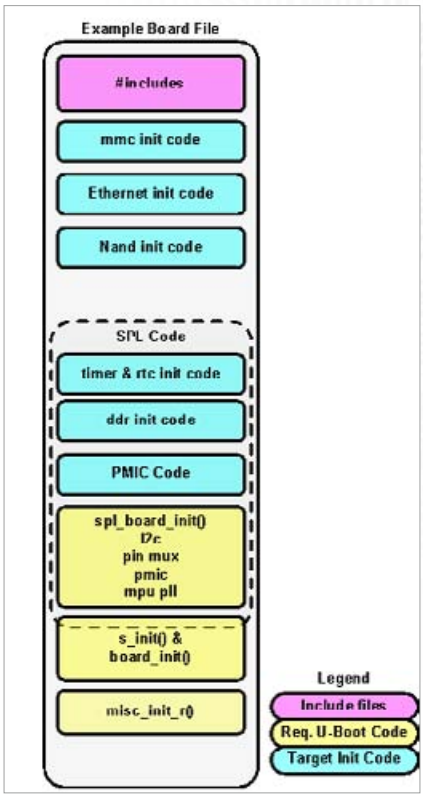


图 2-16

上图介绍的是目前在 SDK 里提供的板级文件的概况。在这个板级文件里定义了一些在 Uboot 和 SPL 里基本和必须的一些外设接口初始化程序，在一个文件里同时支持了 SPL 和 Uboot 两种程序，编译器会根据上面提到的两个宏定义去区分以下的这些代码到底是编译在 Uboot 里还是编译在 MLO 里。同时在这个板级文件里也提供了对 DDR、串口以及外部的电源芯片的支持。但是存在一些函数在 Uboot 里和 SPL 里都有支持，都有所编译。这个板级文件是在进行板级移植时重要的文件，所有的更改和所有的添加都是要在这里进行。

TI 针对移植提供了一个模板文件。在这个模板文件里所提供的只是最基本的功能，比如说 DDR 的初始化，串口的初始化，MPU 频率调整初始化以及 PMIC，也就是外部电源的初始化的程序。在进行移植过程当中可以根据自己的需求增加一些简单的初始化程序，或者去掉一些不用的功能，比如说如果在使用过程当中没有使用到推荐的外部电源芯片，那么完全可以把外部电源初始化这部分代码去掉。

下面是关于移植方面的两个例子。第一个例子详细介绍一下提供的这个模板文件是如何工作的。第二个例子会选择增加一路 MMC 和增加一路以太网来实现板级的移植工作。

主要的目的是希望能够通过例子了解到 TI 提供的模板文件是如何工作的，或者说如何通过 TI 提供的模板文件去进行下一步的移植工作。第一步，DDR 和时钟的初始化在哪里？DDR 初始化总共分四个部分，除一般通用的 Timing 的配置之外，还有包括 DDR PHY 的设置，DDR VTP 的设置，以及 PLL 设置。

可以通过下面这张图，可以看到很清楚这四种配置之间的关系。

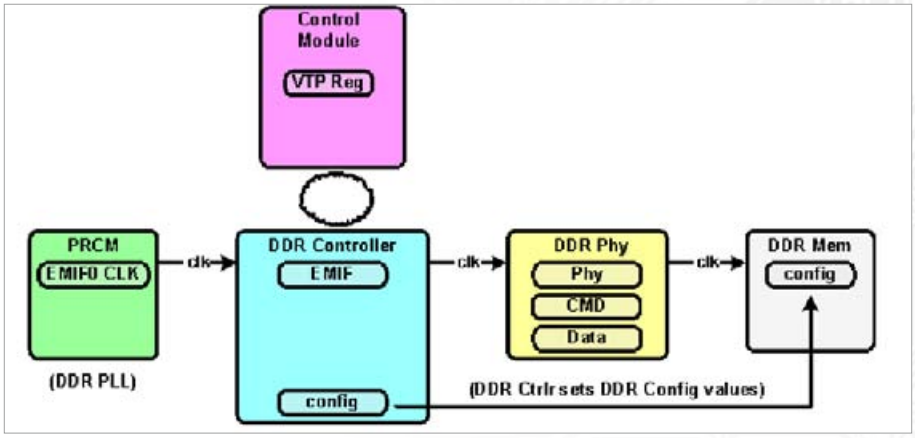


图 2-17

其中 EMIF、CMD、DATA 和 EMIF0 CLK，这些参数是需要根据使用的外部的 DDR 芯片来决定，更多的信息需要去查看 DDR 的数据手册。

下图是在哪里进行 DDR 的一些初始化信息。

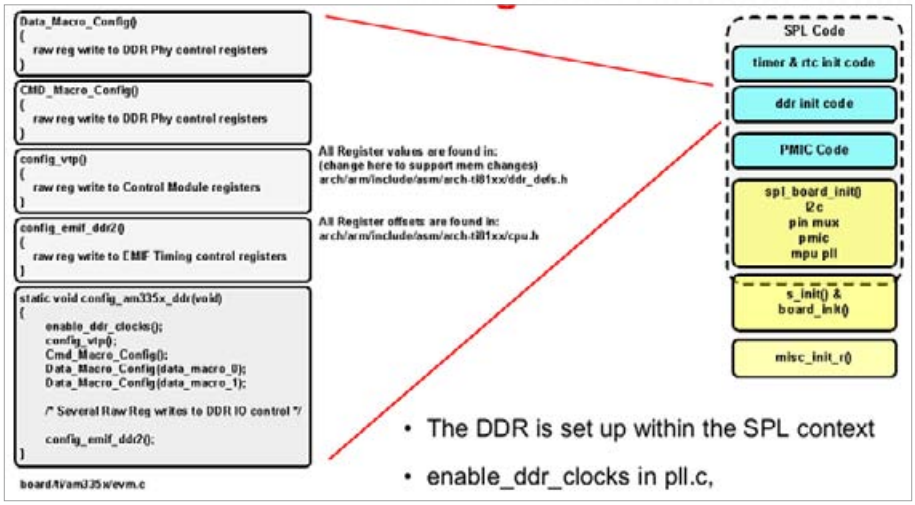


图 2-18

上图右侧的图主要给介绍 TI 提供的模板文件大致的概况，左侧的五个代码是实现 DDR 初始化的具体细节，包括 CMD、DATA、VTP 以及 EMIF 时序的配置，比如说如何去 DDR 相关 Clock，以及 DDR 相关的 Timing，这些参数以及这些函数放置的位置，在右下角说明了。因为 DDR 是一个与时序关系比较紧密的一颗器件，AM335x 针对 DDR 也提供了很多的寄存器。为了方便设计人员，在选择新的 DDR 之后更快的可以让整个 DDR 工作起来，TI 提供了网站（http://processors.wiki.ti.com/index.php/AM335x_EMIF_Configuration_tips）。在这个网站里面提供了一个配置工具，可以在使用过程当中通过查阅一些相关的参数。从 DDR 的数据手册里将这些相关的参数放到提供的工具里，这个工具就可以算出在这些参数下相应的时序寄存器里的值，把这些值写到上面提到的那些文件里，一般 DDR 就可以正常工作了。

SPL，也就是 X-Loader，整个函数的入口点是 S.init 的一个函数。

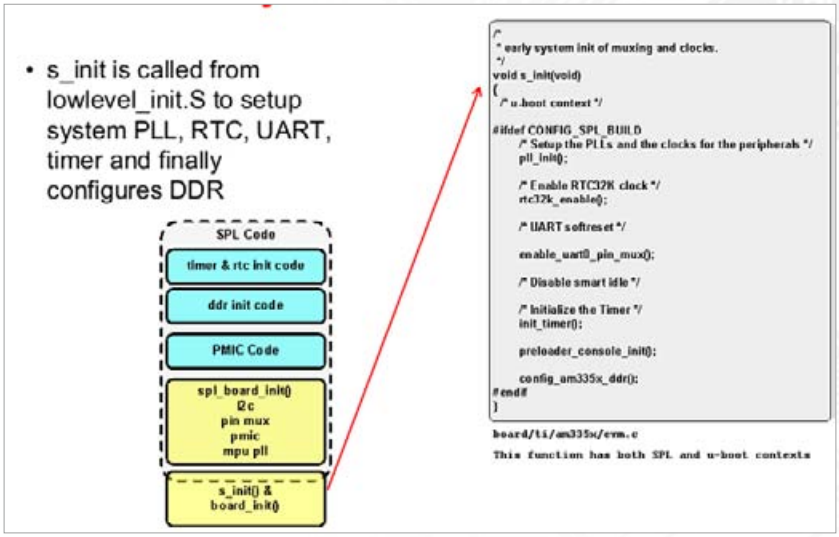


图 2-19

从上图右侧可以看到，在这个函数里实现了很多非常基本的初始化功能，包括上面提到的 DDR，包括配置 MPU，包括一些简单的 PRL 的设置。这个函数是被 Lowlevel_init.s 这个文件去调用。

现在介绍一下如何去设置 MPU 的时钟频率。

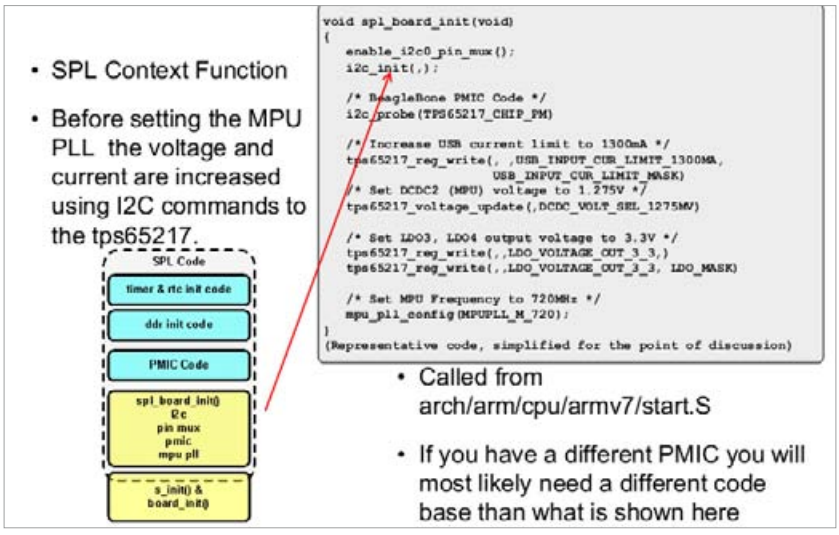


图 2-20

上图左下角介绍了配置 MPU 时钟频率的函数，右上部分是整个函数实现的一个细节。值得注意的是 TI AM335x 出于功耗的考虑，每一个 MPU 会有相应的一个电源的供电值去进行匹配。例如，AM335x 最高可以支持到 720 兆，在 MPU 跑到 720 兆的时候，相应的 VDDMPU 这个供电电压是需要达到 1.26 伏。在 MPU 跑到 600 兆的时候，相应的供电电压是需要达到约 1.2 伏左右，这样做的目的完全是为了节能。

在整个函数里面，第一步先设置电压的值，将 PMU 或者是 PMIC 电压的输入值设置成相应的电压输出之后，然后配置整个 MPU 的时钟频率。这个过程是不可以颠倒过来，如果

颠倒过来，就会出现超频现象。在这种情况下，整个 AM335x 的性能是没办法保证的。

下面介绍一下在 Uboot 里所提供的一个模板文件。

```
int board_init(void)
{
    /* Configure the i2c0 pin mux */
    enable_i2c0_pin_mux();

    i2c_init(CONFIG_SYS_I2C_SPEED, CONFIG_SYS_I2C_SLAVE);

    board_id = BONE_BOARD;

    configure_evm_pin_mux(board_id);

#ifndef CONFIG_SPL_BUILD
    board_evm_init();
#endif

    gpmc_init();

    return 0;
}
```

board/ti/am335x/evm.c

图 2-21

上图所示的这个函数是整个 Uboot 的入口函数。在这里提到的有些函数其实在 SPL 里面已经初始化过一遍，在进入到 Uboot 里面，可能还需要再初始化，但是这样不会有任何问题。有一些 Pin Mux 的设置，比如说像 I2C 的 Pin Mux 设置，在 SPL 已经有过初始化或者是设置，那么在 Uboot 里，需要再重新设置一遍。

第二个例子是实现 MMC 和以太网的支持。在开始进入到移植之前，第一步是先需要查看一下系统的框图以及原理图，弄清楚需要移植的这些器件使用到的 AM335x 上哪些 Pin Mux。以 MMC 和 EtherCAT 为例，当确认了 Pin 脚使用之后，需要完成设备的初始化工作。很多设备的初始化工作其实在 TI 提供的 AM335x 的 SDK 里都已经有所实现，只需要增加相应的初始化函数，或者说打开相应的初始化函数就可以。

首先我们先用 Pin Mux 工具对使用的管脚进行配置。

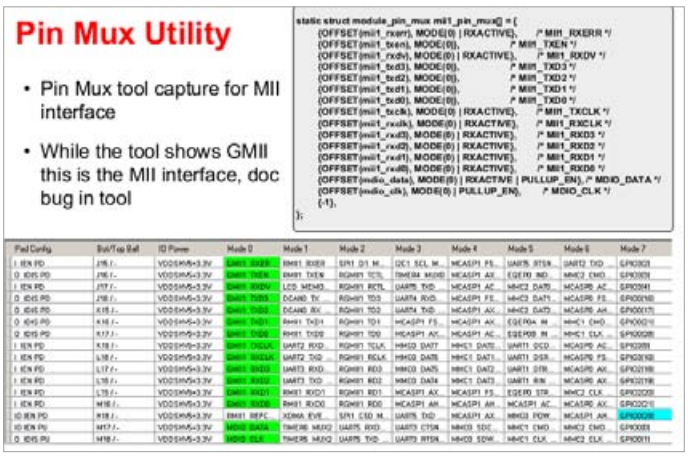


图 2-22

上图展示的是以 MI 接口为例。在下边图里面是所有跟 MI 接口相关的 Pin 脚，右上角是在软件中根据下图配置的管脚，在上图软件中进行的定义。上图的软件中我们使用的是一个

结构体，可以看到在结构体里包含了很多成员，每个成员基本的定义就是说左边的那个参数是这个 Pin 脚的名称，而右边配置的是这个 Pin 脚的模式，在截图里面，每一个管脚，总共配置了大概是 7 种到 8 种的模式，目前选择 MI 这个接口基本上都是模式点。所以在软件设置的时候，要把这个设计配置成模式点。MIC 的配置方式跟 MI 的配置方式是类似的。

下面增加对 MMC 初始化代码，第一步需要在 AM335x-evm.h 这个文件里打开跟 MMC 相关的所有宏，相关宏的定义在右边图里。

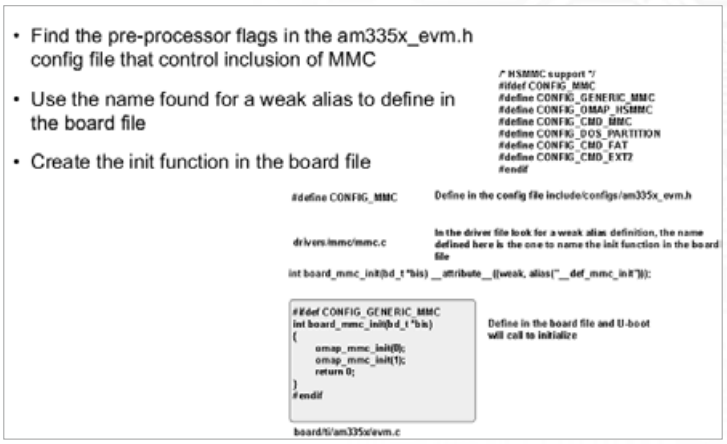


图 2-23

同时在 MMC.C 文件里，需要使用 board_mmc_init 这个文件，这个函数名去进行初始化。最后一步，需要在 board_mmc_init 我们需要增加对 MMC 的初始化。这里举的例子是增加了对于 MMC0 和 MMC1 的初始化，可以根据自己的需求，比如只需要 MMC0，那么就可以把 MMC1 初始化去掉，以节省启动时间。与 MMC 初始化代码类似，以太网也需要执行相似的流程。首先需要在 eth.c 文件里找到 board_eth_init 这样一个文件，在这个函数里可以增加对 PHY 初始化，可以增加对 Mux 地址初始化这样的操作。做完这个操作之后，你的程序就会支持太网了。

以上就是所有关于 Uboot 移植内容。

2.3 优化 Linux 的启动时间

本节介绍一下如何去优化 Linux 的启动时间。主要是介绍一下什么叫快速启动，启动的流程以及如何去测试启动时间,最后分析一下在 Uboot 和 Kernel 里的启动时间以及优化方式。

快速启动就是从板子上电开始到整个系统可用这段时间就叫启动时间。但是因为不同的设计有不同的需求，这个可利用的概念就会有很多不同的区分。首先有些设计人员可能认为这个可利用的时间，或者说可以使用的时间就是到 LED 被点亮的时间，这个启动时间就是从上电到 LED 被电亮的时间，这个就是它的启动时间。在有些场景里，所谓的可利用的时间或者说可应用的时间是指文件系统已经加载，界面已经可以输出，比如像我们提供的 Matix 这样的界面，这个启动时间就是从最开始上电一直到 Matix 界面启动之后，这个时间就叫做启动时间。TI 目前提供的 SDK 没有在启动时间上做很多的优化，因为不同的设计有不同的需求。同时 SDK 里面也包含了一些其他易于开发的组件在里面，所以说 TI 提供的 SDK 没有做任何启动时间的优化。

下图主要大概介绍一下整个初始化的流程。

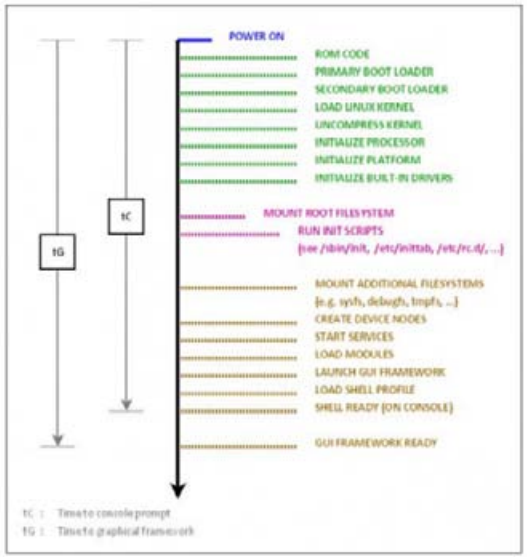


图 2-24

第一步是上电。上电之后，内部的 Rom Code 会执行,，内部的 Rom Code 主要目的就是初始化一些基本的外设。比如从 Nand Flash 启动，那么内部的 Rom Code 就会把与 Nand Flash 相关的一些接口还有设置的频率设置好，然后从 Nand Flash 里去载入第一个镜像，也就是 X-Loader，或者也叫 MLO。MLO 的主要工作是为了初始化 DDR，它初始化 DDR 之后，也会从选定的启动设备里，比如说 Nand Flash 或者 Nor Flash 里读取下一个启动镜像 Uboot.img。Uboot.img 跟 MLO 的区别在于，MLO 是运行在内部的 SRAM 里，也就是芯片内部的 SRAM 里，而 Uboot.img 文件是运行在 DDR 里面，正常来说，如果 DDR 初始化是不正常的，那么程序只能是跑在 X-Loader 里，那么下面 Uboot 就不会正常被执行。Uboot 启动之后，下一个载入的镜像是 U.img，也就是我们经常说的 Kernel。Kernel 的过程主要分载入 Kernel、解压 Kernel 以及加载文件系统等等这样的过程。Kernel 起来之后，就会去加载下一步的文件系统，文件系统在起来之后，就会加载所设定的一些最初始化的一些进程，下一步就会进入到正常的 UI 流程里。

有一些工具是可以用来测试整个启动时间，下边列了三种：第一种是可以通过一种C代码的方式，当你把C代码按如下网站（http://processors.wiki.ti.com/index.php/Measuring_Boot_Time）提供的方式编译好，并加载进去之后，它会告诉你每一个加载的模块以及整个启动的时间大致是多少。第二种是Real Term，第三种是Tera Term，这是两个调试工具。这两个调试工具在使用过程当中它本身就包含了可以帮你计算在从Uboot启动到整个文件系统加载完毕，整个时间都会帮你计算完成。

下面是通过C代码的方式去测试Linux的启动时间的一个例子。

图 2-25

标示出来的是加载的一个模块。可以发现，这个代码，或者说这个模块加载的时间大概约为1.5秒左右。如果在使用过程当中没有使用到这个模块，完全可以通过下面提供的方式，把它从Kernel里去掉，就可以节省时间。

如何去确定刚才提到的这几个启动流程？第一步SPL，也就是说X-Loader或者MLO，当执行到这一步的时候，就会在串口上发现第一行代码，第一行调试信息。当第一行调试信息被输出之后，就说明X-Loader和SPL已经正确被执行了。在Uboot阶段，当进入Uboot的时候，AM335会在串口端打印就可以确定Uboot代码已经被正确执行。再下一步是Linux Kernel，当串口终端上看到了图中相关的信息，也就是说，当看到这样的信息输出在串口上的时候，就可以确定Uboot代码已经被正确执行。再下一步是Linux Kernel，当串口终端上看到了图中相关的信息的时候，可以确定您的Kernel已经被正常加载并执行。同样文件系统，串口终端上发现有如下的调试信息输出的时候，可以确定您的文件系统已经被正确加载，并且被执行。

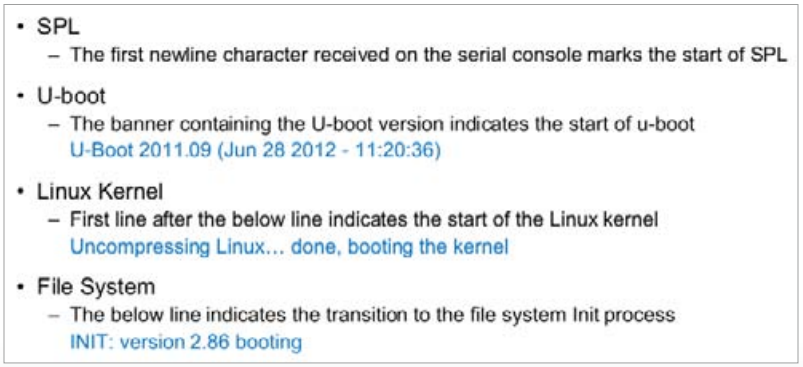


图 2-26

如何去分析Boot Loader的一个启动过程。Boot Loader也就是常说的X-Loader、

MLO 以及 Uboot，这里没有一个非常直接的方式去描述整个的启动流程。但是通过串口调试信息的方式，这将非常有助于理解整个 Uboot 启动的流程。下图截取了一段在 Boot Loader 启动过程中的一段 Log 信息。

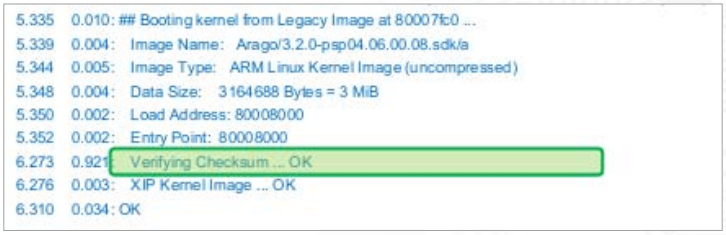


图 2-27

可以看到，在 Uboot 进行 Kernel 解压缩和 Kernel 的验证过程，这个时间大概是 2 秒钟。如果在设计当中，Kernel 被验证失败了，没有任何的处理方式，那么我们的建议是将整个验证过程去掉。这样的话，会节省大概 2 秒钟。具体的使用方法和说明，请参考下面提供的一个测试方法和调试方法。

如何去分析 Linux Kernel 的启动时间。最简单的方式就是在配置中打开 CONFIG_PRINTK_TIME 这个宏。

当打开这个宏之后，它就会把所有 Linux 启动过程当中加载模块的时间和总共启动时间打在所有 Log 信息的前面，如下面的图所示。

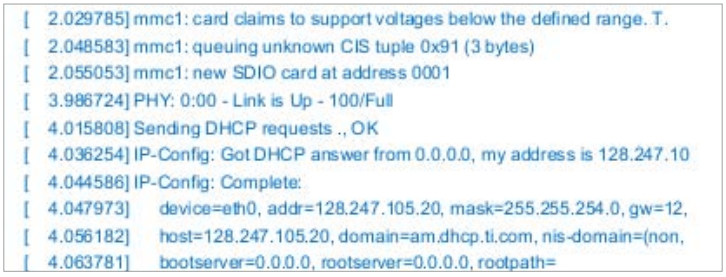


图 2-28

在这可以发现有一个问题。例如，当通过 DHCP 的方式去获取 IP 的时候，这里的时间大概耗费 2 秒钟，如果在系统当中，没有以太网的应用，那么建议在系统编译之前将这部分的代码去掉，这样的话也会为节省了将近 2 秒钟。具体的更改方式也请参考下面提供的这个方式，可以在 Uboot 增加提供的命令。

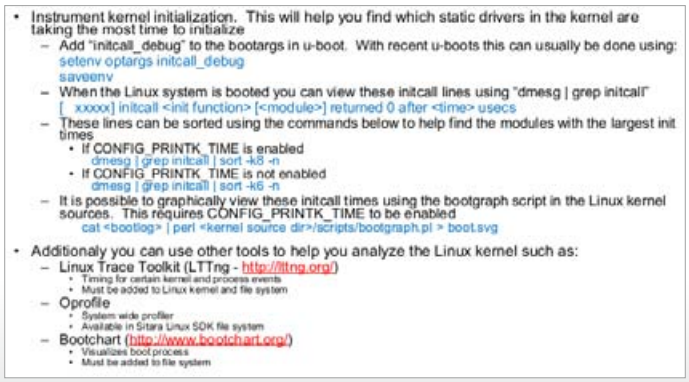


图 2-29

当整个 Linux Kernel 启动完之后，可以在终端里输入上面提供的这个指令。在这种情况下，AM335x 会如下面提供的这个输出信息方式来输出整个启动过程当中每个模块所消耗的时间。当然也可以在这个命令之后增加一些参数，以达到分类的作用。这里举了两个例子，分别是在不同的宏定义之下可以通过增加这个参数来查看在整个模块加载过程当中最耗时的模块有哪些。同样，也可以通过图形化的方式来查看整个启动时间以及哪个模块浪费的时间是最长。同时，除了以上这些指令之外，TI 还提供了三种不同的方式，这三种不同方式的获取方法以及使用方法您可以访问网站（http://lttng.org/，http://ww.bootchart.org）。

下面介绍一下总共有哪些方面可以用来优化启动时间。

首先是容量大小。在整个启动过程当中，整个 AM335x 需要不断的去加载一些 img，包括 U.img，U.boot 等等。在加载的过程当中，如果文件比较大，那么它加载的时间就会变长，所以减小这些镜像文件的大小是加快 Linux 启动时间的一个有效方式。另外，除了大小之外，还有运行速度以及程序加载速度等等，可以达到优化 Linux 启动时间的目的。在下面提供了一些简单的例子供大家进行参考。

下图介绍的是针对 Uboot 的一些优化方法。

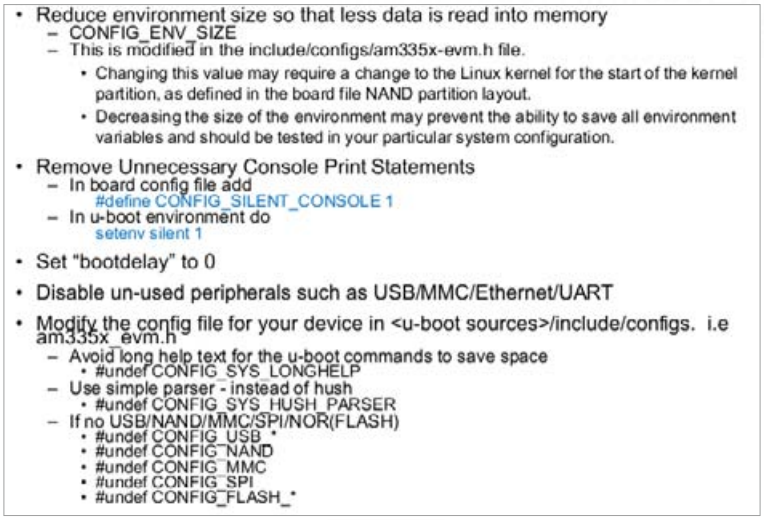


图 2-30

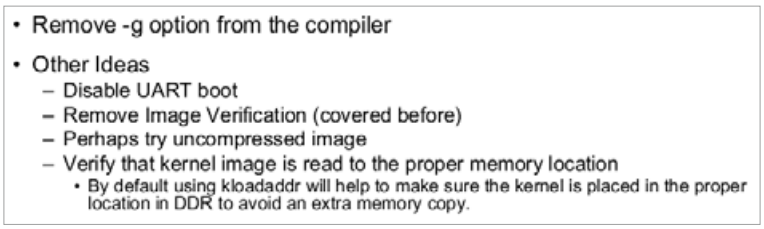


图 2-31

第一点可以通过减小 Uboot 环境变量的方式来做，在提供的 SDK 里，默认的一个值，在很多时候在使用过程当中是不需要有这么大的一个空间去存储相应的 Uboot 的环境变量。在这种情况下，可以将这个默认值改小。改小默认值，最后影响的就是将整个 img 的大小也会小，这是一种方法。另外当调试过程结束之后，不再需要串口的控制台，可以屏蔽掉控制台的输出。

这样的话一定程度上会提升 Linux 的启动时间，同时在 TI 提供的 SDK 里，设置了一个

Bootdelay 参数，主要是为了方便在 Uboot 的启动过程当中，可以随时调出 Uboot 的控制台。如果已经结束了调试阶段，在这种情况下，完全可以将这个 Bootdelay 的时间设置为 0，没有必要让整个 AM335x 再去等待用户的输入。同时在 Uboot 里，可以去掉一些不用的模块，比如说 USB，比如说以太网，比如说串口等等，当然这种需求是由硬件设计所决定的。同时也可以在编译器里去掉 G 选项。另外除了以上提供的一些方法之外，还有其他的一些方式，比如屏蔽掉串口启动，在 AM335x 的串口启动模式里，会先输出一大块的同部头，去等待串口终端的响应。在这种情况下，它也会浪费一段时间。同时如之前提到的，如果系统对验证失败的 img 没有做任何处理的话，完全可以将验证这个过程去掉，这样也会省几秒钟的时间。同时如果有可能，请尽量选择未压缩的 img，这样会节省解压 img 的时间。

下面主要介绍一些针对 Linux Kernel 一些优化的方式。

首先最简单也是最直接的方式，就是去掉一些不用的 Driver。比如说你设计的系统里没有用到 I2C 和 SPI，完全可以在 Kernel 里将 I2C 和 SPI 的驱动去掉。在这种情况下，Linux 就不需要花时间再去初始化相应的一些模块。同时针对有些非快速启动的一些驱动，在编译的时候完全可以将它选成编译成模块，而不是编译成内部文件。在这种情况下，初始化这些驱动的时间，就会放在完成了所有的启动，包括文件系统启动之后再去加载。另外如果有可能，请屏蔽掉串口的调试输出。串口在输出调试信息的时候会占用系统大量的时间，这样的话，会对整个启动时间有比较大的影响。同时，如果系统目前还需要有调试信息的输出，那就尽可能去将一些没有用处的调试信息去掉，这样的话也会节省您的很多时间。在 Linux 里是有可能将整个 PrintK 语句去屏蔽掉，当然这样可能会带来一些问题，这完全取决于系统设计。

另外还可以通过延迟初始化模块的方式，以达到节省系统启动时间的目的。这种操作如果需要去延迟模块的加载时间，可能需要更改 Kernel 的部分。下面提的一种简单的方式，就是只要将 module_init 这样一个函数名替换成 deferred_module_init 就可以了。另外，跟 Uboot 一样，在编译时请将 G 选项去掉。同时也可以通过屏蔽掉系统的调试信息和系统的调试特性，以下举了一些例子。当将这三种调试信息去掉之后，或者说调试方式、调试特性去掉之后，也将有助于提高整个 Linux 的加载时间。请尽可能的使用静态的 IP 地址，如果在系统当不使用以太网，那么可以完全将这个功能屏蔽掉。如果不需要 NFS 功能，建议在 Uboot 的环境变量里设置 IP=off,整个以太网获取 IP 就不会影响到 Linux 的启动时间。另外，针对内存这部分，完全可以通过 mem= 这个选项来设置内存大小的限制，只使用所需要的内存大小，这样的话可以减小 DDR 的初始化时间。

下面介绍优化文件系统的时间的方式。

首先是 BusyBox，可以用 BusyBox 这个工具做出最简单的文件系统，这样的文件系统更容易加载和装载。可以讲 BusyBox 的一个编译选项设置成 static，通过 Static 可以编译出静态的库文件，这样的库文件非常利于加载，可以一定程度上缩短启动时间。但是这样的库文件往往它的容量会比较大，所以需要在系统设计过程当中进行一个权衡。第二种方式是尽量避免使用 Ramdisk 这样一个文件系统，这种文件系统在使用过程当中是需要加载到 RAM 中执行，会花大量的时间。在一般的使用场景当中，这样的文件系统是不会被利用的。所以如果没有特殊的需求，建议屏蔽掉这种文件系统。预链接的方式，就是说在启动过程当中，或者在文件加载之前已经将所有链接的方式、链接的库文件已经做好，那么这样的方式可以有效的缩短启动时间，当然这样的方式又有它自己的一个缺点。当版本或者当库文件发生变更时，这种方式就不再适用。所以这种方式的优化更适用于那种固定的、不需要实时的

去动态加载的这样一个启动方式。下面一个建议是使用 Tmpfs，这是一种非常简单的文件系统，在加载的时候会非常快。另外也可以采用 Split 的文件方式，这种文件方式其实就是分时的加载，可以在第一次加载的时候加载必须的模块，从客户使用角度来讲，客户就认为您的文件系统或者说您的系统已经完全起来了，剩余的一些模块可以在后续的执行过程中慢慢的加载，慢慢的执行。下一个优化方式是通过使用交叉编译器 Strip 这个工具，这个工具可以去掉生成 img 文件里的 Symbols，这样的话可以在一定程度上缩小镜像文件大小，以提高启动时间。另外对于一些静态的硬件，也就是不支持热拔插的硬件，完全可以直接建立设备节点，而不需要通过 Udev 这种方式来动态的建立。同时针对一些不需要用到的初始化脚本，完全可以在初始化脚本里将它去掉，这样也会节省时间。最后提到的 GNU_HASH 这样的一个标志位，完全是为了优化动态连接效应的，这个也会帮助你缩短启动时间。

2.4 如何使用 CCS 去 Debug

本节主要内容是如何去使用 CCS 去 Debug，以及修改和在 AM335x 上去执行 StarterWare 这个软件。

下面介绍一下如何通过 CCS 工具去调试 Uboot。本节需要你以下知识都有所了解，首先熟悉整个 Linux 启动流程（包括 Uboot SPL）；熟悉 TI 的 SDK（包括完整运行了 TI SDK 里面提供的所有脚本），对于 CCS 工具有一定的知识和有一定的了解。

下面我们介绍一下 Sitara Linux SDK 的一些基本知识。

下图提供了可以下载到 SDK 的源码和开发包，以及 CCS 的安装程序的相关内容。

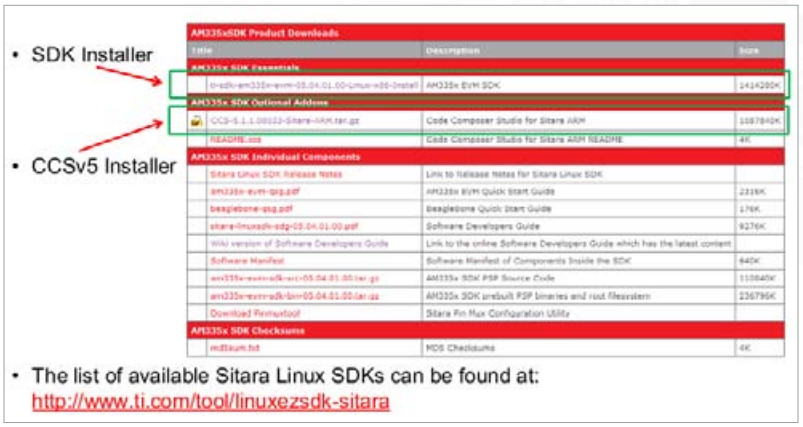


图 2-32

关于 CCS 的使用有几点需要注意，首先 XDS100v2 这个仿真器是不需要任何 License 的，下载并安装了 CCS 软件之后只有 90 天的一个免费试用期，试用期过后，可能需要支付一定的费用。另外，如果将 CCS 安装在 Linux下，那么需要在使用 CCS 的时候，需要获得 Root 权限。

下面简要介绍一下基本的开发环境。

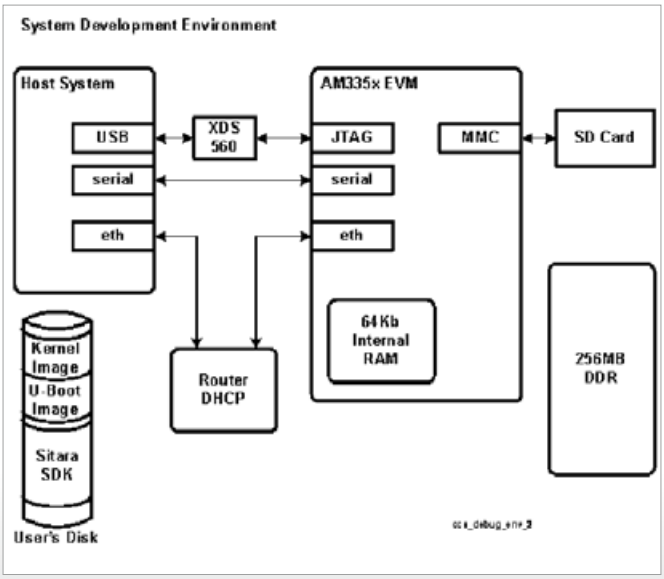


图 2-33

基本开发环境主要包含以下方面，首先 TI SDK 的运行环境是 Ubuntu 10.04，同时需要安装好 TI Linux SDK 以及 CCS。也可以使用 TI 的 SDS560v2 仿真器，或者之前提到的 XSD100v4 仿真器，确保串口终端使用正常。同时在有些场景下还需要使用到以太网的功能。另外也需要准备一张 SD 卡去存储启动镜像，包括 MLO、U-Boot、Kernel 以及文件系统。在进行实验之前，需要了解到 img 存放的位置，X-Loader 或者说 MLO 是存放在内部的 SRAM 里面，而 TI 的 Uboot 和 Kernel 是放在 DDR 里面运行，下图中已经将内部 SRAM 的地址和 DDR 的地址标注出来了。

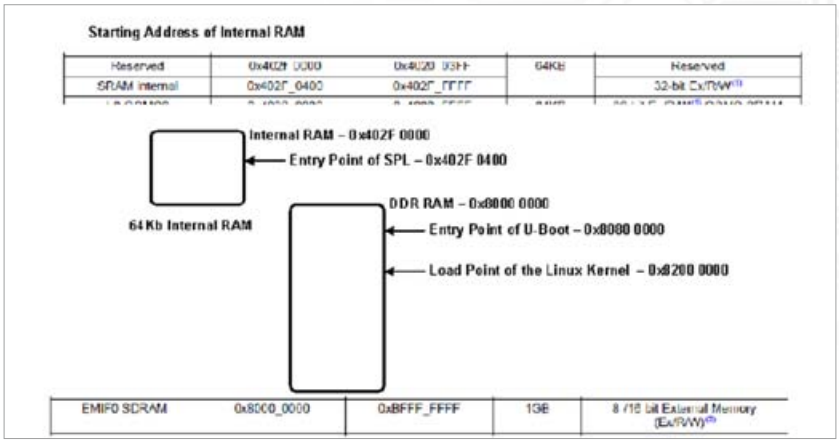


图 2-34

下面我大概介绍一下 TI Uboot 调试的一些基本信息。

首先需要熟悉所有镜像加载的位置，在 TI 现在提供 DSDK 里，MLO 或者 X-Loader 加载在 0x402F0400 这个位置上 (也就是 SRAM 的地址)。而 Uboot 以及 Kernel 是加载在 0X8010000 这个地址上。用户可以根据自己系统的需求 (或者说自己定义的需要)，可以自己更改这个地址。具体的配置地方，可以访问 include/configs/am335x_evm.h 这里去进行更改。确定完地址信息之后，需要创建一个 CCS 工程，并将这个工程指向 Uboot 在 SDK 里的路径。CCS 可能会花一定时间，因为它需要去扫描整个 Uboot 的架构。结束之后，需要创建一个目标配置文件，目标配置文件主要是针对 AM335x 的配置文件，需要选择相应的仿真器类型。同时插上 SD 卡，并上电 EVM 板，需要保证 SD 卡里包含了必要的启动信息和启动的镜像文件，上电之后并执行之前配置的目标配置文件。当 CCS 连接到 AM335x 上时，AM335x 上运行的程序将停止。首先需要将整个 AM335x 的工作模式更改为 ARM 模式，其次需要加载 MLO 或者 X-Loadre 的景象。如果 EVM 板是从 SD 卡启动，那 CCS 不需要加载任何东西。但是如果 EVM 板上没有 SD 卡，或者是没有其他的启动设备，那需要将 X-Loader 或者 MLO 的二进制文件通过 CCS 加载进去。同样，在 Uboot 里，如果 AM335 是通过 SD 卡或者说其他的启动设备启动，那么您只需要将 Symbol 通过 CCS 载入进去就可以进行下一步的调试。但是如果没有使用任何的启动设备，那需要通过 CCS 去加载 ELF.img 文件。

2.5 如何使用 CCS 调试 Linux

2.5.1 实验前准备

Ubuntu 10.04

Linux 版 CCSv5

SDK：本文使用的是 SDK-05.05.00.00

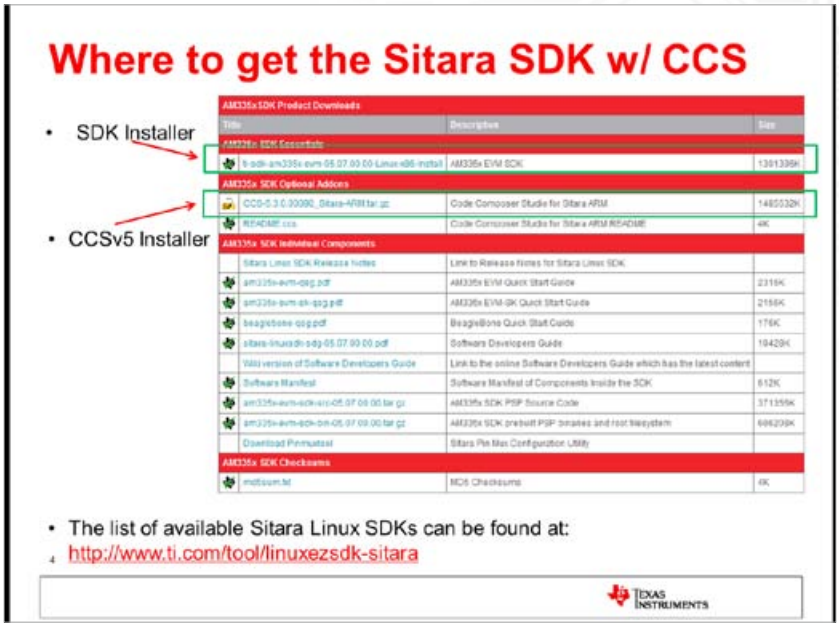


图 2-35

2.5.2 创建工程

需要在 CCS 中创建一个 makefile 工程，并将 Linux 源码导入。具体步骤为：

（1）打开 CCS

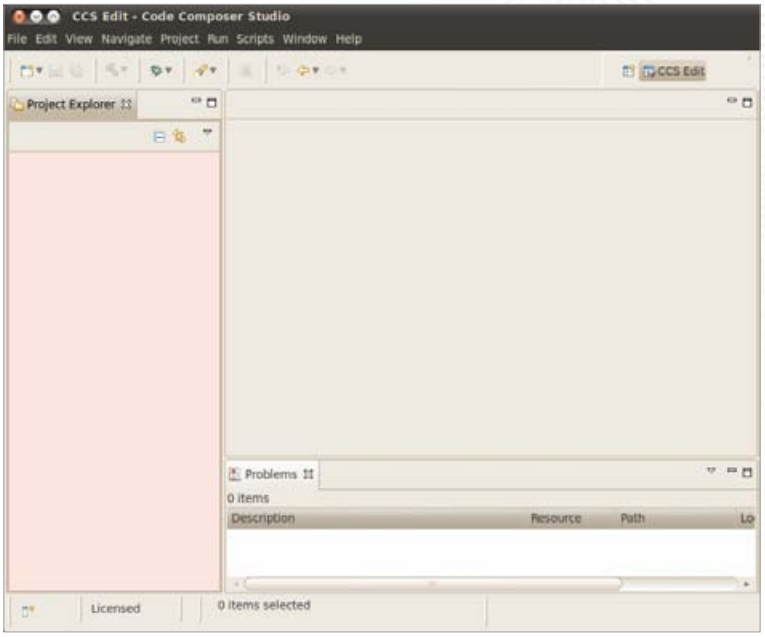


图 2-36

（2）创建 makefile project

选择 File -> New -> Project
展开 C/C++ 分类
选择 Makefile Project with Existing Code

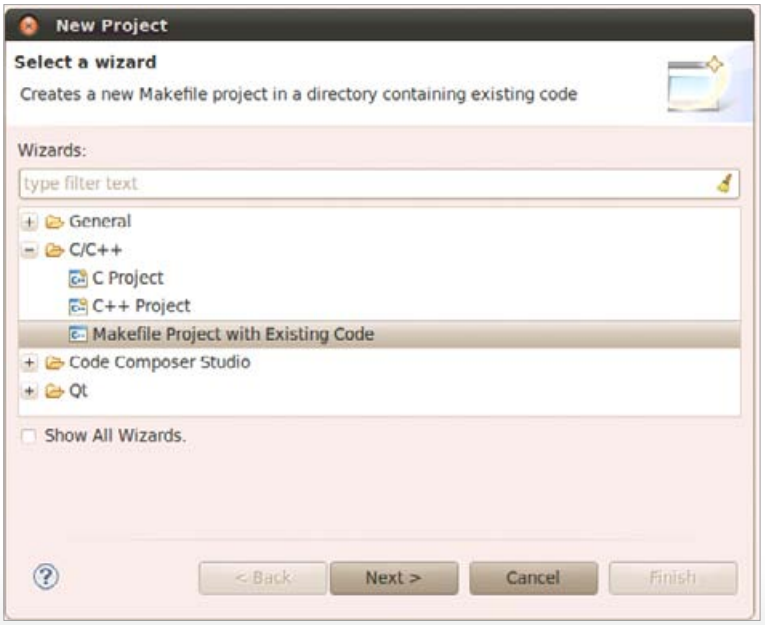


图 2-37

（3）导入 Linux

按如下步骤导入 Linux：

点击 Browse

找到 Linux 源码目录。如 /home/sitara/ti-sdk-am335x-evm-xx.xx.xx.xx/board-support/linux-xx.xx.xx

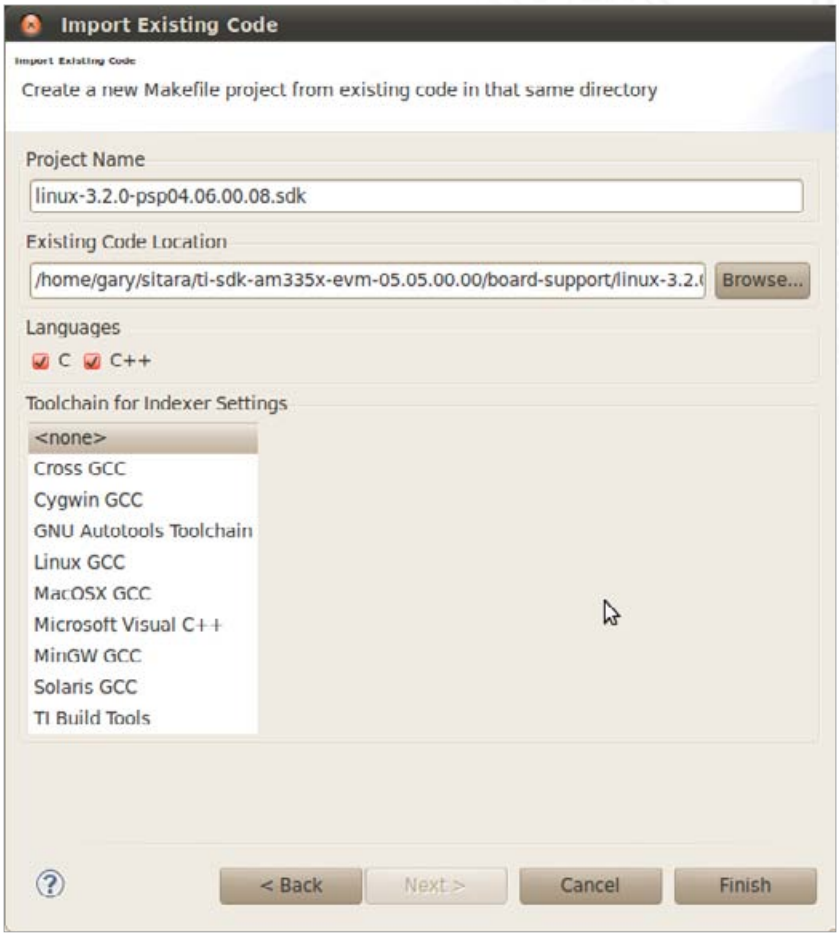


图 2-38

当你导入工程成功后，会看到如下界面：

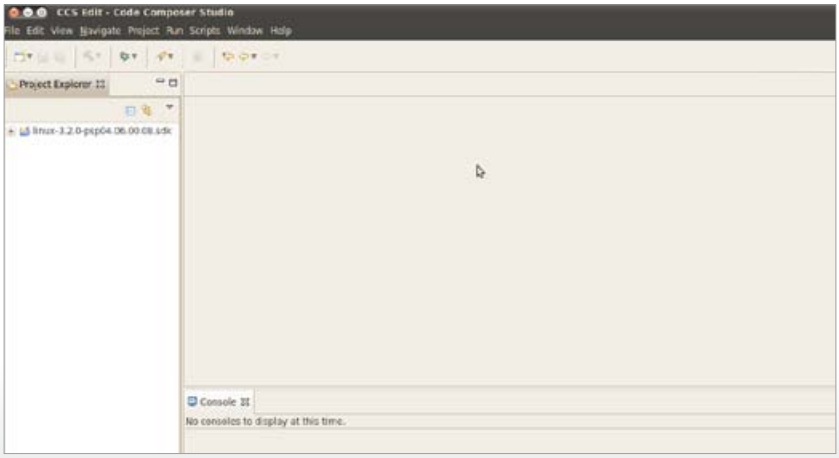


图 2-39

2.5.3 配置仿真器文件

以 XDS100V2 仿真器为例，

选择 File -> New -> Target Configuration File

在 New Target Configuration 对话框的 file name 中输入仿真器配置文件的文件名，如 AM335x-TC

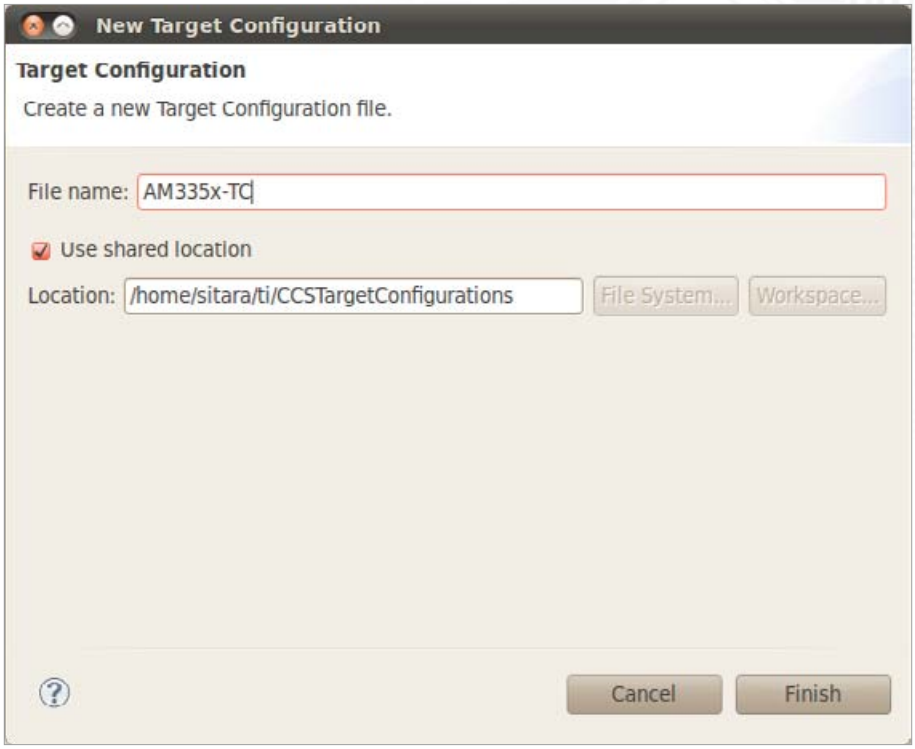


图 2-40

点击 Finish 后，会弹出如下的编辑窗口，在 Connection 中选择”Texas Instruments XDS100v2 USB Emulator”仿真器，并在 Board of Device 中选择 AM335x。之后点击 Save 即可。

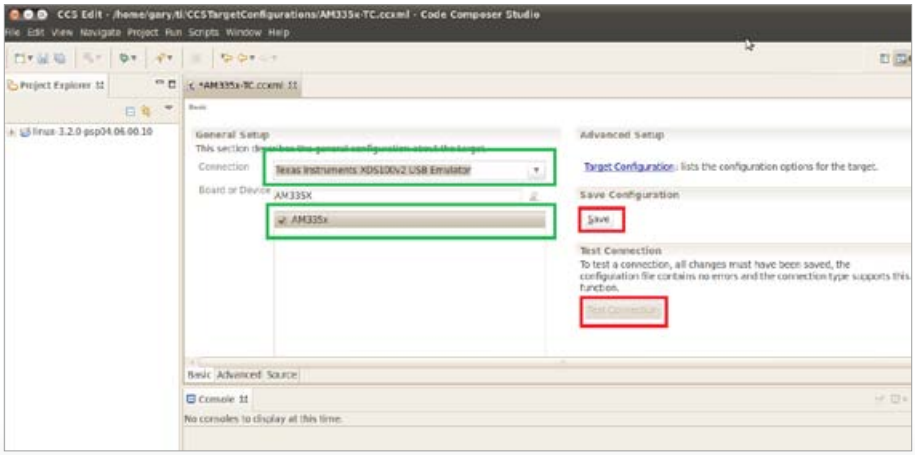


图 2-41

Save 之后，可以点击 Test Connection，如看到如下窗口出现”The JTAG DR

Integrity scan-test has succeeded.”表示仿真器连接正常。

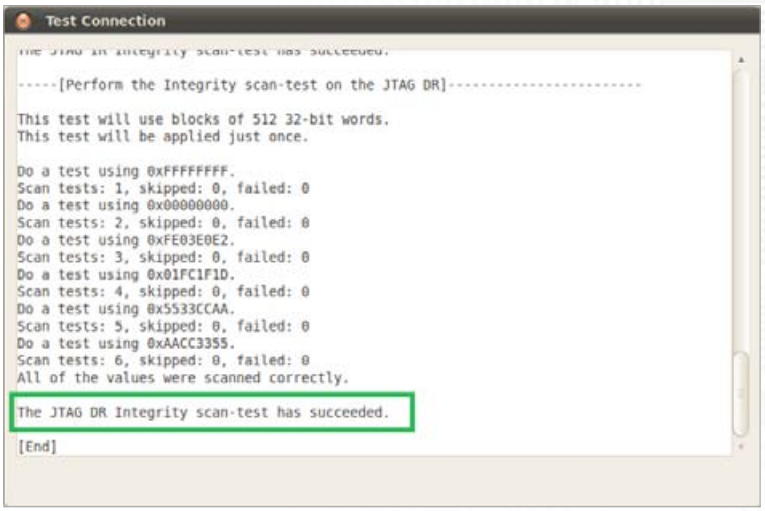


图 2-42

2.5.4 编译 Linux

在调试 Linux 前，需要编译 Linux，并加上相关调试选项。

设置环境变量，指定交叉编译工具的路径

在 terminal 中，输入

```
export PATH=$PATH:/home/sitara/ti-sdk-am335x-evm-xx.xx.xx.xx/linux-devkit/bin
```

配置 Linux，加上相关调试选项

按如下命令进行：

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- mrproper
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- tisdk_am335x-evm_defconfig
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- menuconfig
```

执行上述命令之后，会出现配置窗口，按如下指引进行配置。

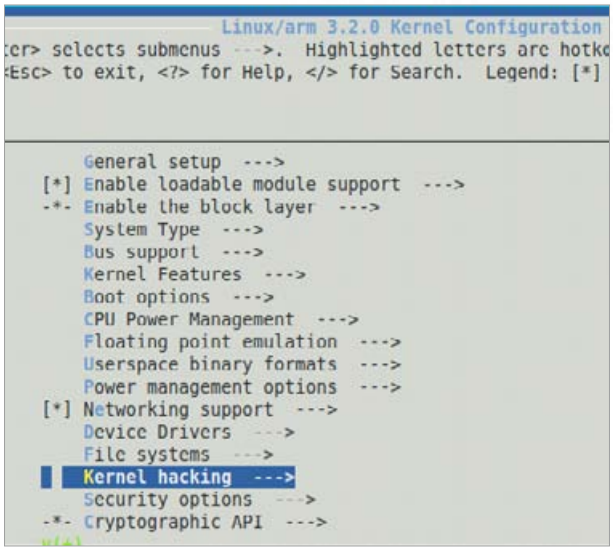


图 2-43

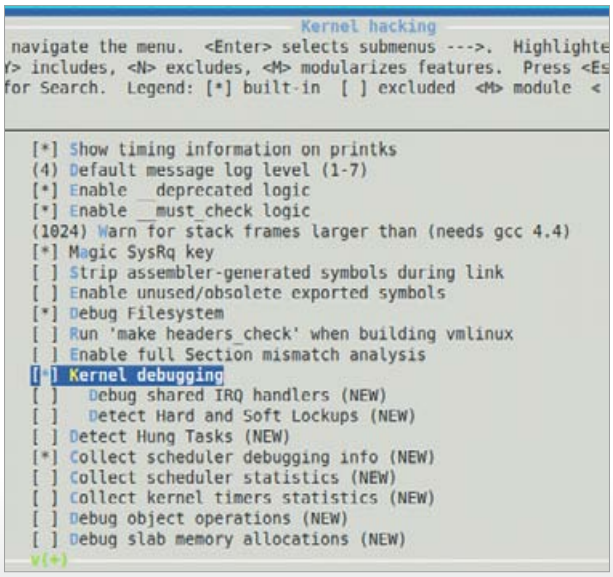


图 2-44

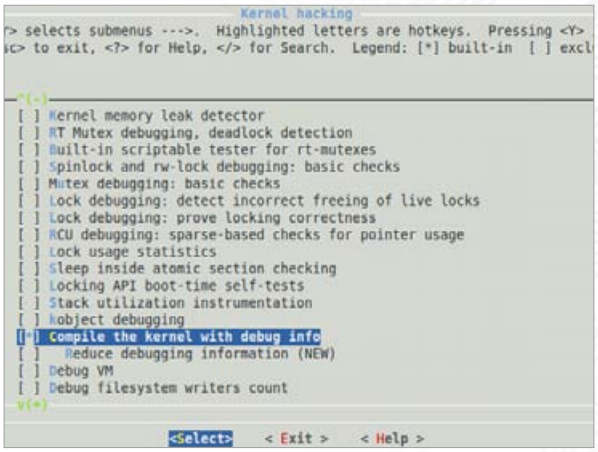


图 2-45

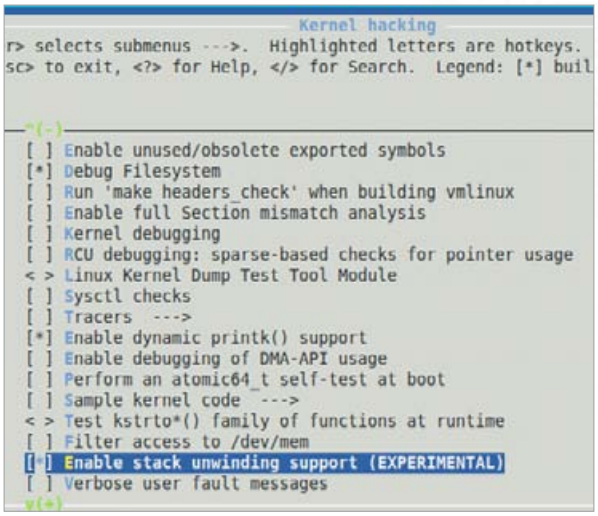


图 2-46

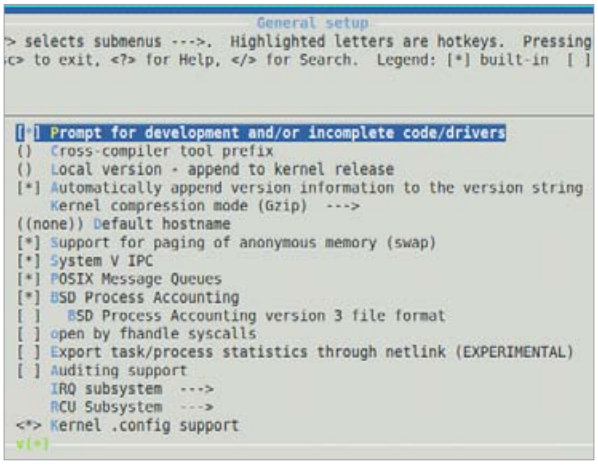


图 2-47

配置完成退出时，注意保存。

编译 Linux

使用如下命令进行 Linux 编译

make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- uImage

2.5.5 调试 Linux

将上一步编译好的 uImage 拷贝到 SD 的 boot 分区中。EVM 板自带的 SD 是默认分好区，并在 boot 分区中有 MLO 和 uImage，在 rootfs 中有文件系统。

使能仿真器。

目标板上电，在使能仿真器之前，确认 SD 未插在目标板上。

切换 CCS 窗口，点击 View -> Target Configurations，展开 User Defined，选中 AM335x-TC 配置文件，右键并选择 Launch Selected Configuration。

CCS 窗口会切换到如下界面。

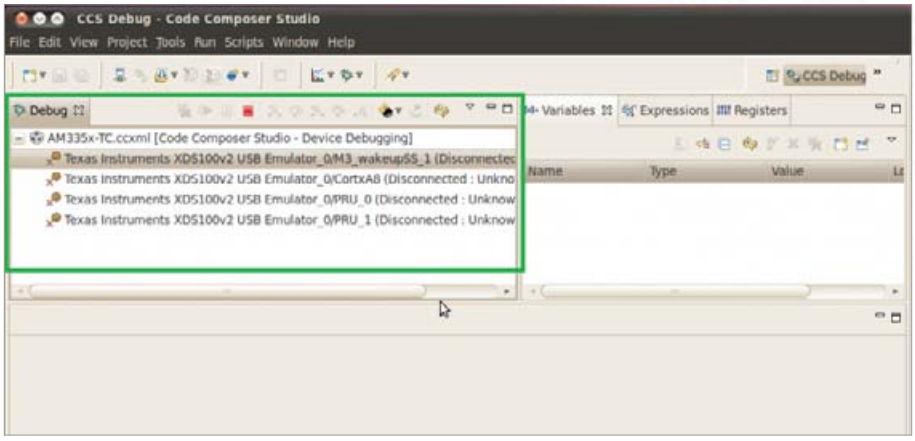


图 2-48

选中 CortexA8，右键并选择 Connect Target，仿真器会连接芯片。连接成功后显示如下。

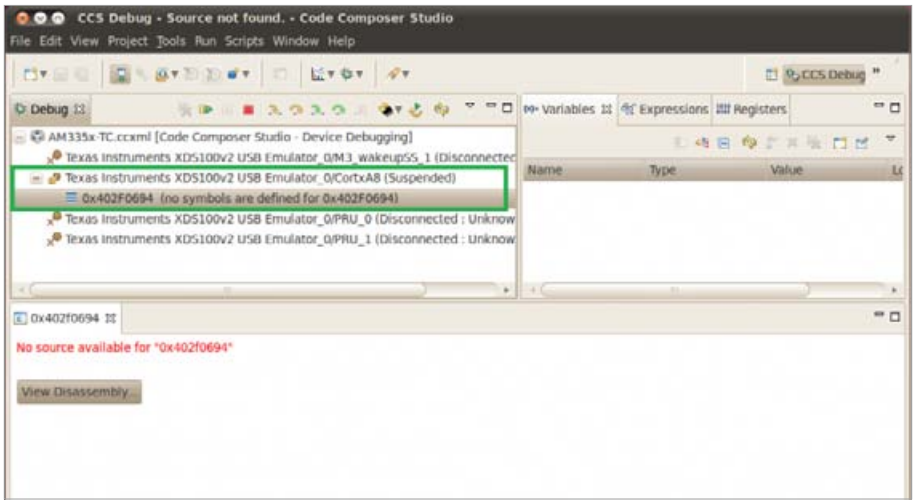


图 2-49

导入符号表

将之前准备好的 SD 卡插入目标板。这里需要用到串口调试窗口，切换到 minicom 窗口。Reset 目标板，在 u-boot 启动后，按 PC 空格键让 u-boot 停止 boot。

```
CCCCCCCC
U-Boot SPL 2011.09 (Jun 08 2012 - 20:41:25)
Texas Instruments Revision detection unimplemented
OMAP SD/MMC: 0
reading u-boot.img
reading u-boot.img


U-Boot 2011.09 (Jun 08 2012 - 20:34:40)

I2C:   ready
DRAM:  256 MiB
WARNING: Caches not enabled
Found a daughter card connected
NAND:  HW ECC Hamming Code selected
256 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - bad CRC, using default environment

Net:   cpsw
Hit any key to stop autoboot:  0
U-Boot#
```

图 2-50

点击黄色 pause 按键，suspend 芯片
选择之前编译的 Linux 中的 vmlinux 文件，导入符号表
点击 CCS 的 Run -> Load -> Load Symbols...
点击 Browse，找到 Linux 源码根目录，如下
/home/sitara/ti-sdk-am335x-evm-xx.xx.xx.xx/board-support/linux-xx.xx.xx/

点击右下角的窗口，选择 All files(*)，找到 vmlinux 并选中。

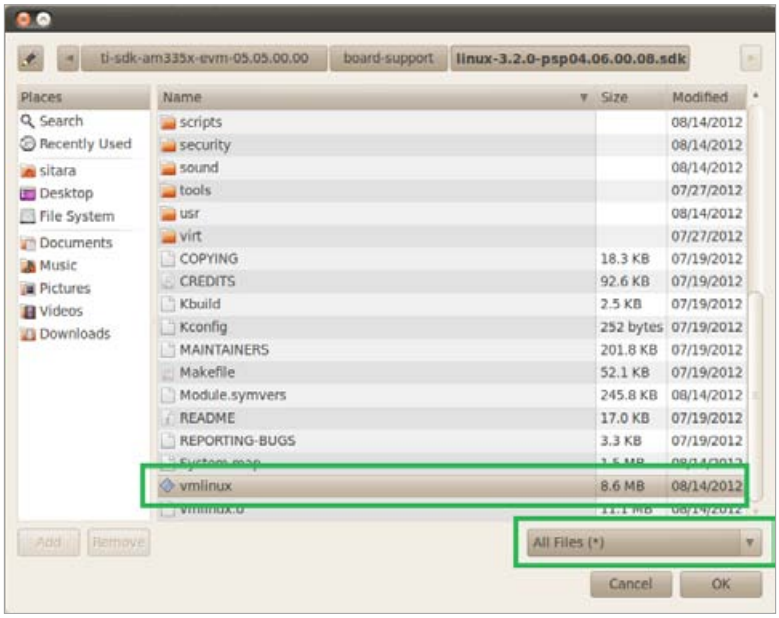


图 2-51

不需要设置 offset，直接点击 OK。

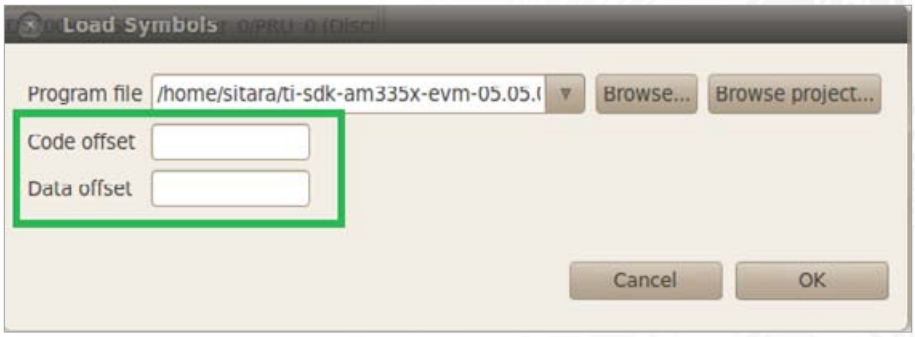


图 2-52

设置硬件断点

选择 File -> Open File

打开以下路径

/home/sitara/ti-sdk-am335x-evm-xx.xx.xx.xx/board-support/linux-xx.xx.xx/arch/arm/mach-omap2

选择 文件 board-am335xevm.c
在函数 am335x_evm_setup 处设置硬件断点
鼠标移至设置断点的代码行处，邮件选择 Breakpoint->Hardware Breakpoint

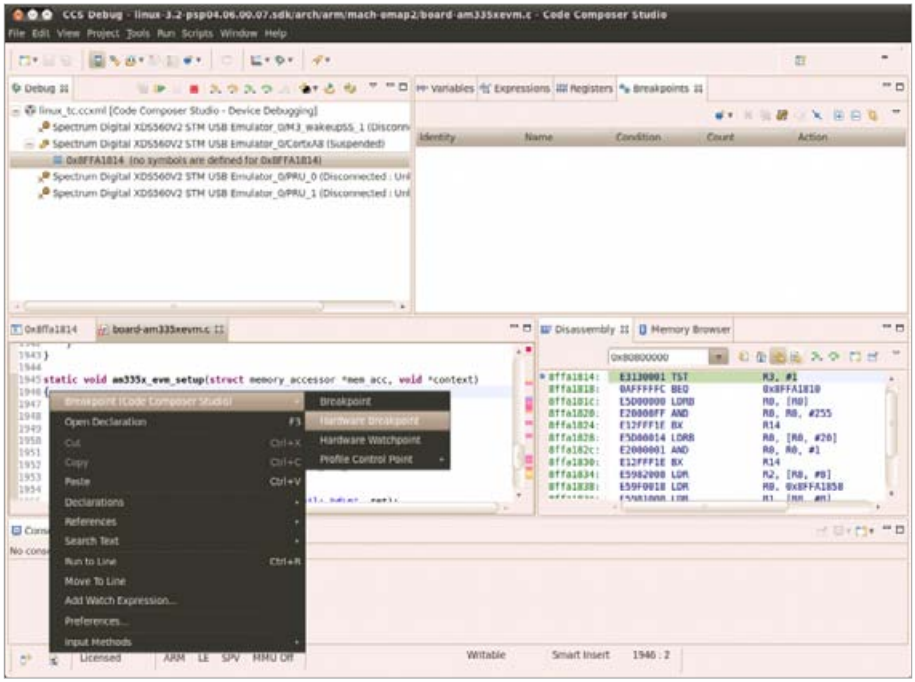


图 2-53

设置完断点后，可以看到 Debug 窗口有关于断点的信息。

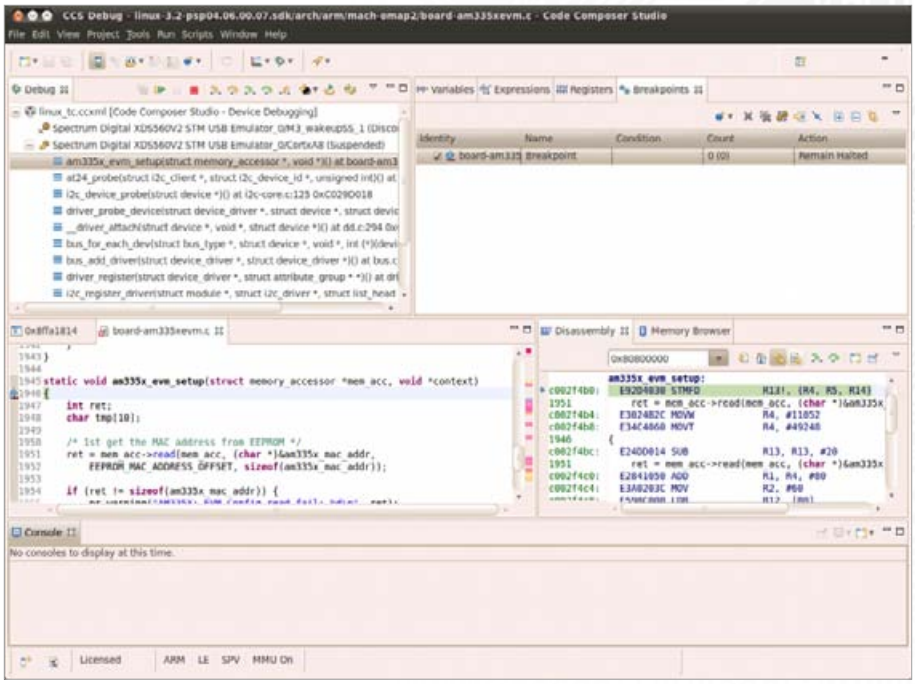


图 2-54

调试

点击 Debug 上方的绿色按钮运行，然后在 minicom 处 u-boot 命令行下输入”boot”即可，也可使用 tftp 将之前编译好的 uImage 传到内存处启动，而不需要通过 SD 卡拷贝的方式。当内核启动后，会发现停到上一步设置的断点处。

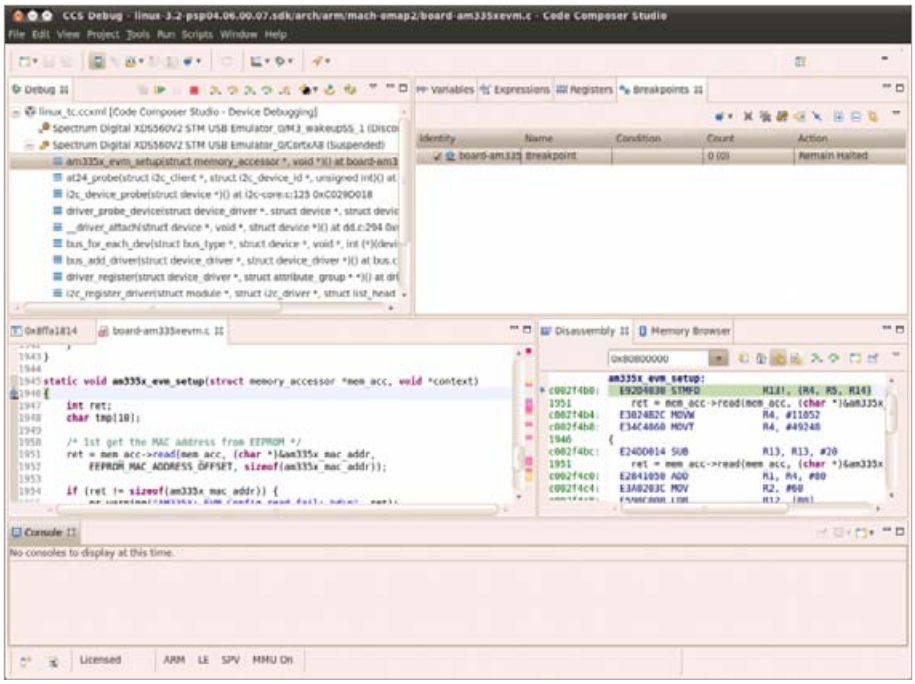


图 2-55

至此，可以通过 F5 进行单步执行，或者通过 CCS 查看寄存器及 memory 等操作。

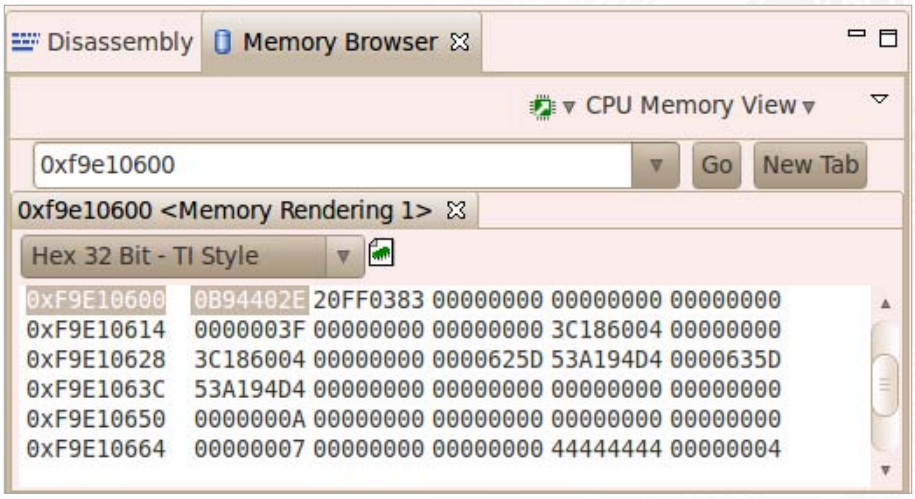


图 2-56

详细资料可以参考 TI Wiki

http://processors.wiki.ti.com/index.php/Sitara_Linux_Training:_uboot_linux_debug_with_ccsv5

2.6 使用 IAR 连接 J-Link 调试 AM335x

随着 TI Sitara ARM AM335x 使用的普及，越来越多的用户会有对 AM335x 进行软硬件仿真调试的需求，而 TI 提供了 CCS 软件及相应的 XDS 系列仿真器，可以对软件、硬件进行仿真、调试。与此同时，也会有不少用户有使用 IAR 与 J-Link 进行仿真调试的想法。所以在本次讲座中，德州仪器半导体有限公司 ARM MPU Sitara 产品线的技术支持工程师刘靖伟将对如何使用 IAR 与 J-Link 调试 Sitara AM335x 进行详细的介绍。

德州仪器半导体有限公司 ARM MPU Sitara 产品线的技术支持工程师刘靖伟。

首先，先介绍一下主机环境的需求。

由于本次示例将使用 IAR 通过 J-Link 仿真器对 Sitara AM335x 进行调试，所以，在 PC 主机上，需要安装最新版本的 IAR 以及 J-Link 的驱动，确保 IAR 和 J-Link 的正常运行。

对于 AM335x 的平台，我们选择的是 AM335x GPEVM 评估板。本次示例所使用程序，源自德州仪器官方所提供的 StarterWare 开发包— uartEcho 工程。StarterWare 开发包的下载地址，可在德州仪器官方网站上找到。

接下来分为四个部分详细介绍如何使用 IAR 与 J-Link 对 AM335x 进行 Debug：

2.6.1 IAR 软件打开项目工程—uartEcho

第一步，打开 IAR Embedded Workbench 软件，准备进行工程文件的导入。

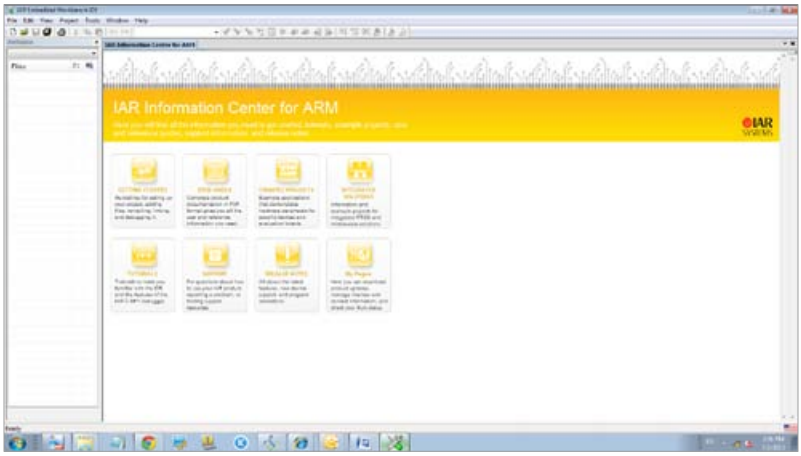


图 2-57

在 Project 选项中选择 Add Existing Project 选项卡，这时，会弹出工程文件选项卡。

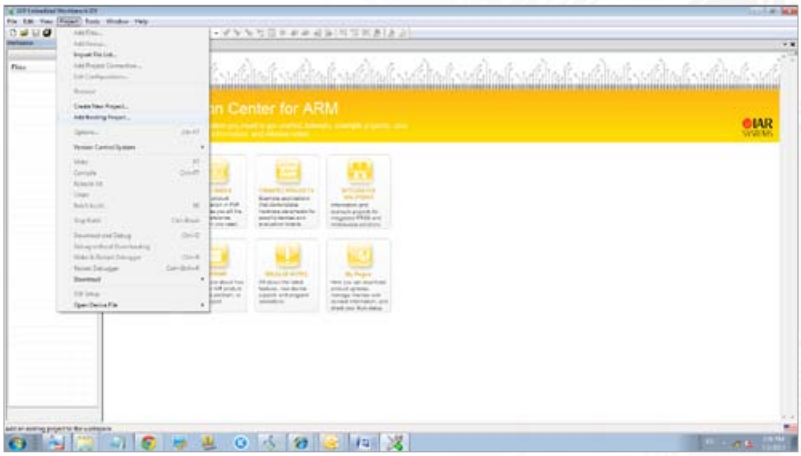


图 2-58

选择我们将要进行的演示的 uartEcho 工程文件，该工程文件位于 StarterWare_02_00_00_07\build\armv7a\ewarm\am335x\evmAM335x\uart 路径下。

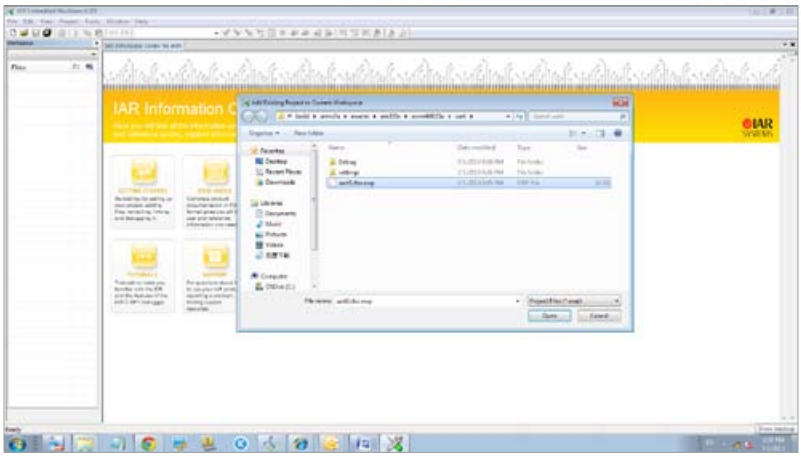


图 2-59

工程文件加载完成后，我们可以看到在当前的 Workspace 下，uartEcho 工程已加载完毕。

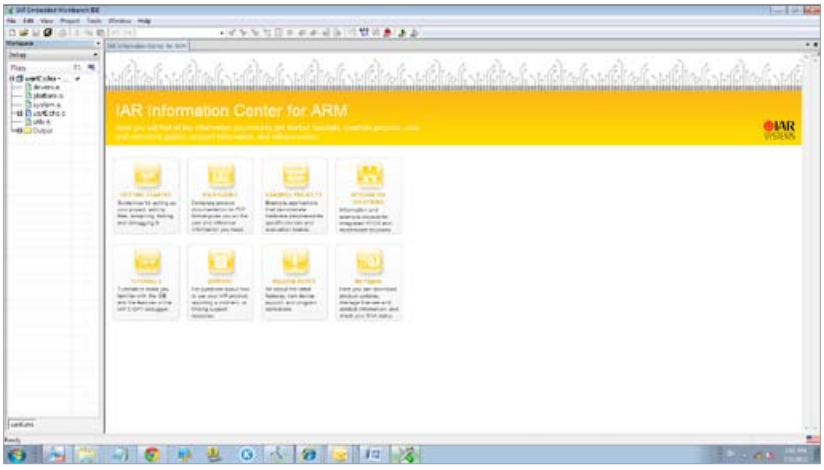


图 2-60

2.6.2 对 uartEcho 工程中所需要使用的 lib 库文件进行编译

Project 选项中选择 Add Existing Project 选项卡。

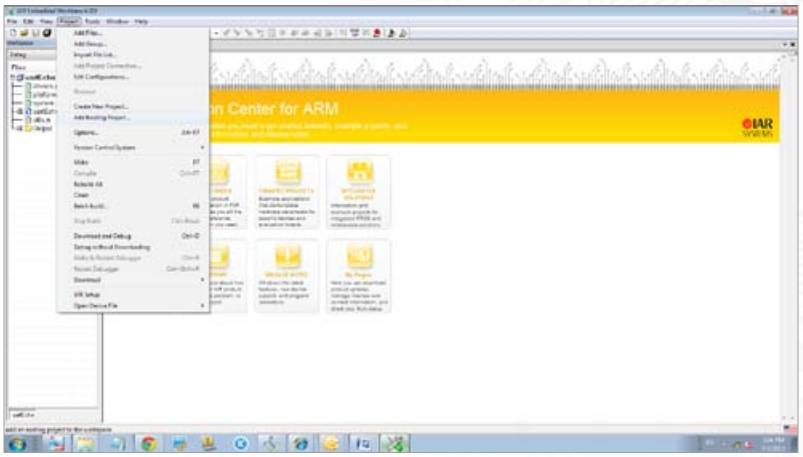


图 2-61

导入 drivers.ewp 工程文件。

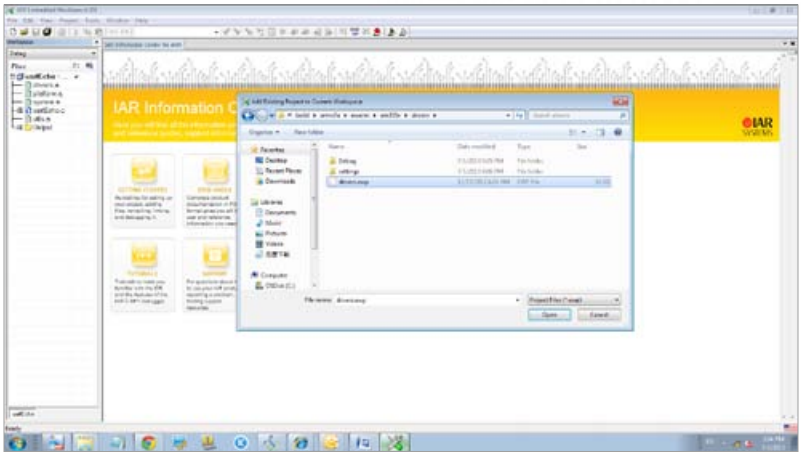


图 2-62

Driver.ewp 工程加载完毕。

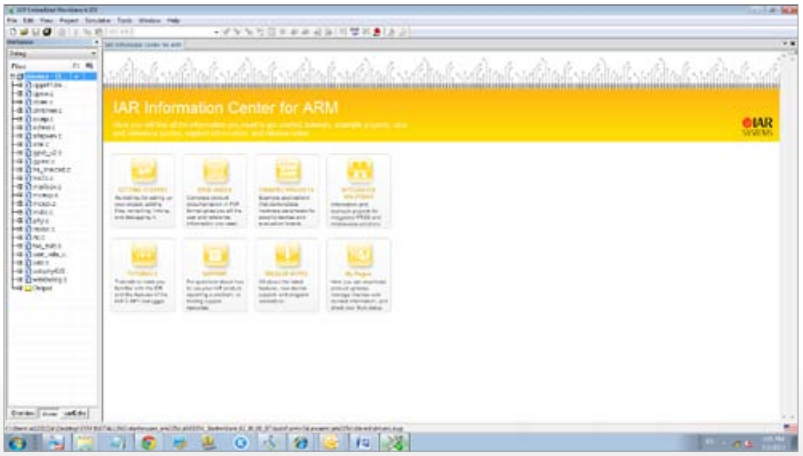


图 2-63

在 File 选项卡中选择 Save Workspace 选项对整个工作站进行保存。

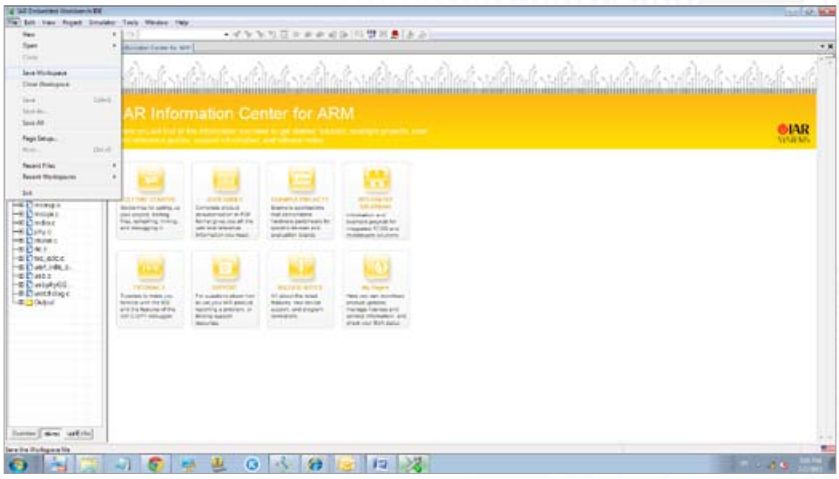


图 2-64

命名工作站保存文件，在示例中为 uart_IAR。

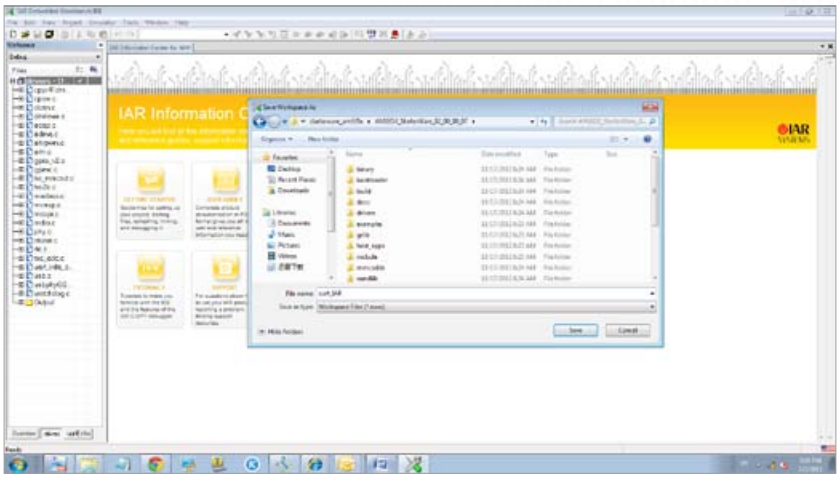


图 2-65

成功保存工作站后可以看到 IAR 工作站名已经更新。

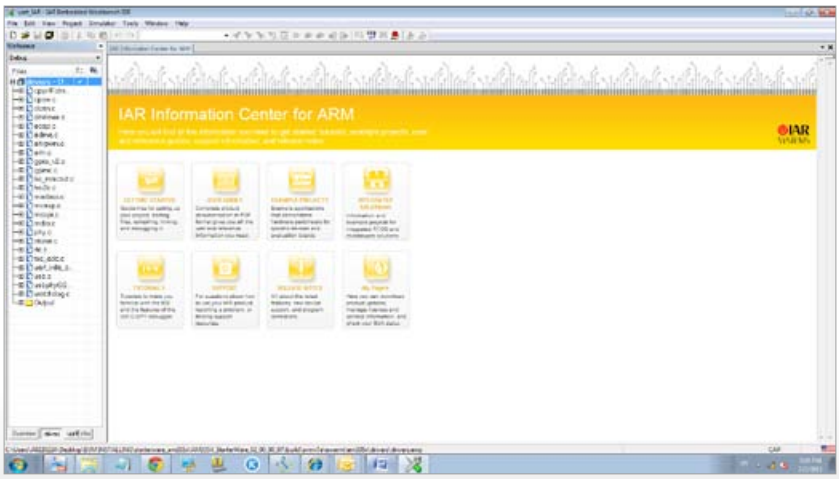


图 2-66

选择 Project 选项卡中的 Edit Configurations 选项对编译选项进行配置。

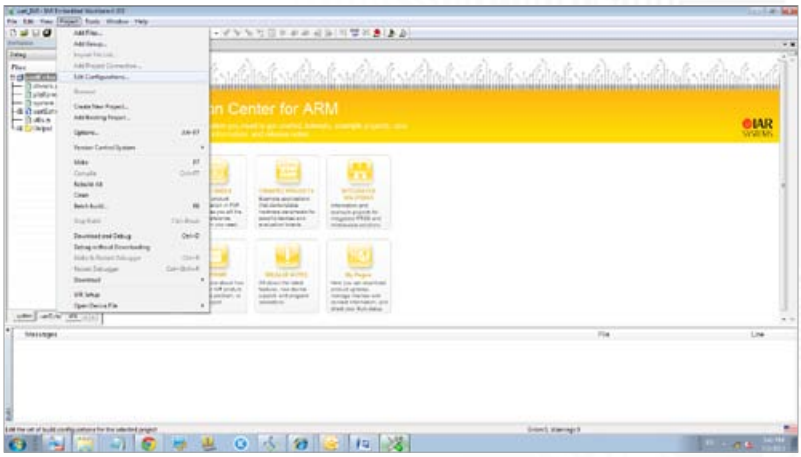


图 2-67

这里我们选择 Debug 作为编译选项。

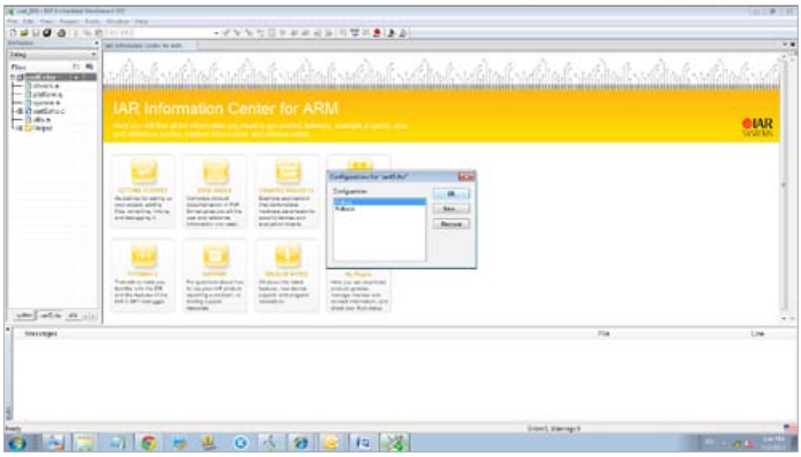


图 2-68

右键点击刚刚导入的 drivers 工程，选择 rebuild all，编译该工程。

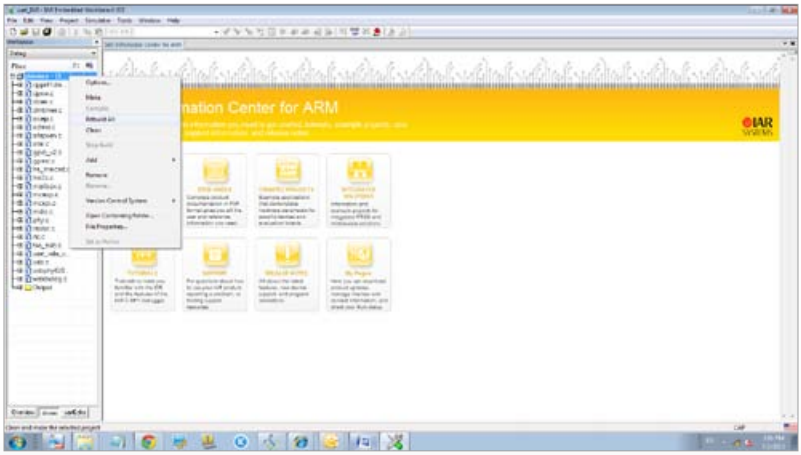


图 2-69

编译完成，无错误信息提示后，drivers.a 库文件成功生成。

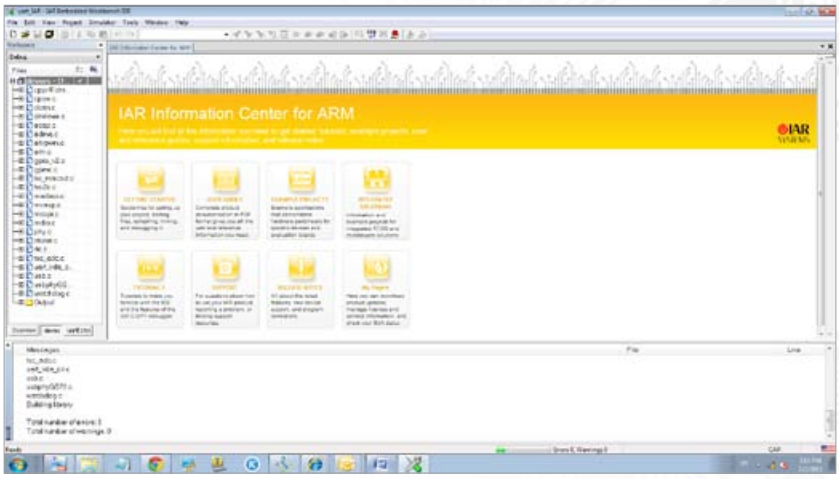


图 2-70

Project 选项中选择 Add Existing Project 选项卡。

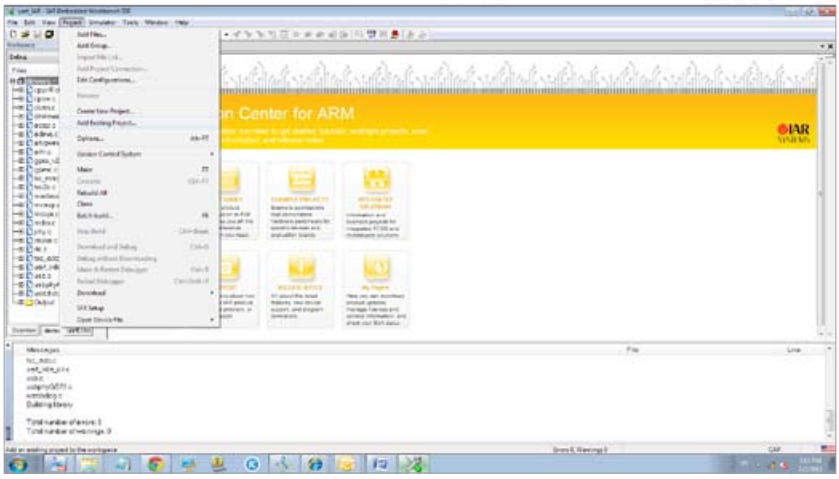


图 2-71

导入 Platform.ewp 工程文件。

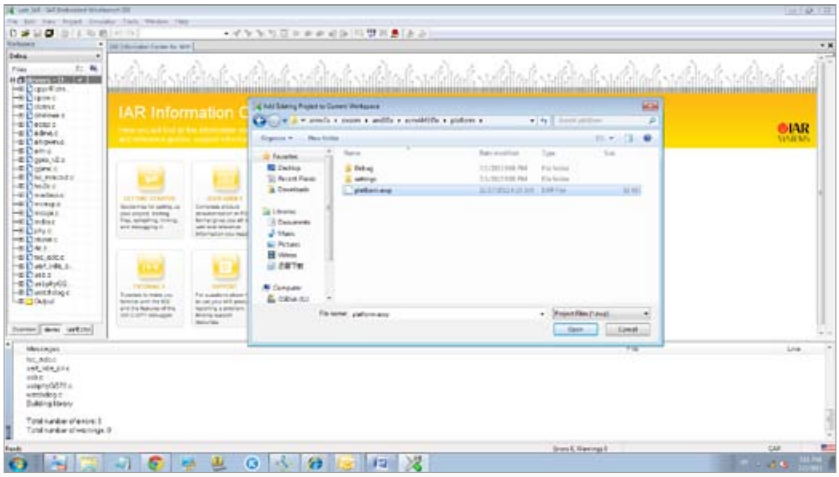


图 2-72

使用 rebuild all 命令，重新编译该工程。

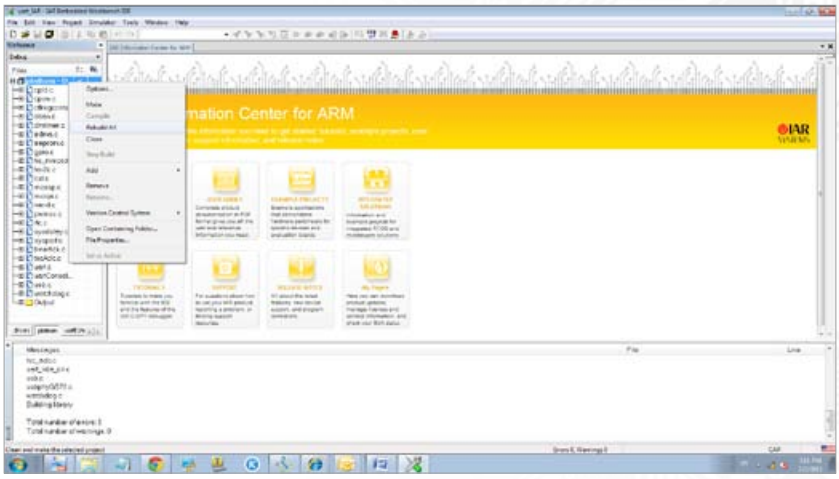


图 2-73

导入 utils.ewp 工程文件。

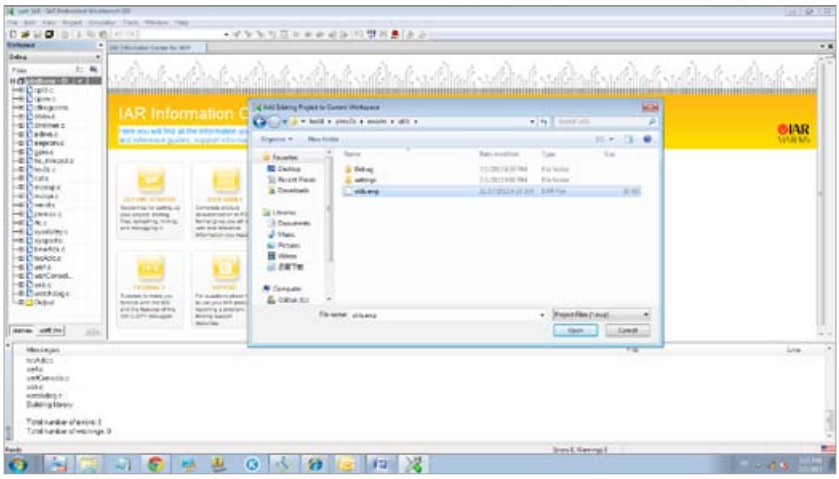


图 2-74

使用 rebuild all 命令，重新编译该工程。

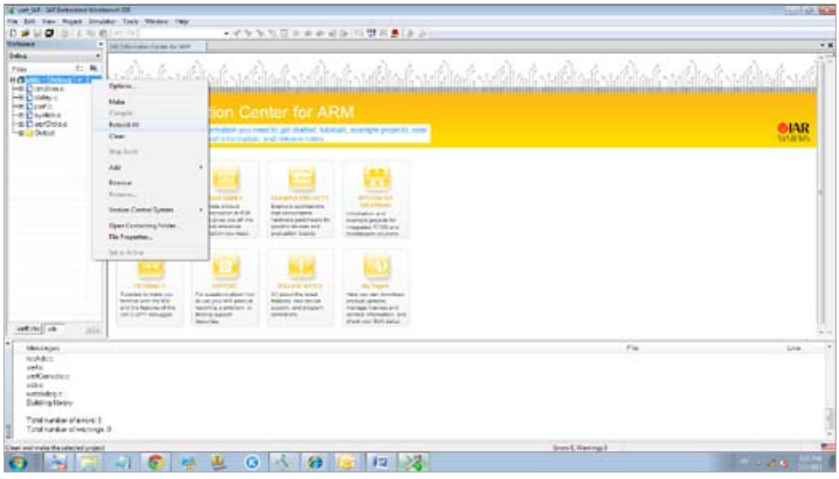


图 2-75

导入 system.ewp 工程文件。

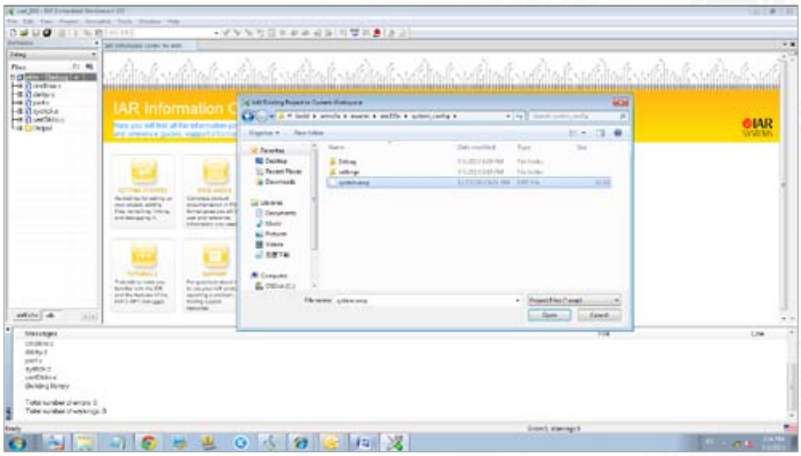


图 2-76

使用 rebuild all 命令，重新编译该工程。

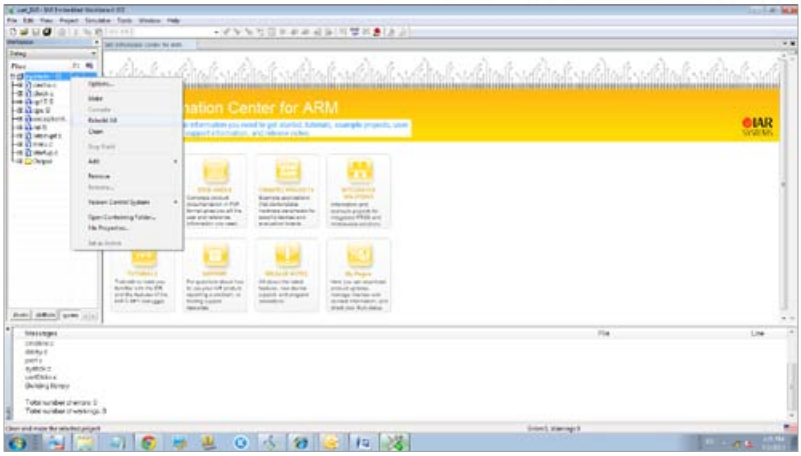


图 2-77

2.6.3 完成库文件的编译后，进行 uartEcho 工程文件的编译

选择 uartEcho 工程，并右键点击，在选项卡中选择 Rebuild All，进行工程文件的编译。

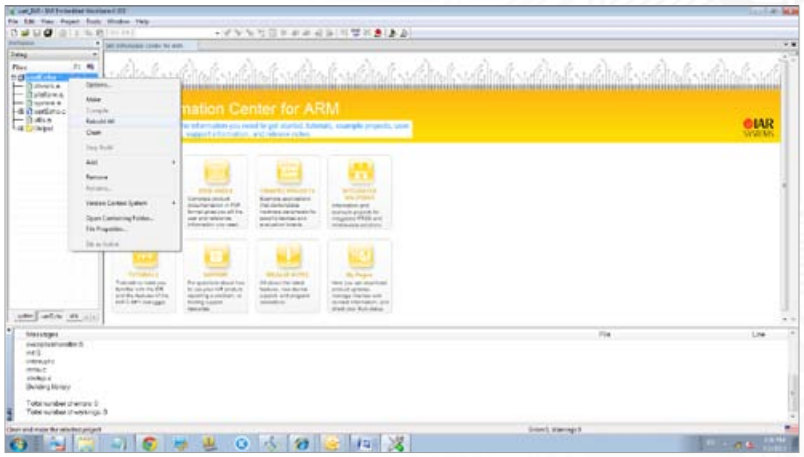


图 2-78

成功编译 uartEcho 工程，生成 uartEcho.out 文件。

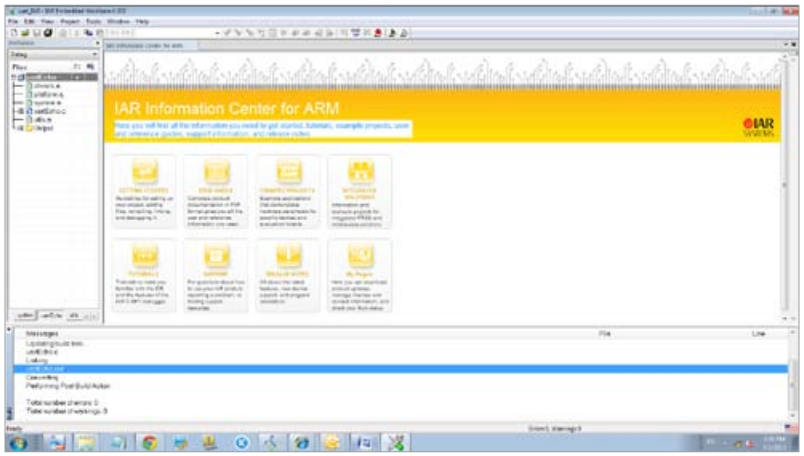


图 2-79

2.6.4 配置并调试 uartEcho 工程

右键点击 uartEcho 工程，选择 Options 选项卡。

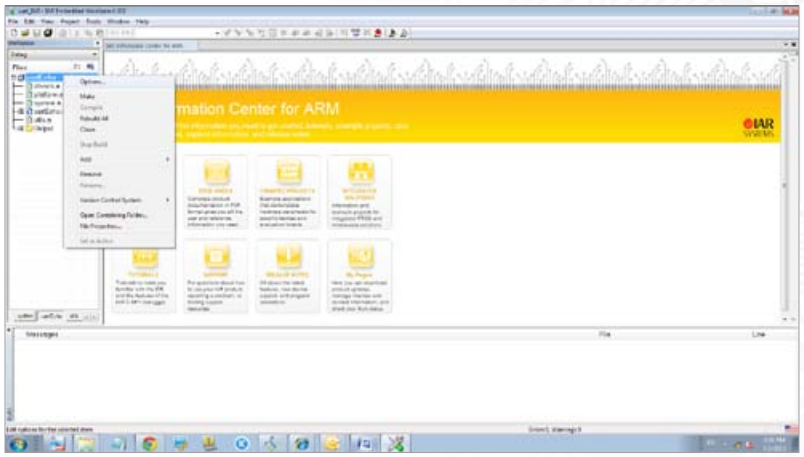


图 2-80

在 General Options 选项卡中 Device 选择 TexasInstrument am3359。

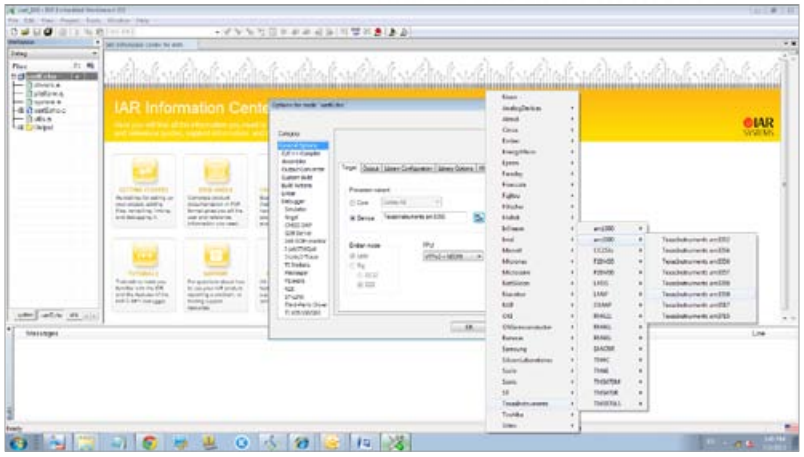


图 2-81

在 Debugger 选项卡中选择 J-Link/J-Trace。

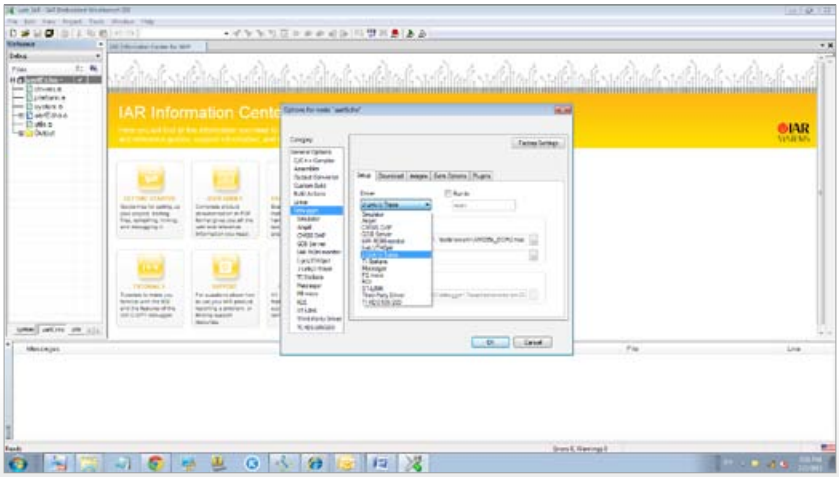


图 2-82

在 Setup macros 选项卡中导入下载的 AM335x_DDR2.mac 配置文件。

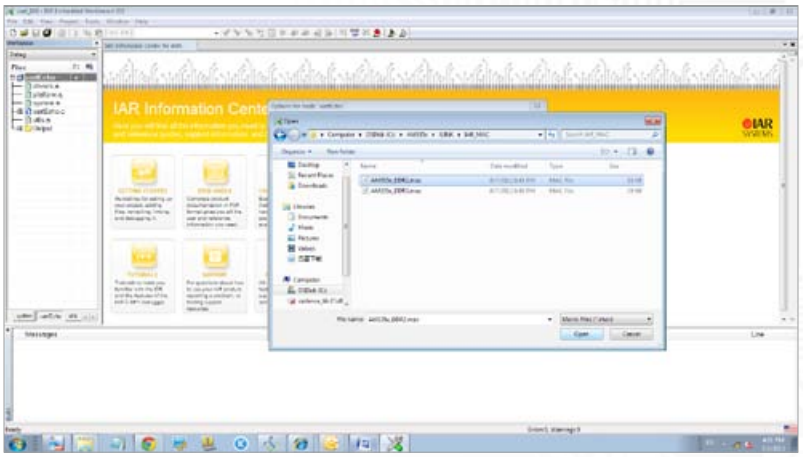


图 2-83

完成以上配置后，选择 OK 确认配置。

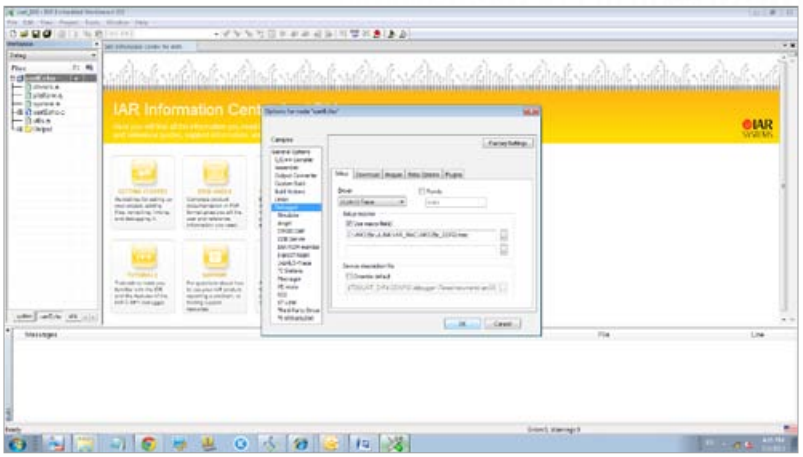


图 2-84

在 Project 选项卡中选择 Download and Debug 选项。点击后系统会自动下载并运行 uartEcho 工程。

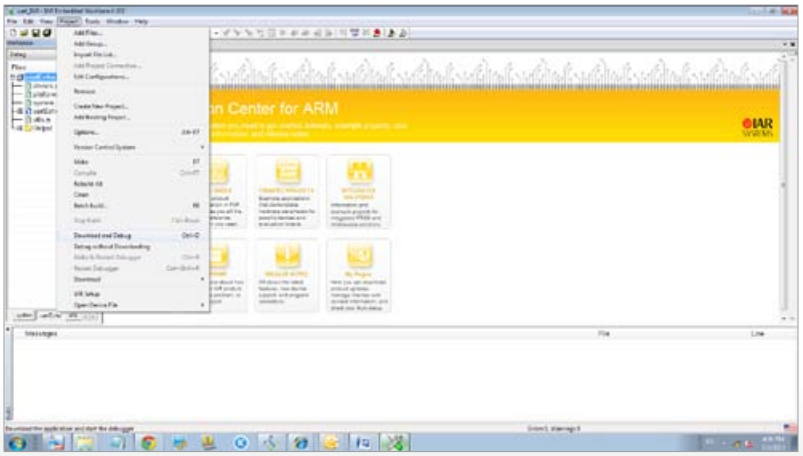


图 2-85

跳入到程序入口，可以查看到 pc 指针。

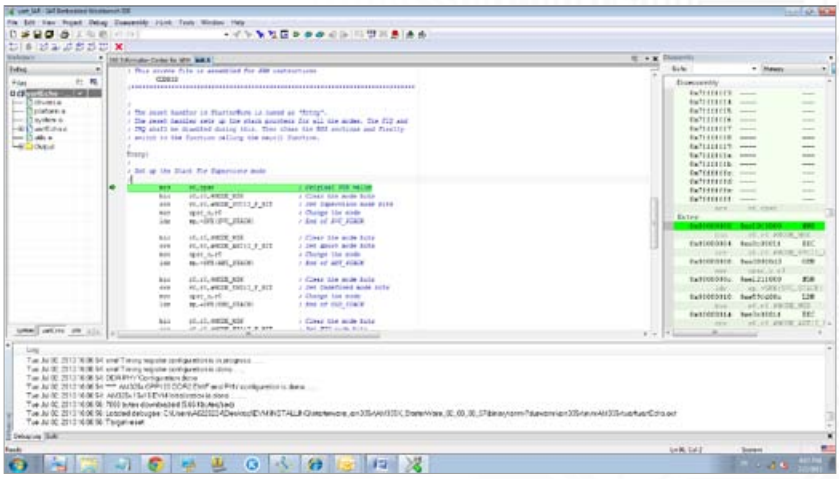


图 2-86

在源文件 uartEcho.c 中设置断点。

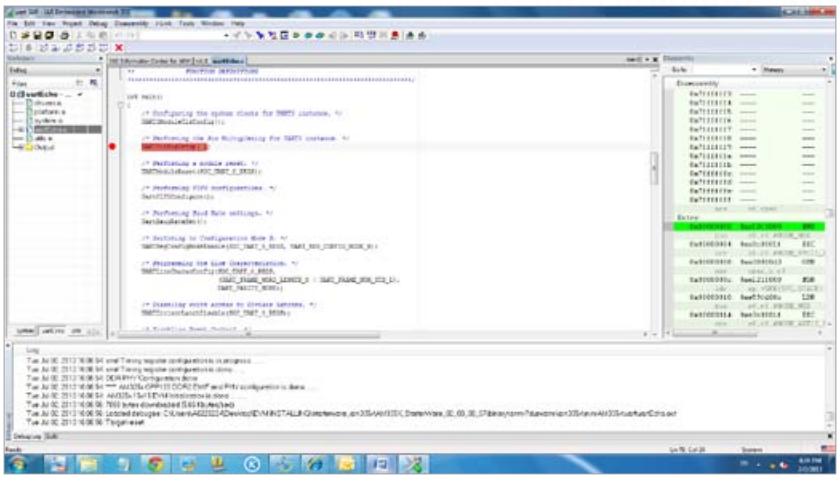


图 2-87

在 Debug 选项中选择 go。

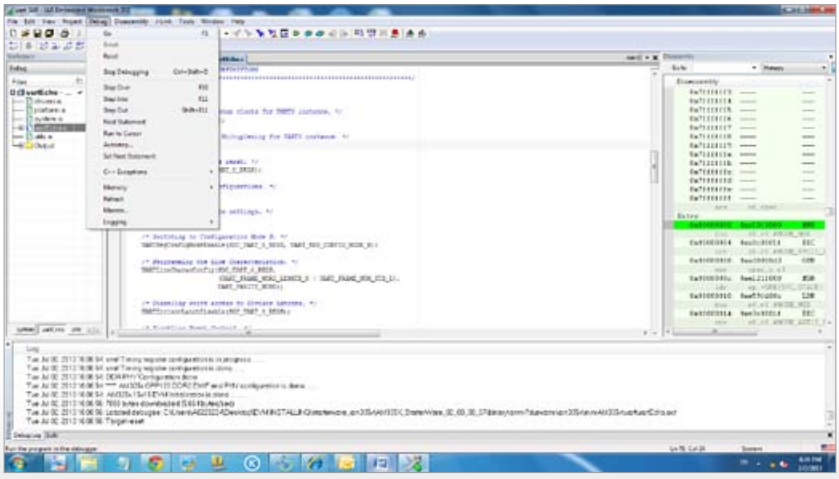


图 2-88

程序会运行，指针会自在断点处停止。

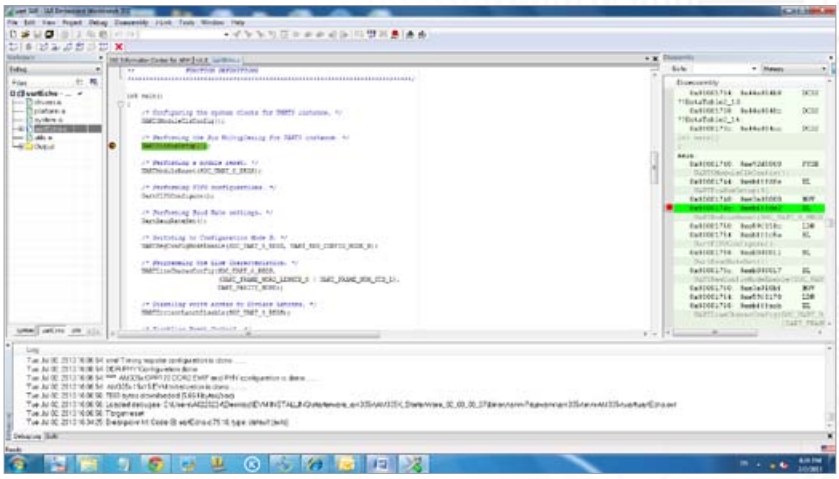


图 2-89

在 View 选项卡中选择 memory，此处在 Go to 后面的窗口输入地址即可查看寄存器地址。

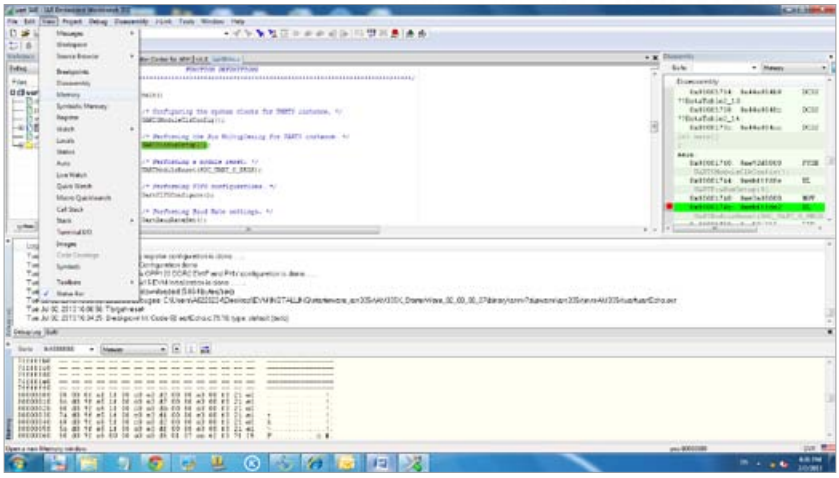


图 2-90

2.7 基于 AM335x 的工业自动化应用

近些年，随着工业通 信技术的飞速发展，在工业自动化控制体系中，信息交换控制技术已广泛运用于工业自动化控制的各个层面：设备层、控制层、管理层。在这个系统中，工业通信的 高效性、鲁棒性是整个自动化控制的核心，是保障高效生产力的关键，同时也对工业控制的性能提出了更大的挑战。

AM335x 立足于 ARM Cortex A8 处理器的高性能，以其丰富的片上资源，多种操作系统的支持，以及独有的逻辑可编程实时单元（PRU 模块），在工业自动化控制中，有着相当广泛的应用。在 Sitara AM335x 中，我们实现了从 300MHz 到 1GHz 工作频率的动态调整功能，使其能够适应高 中低端各种运用；PRU 模块（Programmable Realtime Unit）提供了对 EtherCAT 、Profibus、Powerlink 等多种工业通信协议的支持，使其可以成为真正行之有效的低成本解决方案；此外，在本次专题中，我们还将着重对一些工业自动化设计、运用进行简要的介绍。

本节内容由德州仪器半导体有限公司技术支持工程师刘靖伟讲解。

本节内容主要讲述 AM335x 在工业自动化领域里的应用及优势。主要包含下面三方面内容：一是 TI 嵌入式处理器 AM335x 的片上资源；二是基于 AM335x 工业自动化应用的介绍，深入具体的介绍 AM335x 的设计、运用，以及它特有的优势；三是 AM335x 软硬件设计资源介绍，TI 提供了相当丰富的软硬件设计参考以及多种评估板，有利于客户进行评估以及更为有效的开发。

下面是 TI 嵌入式处理器 AM335x 介绍。

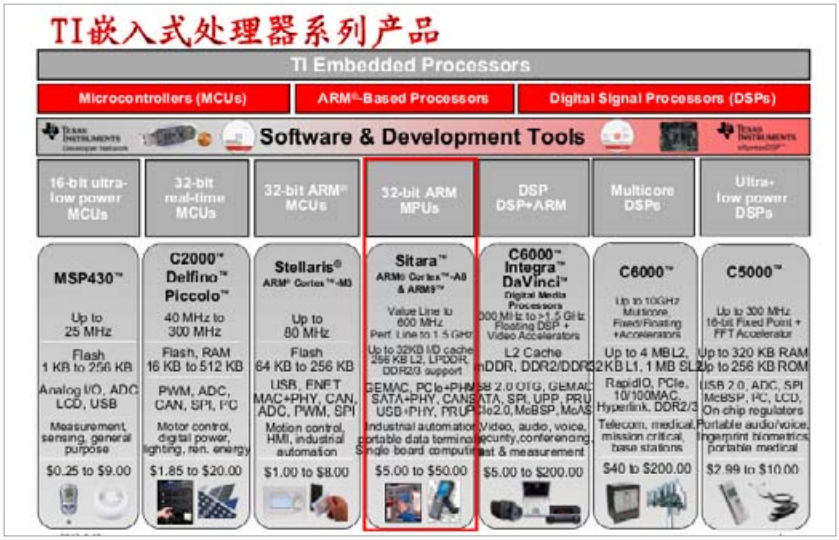


图 2-91

TI 的嵌入式处理器产品资源相当丰富，从微控制器到基于 ARM 的处理器，到高端的

DSP 一应俱全。AM335x 是位于 32-bit ARM MPU 这条产品线，在 AM335x 芯片中有哪些资源可以在工业自动化应用中得到很好的运用呢?

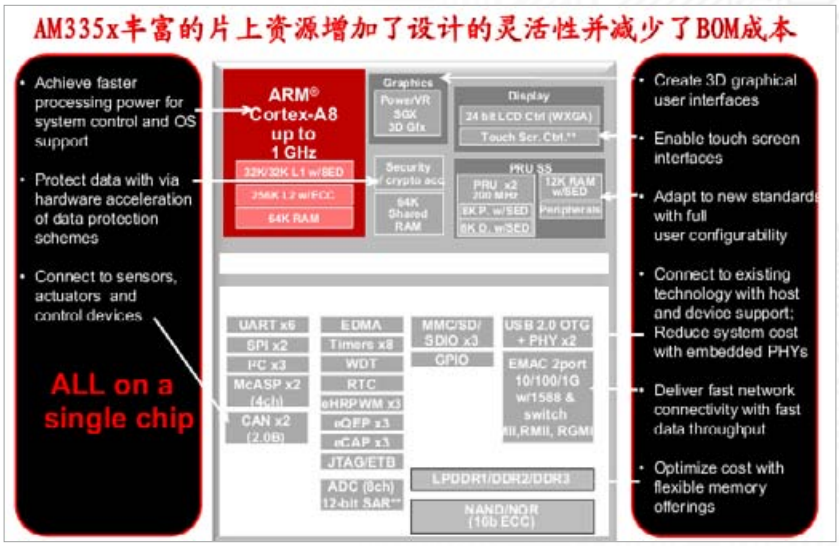


图 2-92

从上图中可以看到，AM335x 具有非常丰富的片上资源。首先映入眼帘的就是主频可以达到 1G 的 Cortex-A8 处理器，高主频加上 Cortex-A8 的架构，正是 AM335x 处理器高性能的保障。显示方面提供对 24bit LCD 屏的支持以及触屏的支持。此外，还有 3D 图形加速器的存在，这使得 AM335x 具有丰富良好的显示效果，适合 HMI 等工业应用。另外需要注意的是，可编程实施单元 PRU 子系统是 TI 所独有的主频为 200 兆赫兹的主核，拥有自己独立的 Run，从而保证了其实时效率。该芯片可以支持各种工业总线协议，比如说 EtherCAT、PROFIBUS，使得 AM335x 在工业方面具有非常广泛的应用。此外，从资源列表中我们可以看到，我们提供六个 UR 的口，这使得 323 与 485 的接口非常丰富。两个带有 PHY 的 USB 2.0 OTG 口，使得 USB 的应用得以实现。两个可以支持千兆以太网的 EtherNET 端口以及对 LPDDR1、DDR2、DDR3 的支持。AM335x 丰富的片上资源增加了设计的灵活性，并且减少了 BOM 的成本。

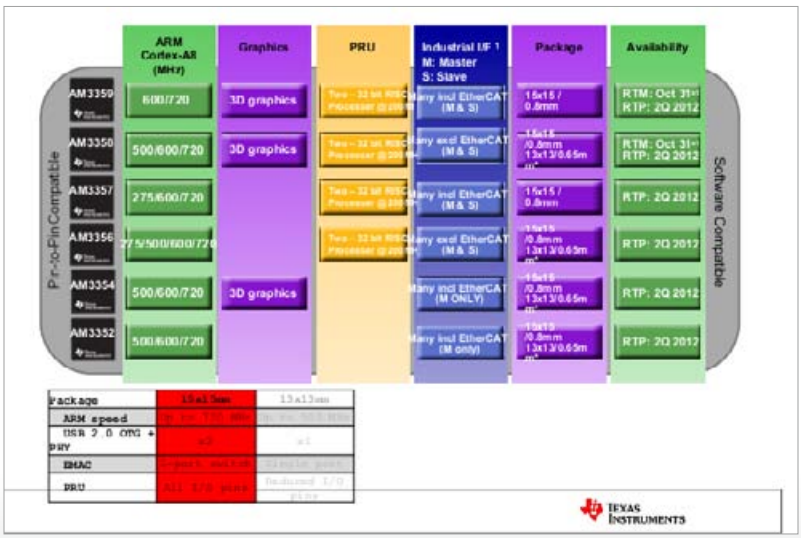


图 2-93

从上图可以看到，TI AM335x 系列产品中提供了多种定制化的选择。用户可自行根据自己对图形加速模块和 PRU 模块以及工业通信接口模块的需求选择不同的产品型号。从不带有图形加速模块 PRU 模块的 AM3352，到这些功能一应俱全的 AM3359，TI 皆有提供。因此，为客户提供了充足的选择空间。

下面介绍基于 AM335x 的工业应用介绍。

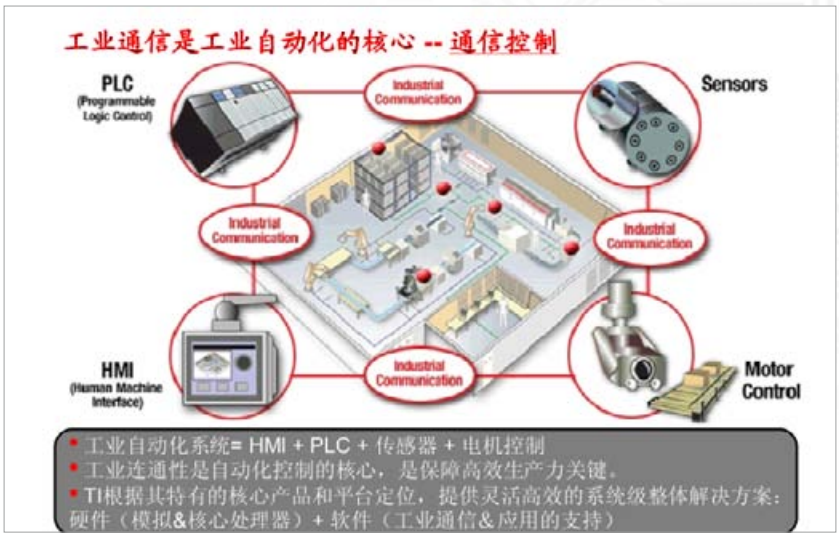


图 2-94

这是工业自动化应用的系统框图。工业自动化系统是由人机交互界面 HMI\ 可编程逻辑控制单元 PLC，还有一些传感器以及电机控制所组成。那么在整个工业自动化系统中，工业通信则是它的系统核心，是保证高效生产力的关键。TI 根据其特有的核心产品 AM335x 和工业自动化平台定位，提供了灵活、高效的系统级整体解决方案。由模拟部件和核心处理器构成的硬件，以及对工业通信以及应用的支持的软件，完整的实现了整套工业化自动控制的流程。

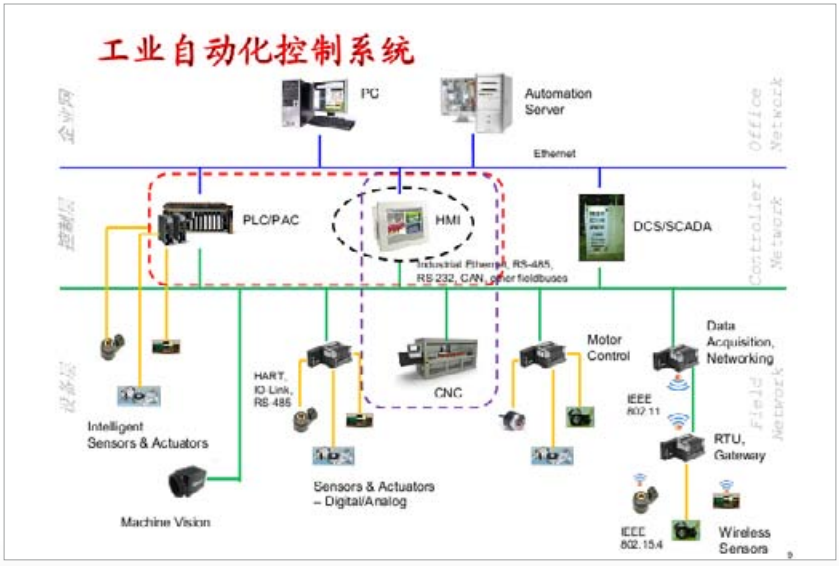


图 2-95

上图是基于层次架构的工业自动化控制系统。在这个系统中可以看到，从最底层是由智

能传感器 234、485 传输以及电机控制等接口组成。控制层有一些 PLC、HMI 单元，企业网层我们有 PC 机以及一些服务器。AM335x 在设备层与控制层凭借本身优良的 Cortex-A8 处理器性能，非常丰富的接口资源，包括对工业以太网、485、232 接口，CAN 总线的支持，有着相当广泛的运用。那么 AM335x 是如何实现对多种工业通信总线的支持?

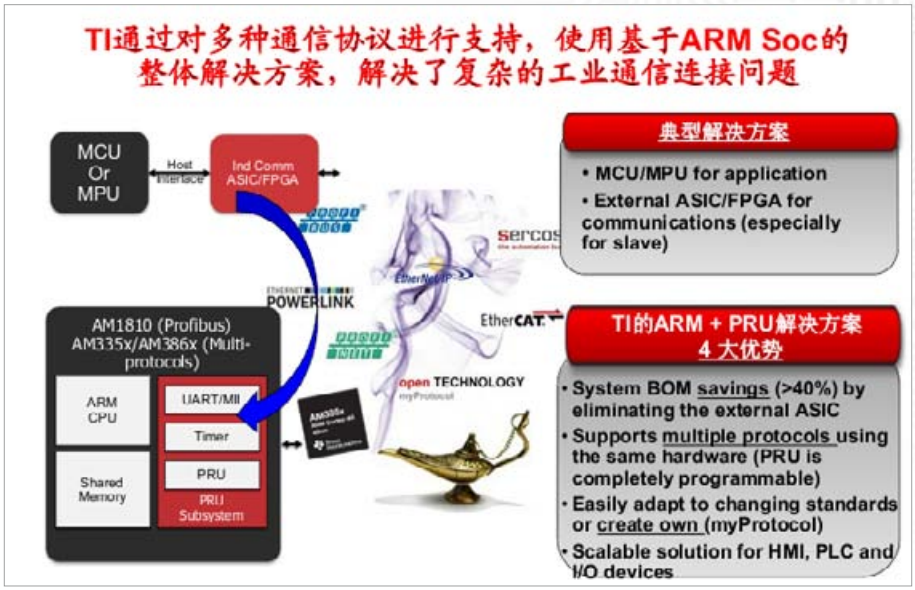


图 2-96

TI 使用基于 ARM Soc 的整体解决方案，通过对多种通信协议的支持，解决了复杂的工业通信的问题。那么传统的典型解决方案是怎样的呢? 传统我们会使用一片 MCU 或 MPU 作为应用程序的载体，并使用外部的 ASIC 或者 FPGA 作为通信协议的实现。那么在 TI 的解决方案中，将采用 ARM 加上 MPU 的整体解决方案，前面我们介绍过，由于 PRU 的实时可编程特性，可提供对 EtherCAT、PROFIBUS 的支持，可以轻易的用 PRU 单元去实现对工业通信总线的实现。在 TI 的解决方案中，采用了 ARM 加 MPU 的整体解决方案。前面我们介绍过 PRU 可编程实施单元的可行，那么这是基于可编程以及实时的特性，使得 TI 的 PRU 单元可以支持 EtherCAT 以及 PROFIBUS 的协议，完成了对工业通信的支持。

TI 的解决方案有怎样的优势呢? 一是系统的 BOM 表大概能够解决 40% 的成本，因为你不需要外加一片 ASIC 或者 FPGI；二是由于 PRU 的可编程特性，使同样的硬件可以支持不同的工业通信协议，从而增加了解决方案的灵活性；三是可以自行修改定式化通信协议；四是由于 AM335x 系列产品具有良好的软硬件兼容特性，所以一套设计可以同时为 HMI、PLC 以及底层 I/O 设备所共用。因此，解决方案更为灵活，研发成本得以节约。

基于刚才介绍的工业自动化应用系统框图，现在介绍一下 AM335x 关于这些具体的应用的设计框图。

首先是 I/O 设备以及传感器。

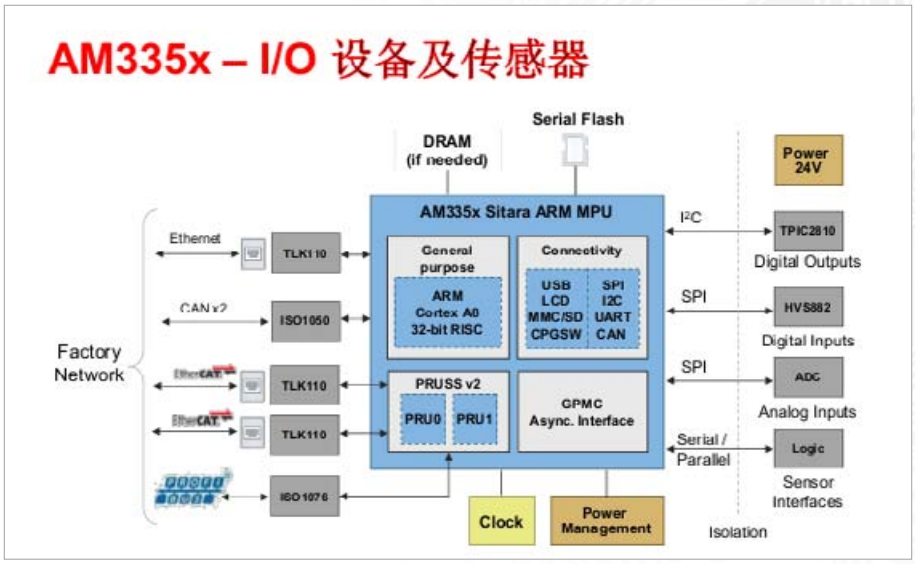


图 2-97

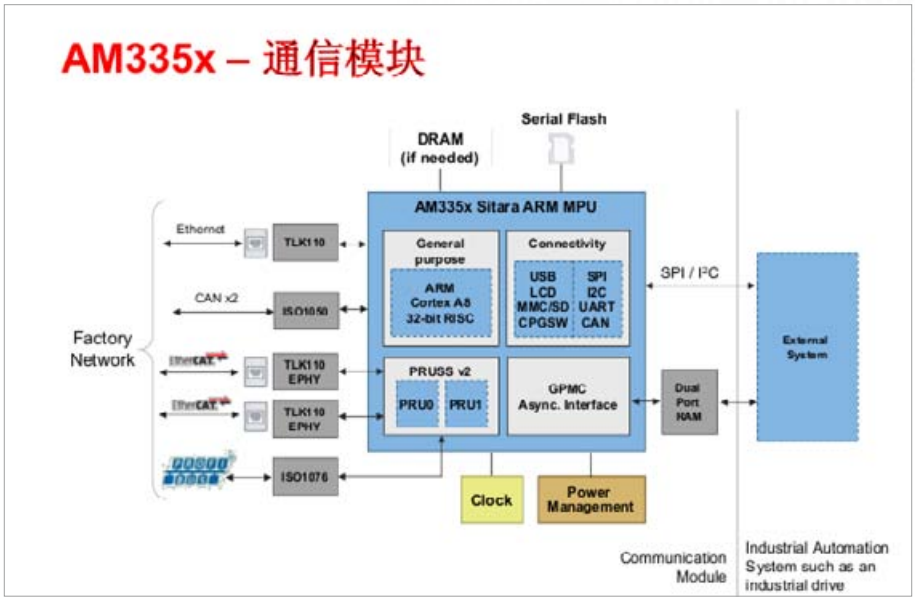


图 2-98

AM335x 基于 I/O 设备及传感器的设计框图。首先，在框图的左侧与工业网的连接，我们提供了 EtherCAT 网络端口以及两条 CAN 总线，以及 EtherCAT、EtherNET、PROFIBUS 的支持。在右侧的通讯端口之中，我们提供了 SPI，I2C 以及 GPMC 接口。在左侧的工业网与之前 I/O 设备一样，我们提供了 PROFIBUS、EtherCAT，EtherNET 以及 CAN 总线的支持。

接下来是关于 PLC 从站的设计框图。

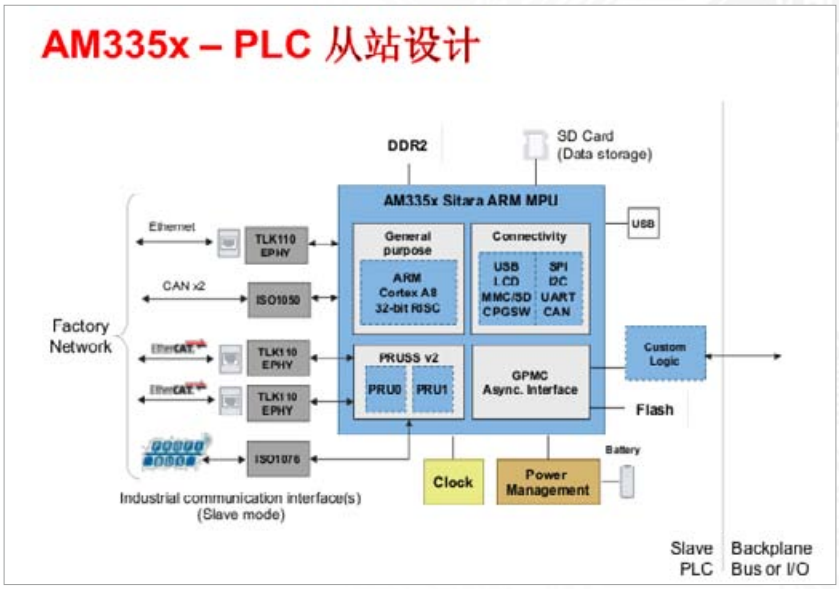


图 2-99

在这里需要重点指出的是，PLC 从站的设计需要利用到 PRU 子系统。因为 PLC 从站对时延、效率等有严格的要求，只有使用 AM335x 的 PRU 子系统作为整个协议的实现载体，才可进行 EtherCAT 的通信。

接下来是 PLC 主站。

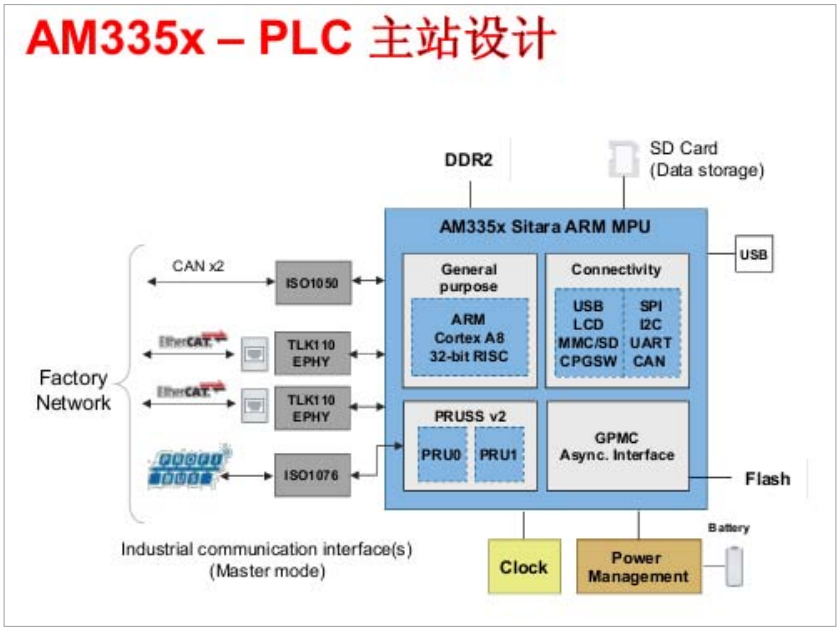


图 2-100

那么与 PLC 从站相对，PLC 主站对于通信的实时性要求没有从站那么严格，所以 PLC 主站的实现可以直接通过我们 335 上所提供的双网口来进行实现，并非一定要使用 PRU 子系统。

接下来是人机交互界面 HMI 的设计框图。

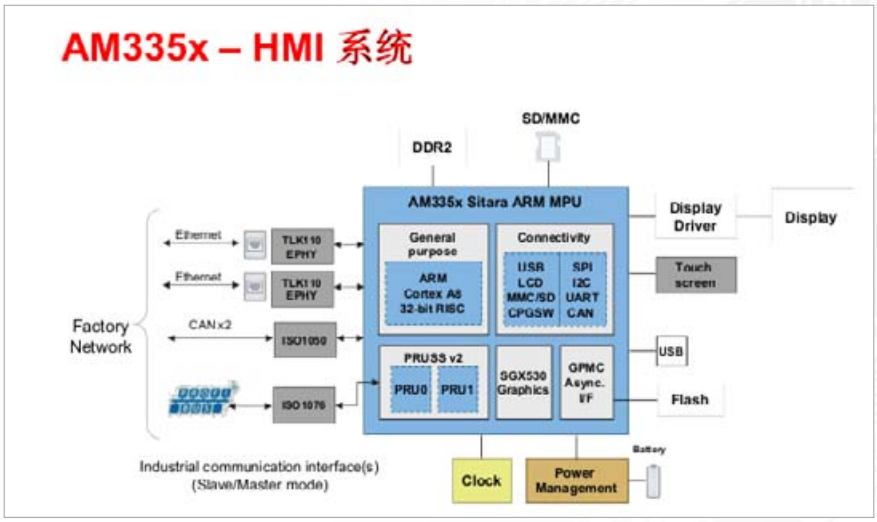


图 2-101

这里我需要指出的是，HMI 系统是 AM335x 较为广泛的运用，通过对触屏模块 3D 显示特性的支持，我们可以进行丰富的 HMI 设计。

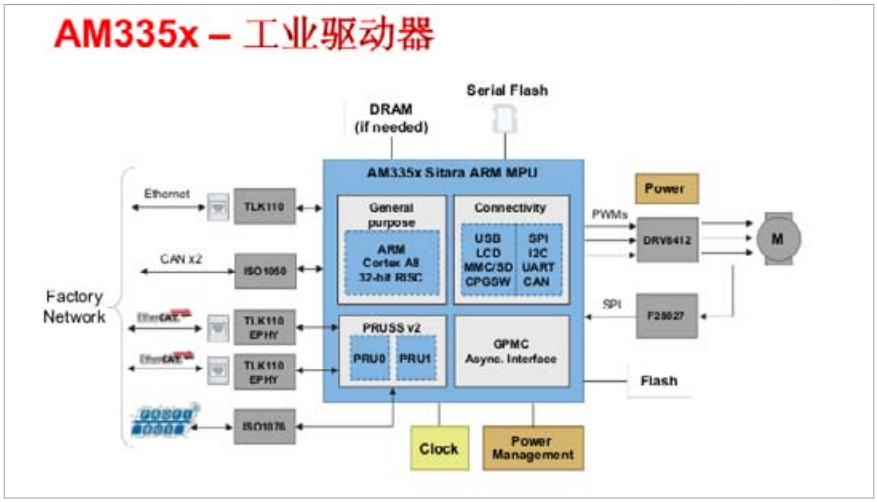


图 2-102

AM335x 关于工业驱动器的设计框图如图所示，在左侧，与工业网的连接与前面的设计如出一辙。在右侧我们提供了 PWM 的支持，因此可以进行电机马达控制，并且有 SPI 的信号返回线，能够进行信号的反馈。

以上就是 AM335x 基于工业自动化应用的一些设计介绍，下面介绍 AM335x 软硬件设计资源。

首先是关于 AM335x 硬件设计资源，TI 提供了很多 AM335x 的硬件评估平台，可以用于高效、低成本的开发。

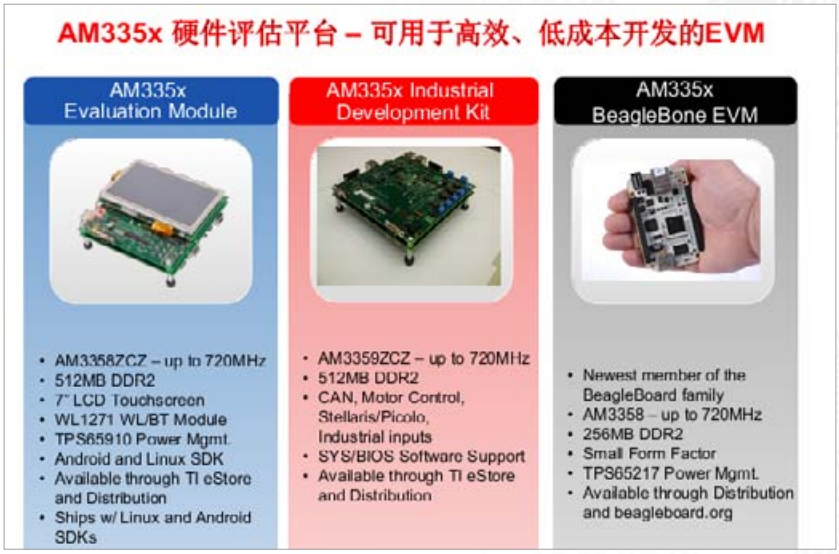


图 2-103

上图是主要的开发板，最左侧的是接口功能最为丰富的 AM335x 通用评估板。中间的是 AM335x 基于工业应用的 IDK 评估板，右侧是低成本，但是片上资源非常丰富的 BeagleBone。

此外，TI 还提供了 AM335x Starter Kit，上面拥有双网口的设计，可以方便用户进行网络的验证以及工业的运用。

图 2-104

◆ TI AM335x EVM 原理图和 PCB 布局设计参考：

- BeagleBone http://beagleboard.org/hardware/design/
- GP EVM http://www.ti.com/tool/tmdxevm3358
- StarterKit http://www.ti.com.cn/tool/cn/tmdssk3358

◆原理图和 PCB 布局设计检查列表：

- http://processors.wiki.ti.com/index.php/AM335x_Schematic_Checklist
- http://processors.wiki.ti.com/index.php/AM335x_Layout_Checklist

- DDR2/3 布局设计、电源解决方案、BSDL/IBIS 仿真模型…
- 更多资源尽在 https://www.ti.com
- http://processors.wiki.ti.com/index.php/Category:AM335x

TI 还有基于工业的 ICE 板子，上图所示。TI 除了提供相应的 EVM 板之外，还提供了相应的 EVM 板的原理图以及 PCB 布局设计作为参考，这些资源都是开源的，在 TI 的 Wiki 上可以找到。除了 EVM 板的原理图和 PCB 的布局设计，还有了一些由非常有经验的工程师撰写的原理图和 PCB 布局设计的检查列表。此外针对 DDR2、DDR3 的布局设计、电源解决方案以及一些仿真模型，都有提供相应的源码，这一切资源都可以在 TI 的网站上找到。

关于 AM335x 软件设计资源的介绍。

图 2-105

从上图中可以看到，TI 提供了非常完善的软件设计方案，方便用户更快捷、更高效的进行开发。从经典的 Linux，到非常流行的 Android，到 TI 所特有的 StarterWare，再到 WinCE，TI 对这些源码都予以提供。至于 RTOS，TI 有自己的 RTOS SystemBios 可供用户参考，用户亦可以跟一些第三方的 RTOS 公司进行联系，进行相应程序的开发。

下图展示了 AM335x ARM MPU 的软件架构。

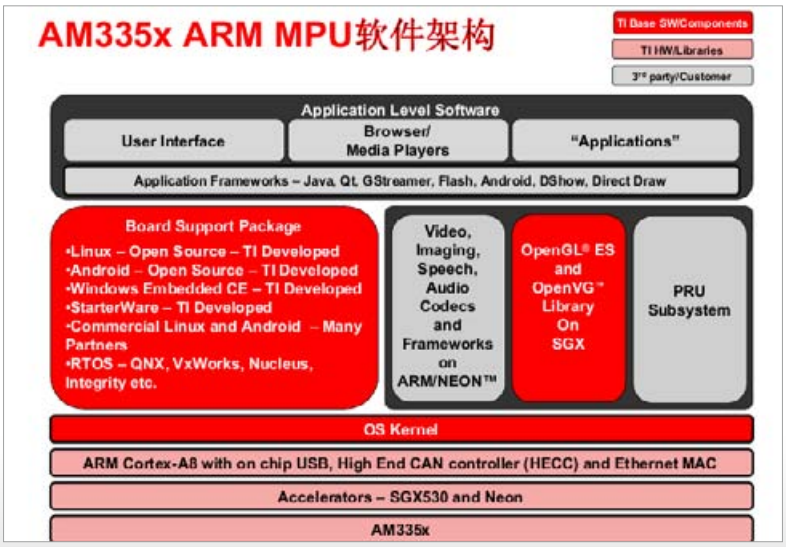


图 2-106

可以看到，从最底层的 AM335x 硬件，到一些 AM335x 的加速器，再到一些片上资源，通过 OS 层的调用，再结合中间的中间件，再结合上层的应用程序，共同完成了 AM335x ARM MPU 的软件架构。所有软件的开发都是基于这个软件架构所执行的。

接下来介绍一下 Sitara Linux 软件开发包。

源自开源工程 -- Arago

http://www.arago-project.org/wiki/index.php/Main_Page

◆最新的 kernel 版本：v3.2

http://www.ti.com/tool/linuxezsdk-sitara

◆ SDK 软件开发包用户指南

Linux 软件开发指南 .
http://processors.wiki.ti.com/index.php/Sitara_Linux_Software_Developer%E2%80%99s_Guide

◆更多资源，请参考

http://processors.wiki.ti.com/index.php/Category:Linux

TI Sitara Linux 软件开发包是源自开源工程 Arago，目前支持到的最新 Kernel 版本 v3.2，除了软件源码之外，TI 还提供了 SDK 软件开发包的用户指南以及开发指南，相关的资源在 TI 的 Processors.wiki 上面都可以下载到。Android，AM335x 同样支持非常流行的 Android 开发系统，最新的 Android 版本，我们已经支持到业界领先的 4.2.2 版本，而且除了源码之外，一样针对 Android 我们也有相应的软件开发包使用指南以及开发指南，这些都可以从 TI 获得。

◆ TI Android 开发包

最新 Android 开发包版本：4.2.2
http://www.ti.com/tool/androidsdk-sitara

◆ AM335x Android 软件开发包使用指南

http://processors.wiki.ti.com/index.php/TI-Android-JB-4.2.2-DevKit4.1.1_UserGuide

◆ AM335x Android SDK 软件开发指南

http://processors.wiki.ti.com/index.php/TI-Android-JB-4.2.2-DevKit4.1.1_DeveloperGuide
http://processors.wiki.ti.com/index.php/TI-Android-JB-4.2.2-DevKit4.1.1_PortingGuide

◆更多资源，请参考

http://processors.wiki.ti.com/index.php/Category:Sitara_Android

SitaraWare 是 TI 一个非常有特色的系统。什么是 SitaraWare 呢?

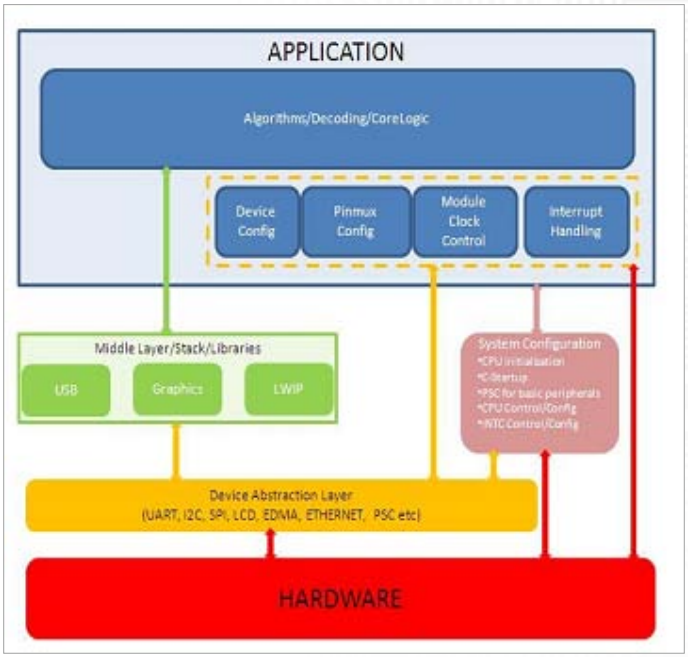


图 2-107

SitaraWare 是由 TI 基于 C 语言开发并进行维护的一个非操作系统的软件开发包，并提供设备抽象层代码示例。比如说 SOC 的启动代码，DDR 内存的初始化代码，以及串口、I2C 等驱动。由于没有操作系统复杂平台化的功能,所以 SitaraWare 是一个清凉剂的软件包，相应的它也不支持多线程或者进程。SitaraWare 非常适合需要高实时性，并且无并行的运用领域，目前在工控、HMI、工业缝纫机、PLC 等工业方向都有了非常成熟的应用。

基于 AM335x 的 WinCE 开发包

http://www.ti.com/tool/wincesdk-a8

◆ WinCE 相关开发文档

Wiki 资源
http://processors.wiki.ti.com/index.php/Category:Sitara_WinCE
e2e 论坛支持
http://e2e.ti.com/support/embedded/wince/default.aspx

AM335x 同样提供了对 WinCE 的支持,对于 WinCE 的开发包可以从 TI 的官网上下载到，并且在 Wiki 资源链接以及 e2e 的论坛支持中，我们动可以找到相应的板块，找到相应的资源去支持 WinCE 的开发。

◆ RTOS 简介

- http://www.ti.com/lsds/ti/dsp/support/software/os_overview.page

◆ TI RTOS -- SYSBIOS :

- http://processors.wiki.ti.com/index.php/AM335x_SYSBIOS_Industrial_S-DK_01.00.00.05_User_Guide

◆ 第三方的 RTOS 支持 , 需要直接和第三方厂商进行联系。

AM335x 对 RTOS 的支持是非常多的，从 TI 的官网上我们可以看到 TI 支持哪些第三方的 RTOS，并且 TI 提供了自己的一个 RTOS SystemBios 的源码下载。但是需要指出的是，对于第三方的 RTOS 支持，需要用户直接与第三方厂商进行联系，并且获得支持。

◆ Wiki 主页

所有的软硬件设计资源可在以下连接中找到

- http://processors.wiki.ti.com/index.php/Sitara

资源介绍 :

- http://processors.wiki.ti.com/index.php/AM335x_Resource_Guide(English/%E4%B8%AD%E6%96%87)

◆ 在线论坛支持

- E2E 论坛 -- 来自全球专家的支持

◆ http://e2e.ti.com/support/dsp/sitara_arm174_microprocessors/default.aspx

◆ http://e2e.ti.com/support/embedded/linux/default.aspx

◆ http://e2e.ti.com/support/embedded/android/default.aspx

◆ http://e2e.ti.com/support/embedded/starterware/default.aspx

◆ http://e2e.ti.com/support/embedded/wince/default.aspx

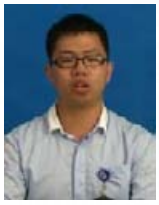
- DeyiSupport 论坛 -- 来自中国区技术支持工程师的支持

◆ http://www.deyisupport.com/question_answer/f/25.aspx

最后需要指出，Sitara 非常重要的在线支持资源。首先 Wiki 是所有 TI Sitara 资源的结合。那么在 Wiki 的主页上有所有软硬件设计的资源，TI 提供了非常丰富的 Sitara 在线支持资源。首先 Wiki 主页就是所有源码存在的地方，也是遇到问题时所能够找寻的依据。所有的软硬件设计资源以及对资源的介绍都可以从 Wiki 的官网中找到。此外我们还提供在线论坛的支持，需要指出的是，e2e 论坛是一个全球的平台，在上面你会得到来自全球专家的支持。此外，我们还有由本地技术支持工程师所组成的论坛 deyisupport 论坛，在上面基本上每天都会有大量的问题以及相应的工程师负责对问题进行解答。大家如果在开发过程之中遇到相应的问题，也可以从这两个论坛上面找寻相应的答案。

2.8 基于 AM335x LED WALL 应用

LED Wall 广泛应用于体育场馆、城市及公共广场、交通、企业形象宣传、商业广告及其它应用中。其中异步全彩 LED Wall 由于其低成本，便于集中管理多块屏的特 点，已逐步成为全彩屏的主流应用。AM335x 以其强劲的处理能力，丰富的接口已经越来越为广大异步全彩控制卡厂商所使用。AM335X 拥有高性能的 Cortex-A8 内核，主频最高可以达到 1GHz，同时可以根据对视频处理的能力的需求，选择 300MHz/600MHz/800MHz/1GHz 。1G Hz 的 AM335x 可以稳定解码 720x576@25fps，并且留出超过 25% 的 CPU 资源供客户进行图片，动态字幕等的显示。在 LED Wall 软件方面，其拥有丰富的软件平台，包括 Linux，Android，Wince 的全部支持，其中 Linux 作为主要的目标平台，并支 持免费开放的 Gstreamer+FFMPEG。



本节内容由德州仪器半导体有限公司嵌入式处理技术支持工程师吴红军讲解。

下面将从三个部分进行讲解：第一部分：LED WALL 的概述及分类，主要介绍市场上 LED WALL 大概有哪几种，主要的应用是什么；第二部分：介绍 AM335x 的主要性能指标以及资源，其中包括 AM335x 的系统框图、接口，以及现有的资源；第三部分：介绍如何基于 AM335x 进行 LED WALL 的开发。

LED WALL 广泛应用于体育场馆、城市公共交通以及企业形象的宣传。比如电影院、一些高铁的候车厅等，都能看到 LED WALL 的应用，上面会显示广告、Video 和一些图片，这是大家比较熟悉的 LED WALL 的一个应用环境。

图 2-108

常用的 LED WALL 分类其实有三种：单色 LED WALL、双基色 LED WALL 以及全彩 LED WALL。单色 LED WALL 主要显示的是文字，主要平台用 MCU 的这种比较低性能的

处理器就可以;双基色的显示首先会要求有一些简单的图片、动画和文字。实现上述这些显示，就需要选用一些性能稍微高的 MCU，或者 ARM9 平台；全彩 LED WALL，目前主要应用是 24 位的全彩色，它显示的不仅包括图片、文字，还有动画以及音视频。这种音视频动画的显示所要求的频段比较高，因而需要一些高性能的 ARM 进行音视频的编解码。

下面我们主要讲述全彩 LED WALL。全彩 LED WALL 有更加细分的分类，可以按照 LED WALL 的分辨率来进行分类，也可以按显示的内容进行分。按分辨率来分的话，可以大致分为四类：一是比较普通的视频显示有 320×240 点阵的；二是数字标准 DVD 显示，它的显示分辨率是大于 640×480 点阵；三是完整计算机视频，大于 800×600 点阵；四是高清视频，也是现在应用的一个趋势，会有 720p 以及 1080p 的分辨率要求。如果按照显示内容进行分类的话，可以分为同步 LED WALL 及异步 LED WALL。

那么什么是同步 LED WALL，什么是异步 LED WALL？

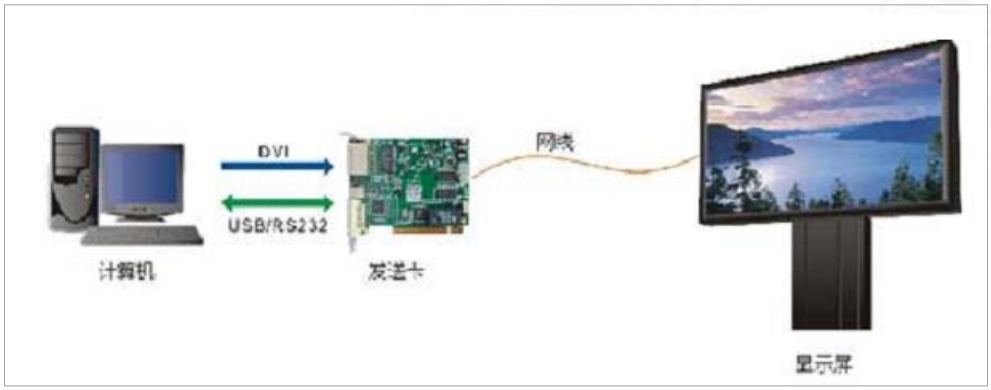


图 2-109

同步就是 LED WALL 显示屏和计算机显示器上面的内容完全同步显示。比如说在某一个广场上面，大屏幕上显示的是工作人员正在打扑克的一个画面，其实那就是一个同步 LED WALL 的显示。因为他正在电脑上进行一些操作，可以同步在广场上的 LED WALL 上进行显示，可同步看到，这是它的一个主要的特征。计算机的操作可以在 LED WALL 上同步显示是通过一个发送卡来进行执行的，它的作用是将 DVI 发送过来的一些信号转化为 LED 屏幕所需的信号。最主要的应用是通过 FPGA 这样一个平台来实现，这个方案的特点除了同步之外是灵活性较小，计算机要加上发送卡这样组合，这种成本比较高。

图 2-110

什么是异步全彩 LED WALL？异步全彩 LED WALL 就是 LED 屏上显示的内容和计算机上显示的内容是不同步的，可以事先将一些显示的内容在计算机上编辑好，发送到一个控

制卡上，控制卡再进行显示。这种就是异步的特点，这样的特点首先是可以从一个集中内容的内容库获取内容，然后进行多个 LED 屏幕的显示。异步系统 LED WALL 的特点是比较便于集中管理多块屏，同时其成本比较低，可以一台 PC 配多个控制卡进行多个 LED 的显示。在这里需要重点强调，TI AM335x 以其强劲的处理性能，具有很多接口，可以越来越为广大异步全彩控制卡的厂商所使用。

对于一个全彩 LED WALL，下面是 AM335x 的系统框图。

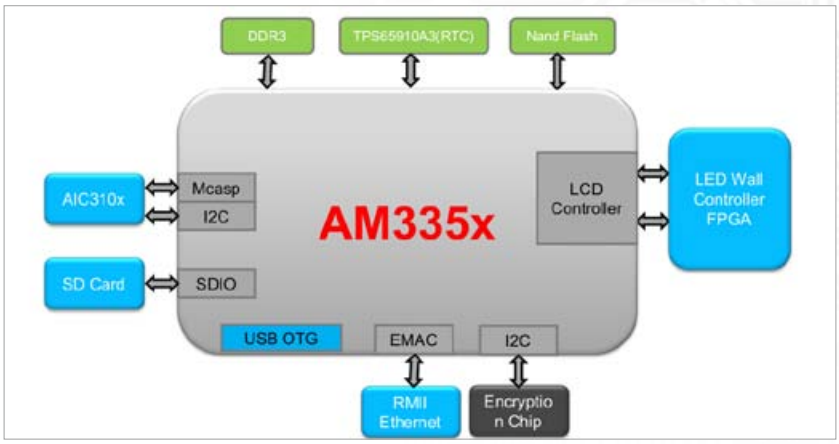


图 2-111

全彩 LED WALL 的框图，AM335x 全彩 LED WALL 在这里可以看到有一片 DDR3，可以支持 Nand Flash，需要一个 Audio 的输入和输出以及 LED Controller，可以外扩一个 FPGA 进行一个 LED WALL 的控制，可以扩展 SD 卡以及 USB。这些接口可以进行一个数据的扩展以及用户数据的管理。另外 AM335x 支持两个网口，这里网口也是通用的一个应用。同时在客户的开发过程中可能需要一些芯片以及应用的保密，可以通过 I2C 进行一个加密芯片的扩展。对于这些接口我在后面的一个内容中会进行一些详述。

接下来在简单介绍一下软件的结构图。

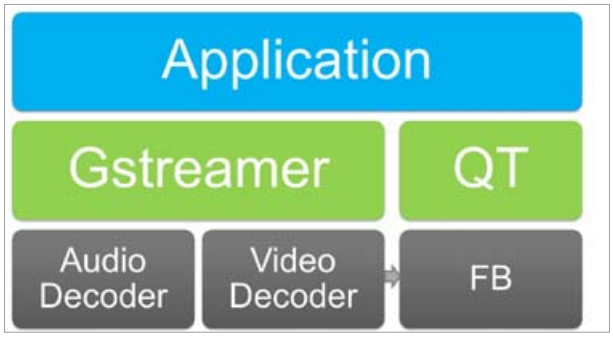


图 2-112

全彩 LED WALL 需要进行一个视频、音频的处理。首先需要一个 Audio，就是音频的一个编解码。当然 AM335x 没有一个视频的协助仪器，在这里可以做一个软件的音频解码，同时一个视频的解码。基于这个解码之上，支持 Gstreamer，Gstreamer 是一种开源的、基于流水线的多媒体框架，可以基于 Gstreamer 加 QT 进行这样一个 LED WALL 的应用开发。基于如何进行 Gstreamer 的开发，后续会有一个详细的解释。

刚才简单讲了全彩 LED WALL 基本的硬件框图以及软件的框图，重点还是强调一下硬件平台的整个硬件需求。在这里可以讲一下，首先全彩的 LED WALL 需要进行一个 UI 的显示，有一个视音频的编解码。AM335x 在这里具有一个主频可以到 1G 的 A8 的核，可以满足进行 UI 的绘制以及视音频的编解码。常见的 LED WALL 其实都是放在户外的。比如说在中国，要把 LED WALL 用在北方，在冬天时候它的室外温度可以达到零下 30 度，甚至零下 40 度，那就需要一个工业级的，温度范围比较广的芯片进行这样一个应用这样一个应用的要求，AM335x 刚好就是可以满足这样的需求，因为 AM335x 有一个工业级的芯片，是支持零下 40 度到 90 度的，可以在户外进行一个稳定的运行。系统框图中有许多接口，比如说音频接口、SD 卡接口、USB 接口，这些接口都是必须的，是应用中不可缺少的，AM335x 也能满足这些接口的要求。

主流的软件有 Linux、WinCE、Android，在 TI 推广的 LED WALL 这个过程当中，重点推的还是 Linux。为什么呢? 因为现在目前行业中应用最多的是 Linux，它的一个开源性系统，它的资源是十分丰富的。现在应用的 GUI 可以用 QT，多媒体框架可以用 Gstreamer，播放器可以用 MPlayer，都是一个开源的软件资源，可以方便客户进行自由的开发。同时在 Linux 系统上还支持 LED WALL 的组态软件的支持，因为在 LED WALL 的显示中，可能要对组态软件进行图像的设置然后进行传送，因而组态软件是非常重要的。除 Linux 之外，还有一部分客户选择 WinCE，这是由于之前一些工业客户的习惯，以前一些老的用户都是用 WinCE 进行开发的，TI 也会有一些对于 WinCE 的支持。对于第三种操作系统 Android 目前还没有领先的客户进行使用，但是它良好的界面、丰富的应用软件也是比较有吸引力的。缺点是缺乏组态软件的支持。

全彩 LED WALL 控制卡软件的平台需求主要有三点：一是显示分辨率，一般高端的有 720p/1080p，中低端的能到 640×480。另外一个要求是显示分辨率可以动态调整，因为在 LED WALL 终端客户进行调试时，会动态的拼接 LED WALL 模块以达到最好的显示效果。所以在要求不重启的情况下，动态的进行分辨率的显示修改，这是对显示分辨率的要求。二是视频编解码的要求。高端的 LED WALL 会支持多种视频格式，一些手机上面会主要支持 MP4，中低端的至少也会有一个 MP4 制品格式的支持。另外由于 LED WALL 的尺寸比较大，亮度比较高，所以在视频帧率上面会要求至少达到 25 帧及以上。三是 UI 和视频界面的要求，需要进行 UI 界面可以叠加，对视频可以进行调整大小，进行一个旋转功能的实现。

AM335x 为什么能够作为一个 LED WALL 的最佳选择? 主要有以下几种特点。一是拥有高性能 A8 核的芯片，它的主频最高可以达到 1G。二是客户可以根据自己对视频处理能力的要求选择，有 300、600、800M 和 1G 的芯片，而且都是 Pin to Pin 兼容的，如果客户在前期对视频处理要求不高，可以选择低主频的芯片。后期如果对视频能力要求逐渐有更多的一些定制化的要求，可以选择高主频的芯片，比如 1G。这里有一个参数，TI 1G 的 AM335x 的芯片可以稳定进行解码 720×576，同时帧率可以达到 25 帧。同时在这样的条件下可以留出超过 25% 的 CPU 资源，供客户进行图片和动态字幕的显示，这是 TI AM335x 一个比较大的特点。TI AM335x 可以进行多图层叠加硬件加速。因为在 AM335x 中集成了一个 3D 协处理器 SGX530，软件也是支持通过 OpenGL ES 的 SGX530 进行多图层叠加的硬件加速。三是有丰富的外设。刚才讲了 LED WALL 的硬件框图的需求，包括 LCD 接口、音频接口、DDR 的支持以及双网卡的支持，都可以满足客户全部的要求。四是 AM335x 是一款工业级的应用芯片，温度可以达到 -40 度到 -90 度，可以满足 LED WALL 整个的工作温度要求。五是比较有特点的地方，就是有丰富的软件平台，在 LED WALL 进行开发

时，可以提供丰富的一个全套的软件支持。在刚才讲的主流支持的 Linux 以及 WinCE 以及 Android 都有。但是现在主要客户用于 Linux，所以大部分资源，包括我们一整套方案都是基于 LED WALL 来进行实现的。同时具有免费开放的 Gstreamer+FFMPEG 平台，它是一个开源的平台，可以方便客户进行一些定制化的工作。

下面详细介绍 AM335x 芯片特点。

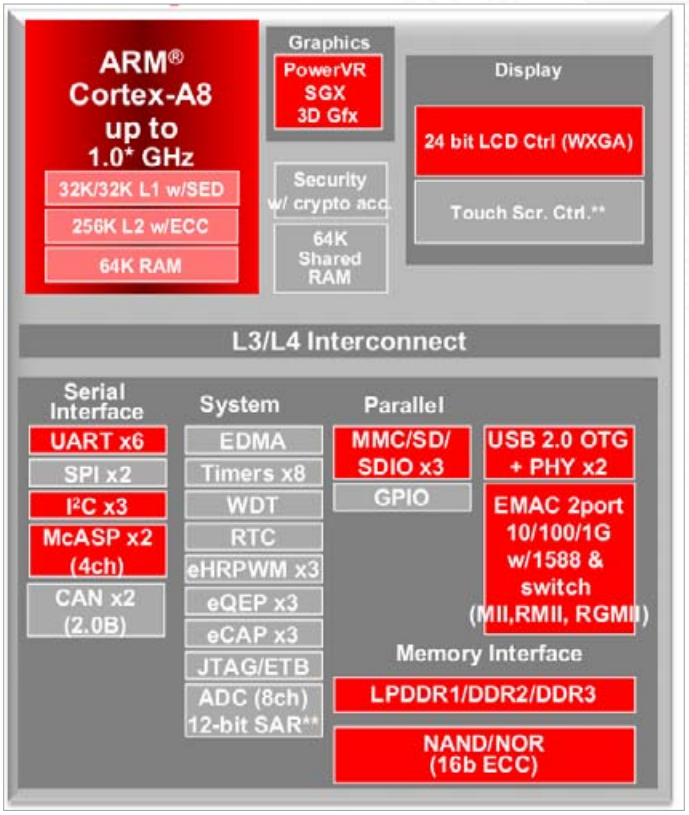


图 2-113

从框图我们可以看出，标红的这部分是代表 AM335x 对 LED WALL 硬件需求的满足。红色的左上角可以看到，这是高达 1G 的 Cortex-A8 的芯片，它是一个高性能处理器，可以进行复杂的运算和处理。在主芯片的旁边，中间上方是一个 3D 协处理器 SGX530，它可以在对图形要求比较高的情况下进行图像的处理，包括叠加、旋转。在右上方可以看到有一个 24 位的 LCD Controller（LCD 控制器）。因为在 LED WALL 中，需要通过 LCD 控制器连接进行 FPG 的扩展或者进行 LCD 的显示，这是一个必须的接口。除了上述主要三个接口以外，看到下方标红的其他接口。这里需要重点强调的是 DDR3，因为作为一个主流的 DDR，可以帮助客户降低 BOM 成本，它的采购比较容易，价格也比较便宜。另外包括支持 Nand Flash，Nor Flash，SPI Flash。但是在 LED WALL 中，用的比较多的还是 Nand Flash，一个特点是因为它容量比较大，另一个特点是它价格比较便宜。其他接口包括串口可以用于调试，I2C 接口可以用于扩展加密芯片，McASP 可以用于扩展 Audio 芯片进行音频的输入输出。另外 USB 和 SD 卡客户可以用于数据的存储和系统对外的一个接口。AM335x 支持双网口，可以满足客户进行一些网口的一些应用需求。

下面讲述一下 AM335x 整个产品线图。

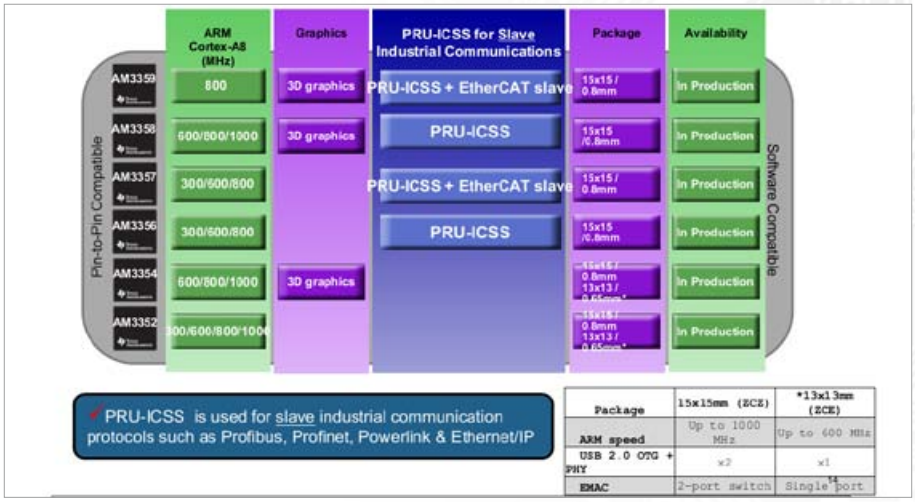


图 2-114

从这个框图上可以看到，AM335x 其实有 6 个系列，AM3352、AM3354、AM3556、AM3357、AM3358、AM3359，其实他们都是 Pin to Pin 兼容的，每个型号的芯片也有多种组频可供选择。6 款系列芯片在 Pin to Pin 兼容情况下其实有一些外设是不同的。例如，如果有客户要进行 UI 的开发，或者进行图像处理，那他就需要具有 3D 协处理器的芯片，在这里就可以选择 AM3354、AM3358、AM3359，它们都是带 3D 协处理器的。如果反之，不需要进行 UI 开发，就可以选择 AM3352、AM3356、AM3357。另外 AM335x 还集成了一些工业协议，它是通过一个比较独特的接口叫 PRU（可编程的实时接口），在其中实现的一些方案，包括 EtherCAT、PROFIBUS 的一些接口，如果不需要，可以选择 AM3352、AM3354，如果需要 EtherCAT、PROFIBUS 接口，可以选择 AM3356、AM3357、AM3358、AM3359。

接下来介绍 TI 在 AM335x 这款芯片上面支持哪些软件。

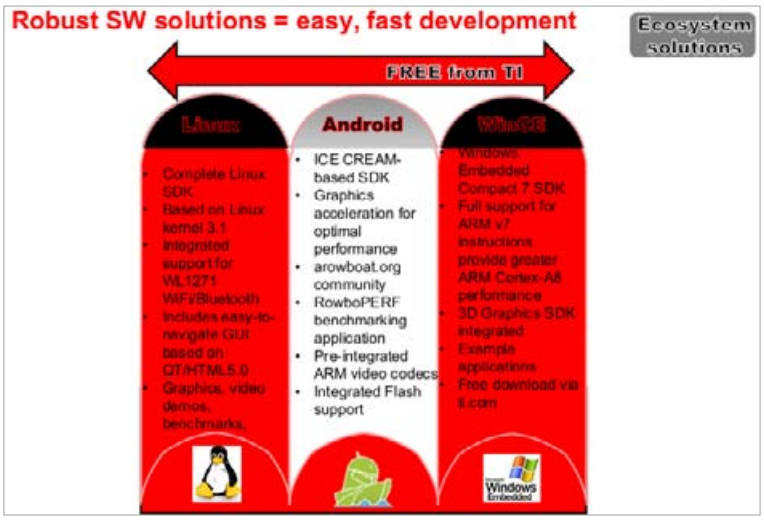


图 2-115

主流的系统软件包括图中显示的 Linux、Android、WinCE 都支持。WinCE 是收费软件，TI 提供在 UM 板上进行 Demo，包括 BSP 包。但是后续的支持只能提供芯片级的支持，这个需要客户在开发过程当中需要考虑。而对于另外两种操作系统 Linux 和 Android，客户就

不用担心，TI 重点会支持这两种操作系统，包括 Linux 开发包 TI 每隔 6 个月会有新版本更新，主要更新包括 Linux 版本以及一些常用的应用更新。在图中，我们可以看到一些 Linux 支持的信息，包括它还支持 Wi-Fi 的驱动、蓝牙以及 QT GUI，Graphic 的一些 Video 的 Demo，在里面都会有。整个软件的支持其实到现在为止已经有一个丰富的软件例子，以及一些相应的文档，有一个非常好的生态系统开发，客户只要根据 TI 文档就可以进行轻松快速的开发流程。

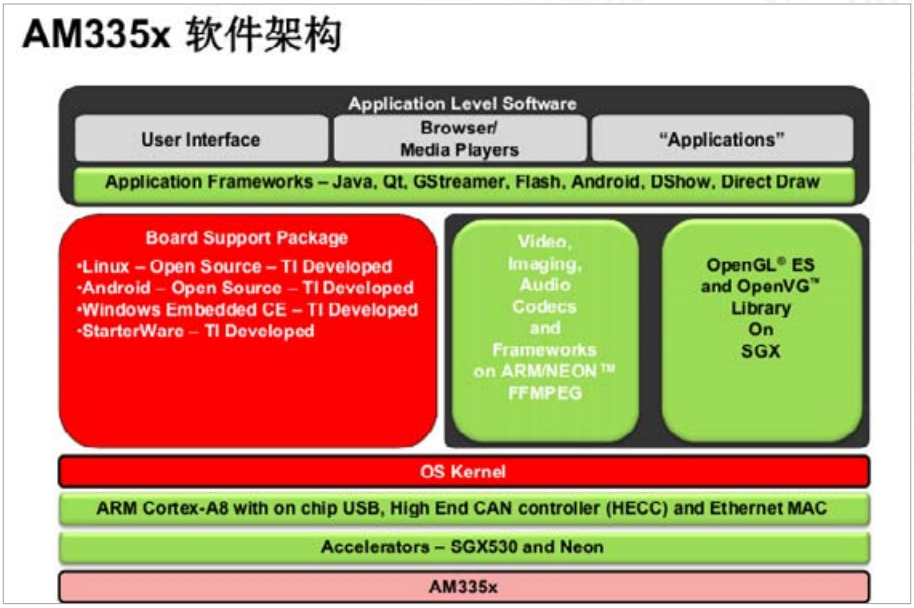


图 2-116

在上面软件框图中可以看到，TI 的这个软件在图上的显示以及 Audio 的 Codec 是通过 Neon 进行支持的。它调用的是一个 FFMPEG 的开源库。在 3D 的支持方面基于 SGS530，它的开源库是 OpenGL ES 进行 3D 的加速以及图像的处理。基于这些之上，同时集成了一个应用的固件，包括 JAVA、QT、Gstreamer、Flash 等等。

下面讲 LED WALL 软件方案的关键点。可以从这个 PPT 上看到，重点有三个：一是视频播放；二是 UI 视频层叠加，包括调整大小、旋转；三是动态修改显示分辨率。针对这上述三个性能，介绍 AM335x 如何进行实现。

LED WALL 要求一个视频的播放帧率在 25 帧以上，AM335x 对视频的支持主要有哪些特点呢? 首先 AM335x 没有视频加速模块，这一点其实可以有效的控制成本，因为有了视频加速模块,整个芯片成本会比较高。但是没有的话,也可以通过强劲的 A8 内核进行软件编码，做到不错的性能。另外软件上采用 Gstreamer+FFMPEG 这样的框架，可以是一个开源的框架，进行有效的开发。推荐一个支持的视频格式，可以支持 MP4，这种格式比较简单，但是显示效果确实不错。在 AM335x、Android 和 Linux 的性能上，可以支持的视频最大性能 D1 可以大于等于 25 帧。同时在这种应用的情况下，CPU 的 Loading 在 1G AM335x 芯片的应用中可以小于等于 75%，仍旧有大于等于 25%CPU 的 Loading 可以供客户进行另外的一些应用，包括之前讲的简单的图片编辑和显示。在 AM335x Linux 和 Android 视频的显示上面，刚才也重点强调的是 Gstreamer+FFMPEG。由于现在 AM335x 的芯片有 6 个档次，同时也有多个主频，客户可以根据视频的分辨率选择不同的主频 AM335x，以达到最佳的成本控制。

刚才讲了针对于应用的三个问题的第一个问题。第二个问题是 AM335x 怎样对图形叠加，包括图形调整大小，旋转性能要求的满足？AM335x 有一个 LCD Controller，LCD Controller 仅供于一个 LCD 显示功能，对于一个低分辨率小于等于 320×240 的，我们可以通过 ARM Core 进行运算实现。但是对于高分辨率，它的要求就比较高，计算也比较复杂。对于 ARM Core 的 Loading 就比较重。这个时候我们就可以通过 GPU，通过在 AM335x 上 SGX530 进行硬件加速，这样可以减轻 ARM Core 的 Loading 负担。作为一个 GPU 模块，SGX530 的配置也是非常简单的，TI 的软件已经可以做到，也可以通过 OpenGL ES 软件接口提供给大家。目前 TI 对于这一块的开发有一个 Graphic SDK，它是在官网上可以进行免费下载的软件包，下载之后，可以在上面进行视频层的开发、叠加等相关的操作。

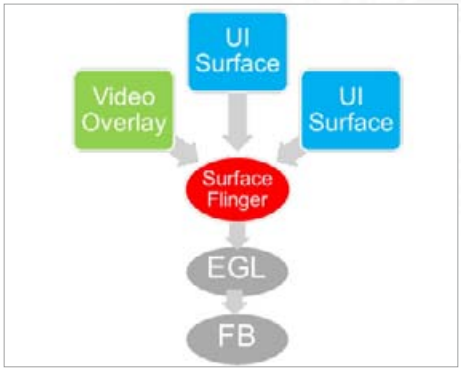


图 2-117

Android 是通过 Surface Flinger 进行应用的，Surface Flinger 对于显示的管理是通过一个或者多个 Graphic Playing 对象进行管理的，目前 Android 只实现了一个。当 Android 支持多个显示系统时，包括 UI 的一些 Surface，还有一些 Video 的 Overlay，是可以通过该类进行各自系统的管理。基于 Surface Flinger 之后，底层是一个 EGL，EGL 其实是一个 API，是位于 OpenGL ES 于底层的一个平台，视窗直接的一个接口，这是 Android 的一个系统框图。

下面是 Linux 的一个应用框图。

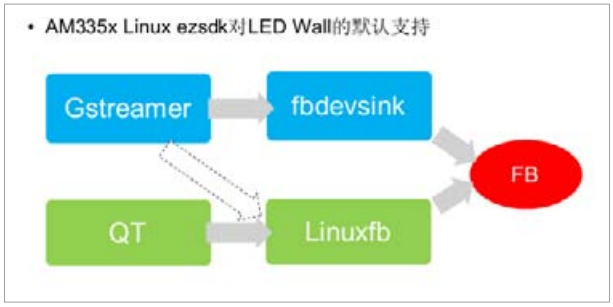


图 2-118

通过软件的方式，用 ARM Core 进行大小的调整，图层的叠加，包括重叠以及 Alpha 等操作。但是这些操作会导致 ARM Core 整个 CPU 的 Loading 过高，从而导致系统的性能不够。在这里需要指出的，视频编解码出的视频帧一般是 YUA 格式，而 FB 仅支持 RGB 格式，所以需要通过色彩空间转换，而恰巧这一色彩空间转换，会更加加剧整个系统的 Loading。

刚才介绍了 Android 和 Linux 的应用方案，接下来介绍怎样基于 TI 的 Graphic SDK 进行 GPU 的加速的方案。GPU 主要完成以下几点工作，可以降低 ARM Core 上面的预算压力。一是 Alpha 叠加；二是调整大小；三是进行图片的旋转；四是色彩空间的转换。通过 GPU 组合，可以仅占据 ARM Core 的 Loading 的 10%，同时它的 Buffer 传递是以指针的方式进行传递的，没有额外的内存拷贝。这里还需要强调的，Alpha 叠加需要对每一个点进行乘加运算，所以它的运算量是非常高的，非常大的。如果由 GPU 来负担的话，可以很大程度的减轻 ARM Core 的运算压力。同时在图像的大小调整、旋转上也是有负责的算法，运算量也不小。色彩空间刚才也讲过了，由于涉及到复杂的运算，其所需的 CPU 甚至和视频编解码的需求相当。GPU 如果能完成上面的工作，从很大程度来说可以降低 ARM Core 的负担。

大家可以看到 GPU 组合的这样一个框图。

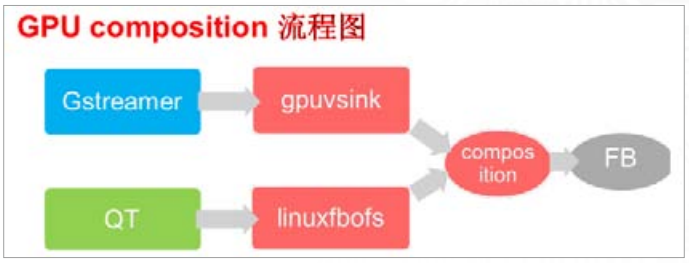


图 2-119

是通过 Gstreamer 然后码流到 GPU Video Sink，什么是 GPU Video Sink？它是基于 Open GL ES 的 Gstreamer Video Sink 插件，可以将 Gstreamer 解决出来的视频帧以 Texture 的形式（就是经常翻译的纹理形式），输入到组合模块中进行叠加操作，这是一路。另一路是通过 QT，QT 可以将生成的一些 Framebuffer 以 Texture 的形式送到组合当中进行叠加。这两路都是以纹理的方式送到组合模块当中，组合模块再进行下列操作，比如说根据输入的 Framebuffer 的设置进行色彩空间转换，这需要一个比较大的预算。另一个是通过 Framebuffer 进行一个图片的大小调整、旋转以及组合，这些都是基于一个管道机制来进行的。最后将这些处理好的数据送到 Framebuffer 当中。

GPU 组合的关键主要有以下四点：一是在组合中我们会用到多功能的纹理方式进行叠加，这种模式和 Android 的 Surface Flinger 的模式一样，有较高的效率。二是对于输入的视频帧界面的 Framebuffer，这里需要注意的，它的尺寸宽必须是 8 个像素对齐。三是 QT 中定义的显示属性和组合模块中定义的显示属性应该一致，因为两者都是通过 Fbset 标准接口获取显示的属性。四是系统可以有多个 UI 层和一个视频层，UI 层是在视频层之下，但是 UI 层的视频透明度是通过 Alpha 这样一个参数进行调节的。

Graphic 组合对于内存的需求到底是怎样，可以用一个公式来计算。首先 Graphic 组合由于调用了 Graphic SDK，所以会增加部分内存的需求，大致可以分为两类：一部分是 Graphic SDK 工作的 Buffer，这部分 Buffer 是在 GPU 的内部使用，它是可以通过 Linux 的内存子系统进行分配，大概需要 10M Bytes 的大小。另一类是对于 GPU Video Sink、Linux Framebuffer OFS 和组合中使用的 Plane Buffer，这部分 Buffer 是在组合模块中由相应的模块传递帧的时候使用，这部分 Buffer 需要的是一个物理连续的 Buffer。所以可以通过 TI 开发的一个 Cmem 工具进行分配，进行连续的物理地址的分配。一个 Plane 对于一个图层可以由以下的计算公式进行计算，对于 UI 层，Framebuffer 的大小可以通过它的宽乘高，乘以每个像素的 Bytes 数来进行计算。对于视频层，同样可以通过 Video 一个帧

的宽、高以及每个像素点 Bytes 的大小，乘以 6 个 Buffer 的数进行计算，这里就可以很轻松的计算出对于一个内存的需求。

刚才介绍了 GPU 组合的一些设置以及性能之后，接下来我们讲如何去编译 GPU 组合和安装 GPU 组合? 这里我给出一些链接。

◆相关模块的编译

- 最新的 Ezsdk，可以从如下链接下载

◆ http://downloads.ti.com/sitara_linux/esd/AM335xSDK/latest/index_FDS.html

◆按照对应的 guide 编译好 kernel

- http://processors.wiki.ti.com/index.php/AM335x_PSP_User%27s_Guide
- 最新的 Grpahic sdk，可以在这里下载

◆ http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/gfxsdk/latest/index_FDS.html

◆按照 guide，编译好 Graphic sdk

- http://processors.wiki.ti.com/index.php/Graphics_SDK_Quick_installation_and_user_guide#About_Graphics_SDK

- 最新的 GPU composition 源码，可以在这里下载

◆ https://github.com/murthygp/gpu-compositing

◆ 按照 guide，这里就可以编译好整个的 GPU composition 了

- http://processors.wiki.ti.com/index.php/GPU_Compositing

◆ GPU composition 的安装

- 可以参考这里进行安装和 demo 测试。

◆ http://processors.wiki.ti.com/index.php/GPU_Compositing#Overall_Execution_steps_on_Target

最新的 Ezsdk，我们可以通过上述链接进行下载，上面会有详细的文档，教您如何去进行内核的编译。Graphic SDK，就是我们常说的图像的软件包也可以在上述链接进行下载，同时也有对应的公开文档指导您进行 Graphic SDK 的编译。另外 GPU 组合的源码，我们也可以在上述链接下载，它是一个开源的，在 TI Wiki 上面也会有一个详细的文档，教您如何进行编译好整个 GPU 的组合，安装也有对应的文档和 Demo，这里大家都可以看到。

GPU composition 配置

- UI 层

◆可以通过 QT 的 linuxfbofs 插件配置，包括 显示的位置，区域大小，全局 / 局部 alpha.

◆具体的参数解析可以参考

http://processors.wiki.ti.com/index.php/GPU_Compositing#Driver_Specific_Options

◆这里的 alpha 值，决定了视频层显示的透明度。

- 视频层

◆可以通过命令行或者 gpusink 的接口，对视频显示的位置，区域大小，旋转角度等进行配置，具体的参数可以参考

http://processors.wiki.ti.com/index.php/GPU_Compositing#Usage_Examples

◆视频层不需要设置 alpha。

鉴于 AM335x 处理能力有限，建议使用两个图层，一个是 UI 层，一个是视频层。UI 层可以通过 QT 的 Linux Framebuffer OFS 插件配置，包括显示的位置、区域大小、全局、局部的 Alpha，这些具体的参数可以参考上述 Wiki 上的链接进行配置。这里的 Alpha 值是决定了视频层的显示透明度。另一个是视频层，视频层可以通过命令行或者 gpusink 接口，对视频显示的位置、区域大小、旋转角度等进行配置，具体也有相应的链接进行参考。视频层这里就不需要设置 Alpha 值了。

第三点：显示动态的调整分辨率 AM335x 如何做到? 在 Ezsdk 提供的 Kernel 中，Framebuffer Driver 中提供了对于动态显示分辨率的支持。可以通过以下的命令，比如说 fbset mode，加上你的显示分辨率就可以进行一个动态的显示，这是一个非常方便的接口已经做好了。

总体来说该方案非常适用中低端的全彩 LED WALL 的控制板的应用。如果大家对于 AM335x 有更多感兴趣的内容，可以访问 TI 的官网进行一个资源的查询。

2.9 AM335x 在 HMI 系统中的应用

从常见的手机到大型 特种机械，都可以看到 HMI 系统的影子。HMI 系统正在广泛被各行各业所应用。特别在工业领域，随着行业的发展，新技术不断被引进以及工业子系统间的功能 整合，使得如今的 HMI 系统已经不再是一个简单的显示设备。Ti sitara 产品线推出了一系列 ARM9 & Cortex-A8 芯片，不仅在 成本上可以满足高、中、低端 HMI 系统的需求，而且以其强劲的处理性能以及专门针对工业领域的丰富接口更好地帮助客户顺应了这样的市场发展趋势。软件方 面，Ti sitara 产品支持主流的操作系统，如 android、linux 和 wince，以及 sys/bios 和 starterware 等 RTOS，并免费开放所有源码，大大降低开发难度，帮助客户节约了宝贵的开发时间。

本节内容由德州仪器半导体有限公司技术支持工程师鄂舒扬讲解。

本节介绍 HMI 系统以及 TI Sitara 系统产品在 HMI 领域里的优势。涉及到以下几个方面：第一 HMI 是什么? 第二 TI 在 HMI 系统里的解决方案 Sitara 产品线；第三 TI 提供的开发工具和参考设计；第四 TI Sitara SDK 的软件架构。

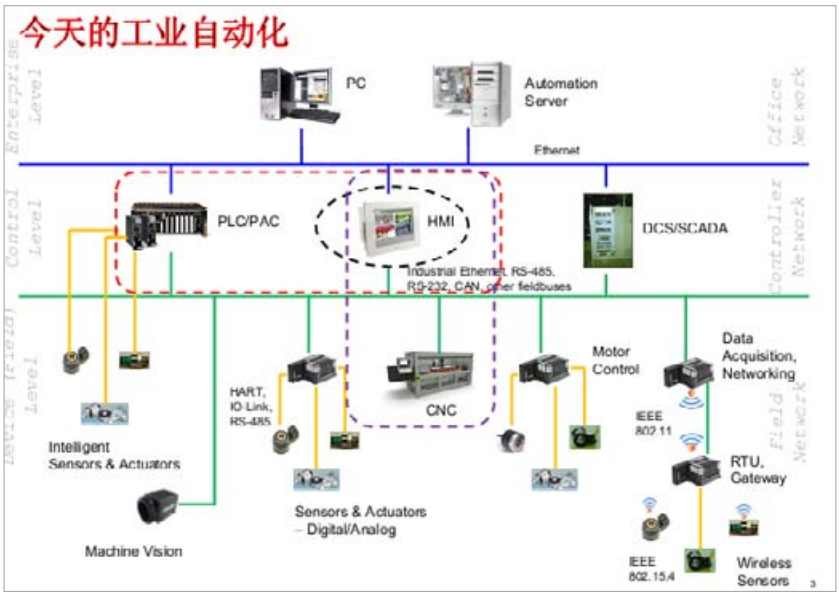


图 2-120

首先看一下今天的工业自动化。从上图可以看到，HMI 属于控制这一级，它的作用主要是一方面收集从底下各个设备节点传来的信息并反馈给操作者。另一方面，也可以通过工业总线对各个设备节点进行控制。随着工业系统的发展，在部分应用中，HMI 有时候也会集成 PLC 或者 CNC 的功能。这样一来，对 CMI 系统中所选用的主控处理器就有了更高的要求。在这里 TI Sitara 产品线推出了一系列高性能的处理器，以满足当今工业市场的各个需求。不仅可以帮助客户降低开发难度，缩短开发时间，同时也帮助客户节省了设计成本。

下面了解一下 HMI 的基本知识。

图 2-121

HMI 又称人机界面，是使用者和嵌入式计算机之间交互的窗口，HMI 运用非常广泛，包括但不限于消费类的产品，比如智能手机、常用的 PND 手持设备以及车载终端。在商业产品当中，包含自动贩售机，以及经常使用的打印机系统。在工业领域里，它还主要应用在工业控制终端，家庭、楼宇智能控制终端。

在工业应用的 HMI 系统中，还会被细分为简单的带字符型显示设备输出的 HMI 系统，比如我们经常用到的小型计算机。还有一种是带 2D 图像显示功能，并支持触摸屏功能的 HMI 系统，比如说 CNC、数字机床等。还有一种是带 3D 图像功能，并支持视频加速性能，这种应用就属于比较高端的应用了。如最开始的介绍，HMI 的一个重要功能就是通过工业总线的方式对各个设备节点进行控制以及信息的采集。

这里列出了一些最常用的总线。

图 2-122

串口一般接口是 RS232，或者 RS485 总线，I2C、CAN 总线、USB、以太网，以及在部分 HMI 场景中还会用到的无线技术，比如蓝牙、Zigbee 和 Wi-Fi 等。这样所选的 HMI 系统的主控处理器也应该具备这些功能，或者说至少是支持这些功能。除了以上对总线的需求外，当今的 HMI 还会有以下更多的需求。比如同一硬件平台同时需要满足不同的性能要求以及成本需求，在有些 HMI 的生产厂家里，它的产品不仅仅是单一的针对高端、中端或者低端的 HMI，往往是同一平台会覆盖整个低端、高端和中端的 HMI 系统。在这三种 HMI 系统

当中，除了性能差异之外，还会对成本上有很高的需求。另外，HMI 系统还需要支持不同的显示设备，比如说 LCD 屏，还有字符型的显示器。对于 LCD 来讲，也需要有多种分辨率的支持。而且随着工业自动化的发展，传统通讯逐渐的向高性能、高速和高可靠性的通讯方式去发展，比如说 EtherCAT 和 PROFIBUS，以上这些需求对于系统，对于 HMI 的主控芯片提出了新的要求和挑战。正是为了顺应这样的市场发展和变化，TI 公司推出的 Sitara 系列产品，可以很好的为客户应对这些要求和挑战。

下面介绍一下 TI Sitara 系列在 HMI 系统中的优势。

TI Sitara 产品线覆盖了 HMI 高端、中端、低端系统的需求，低端的 HMI 系统可以采用 AM335x 系列的芯片，中端的 HMI 可以采用 AM335x 以及 AM3x5 系列的芯片，高端的 HMI 系统可以采用 AM335x 以及 AM37x 系列芯片。针对以上这些芯片，TI 除了会提供相应的开发板资源外，还会提供类似于 Linux、Android 和 WinCE 的系统开发和系统源代码，以方便客户开发设计。另外，TI 还提供了 Systembois 和 StarterWare 等操作系统，以满足低端 HMI 的需求。这里我们可以注意到，AM335x 系列是可以覆盖从低端到高端 HMI 系统的所有应用的，下面我们来重点介绍一下 AM335x 系列芯片。

AM335x 是一个 Cortex-A8 的芯片，Cortex-A8 是一个高性能的 ARM 处理器，主频可以从 300MHz 到 1GHz。同时除了高端的 ARM Cortex-A8 核之外，还支持 LCD 模块，在 LCD 模块里除了支持常用的 TFPT 之外，还有一种叫做 LIDD RDD 的模块，用于支持字符形成的显示设备。同时 AM335x 里内嵌了 3D 加速模块，在高端 HMI 应用场景里，可以保证 HMI 的 3D 性能。另外 AM335x 有丰富的外设，包含两路的 CAN，两路十兆、百兆、千兆的以太网，两路 USB、OTG2.0 以及六路的串口，PWM、时钟捕获、ADC 等，基本上已经可以包含了所有 HMI 系统中用到的需求。另外 AM335x 支持多种外部处理器，支持 Nand Flash、Nor Flash、SRUM、eMMC 以及 SPI Flash。这些 Flash 在工业领域里都是最常用的外部处理器。同时 AM335x 还支持 Memory Power DDR1、DDR2，特别是增加了对于 DDR3 的支持。我们都知道，DDR 的发展是十分迅速的，尽管目前工业市场里主流还是 DDR2，但是在未来的几年里，DDR2 很可能会遇到现在 DDR1 的这些问题，比如说采购难、价格高等问题。随着市场的发展，DDR3 正以更低的成本、更高的性能和更高的可靠性逐步被更多的工业市场所接受。AM33x 里还包含一个叫 PRU 的系统，PRU 是 TI ARM 芯片的一个特色，PRU 是独立于 ARM 运行的运算单元，它包含独立的 RUM 和 ROM，也需要独立的变成。可以把 PRU 理解成是一个独立于 ARM 的小的处理器。目前 PRU 上可以做一些总线扩展，比如说我们已经在 PRU 里实现了串口的扩展。大家都知道，AM335x 有六路的串口，如果加上 PRU 里扩展的串口，目前我们总共支持 10 路的串口。同时在一些工业总线上面，比如像 EtherCAT 和 PROFIBUS 上面，已经在 PRU 里集成了它的协议。在一般的应用当中 EtherCAT 和 PROFIBUS 是需要额外增加一颗芯片。这样不仅仅增加了整个 PCB 的成本，同时也增加了 PCB 的面积，AM335x 可以更好的去解决现在常用的模式所遇到的一些问题。那么 AM335x 又是如何去覆盖低端、中端和高段的 HMI 系统呢?

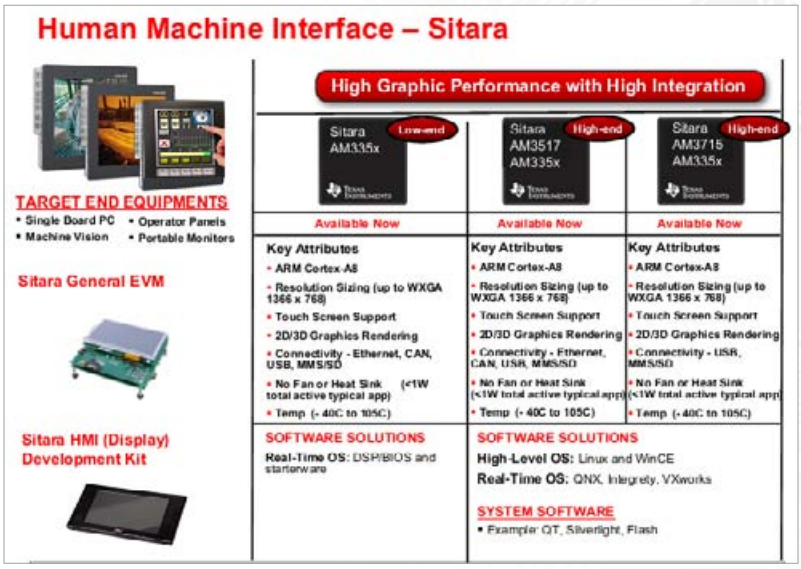


图 2-123

从这张图可以看到，HMI 是一个家族系列，包括从最低端的 AM3352 到最高端的 AM3359，针对不同的应用，我们提供了不同的排列方式。比如说在低端的 AM3352 里，我们没有集成 3D 加速模块、PRU 系统，最低的主频是可以做到 300 兆主频。在低端的 HMI 里，我们可以采用 AM3352 来做。在高端的 HMI 中，如果有 3D 的需求，有工业总线的需求，推荐采用 AM3359。一方面 AM3359 的主频最高可以跑到 1G，同时在 AM3359 里也包含了 PRU 系统、3D 模块，而且也会将 EtherCAT 和 PROFIBUS 的协议集成在 PRU 系统里面。对于中端的 HMI 来讲，如果说需要一些像 3D PRU 的系统的话，推荐采用 AM3356 或者 AM3357。如果只是一般的中端 HMI，没有 3D 的需求，没有工业总线的需求，推荐的是 800 兆或者 1G 的 AM3352。在所有的 AM335 家族产品当中，这些芯片全部都是 Pin to Pin 兼容，而且软件上兼容，这样就可以方便客户在一个板上进行所有 HMI 平台的开发，大大降低了开发难度和缩短了开发时间。

下面列出了一个基于 AM335x 简单的 HMI 的一个系统框图。

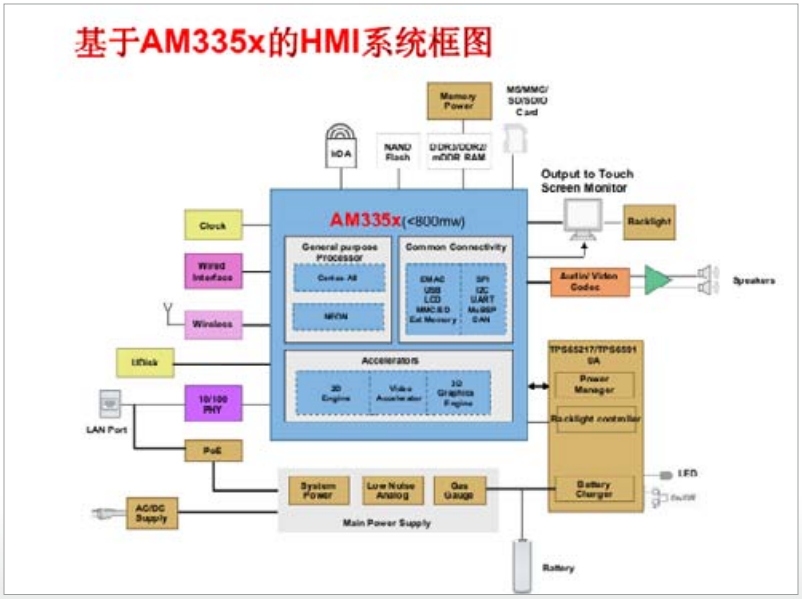


图 2-124

在这个系统框图里基本上涵盖了所有 HMI 系统中用到的基本的总线，比如说网口、串口，包括红外。在这里我们也有一些电源上的推荐，现在这套方案已经广泛的被市场所应用。为了方便客户的开发，降低开发难度，TI 提供了很多种开发板以供设计参考。

图 2-125

第一个开发板是通用 EVM 板，它基本上涵盖了所有 AM335x 的接口。除了这个开发板之外，TI 还提供了工业开发板，工业开发板主要面向的是工业客户，在里面 TI 提供了很多工业协议，比如 EtherCAT、PROFIBUS、CAN 等等。针对 AM335x 可以支持 DDR3 的特性，TI 还推出了支持 DDR3 的 AM335x 开发板，叫 Starter Kit。另外 TI 还有推出了两种低成本的开发板，一个是基于面向工业的低成本开发板叫 ICE，以及通用的低成本开发板我们叫 BeagleBone。每个开发板都有自己的特色，如果只是面对中端或者是低端的 HMI，我推荐各位使用中间的 DDR3 的 Starter Kit 开发板。如果说有工业总线的需求，比如像 EtherCAT、PROFIBUS 或者 CAN，我推荐大家使用工业 IDK 或者是 ICE 的板子。如果只是说想做一个平台级的核心版，我推荐大家使用最右侧的 BeagleBone 开发板。

TI 公司还针对中国客户特别推出了 HMI 参考设计。

图 2-126

大家可以看到，在这个参考设计里，已经有完整的 HMI 图片和一套完整的 HMI 系统。

现在来看一下 TI HMI 参考设计当中的硬件资源。在 TI HMI 参考设计里，采用的是 AM3517 和 AM3505 主芯片，主频是可以跑到 600 兆赫兹，同时采用的是 128 兆的 Nand Flash 和 128 兆的 DDR2，支持 SD 卡，5 路串口，包含 RS232 和 RS485，1 路的以太网，2 路的 USB2.0 Host 接口。软件方面，TI 的参考设计采用的是 Linux 系统加 MiniGUI/QT，开发工具采用的是 ArgusSoft 软件。

以下是参考设计当中使用的软件开发工具截图。

图 2-127

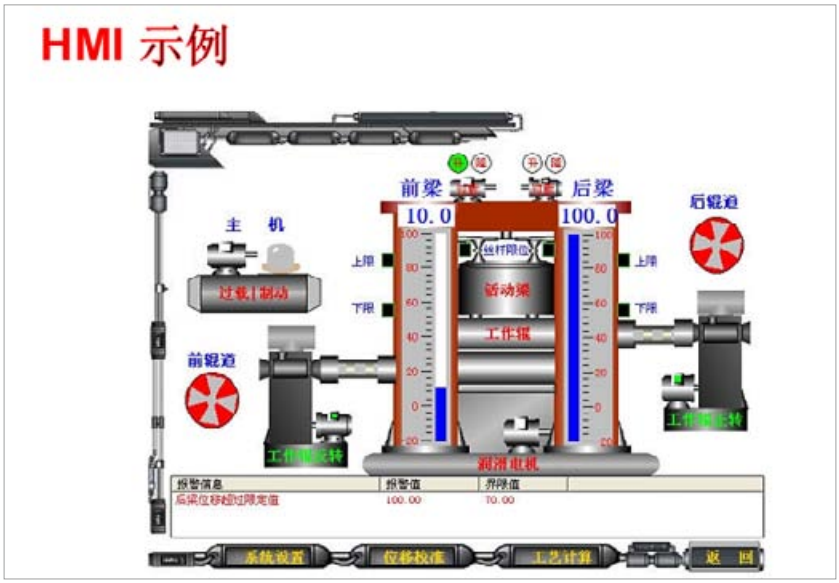


图 2-128

我们可以看到，尽管 HMI 系统中使用的绘图元素多种多样，而且界面也可能非常复杂。但是在这个设计软件中，已经提供了丰富的绘图元素。而且通过简单的输入，就可以轻松的设计显示样式、位置、参数等设置。下面是我们利用这个软件设计出来的一个样例。大家可以看出在这个样例当中，我们已经实现了一个非常复杂的 HMI 显示界面。这套参考设计有什

么样的技术特点呢？首先采用的是高清显示设备，目前已经支持 1280×800 分辨率显示，并且开发了基于 Neon 的 2D 图形加速器，拥有非常强大的 RDE 开发环境。在这里我们使用了 WYSWYG 的这种开发软件完全是图形化的开发环境，通过简单的拖拽等动作，就可以完成界面的设计。对于一些简单的、简易的 HMI 系统，甚至不需要任何编程工作。而且通过这套软件编译出来的镜像文件，可以直接在目标板上运行。同时这套开发软件拥有非常强大的在线调试功能，首先可以通过以太网的方式进行调试，另外如刚才所介绍的，所有的图形或者说是 HMI 元素都通过库的形式提供，这样就大大的降低了开发的难度。

现在看一下 TI Sitara SDK 里的一些软件架构。

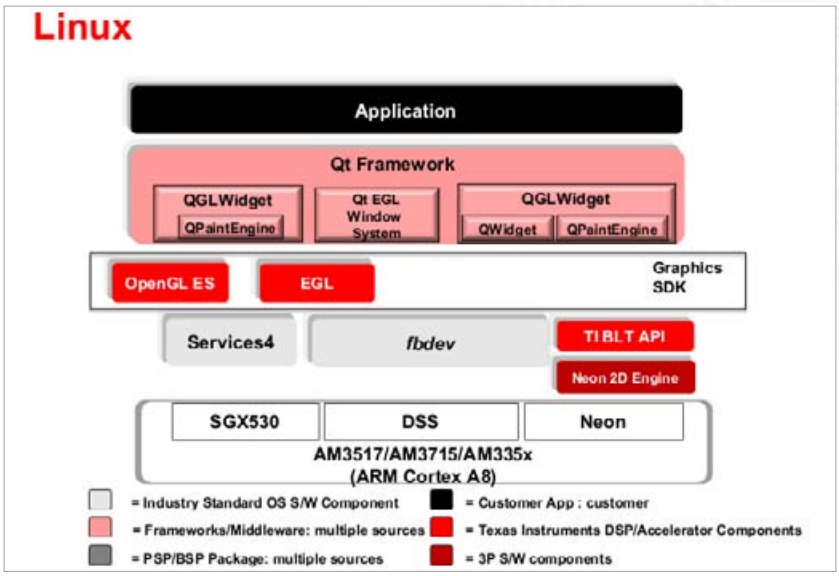


图 2-129

这一幅是 Linux下的一些软件架构。在目前 TI Sitara SDK 里已经提供了 HMI 系统中所需要的基本的框架，包括上层的 QT 程序，以及底层的驱动支持。

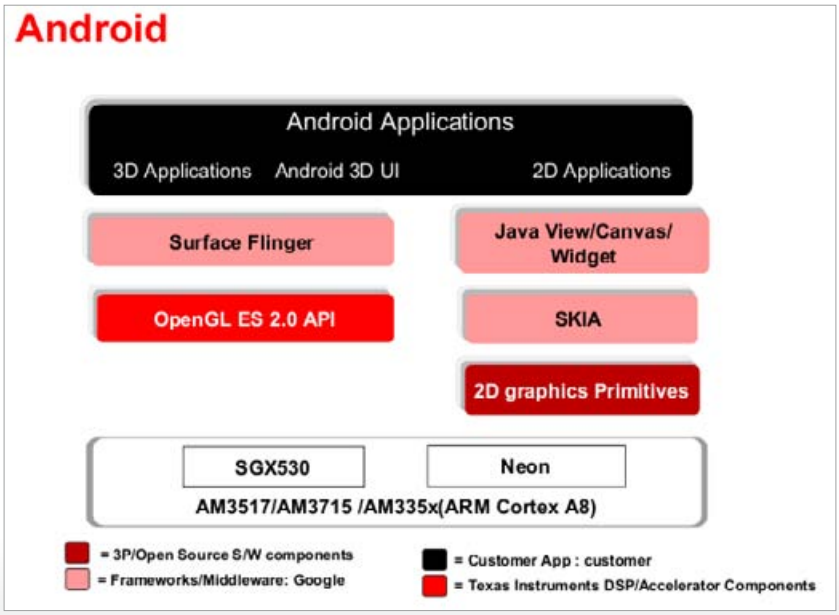


图 2-130

在 Android 系统当中也是一样，除了底层的驱动支持外，还增加了对 3D 图形的支持，并通过完整的上层接口，供上层的应用程序所调用。

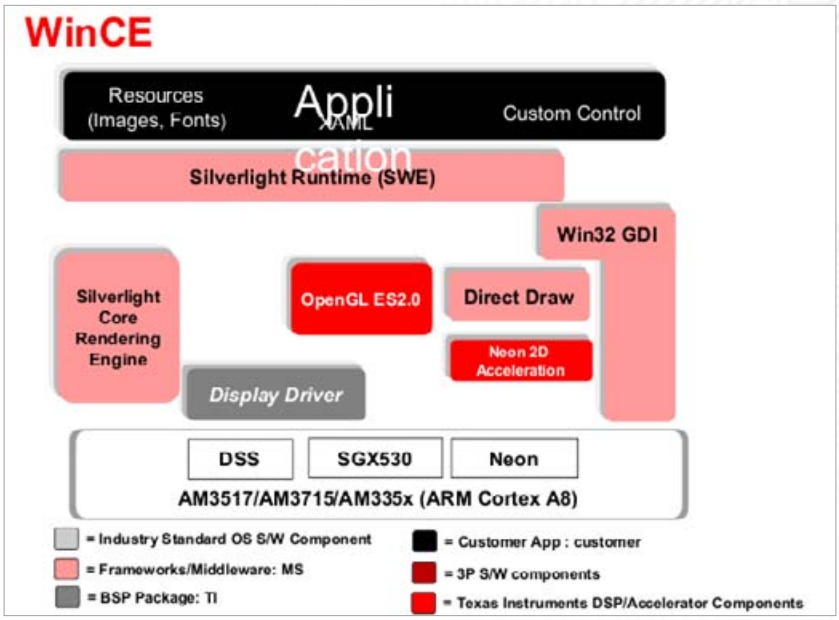


图 2-131

在 WinCE 里也提供了完整的开发，包括底层的驱动，上层的应用。在上层应用时可以基于微软提供的 Silverlight 等工具，可以非常简易的开发出 HMI 相关的软件。

HMI 系统就介绍到这里。

2.10 基于 AM335x 的 CAN 和 EtherCAT 功能演示

在工业自动化控制的运用中，工业通信总线占据了非常重要的地位，是工业自动化控制的核心，是确保整个系统的高效性和稳定性的关键。在当下流行的工业通信总线协议中，EtherCAT 和 CAN 以其高可用性、实时安全特性具有着举足轻重的地位。

在 Sitara AM335x 上，特有的 PRU 单元提供了对 EtherCAT 的支持；芯片本身还提供了两个 CAN 2.0 的接口，可满足用户对 CAN 接口的 需求。在本专题中，我们将为大家演示 AM335x 的 EtherCAT 和 CAN 的通信功能，并对整个过程进行详细的介绍：在 EtherCAT 环节，我们使用 装有 TwinCAT 软件的 PC 机作为 EtherCAT 主站，AM335x ICE EVM 作为 EtherCAT 从站，通过 PC 主机控制 ICE EVM 上的 LED 灯，进行 EtherCAT 主从通信的演示；在 CAN 环节中，我们将采用两块 GP EVM 评估板进行 CAN 通信的演示。

本节内容由德州仪器半导体有限公司技术支持工程师刘靖伟讲解。

在本次功能演示中，我们将使用装有 EtherCAT 主站协议驱动的 PC 机，作为 EtherCAT 主站，与从站 AM335x ICE 评估板进行 EtherCAT 通信。通过控制 AM335x 评估板的 LED 灯来验证 AM335x 的 EtherCAT 通信功能。

在硬件环境方面，选择 AM335x 工业通信评估板 ICE EVM 做为本次功能演示中的 EtherCAT 从站。通过对 AM335x 进行示例镜像的加载，使其内部的可编程实时单元 -PRU 具备 EtherCAT 从站通信能力，并通过网口的 PHY 实现通信；EtherCAT 主站方面，在台式电脑上安装 TwinCAT 的软件，通过该款软件的配置即可实现基于 PC 机的 EtherCAT 主站功能。TwinCAT 的主要功能是将台式电脑的网卡，安装上 EtherCAT 主站协议，使其具备实时控制能力，成为 EtherCAT 主站；EtherCAT 的链接方面，实验中直接通过网线即可完成从 EtherCAT 的主站到从站的连接。

下图介绍了关于 EtherCAT 从站使用的 AM335x 的工业评估板。

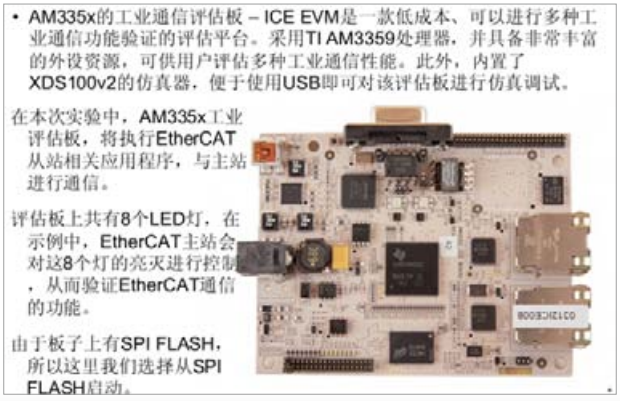


图 2-132

关于 EtherCAT 从站中示例程序可以通过以下方式可以获得。

◆ SYS/BIOS 软件开发包的下载、安装

– 在 TI 的 wiki 资源中，我们不但提供了项目源码，还提供了一些已经编译生成的镜像文件（pre-built）。

◆ SYS/BIOS SDK 软件开发包

◆ SYS/BIOS 已编译生成的可执行文件

– 链接：http://www.ti.com/tool/sysbiossdk-ind-sitara

◆ CCS 对评估板的 SPI FLASH 进行相关文件的烧写。

– 使用 CCS 对评估板的 SPI FLASH 进行烧写初始化

◆ CCS 配置和初始化 GEL 文件

◆运行 SPI FLASH 烧写程序

– 烧写 Pre-built 中 EtherCAT 相关的程序：

◆ Bootloader：启动引导文件 – boot_SPI.bin

◆ Application：应用示例程序 – ethercat_SPI_pru.bin

– 链　接：http://processors.wiki.ti.com/index.php/AM335x_SYSBIOS_Industrial_SDK_Getting_Started_Guide

主站方面的 TwinCAT 的安装与配置请参见下图网站。特别注意，在安装 TwinCAT 软件之前，确保当前的网卡型号在 TwinCAT 软件的支持列表中。

◆ TwinCAT 的安装与运行：

参考资源链接：

http://processors.wiki.ti.com/index.php/Configuring_TwinCAT_For_AM335x

下面讲解基于 AM335x 的 CAN 功能演示。功能演示中我们将使用 AM335x 的功能、接口最为丰富的通用的评估板（GPEVM），作为 CAN 通信功能演示平台。通过一端发一端收的方式，验证 CAN 总线通信的功能。

图 2-133

硬件环境方面，CAN 接口平台方面，AM335x 通用评估板是目前评估板中接口资源最为丰富的一个，通过对 CPLD 模式的选择，以及应用程序的设置，可以使能 AM335x 的 CAN 通信接口（DB9），进行 CAN 通信功能演示。

CAN 的连接方面，实验中通过对我们常用的串口线进行如下图所示的改制，就可使其成为适用 CAN 通信的串口线。通过修改线序的串口线连接两块评估板，便可进行 CAN 通信功能的演示。

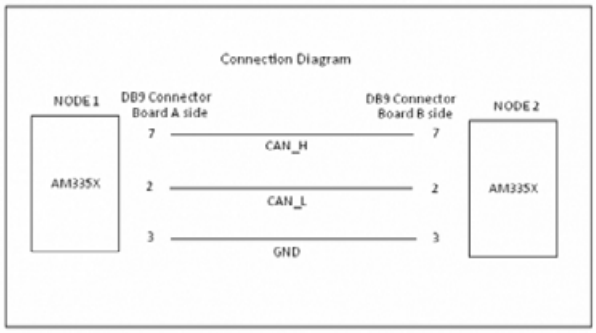


图 2-134

CAN 功能演示的操作步骤如下图所示。

图 2-135

CAN 功能演示的操作内容：

图 2-136

相关的演示视频参见 EEworld 大学堂 :>> 点击学习

第三章 EEworld 论坛网友实战篇

3.1 家居生活管理中心

I 方案策划

所要实现功能描述：

家居生活管理中心覆盖面比较广，本次方案仅针对实现利用 BB 作为家电管理中心，通过移动终端访问 BB 服务站点来控制家里的电器。

硬件设计思路：

我一个软件出身的穷吊，没有什么硬件基础，谈不上硬件设计，也没有什么思路。针对本篇作业内容，采用最简单的红外线遥控的方式，那么需要将红外线发射器连接在 BB 上。

软件设计思路：

软件方面，终端访问 BB 服务站点，利用手机显示相关页面，来代替遥控器的功能；BB 端，将接收到的页面信息，转为红外线信号对电器进行控制。

时间安排 / 项目实施细则：

第一个月用来学习 Linux 以及驱动的开发，第二个月学习页面与地层驱动的交互，第三个月前一周进行整体设计和规划，第二三周进行详细设计和编码，最后一周进行验收和调试。

预期成果：

手机或平板打开家居管理中心，通过点击页面按钮，可以实现对指定电器的控制。

II 设计框图

图 3-1

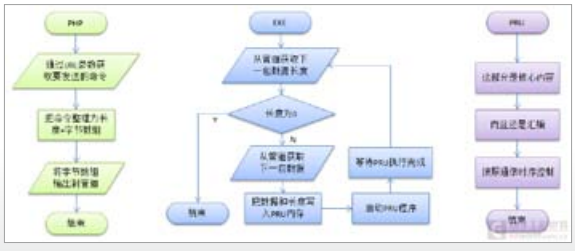


图 3-2

III 代码实现部分

PHP 部分

PHP 服务器的搭建方面参考了在 beaglebone black 上安装 lighttp，通过 PHP 教程学习了 PHP。

代码实现：

作为接收命令的起始点，使用 URL 参数来接收要通过红外线发送的数据串：

```
<html>
<head>
<title>Test Pipe Page</title>
</head>
<body>

<?php

        $str = $_REQUEST['str'];
  $fp = @fopen("////tmp//irfifo", 'w');
        echo $fp;
        echo nl2br("\n");
        $num  = 0x5a;

  $data = pack("H2", "5a");
  echo @fwrite($fp, $data);
        echo nl2br("\n");
  $data = pack("C1", strlen($str)/2);
  echo @fwrite($fp, $data);//д
        echo nl2br("\n");
  $data = pack("H*", $str);
  echo @fwrite($fp,$data);//д
        echo nl2br("\n");
  fclose($fp);
        echo nl2br("\n");
        print("Hello World!");
        echo nl2br("\n");
        print("sjtitr");
        echo nl2br("\n");
?>
<br><br>
<a href="test_pipe.php?str=00ff48b7"><-</a><br><br>
<a href="test_pipe.php?str=00ff4ab5">-></a><br><br>
</body>
</html>
```

使用的时候，就像这样咯：test_pipe.php?str=00ff48b7

这样一来，C 程序也很简单，目标是处理管道文件，然后启动 pru：

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>

#define PRU_NUM 0
#define FIFO "/tmp/irfifo"

void prurun(char *buf)
{
    unsigned int ret;
    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;

    prussdrv_init ();//Initialize the PRU
    if (prussdrv_open(PRU_EVTOUT_0))//Open PRU Interrupt
    {
        printf("prussdrv_open open failed");
        return;
    }
    prussdrv_pruintc_init(&pruss_intc_initdata);
    {
        prussdrv_pru_write_memory(PRUSS0_PRU0_DATARAM, 0, buf, buf[0]+1);
    }
    prussdrv_exec_program (PRU_NUM, "./prucode.bin");//Execute example on PRU
    prussdrv_pru_wait_event (PRU_EVTOUT_0);//Waiting for this instruction: MOV r31.
b0, PRU0_ARM_INTERRUPT+16
    prussdrv_pru_clear_event (PRU0_ARM_INTERRUPT,0);
    prussdrv_pru_disable (PRU_NUM);//Disable PRU and close memory mapping
    prussdrv_exit ();

}

int main(int argc, char** argv)
{
        char buf[64];
        int fd;
        int nread;

        if((mkfifo(FIFO, O_CREAT|O_EXCL)<0) && (errno!=EEXIST))
        {
                printf("cannot create fifo\n");
        }

        fd = open(FIFO, O_RDONLY);
        if(fd==-1)
        {
                printf("cannot open fifo\n");
```

```
                exit(1);
        }

        while(1)
        {
                int i = 0;
                memset(buf, 0, sizeof(buf));
                read(fd, buf, 1);
                if(buf[0]!=0x5A)
                {
                //         printf("%X,", buf[0]);
                        continue;
                }
                read(fd, buf, 1);
                if(buf[0] == 0)
                {
                        break;
                }
                read(fd, &buf[1], buf[0]);
                printf("%d: ", buf[0]);
                while(i < buf[0])
                {
                        i++;
                        printf("%X,",buf[i]);
                }
                printf("\n");
                prurun(buf);
        }
        unlink(FIFO);
        exit(0);
}
```

PRU 部分之熟悉 PRU

pruss 初次探秘

AM335x 等芯片上自带的，独立于 ARM CPU 运行的子系统。其时钟频率为 200MHz，可以直接控制特定的 IO 口，可以达到非常高的实时性要求。一般两种情况需要用到它：一是 linux 系统的实时性不满足要求的时候；二是芯片的功能模块不够用的时候（比如你想要 10 个 UART，但芯片上只有 6 个，那你可以用它再创造 4 个）。详细内容参见 AM335x 的 PRUSSv2 简介与使用。

参照该文，建立 pruss 的开发环境。最后可以操纵 pruss 对端口进行控制。

把前面控制小灯的实验，搬到 pru 中来进行。先用命令行把端口设置妥当：

```
echo 44 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio44/direction
echo 1 > /sys/class/gpio/gpio44/value
```

之后 pru 只需要控制输出值的 0、1 就可以了。

```
//prucode.p
.origin 0
.entrypoint START

//Refer to this mapping in the file - prussdrvincludepruss_intc_mapping.h
#define PRU0_ARM_INTERRUPT      19
#define CONST_PRUCFG            C4

//Refer to AM335X Technical Reference Manual & BBB SRM
#define GPIO1 0x4804c000
#define GPIO_CLEARDATAOUT 0x190
#define GPIO_SETDATAOUT 0x194

START:
    // Enable OCP master port
    LBCO      r0, CONST_PRUCFG, 4, 4
    CLR       r0, r0, 4         // Clear SYSCFG[STANDBY_INIT] to enable OCP
master port
    SBCO      r0, CONST_PRUCFG, 4, 4

    MOV r1, 3 // loop 3 times
LOOP0:
    MOV r2, 1<<12
    MOV r3, GPIO1 | GPIO_SETDATAOUT
    SBBO r2, r3, 0, 4

    MOV r0, 20000000
DEL1:
    SUB r0, r0, 1
    QBNE DEL1, r0, 0

    MOV r2, 1<<12
    MOV r3, GPIO1 | GPIO_CLEARDATAOUT
    SBBO r2, r3, 0, 4

    MOV r0, 20000000
DEL2:
    SUB r0, r0, 1
    QBNE DEL2, r0, 0

    SUB r1, r1, 1
    QBNE LOOP0, r1, 0

    // Send notification to Host for program completion
    MOV       r31.b0, PRU0_ARM_INTERRUPT+16

    // Halt the processor
    HALT
```

这段代码，参考原文编译加载执行，现象是小灯一闪一闪亮 3 次，时间上相当精准。

但是要实现利用这代码来控制实现红外线遥控器波形，还是不行。所以必须要继续研究。

与 pruss 交换数据

想用 pruss 控制波形，第一件麻烦事就是要发送的数据由 C 语言程序传递到 pru 程序。简单说，其实 pru 使用的空间（无论数据还是代码），都存在在我们的 flat 内存空间上。所以只要找对地方，把数据写入。

```
//mytest.c
#include <stdio.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>

#define PRU_NUM 0

int main (void)
{
    unsigned int ret;
    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;

    prussdrv_init ();//Initialize the PRU
    if (prussdrv_open(PRU_EVTOUT_0))//Open PRU Interrupt
    {
        printf("prussdrv_open open failed");
        return (-1);
    }
    prussdrv_pruintc_init(&pruss_intc_initdata);
    {
        unsigned int v = 10;
        prussdrv_pru_write_memory(PRUSS0_PRU0_DATARAM, 0, &v, 4);
    }
    prussdrv_exec_program (PRU_NUM, "./prucode.bin");//Execute example on PRU
    prussdrv_pru_wait_event (PRU_EVTOUT_0);//Waiting for this instruction: MOV r31.
b0, PRU0_ARM_INTERRUPT+16
    prussdrv_pru_clear_event (PRU0_ARM_INTERRUPT,0);
    prussdrv_pru_disable (PRU_NUM);//Disable PRU and close memory mapping
    prussdrv_exit ();

    return(0);
}
```

注意到其中的

```
    {
        unsigned int v = 10;
        prussdrv_pru_write_memory(PRUSS0_PRU0_DATARAM, 0, &v, 4);
    }
```

这就是向 pru 单元写入数据（事实上传程序进去，也是类似的就这样一个实质上的数据 copy）。

而我们在 pru 中，则先对数据进行一个简单分析——再确定闪灯的次数。

```
//prucode.p
.origin 0
.entrypoint START
//Refer to this mapping in the file - prussdrvincludepruss_intc_mapping.h
#define PRU0_ARM_INTERRUPT      19
#define CONST_PRUCFG            C4
//Refer to AM335X Technical Reference Manual & BBB SRM
#define GPIO1 0x4804c000
#define GPIO_CLEARDATAOUT 0x190
#define GPIO_SETDATAOUT 0x194
START:
    // Enable OCP master port
    LBCO      r0, CONST_PRUCFG, 4, 4
    CLR       r0, r0, 4         // Clear SYSCFG[STANDBY_INIT] to enable OCP
master port
    SBCO      r0, CONST_PRUCFG, 4, 4

    //MOV r1, 3 // loop 3 times
    MOV r2, 0
    LBBO r1, r2, 0, 4

LOOP0:
    MOV r2, 1<<12
    MOV r3, GPIO1 | GPIO_SETDATAOUT
    SBBO r2, r3, 0, 4

    MOV r0, 20000000
DEL1:
    SUB r0, r0, 1
    QBNE DEL1, r0, 0

    MOV r2, 1<<12
    MOV r3, GPIO1 | GPIO_CLEARDATAOUT
    SBBO r2, r3, 0, 4

    MOV r0, 20000000
DEL2:
    SUB r0, r0, 1
    QBNE DEL2, r0, 0

    SUB r1, r1, 1
    QBNE LOOP0, r1, 0

    // Send notification to Host for program completion
    MOV       r31.b0, PRU0_ARM_INTERRUPT+16
    // Halt the processor
    HALT
```

成功运行。

控制红外线的时序，交给 pru 就可以，没必要再加一个内核驱动了。接下来，应用程序从 php 获取数据，然后传递给 pru，再让 pru 执行生成红外线时序的代码。

PHP 通过命名管道和应用程序交换数据无疑就是 PHP 怎样把数据交给到应用程序里了。实现一个监听程序：这里做了个很简单的协议。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FIFO "/tmp/irfifo"

int main(int argc, char** argv)
{
        char buf[64];
        int fd;
        int nread;

        if((mkfifo(FIFO, O_CREAT|O_EXCL)<0) && (errno!=EEXIST))
        {
                printf("cannot create fifo\n");
        }
        fd = open(FIFO, O_RDONLY);
        if(fd==-1)
        {
                printf("cannot open fifo\n");
                exit(1);
        }
        while(1)
        {
                int i = 0;
                memset(buf, 0, sizeof(buf));
                read(fd, buf, 1);
                if(buf[0]!=0x5A)
                {
                //      printf("%X,", buf[0]);
                        continue;
                }
                read(fd, buf, 1);
                if(buf[0] == 0)
                {
                        break;
                }
                read(fd, &buf[1], buf[0]);
                printf("%d: ", buf[0]);
                while(i < buf[0])
                {
                        i++;
```

```
                        printf("%X,",buf[i]);
                }
                printf("\n");
        }
        unlink(FIFO);
        exit(0);
}
```

其次，不要着急 PHP，再写个工作程序给监听程序送些试验数据：很简单，执行时候的第一个参数字符串和它的长度作为内容传递给 pipe

```
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define FIFO "/tmp/irfifo"

int main(int argc, char** argv)
{
        char buf[64];
        int fd;
        int nread;
        fd = open(FIFO, O_WRONLY);
        if(fd==-1)
        {
                printf("cannot open fifo\n");
                exit(1);
        }
        buf[0] = 0x5A;
        if(argc <=1)
        {
                buf[1] = 0;
        }
        else
        {
                buf[1] = strlen(argv[1]);
                memcpy(&buf[2], argv[1], buf[1]);
        }
        if(write(fd, &buf[0], buf[1]+2)==-1)
        {
                printf("error\n");
        }
        close(fd);
        exit(0);
}
```

程序只是实现基本功能，并不完美。

经过试验，确信监听程序可用了。接下来准备 PHP。

如果利用 php 处理字节流数据，很好很强大的一个函数 pack。最终形成了 php 的测试页面，可以把字节流传递给 pipe 文件。

```
<html>
<head>
<title>Test Pipe Page</title>
</head>
<body>
<?php
        $str = $_REQUEST['str'];
  $fp = @fopen("////tmp//irfifo", 'w');
        echo $fp;
        echo nl2br("\n");
        $num  = 0x5a;
  $data = pack("H2", "5a");
  echo @fwrite($fp, $data);
        echo nl2br("\n");

  $data = pack("C1", strlen($str)/2);
  echo @fwrite($fp, $data);//D'
        echo nl2br("\n");

  $data = pack("H*", $str);
  echo @fwrite($fp,$data);//D'
        echo nl2br("\n");
  fclose($fp);
        echo nl2br("\n");
        print("Hello World!");
        echo nl2br("\n");
        print("sjtitr");
        echo nl2br("\n");
?>
</body>
</html>
```

PRU 部分之项目代码实现部分

最后是 PRU 程序，使用汇编语言完成，根据给下来的数据，执行红外编码的时序：

```
//prucode.p
.origin 0
.entrypoint START
//Refer to this mapping in the file - prussdrvincludepruss_intc_mapping.h
#define PRU0_ARM_INTERRUPT      19
#define CONST_PRUCFG            C4
//Refer to AM335X Technical Reference Manual & BBB SRM
#define GPIO1 0x4804c000
#define GPIO_CLEARDATAOUT 0x190
```

```
#define GPIO_SETDATAOUT 0x194
START:
    // Enable OCP master port
    LBCO      r0, CONST_PRUCFG, 4, 4
    CLR       r0, r0, 4         // Clear SYSCFG[STANDBY_INIT] to enable OCP
master port
    SBCO      r0, CONST_PRUCFG, 4, 4
    MOV r5, 0
    LBBO r4, r5, 0, 1
    AND r4, r4, 255
LEADER:
    MOV r1, 340
    CALL WAV_S
    MOV r0, 450000
LEADER_T:
    SUB r0, r0, 1
    QBNE LEADER_T, r0, 0
NEXT_BYTE:
    ADD r5, r5, 1
    LBBO r6, r5, 0, 1
    MOV r7, 8
NEXT_BIT:
    MOV r1, 21
    CALL WAV_S
    AND r0, r6, 1<<7
    QBNE BIT_1, r0, 0
BIT_0:
    MOV r0, 56500
    JMP BIT_T
BIT_1:
    MOV r0, 169000
BIT_T:
    SUB r0, r0, 1
    QBNE BIT_T, r0, 0
    LSL r6, r6, 1
    SUB r7, r7, 1
    QBNE NEXT_BIT, r7, 0
BYTE_OK:
    SUB r4, r4, 1
    QBNE NEXT_BYTE, r4, 0

STOP_S:
    MOV r1, 21
    CALL WAV_S
    MOV r0, 5000000
STOP_T:
    SUB r0, r0, 1
    QBNE STOP_T, r0, 0
    // Send notification to Host for program completion
    MOV       r31.b0, PRU0_ARM_INTERRUPT+16
    // Halt the processor
```

```
    HALT
WAV_S:
    MOV r2, 1<<12
    MOV r3, GPIO1 | GPIO_SETDATAOUT
    SBBO r2, r3, 0, 4
    MOV r0, 850
WAV_1:
    SUB r0, r0, 1
    QBNE WAV_1, r0, 0
    MOV r2, 1<<12
    MOV r3, GPIO1 | GPIO_CLEARDATAOUT
    SBBO r2, r3, 0, 4
    MOV r0, 1800
WAV_0:
    SUB r0, r0, 1
    QBNE WAV_0, r0, 0
    SUB r1, r1, 1
    QBNE WAV_S, r1, 0
RET
```

视频中出现的操控界面调用代码

实物展示：功能演示视频

点击查看详情：>> 家居生活管理中心

3.2 BeagleBone 建立自己的家庭多媒体服务器

用 BBB 制作了一个家庭多媒体服务器，实现局域网的 DLNA 播放，在电视上可以直接播放插在狗板上 u 盘里的视频了。

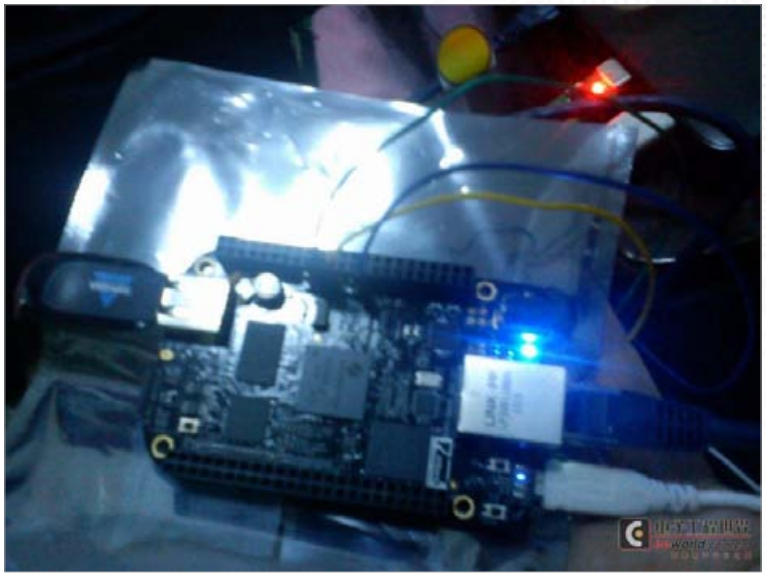


图 3-3

图 3-4

图 3-5

图 3-6

图 3-7

基本思路是 Ubuntu+minidlna，下面简单说明下实现方法：

1、网络。

我是把狗板直接用网线连笔记本，笔记本用无线上网共享过去的，方便串口调试。网络地址用的自动，所以平时直接连接路由器即可。

执行一次 ping www.baidu.com 测试网络

2、安装 Ubuntu.

看我的上一篇 http://bbs.eeworld.com.cn/thread-427100-1-1.html

或高手 lonerzf 的 http://bbs.eeworld.com.cn/thread-426934-1-1.html 系列，我受益匪浅。

3、安装 minidlna

先更新

```
sudo apt-get update
sudo apt-get upgrade
```

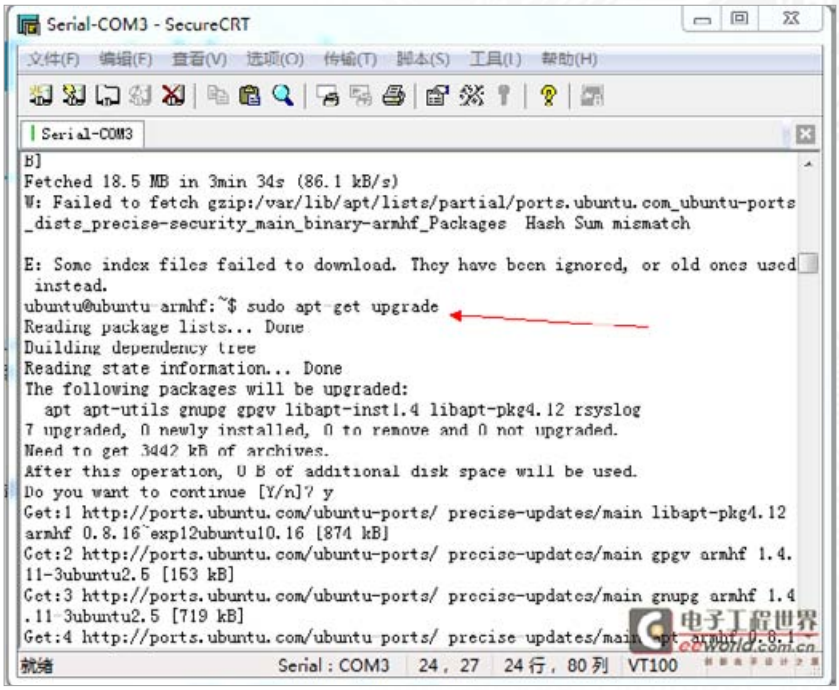


图 3-8

再安装 minidlna

sudo apt-get install minidlna

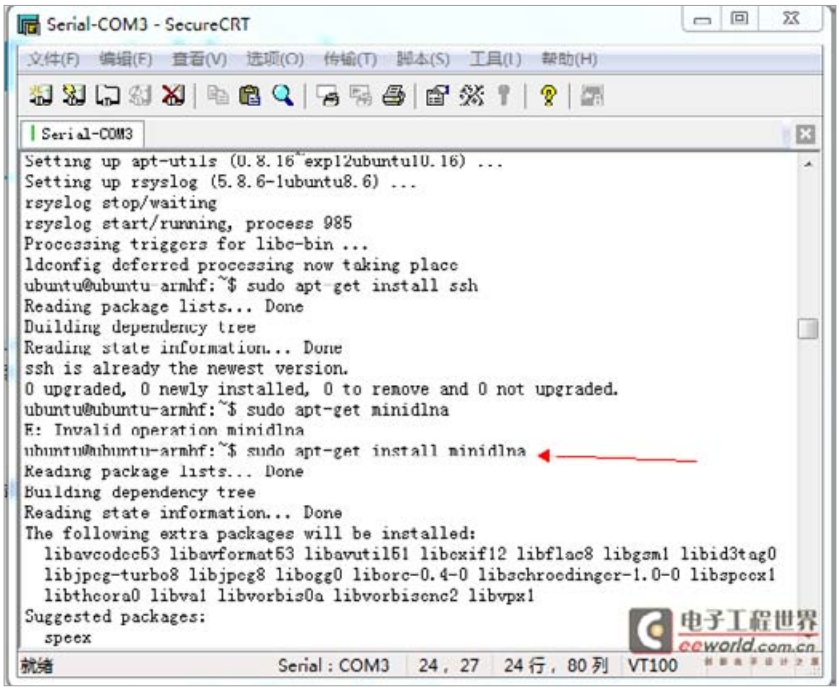


图 3-9

网站看这里 https://wiki.archlinux.org/index.php/MiniDLNA#Installation

配置

sudo vi /etc/minidlna.conf

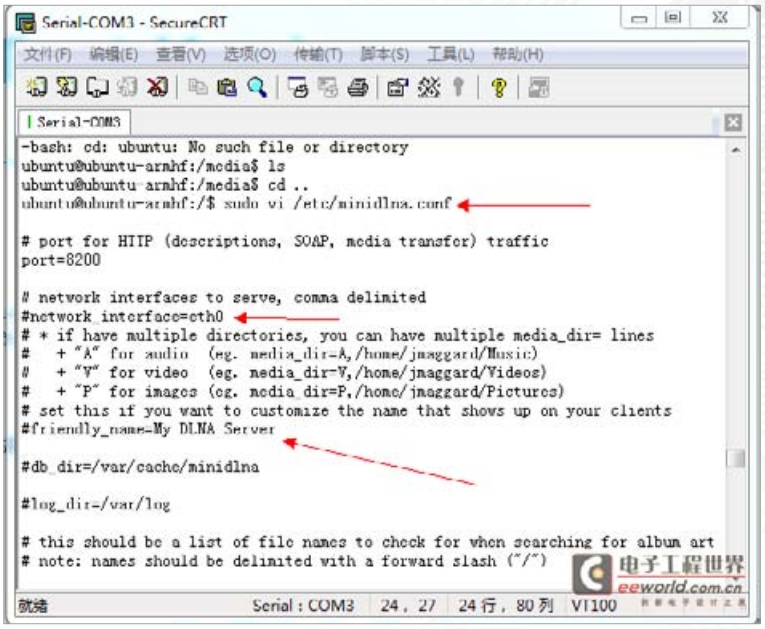


图 3-10

我们主要修改的是搜索路径，加入

media_dir=V,/tmp

因为我们一会要把多媒体文件的 u 盘枚举到 /tmp 文件夹

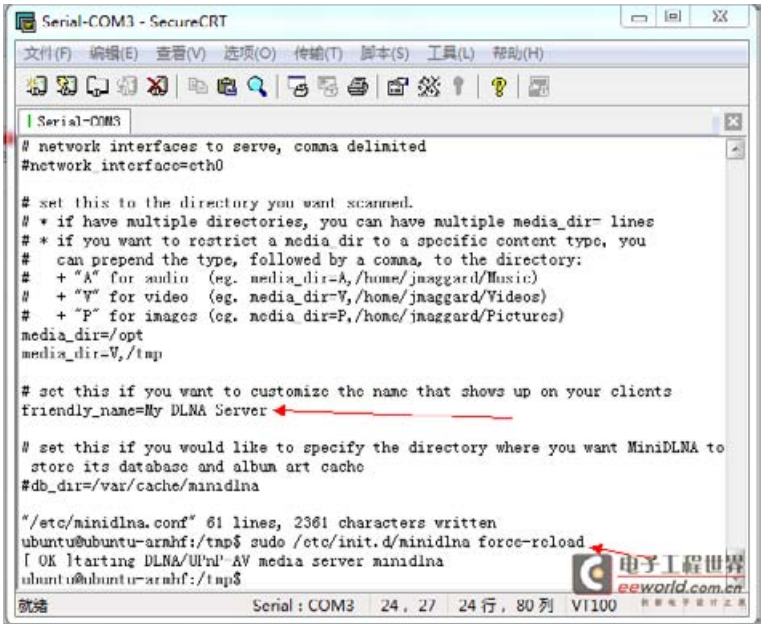


图 3-11

friendly_name 服务名称，在其它设备中看到的名称

inotify 设置为 true，将自动发现媒体目录中的新文件

一般也就设置以上的参数就可以了。

:wq 保存退出，vi 用法自己百度

4、重启 minidlna 服务，使配置生效

sudo /etc/init.d/minidlna force-reload

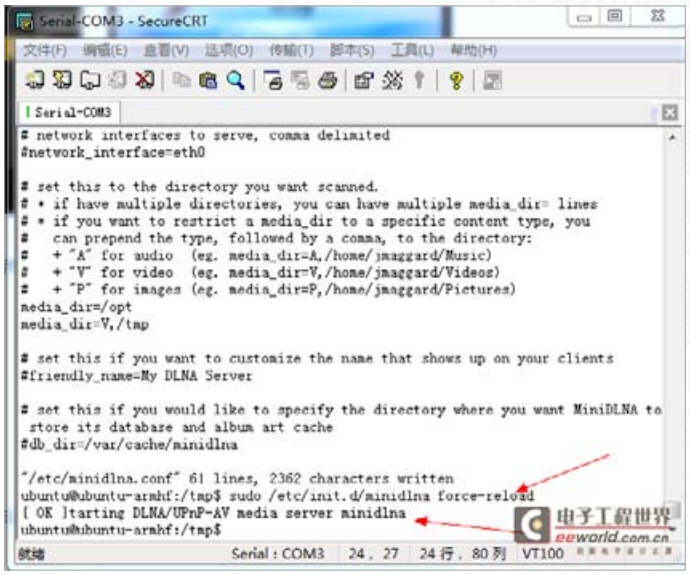


图 3-12

5、准备个 u 盘，移动硬盘没试，可能需要外部供电。放入一些视频之类的

插入狗板的 USB host 口，在终端那查看

sudo fdisk -l

一般是在 /dev/sda1

枚举到我们设置的目录

sudo mount /dev/sda1 /tmp

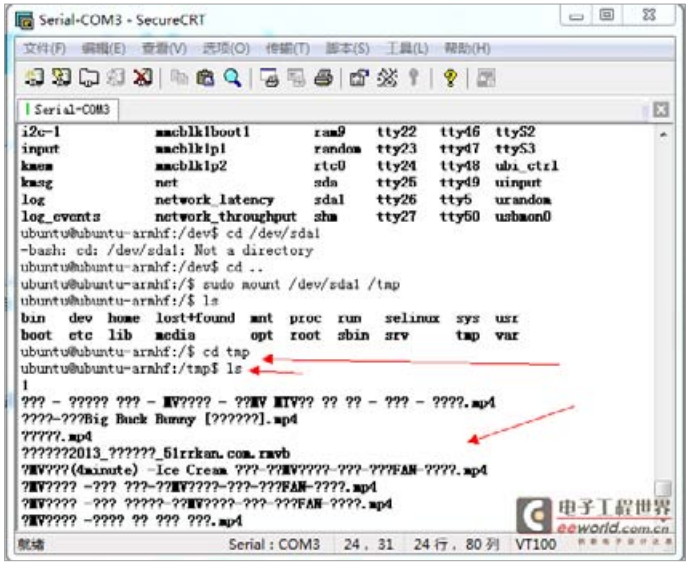


图 3-13

查看电脑网络里面，就看到你的 dlna 服务器了，用局域网内支持 dlna 服务的手机，电视，电脑进入即可播放

图 3-14

图 3-15

至此完结

点击查看详情：>>BeagleBone 建立自己的家庭多媒体服务器

3.3 基于 BB 及 FPGA 的高速数据采集及 DDS 信号发生器设计与实现

离活动结束的时间越来越近了，之前针对该方案做了一些工作，还有一些工作要完成，零散地在论坛里发过两个进展贴，其实部分已经在实际项目中有用到，比如 DDS 信号发生器用于控制扫描振镜，尽管项目中实现的细节及采用的软硬件与这次提交的方案还是有所不同，但基本思路是吻合的。本来打算做到这就算了，后来考虑了下，认为还是应该把这个方案相对完整地做出来，这样一方面可以很好的锻炼自己的能力，另一方面可以增加自己在论坛的参与度，EEworld 在我工作的两年多时间里还是让我受益匪浅。

这个活动从论坛奖励给我的的 88 个 E 芯币开始，我用它们买了两本关于信号完整性方面的书籍，完整的看了一本，另一本还在睡觉，至少得有一个稍微像样的结尾吧，于是利用最近工作上稍微空闲的一周多时间，将该方案当作一个小项目完整地做了一下，花了一个周末整理文档，整个做下来感觉还是很有收获的，特别是调试数据采集的代码，涉及计算机与 BB 数据传输的同步以及 BB 与 FPGA 数据传输的同步，好好的把 Linux 驱动同步机制及应用程序多线程理解了一番，调试时真是机关重重，陷阱多多。由于时间有限，这个方案还存在很多问题，特别是模拟部分的设计，后面会根据实际项目中的需要进行改进或重新设计。

3.3.1 方案总体设计

活动开始时最初提交的初步方案：http://bbs.eeworld.com.cn/thread-432527-1-1.html，图 3-16 是方案总体设计原理图，其中 CAN 及 UART 部分是实际项目中用到的功能，本方案没有用到，可以作为该方案的后续扩展部分。

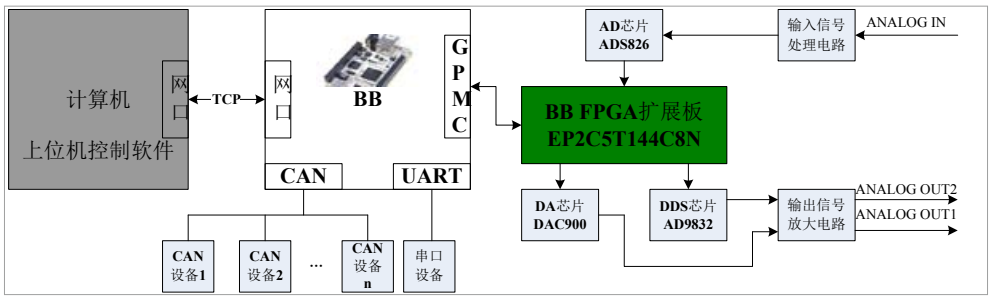


图 3-16: 方案总体设计原理图

该方案中，计算机作为上位机，BB 作为下位机，上位机和下位机通过以太网口实现基于 TCP 协议的 SOCKET 通信，其中计算机上位机控制软件为 TCP Client 端，BB 软件为 TCP Server 端，另外 BB 通过 TI AM335X 的 GPMC 接口实现与 BB FPGA 扩展板的通信。

这样，上位机控制软件通过网口发送信号波形命令及信号频率参数给 BB Linux 软件，BB Linux 软件处理接收到的信号波形命令及信号频率参数生成 DDS RAM 区的数据及相位累加器的频率控制字并通过 GPMC 送到 FPGA 中，最后由 FPGA 控制 DA 芯片完成 DDS 信号发生的功能；另外一种 DDS 信号发生的方法采用 DDS 芯片完成，BB 的 GPIO 口产生控制 DDS 芯片 AD9832 的时序，FPGA 用作连线，这种方法较前叙方法灵活性要差很多，在本方案中一起实现了。

另外，FPGA 控制 AD 芯片高速（可能谈不上高速）采集模拟信号输入，FPGA 内

部做一个数据缓存，BB 通过 GPMC 读取缓存数据并通过以太网口将数据送给上位机。FPGA 传输数据给 BB，BB 传输数据给上位机，上位机利用得到的数据实时成像，灰度图像的灰度值即为采集得到的数字量，这里面利用成像的行帧同步机制来同步 FPGA 与 BB 的 GPMC 传输、BB 与上位机的以太网 TCP 协议传输以及成像的速度，这一块基本功能实现了，目前点采样频率 25KHz，一帧 512*512 的灰度图像理论上的成像时间为 512*512/25KHz=10.48576s，实际成像时间在 11s 以内，说明 25KHz 的采样频率对于 GPMC 以及以太网来说没问题，可能行与行之间有一定的时间间隔，帧与帧之间也有一定的时间间隔以保证上述的同步性。

该方案的硬件部分扩展了一块 BB FPGA 扩展板，即方案总体设计原理图中绿色部分，BB FPGA 扩展板原理图、PCB 图 pdf 下载及相应的实物图见之前的帖子：http://bbs.eeworld.com.cn/thread-435781-1-1.html，从成本角度考虑，FPGA 芯片采用相对比较便宜的 Altera 公司的 EP2C5T144C8N，在这个原理验证方案中基本够用了，用于数据采集的 10 位 AD 芯片 ADS 826，用于 FPGA DDS 的 10 位 DA 芯片 DAC900 以及 DDS 芯片 AD9832 也都是从 TI 和 ADI 公司官网免费申请的，在模拟部分，无论是 DA 输出模拟信号还是 AD 输入模拟信号都是使用易受干扰的单端正电压信号，输出模拟信号的放大也没有考虑到信号失真等信号完整性问题，这些是硬件后续需要深入改进的地方，用 E 金币买的两本信号完整性方面的书籍也许可以派上点用场，这方面的知识我很欠缺。

图 3-17 是方案调试过程的照片。

图 3-17：方案调试过程照片

该方案的软件包括三部分：

（1）计算机端的基于 Visual Studio2010 C# 的控制及数据采集成像软件；

（2）BB 端的基于 Linux C 的 DDS 芯片 AD9832 驱动、读写 FPGA 的 GPMC 驱动及应用程序，包括 TCP Server Socket、DDS 芯片控制、FPGA DDS 波形数据及频率控制字设置、数据采集与传输以及项目中调试过的 CAN 通信程序；

（3）FPGA 端的基于 QuartusII Verilog 与原理图输入结合的 FPGA 程序，包括硬连接 BB 与 DDS 芯片模块、用 FPGA 搭建的 DDS 模块、数据采集与传输模块。

下面将简单介绍这三部分的实现，并最终给出所有的源程序代码。

3.3.2 计算机端软件实现

（1）开发平台及语言为：

Visual Studio2010 C#

（2）主要功能如下：

TCP Client，发送 DDS 芯片及 FPGA DDS 设置命令；
发送数据采集开始及停止命令；
接收数据并实现成像。

（3）网络通信应用层协议如图 3-18 所示：

第 0 字节	第 1 字节	第 2-5 字节	第 6 字节
开始标志	控制命令	频率 / 幅值 / 保留	结束标志
0xbb	0x01：控制 FPGA DDS 产生正弦波 0x02：控制 FPGA DDS 产生斜波 0x03：控制 FPGA DDS 产生方波 0x11：设置 DDS 芯片输出频率	波形频率 =Byte[2]*256*256*256+ Byte[3]*256*256+ Byte[4]*256+ Byte[5]	0xee
	0x04：控制 FPGA DDS 输出直流	幅值 =Byte[2]*256*256*256+ Byte[3]*256*256+ Byte[4]*256+ Byte[5]	
	0x21：开始数据采集 0x22：停止数据采集 0x31：一帧数据结束	保留	

图 3-18：网络通信应用层协议

（4）软件界面说明如图 3-19 所示：

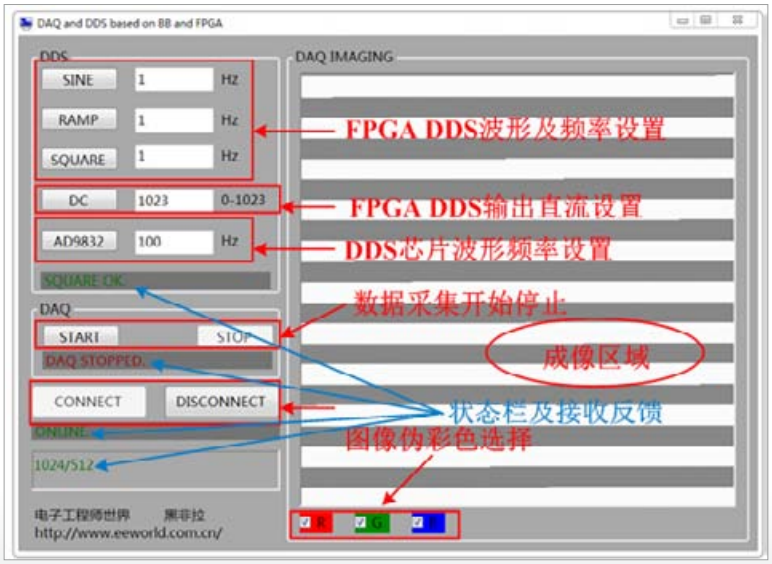


图 3-19：计算机端软件界面

（5）二维成像功能说明：

上位机实现了二维成像功能，将采集到的数据转换成点对应的灰度值，由于 AD 是 10 位的，而灰度值只有 8 位，所以在转换的过程中要将采集的数据除以 4，最终实时动态地成出 512*512 的图像，也就是每行 512 个点，每帧 512 行。程序采用两个定时器线程，分别用来采集数据及实时成像，通过全局变量 line_index（采集行索引）及 print_index（成像行索引）的关系用来严格同步采集与成像的过程。数据采集的定时器线程源码如下，标红加粗的代码是实现采集与成像行帧同步的关键部分：

```
private void timer_daq_Tick(object sender, EventArgs e)//数据采集定时器线程
        {
            int len;

            if (line_index < LINES)//一帧数据未采集完成
            {
                if (ns.CanRead)
                {
                    if (ns.DataAvailable)
                    {
                        try
                        {
                            len = ns.Read(BUF[line_index], 0, 1024);
                            line_index++;
                            richTextBox_recv.Text = len.ToString() + "/" + line_
index.ToString();//显示每行点数和对应行索引
                        }
                        catch { }
                    }

                }
            }
            if (line_index == LINES)
                line_index = line_index + 2;//显示比采集慢两行

            if (print_index == LINES)
            {
                if (daq_flag)
                {
                    send_cmd_line();//发送一帧结束命令
                    line_index = 0;//置零采集行索引
                    print_index = 0; //置零成像行索引
                }
                else//停止数据命令已经发送，等待一帧数据打印完成
                {
                    timer_daq.Enabled = false;
                    timer_print.Enabled = false;
                    timer_recv.Enabled = true;
```

```
                    pictureBox_daq.Enabled = false;
                    groupBox_DDS.Enabled = true;
                    button_stop.Enabled = false;
                    button_start.Enabled = true;
                    button_disconnect.Enabled = true;
                    label_daq.ForeColor = Color.DarkRed;
                    label_daq.Text = "DAQ STOPPED.";
                }
            }
        }
```

成像的定时器线程源码如下，标红加粗的代码是实现动态成像的关键代码，包括计算灰度值、设置灰度及伪彩色、填充灰度及伪彩色及保存并显示图像四个步骤：

```
private void timer_print_Tick(object sender, EventArgs e)
        {


            if (line_index > print_index + 2)//显示比采集两行
            {
                for (int i = 0; i < POINTS; i = i + 2)
                {
                    int val = (BUF[print_index][i] + (BUF[print_index][i + 1] &
0x3) * 256) / 4;//计算点灰度值
                    Color c = Color.FromArgb(red_en * val, green_en * val, blue_
en * val);//设置灰度及伪彩色
                    gp.FillRectangle(new SolidBrush(c), i / 2, print_index, 1, 1);//
填充灰度及伪彩色
                }
                print_index++;
            }


            gp.Save();
            pictureBox_daq.Image = bmp;//显示图像
        }
```

图 3-20 展示了分别采集 1Hz 的方波、斜波及正弦波成的图像，其中方波成像还设置了绿色伪彩色。从图中可以看出，方波的条纹明暗交替，条纹内灰度一致，或全白，代表方波的低电平，或全灰，代表方波的高电平；斜波的条纹内明暗均匀变化，条纹间明暗突变，这与斜波的波形非常吻合；正弦波明暗变化如正弦波的波峰波谷交替一般。另外一个很重要的发现是，一幅图像的条纹基本为 10 个到 11 个之间，根据波形 1Hz 的频率可以推算出一帧图像的成像时间在 10s 到 11s 之间，与方案总体设计中的理论计算一帧成像时间吻合。

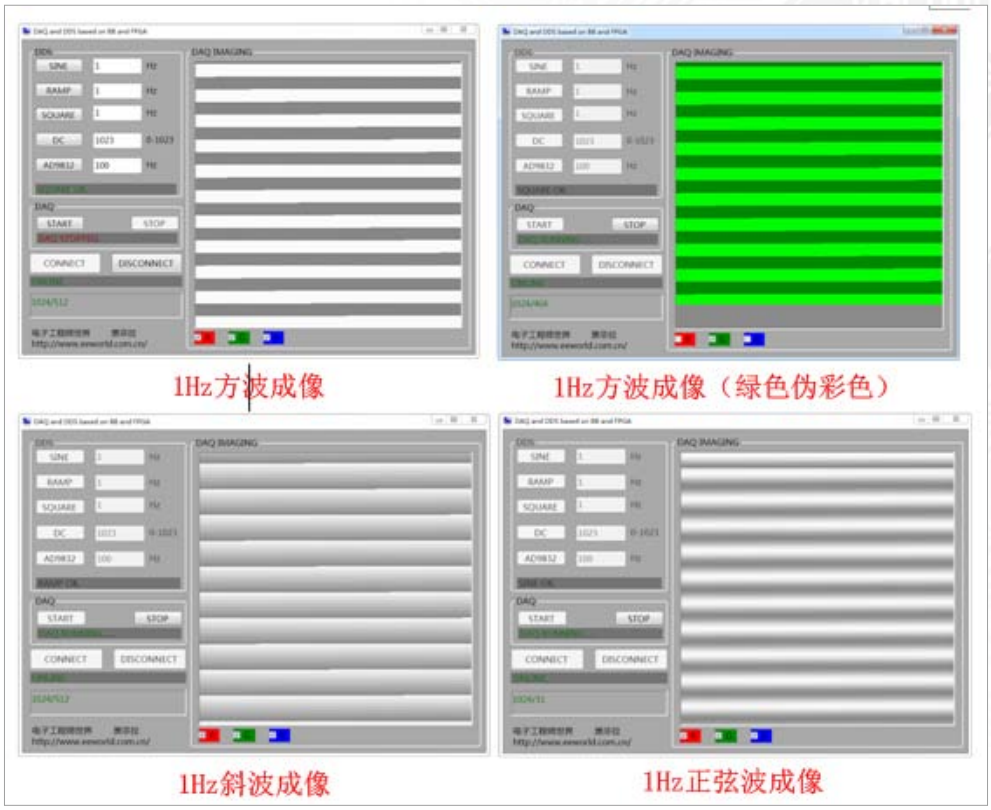


图 3-20：1Hz 的方波、斜波及正弦波

（6）计算机端软件的全部源代码：DDS_DAQ_TcpClient.rar

3.3.3 BB 端设备驱动及应用程序实现

（1）开发平台及语言为：

Linux C

（2）主要包括如下三个层次：

系统层：
BB 管脚配置，对应板文件的修改。
驱动层：
DDS 芯片 AD9832 驱动；
读写 FPGA 的 GPMC 驱动。
应用层：
TCP Server Socket 通信；
DDS 芯片控制；
GPMC 读写 FPGA：DDS 波形数据及频率控制字设置（GPMC 写），数据采集与传输（GPMC 读）；
CAN 通信（本方案没有用到，实际项目中用过）。

（3）系统层的 BB 管脚配置

需要修改板文件 board-am335xevm.c 的内容，该文件路径如下：

root@gao:~# cd bone-linux/arch/arm/mach-omap2/

root@gao:~/bone-linux/arch/arm/mach-omap2# ls board-am335xevm.c

board-am335xevm.c

关键部分代码：

```
/* Beaglebone Rev A3 and after */
static struct evm_dev_cfg beaglebone_dev_cfg[] = {
        {tps65217_init,DEV_ON_BASEBOARD, PROFILE_NONE},
        {mii1_init,     DEV_ON_BASEBOARD, PROFILE_NONE},
        {usb0_init,     DEV_ON_BASEBOARD, PROFILE_NONE},
        {usb1_init,     DEV_ON_BASEBOARD, PROFILE_NONE},
        {mmc0_init,     DEV_ON_BASEBOARD, PROFILE_NONE},
        {dcan0_beaglebone_init, DEV_ON_BASEBOARD, PROFILE_NONE},
        {dcan1_beaglebone_init, DEV_ON_BASEBOARD, PROFILE_NONE},
        {bone_fpga_init, DEV_ON_BASEBOARD, PROFILE_NONE},//配置相关管脚
        {NULL, 0, 0},
};

static void bone_fpga_init(int evm_id, int profile)
```

```
{
        setup_pin_mux(fpga_pin_mux);
}
/* Pin mux for fpga module */
static struct pinmux_config fpga_pin_mux[] = {
        //配置GPMC相关管脚
        //配置GPMC数据端口
        {"gpmc_ad0.gpmc_ad0",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad1.gpmc_ad1",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad2.gpmc_ad2",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad3.gpmc_ad3",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad4.gpmc_ad4",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad5.gpmc_ad5",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad6.gpmc_ad6",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad7.gpmc_ad7",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad8.gpmc_ad8",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad9.gpmc_ad9",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad10.gpmc_ad10",         OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad11.gpmc_ad11",         OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad12.gpmc_ad12",         OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad13.gpmc_ad13",         OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad14.gpmc_ad14",         OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"gpmc_ad15.gpmc_ad15",    OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        //配置GPMC地址端口
        {"lcd_data0.gpmc_a0",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_data1.gpmc_a1",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_data2.gpmc_a2",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_data3.gpmc_a3",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_data4.gpmc_a4",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_data5.gpmc_a5",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_data6.gpmc_a6",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_data7.gpmc_a7",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_vsync.gpmc_a8",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_hsync.gpmc_a9",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
```

```
        {"lcd_pclk.gpmc_a10",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
        {"lcd_ac_bias_en.gpmc_a11",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT |
AM33XX_PULL_DISA},
        {"lcd_data8.gpmc_a12",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
//      {"lcd_data9.gpmc_a13",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
//      {"lcd_data10.gpmc_a14",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
//      {"lcd_data11.gpmc_a15",      OMAP_MUX_MODE1 | AM33XX_PIN_OUTPUT | AM33XX_
PULL_DISA},
//配置GPMC片选管脚，gpmc_csn1作为片选
//      {"gpmc_csn0.gpmc_csn0",         OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
{“gpmc_csn1.gpmc_csn1”,        OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
//      {“gpmc_csn2.gpmc_csn2”,        OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
//配置GPMC控制管脚
//      {"gpmc_wait0.gpmc_wait0", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
//      {"gpmc_wpn.gpmc_wpn",   OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
//      {"gpmc_advn_ale.gpmc_advn_ale",   OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
        {“gpmc_oen_ren.gpmc_oen_ren”,       OMAP_MUX_MODE0 | AM33XX_PULL_
DISA},//读
        {“gpmc_wen.gpmc_wen”,      OMAP_MUX_MODE0 | AM33XX_PULL_DISA},//写
//      {“gpmc_ben0_cle.gpmc_ben0_cle”,      OMAP_MUX_MODE0 | AM33XX_PULL_DISA},
        {“gpmc_clk.gpmc_clk”,      OMAP_MUX_MODE0 | AM33XX_PULL_DISA | AM33XX_
INPUT_EN},//时钟，AM33XX_INPUT_EN属性一定要加上！！！
        //配置控制DDS芯片AD9832的管脚为GPIO
        {“lcd_data9.gpio2_15”, OMAP_MUX_MODE7 | AM33XX_PIN_INPUT},//AD9832 SCLK
        {“lcd_data10.gpio2_16”, OMAP_MUX_MODE7 | AM33XX_PIN_INPUT},//AD9832 SDATA
        {“lcd_data11.gpio2_17”, OMAP_MUX_MODE7 | AM33XX_PIN_INPUT},//AD9832 FSYNC
        //配置同步数据采集与数据传输的WR_RDN信号管脚及RD_INT_PULSE信号管脚
        {“gpmc_csn0.gpio1_29”, OMAP_MUX_MODE7 | AM33XX_PIN_INPUT},//RD_INT_PULSE
信号是FPGA在采集完一行（512个点）数据后发送的一个正脉冲信号，该正脉冲信号会引起BB发
生外部IO中断，从而触发一次读操作！！！
{"gpmc_csn2.gpio1_31", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT},//WDRDN信号是控制采集数
据与读数据的信号，在高电平期间FPGA会采集数据并写入缓存，在低电平期间BB会读取缓存数
据，这时FPGA不会写缓存，从而有效防止了读写错乱！！！
        {NULL, 0},
};
```

（4）DDS 芯片 AD9832 驱动及读写 FPGA 的 GPMC 驱动的实现

（4.1）DDS 芯片 AD9832 驱动程序

AD9832 是 ADI 公司集成 DDS 芯片，可以很方便快速地设置输出正弦波的频率，这是本方案实现 DDS 信号发生器的方法之一，类比基于 FPGA 的 DDS 信号发生器，其优点是

使用方便，缺点是输出波形单一，应用灵活性差。

AD9832 的 datasheet：http://www.analog.com/static/imported-files/data_sheets/AD9832.pdf

驱动源文件为：ddsdev.c

设备驱动文件为：#define DDS_CHIP_DEV “/dev/ddsdev0”

主要功能函数：

图 3-21 是 AD9832 的读写时序图，dds_byte_shift 函数通过 GPIO 端口模拟了 AD9832 读写时序，其代码如下：

```
static void dds_byte_shift(unsigned int data_16)
{
    unsigned int temp;
    gpio_set_value(GPIO_FSYNC, 0);//FSYNC = 0;
    gpio_set_value(GPIO_SCLK, 1);//SCLK = 1;
    udelay(1);

    for (temp = 0x8000; temp != 0; ){
        if ((data_16 & temp) == 0)//MSB First
            gpio_set_value(GPIO_SDATA, 0);//SDATA = 0;
        else
            gpio_set_value(GPIO_SDATA, 1);//SDATA = 1;
        udelay(1);//Data setup time
        gpio_set_value(GPIO_SCLK, 0);//SCLK = 0;
        temp = temp >> 1;
        udelay(1);//Data hold time
        gpio_set_value(GPIO_SCLK, 1);//SCLK=1;
    }
    udelay(1);
    gpio_set_value(GPIO_FSYNC, 1);//FSYNC = 1;

    udelay(1000);
}
```

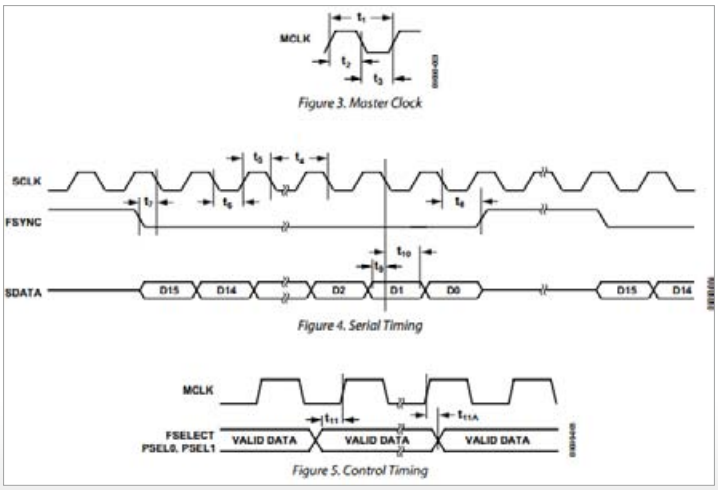


图 3-21：AD9832 读写时序

函数 dds_freq_process 及函数 dds_freq_adjust 用于设置相关控制方式及频率控制字，具体含义可以参考 AD9832 的 datasheet，并提供两篇 CNKI 上的论文作为参考：

直接数字频率合成芯片 AD9832 原理及其典型应用设计【作者】沈拓；董德存；

DDS 芯片 AD9832 的原理及应用【作者】江玉蓉；周有庆；吴桂清；

相关代码如下：

```
static void dds_freq_process(
        unsigned char freq0_lsbs_l,
        unsigned char freq0_lsbs_h,
        unsigned char freq0_msbs_l,
        unsigned char freq0_msbs_h,
        unsigned char freq1_lsbs_l,
        unsigned char freq1_lsbs_h,
        unsigned char freq1_msbs_l,
        unsigned char freq1_msbs_h
        )
{
        dds_byte_shift(0xd000);                 //11 010 00000000000
                                                     //SLEEP RESET CLR
        dds_byte_shift(0x3000 + freq0_lsbs_l);
        dds_byte_shift(0x2100 + freq0_lsbs_h);
        dds_byte_shift(0x3200 + freq0_msbs_l);
        dds_byte_shift(0x2300 + freq0_msbs_h);

        dds_byte_shift(0x3400 + freq1_lsbs_l);
        dds_byte_shift(0x2500 + freq1_lsbs_h);
        dds_byte_shift(0x3600 + freq1_msbs_l);
        dds_byte_shift(0x2700 + freq1_msbs_h);

        dds_byte_shift(0x9000);                 //10 01 000000000000
                                                     //SYNC SELSRC
        dds_byte_shift(0xc000);                 //11 000 00000000000
                                                     //SLEEP RESET CLR
                                                //dds_byte_
                                                shift(0x5555);
}
static void dds_freq_adjust(unsigned long freq_reg0, unsigned long freq_reg1)
{
        unsigned char ll0, lh0, hl0, hh0;
        unsigned char ll1, lh1, hl1, hh1;

        printk(KERN_ALERT "Adjust DDS
        frequency0: %08lx\n", freq_reg0);
        printk(KERN_ALERT "Adjust DDS
        frequency1: %08lx\n", freq_reg1);

        ll0 = (freq_reg0 >> 0) & 0x000000ff;
        lh0 = (freq_reg0 >> 8) & 0x000000ff;
```

```
        hl0 = (freq_reg0 >> 16) & 0x000000ff;
        hh0 = (freq_reg0 >> 24) & 0x000000ff;
        ll1 = (freq_reg1 >> 0) & 0x000000ff;
        lh1 = (freq_reg1 >> 8) & 0x000000ff;
        hl1 = (freq_reg1 >> 16) & 0x000000ff;
        hh1 = (freq_reg1 >> 24) & 0x000000ff;

        dds_freq_process(ll0, lh0, hl0, hh0, ll1, lh1, hl1, hh1);
}
```

（4.2）读写 FPGA 的 GPMC 驱动程序

关于读写 FPGA 的 GPMC 驱动最早之前有过测试，可以看下我之前发的一片帖子：http://bbs.eeworld.com.cn/thread-421181-1-1.html

驱动源文件为：fpgadev.c

设备驱动文件为：#define DDS_FPGA_DEV “/dev/fpgadev0”

这里面最关键的几个函数是：

(A) 设置 GPMC 相关寄存器的函数：

fpga_gpmc_init（对应的释放函数为 fpga_gpmc_exit），其关键代码是对 6 个 GPMC 配置基础的设置，如下标红部分：

```
static int fpga_gpmc_init(void)
{
      u32 l;
      int ret = -EINVAL;

      gpmc = kmalloc(sizeof(struct gpmc), GFP_KERNEL);


      if (!gpmc){
            ret = -ENOMEM;
            goto err_kmalloc;
      }
//    printk(KERN_ALERT "gpmc: %p, gpmc->mem_root: %p, gpmc->cs_mem: %p\n",
//          gpmc, &gpmc->mem_root, gpmc->cs_mem);
      printk(KERN_ALERT "gpmc: %p, gpmc_mem_root: %p, gpmc_cs_mem: %p\n",
            gpmc, &gpmc_mem_root, gpmc_cs_mem);

      gpmc->phys_base = AM33XX_GPMC_BASE;
      gpmc->memsize = AM33XX_GPMC_SIZE;

      if (request_mem_region(gpmc->phys_base,
            gpmc->memsize, "gpmc") == NULL) {

//    if (request_mem_region(AM33XX_GPMC_BASE,
//          AM33XX_GPMC_SIZE, "gpmc") == NULL) {
            ret = -ENOMEM;
```

```
            printk(KERN_ALERT "Failed to request memory region\n");
            goto err_mem;
      }

      gpmc->io_base = ioremap(gpmc->phys_base, gpmc->memsize);
//    gpmc->io_base = ioremap(AM33XX_GPMC_BASE, AM33XX_GPMC_SIZE);
      if (!gpmc->io_base) {
            ret = -ENOMEM;
            printk(KERN_ALERT "Failed to ioremap memory\n");
            goto err_remap;
      }

      printk(KERN_ALERT "GPMC physical base: %08lx, GPMC IO base: %p\n", gpmc-
      >phys_base, gpmc->io_base);
//    printk(KERN_ALERT "GPMC physical base: %08lx, GPMC IO base: %p\n", AM33XX_
      GPMC_BASE, gpmc->io_base);

      spin_lock_init(&gpmc->mem_lock);

      //Configure GPMC registers
      printk(KERN_ALERT "*****Configure GPMC registers*****\n");

      l = gpmc_read_reg(GPMC_REVISION);
      printk(KERN_ALERT "GPMC revision %d.%d\n", (l >> 4) & 0x0f, l & 0x0f);

      gpmc_write_reg(GPMC_IRQENABLE, 0x0000);//Interrupts are masked
      gpmc_write_reg(GPMC_TIMEOUT_CONTROL, 0x0000);//TimeOut feature is disabled

      printk(KERN_ALERT "SYNC_NOR_CONFIG1: %08x\n", SYNC_NOR_CONFIG1);
      printk(KERN_ALERT "SYNC_NOR_CONFIG2: %08x\n", SYNC_NOR_CONFIG2);
      printk(KERN_ALERT "SYNC_NOR_CONFIG3: %08x\n", SYNC_NOR_CONFIG3);
      printk(KERN_ALERT "SYNC_NOR_CONFIG4: %08x\n", SYNC_NOR_CONFIG4);
      printk(KERN_ALERT "SYNC_NOR_CONFIG5: %08x\n", SYNC_NOR_CONFIG5);
      printk(KERN_ALERT "SYNC_NOR_CONFIG6: %08x\n", SYNC_NOR_CONFIG6);

      gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG1, SYNC_NOR_CONFIG1);
      gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG2, SYNC_NOR_CONFIG2);
      gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG3, SYNC_NOR_CONFIG3);
      gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG4, SYNC_NOR_CONFIG4);
      gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG5, SYNC_NOR_CONFIG5);
      gpmc_cs_write_reg(GPMC_CS, GPMC_CS_CONFIG6, SYNC_NOR_CONFIG6);
      l = gpmc_cs_read_reg(GPMC_CS, GPMC_CS_CONFIG7);
      printk(KERN_ALERT  “Before CS, GPMC_CS_CONFIG7: %08x\n”, l);

      gpmc_mem_init();
return 0;

      err_remap:
      release_mem_region(gpmc->phys_base, gpmc->memsize);
//    release_mem_region(AM33XX_GPMC_BASE, AM33XX_GPMC_SIZE);
```

```
        err_
        mem:
                kfree(gpmc);
        err_kmalloc:
                return ret;
}
```

这 6 个寄存器的值如何设置需要非常详细地阅读《AM335x ARM® Cortex™-A8 Microprocessors(MPUs) TechnicalReferenceManual 》文档的 7.1 GPMC 部分，驱动中给出了这 6 个寄存器设置值的详细含义说明，如下代码所示：

```
/* GPMC register offsets */
#define GPMC_REVISION    0x00
#define GPMC_SYSCONFIG   0x10
#define GPMC_SYSSTATUS   0x14
#define GPMC_IRQSTATUS   0x18
#define GPMC_IRQENABLE   0x1c
#define GPMC_TIMEOUT_
CONTROL                  0x40
#define GPMC_ERR_ADDRESS 0x44
#define GPMC_ERR_TYPE    0x48
#define GPMC_CONFIG      0x50
#define GPMC_STATUS      0x54
#define GPMC_PREFETCH_   0x1e0
CONFIG1
#define GPMC_PREFETCH_   0x1e4
CONFIG2
#define GPMC_PREFETCH_
CONTROL                  0x1ec
#define GPMC_PREFETCH_
STATUS                   0x1f0
#define GPMC_ECC_CONFIG  0x1f4
#define GPMC_ECC_CONTROL
0x1f8
#define GPMC_ECC_SIZE_CONFIG                0x1fc
#define GPMC_ECC1_RESULT                    0x200
#define GPMC_ECC_BCH_RESULT_0               0x240

#define GPMC_CS0_OFFSET  0x60
#define GPMC_CS_SIZE     0x30

#define GPMC_MEM_START   0x00000000
#define GPMC_MEM_END     0x3FFFFFFF
#define BOOT_ROM_SPACE   0x100000      /* 1MB */

#define GPMC_CHUNK_SHIFT 24            /* 16 MB */
#define GPMC_SECTION_
SHIFT                    28            /* 128 MB */
```

```
#define CS_NUM_SHIFT     24
#define ENABLE_PREFETCH  (0x1 << 7)
#define DMA_MPU_MODE     2

#define AM33XX_GPMC_BASE             0x50000000
#define AM33XX_GPMC_SIZE SZ_16M
#define GPMC_CS                          1
#define FPGA_SIZE SZ_512

#define GPMCFCLKDIVIDER 0<<0//GPMC_CLK frequency = GPMC_FCLK frequency
#define TIMEPARAGRANULARITY 0<<4//脑1 latencies
#define MUXADDDATA 0<<8//Non-multiplexed attached device
#define DEVICETYPE 0<<10//NOR Flash like, asynchronous and synchronous devices
#define DEVICESIZE 1<<12//16 bit
#define WAITPINSELECT 0<<16//WAIT input pin is WAIT0
#define WAITMONITORINGTIME 0<<18//WAIT pin is monitored with valid data
#define WAITWRITEMONITORING 0<<21//WAIT pin is not monitored for write accesses
#define WAITREADMONITORING 0<<22//WAIT pin is not monitored for read accesses
#define ATTACHEDDEVICEPAGELENGTH 2<<23//Specifies the attached device page (burst)
length (1 Word = Interface size), 16Words
#define CLKACTIVATIONTIME 0<<25//First rising edge of GPMC_CLK at start access
time
#define WRITETYPE 1<<27//Write Synchronous
#define WRITEMULTIPLE 1<<28//Selects the write single or multiple access, Single
access
#define READTYPE 1<<29//Read Synchronous
#define READMULTIPLE 1<<30//Selects the read single or multiple access, Single
access
#define WRAPBURST 1<<31//Synchronous wrapping burst not supported

#define SYNC_NOR_CONFIG1 WRAPBURST | \
    READMULTIPLE | READTYPE | WRITEMULTIPLE | WRITETYPE | \
    CLKACTIVATIONTIME | \
    ATTACHEDDEVICEPAGELENGTH | \
WAITREADMONITORING | WAITWRITEMONITORING | WAITMONITORINGTIME | WAITPINSELECT | \
    DEVICESIZE | DEVICETYPE | MUXADDDATA | \
    TIMEPARAGRANULARITY | GPMCFCLKDIVIDER

#define CSONTIME 1<<0//CS# assertion time from start cycle time, 1 GPMC_FCLK
cycle
#define CSEXTRADELAY 0<<7//CS i Timing control signal is not delayed
#define CSRDOFFTIME 15<<8//CS# de-assertion time from start cycle time for read
accesses, 15 GPMC_FCLK cycles
#define CSWROFFTIME 15<<16//CS# de-assertion time from start cycle time for write
accesses, 15 GPMC_FCLK cycles

#define SYNC_NOR_CONFIG2 CSWROFFTIME | CSRDOFFTIME | CSEXTRADELAY | CSONTIME

#define ADVONTIME 1<<0//ADV# assertion time from start cycle time, 1 GPMC_FCLK
cycle
```

```
#define ADVAADMUXONTIME 0<<4//ADV# assertion for first address phase when using
the AAD-Multiplexed protocol
#define ADVEXTRADELAY 0<<7//ADV Timing control signal is not delayed
#define ADVRDOFFTIME 15<<8//ADV# de-assertion time from start cycle time for read
accesses, 15 GPMC_FCLK cycles
#define ADVWROFFTIME 15<<16//ADV# de-assertion time from start cycle time for
write accesses, 15 GPMC_FCLK cycles
#define ADVAADMUXRDOFFTIME 0<<24//ADV# de-assertion for first address phase when
using the AAD-Mux protocol for read accesses
#define ADVAADMUXWROFFTIME 0<<28//ADV# de-assertion for first address phase when
using the AAD-Mux protocol for write accesses

#define SYNC_NOR_CONFIG3 ADVAADMUXWROFFTIME | ADVAADMUXRDOFFTIME | \
    ADVWROFFTIME | ADVRDOFFTIME | ADVEXTRADELAY | ADVAADMUXONTIME | ADVONTIME

#define OEONTIME 1<<0//OE# assertion time from start cycle time
#define OEAADMUXONTIME 0<<4//OE# assertion time for the first address phase in an
AAD-Multiplexed access
#define OEEXTRADELAY 0<<7//OE Timing control signal is not delayed
#define OEOFFTIME 15<<8//OE# de-assertion time from start cycle time
#define OEAADMUXOFFTIME 0<<13//OE# de-assertion time for the first address phase
in an AAD-Multiplexed access
#define WEONTIME 1<<16//WE# assertion time from start cycle time
#define WEEXTRADELAY 0<<23//WE Timing control signal is not delayed
#define WEOFFTIME 15<<24//WE# de-assertion time from start cycle time

#define SYNC_NOR_CONFIG4 WEOFFTIME | WEEXTRADELAY | WEONTIME | \
    OEAADMUXOFFTIME | OEOFFTIME | OEEXTRADELAY | OEAADMUXONTIME |OEONTIME

#define RDCYCLETIME 15<<0//Total read cycle time
#define WRCYCLETIME 15<<8//Total write cycle time
#define RDACCESSTIME 12<<16//Delay between start cycle time and first data valid
#define PAGEBURSTACCESSTIME 3<<24//Delay between successive words in a multiple
access

#define SYNC_NOR_CONFIG5 PAGEBURSTACCESSTIME | RDACCESSTIME | WRCYCLETIME |
RDCYCLETIME

#define BUSTURNAROUND 3<<0//Bus turn around latency between two successive
accesses to the same chip-select (read to write) or to a different chip-select (read
to read and read to write)
#define CYCLE2CYCLEDIFFCSEN 1<<6//Add Cycle2CycleDelay between two successive
accesses to a different chip-select (any access type)
#define CYCLE2CYCLESAMECSEN 1<<7//Add Cycle2CycleDelay between two successive
accesses to the same chip-select (any access type)
#define CYCLE2CYCLEDELAY 3<<8//Chip select high pulse delay between two successive
accesses
#define WRDATAONADMUXBUS 3<<16//Specifies on which GPMC.FCLK rising edge the first
data of the synchronous burst write is driven in the add/data multiplexed bus
```

```
#define WRACCESSTIME 12<<24//Delay from StartAccessTime to the GPMC.FCLK rising
edge corresponding the the GPMC.CLK rising edge used by the attached memory for
the first data capture

#define SYNC_NOR_CONFIG6 WRACCESSTIME | WRDATAONADMUXBUS | \
    CYCLE2CYCLEDELAY | CYCLE2CYCLESAMECSEN | CYCLE2CYCLEDIFFCSEN | \
     BUSTURNAROUND
```

这些设置确实很繁琐，所以真的需要仔细阅读下文档。

(B)！！！用于申请 GPMC 片选的函数：

gpmc_cs_request（对应的释放函数为 gpmc_cs_free）得到申请到的 GPMC 物理地址空间 fpga_phy_base，fpga_cs_request（对应的释放函数为 fpga_cs_free）根据上述物理地址空间 fpga_phy_base 最终获得操作 GPMC 的虚拟地址空间 fpga_vir_base。

(C)GPMC 读写函数：

写函数，在设置 FPGA DDS 波形数据及频率控制字的时候应用层会调用该函数，代码如下：

```
static ssize_t fpgadev_write(struct file *p_file, const char __user *u_buf, size_t
size, loff_t *p_pos)
{
    size_t len = FPGADEV_SIZE;
    int i;
    int tmp;
    struct fpgadev *p_fpgadev = p_file->private_data;

    if (size == 0)
            return 0;
    if (len > size)
            len = size;

    memset(p_fpgadev->m_buf, 0, FPGADEV_SIZE);
    if (copy_from_user(p_fpgadev->m_buf, u_buf, len))
            return -EFAULT;
    for (i=0; i<len; i=i+2)
    {
    tmp = (p_fpgadev->m_buf[i]) | ((p_fpgadev->m_buf[i+1])<<8);
            writew(tmp, fpga_vir_base+i);
            printk(KERN_ALERT  “*****Write*****%p, %04x\n”, fpga_vir_base+i, tmp);
    }
    return 0;
}
```

读函数，在数据采集读取采集的数据时会调用该函数，代码如下，特别注意标红加粗的部分：

```
    static ssize_t fpgadev_read(struct file *p_file, char __user *u_buf, size_t
size, loff_t *p_pos)
   {
      size_t len = FPGADEV_SIZE;
      int i;
      int tmp;
      struct fpgadev *p_fpgadev = p_file->private_data;

      if (size == 0)
              return 0;
      if (len > size)
              len = size;
      if (*p_pos > 0)
              return 0;
      memset(p_fpgadev->m_buf, 0, FPGADEV_SIZE);
      down(&sync_sema);//wait until read enabled
      for (i=0; i<len; i=i+2)
      {
      tmp = readw(fpga_vir_base + i);
              p_fpgadev->m_buf[i] = tmp&0xff;
              p_fpgadev->m_buf[i+1] = (tmp>>8)&0xff;
              //printk(KERN_ALERT  "*****Read*****%p, %04x\n", fpga_vir_base+i,

              tmp);
}
      gpio_set_value(GPIO_RSTN, 1);//enable write
      if (copy_to_user(u_buf, p_fpgadev->m_buf, len))
              return -EFAULT;
      return 0;
}
```

sync_sema 是一个初值为 0 的信号量，如果没有数据可读，读函数会进入阻塞状态，在 FPGA 采集完一行数据后会产生一个 RD_INT_PULSE 正脉冲信号，该正脉冲信号会引起 BB 发生外部 IO 中断，中断处理函数会 up sync_sema 信号量，从而触发一次读操作，外部中断处理函数会同时清零 WDRDN(对应代码中的 GPIO_RSTN)，使能读缓存失能写缓存，在读函数完成读操作后会置一 WDRDN，使能写缓存失能读缓存，WDRDN 信号是控制读写数据缓的信号，在高电平期间 FPGA 将采集的数据写入缓存，在低电平期间 BB 从缓存中读取数据，这时 FPGA 不会写缓存，从而保证了读写缓存不会发生错乱。外部中断处理函数代码如下：

```
static irqreturn_t irq_handler(int irq, void *dev_id)
{
  disable_irq_nosync(irq_no);
  up(&sync_sema);
  gpio_set_value(GPIO_RSTN, 0);//enable read
  enable_irq(irq_no);
  return IRQ_HANDLED;
}
```

（5）BB 应用层程序流程如图 3-22 所示：

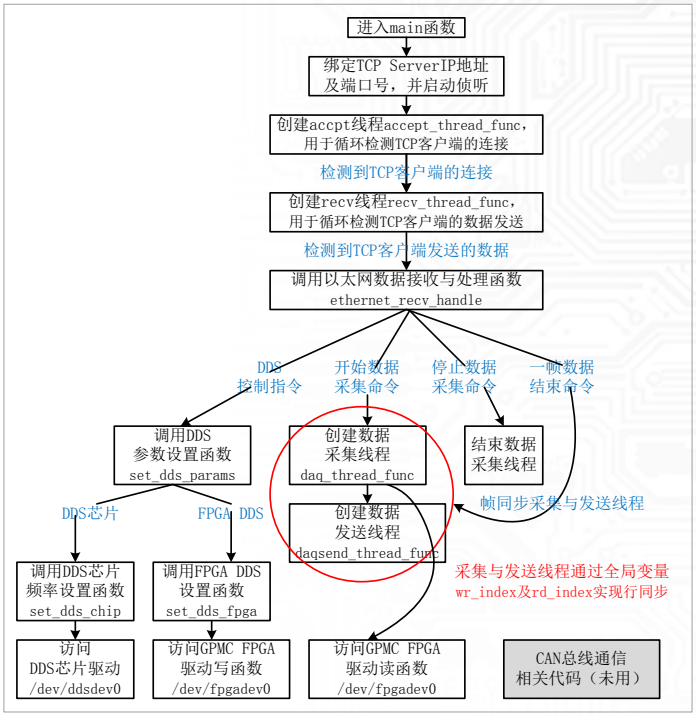


图 3-22：BB 应用程序简易流程图

（6）附件 2 为 BB 端的全部源代码：

附件 2：DDS_DAQ_BB.rar

（6.1）其目录树如下所示：

root@gao:~/bone-fs/home/root/DDS_DAQ_BB# tree.

```
├─── APP（应用程序）
│    ├─── canbus.c
│    ├─── canbus.o
│    ├─── canconfig.c
│    ├─── can_config.h
│    ├─── canconfig.o
│    ├─── candump.c
│    ├─── candump.o
│    ├─── cansend.c
│    ├─── cansend.o
│    ├─── dds.c
│    ├─── dds.o
│    ├─── ethernet.c
│    ├─── ethernet.o
│    ├─── libsocketcan.a
│    ├─── lscm_debug.h
│    ├─── lscm_protocol.h
│    ├─── main
│    ├─── main.c
```

```
│   ├── main.o
│   └── Makefile
├── board-am335xevm.c（板文件）
├── DDS_DRV（DDS芯片驱动程序）
│   ├── ddsdev.c
│   ├── ddsdev.ko
│   ├── ddsdev.mod.c
│   ├── ddsdev.mod.o
│   ├── ddsdev.o
│   ├── Makefile
│   ├── modules.order
│   ├── Module.symvers
│   ├── test_ddsdev
│   └── test_ddsdev.c
└── GPMC_DRV（GPMC FPGA驱动程序）
    ├── fpgadev.c
    ├── fpgadev.ko
    ├── fpgadev.mod.c
    ├── fpgadev.mod.o
    ├── fpgadev.o
    ├── Makefile
    ├── modules.order
    ├── Module.symvers
    ├── test_fpgadev
    └── test_fpgadev.c

3 directories, 41 files
```

（6.2）BB 端源代码的编译过程如下：

（A）应用程序编译：

```
root@gao:~/bone-fs/home/root/DDS_DAQ_BB# ls
APP  board-am335xevm.c  DDS_DRV  GPMC_DRV
root@gao:~/bone-fs/home/root/DDS_DAQ_BB# cd APP
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/APP# ls
canbus.c      can_config.h  cansend.c  ethernet.c      lscm_debug.h     main.c
canconfig.c   candump.c     dds.c      libsocketcan.a  lscm_protocol.h  Makefile
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/APP# make
arm-arago-linux-gnueabi-gcc -c main.c
arm-arago-linux-gnueabi-gcc -c ethernet.c
arm-arago-linux-gnueabi-gcc -c canbus.c
arm-arago-linux-gnueabi-gcc -c dds.c
arm-arago-linux-gnueabi-gcc -c canconfig.c
arm-arago-linux-gnueabi-gcc -c cansend.c
arm-arago-linux-gnueabi-gcc -c candump.c
arm-arago-linux-gnueabi-gcc -o main main.o \
          ethernet.o canbus.o \
```

```
        dds.o \
        canconfig.o cansend.o candump.o libsocketcan.a \
        -lpthread \
        -lm
```

（B）GPMC FPGA 设备驱动程序编译：

```
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/APP# cd ../GPMC_DRV/
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV# ls
fpgadev.c  Makefile  test_fpgadev.c
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV# make
make -C /root/bone-linux M=/root/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV modules
ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-
make[1]: 正在进入目录 `/root/ti-sdk-am335x-evm/board-support/linux-3.2.0-
psp04.06.00.08.sdk'
  CC [M]  /root/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV/fpgadev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV/fpgadev.mod.o
  LD [M]  /root/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV/fpgadev.ko
make[1]:正在离开目录 `/root/ti-sdk-am335x-evm/board-support/linux-3.2.0-
psp04.06.00.08.sdk'
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV# make app
arm-arago-linux-gnueabi-gcc test_fpgadev.c -o test_fpgadev
```

（C）DDS 芯片驱动程序编译：

```
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/GPMC_DRV# cd ../DDS_DRV/
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/DDS_DRV# ls
ddsdev.c  Makefile  test_ddsdev.c
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/DDS_DRV# make
make -C /root/bone-linux M=/root/bone-fs/home/root/DDS_DAQ_BB/DDS_DRV modules
ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-
make[1]: 正在进入目录 `/root/ti-sdk-am335x-evm/board-support/linux-3.2.0-
psp04.06.00.08.sdk'
  CC [M]  /root/bone-fs/home/root/DDS_DAQ_BB/DDS_DRV/ddsdev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/bone-fs/home/root/DDS_DAQ_BB/DDS_DRV/ddsdev.mod.o
  LD [M]  /root/bone-fs/home/root/DDS_DAQ_BB/DDS_DRV/ddsdev.ko
make[1]:正在离开目录 `/root/ti-sdk-am335x-evm/board-support/linux-3.2.0-
psp04.06.00.08.sdk'
root@gao:~/bone-fs/home/root/DDS_DAQ_BB/DDS_DRV# make app
arm-arago-linux-gnueabi-gcc test_ddsdev.c -o test_ddsdev
```

（6.3）BB 端程序如下执行：

（A）挂载开发主机文件系统，开发主机的 IP 地址为 192.168.1.106，BB 的 IP 地址为 192.168.1.2：

```
root@am335x-evm:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr D4:94:A1:97:90:BA
          inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:98 errors:0 dropped:40 overruns:0 frame:0
          TX packets:51 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23520 (22.9 KiB)  TX bytes:4596 (4.4 KiB)

root@am335x-evm:~# ls
fs_mount.sh  tftptest
root@am335x-evm:~# ./fs_mount.sh
mount 192.168.1.106/root/bone-fs /mnt/nfs -o nolock,proto=tcp
Can't set permissions on mtab: Operation not permitted
root@am335x-evm:~# cd /mnt/nfs/home/root/DDS_DAQ_BB/
```

（B）加载驱动程序：

```
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB# ls
APP                 DDS_DRV             GPMC_DRV            board-am335xevm.c
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB# cd DDS_DRV/
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/DDS_DRV# ls
Makefile         ddsdev.ko        ddsdev.o         test_ddsdev.c
Module.symvers   ddsdev.mod.c     modules.order
ddsdev.c         ddsdev.mod.o     test_ddsdev
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/DDS_DRV# insmod ddsdev.ko
[  231.552669] *****Init*****
[  231.555528] GPIO_SCLK is valid return: 1
[  231.559650] GPIO_SDATA is valid return: 1
[  231.563911] GPIO_FSYNC is valid return: 1
[  231.568135] Request GPIO_SCLK return: 0
[  231.572184] Request GPIO_SDATA return: 0
[  231.576308] Request GPIO_FSYNC return: 0
[  231.580434] GPIO_SCLK set output return: 0
[  231.584881] GPIO_SDATA set output return: 0
[  231.589283] GPIO_FSYNC set output return: 0

root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/DDS_DRV# cd ../GPMC_DRV/
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/GPMC_DRV# ls
Makefile         fpgadev.ko       fpgadev.o        test_fpgadev.c
Module.symvers   fpgadev.mod.c    modules.order
fpgadev.c        fpgadev.mod.o    test_fpgadev
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/GPMC_DRV# insmod fpgadev.ko
[  268.803402] *****Init*****
[  268.809626] GPIO_EINT is valid return: 1
[  268.813844] GPIO_RSTN is valid return: 1
[  268.817976] Request GPIO_EINT return: 0
[  268.822024] Request GPIO_RSTN return: 0
[  268.826059] GPIO_RSTN set input return: 0
[  268.830276] GPIO_RSTN set output return: 0
```

```
[  268.834599] EINT irq: 221
[  268.837359] Set EINT rising edge return: 0
[  268.841805] Request irq return: 0(0)
[  268.845572] gpmc: cee42680, gpmc_mem_root: bf016624, gpmc_cs_mem: bf016544
[  268.853412] GPMC physical base: 50000000, GPMC IO base: d1000000
[  268.859727] *****Configure GPMC registers*****
[  268.864420] GPMC revision 6.0
[  268.867537] SYNC_NOR_CONFIG1: f9001000
[  268.871476] SYNC_NOR_CONFIG2: 000f0f01
[  268.875428] SYNC_NOR_CONFIG3: 000f0f01
[  268.879367] SYNC_NOR_CONFIG4: 0f010f01
[  268.883339] SYNC_NOR_CONFIG5: 030c0f0f
[  268.887279] SYNC_NOR_CONFIG6: 0c0303c3
[  268.891220] Before CS, GPMC_CS_CONFIG7: 00000f00
[  268.896091] *****Request GPMC CS*****
[  268.899949] Got GPMC CS1, FPGA physical base: 01000000
[  268.905364] After CS, GPMC_CS_CONFIG7: 00000f41
[  268.910149] FPGA virtual base: d087e000
```

（C）运行应用程序 main 可执行文件：

```
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/GPMC_DRV# cd ../APP/
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/APP# ls
Makefile          canconfig.o       dds.c             lscm_debug.h
can_config.h      candump.c         dds.o             lscm_protocol.h
canbus.c          candump.o         ethernet.c        main
canbus.o          cansend.c         ethernet.o        main.c
canconfig.c       cansend.o         libsocketcan.a    main.o
root@am335x-evm:/mnt/nfs/home/root/DDS_DAQ_BB/APP# ./main
this is main function.
socket ok.
bind ok.
listen.
server_bind_listen ok, server_ip: 192.168.1.2, server_port: 5000.
pthread_create ok, thread id: 1086289008.
accept thread is waiting a client to connect.....
```

3.3.4 FPGA 端软件实现

这一部分主要说明一下在 FPGA 端软件的设计中采用的小技巧并测试了 GPMC 对 FPGA RAM 的读取速率，展示一些实验的图片数据。

（1）开发平台及语言为：

QuartusII，Verilog 与原理图输入相结合

（2）主要包括三个模块：

硬线连接 DDS 芯片与 BB GPIO 端口模块；

FPGA DDS 模块；

数据采集与传输模块。

（3）FPGA 管脚分配如下：

```
#连接DDS芯片的管脚
set_location_assignment PIN_121 -to DDSCHIP_FSEL
set_location_assignment PIN_122 -to DDSCHIP_FSYNC
set_location_assignment PIN_129 -to DDSCHIP_MCLK
set_location_assignment PIN_119 -to DDSCHIP_PSEL0
set_location_assignment PIN_120 -to DDSCHIP_PSEL1
set_location_assignment PIN_126 -to DDSCHIP_SCLK
set_location_assignment PIN_125 -to DDSCHIP_SDATA
#连接BB GPIO，用于控制DDS芯片的管脚
set_location_assignment PIN_96 -to IN_FSYNC
set_location_assignment PIN_100 -to IN_SCLK
set_location_assignment PIN_97 -to IN_SDATA
//FPGA时钟与复位管脚
set_location_assignment PIN_17 -to FPGA_CLK0
set_location_assignment PIN_90 -to FPGA_RST
//连接BB GPMC接口的管脚
set_location_assignment PIN_72 -to GPMC_CLK
set_location_assignment PIN_118 -to GPMC_A[1]
set_location_assignment PIN_113 -to GPMC_A[2]
set_location_assignment PIN_114 -to GPMC_A[3]
set_location_assignment PIN_104 -to GPMC_A[4]
set_location_assignment PIN_112 -to GPMC_A[5]
set_location_assignment PIN_101 -to GPMC_A[6]
set_location_assignment PIN_103 -to GPMC_A[7]
set_location_assignment PIN_87 -to GPMC_A[8]
set_location_assignment PIN_93 -to GPMC_A[9]
set_location_assignment PIN_81 -to GPMC_AD[0]
set_location_assignment PIN_80 -to GPMC_AD[1]
set_location_assignment PIN_57 -to GPMC_AD[2]
```

```
set_location_assignment PIN_58 -to GPMC_AD[3]
set_location_assignment PIN_79 -to GPMC_AD[4]
set_location_assignment PIN_76 -to GPMC_AD[5]
set_location_assignment PIN_53 -to GPMC_AD[6]
set_location_assignment PIN_55 -to GPMC_AD[7]
set_location_assignment PIN_73 -to GPMC_AD[8]
set_location_assignment PIN_65 -to GPMC_AD[9]
set_location_assignment PIN_67 -to GPMC_AD[10]
set_location_assignment PIN_71 -to GPMC_AD[11]
set_location_assignment PIN_64 -to GPMC_AD[12]
set_location_assignment PIN_63 -to GPMC_AD[13]
set_location_assignment PIN_70 -to GPMC_AD[14]
set_location_assignment PIN_69 -to GPMC_AD[15]
set_location_assignment PIN_75 -to GPMC_CSN
set_location_assignment PIN_59 -to GPMC_OEN_REN
set_location_assignment PIN_60 -to GPMC_WEN
#连接DAC的管脚
set_location_assignment PIN_144 -to DAC_CLK
set_location_assignment PIN_143 -to DAC_BIT[10]
set_location_assignment PIN_142 -to DAC_BIT[9]
set_location_assignment PIN_141 -to DAC_BIT[8]
set_location_assignment PIN_139 -to DAC_BIT[7]
set_location_assignment PIN_137 -to DAC_BIT[6]
set_location_assignment PIN_136 -to DAC_BIT[5]
set_location_assignment PIN_135 -to DAC_BIT[4]
set_location_assignment PIN_134 -to DAC_BIT[3]
set_location_assignment PIN_133 -to DAC_BIT[2]
set_location_assignment PIN_132 -to DAC_BIT[1]
#连接ADC的管脚
set_location_assignment PIN_7 -to ADC_CLK
set_location_assignment PIN_8 -to ADC_BIT[1]
set_location_assignment PIN_9 -to ADC_BIT[2]
set_location_assignment PIN_24 -to ADC_BIT[3]
set_location_assignment PIN_25 -to ADC_BIT[4]
set_location_assignment PIN_26 -to ADC_BIT[5]
set_location_assignment PIN_27 -to ADC_BIT[6]
set_location_assignment PIN_28 -to ADC_BIT[7]
set_location_assignment PIN_30 -to ADC_BIT[8]
set_location_assignment PIN_31 -to ADC_BIT[9]
set_location_assignment PIN_32 -to ADC_BIT[10]
#连接BB GPIO端口，辅助数据采集与传输的管脚
set_location_assignment PIN_86 -to RD_INT_PULSE
set_location_assignment PIN_74 -to WR_RDN
```

（4）硬线连接 DDS 芯片与 BB GPIO 端口模块

这部分 FPGA 原理图如图 3-23 所示：

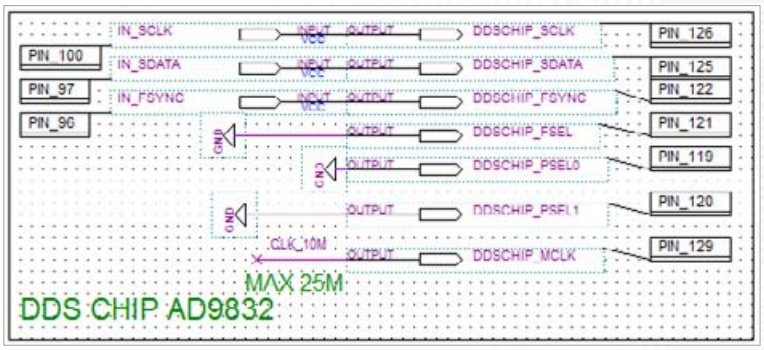


图 3-23：硬线连接 DDS 芯片与 BB GPIO 端口模块 FPGA 原理图

图 3-24 展示了 DDS 芯片输出的正弦波形，从图中可以看出，频率达到 KHz 后波形会有衰减，频率越高，衰减越厉害，这涉及放大器带宽及信号完整性问题，模拟部分有待深入修改。

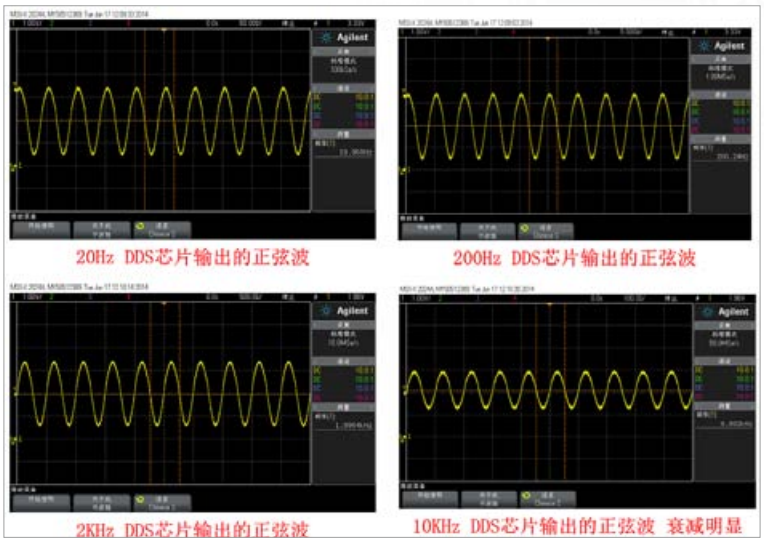


图 3-24：DDS 芯片输出的正弦波

（5）FPGA DDS 模块

这个模块的设计采用了一个小技巧，只用到了 GPMC 的 A[1]、A[2]、A[3] 三根地址线（特别注意 16 位模式下的 GPMC A[0] 地址线未用，具体可以参考 GPMC 的相关文档），这三根地址线配合 GPMC 写控制线 WEN，利用 FPGA 搭建的若干三八译码器及寄存器产生中间控制信号 CTL[15..0]，写频率控制字的控制信号 WFREQH、WFREQM、WFREQL，写相位的控制信号 WPHASE，写波形数据的控制信号 WDATA，写直流幅值的控制信号 WDC，如图 3-25 所示：

这一部分不再对 DDS 的基本原理进行阐述，关于基本原理可以参考之前我发的一篇帖子：http://bbs.eeworld.com.cn/thread-439651-1-1.html，理解这一部分一定需要与 BB 端应用程序 dds.c 文件中的相关函数配合起来一起看，否则很难理解，总之，一句话代过：只可意会，不可言传。

图 3-26 展示了 FPGA DDS 输出的各种波形，含直流、正弦波、斜波、以及方波，仍然存在波形失真的问题。

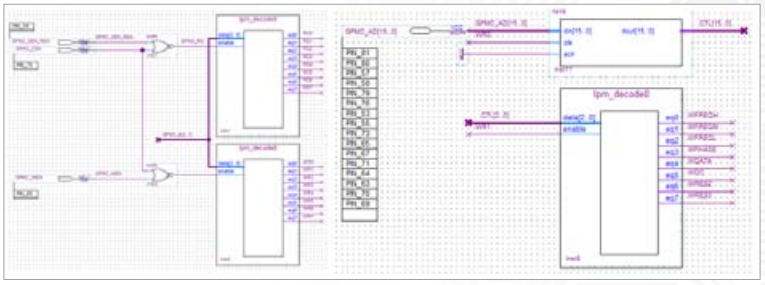


图 3-25：FPGA DDS 模块控制信号生成部分 FPGA 原理图

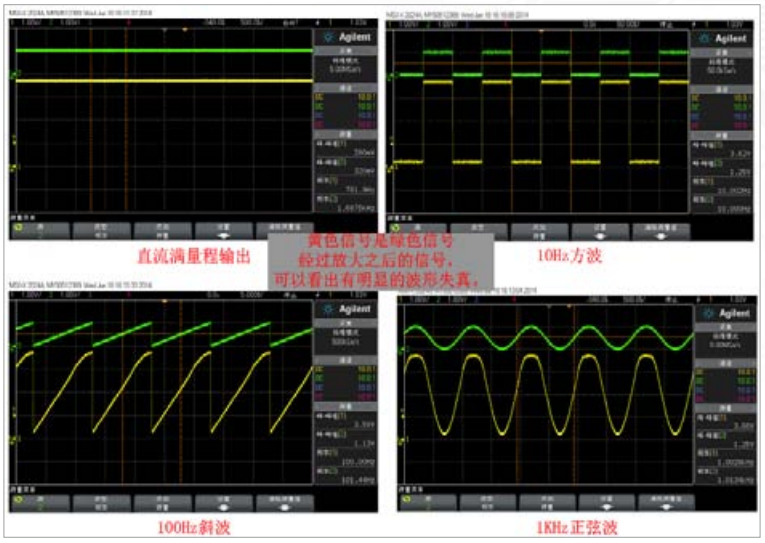


图 3-26：FPGA DDS 输出的各种波形

（6）数据采集与传输模块

这个模块采用 FPGA 内的双口 RAM 作为数据缓存，缓存的容量为一行数据的容量，采样频率为 25KHz，FPGA 原理图下图 3-27 所示：

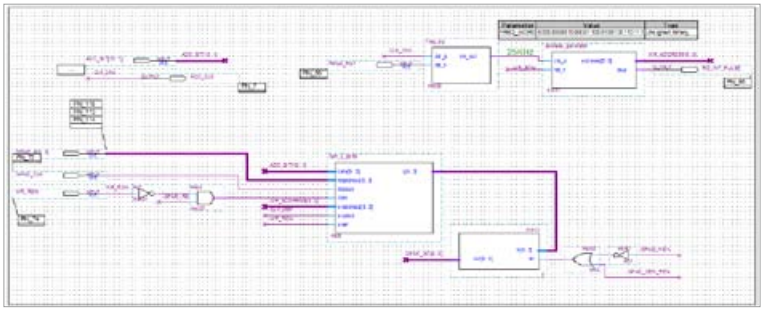


图 3-27：数据采集与传输模块 FPGA 原理图

这个部分测试了 GPMC 对 FPGA RAM 的读取速率，BB 通过 GPMC 接口连续读 FPGA RAM，用示波器分别探测 A[1] 地址线以及 WR_RDN 管脚信号，如图 13 所示。

图 3-28 的最上面一幅图是示波器探测的 A[1] 地址线，其高或低电平持续的时间约为 +441.000ns，一个高电平或低电平期间读取 16bits 的数据，数据读取速率约为 36.3Mbps。

中间一幅图测量了读取 512 个点，即一行数据的时间为 +225.600000us，根据上面的数据读取速率计算一行数据的读取时间理论值为 +441.000ns*512=+225.792000us，理论

值与测量值基本吻合。

下面一幅图是示波器探测的WR_RDN管脚信号，WR_RDN高电平期间读一行数据，WR_RDN低电平期间写一行数据。读一行数据的时间显示为+242.000000us，与上述的值基本吻合，另外测量时数据采样频率设置的为500KHz，512/500KHz=+1.024000000ms，与测量值+1.030000000ms也是相当吻合的。

这个部分的研究基本回答了wy52119514在论坛这个帖子中的提问，http://bbs.eeworld.com.cn/thread-421181-1-1.html

大哥你测过GPMC的速度可以达到多少呢?，以你这个方式

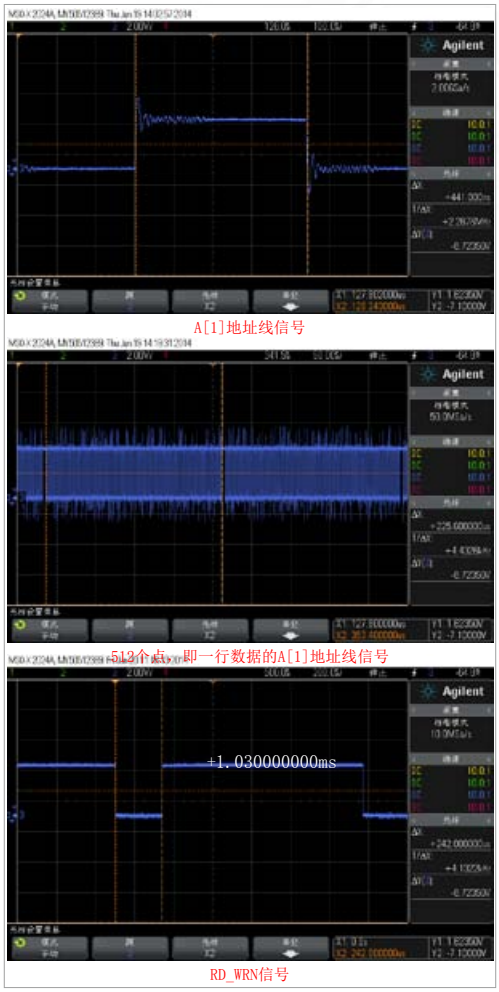


图 3-28：GPMC 读 FPGA RAM 速率测试图

FPGA端软件的全部源代码：DDS_FPGA.rar

点击查看详情：>> 基于BB及FPGA的高速数据采集及DDS信号发生器设计与实现

3.4 基于 BeagleBone 的 Android4.0.3 ICS 的移植

起初打算使用 QT 来做界面，但是随着 NOKIA 的衰落，QT 的前景也变得暗淡，关于这方面的资料也不多。

Android 不仅界面华丽，而且它的功能库包含了众多开源产品，包括 WebKit，OpenCore，SQlite 等。

Opencore 除了能解析一般的本地多媒体文件外，还提供了对网络 i 流媒体的解码功能。网络流媒体主要有 HTTP,RTP 和 RTSP 等。

而小车的视频传输正是基于 RTSP 协议。基于此，在 BeagleBone 上改用 Android。其实 Android 也是基于 Linux 的，Android 内核的编译和编译一般的 linux 并没有区别。但是 Linux 内核仅是 Android 架构中的一层。整个 Android 框架如下：

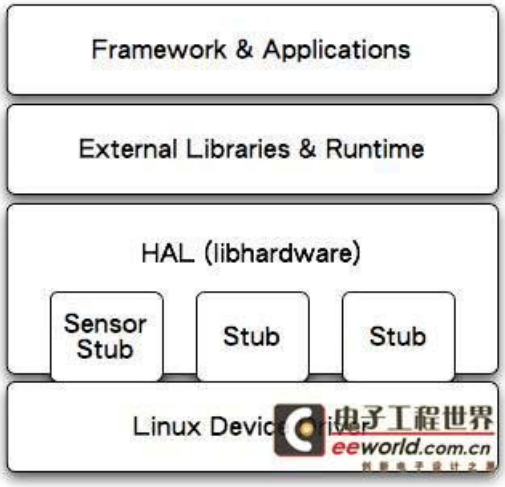


图 3-29

整个开发过程包括每个层之间的框架接口编写，涉及 C，C++，JAVA 编程。

以下是在 BeagleBone 上移植 Android 的一点心得分享：

第一步：下载 android 源码，TI 官网有提供。

地址如下：

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/TI_Android_ICS_4_0_3_DevKit_3_0_1/index_FDS.html

下载：TIAndroid ICS 4.0.3 DevKitV3.0.1 AM335x EMV-SK Sources

下载环境：UBUNTU 10.04（我用 WIN7 下载 3 两次卡死在了 99.99% ）PS：为了后面编译，虚拟机的空间要足够大（25G 以上）

下载完成解压前，得先安装需要的软件。

sudo gedit /etc/apt/sources.list

在最后一行加入：

deb http://us.archive.ubuntu.com/ubuntu/hardymultiverse 保存退出

接下来：$ sudo apt-get update

$ sudo apt-get install git-core gnupg sun-java6-jdk flexbison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curllibncurses5-dev zlib1g-dev minicom tftpd uboot-mkimage expect

安装软件时，安步骤走即可。

之后就可以安装下载好的 android 源码啦。

步骤：

$ mkdir $HOME/rowboat-android $ cd $HOME/rowboat-android $ chmod a+x TI-Android-ICS-4.0.3_AM335x_3.0.1.bin $ ./TI-Android-ICS-4.0.3_AM335x_3.0.1.bin

大概 15 分钟后，就可以安装完成。

之后得到包括 U-boot 源码，Linux 源码以及 android 文件系统源码。

编译源码前，得先安装好交叉编译工具。

工具的位置：$HOME/rowboat-android/TI-Android-ICS-4.0.3_AM335x_3.0.1/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin

可以在 /etc/environment 之内加入也可以用如下方式：

$ export PATH=$HOME/rowboat-android/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin:$PATH

编译 Bootloader: $ cd <android source path>/u-boot

Execute following commands:

$ make CROSS_COMPILE=arm-eabi- distclean

$ make CROSS_COMPILE=arm-eabi- am335x_evm_config

$ make CROSS_COMPILE=arm-eabi-

编译内核：

Change directory to kernel

$ cd <android source path>/kernel

Execute following commands

$ make ARCH=arm CROSS_COMPILE=arm-eabi- distclean

$ make ARCH=arm CROSS_COMPILE=arm-eabi- am335x_evm_android_

defconfig

$ make ARCH=arm CROSS_COMPILE=arm-eabi- uImage

编译 Android 文件系统稍微复杂一点：

1. 配置好 java 环境

安装插件：

sudo apt-getinstall sun-java6-jdk

可以通过下面的方法设置机器的默认 jdk 版本：

sudoupdate-alternatives --config java 选 auto java

2. 编辑 :/etc/environment 文件，用命令 sudogedit/etc/environment 打开

environment 文件，修改：

```
PATH=”/usr/lib/jvm/bin:$PATH”
CLASSPATH="/usr/lib/jvm/lib"
JAVA_HOME="/usr/lib/jvm"
```

根据系统的具体路径修改。

3. 给文件系统 Integration RowboPERF · 在 android 源码中下载：
git clone git://gitorious.org/rowboat/rowboperf.git

4. 开始编译：
make TARGET_PRODUCT=<product-name> OMAPES=4.x -j<N>

注：N 为编译所在主机的处理器数量乘以 2。

最后，经过漫长的等待，约 6 到 7 个小时后编译完成。

制作用于启动的 SD 卡：

（1）得到 rootfs.tar.bz2：

cd out/target/product/<product-name>
mkdir android_rootfs
cp -r root/* android_rootfs
cp -r system android_rootfs
../../../../build/tools/mktarball.sh ../../../host/linux-x86/bin/fs_get_stats android_rootfs . rootfs rootfs.tar.bz2

（2）制作

mkdir image_folder
cp uEnv.txt image_folder
cp kernel/arch/arm/boot/uImage image_folder

cp u-boot/u-boot.img image_folder
cp u-boot/MLO image_folder
cp out/target/product/<product-name>/rootfs.tar.bz2 image_folder
cp Media_Clips image_folder
sudo ./mkmmc-android.sh /dev/sd<mount-point> MLO u-boot.img uImage uEnv.txt rootfs.tar.bz2 Media_Clips

注：uEnv.txt Media_Clips mkmmc-android.sh 可以从下载 android 源码同一个地址中下载到。最简单的就是下载一个现成的 prebuild-image，从中解压得到。

点击查看详情：>> 基于 BeagleBone 的 Android4.0.3 ICS 的移植

3.5 基于 Qt 和 MySQL 的 ZigBee 数据采集系统

所要实现功能描述：

看中了AM3359 的图形处理能力，决定开发一款基于 Qt 和 MySQL 的 ZigBee 数据采集系统。通过串口与 CC2530 通讯，将 Qt 库和 MySQL 库加入到 BBB 的 linxu 系统中，开发应用界面，显示组网上传的数据曲线，实现数据的存储和读取等功能。

硬件设计思路：

所需模块与器材：LCD 是必不可少的，当然其需要带有触摸功能；然后就是用于组网的 ZigBee 模块，我决定采用 CC2530 控制芯片；然后就是起到辅助作用的功能按键。硬件设计思路见原理框图。

软件设计思路：

因为 Ti 关于 ZigBee 技术的 CC2530 芯片提供了很全面地例程，我记得有一款开源的用于观察组网结构的上位机软件，其源代码用的就是 Qt 图形开发工具，我在 PC 机上曾经修改和扩展过此软件，增加了许多曲线图形功能，还加入了 MySQL 数据库和 QWT 插件等。源代码修改之后也编译成功了，在 PC 上工作还是很稳定的! 所以我决定将此软件搬到 BBB 上去。

时间安排 / 项目实施细则：

3 月 8 日—3 月 31：设计 LCD 扩展板，并制板成功，然后焊接扩展板，调试 LCD 基本显示功能和触摸功能；

4 月 1 日 -4 月 30 日：调试 BBB 板与 ZigBee 模块的串口通讯，制定串口数据的格式，解析上传数据。

5月1日-5月31日:添加Qt和MySQL数据库到系统,并调试其功能,绘制数据采集界面,数据曲线显示界面，数据存储查看界面，还有 ZigBee 网络节点显示界面等。

预期成果：

LCD 显示正常，触摸功能正常，成功运行 Qt 和 MySQL 库，成功与 ZigBee 模块通讯，成功显示网络节点拓扑图，数据曲线图，成功存取数据。

功能图：

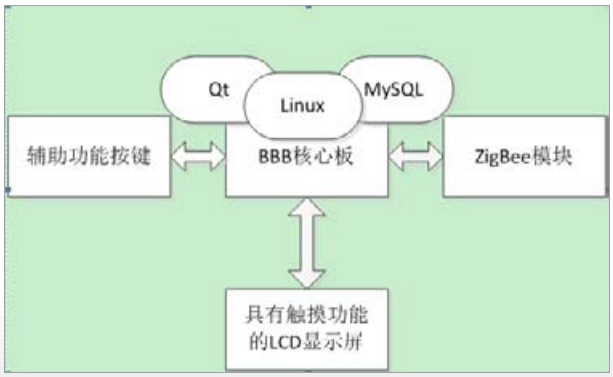


图 3-30

一、最终设计：

终于赶在 27 号结束之前将设计完成了，界面简陋了点，有些 Low！但是整体功能已经实现的差不多了。 在上一帖里面提到了，我要对数据解析，感兴趣的朋友可以研究一下，CC2530 ZigBee 组网的数据格式，我将子节点的上传数据格式进行了扩展，在一帧数据中加入了其独一无二的标识，如下图：

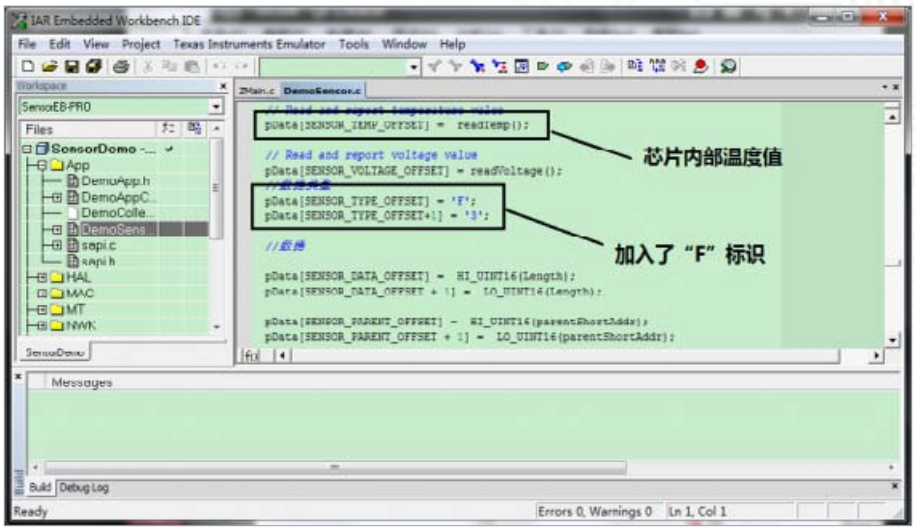


图 3-31

可见，我在一帧数据中加入了“F”，这样在协调器的数据格式则如下图：

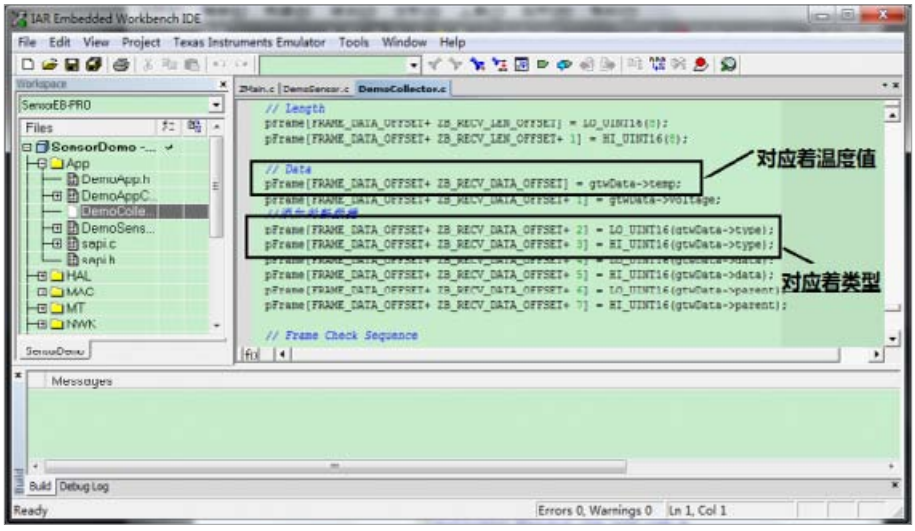


图 3-32

因此数据就能够按此格式上传到 BBB 板的上位机上!

在上位机程序中，只需要挑出有用的数据和表示加以显示即可!

数据处理方式如下图的 Qt 程序：

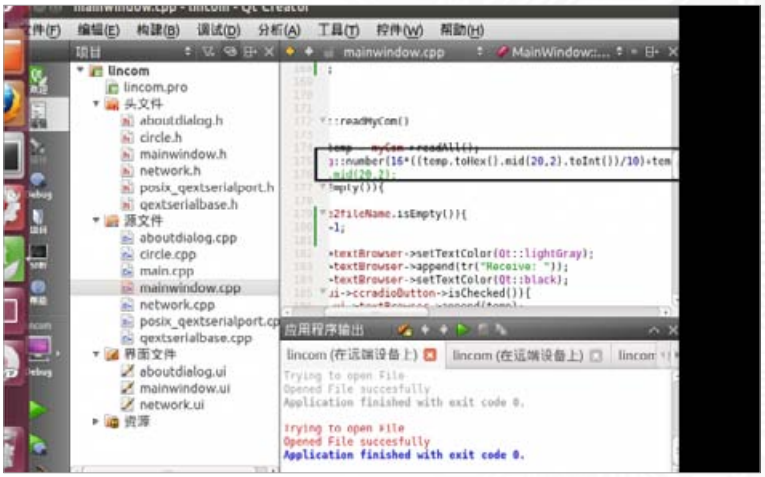


图 3-33

最终效果则如下图所示：

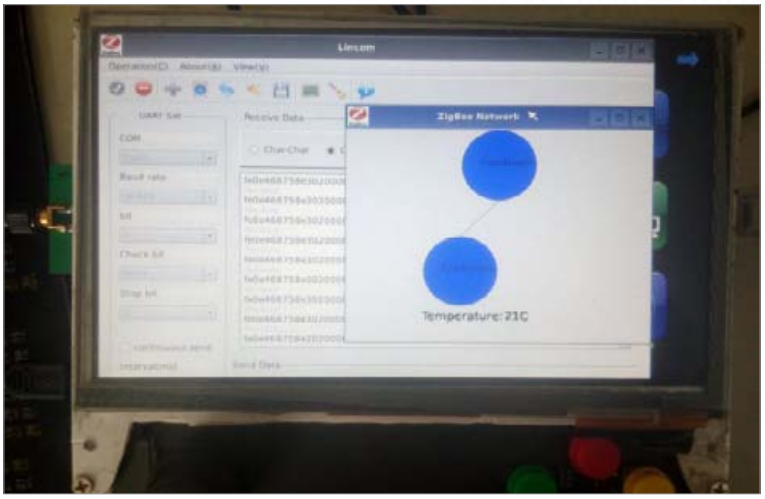


图 3-34

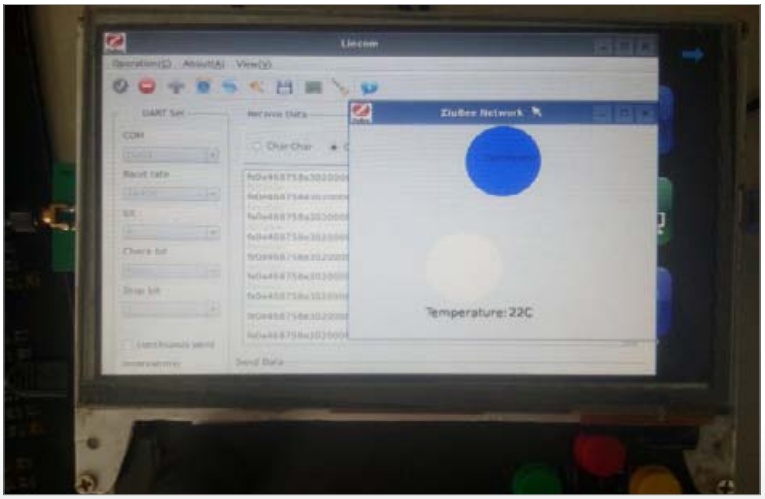


图 3-35

图标会不断地闪烁，表示网络中的节点工作正常!

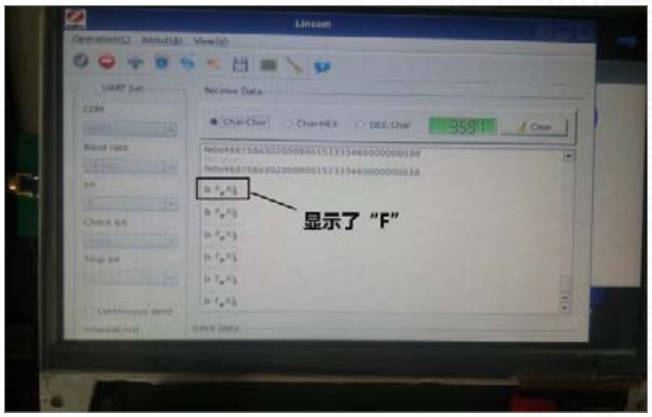


图 3-36

可见一帧数据中确实有“F”出现!

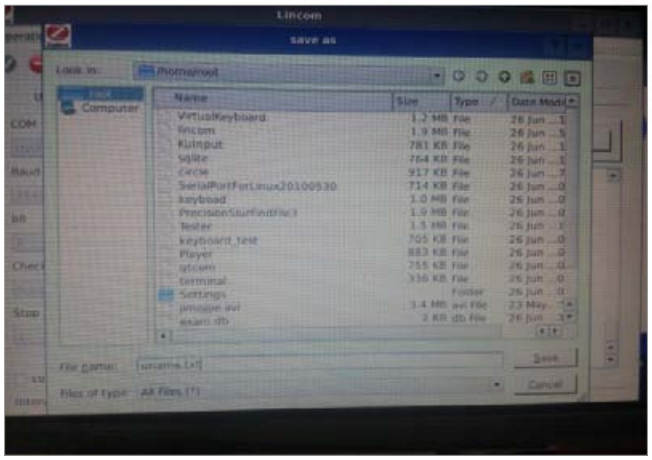


图 3-37

保存数据的功能!

图 3-38

>> 项目测试视频

二、本次活动所有的帖子在此：

1、BBB 方案原理图改进！：

http://bbs.eeworld.com.cn/thread-435621-1-1.html

2、BeagleBone Black 设计 :BBB 板底板 PCB 绘制完成！：

http://bbs.eeworld.com.cn/thread-437613-1-1.html

3、BeagleBone Black 设计 :BBB 板运行 Ti 的官方 Linux 操作系统：

http://bbs.eeworld.com.cn/thread-437690-1-1.html

4、BeagleBone Black 设计：更新 BBB 底板 PCB 的 3D 效果，分享一下 Altium Designer PCB：

http://bbs.eeworld.com.cn/thread-437948-1-1.html

5、BeagleBone Black 设计：制板归来以及焊接准备和计划：

http://bbs.eeworld.com.cn/thread-437971-1-1.html

6、BeagleBone Black 设计 :BBB 底板焊接完成及驱动测试！：

http://bbs.eeworld.com.cn/thread-438231-1-1.html

7、BeagleBone Black 设计：关于 WAV 音乐播放音量调节的心得！：

http://bbs.eeworld.com.cn/thread-438241-1-1.html

8、BeagleBone Black 设计 :BBB 底板添加电容屏和 tslib 测试！：

http://bbs.eeworld.com.cn/thread-438642-1-1.html

9、BeagleBone Black 设计 :BBB 底板添加电池和完善电容屏效果！：

http://bbs.eeworld.com.cn/thread-438716-1-1.html

10、BeagleBone Black 设计 :ADC 应用和摇杆调试！：

http://bbs.eeworld.com.cn/thread-438761-1-1.html

11、BeagleBone Black 设计 :Qt 程序编写和桌面图标添加！：

http://bbs.eeworld.com.cn/thread-438921-1-1.html

12、BeagleBone Black 设计：数据库的添加和测试：

http://bbs.eeworld.com.cn/thread-440208-1-1.html

13、BeagleBone Black 设计 :ZigBee 组网的 Qt 界面设计思路：

http://bbs.eeworld.com.cn/thread-440219-1-1.html

14、BeagleBone Black 设计：ZigBee 模块通讯测试，活动倒数第二帖：

http://bbs.eeworld.com.cn/thread-440818-1-1.html

15、BeagleBone Black 设计：活动设计最终版以及扩展板详细介绍！：

http://bbs.eeworld.com.cn/thread-440975-1-1.html

三、介绍一下我设计的 BBB 扩展板：

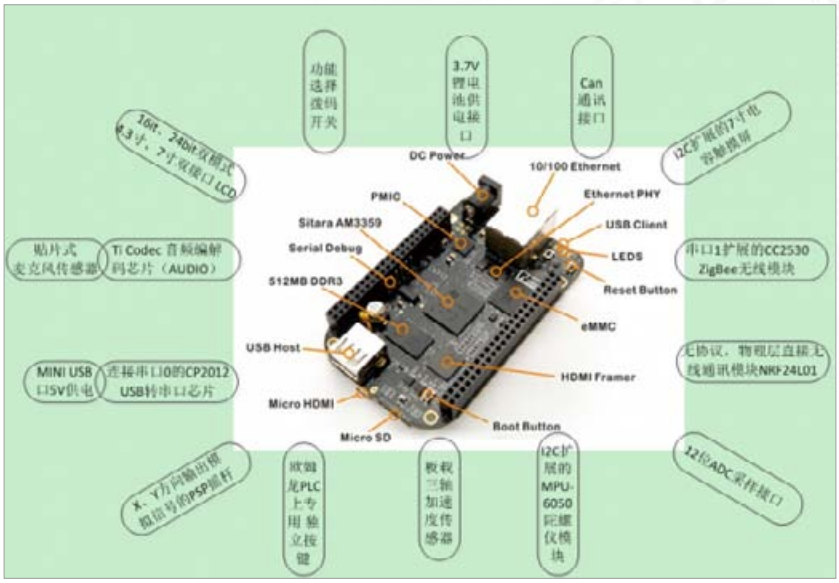


图 3-39

图 3-40

整个 BBB 扩展板的功能如上图所示，具体原理图前面的帖子有介绍!

点击查看详情：>> 基于 Qt 和 MySQL 的 ZigBee 数据采集系统

3.6 基于 BBB 的简单频谱仪

所要实现功能描述：

将音频频谱显示在 BBB 的液晶屏上，如果时间足够的话会加上一些储存的功能

硬件设计思路：

使用拾音器将音频信号采集后使用运放进行放大等处理后使用 AD 进行转换

软件设计思路：

使用 BBB 的 AD 将信号前端处理后的信号进行 AD 转换，然后使用 fftw 进行傅里叶变换，根据需要决定是否使用 neon 指令集和 posix 线程库

时间安排 / 项目实施细则：

三月二十四号到五月底 软件的移植 fftw 和 Qt 显示等

五月底到技术时间 信号的电路的设计

预期成果：

可以将音频频谱显示在屏幕上
储存音频信号

>> 项目视频

附录 1 BeagleBone 创意列表

1、让 BB Black 飞起来 ~ 基于 BeagleBone Black 的四轴飞行器设计	基于 bb 的四轴飞行器。
2、机器视觉循迹避障小车	在 BB 上基于 andriod 系统开发，实现小车的前进、后退、加速以及避障功能，通过无线传输实时给电脑传输数据，来实时跟踪小车状态。
3、基于 Qt 和 MySQL 的 ZigBee 数据采集系统	通过串口与 CC2530 通讯，将 Qt 库和 MySQL 库加入到 BBB 的 linxu 系统中，开发应用界面，显示组网上传的数据曲线，实现数据的存储和读取等功能。
4、基于 BBB 的移动图像采集终端	基于 USB camera 图像采集，实现图片或者视频的实时上传。
5、基于 BB-Black 的 Data Processing Center	设计并开发一款基于 BB-Black 的 yo 数据采集和分析的便携式平
6、基于 BBB 开发析的 android 车载导航影音多媒体系统	使用 BBB 作为主控板，外挂 GPS 模块 ,WIFI 模块，蓝牙模块，收音高频头模块，连接 7 寸电容屏，运行 android 系统，实现车载导航影音系统。
7、婴幼儿监控系统	通过手机远程观测婴幼儿生活。
8、基于 BeagleBone Black 的彩色 LED 屏幕动态图像解码驱动器	本设计使用 BeagleBone 完成图像和视频的解码工作，同时完成彩色 LED 屏幕的驱动工作。使用时由电脑制作好视频文件，放入 SD 卡或者 U 盘内由 BeagleBone 进行播放。
9、基于 BB 的智能电表集中器	本方案以 BB 为基础设计一个居民电表集中器，抄取一台配电变压器下属的所有用户的电表表底，并实时上传到电力系统售电中心。
10、基于 linux 的数字示波表设计	BB-black 设计一款多功能便携示波器
11、医疗设备基于图像处理的细胞分析	依托 AM3359 强大的主频和数据处理功能，实现显微镜下血液涂片图像的处理，实现白细胞和红细胞的统计计数，白细胞三分类等血液分析仪的大部分功能，代替医生显微镜下人工计数的困境。并实现病人数据存储记录功能。
12、智能广播广告系统	基于 BeagleBone Black 实现广播广告终端。终端的功能包括播放文字、图片、视频、音频等。用途包括播放视频广告，文字滚动播放最新资讯，以及紧急状况下播放文字、语音警报。

13、基于 BB 及 FPGA 的高速数据采集及 DDS 信号发生	BeagleBone Black 开发板通过 GPMC 接口读写 Altera FPGA 的双口 RAM，FPGA 外接高速 AD 及 DA 转换器，加上外围信号处理电路，可以实现高速数据采集及 DDS 信号生成。 该设计利用 DDS 原理生成两路模拟三角或者正弦信号，控制 XY 扫描振镜的运动，采集光电倍增管的信号 ，并能够同步扫描与采集过程，实现二维扫描图像的生成。 BeagleBone Black 外扩 CAN 总线接口，作为系统的主控制单元，可以控制其他相关的控制模块,并通过 以太网接口实现与上位机的信息交换。
14、BeagleBone Black 家庭娱乐系统	随着生活水平提高，对物质文化要求也越来越高吗，传统的娱乐场所的大众化无法满足个人喜好的需求。形形色色的智能电子产品出现为个性化设计提供了可能性。最经有幸获得一款蓝牙 LED 灯，所以想利用板卡的强大控制能力，初步设计一个灯光控制系统，如果可行逐步实现音乐灯光一体化控制，打造个性化家庭娱乐系统，足不出户享受生活提高生活情趣。
15、基于 BB Black 的室内移动目标检测系统	基于 BB Black 嵌入式平台，使用网络摄像头，利用 OpenCV 实现室内环境下移动目标的检测。
16、智能可视化嵌入式多点控制和加热控制器	初步实现一个控制系统的框架。便于应用到多个相关的自动控制项目中。实现可视化的友好操作界面。能够直观清楚的使用户了解设备工作状态和提醒用户操作。
17、基于 Beaglebone black 平台的紫外激光 电路板扫描直刻系统	借助 Beagleboneblack 的强大性能，处理电路板 pdf 文件，驱动外围激光扫描设备对感光板进行扫描，实现直接光刻制作电路板。
18、TCP IP 的门禁系统	一个带监控、拍照、录像、移动侦测功能的门禁室内机
19、基于 BeagleBone Black 的分布式电机控制器设计	利用 BeagleBone Black 构建一个嵌入式的分布式电机控制系统，在 BeagleBone Black 构建一个 EtherCAT 的主站，通过 EtherCAT 协议来控制从站的电机驱动器，利用液晶提供操作的界面和数据显示。
20、家居生活管理中心	本次方案仅针对实现利用 BB 作为家电管理中心，通过移动终端访问 BB 服务站点来控制家里的电器。
21、嵌入式工业控制系统	上位机通过网络对 BB 下发命令，BB 按照上位机命令监控控制整个工业系统及反馈采集信息。

22、用 BBB 实现的 BLT 连接的智能数据集中器	用 BBB 实现的 BLT 连接的智能数据集中器
23、网络存储器	运用 BeagboneBlack 实现网络存储器，能够将挂载的硬盘的内容通过网络访问，HDMI 接口能够与电视机相连，实现移动硬盘内容共迎享。如果长时间没有数据请求，移动硬盘自动关闭。
24、胎儿监护仪	用 Beagle Bone Black 做胎儿监护仪。将仪器放到怀孕母亲的肚子上可以通过显示屏观察到孕妇肚子中的胎儿的情况。对安全健康孕育有很重要的意义。
25、简版嵌入式视频会议终端	嵌入式平台下面的视频会议终端，实现和 PC 端的软终端进行视频通话。
26、基于 BeagleBone Black 的家庭服务器	利用 BeagleBone Black 构建一个家庭服务器，可以电脑连接读取信息，通过电视来播放视频以及上网，还有外部环境的检测，来监控环境变化。
27、GPS/GPRS 汽车行驶记录仪	汽车行驶记录仪主要记录行驶状态数据（行驶速度、GPS 坐标、驾驶员信息采集等）测量和存储，数据通信功能可以通过 RS232 接口与外部设备通信，USB 读写功能和 GPRS 通信功能。 断油断电功能。紧急报警求救功能。
28、智能家居监控系统	对住房内进行视屏监控，可在网上对室内进行实时监控，高温，烟雾，煤气进行报警
29、智能家居之微博版	通过 BeagleBone 采集各种传感器，如温感的数值并自动发到微博上。家用 USB 摄像头定时拍照，并上传到微博。发微博控制继电器板从而控制家电的开关。
30、智能家居方案之家庭安防	实现通过智能手机，平板电脑等随身设备对家庭环境进行远程监控，和部分设备控制
31、仿玉兔月球车方案	以 BeagleBone Black 为主控载体，下设各个控制部分和传感器及通讯、电源等机构。按照规则运动、太阳能充电，能完成基础的自我修复、钻孔、采集土壤、全景摄像、远程通讯、排除障碍等功能
32、基于 BBB 的简单频谱仪	将音频频谱显示在 BBB 的液晶屏上，如果时间足够的话会加上一些储存的功能

附录 2 编委信息与后记

《深入浅出玩 TI Sitara——TI ARM MPU 资源介绍及 Beaglebone Black 实战攻略》整合了 TI 官方培训文字资料，以及 EEworld 诸多热爱 BeagleBone Black 的工程师实战项目开发资料，使得 初次使 用 TI Sitara 的工程师能够快速上手开发。

在此，特别感谢：EEworld 社 区 热心坛友对我们活动的支持与关注，尤其感谢 BeagleBone Black 设计方案分享的工程师，为 我 们 分享 了大 量 新 鲜、一 手 的应 用经验。

特 别 感 谢 ：TI 公司的大力支持。

希 望这本书，能够为你带来小小的帮助、小小的惊喜。

EEworld 社区
2014. 7

附录 3 版权说明

1、《深入浅出玩 TI Sitara——TI ARM MPU 资源介绍及 Beaglebone Black 实战攻略》著作权属 TI 和 EEworld 共同所拥有；

2、本着开源思想，我们授权任何对 TI Sitara 有兴趣的工程师免费下载、复制、传播该书；

3、用于商业用途须经 TI 和 EEworld 书面同意。